Project_2

In [51]:

```
# import Pyhton modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Part 1

In [52]:

```
# load Datasets from Url
# Using read_csv() function to laod data

df =pd.read_csv('https://raw.githubusercontent.com/jackiekazil/data-wrangling/master/data/chp3/data-text.csv')
df.head(2)
```

Out[52]:

	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

In [53]:

```
# load Datasets from Url

df1 =pd.read_csv('https://raw.githubusercontent.com/kjam/data-wrangling-pycon/master/data/berlin_weather_oldest.csv')
df1.head(2)
```

Out[53]:

	STATION	STATION_NAME	DATE	PRCP	SNWD	SNOW	TMAX	TMIN	WDFG	PGTM	...	WT09	WT07	WT01	WT
0	GHCND:GME00111445	BERLIN TEMPELHOF GM	19310101	46	-9999	-9999	-9999	-11	-9999	-9999	...	-9999	-9999	-9999	-99
1	GHCND:GME00111445	BERLIN TEMPELHOF GM	19310102	107	-9999	-9999	50	11	-9999	-9999	...	-9999	-9999	-9999	-99

2 rows × 21 columns

In [54]:

```
# copy data to new datafarme
df_data_text = df
df_berlin_weather = df1
```

Task

1. Get the Metadata from the above files.

In [55]:

```
print("Metadata infon about data frame\n")
df_data_text.info()  #using ' info() ' function
```

```
Metadata infon about data frame

<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
```

In [56]:

```
print("Metadata infon about data frame\n")
df_berlin_weather.info()  # using ' info() ' function
```

```
Metadata infon about data frame

<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
```

2. Get the row names from the above files.

In [57]:

```
print("Row names of data frame : for data-text.csv")
np.array(df_data_text.columns)  # Get Mentioned dataframe columns information using numpy array
```

```
Row names of data frame : for data-text.csv
```

Out[57]:

```
array(['Indicator', 'PUBLISH STATES', 'Year', 'WHO region',
       'World Bank income group', 'Country', 'Sex', 'Display Value',
       'Numeric', 'Low', 'High', 'Comments'], dtype=object)
```

In [58]:

```
print("Row names of data frame : for berlin_weather_oldest.csv")
np.array(df_berlin_weather.columns) # Get Mentioned dataframe columns information using numpy arra
y
```

```
Row names of data frame : for berlin_weather_oldest.csv
```

Out[58]:

```
array(['STATION', 'STATION_NAME', 'DATE', 'PRCP', 'SNWD', 'SNOW', 'TMAX',
       'TMIN', 'WDFG', 'PGTM', 'WSFG', 'WT09', 'WT07', 'WT01', 'WT06',
       'WT05', 'WT04', 'WT16', 'WT08', 'WT18', 'WT03'], dtype=object)
```

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

In [59]:

```
# Glimpse of data before column change
print("Data frame :before column rename")
df_data_text.head(2)
```

```
Data frame :before column rename
```

Out[59]:

	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

In [60]:

```
# Rename dataframe columns using pandas rename() method

print("Data frame :after column rename")
df_data_text.rename(columns={'Indicator': 'Indiacator_Id'}).head(2)
```

```
Data frame :after column rename
```

Out[60]:

	Indiacator_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

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

In [61]:

```
# Glimps of data before column change
print("Data frame :before column rename")
df_data_text.head(2)
```

```
Data frame :before column rename
```

Out[61]:

		PUBLISH		WHO	World Bank			Display				

	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

In [62]:

```
print(" Rename column name permanently\n")

# Note : use " inplace=True " property for permanent changes

df_data_text.rename(columns={'Indicator': 'Indiacator_Id'}, inplace=True)
df_data_text.head(2)
```

```
 Rename column name permanently
```

Out[62]:

	Indiacator_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

---

5. Change the names of multiple columns.

In [63]:

```
# Glimps of data before column change
print("Dataframe : before renaming columns")
df_data_text.head(2)
```

```
Dataframe : before renaming columns
```

Out[63]:

	Indiacator_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 [64]:

```
# Rename dataframe columns using pandas rename() method . To ranem multiple columns put valus in d
ictionary format
print(" Rename multiple columns\n")
df_data_text.rename(columns = {'PUBLISH STATES': 'Publication Status', 'WHO region': 'WHO
Region'}, inplace=True)
df_data_text.head(2)
```

```
 Rename multiple columns
```

Out[64]:

	Indiacator_Id	Publication Status	Year	WHO Region	World Bank income group	Country	Sex	Display Value	Numeric	Low	High	Comments

	Indiacator_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 [65]:

```
# to sort values of datafarme /dataset in pandas use sort_values()  function

df_data_text.sort_values('Year', inplace=True) # use 'inplace=True' for making permanent changes i
n dataframe
df_data_text.head(5)
```

Out[65]:

	Indiacator_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 [66]:

```
print("Arrange multiple column values in ascending order")
df_data_text.sort_values(by=['Indiacator_Id','Country','Year','WHO Region','Publication Status'] ,
axis=0, inplace=True)

df_data_text.sort_index(inplace=True) # use sort_index() to dataframe by it's index

df_data_text[['Indiacator_Id','Country','Year','WHO Region','Publication Status']].head(3)
```

```
Arrange multiple column values in ascending order
```

Out[66]:

	Indiacator_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 age 60 (years)	Andorra	2012	Europe	Published

---

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

In [67]:

```
# get list of columns abvailbelbe in datafrme
columns= list(df_data_text.columns)
```

```
columns= list(df_data_text.columns)
# move the column " Country " at first postion
columns.insert(0, columns.pop(columns.index('Country')))

print(" Data frame : after making " 'Country'" as first column" )
df_data_text[columns].head(5)
```

```
 Data frame : after making Country as first column
```

Out[67]:

	Country	Indiacator_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 [68]:

```
print("Column array using a variable ")
WHO_Region_Data=df_data_text['WHO Region'] # Get the data of specific column and assign values to variable
np.array(WHO_Region_Data) # use numpy array to convert Pandas dataframe series object to array
```

```
Column array using a variable
```

Out[68]:

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

10. Get the subset rows 11, 24, 37

In [69]:

```
print(" Get subset data of rows no. 11, 24, 37 ")

# use pandas dataframe .loc methoc to slice or select data at specific position
row_index=[ 11, 24, 37] #temp list
df_data_text.loc[row_index] #fetch data based on index location
```

```
 Get subset data of rows no. 11, 24, 37
```

Out[69]:

	Indiacator_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 [70]:

```
print(" Get subset data of rows  excluding rows no. 5, 12, 23, and 56 \n Output should not display data at rows 5,12,23,56")

exclude_rows = [5, 12, 23, 56]  # create temporary list to crate list type data for index

# isin()  mehtod will select data based upon indexes if data available at those index position
exclude_row_index = df_data_text.index.isin(exclude_rows)

df_data_text[~exclude_row_index].head(14) # use "~ " opeartor to exclude values from dataframe
```

```
 Get subset data of rows  excluding rows no. 5, 12, 23, and 56
 Output should not display data at rows 5,12,23,56
```

Out[70]:

	Indiacator_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
6	Life expectancy at age 60 (years)	Published	1990	Americas	High-income	Antigua and Barbuda	Male	17	17.0	NaN	NaN	NaN
7	Life expectancy at age 60 (years)	Published	2012	Americas	High-income	Antigua and Barbuda	Both sexes	22	22.0	NaN	NaN	NaN
8	Life expectancy at birth (years)	Published	2012	Western Pacific	High-income	Australia	Male	81	81.0	NaN	NaN	NaN
9	Life expectancy at birth (years)	Published	2000	Western Pacific	High-income	Australia	Both sexes	80	80.0	NaN	NaN	NaN
10	Life expectancy at birth (years)	Published	2012	Western Pacific	High-income	Australia	Both sexes	83	83.0	NaN	NaN	NaN
11	Life expectancy at birth (years)	Published	2012	Europe	High-income	Austria	Female	83	83.0	NaN	NaN	NaN
13	Life expectancy at birth (years)	Published	2012	Europe	High-income	Belgium	Female	83	83.0	NaN	NaN	NaN
14	Life expectancy at birth (years)	Published	2000	Eastern Mediterranean	High-income	Bahrain	Male	73	73.0	NaN	NaN	NaN
15	Life expectancy at birth (years)	Published	1990	Eastern Mediterranean	High-income	Bahrain	Female	74	74.0	NaN	NaN	NaN

Part 2

Load datasets from CSV

In [71]:

```
# Load Data Sets
# use Pandas read_csv() to load data from web url/csv
users = pd.read_csv('https://raw.githubusercontent.com/ben519/DataWrangling/master/Data/users.csv'
)
sessions
=pd.read_csv('https://raw.githubusercontent.com/ben519/DataWrangling/master/Data/sessions.csv' )
products
=pd.read_csv('https://raw.githubusercontent.com/ben519/DataWrangling/master/Data/products.csv' )
transactions
=pd.read_csv('https://raw.githubusercontent.com/ben519/DataWrangling/master/Data/transactions.csv'
)
```

In [72]:

```
# gimpse of users data
print("Users Data")
users.head(2) # use head() function display glimpse of data
```

```
Users Data
```

Out[72]:

	UserID	User	Gender	Registered	Cancelled
0	1	Charles	male	2012-12-21	NaN
1	2	Pedro	male	2010-08-01	2010-08-08

In [73]:

```
# gimpse of Sessions data
print("Sessions Data")
sessions.head(2)
```

```
Sessions Data
```

Out[73]:

	SessionID	SessionDate	UserID
0	1	2010-01-05	2
1	2	2010-08-01	2

In [74]:

```
# gimpse of Products data
print("Products Data")
products.head(2)
```

```
Products Data
```

Out[74]:

	ProductID	Product	Price
0	1	A	14.16
1	2	B	33.04

In [75]:

```
# gimpse of Transactions data
print("Transactions Data")
transactions.head(2)
```

```
Transactions Data
```

Out[75]:

	TransactionID	TransactionDate	UserID	ProductID	Quantity
0	1	2010-08-21	7.0	2	1
1	2	2011-05-26	3.0	4	1

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

In [76]:

```
# to join two or more data frame use merge() function

df_data_left= pd.merge(transactions, users, on='UserID',how='left')
print("Data after, Join users to transactions, keeping all rows from transactions and only matchin
g rows from users (left join) ")
df_data_left
```

```
Data after, Join users to transactions, keeping all rows from transactions and only matching rows
from users (left join)
```

Out[76]:

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

13. Which transactions have a UserID not in users?

In [77]:

```
# To get data for those users which are not available in Transactions , get userID infromation fro
m users table
print("Data for those users , whose information is not available in Transaction table ")
transactions[~transactions.UserID.isin(users.UserID)]
```

```
Data for those users , whose information is not available in Transaction table
```

Out[77]:

	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 [78]:

```
df_data_inner= pd.merge(transactions, users, on='UserID',how='inner') # funtion merge() to join two
or more dataset/dataframe
print("Data after, Join users to transactions, keeping only rows from transactions and users that
```

```
match via UserID (inner join) ")
df_data_inner
```

```
Data after, Join users to transactions, keeping only rows from transactions and users that match v
ia UserID (inner join)
```

Out[78]:

	TransactionID	TransactionDate	UserID	ProductID	Quantity	User	Gender	Registered	Cancelled
0	2	2011-05-26	3.0	4	1	Caroline	female	2012-10-23	2016-06-07
1	3	2011-06-16	3.0	3	1	Caroline	female	2012-10-23	2016-06-07
2	7	2013-12-30	3.0	4	1	Caroline	female	2012-10-23	2016-06-07
3	10	2016-05-08	3.0	4	4	Caroline	female	2012-10-23	2016-06-07
4	4	2012-08-26	1.0	2	3	Charles	male	2012-12-21	NaN
5	5	2013-06-06	2.0	4	1	Pedro	male	2010-08-01	2010-08-08
6	6	2013-12-23	2.0	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 [79]:

```
df_data_outer= pd.merge(transactions, users, on='UserID',how='outer') # funtion merge() to join two
or more dataset/dataframe
print("Data after, Join users to transactions, displaying all matching rows AND all non-matching r
ows (full outer join)  ")
df_data_outer
```

```
Data after, Join users to transactions, displaying all matching rows AND all non-matching rows (fu
ll outer join)
```

Out[79]:

	TransactionID	TransactionDate	UserID	ProductID	Quantity	User	Gender	Registered	Cancelled
0	1.0	2010-08-21	7.0	2.0	1.0	NaN	NaN	NaN	NaN
1	9.0	2015-04-24	7.0	4.0	3.0	NaN	NaN	NaN	NaN
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	NaN
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	NaN	NaN
10	NaN	NaN	4.0	NaN	NaN	Brielle	female	2013-07-17	NaN
11	NaN	NaN	5.0	NaN	NaN	Benjamin	male	2010-11-25	NaN

---

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

In [80]:

```
# To get data for sessions , for same day users had been registered ,
# use " UserID Registered " columns from users data and " UserID SessionDate" from Session data
```

```
# use   UserID,Registered   columns from users data and   UserID,SessionDate  from session data


df_user_data = pd.merge(users,sessions, how='inner',left_on=['UserID','Registered'],right_on=['User
ID','SessionDate'])
print("Determine which sessions occurred on the same day each user registered ")
df_user_data
```

```
Determine which sessions occurred on the same day each user registered
```

Out[80]:

	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	NaN	9	2013-07-17

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

In [81]:

```
# Create temp column in users data and products data

users['cross_join']='1'
products['cross_join']='1'

# join the data
df_users_products_data = pd.merge(users, products, on=['cross_join'])

# drop the non-required columns
df_users_products_data = df_users_products_data.drop('cross_join', axis=1)
users=users.drop('cross_join', axis=1)
products=products.drop('cross_join', axis=1)

# display Data
df_users_products=df_users_products_data[["UserID","ProductID"]]
print("dataset with every possible (UserID, ProductID) pair (cross join)")
df_users_products
```

```
dataset with every possible (UserID, ProductID) pair (cross join)
```

Out[81]:

	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
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

	UserID	ProductID
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 [82]:

```
# to get data for quantity for products which was purchased by users
# merge and group the data in users and transaction data based upon userID and ProductID & get sum
of values in quantity column based upon pair of userId and productId (using apply() function)
# and fill na values with zero (0)
df_Quantity_Data=pd.merge(df_users_products, transactions, how='left', on=['UserID', 'ProductID']).
groupby(['UserID', 'ProductID']).apply(lambda x: pd.Series(dict(Quantity=x.Quantity.sum()))).reset_
index().fillna(0)
print("quantity of each product was purchased by each user")
df_Quantity_Data
```

```
quantity of each product was purchased by each user
```

Out[82]:

	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
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 [83]:

```
# merge tarnsaction data with itself to get par of transaction done by user
df_Transaction_Data=pd.merge(transactions,transactions,on="UserID")
print("For each user, get each possible pair of pair transactions (TransactionID1,
TransacationID2) ")
df_Transaction_Data
```

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

Out[83]:

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

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

In [84]:

```
# first tarnsaction of user based upon user Id , by sing first() method which specifically work wi
th date/time data
# get data from tansations for users first transaction
df_first_transaction= transactions.groupby('UserID').first().reset_index()

df_first_transaction
```

Out[84]:

	UserID	TransactionID	TransactionDate	ProductID	Quantity
0	1.0	4	2012-08-26	2	3
1	2.0	5	2013-06-06	4	1

	UserID	TransactionID	TransactionDate	ProductID	Quantity
2	3.0	2	2011-05-26	4	1
3	7.0	1	2010-08-21	2	1

In [85]:

```
# use data from tansations for users first transaction i.e "df_first_transaction" ,
# merge it with users data to get users first tarnsaction
df_user_Transactions = pd.merge(users,df_first_transaction,on="UserID",how="left")
print("users data based upon first occuring transaction in the transactions table ")
df_user_Transactions
```

```
users data based upon first occuring transaction in the transactions table
```

Out[85]:

	UserID	User	Gender	Registered	Cancelled	TransactionID	TransactionDate	ProductID	Quantity
0	1	Charles	male	2012-12-21	NaN	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	NaN	NaN	NaN	NaN	NaN
4	5	Benjamin	male	2010-11-25	NaN	NaN	NaN	NaN	NaN

21. Test to see if we can drop columns

In [86]:

```
# Get cloumn's list

my_columns= list(df_user_Transactions) # convert dataframe columns into list
my_columns
```

Out[86]:

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

In [87]:

```
# set threshold to drop NAs

list(df_user_Transactions.dropna(thresh=int(df_user_Transactions.shape[0] * .9), axis=1).columns)
```

Out[87]:

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

In [88]:

```
#get columns information which have null values
missing_info = list(df_user_Transactions.columns[df_user_Transactions.isnull().any()])
missing_info
```

Out[88]:

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

In [89]:

```
# count of missing values in cloumns

print("Output: Count of missing data\n")
for col in missing_info:
    num_missing = df_user_Transactions[df_user_Transactions[col].isnull() == True].shape[0]
    print('number missing for column {}: {}'.format(col, num_missing))
```

```
Output: Count of missing data

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
```

In [90]:

```
# percentage  of missing values in cloumns

print("Output of percentage missing data\n")
for col in missing_info:
    percent_missing = df_user_Transactions[df_user_Transactions[col].isnull() == True].shape[0]/df_
user_Transactions.shape[0]
    print('percent missing for column {}: {}'.format(col, percent_missing))
```

```
Output of percentage missing data

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
```