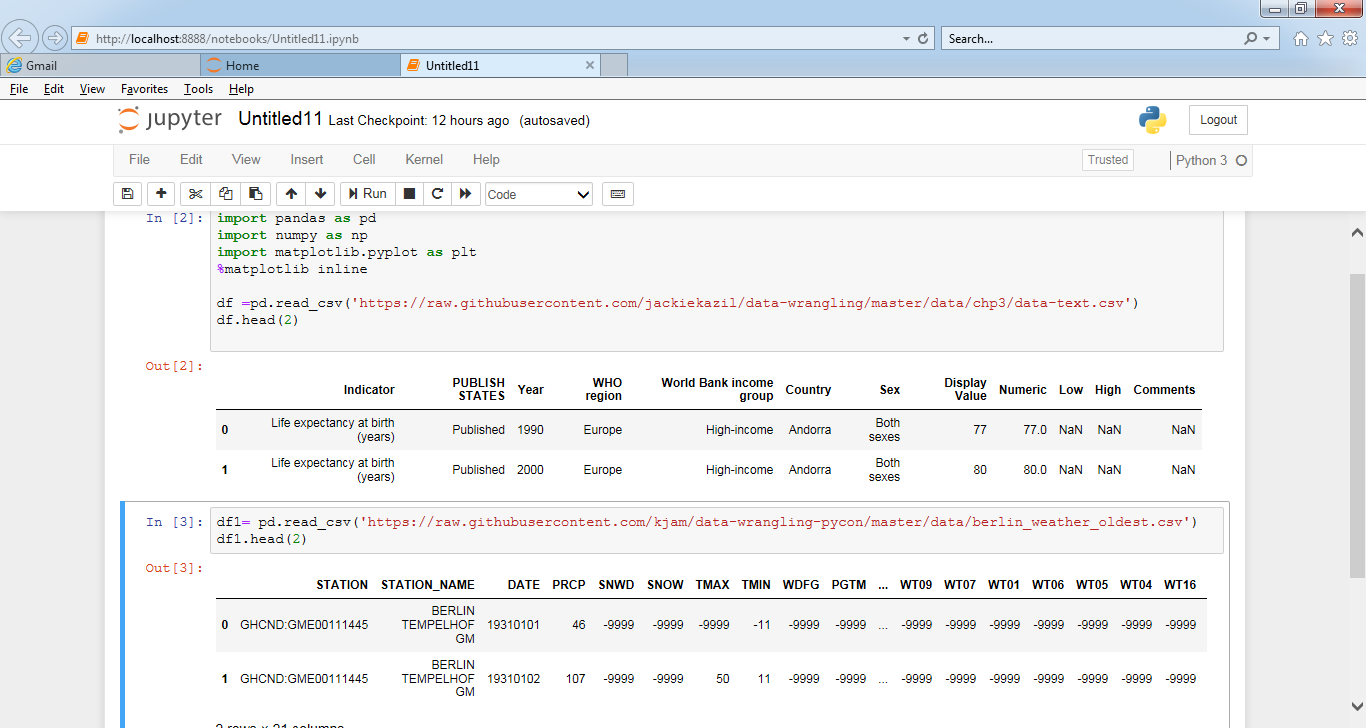
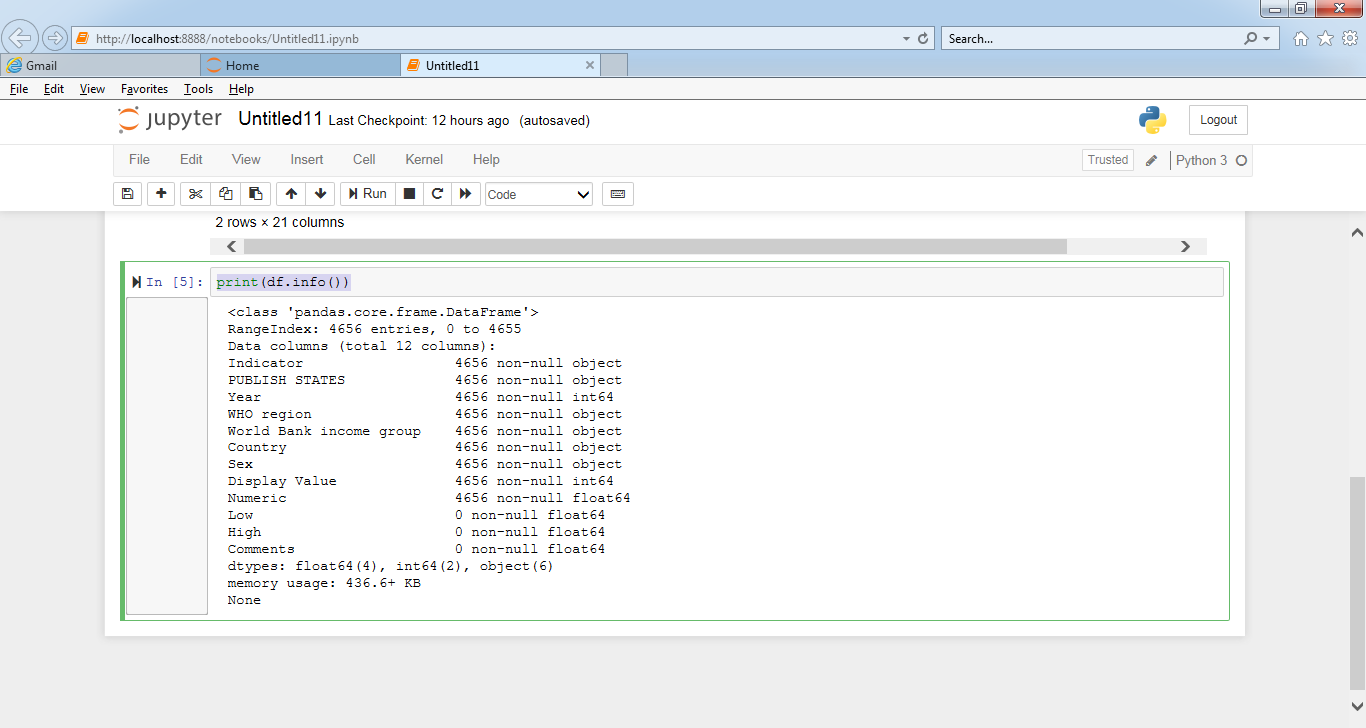
Read Dataset:



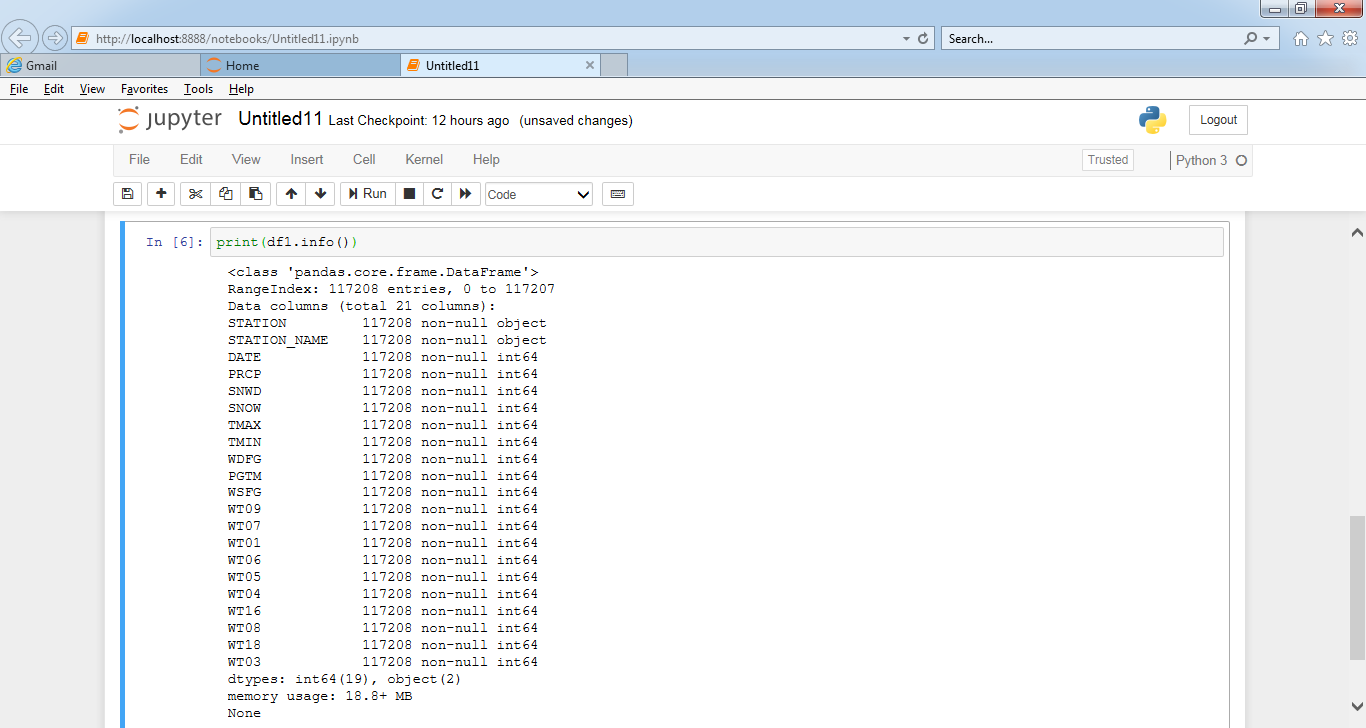
1. Get the Metadata from the above files.

OutPut:

print(df.info())



print(df1.info())



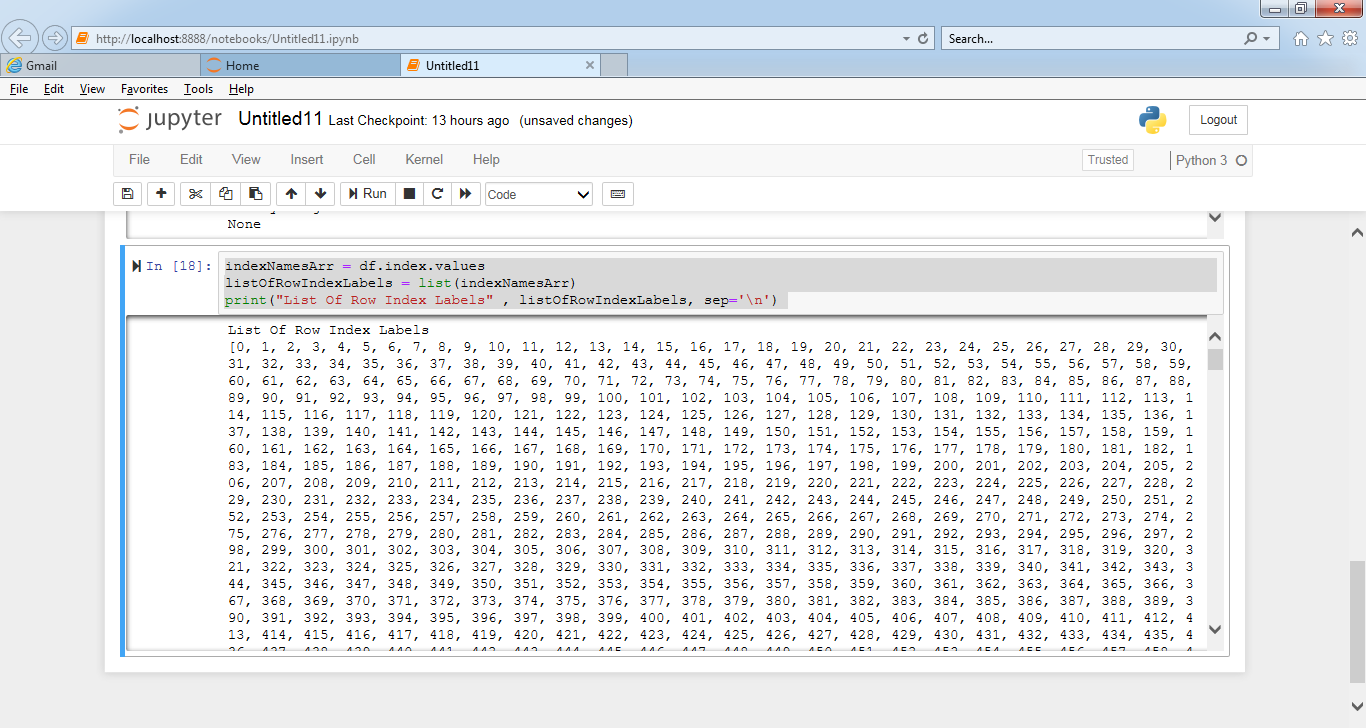
2. Get the row names from the above files.

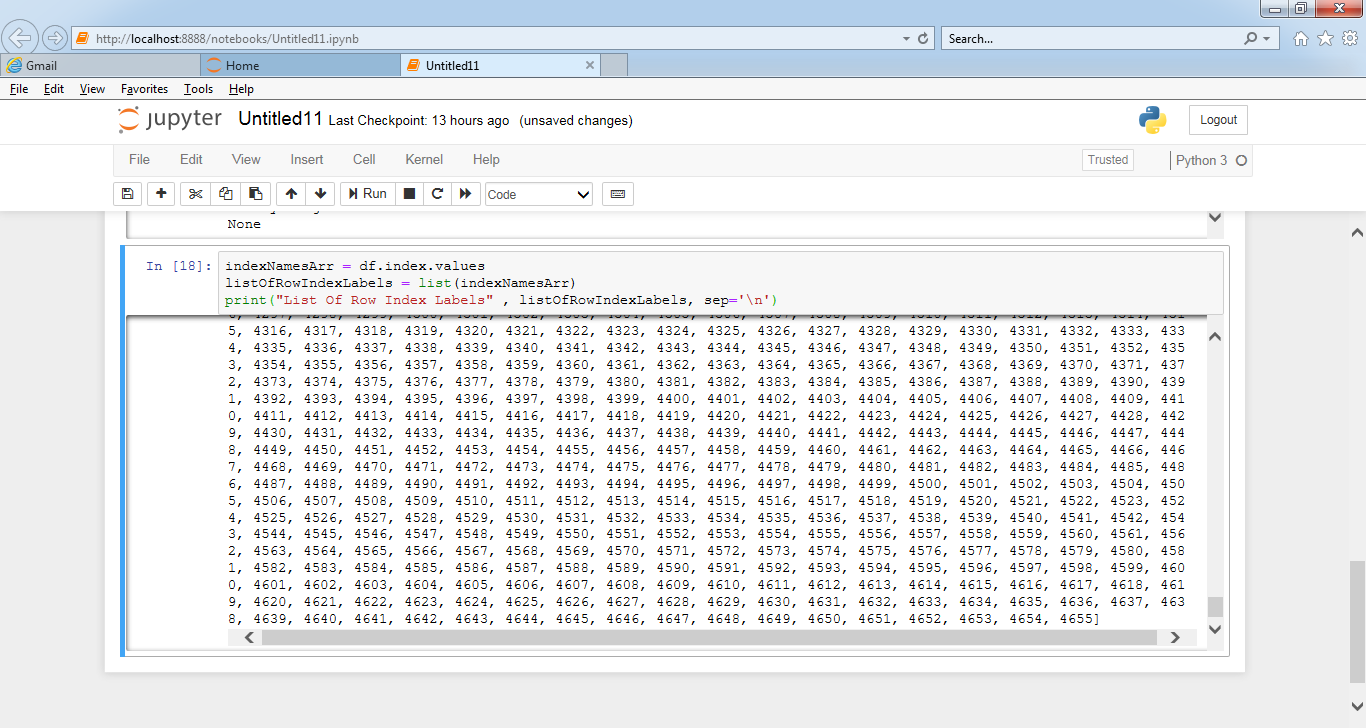
Code: indexNamesArr = df.index.values

listOfRowIndexLabels = list(indexNamesArr)

print("List Of Row Index Labels" , listOfRowIndexLabels, sep='\n')

Output :

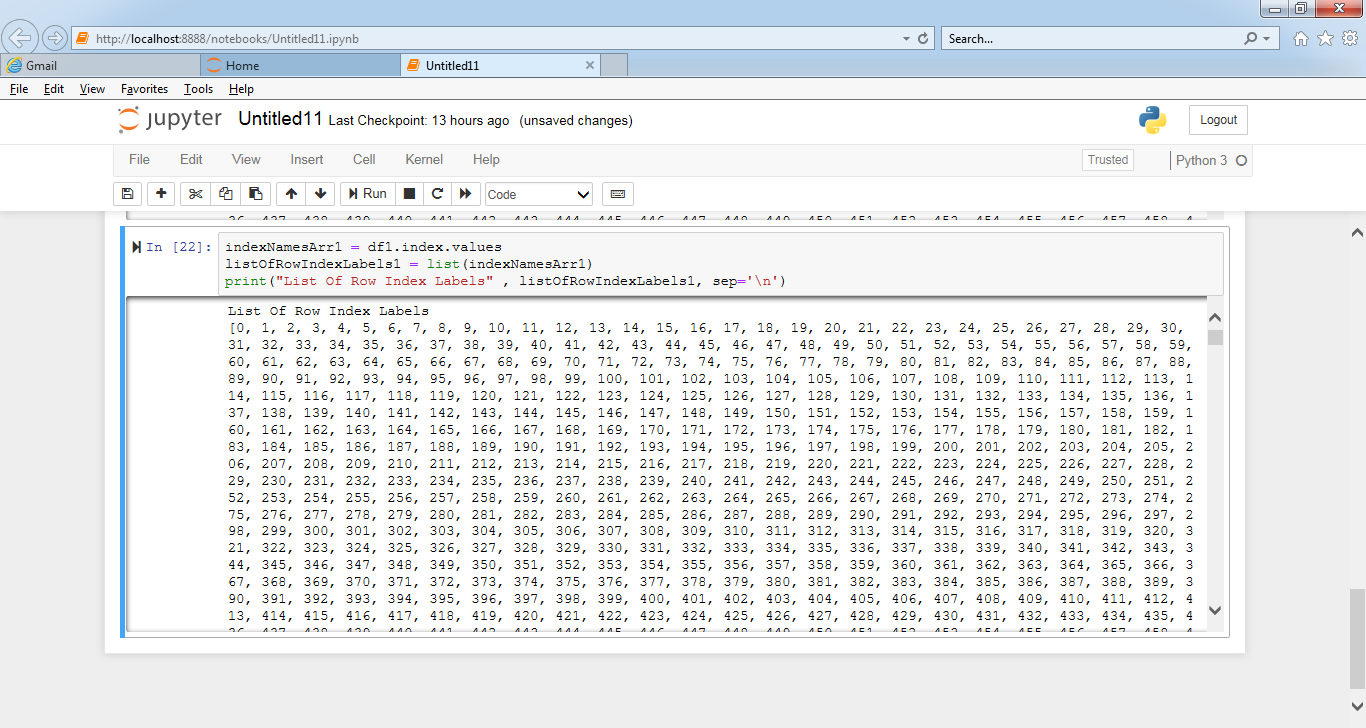


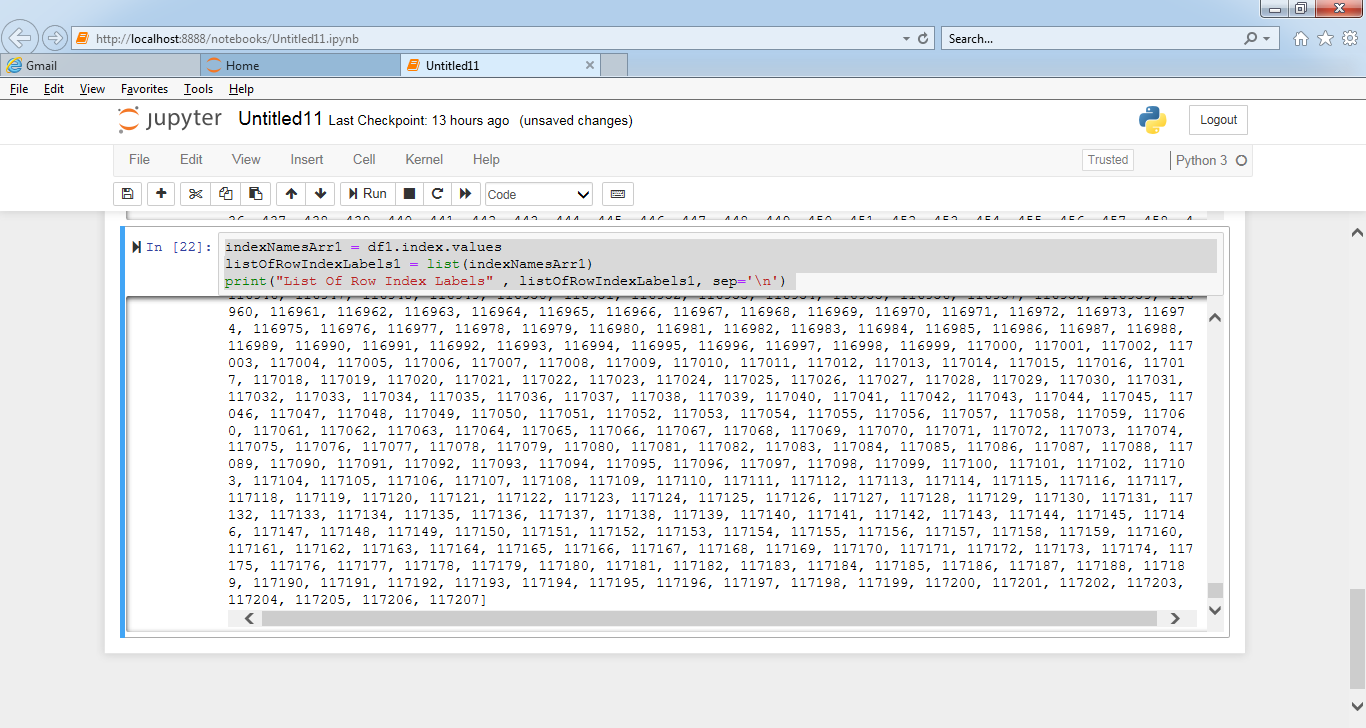


indexNamesArr1 = df1.index.values

listOfRowIndexLabels1 = list(indexNamesArr1)

print("List Of Row Index Labels" , listOfRowIndexLabels1, sep='\n')



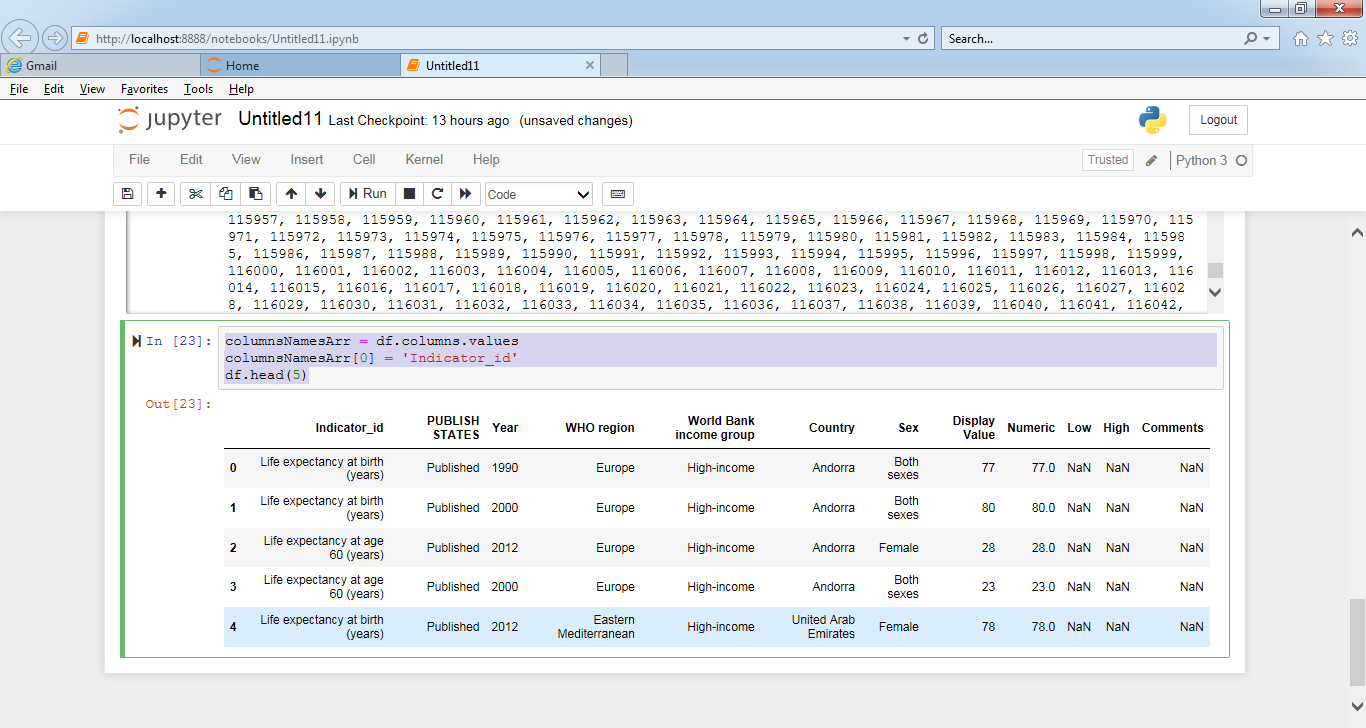


3. Change the column name from any of the above file.

columnsNames = list(df)

columnsNames[0] = 'Indicator\_id'

df.head(5)

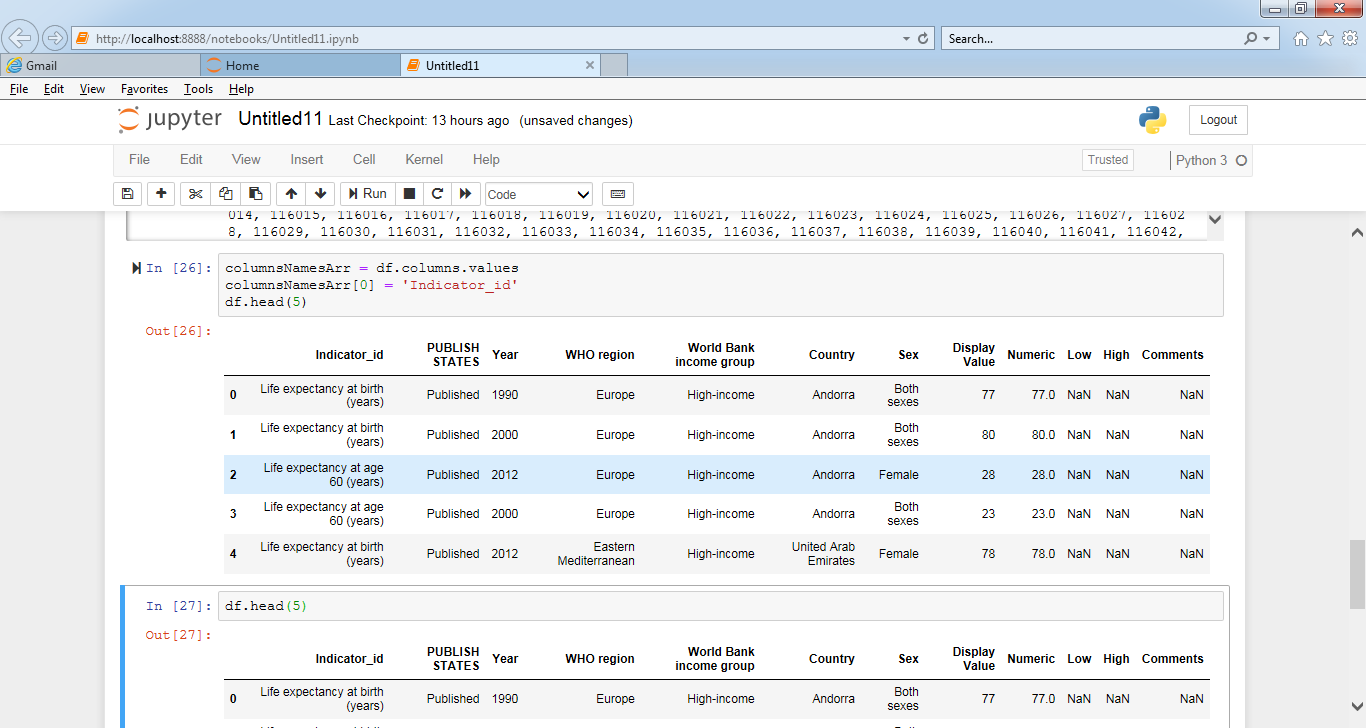


4. Change the column name from any of the above file and store the changes made permanently.

columnsNamesArr = df.columns.values

columnsNamesArr[0] = 'Indicator\_id'

df.head(5)



5. Change the names of multiple columns.

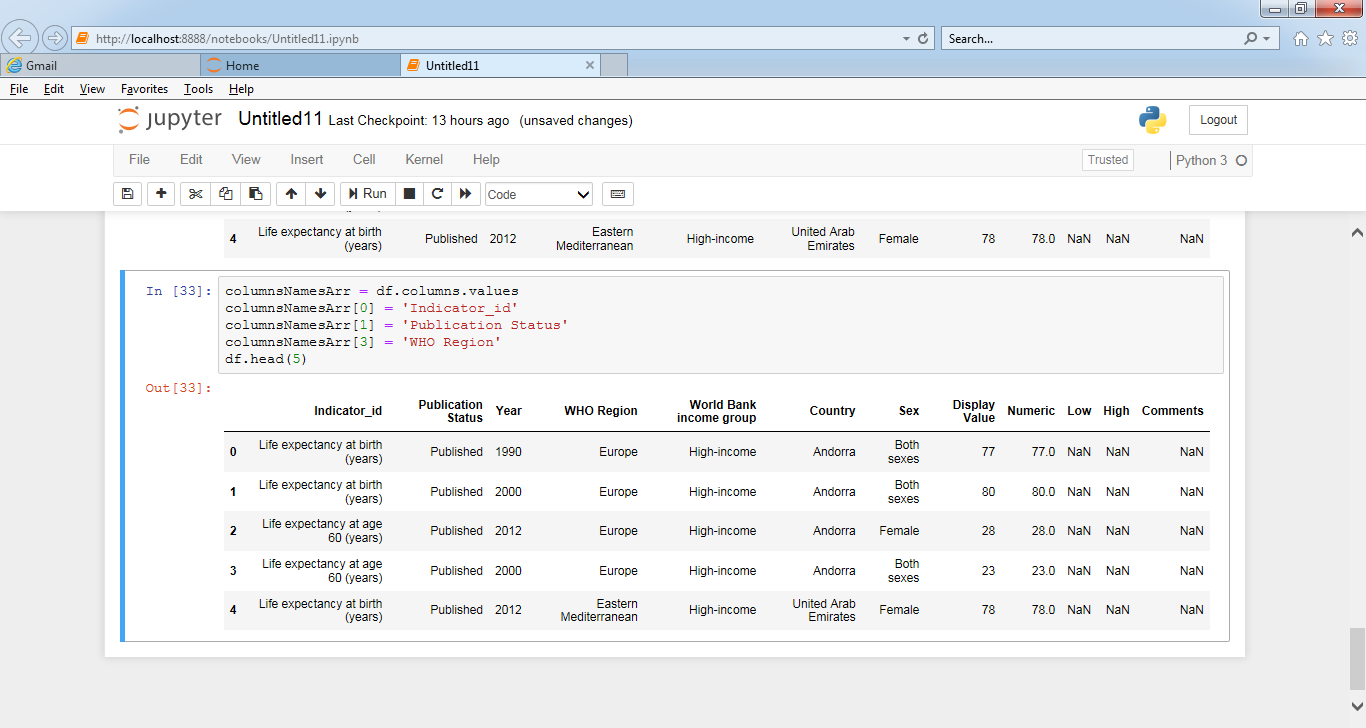
columnsNamesArr = df.columns.values

columnsNamesArr[0] = 'Indicator\_id'

columnsNamesArr[1] = 'Publication Status'

columnsNamesArr[3] = 'WHO Region'

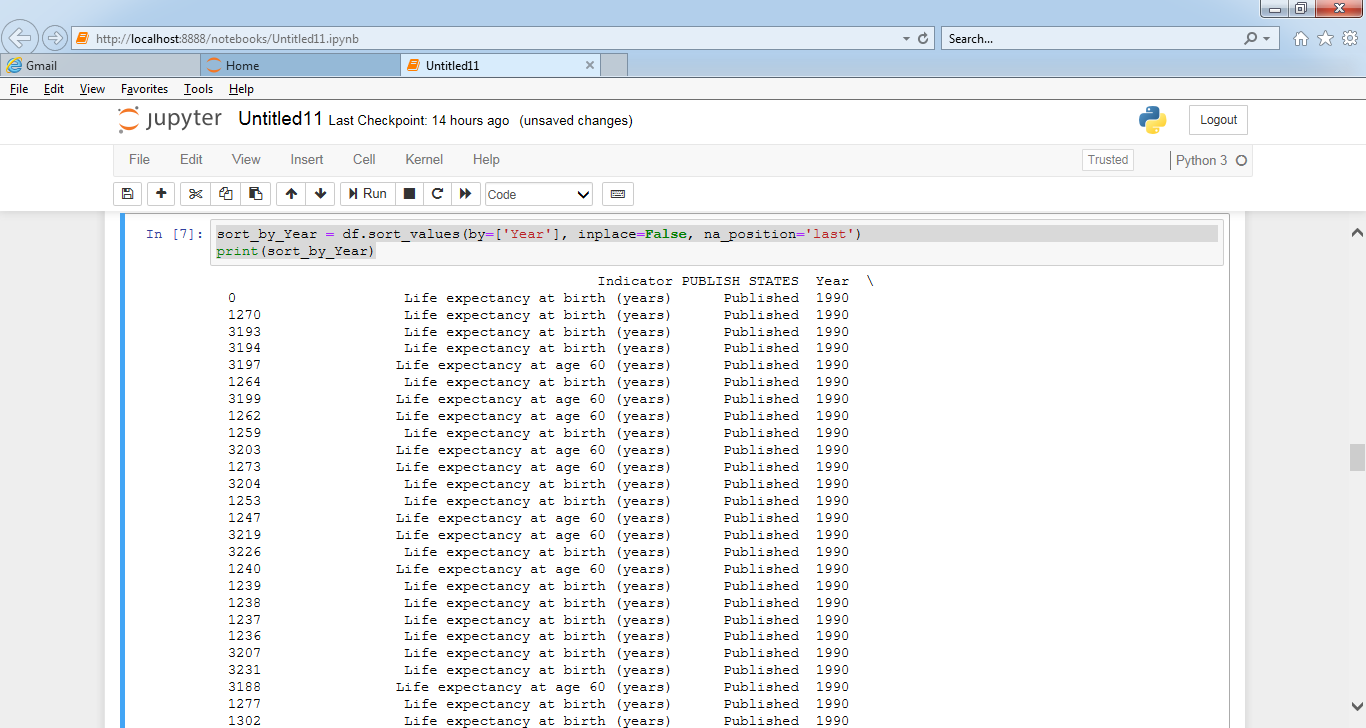
df.head(5)

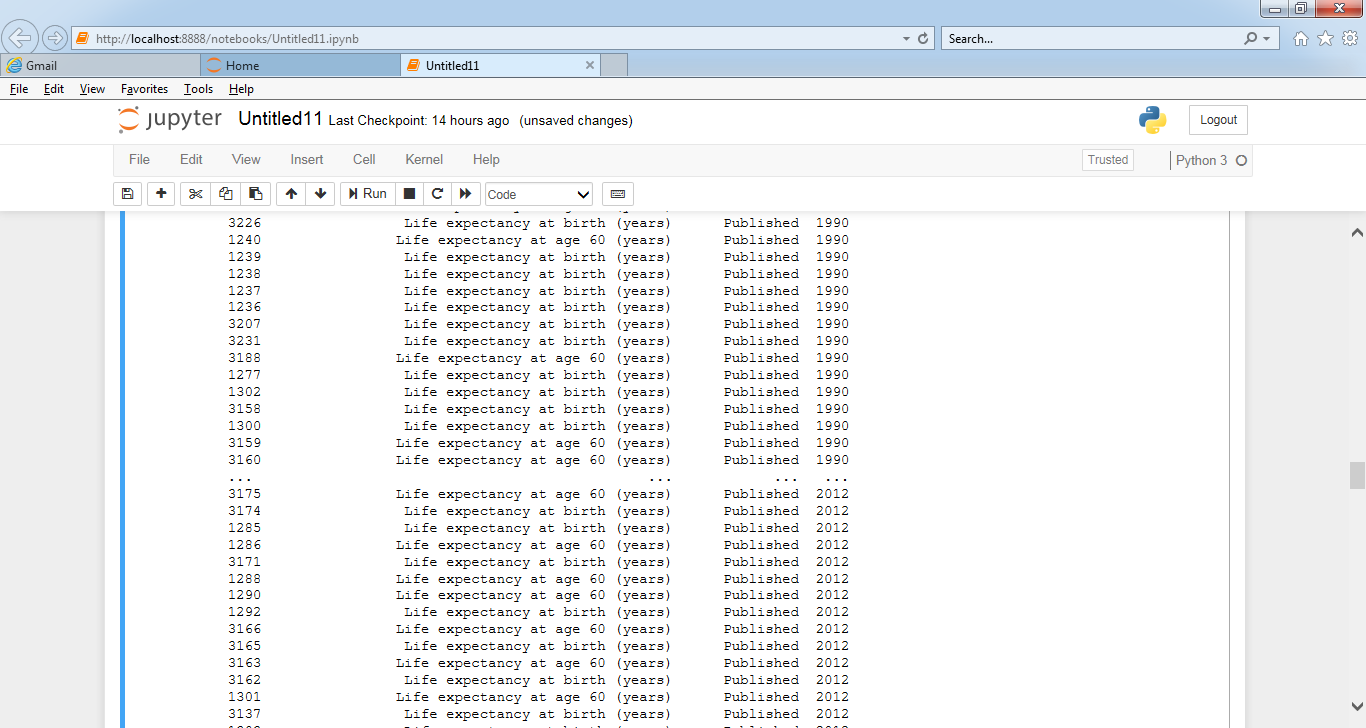


6. Arrange values of a particular column in ascending order.

sort\_by\_Year = df.sort\_values(by=['Year'], inplace=False, na\_position='last')

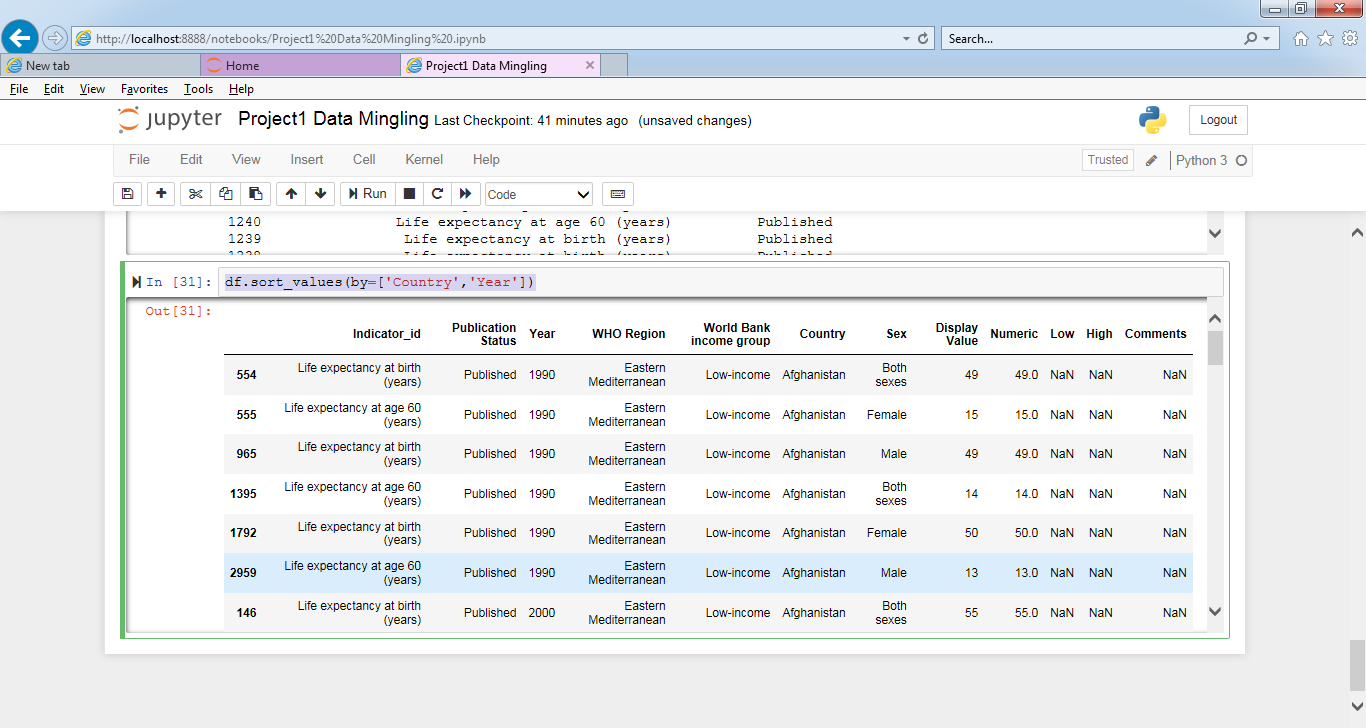
print(sort\_by\_Year)





7. Arrange multiple column values in ascending order.

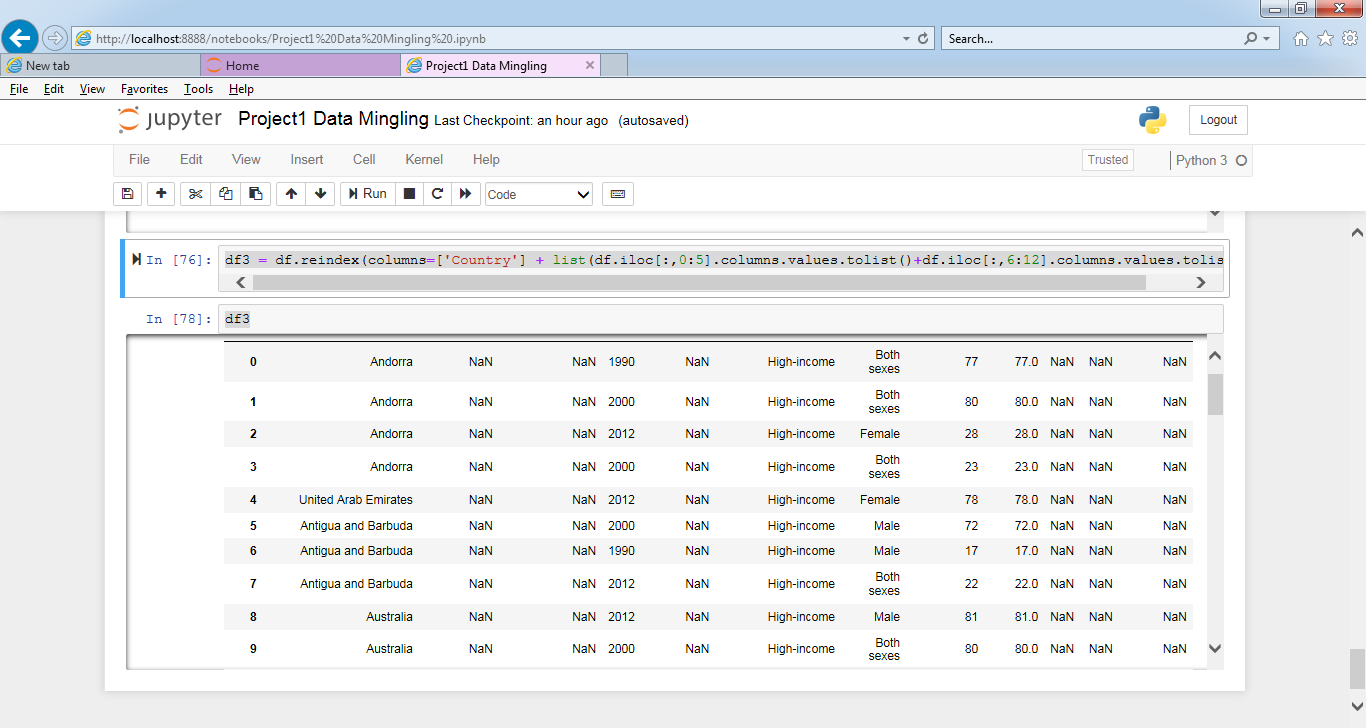
df.sort\_values(by=['Country','Year'])



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

df3 = df.reindex(columns=['Country'] + list(df.iloc[:,0:5].columns.values.tolist()+df.iloc[:,6:12].columns.values.tolist()))

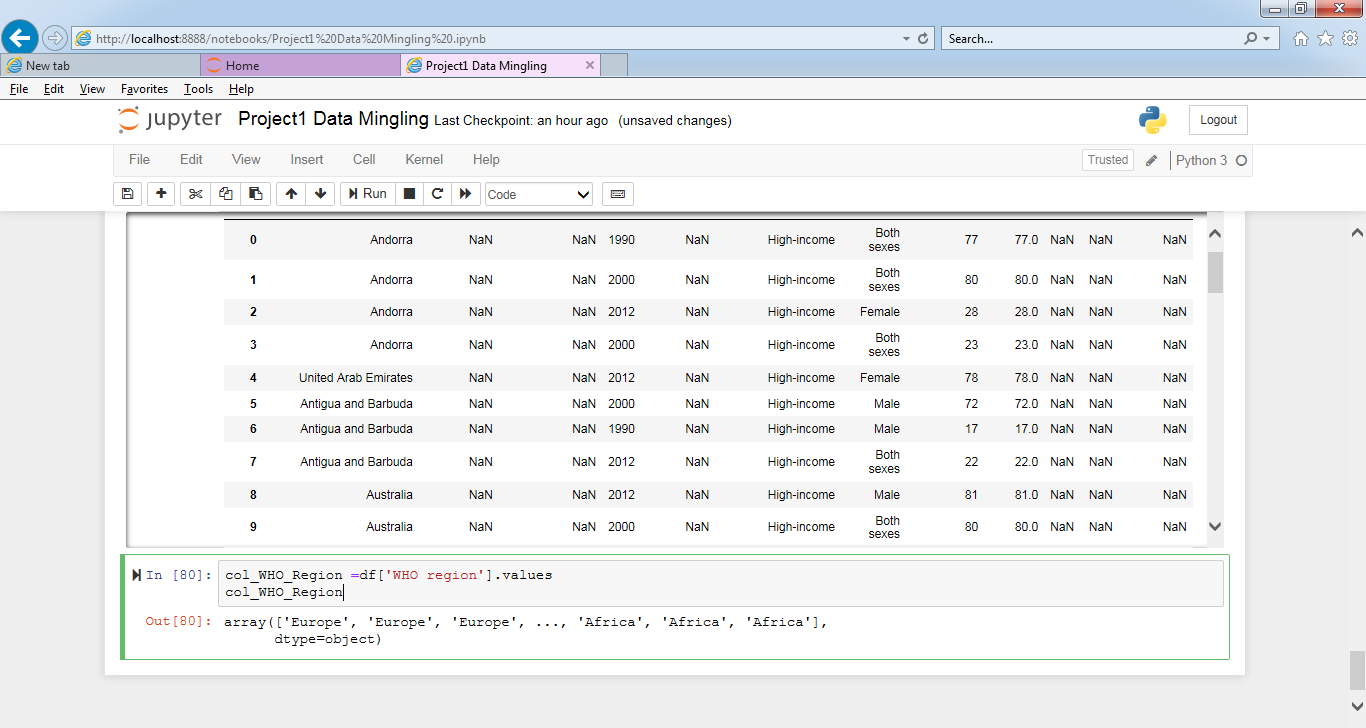
df3



9. Get the column array using a variable **Expected Output:**

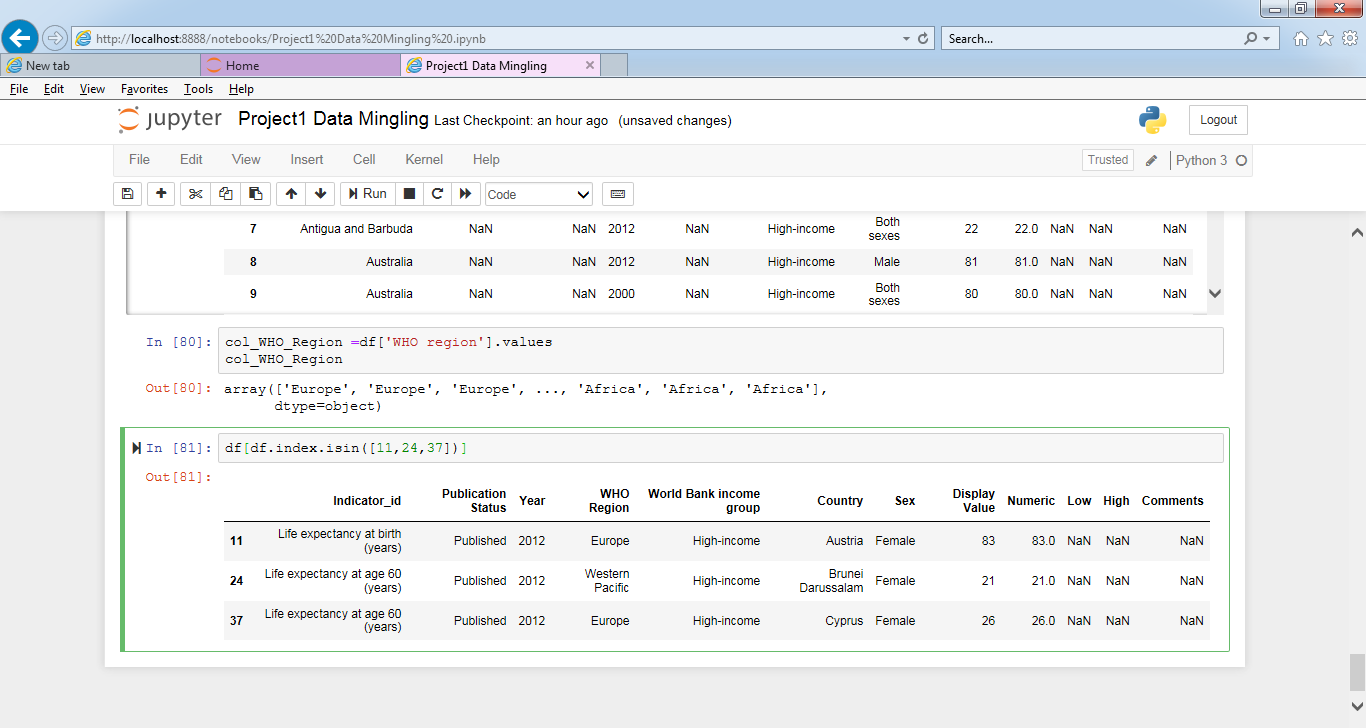
col\_WHO\_Region =df['WHO region'].values

col\_WHO\_Region



10. Get the subset rows 11, 24, 37 **Expected Output:**

df[df.index.isin([11,24,37])]



11.

Get the subset rows excluding 5, 12, 23, and 56 **Expected Output:**

l1 = list(df.index) #first storing all row indexes in l1 list

l1.remove(5) #removing index 5 from l1

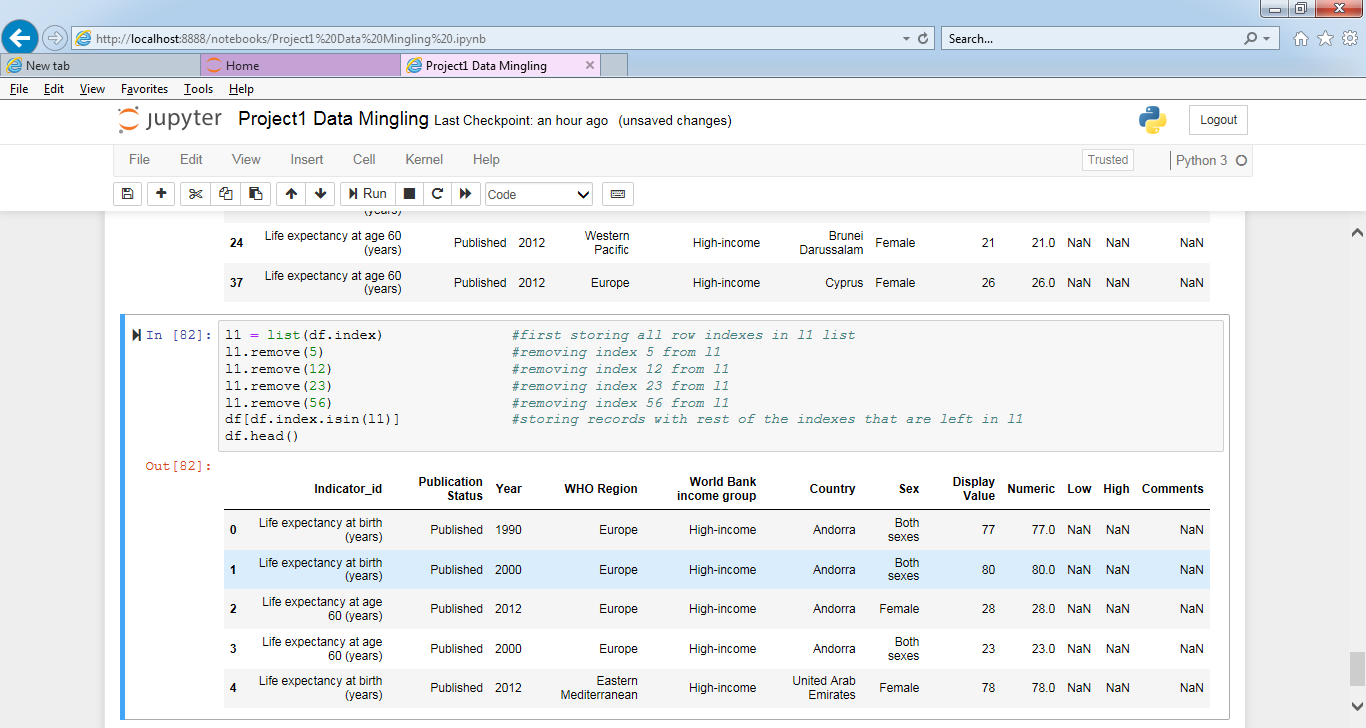
l1.remove(12) #removing index 12 from l1

l1.remove(23) #removing index 23 from l1

l1.remove(56) #removing index 56 from l1

df[df.index.isin(l1)] #storing records with rest of the indexes that are left in l1

df.head()



12.

**Load datasets from CSV**

users = pd.read\_csv('https://raw.githubusercontent.com/ben519/DataWrangling/master/Data / users.csv' )

sessions =

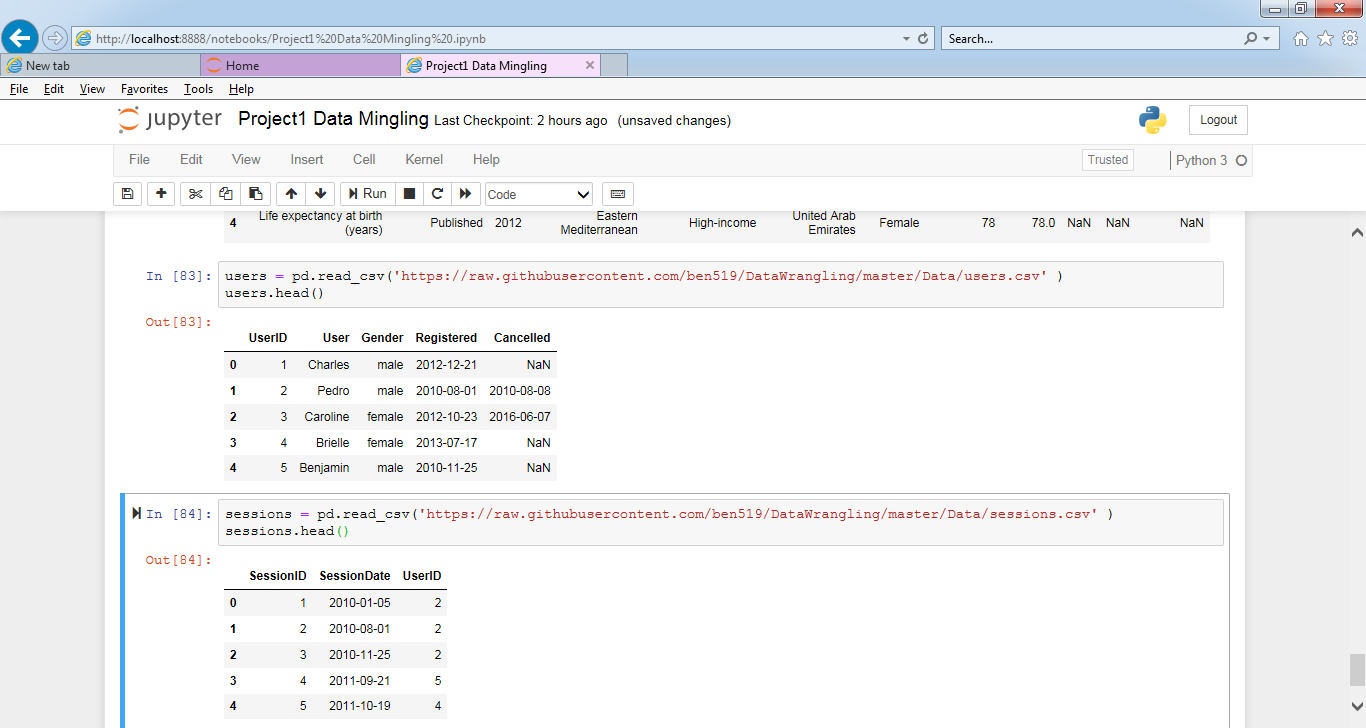
pd.read\_csv('https://raw.githubusercontent.com/ben519/DataWrangling/master/Data / sessions.csv' )

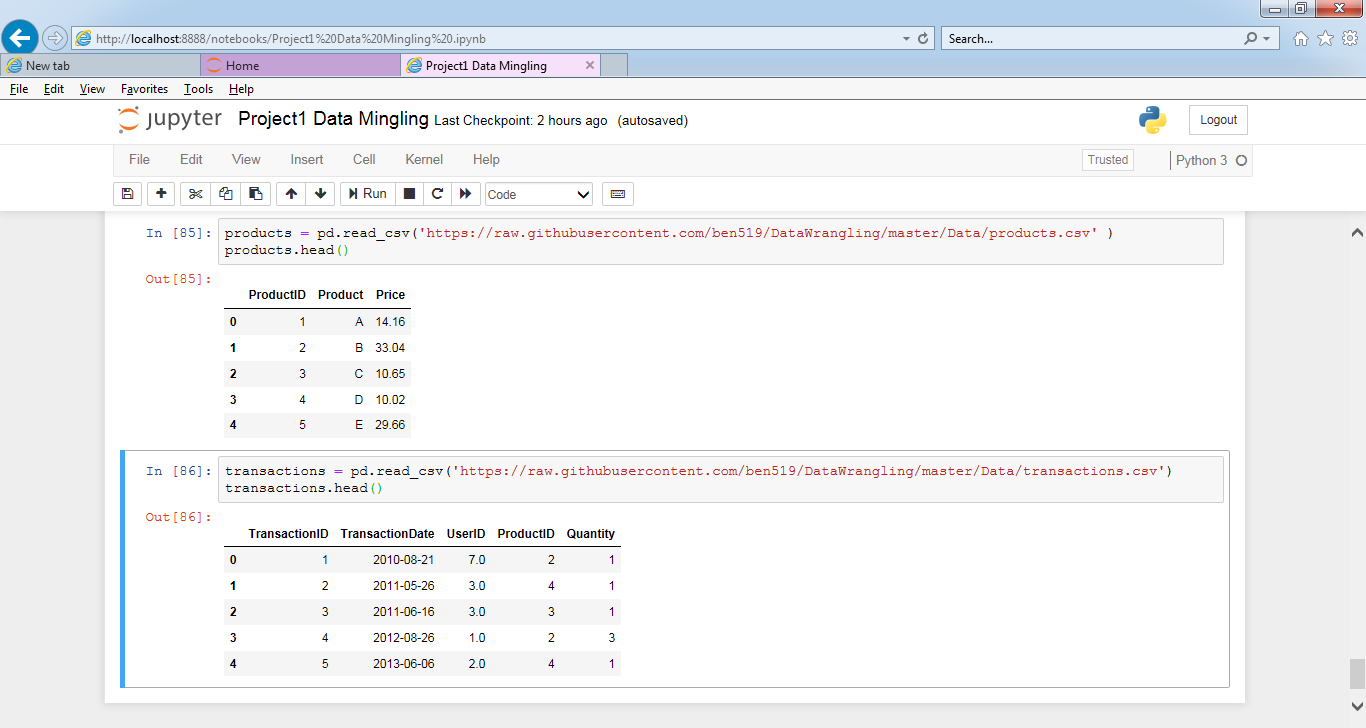
products =

pd.read\_csv('https://raw.githubusercontent.com/ben519/DataWrangling/master/Data / products.csv' )

transactions =

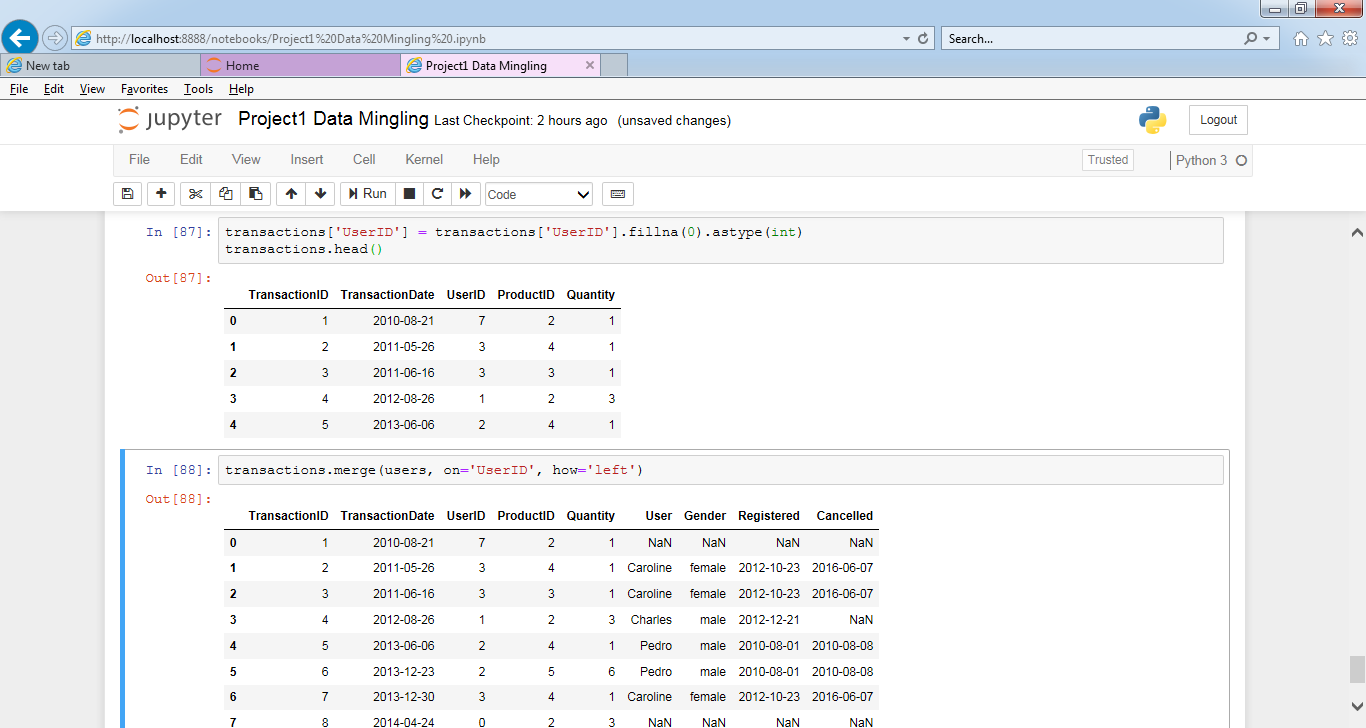
pd.read\_csv('https://raw.githubusercontent.com/ben519/DataWrangling/master/Data / transactions.csv') users.head() sessions.head() transactions.head()





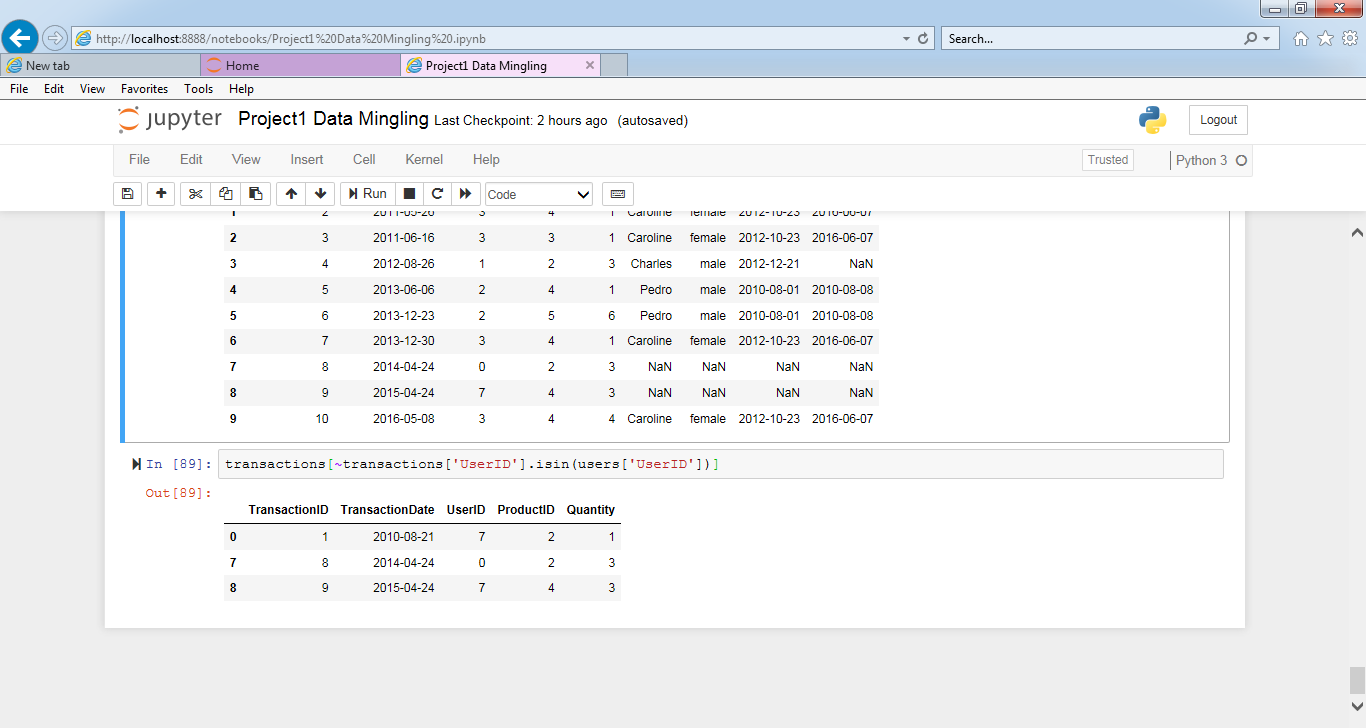
Join users to transactions, keeping all rows from transactions and only matching rows from users (left join)

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



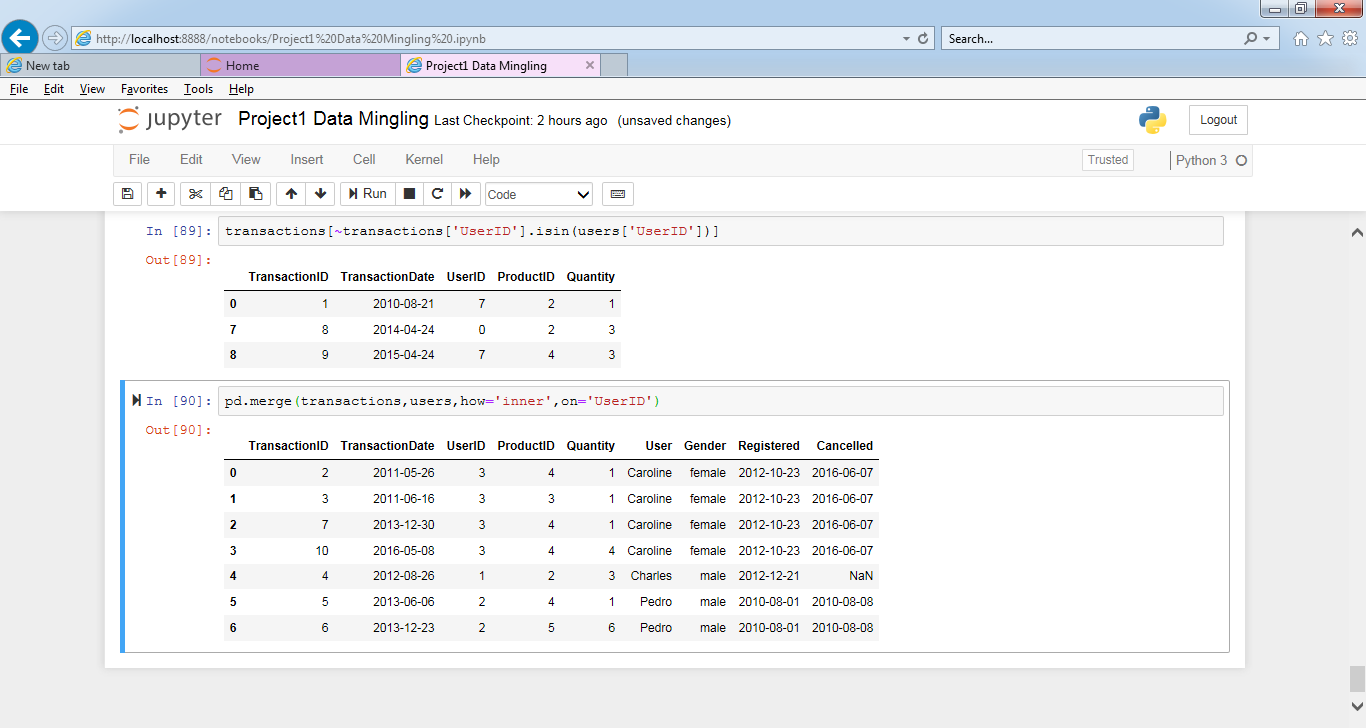
13. Which transactions have a UserID not in users?

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



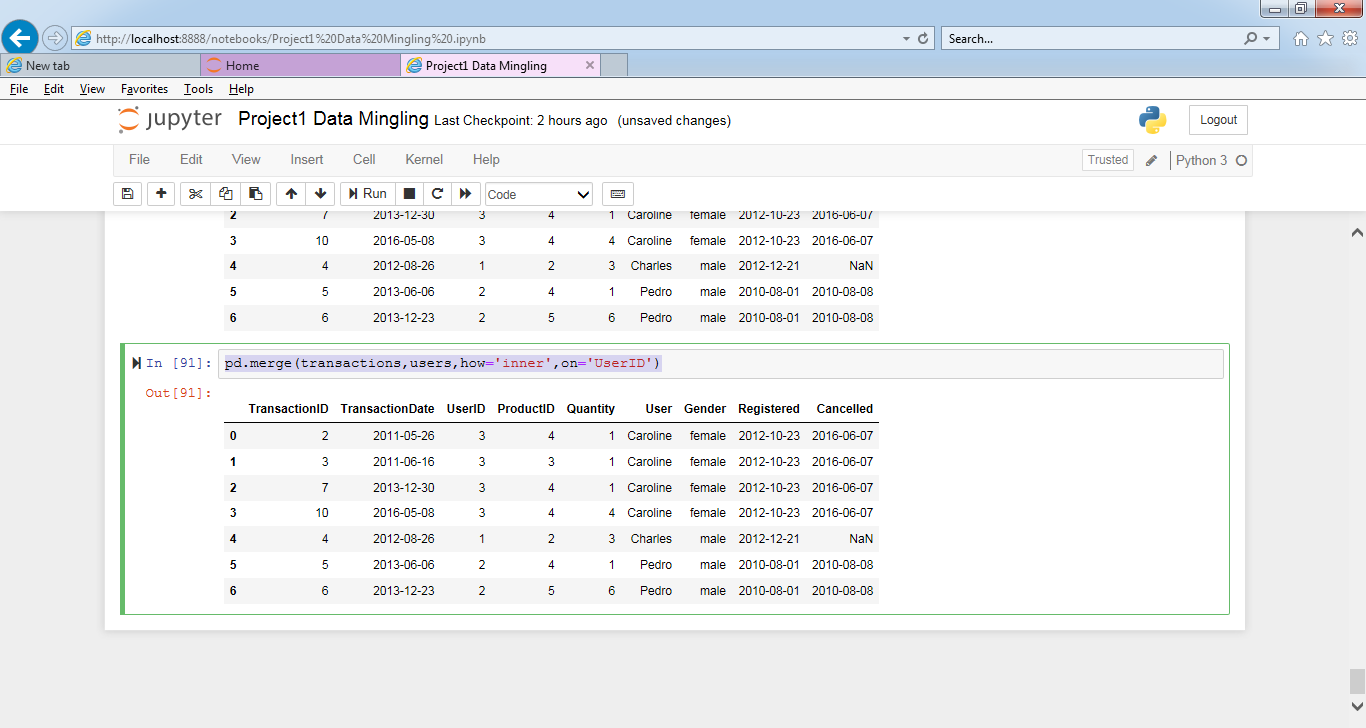
14. Join users to transactions, keeping only rows from transactions and users that match via UserID (inner join) **Expected Output:**

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



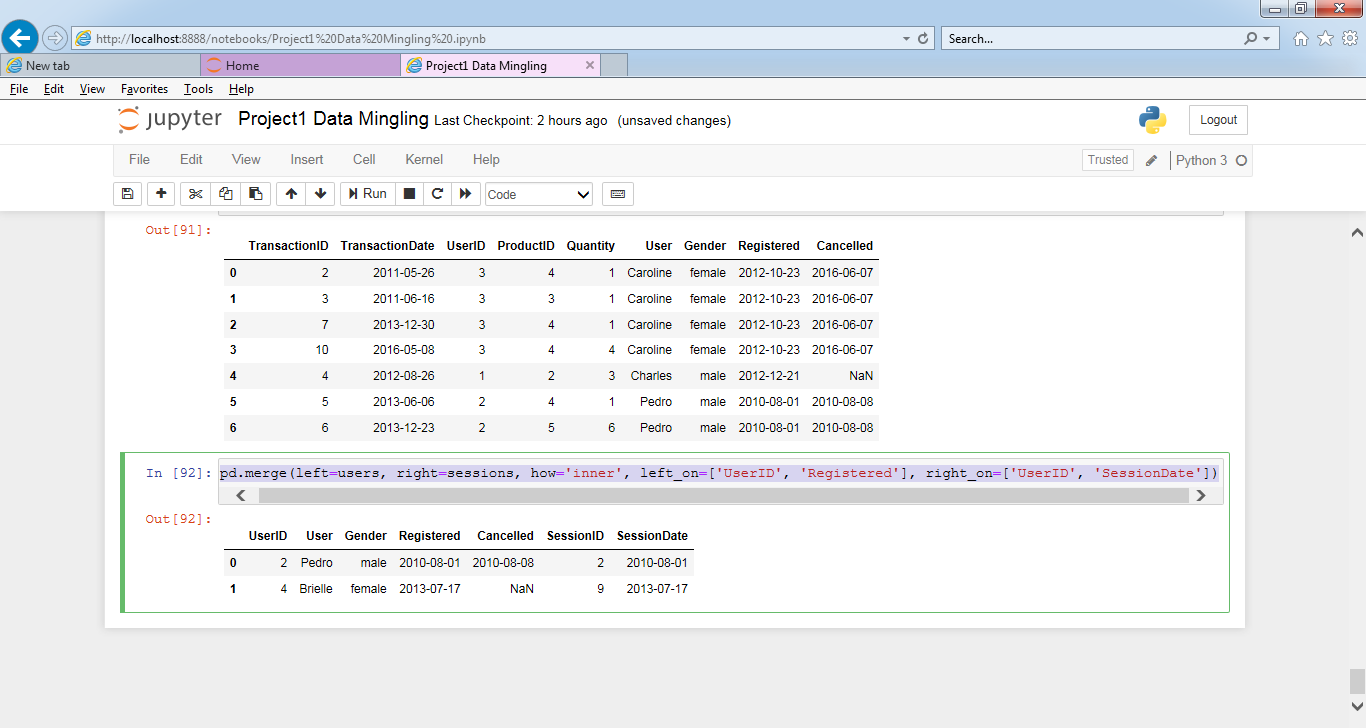
15. Join users to transactions, displaying all matching rows AND all non-matching rows (full outer join)

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



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

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

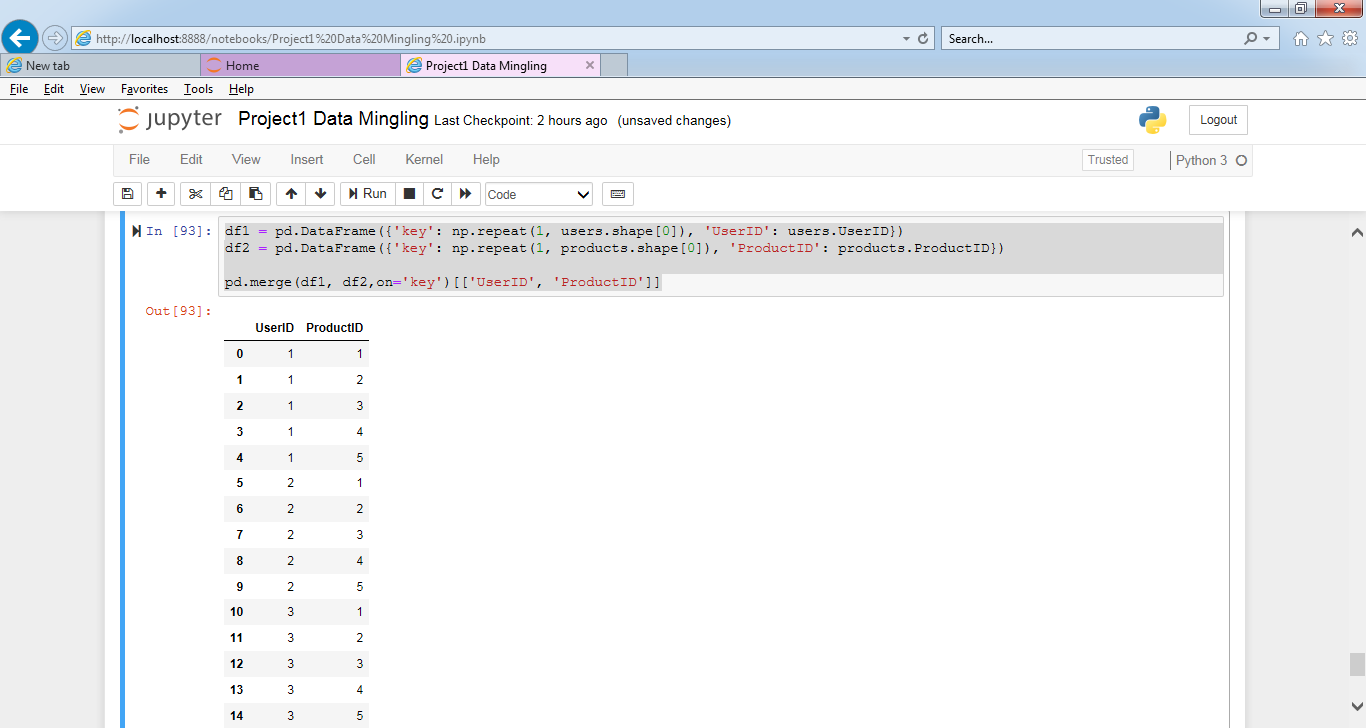


17. Build a dataset with every possible (UserID, ProductID) pair (cross join)

df1 = pd.DataFrame({'key': np.repeat(1, users.shape[0]), 'UserID': users.UserID})

df2 = pd.DataFrame({'key': np.repeat(1, products.shape[0]), 'ProductID': products.ProductID})

pd.merge(df1, df2,on='key')[['UserID', 'ProductID']]



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

df1 = pd.DataFrame({'key': np.repeat(1, users.shape[0]), 'UserID': users.UserID})

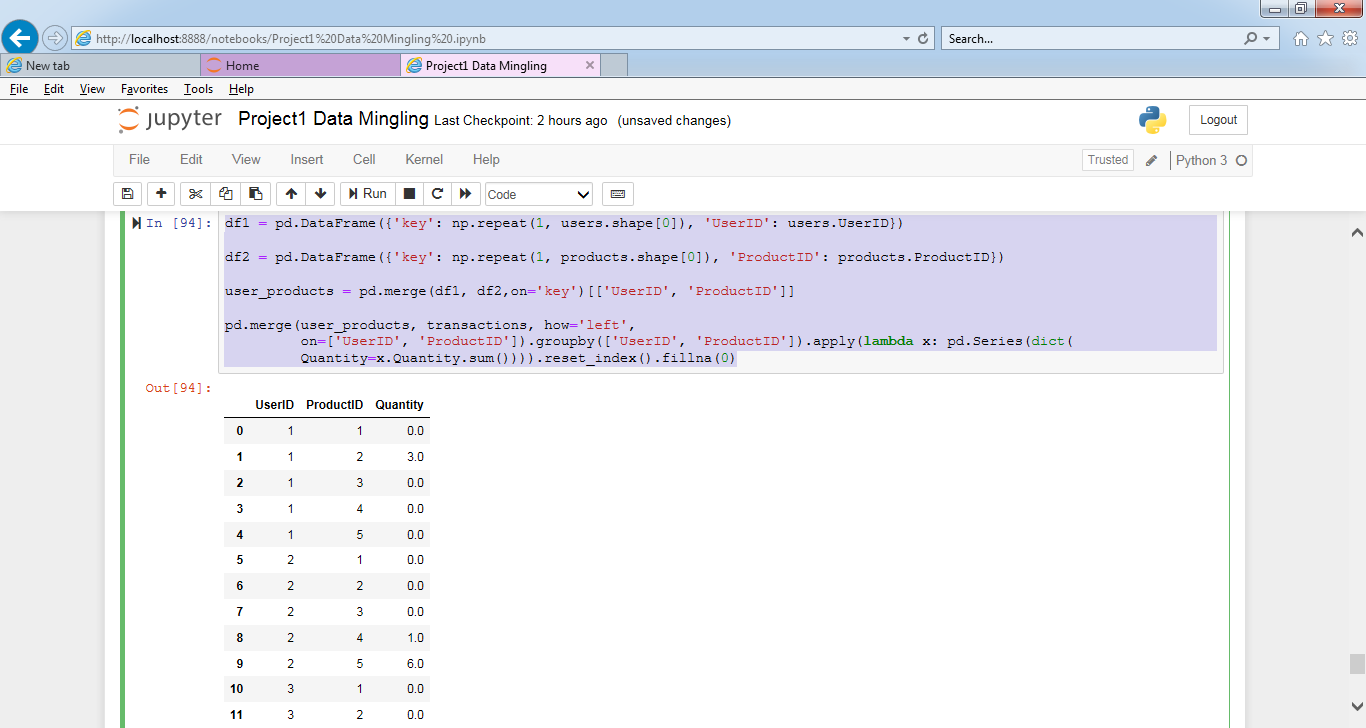
df2 = pd.DataFrame({'key': np.repeat(1, products.shape[0]), 'ProductID': products.ProductID})

user\_products = pd.merge(df1, df2,on='key')[['UserID', 'ProductID']]

pd.merge(user\_products, transactions, how='left',

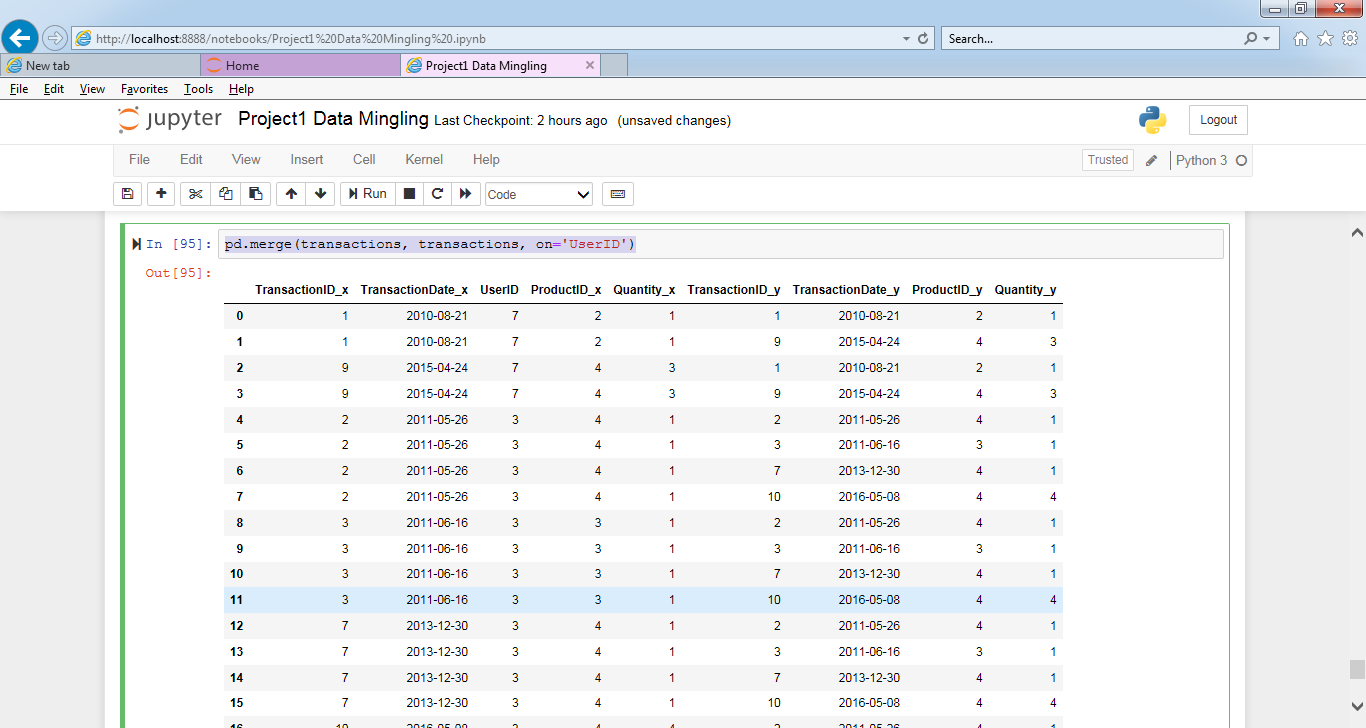
on=['UserID', 'ProductID']).groupby(['UserID', 'ProductID']).apply(lambda x: pd.Series(dict(

Quantity=x.Quantity.sum()))).reset\_index().fillna(0)



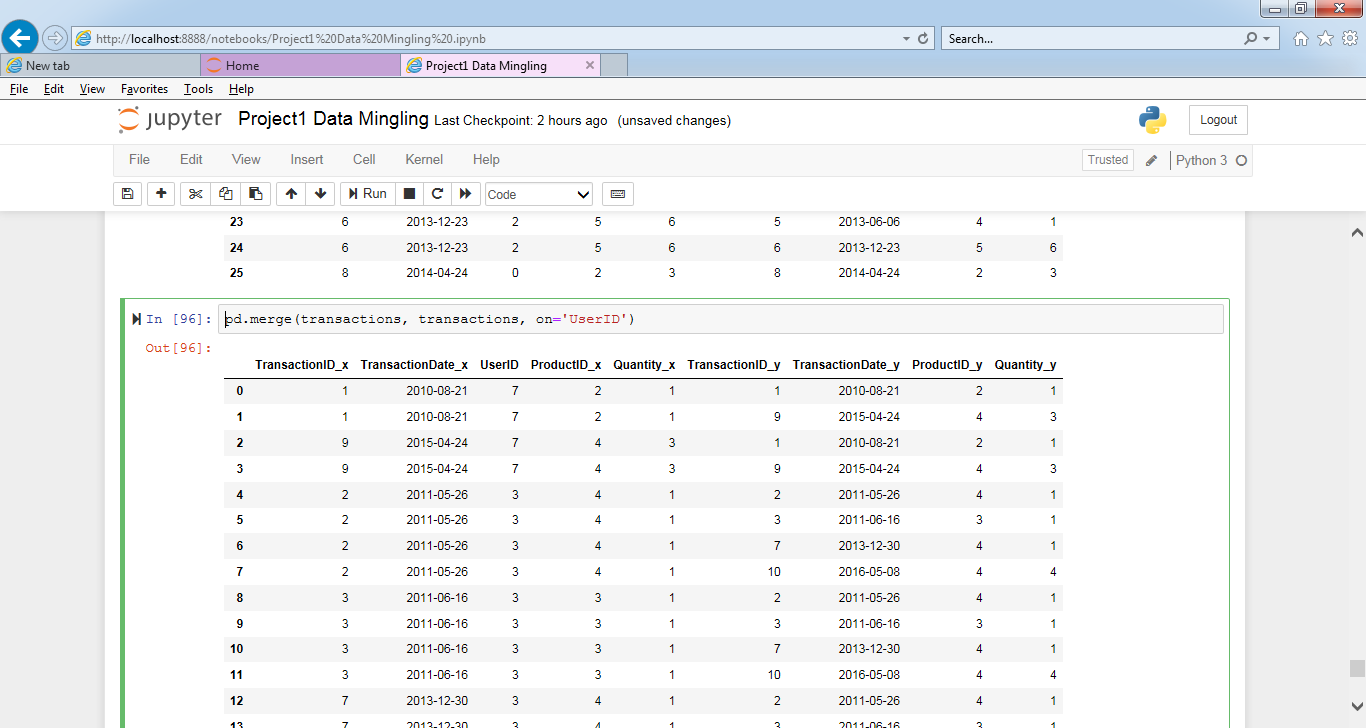
19. For each user, get each possible pair of pair transactions (TransactionID1, TransacationID2)

pd.merge(transactions, transactions, on='UserID')



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

pd.merge(transactions, transactions, on='UserID')



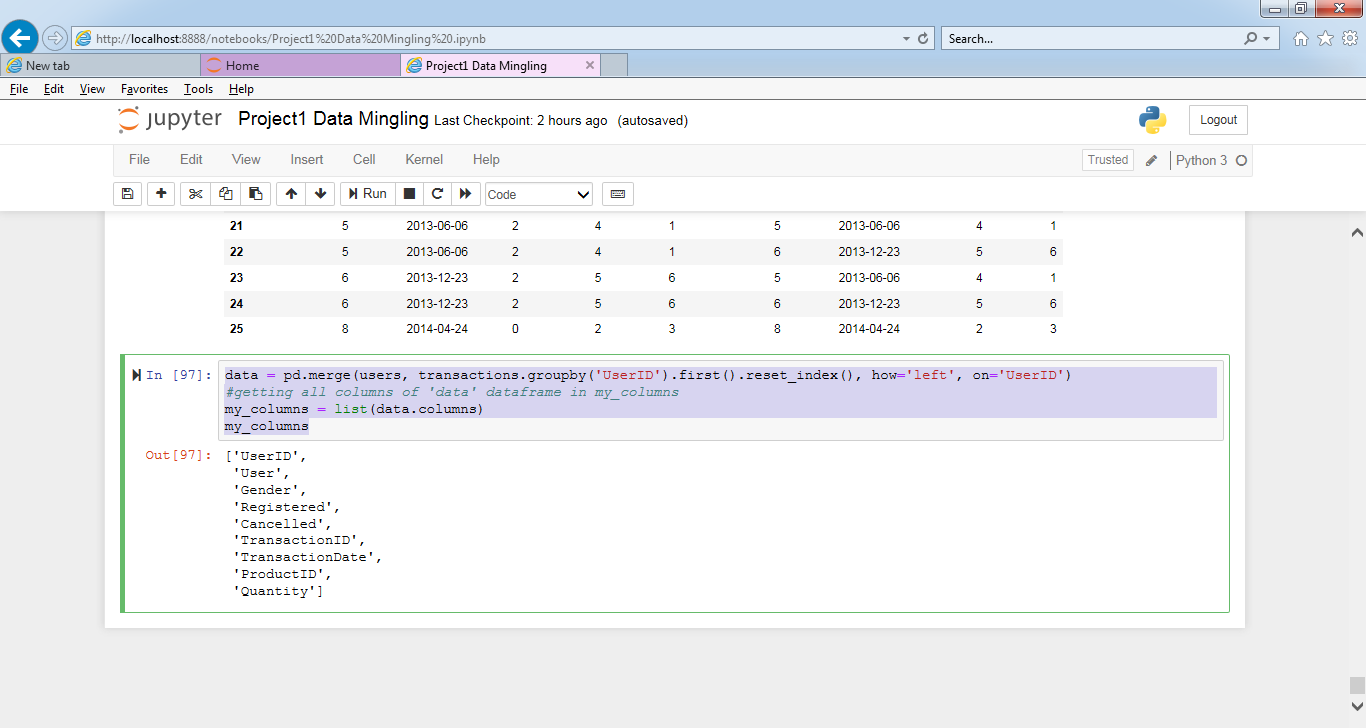
21. Test to see if we can drop columns

data = pd.merge(users, transactions.groupby('UserID').first().reset\_index(), how='left', on='UserID')

#getting all columns of 'data' dataframe in my\_columns

my\_columns = list(data.columns)

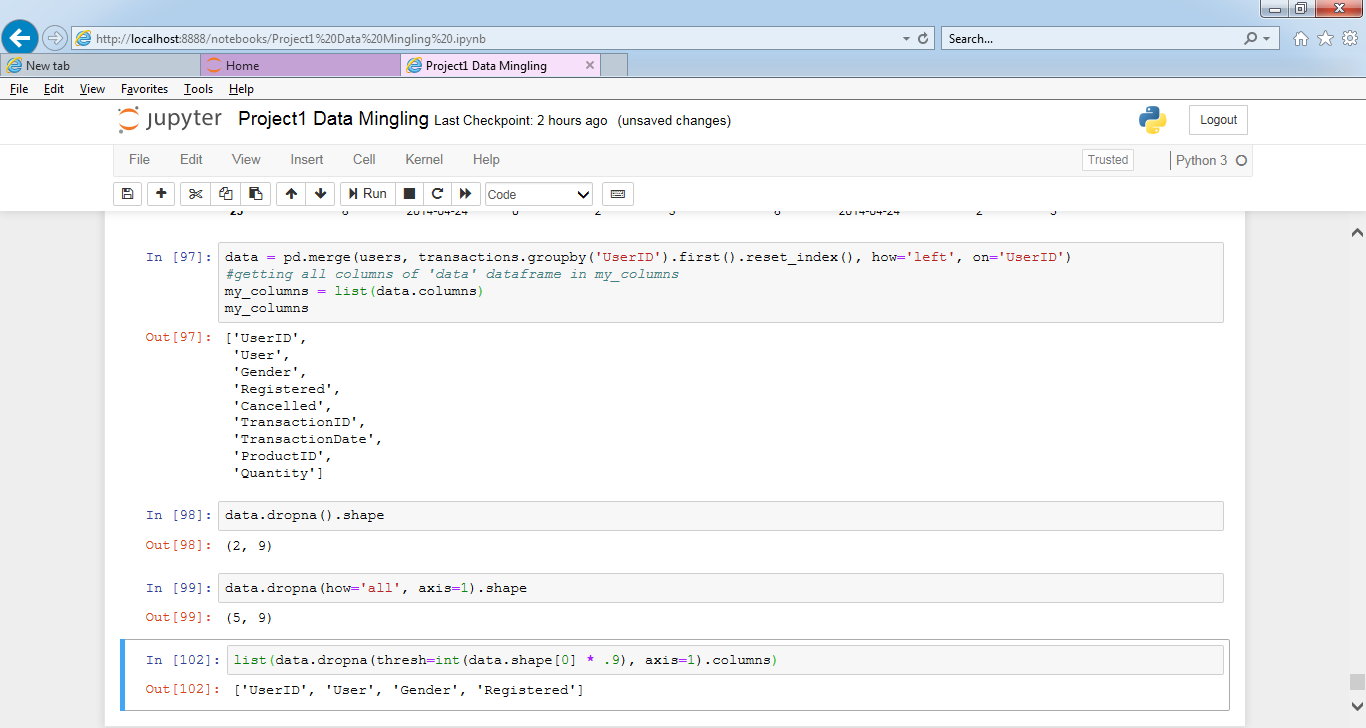
my\_columns



data.dropna().shape

data.dropna(how='all', axis=1).shape

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



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

missing\_info

for col in missing\_info:

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

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

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))

