**Project-1**

# Get the Metadata from the above files.

Answer :

import pandas as pd

import numpy as np

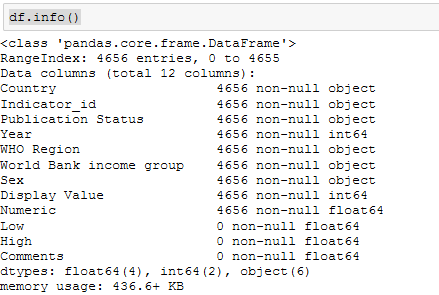
import matplotlib.pyplot as plt

%matplotlib inline

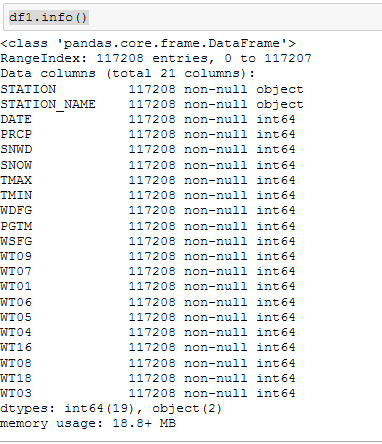
df=pd.read\_csv('https://raw.githubusercontent.com/jackiekazil/data-wrangling/master/data/chp3/data-text.csv')

df1=pd.read\_csv('https://raw.githubusercontent.com/kjam/data-wrangling-pycon/master/data/berlin\_weather\_oldest.csv ')

df.info()



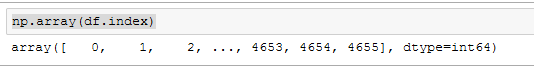
df1.info()



# Get the row names from the above files.

Answer:

np.array(df.index)



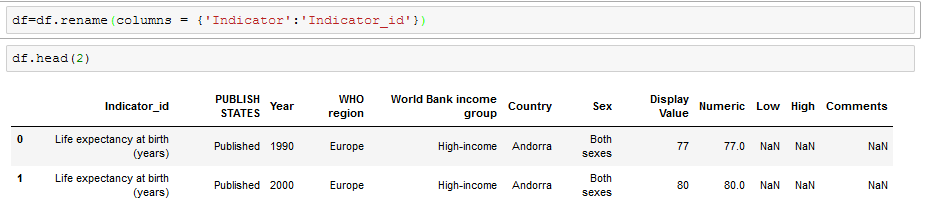
np.array(df1.index)



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

Answer:

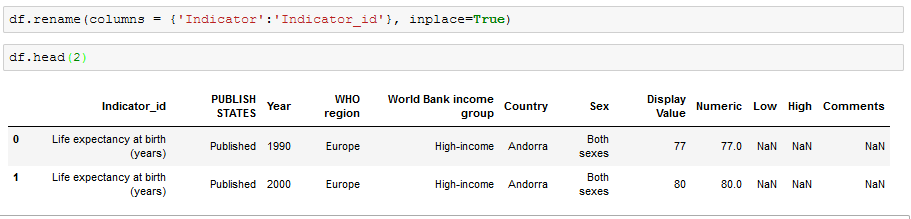
df.rename(columns = {'Indicator':'Indicator\_id'})



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

Answer :

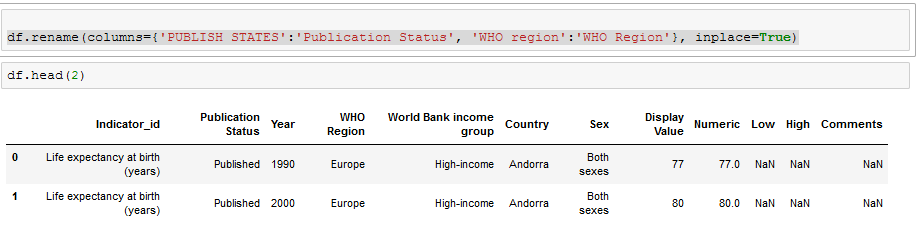
df.rename(columns = {'Indicator':'Indicator\_id'}, inplace=True)



# Change the names of multiple columns.

Answer :

df.rename(columns={'PUBLISH STATES':'Publication Status', 'WHO region':'WHO Region'}, inplace=True)



# Arrange values of a particular column in ascending order.

Answer:

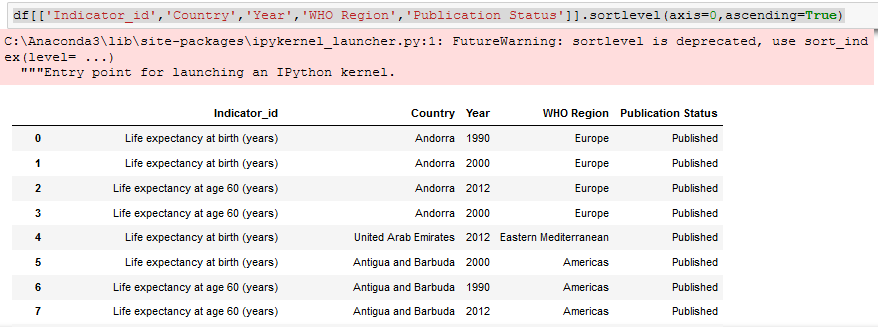
df.sort\_values('Year')



# Arrange multiple column values in ascending order.

Answer:

df[['Indicator\_id','Country','Year','WHO Region','Publication Status']].sortlevel(axis=0,ascending=True)



# Make country as the first column of the dataframe.

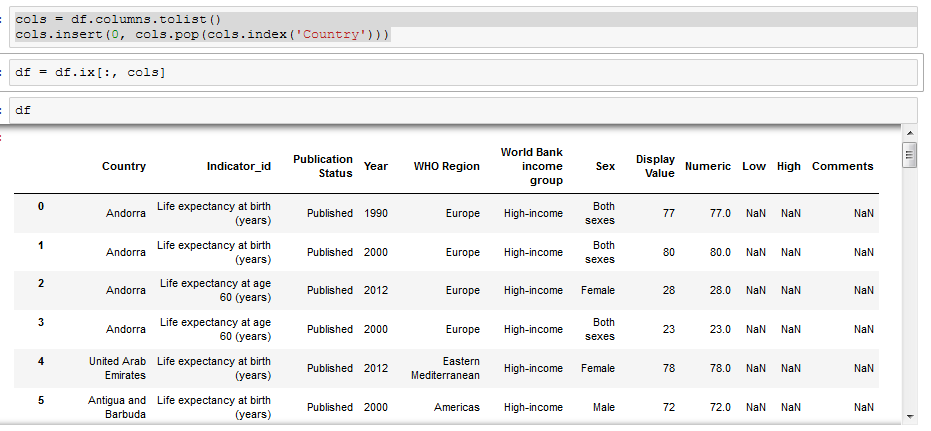
Answer :

cols = df.columns.tolist()

cols.insert(0, cols.pop(cols.index('Country')))

df = df.ix[:, cols]

df



# Get the column array using a variable

Answer:

np.array(df["WHO Region"])



# Get the subset rows 11, 24, 37

Answer:

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



# Get the subset rows excluding 5, 12, 23, and 56

Answer:

exclude\_df = df.index.isin([5, 12, 23,56])

bad\_df

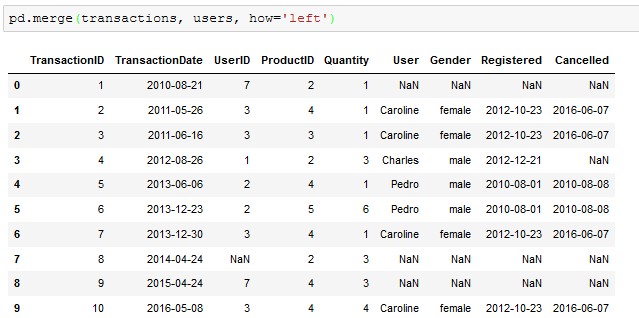
df[~exclude\_df]



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

Answer:

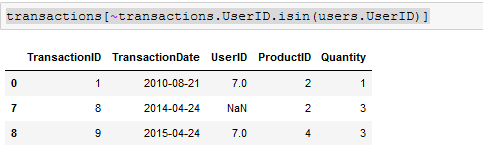
pd.merge(transactions, users, how='left')



# Which transactions have a UserID not in users?

Answer:

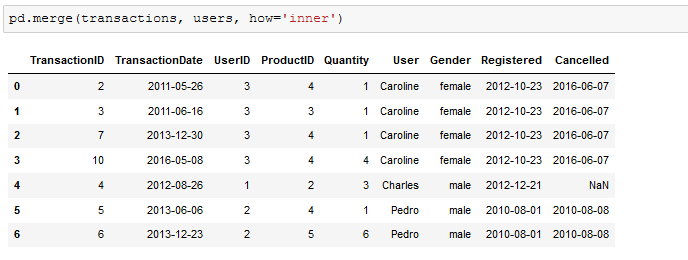
transactions[~transactions.UserID.isin(users.UserID)]



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

Answer:

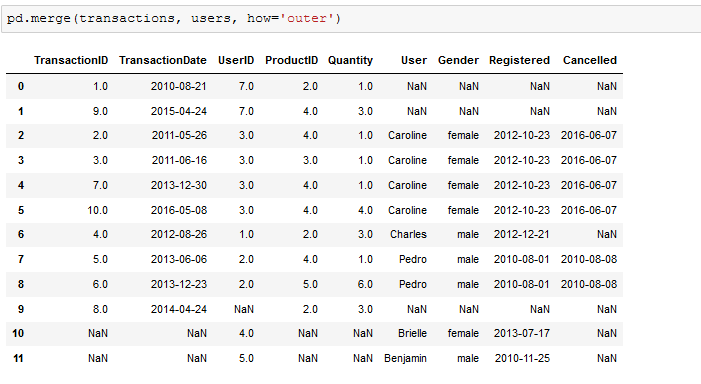
pd.merge(transactions, users, how='inner')



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

Answer

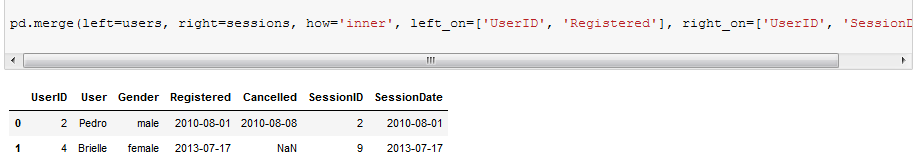
pd.merge(transactions, users, how='outer')



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

Answer:

pd.merge(left=users, right=sessions, how='inner', left\_on=['UserID', 'Registered'], right\_on=['UserID', 'SessionDate'])



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

Answer:

pysqldf(""" SELECT m.UserID ,b.ProductID

FROM users m

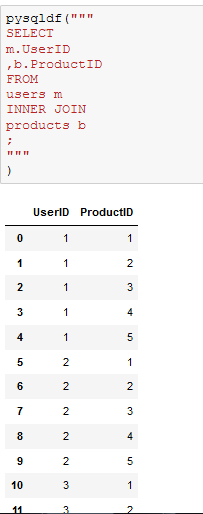
INNER JOIN

products b

;

"""

)



# Determine how much quantity of each product was purchased by each user

Answer:

pysqldf("""

SELECT

UserID ,ProductID,sum(Quantity) Quantity

FROM

(

SELECT

u.UserID

,p.ProductID, 0 as Quantity

FROM

users u

INNER JOIN

products p

Union All

SELECT

m.UserID ,b.ProductID,b.Quantity

FROM

users m

INNER JOIN

transactions b

Where m.UserID = b.UserID order by m.UserID,b.Productid

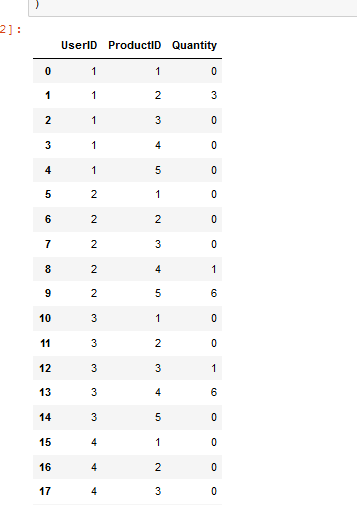
)

Group by UserID,ProductID;

"""

)





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

Answer:

pysqldf("""

SELECT t1.\*,t2.\*

from

transactions t1

INNER JOIN

transactions t2

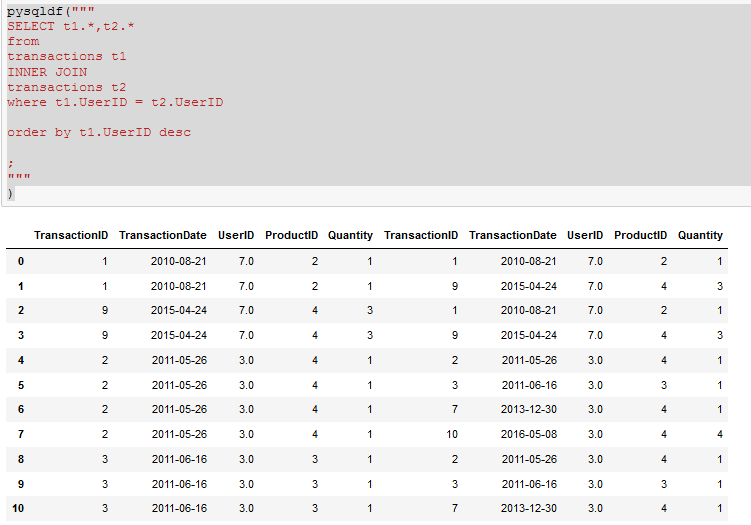
where t1.UserID = t2.UserID

order by t1.UserID desc

;

"""

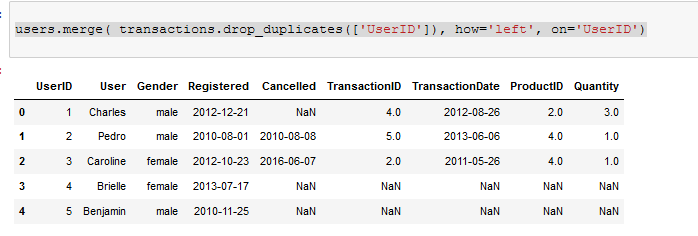
)



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

Answer:

users.merge( transactions.drop\_duplicates(['UserID']), how='left', on='UserID')



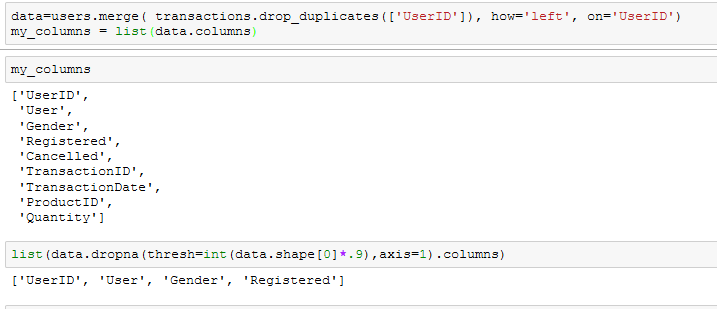
# Test to see if we can drop columns

Answer:

data=users.merge( transactions.drop\_duplicates(['UserID']), how='left', on='UserID')

my\_columns = list(data.columns)

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

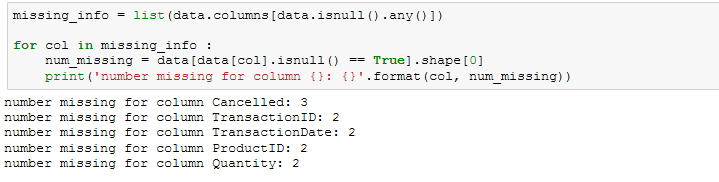


missing\_info = list(data.columns[data.isnull().any()])

for col in missing\_info :

num\_missing = data[data[col].isnull() == True].shape[0]

print('number missing for column {}: {}'.format(col, num\_missing))



num\_missing = data[data[col].isnull() == True].shape[0]

for col in missing\_info:

percent\_missing = data[data[col].isnull() == True].shape[0] / data.shape[0]

print('percent missing for column {}: {}'.format(col, percent\_missing))

