Pandas

Reading the data from csv files

In [2]:

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

In [3]:

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

Out[3]:

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

In [4]:

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

Out[4]:

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

2 rows × 21 columns

1. Get the Metadata from the above files.

In [5]:

```
df.info()
```

```
<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 [6]:

```
df1.info()
```

```
<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 [13]:

```
row = df.index.values
row
```

Out[13]:

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

In [14]:

```
row1=df1.index.values
row1
```

Out[14]:

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

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

In [17]:

```
df.rename(columns={'Indicator':'Indicator_id'}).head(2)
```

Out[17]:

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

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

In [19]:

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

Out[19]:

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

5. Change the names of multiple columns.

In [20]:

```
df.rename(columns={'PUBLISH STATES':'Publication Status','WHO region':'WHO Regio
n'}, inplace=True)
df.head(2)
```

Out[20]:

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

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

In [22]:

```
df.sort_values(by=['Year']).head(5)
```

Out[22]:

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

7. Arrange multiple column values in ascending order.

In [23]:

```
newdf=df.loc[:,['Indicator_id','Country','Year','WHO Region','Publication Statu
s']]
newdf.sort_values(by=['Country','Year'])
newdf.head(4)
```

Out[23]:

	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 age 60 (years)	Andorra	2012	Europe	Published
3	Life expectancy at age 60 (years)	Andorra	2000	Europe	Published

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

In [24]:

```
cols = list(df)
# move the column to head of list using index, pop and insert
cols.insert(0, cols.pop(cols.index('Country')))
# using loc to reorder the columns
df_update = df.loc[:, cols]
df_update.head(4)
```

Out[24]:

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

9. Get the column array using a variable

In [25]:

```
col2=df['WHO Region'].values
#Displaying vallues of the array variable
col2
```

Out[25]:

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

10. Get the subset rows 11, 24, 37

In [26]:

```
subset=df.iloc[[11,24,37],:]
subset
```

Out[26]:

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

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

In [27]:

```
# Dropping the rows 5,12,23 and 56 using drop by row index vlues
df.drop([5,12,23,56]).head(10) # row index 5 is missed in below output of first
 10 rows
```

Out[27]:

	Indicator_id	Publication Status	Year	WHO Region	World Bank income group	Country	Sex	Display Value	Numeric
0	Life expectancy at birth (years)	Published	1990	Europe	High-income	Andorra	Both sexes	77	77.0
1	Life expectancy at birth (years)	Published	2000	Europe	High-income	Andorra	Both sexes	80	80.0
2	Life expectancy at age 60 (years)	Published	2012	Europe	High-income	Andorra	Female	28	28.0
3	Life expectancy at age 60 (years)	Published	2000	Europe	High-income	Andorra	Both sexes	23	23.0
4	Life expectancy at birth (years)	Published	2012	Eastern Mediterranean	High-income	United Arab Emirates	Female	78	78.0
6	Life expectancy at age 60 (years)	Published	1990	Americas	High-income	Antigua and Barbuda	Male	17	17.0
7	Life expectancy at age 60 (years)	Published	2012	Americas	High-income	Antigua and Barbuda	Both sexes	22	22.0
8	Life expectancy at birth (years)	Published	2012	Western Pacific	High-income	Australia	Male	81	81.0
9	Life expectancy at birth (years)	Published	2000	Western Pacific	High-income	Australia	Both sexes	80	80.0
10	Life expectancy at birth (years)	Published	2012	Western Pacific	High-income	Australia	Both sexes	83	83.0

Load datasets from CSV

In [28]:

```
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 [29]:

```
users.head(2)
```

Out[29]:

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

In [30]:

```
sessions.head(2)
```

Out[30]:

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

In [31]:

```
products.head(2)
```

Out[31]:

	ProductID	Product	Price
0	1	A	14.16
1	2	B	33.04

In [32]:

```
transactions.head(2)
```

Out[32]:

	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 [33]:

```
result = pd.merge(transactions, users, on='UserID', how='left', sort=False);
result
```

Out[33]:

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

13. Which transactions have a UserID not in users?

In [34]:

```
Merged = pd.merge(transactions, users, on='UserID', how='left', sort=False,indic
ator=True);
result1=Merged[Merged['_merge']=='left_only']
result1.iloc[:,0:5]
```

Out[34]:

	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 [35]:

```
result2 = pd.merge(transactions, users, on='UserID', how='inner', sort=False);
result2
```

Out[35]:

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

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

In [36]:

```
result3 = pd.merge(transactions, users, on='UserID', how='outer', sort=False);
result3
```

Out[36]:

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

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

In [37]:

```
result4=pd.merge(users,sessions,left_on=[users.UserID,users.Registered],right_on
=[sessions.UserID,sessions.SessionDate],copy=False,sort=False)
result4.iloc[:,2:-1]
```

Out[37]:

	UserID_x	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 [39]:

```
users['temp']=1
products['temp']=1
result5=pd.merge(users,products,on='temp',how='outer')
result5.loc[:,["UserID","ProductID"]]
```

Out[39]:

	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
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 [ ]:

```
df_Quantity_Data=pd.merge(products, transactions, how='left', on=['UserID', 'Pro
ductID']).groupby(['UserID', 'ProductID']).apply(lambda x: pd.Series(dict(Quanti
ty=x.Quantity.sum()))).reset_index().fillna(0)
print("quantity of each product was purchased by each user")
df_Quantity_Data
```

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

In [94]:

```
df_Transaction_Data=pd.merge(transactions,transactions,on="UserID")
print("For each user, get each possible pair of pair transactions (TransactionID 1, TransacationID2) ")
df_Transaction_Data
```

```
For each user, get each possible pair of pair transactions (Transact
ionID1, TransacationID2)
```

Out[94]:

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

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

In [95]:

```
df_first_transaction= transactions.groupby('UserID').first().reset_index()
df_first_transaction
```

Out[95]:

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

In [96]:

```
df_user_Transactions = pd.merge(users,df_first_transaction,on="UserID",how="lef
t")
print("users data based upon first occuring transaction in the transactions tabl
e ")
df_user_Transactions
```

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

Out[96]:

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

21. Test to see if we can drop columns

In [97]:

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

Out[97]:

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

In [98]:

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

Out[98]:

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

In [103]:

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

Out[103]:

```
['Cancelled', 'TransactionID', 'TransactionDate', 'ProductID', 'Quan
tity']
```

In [102]:

```
# 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() == Tru
e].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 [101]:

```
# 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
```

In [ ]:

```

```