import libraries into the operating environment

In [1]:

```
#import libraries into the operating environment
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Read data from the url into pandas dataframe df

In [2]:

```
#load data into pandas dataframe df
df =pd.read_csv('https://raw.githubusercontent.com/jackiekazil/data-
wrangling/master/data/chp3/data-text.csv')
#display first two rows from the dataframe df
df.head(2)
```

Out[2]:

	Indicator	PUBLISH STATES	Year	WHO region	World Bank income group	Country	Sex	Display Value	Numeric	Low	High	Comments
0	Life expectancy at birth (years)	Published	1990	Europe	High-income	Andorra	Both sexes	77	77.0	NaN	NaN	NaN
1	Life expectancy at birth (years)	Published	2000	Europe	High-income	Andorra	Both sexes	80	80.0	NaN	NaN	NaN

Read data from the url into pandas dataframe df1

In [3]:

```
#load data into pandas dataframe df1
df1 =pd.read_csv('https://raw.githubusercontent.com/kjam/data-wrangling-
pycon/master/data/berlin_weather_oldest.csv')
#display first two rows from the dataframe df1
df1.head(2)
```

Out[3]:

	STATION	STATION_NAME	DATE	PRCP	SNWD	SNOW	TMAX	TMIN	WDFG	PGTM	...	WT09	WT07	V
0	GHCND:GME00111445	BERLIN TEMPELHOF GM	19310101	46	-9999	-9999	-9999	-11	-9999	-9999	...	-9999	-9999	-
1	GHCND:GME00111445	BERLIN TEMPELHOF GM	19310102	107	-9999	-9999	50	11	-9999	-9999	...	-9999	-9999	-

2 rows × 21 columns

In [4]:

```
print(df.shape)
print(df1.shape)
```

```
(4656, 12)
(117208, 21)
```

1. Get the Metadata from the above files.

In [5]:

```
# pd.Dataframe.info()- retrieves the metadata about the dataframe
print("Metadata - df")
print('*'*80)
print(df.info())
print("\n")
print("Metadata - df1")
print('*'*80)
print(df1.info())
```

```
Metadata - df
********************************************************************************
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4656 entries, 0 to 4655
Data columns (total 12 columns):
Indicator                  4656 non-null object
PUBLISH STATES             4656 non-null object
Year                       4656 non-null int64
WHO region                 4656 non-null object
World Bank income group    4656 non-null object
Country                    4656 non-null object
Sex                        4656 non-null object
Display Value              4656 non-null int64
Numeric                    4656 non-null float64
Low                        0 non-null float64
High                       0 non-null float64
Comments                   0 non-null float64
dtypes: float64(4), int64(2), object(6)
memory usage: 436.6+ KB
None


Metadata - df1
********************************************************************************
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 117208 entries, 0 to 117207
Data columns (total 21 columns):
STATION         117208 non-null object
STATION_NAME    117208 non-null object
DATE            117208 non-null int64
PRCP            117208 non-null int64
SNWD            117208 non-null int64
SNOW            117208 non-null int64
TMAX            117208 non-null int64
TMIN            117208 non-null int64
WDFG            117208 non-null int64
PGTM            117208 non-null int64
WSFG            117208 non-null int64
WT09            117208 non-null int64
WT07            117208 non-null int64
WT01            117208 non-null int64
WT06            117208 non-null int64
WT05            117208 non-null int64
WT04            117208 non-null int64
WT16            117208 non-null int64
WT08            117208 non-null int64
WT18            117208 non-null int64
WT03            117208 non-null int64
dtypes: int64(19), object(2)
memory usage: 18.8+ MB
None
```

2. Get the row names from the above files.

In [6]:

```
#the row names in a dataframe correspond to the index values which can be retieved using .index co
mmand
print("dataframe df row names as follows : ")
df.index.values
```

```
dataframe df row names as follows :
```

Out[6]:

```
array([   0,    1,    2, ..., 4653, 4654, 4655], dtype=int64)
```

In [7]:

```
print("dataframe df1 row names as follows : ")
df1.index.values
```

```
dataframe df1 row names as follows :
```

Out[7]:

```
array([     0,      1,      2, ..., 117205, 117206, 117207], dtype=int64)
```

3. Change the column name from any of the above file.

In [8]:

```
#changing the column name of dataframe df from indicator to indicator_id using the rename method
df.rename(columns={'Indicator':'Indicator_id'}).head(2)
```

Out[8]:

	Indicator_id	PUBLISH STATES	Year	WHO region	World Bank income group	Country	Sex	Display Value	Numeric	Low	High	Comments
0	Life expectancy at birth (years)	Published	1990	Europe	High-income	Andorra	Both sexes	77	77.0	NaN	NaN	NaN
1	Life expectancy at birth (years)	Published	2000	Europe	High-income	Andorra	Both sexes	80	80.0	NaN	NaN	NaN

In [9]:

```
#verify if the column name in the original dataframe df if it has changed
df.head(2)
```

Out[9]:

	Indicator	PUBLISH STATES	Year	WHO region	World Bank income group	Country	Sex	Display Value	Numeric	Low	High	Comments
0	Life expectancy at birth (years)	Published	1990	Europe	High-income	Andorra	Both sexes	77	77.0	NaN	NaN	NaN
1	Life expectancy at birth (years)	Published	2000	Europe	High-income	Andorra	Both sexes	80	80.0	NaN	NaN	NaN

4. Change the column name from any of the above file and store the changes made permanently.

In [10]:

```
#"original datafarme df  column changes being made permanent by using inplace=True option.
df.rename(columns={'Indicator':'Indicator_id'},inplace=True)
df.head(2)
```

Out[10]:

	Indicator_id	PUBLISH STATES	Year	WHO region	World Bank income group	Country	Sex	Display Value	Numeric	Low	High	Comments
0	Life expectancy at birth (years)	Published	1990	Europe	High-income	Andorra	Both sexes	77	77.0	NaN	NaN	NaN

	Indicator_id	PUBLISH STATES	Year	WHO region	World Bank income group	Country	Sex	Display Value	Numeric	Low	High	Comments
	at birth (years)						sexes					
1	Life expectancy at birth (years)	Published	2000	Europe	High-income	Andorra	Both sexes	80	80.0	NaN	NaN	NaN

5. Change the names of multiple columns.

In [11]:

```
#changing multiple column names PUBLISH STATES to Publication Status, WHO region to WHO Region
df.rename(columns={"PUBLISH STATES":"Publication Status","WHO region":"WHO Region"},inplace=True)
df.head(2)
```

Out[11]:

	Indicator_id	Publication Status	Year	WHO Region	World Bank income group	Country	Sex	Display Value	Numeric	Low	High	Comments
0	Life expectancy at birth (years)	Published	1990	Europe	High-income	Andorra	Both sexes	77	77.0	NaN	NaN	NaN
1	Life expectancy at birth (years)	Published	2000	Europe	High-income	Andorra	Both sexes	80	80.0	NaN	NaN	NaN

6. Arrange values of a particular column in ascending order.

In [12]:

```
#sorting the values in Year column in ascending order of year and making the sort operation perman
ent using inplace =True option.
df.sort_values('Year',ascending=True).head()
```

Out[12]:

	Indicator_id	Publication Status	Year	WHO Region	World Bank income group	Country	Sex	Display Value	Numeric	Low	High	Comments
0	Life expectancy at birth (years)	Published	1990	Europe	High-income	Andorra	Both sexes	77	77.0	NaN	NaN	NaN
1270	Life expectancy at birth (years)	Published	1990	Europe	High-income	Germany	Male	72	72.0	NaN	NaN	NaN
3193	Life expectancy at birth (years)	Published	1990	Europe	Lower-middle-income	Republic of Moldova	Male	65	65.0	NaN	NaN	NaN
3194	Life expectancy at birth (years)	Published	1990	Europe	Lower-middle-income	Republic of Moldova	Both sexes	68	68.0	NaN	NaN	NaN
3197	Life expectancy at age 60 (years)	Published	1990	Europe	Lower-middle-income	Republic of Moldova	Male	15	15.0	NaN	NaN	NaN

7. Arrange multiple column values in ascending order.

In [13]:

```
#Arange multiple column values in ascending order for sample rows having country as 'Andorra' as s
pecified in the
#expected output in the project for problem statement 7.

#filtered out the data whose Country name is Andorra from the original dataframe df1 into df_andor
ra
```

```
df_andorra = df[df['Country'] =='Andorra']
#only keep the columns specified in the list
df_andorra = df_andorra[['Indicator_id','Country','Year','WHO Region','Publication Status']]
#sort the columns Indicator_id in descending order and the column Year in ascending order. Also dr
op the duplicate  rows
df_andorra = df_andorra.sort_values(['Indicator_id','Country','Year','WHO
Region'],ascending=[False,False,True,False]).drop_duplicates(keep="first")

#reset the indexes after sorting as the index values will have row numbers of the original datafra
me df
df_andorra =df_andorra.reset_index(drop=True)

# display first five ros of the sorted dataframe
df_andorra.head(3)
```

Out[13]:

	Indicator_id	Country	Year	WHO Region	Publication Status
0	Life expectancy at birth (years)	Andorra	1990	Europe	Published
1	Life expectancy at birth (years)	Andorra	2000	Europe	Published
2	Life expectancy at birth (years)	Andorra	2012	Europe	Published

8. Make country as the first column of the dataframe.

In [14]:

```
df[pd.unique(['Country'] + df.columns.values.tolist()).tolist()].head()
```

Out[14]:

	Country	Indicator_id	Publication Status	Year	WHO Region	World Bank income group	Sex	Display Value	Numeric	Low	High	Comments
0	Andorra	Life expectancy at birth (years)	Published	1990	Europe	High-income	Both sexes	77	77.0	NaN	NaN	NaN
1	Andorra	Life expectancy at birth (years)	Published	2000	Europe	High-income	Both sexes	80	80.0	NaN	NaN	NaN
2	Andorra	Life expectancy at age 60 (years)	Published	2012	Europe	High-income	Female	28	28.0	NaN	NaN	NaN
3	Andorra	Life expectancy at age 60 (years)	Published	2000	Europe	High-income	Both sexes	23	23.0	NaN	NaN	NaN
4	United Arab Emirates	Life expectancy at birth (years)	Published	2012	Eastern Mediterranean	High-income	Female	78	78.0	NaN	NaN	NaN

9. Get the column array using a variable

In [15]:

```
#the column array values for WHO Region column is retieved using values method and stored in a var
iable WHOregion
WHOregion = df['WHO Region'].values
WHOregion
```

Out[15]:

```
array(['Europe', 'Europe', 'Europe', ..., 'Africa', 'Africa', 'Africa'],
      dtype=object)
```

10. Get the subset rows 11, 24, 37

In [16]:

```
# the row numbers specified can be retrieved using loc method which uses index values or row numbe
rs as input
df.iloc[[11,24,37]]
```

Out[16]:

	Indicator_id	Publication Status	Year	WHO Region	World Bank income group	Country	Sex	Display Value	Numeric	Low	High	Comments
11	Life expectancy at birth (years)	Published	2012	Europe	High-income	Austria	Female	83	83.0	NaN	NaN	NaN
24	Life expectancy at age 60 (years)	Published	2012	Western Pacific	High-income	Brunei Darussalam	Female	21	21.0	NaN	NaN	NaN
37	Life expectancy at age 60 (years)	Published	2012	Europe	High-income	Cyprus	Female	26	26.0	NaN	NaN	NaN

11. Get the subset rows excluding 5, 12, 23, and 56

In [17]:

```
#first genearte a boolean array generating true values for index or row numbers in the list
bad_df = df.index.isin([5,12,23,56])

#filter out only the false values using ~ operator which will retrieve values corresponding
#to all index values except for 5,12,23,56

df[~bad_df].head()
```

Out[17]:

	Indicator_id	Publication Status	Year	WHO Region	World Bank income group	Country	Sex	Display Value	Numeric	Low	High	Comments
0	Life expectancy at birth (years)	Published	1990	Europe	High-income	Andorra	Both sexes	77	77.0	NaN	NaN	NaN
1	Life expectancy at birth (years)	Published	2000	Europe	High-income	Andorra	Both sexes	80	80.0	NaN	NaN	NaN
2	Life expectancy at age 60 (years)	Published	2012	Europe	High-income	Andorra	Female	28	28.0	NaN	NaN	NaN
3	Life expectancy at age 60 (years)	Published	2000	Europe	High-income	Andorra	Both sexes	23	23.0	NaN	NaN	NaN
4	Life expectancy at birth (years)	Published	2012	Eastern Mediterranean	High-income	United Arab Emirates	Female	78	78.0	NaN	NaN	NaN

Load datasets from CSV

In [18]:

```
#load into users dataframe
users = pd.read_csv('https://raw.githubusercontent.com/ben519/DataWrangling/master/Data/users.csv'
)
users.head()
```

Out[18]:

	UserID	User	Gender	Registered	Cancelled
0	1	Charles	male	2012-12-21	NaN
1	2	Pedro	male	2010-08-01	2010-08-08
2	3	Caroline	female	2012-10-23	2016-06-07
3	4	Brielle	female	2013-07-17	NaN
4	5	Benjamin	male	2010-11-25	NaN

In [19]:

```
#load into sessions dataframe
sessions
=pd.read_csv('https://raw.githubusercontent.com/ben519/DataWrangling/master/Data/sessions.csv')
sessions.head()
```

Out[19]:

	SessionID	SessionDate	UserID
0	1	2010-01-05	2
1	2	2010-08-01	2
2	3	2010-11-25	2
3	4	2011-09-21	5
4	5	2011-10-19	4

In [20]:

```
#load into products dataframe
products =
pd.read_csv('https://raw.githubusercontent.com/ben519/DataWrangling/master/Data/products.csv')
products.head()
```

Out[20]:

	ProductID	Product	Price
0	1	A	14.16
1	2	B	33.04
2	3	C	10.65
3	4	D	10.02
4	5	E	29.66

In [21]:

```
#load into transactions dataframe
transactions =
pd.read_csv('https://raw.githubusercontent.com/ben519/DataWrangling/master/Data/transactions.csv')
transactions.head()
```

Out[21]:

	TransactionID	TransactionDate	UserID	ProductID	Quantity
0	1	2010-08-21	7.0	2	1
1	2	2011-05-26	3.0	4	1
2	3	2011-06-16	3.0	3	1
3	4	2012-08-26	1.0	2	3
4	5	2013-06-06	2.0	4	1

In [22]:

```
users['Registered'] = pd.to_datetime(users['Registered'])
users['Cancelled'] = pd.to_datetime(users['Cancelled'])
sessions['SessionDate'] = pd.to_datetime(sessions['SessionDate'])
transactions['TransactionDate'] = pd.to_datetime(transactions['TransactionDate'])
```

12. Join users to transactions, keeping all rows from transactions and only matching rows from users (left join)

In [23]:

```
#doing a left outter join on transactions and user on the basis of UserID column
#result stored in a dataframe df_merge_trans_users
df_Left_trans_users = pd.merge(transactions,users,how="left",on="UserID")
df_Left_trans_users
```

Out[23]:

	TransactionID	TransactionDate	UserID	ProductID	Quantity	User	Gender	Registered	Cancelled
0	1	2010-08-21	7	2	1	NaN	NaN	NaT	NaT
1	2	2011-05-26	3	4	1	Caroline	female	2012-10-23	2016-06-07
2	3	2011-06-16	3	3	1	Caroline	female	2012-10-23	2016-06-07
3	4	2012-08-26	1	2	3	Charles	male	2012-12-21	NaT
4	5	2013-06-06	2	4	1	Pedro	male	2010-08-01	2010-08-08
5	6	2013-12-23	2	5	6	Pedro	male	2010-08-01	2010-08-08
6	7	2013-12-30	3	4	1	Caroline	female	2012-10-23	2016-06-07
7	8	2014-04-24	NaN	2	3	NaN	NaN	NaT	NaT
8	9	2015-04-24	7	4	3	NaN	NaN	NaT	NaT
9	10	2016-05-08	3	4	4	Caroline	female	2012-10-23	2016-06-07

13. Which transactions have a UserID not in users?

In [24]:

```
#filter elements in transaction table which have UserID present in transactions table but not regi
stered in users table
transactions[~transactions['UserID'].isin(users['UserID'])]
```

Out[24]:

	TransactionID	TransactionDate	UserID	ProductID	Quantity
0	1	2010-08-21	7.0	2	1
7	8	2014-04-24	NaN	2	3
8	9	2015-04-24	7.0	4	3

14. Join users to transactions, keeping only rows from transactions and users that match via UserID (inner join)

In [27]:

```
df_Inner_trans_users = pd.merge(transactions,users,how="inner",on="UserID")
df_Inner_trans_users
```

Out[27]:

	TransactionID	TransactionDate	UserID	ProductID	Quantity	User	Gender	Registered	Cancelled
0	2	2011-05-26	3	4	1	Caroline	female	2012-10-23	2016-06-07
1	3	2011-06-16	3	3	1	Caroline	female	2012-10-23	2016-06-07
2	7	2013-12-30	3	4	1	Caroline	female	2012-10-23	2016-06-07
3	10	2016-05-08	3	4	4	Caroline	female	2012-10-23	2016-06-07
4	4	2012-08-26	1	2	3	Charles	male	2012-12-21	NaT
5	5	2013-06-06	2	4	1	Pedro	male	2010-08-01	2010-08-08
6	6	2013-12-23	2	5	6	Pedro	male	2010-08-01	2010-08-08

15. Join users to transactions, displaying all matching rows AND all non-matching rows (full outer join)

In [29]:

```
df_Outter_trans_users = pd.merge(transactions,users,how="outer",on="UserID")
df_Outter_trans_users
```

Out[29]:

	TransactionID	TransactionDate	UserID	ProductID	Quantity	User	Gender	Registered	Cancelled
0	1.0	2010-08-21	7.0	2.0	1.0	NaN	NaN	NaT	NaT
1	9.0	2015-04-24	7.0	4.0	3.0	NaN	NaN	NaT	NaT
2	2.0	2011-05-26	3.0	4.0	1.0	Caroline	female	2012-10-23	2016-06-07
3	3.0	2011-06-16	3.0	3.0	1.0	Caroline	female	2012-10-23	2016-06-07
4	7.0	2013-12-30	3.0	4.0	1.0	Caroline	female	2012-10-23	2016-06-07
5	10.0	2016-05-08	3.0	4.0	4.0	Caroline	female	2012-10-23	2016-06-07
6	4.0	2012-08-26	1.0	2.0	3.0	Charles	male	2012-12-21	NaT
7	5.0	2013-06-06	2.0	4.0	1.0	Pedro	male	2010-08-01	2010-08-08
8	6.0	2013-12-23	2.0	5.0	6.0	Pedro	male	2010-08-01	2010-08-08
9	8.0	2014-04-24	NaN	2.0	3.0	NaN	NaN	NaT	NaT
10	NaN	NaT	4.0	NaN	NaN	Brielle	female	2013-07-17	NaT
11	NaN	NaT	5.0	NaN	NaN	Benjamin	male	2010-11-25	NaT

16. Determine which sessions occurred on the same day each user registered

Solution 1 using Pandas Merge

In [30]:

```
#performing an inner join on dataframe users and sessions for all matching Userid and matching Reg
istered and SessionDate
pd.merge(left=users,right=sessions,how="inner",left_on=['UserID','Registered'],right_on=['UserID','
SessionDate'])
```

Out[30]:

	UserID	User	Gender	Registered	Cancelled	SessionID	SessionDate
0	2	Pedro	male	2010-08-01	2010-08-08	2	2010-08-01
1	4	Brielle	female	2013-07-17	NaT	9	2013-07-17

Solution 2 :- Breaking into individual steps

1.Using Pandas Merge(inner join) on table users and sessions with join on the UserID key

2.Filter row having matching Registered and SessionDate columns

In [32]:

```
#creating a dataframe df_inner_users_sess by
df_inner_users_sess = pd.merge(users,sessions,how='left',on='UserID')
#df_inner_users_sess
#filtering out rows from dataframe df_inner_users_sess having the same Registered Date and Session
Date
df_inner_users_sess[df_inner_users_sess['Registered']== df_inner_users_sess['SessionDate'] ].reset_
index().drop('index',axis=1)
```

Out[32]:

	UserID	User	Gender	Registered	Cancelled	SessionID	SessionDate
0	2	Pedro	male	2010-08-01	2010-08-08	2.0	2010-08-01
1	4	Brielle	female	2013-07-17	NaT	9.0	2013-07-17

17. Build a dataset with every possible (UserID, ProductID) pair (cross join)

In [33]:

```
#create two different dataframes with unique UserID and ProductID from users and transactions data
frame respectively.
df_userid = pd.DataFrame({"UserID":users["UserID"]})
df_Tran = pd.DataFrame({"ProductID":products["ProductID"]})
#create new column Key with value as 1 for both the dataframe as this would become the common key
to be merged
df_userid['Key'] = 1
df_Tran['Key'] = 1
```

In [34]:

```
#do a outer join on df_userid and df_Tran dataframe
df_out = pd.merge(df_userid,df_Tran,how='outer',on="Key")[['UserID','ProductID']]
```

In [35]:

```
#final dataframe df_out which has every possible(UserID,ProductID) combination
df_out
```

Out[35]:

	UserID	ProductID
0	1	1
1	1	2
2	1	3
3	1	4
4	1	5
5	2	1
6	2	2
7	2	3

	UserID	ProductID
8	2	4
9	2	5
10	3	1
11	3	2
12	3	3
13	3	4
14	3	5
15	4	1
16	4	2
17	4	3
18	4	4
19	4	5
20	5	1
21	5	2
22	5	3
23	5	4
24	5	5

18. Determine how much quantity of each product was purchased by each user

In [36]:

```
#do a left join on the output table df_out from previous step with transactions table on the keys
['UserID','ProductID]
df_user_prod_quant = pd.merge(df_out,transactions,how='left',on=['UserID','ProductID'])

#Groupby the table on ['UserID','ProductID] and calculate the sum of Qunatity entity for each grou
p
df_user_quantity = df_user_prod_quant.groupby(['UserID','ProductID'])['Quantity'].sum()

#reset index so that the index column will have consecutive default numbers and fill NAN values wi
th 0
df_user_quantity.reset_index().fillna(0)
```

Out[36]:

	UserID	ProductID	Quantity
0	1	1	0.0
1	1	2	3.0
2	1	3	0.0
3	1	4	0.0
4	1	5	0.0
5	2	1	0.0
6	2	2	0.0
7	2	3	0.0
8	2	4	1.0
9	2	5	6.0
10	3	1	0.0
11	3	2	0.0
12	3	3	1.0
13	3	4	6.0

	UserID	ProductID	Quantity
14	3	5	0.0
15	4	1	0.0
16	4	2	0.0
17	4	3	0.0
18	4	4	0.0
19	4	5	0.0
20	5	1	0.0
21	5	2	0.0
22	5	3	0.0
23	5	4	0.0
24	5	5	0.0

19. For each user, get each possible pair of pair transactions (TransactionID1,TransacationID2)

In [37]:

```
#merge transactions dataframe performing a outer join to derive all possible pair of transactions
pd.merge(transactions,transactions,how='outer',on='UserID')
```

Out[37]:

	TransactionID_x	TransactionDate_x	UserID	ProductID_x	Quantity_x	TransactionID_y	TransactionDate_y	ProductID_
0	1	2010-08-21	7.0	2	1	1	2010-08-21	2
1	1	2010-08-21	7.0	2	1	9	2015-04-24	4
2	9	2015-04-24	7.0	4	3	1	2010-08-21	2
3	9	2015-04-24	7.0	4	3	9	2015-04-24	4
4	2	2011-05-26	3.0	4	1	2	2011-05-26	4
5	2	2011-05-26	3.0	4	1	3	2011-06-16	3
6	2	2011-05-26	3.0	4	1	7	2013-12-30	4
7	2	2011-05-26	3.0	4	1	10	2016-05-08	4
8	3	2011-06-16	3.0	3	1	2	2011-05-26	4
9	3	2011-06-16	3.0	3	1	3	2011-06-16	3
10	3	2011-06-16	3.0	3	1	7	2013-12-30	4
11	3	2011-06-16	3.0	3	1	10	2016-05-08	4
12	7	2013-12-30	3.0	4	1	2	2011-05-26	4
13	7	2013-12-30	3.0	4	1	3	2011-06-16	3
14	7	2013-12-30	3.0	4	1	7	2013-12-30	4
15	7	2013-12-30	3.0	4	1	10	2016-05-08	4
16	10	2016-05-08	3.0	4	4	2	2011-05-26	4
17	10	2016-05-08	3.0	4	4	3	2011-06-16	3
18	10	2016-05-08	3.0	4	4	7	2013-12-30	4
19	10	2016-05-08	3.0	4	4	10	2016-05-08	4
20	4	2012-08-26	1.0	2	3	4	2012-08-26	2
21	5	2013-06-06	2.0	4	1	5	2013-06-06	4
22	5	2013-06-06	2.0	4	1	6	2013-12-23	5
23	6	2013-12-23	2.0	5	6	5	2013-06-06	4
24	6	2013-12-23	2.0	5	6	6	2013-12-23	5
25	8	2014-04-24	NaN	2	3	8	2014-04-24	2

	TransactionID_x	TransactionDate_x	UserID	ProductID_x	Quantity_x	TransactionID_y	TransactionDate_y	ProductID

20. Join each user to his/her first occuring transaction in the transactions table

In [38]:

```
#do an left outer join on user and transactions dataframe on the UserID column
df_usertran = pd.merge(users,transactions,how='left',on='UserID')
# craete a new dataframe df_ with all duplicates on UserID being dropped , only keeping the first
entry
df_ = df_usertran.drop_duplicates(subset='UserID')
#reset the index to the default integer index.
df_ = df_.reset_index(drop=True)

#display the contents of the dataframe df_
df_
```

Out[38]:

	UserID	User	Gender	Registered	Cancelled	TransactionID	TransactionDate	ProductID	Quantity
0	1	Charles	male	2012-12-21	NaT	4.0	2012-08-26	2.0	3.0
1	2	Pedro	male	2010-08-01	2010-08-08	5.0	2013-06-06	4.0	1.0
2	3	Caroline	female	2012-10-23	2016-06-07	2.0	2011-05-26	4.0	1.0
3	4	Brielle	female	2013-07-17	NaT	NaN	NaT	NaN	NaN
4	5	Benjamin	male	2010-11-25	NaT	NaN	NaT	NaN	NaN

21. Test to see if we can drop columns

In [39]:

```
#Retieve the column list for the dataframe df_ created in problem statement 20
my_columns = list(df_.columns)
print(my_columns)
```

```
['UserID', 'User', 'Gender', 'Registered', 'Cancelled', 'TransactionID', 'TransactionDate',
'ProductID', 'Quantity']
```

In [40]:

```
list(df_.dropna(thresh=int(df_.shape[0] * .9), axis=1).columns) #set threshold to drop NAs
```

Out[40]:

```
['UserID', 'User', 'Gender', 'Registered']
```

In [41]:

```
missing_info = list(df_.columns[df_.isnull().any()])
missing_info
```

Out[41]:

```
['Cancelled', 'TransactionID', 'TransactionDate', 'ProductID', 'Quantity']
```

In [42]:

```
for col in missing_info:
    num_missing = df_[df_[col].isnull() == True].shape[0]
    print('number missing for column {}: {}'.format(col, num_missing))
```

```
number missing for column Cancelled: 3
number missing for column TransactionID: 2
number missing for column TransactionDate: 2
number missing for column ProductID: 2
number missing for column Quantity: 2
number missing for column Quantity: 2
```

In [43]:

```
for col in missing_info:
    num_missing = df_[df_[col].isnull() == True].shape[0]
    print('number missing for column {}: {}'.format(col, num_missing)) #count of missing df_
for col in missing_info:
    percent_missing = df_[df_[col].isnull() == True].shape[0] / df_.shape[0]
    print('percent missing for column {}: {}'.format(col, percent_missing))
```

```
number missing for column Cancelled: 3
number missing for column TransactionID: 2
number missing for column TransactionDate: 2
number missing for column ProductID: 2
number missing for column Quantity: 2
percent missing for column Cancelled: 0.6
percent missing for column TransactionID: 0.4
percent missing for column TransactionDate: 0.4
percent missing for column ProductID: 0.4
percent missing for column Quantity: 0.4
```

In [ ]: