In [1]:
```
import pandas as pd
df =pd.read_csv('https://raw.githubusercontent.com/jackiekazil/data-wrangling/mast
df.head(2)
df1 =pd.read_csv('https://raw.githubusercontent.com/kjam/data-wrangling-pycon/mast
df1.head(2)
```

Out[1]:

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

2 rows × 21 columns

In [2]:
```
# Get the Metadata from the above files.

df.info()
df1.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
<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
```

In [3]:
```
# Get the row names from the above files.

df.index.values
```

Out[3]:
```
array([   0,    1,    2, ..., 4653, 4654, 4655], dtype=int64)
```

In [4]:
```
df1.index.values
```

Out[4]:
```
array([     0,      1,      2, ..., 117205, 117206, 117207], dtype=int64)
```

In [5]:
```
# Change the column name from any of the above file.

df.rename(columns = {'Indicator' : 'Indicator_ID'}, inplace=False)
df.head()
```

Out[5]:

	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	N
1	Life expectancy at birth (years)	Published	2000	Europe	High-income	Andorra	Both sexes	80	80.0	NaN	N
2	Life expectancy at age 60 (years)	Published	2012	Europe	High-income	Andorra	Female	28	28.0	NaN	N
3	Life expectancy at age 60 (years)	Published	2000	Europe	High-income	Andorra	Both sexes	23	23.0	NaN	N
4	Life expectancy at birth (years)	Published	2012	Eastern Mediterranean	High-income	United Arab Emirates	Female	78	78.0	NaN	N

In [6]:
```
# Change the column name from any of the above file and store the changes made per

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

Out[6]:

	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
2	Life expectancy at age 60 (years)	Published	2012	Europe	High-income	Andorra	Female	28	28.0	NaN
3	Life expectancy at age 60 (years)	Published	2000	Europe	High-income	Andorra	Both sexes	23	23.0	NaN
4	Life expectancy at birth (years)	Published	2012	Eastern Mediterranean	High-income	United Arab Emirates	Female	78	78.0	NaN

In [7]:
```
# Change the names of multiple columns.

df.rename(columns = {'PUBLISH STATES' : 'Publication Status', 'WHO region' : 'WHO
df.head()
```

Out[7]:

	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
2	Life expectancy at age 60 (years)	Published	2012	Europe	High-income	Andorra	Female	28	28.0	NaN
3	Life expectancy at age 60 (years)	Published	2000	Europe	High-income	Andorra	Both sexes	23	23.0	NaN
4	Life expectancy at birth (years)	Published	2012	Eastern Mediterranean	High-income	United Arab Emirates	Female	78	78.0	NaN

In [8]:
```
# Arrange values of a particular column in ascending order.

df.sort_values(by=['Year'], ascending=True)
```

Out[8]:

	Indicator_ID	Publication Status	Year	WHO Region	World Bank income group	Country	Sex	Display Value	Numeri
0	Life expectancy at birth (years)	Published	1990	Europe	High-income	Andorra	Both sexes	77	77.
1270	Life expectancy at birth (years)	Published	1990	Europe	High-income	Germany	Male	72	72.
3193	Life expectancy at birth (years)	Published	1990	Europe	Lower-middle-income	Republic of Moldova	Male	65	65.

In [9]:
```
# Arrange multiple column values in ascending order.

df.sort_values(by=['Indicator_ID', 'Country', 'Year', 'WHO Region','Publication St
```

Out[9]:

	Indicator_ID	Publication Status	Year	WHO Region	World Bank income group	Country	Sex	Display Value	Numeri
2426	Healthy life expectancy (HALE) at birth (years)	Published	2000	Africa	Low-income	Zimbabwe	Male	37	37.
2797	Healthy life expectancy (HALE) at birth (years)	Published	2000	Africa	Low-income	Zimbabwe	Female	36	36.
3886	Healthy life expectancy (HALE) at birth (years)	Published	2000	Africa	Low-income	Zimbabwe	Both sexes	37	37.
	Healthy life				Low		Both		

In [ ]:
```

```

In [10]:
```
# Make country as the first column of the dataframe.

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

Out[10]:

	Country	Indicator_ID	Publication Status	Year	WHO Region	World Bank income group	Sex	Display Value	Numeri
0	Andorra	Life expectancy at birth (years)	Published	1990	Europe	High-income	Both sexes	77	77.
1	Andorra	Life expectancy at birth (years)	Published	2000	Europe	High-income	Both sexes	80	80.
2	Andorra	Life expectancy at age 60 (years)	Published	2012	Europe	High-income	Female	28	28.
		Life expectancy				High-	Both		

In [11]:
```
# Get the column array using a variable Expected Output:

col1 = 'Country'
df[[col1]].values[:, 0]
```

Out[11]:
```
array(['Andorra', 'Andorra', 'Andorra', ..., 'South Africa', 'Zambia',
       'Zimbabwe'], dtype=object)
```

In [12]:
```
# Get the subset rows 11, 24, 37

df.iloc[[11, 24, 37]]
```

Out[12]:

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

In [13]:
```
# Get the subset rows excluding 5, 12, 23, and 56

df.drop([5, 12, 23, 56], axis= 0)
```

Out[13]:

	Indicator_ID	Publication Status	Year	WHO Region	World Bank income group	Country	Sex	Display Value	Numeri
0	Life expectancy at birth (years)	Published	1990	Europe	High-income	Andorra	Both sexes	77	77.
1	Life expectancy at birth (years)	Published	2000	Europe	High-income	Andorra	Both sexes	80	80.
2	Life expectancy at age 60 (years)	Published	2012	Europe	High-income	Andorra	Female	28	28.
	Life								

In [14]:
```
# Load into users dataframe

users = pd.read_csv('https://raw.githubusercontent.com/ben519/DataWrangling/master

users.head()
```

Out[14]:

	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 [16]:
```
# Load into sessions dataframe

sessions =pd.read_csv('https://raw.githubusercontent.com/ben519/DataWrangling/mast

sessions.head()
```

Out[16]:

	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 [17]:
```
# Load into products dataframe

products = pd.read_csv('https://raw.githubusercontent.com/ben519/DataWrangling/mas

products.head()
```

Out[17]:

	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 [18]:
```
# Load into transactions dataframe

transactions = pd.read_csv('https://raw.githubusercontent.com/ben519/DataWrangling

transactions.head()
```

Out[18]:

	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 [19]:
```
print(users['Registered'].dtype)
print(users['Cancelled'].dtype)
print(sessions['SessionDate'].dtype)
print(transactions['TransactionDate'].dtype)
```

```
datetime64[ns]
datetime64[ns]
object
object
```

In [20]:
```
#converting to datetime values using to_datetime method in pandas as these column

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'])
```

In [21]:
```
# Join users to transactions, keeping all rows from transactions and only matching

df_left_trans_users = pd.merge(transactions,users,how="left", on="UserID")

df_left_trans_users
```

Out[21]:

	TransactionID	TransactionDate	UserID	ProductID	Quantity	User	Gender	Registered	Canc
0	1	2010-08-21	7.0	2	1	NaN	NaN	NaT	
1	2	2011-05-26	3.0	4	1	Caroline	female	2012-10-23	201
2	3	2011-06-16	3.0	3	1	Caroline	female	2012-10-23	201
3	4	2012-08-26	1.0	2	3	Charles	male	2012-12-21	
4	5	2013-06-06	2.0	4	1	Pedro	male	2010-08-01	201
5	6	2013-12-23	2.0	5	6	Pedro	male	2010-08-01	201
6	7	2013-12-30	3.0	4	1	Caroline	female	2012-10-23	201
7	8	2014-04-24	NaN	2	3	NaN	NaN	NaT	
8	9	2015-04-24	7.0	4	3	NaN	NaN	NaT	
9	10	2016-05-08	3.0	4	4	Caroline	female	2012-10-23	201

In [22]:
```
# Which transactions have a UserID not in users?

transactions[~transactions['UserID'].isin(users['UserID'])]
```

Out[22]:

	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

In [23]:
```
# Join users to transactions, keeping only rows from transactions and users that m

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

df_Inner_trans_users
```

Out[23]:

	TransactionID	TransactionDate	UserID	ProductID	Quantity	User	Gender	Registered	Canc
0	2	2011-05-26	3.0	4	1	Caroline	female	2012-10-23	201
1	3	2011-06-16	3.0	3	1	Caroline	female	2012-10-23	201
2	7	2013-12-30	3.0	4	1	Caroline	female	2012-10-23	201
3	10	2016-05-08	3.0	4	4	Caroline	female	2012-10-23	201
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	201
6	6	2013-12-23	2.0	5	6	Pedro	male	2010-08-01	201

In [24]:
```
# Join users to transactions, displaying all matching rows AND all non-matching ro

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

df_Outer_trans_users
```

Out[24]:

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

In [25]:
```
# Determine which sessions occurred on the same day each user registered

# Using Panda Merge

pd.merge(left=users,right=sessions,how="inner", left_on=['UserID','Registered'], r
```

Out[25]:

	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

In [31]:
```
# Build a dataset with every possible (UserID, ProductID) pair (cross join)

#create two different dataframes with unique UserID and ProductID from users and t

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

df_userid['Key'] = 1
df_Tran['Key'] = 1
```

In [32]:
```
#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 [33]:
```
#final dataframe df_out which has every possible(UserID,ProductID) combination

df_out
```

Out[33]:

	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

In [34]:
```
# Determine how much quantity of each product was purchased by each user

#do a left join on the output table df_out from previous step with transactions ta

df_user_prod_quant = pd.merge(df_out,transactions,how='left',on=['UserID','Product

#Groupby the table on ['UserID','ProductID] and calculate the sum of Qunatity enti

df_user_quantity = df_user_prod_quant.groupby(['UserID','ProductID'])['Quantity'].

#reset index so that the index column will have consecutive default numbers and fi

df_user_quantity.reset_index().fillna(0)
```

Out[34]:

	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

In [36]:
```
# For each user, get each possible pair of pair transactions (TransactionID1,Trans

pd.merge(transactions,transactions,how='outer',on='UserID')
```

Out[36]:

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

In [38]:
```
# Join each user to his/her first occuring transaction in the transactions table

df_usertran = pd.merge(users,transactions,how='left',on='UserID')

# craete a new dataframe df_ with all duplicates on UserID being dropped , only ke

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	Qu
0	1	Charles	male	2012-12-21	NaT	4.0	2012-08-26	2.0	
1	2	Pedro	male	2010-08-01	2010-08-08	5.0	2013-06-06	4.0	
2	3	Caroline	female	2012-10-23	2016-06-07	2.0	2011-05-26	4.0	
3	4	Brielle	female	2013-07-17	NaT	NaN	NaT	NaN	
4	5	Benjamin	male	2010-11-25	NaT	NaN	NaT	NaN	

In [39]:
```
# Test to see if we can drop columns
```

In [40]:
```
# #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', 'Tran
sactionDate', 'ProductID', 'Quantity']
```

In [43]:
```
#set threshold to drop NAs

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

Out[43]:
```
['UserID', 'User', 'Gender', 'Registered']
```

In [44]:
```
missing_info = list(df_.columns[df_.isnull().any()])

missing_info
```

Out[44]:
```
['Cancelled', 'TransactionID', 'TransactionDate', 'ProductID', 'Quantity']
```

In [45]:
```
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
```

In [46]:
```
for col in missing_info:
    num_missing = df_[df_[col].isnull() ==True].shape[0]
    print('number missing for column {}: {}'.format(col, num_missing))

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
```