Python – Pandas Dataframe

|  |
| --- |
| Python Packages for Data Science  *A Python library is a collection of functions and methods that allow you to perform lots of actions without writing any code. The libraries usually contain built in modules providing different functionalities which you can use directly. There are extensive libraries offering a broad range of facilities. Libraries are broadly divided in three groups.*  Scientifics computing libraries:   * + **Pandas:** offers *data structure and tools* for effective data manipulation and analysis. Offers data structure and tools for effective data manipulation and analysis. It provides fast access to structured data. The primary instrument of Pandas is a two-dimensional table consisting of columns and rows labels which are called a DataFrame. It is designed to provide an easy indexing function   + **Numpy:** Uses arrays as their inputs and outputs. It can be extended to objects for matrices, and with a little change of coding, developers perform fast array processing Arrays & Matrices   + **SciPy:** includes functions for some advanced math problems as listed here, as well as data visualization. Integrals, solving differential equations and optimisation |
| Exploring Datasets with pandas  *Pandas is an essential data analysis toolkit for Python.*  *“pandas” is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.*  *The course heavily relies on Pandas for data wrangling, analysis, and visualization. We encourage you to spend some time and familiarize yourself with the Pandas API Reference*: <http://Pandas.pydata.org/pandas-docs/stable/api.html>  # In this lab, we will focus on the Canadian immigration data.  The Dataset: Immigration to Canada from 1980 to 2013  Dataset Source: International migration flows to and from selected countries - The 2015 revision.  The dataset contains annual data on the flows of international immigrants as recorded by the countries of destination. The data presents both inflows and outflows according to the place of birth, citizenship or place of previous / next residence both for foreigners and nationals.  You can fetch the data from (https://ibm.box.com/shared/static/lw190pt9zpy5bd1ptyg2aw15awomz9pu.xlsx).  Pandas Basics  *#* The first thing we'll do is import two key data analysis modules: pandas and numpy.  import numpy as np # useful for many scientific computing in Python  import pandas as pd # primary data structure library  # Let's download and import our primary Canadian Immigration dataset using pandas read\_excel() method. Normally, before we can do that, we would need to download a xlrd module which pandas requires to read in excel files. You would need to run the following line of code to install the xlrd module:  !conda install -c anaconda xlrd –yes  # Now we are ready to read in our data.  df\_can = pd.read\_excel('https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DV0101EN/labs/Data\_Files/Canada.xlsx',  sheet\_name='Canada by Citizenship',  skiprows=range(20),  skipfooter=2)  ​  print ('Data read into a pandas dataframe!')  # Let's view the rows of the dataset using the head() or tail() function.  df\_can.head() # View the Top 5 rows, You can specify the number of rows you'd like to see in brackets  df\_can.tail() # View the bottom 5 rows of the dataset  *When analyzing a dataset, it's always a good idea to start by getting basic information about your dataframe. We can do this by using the info() method along with other listed below.*  df\_can.info() # *getting basic information*  df\_can.columns.values # To get the list of column headers we can call upon the dataframe's .columns parameter.  df\_can.index.values # Similarly, to get the list of indicies we use the dataframe's.index parameter.  print(type(df\_can.columns))  print(type(df\_can.index))  *Note: The default type of index and columns is NOT list.*  # To get the index and columns as lists, we can use the tolist() method.  df\_can.columns.tolist()  df\_can.index.tolist()  ​print (type(df\_can.columns.tolist()))  print (type(df\_can.index.tolist()))  # To view the dimensions of the dataframe, we use the dataframe's.shape parameter.  df\_can.shape # size of dataframe (rows, columns)  *Note: The main types stored in pandas objects are float, int, bool, datetime64[ns] and datetime64[ns, tz] (in >= 0.17.0), timedelta[ns], category (in >= 0.15.0), and object (string). In addition, these dtypes have item sizes, e.g. int64 and int32.*  # Let's clean the data set to remove a few unnecessary columns. We can use pandas drop() method as follows:  # In pandas axis=0 represents rows (default) and axis=1 represents columns.  df\_can.drop(['AREA','REG','DEV','Type','Coverage'], axis=1, inplace=True)  df\_can.head(2)  # Let's rename the columns using rename() method by passing in a dictionary of old and new names as follows:  df\_can.rename(columns={'OdName':'Country', 'AreaName':'Continent', 'RegName':'Region'}, inplace=True)  df\_can.columns  # Let’s add a 'Total' column that sums up the total immigrants by country over the entire period 1980 - 2013:  df\_can['Total'] = df\_can.sum(axis=1)  # We can check to see how many null objects we have in the dataset as follows:  df\_can.isnull().sum()  # Finally, let's view a quick summary of each column in our dataframe using the describe() method.  df\_can.describe()  Pandas Intermediate: *Let’s see the ways for Indexing and Selection (slicing) of dataframe.*  # Select Column, there are two ways to filter on a column name:   * Method 1: Quick and easy, but only works if the column name does NOT have spaces or special characters. df.column\_name # Returns series   # Let's try filtering on the list of countries ('Country')  df\_can.Country   * Method 2: More robust, and can filter on multiple columns.   df['column'] # Returns series  df[['column 1', 'column 2']] # Returns dataframe  # Let's try filtering on the list of countries ('OdName') and the data for years: 1980 - 1985.  df\_can[['Country', 1980, 1981, 1982, 1983, 1984, 1985]] # returns a dataframe  *Note: 'Country' is string, and the years are integers.* *For consistency, we will convert all column names to string later on.*  # Select Row, there are main 3 ways to select rows:   * Method 1: Filters by the labels of the index/column   df.loc[label]   * Method 2: Filters by the positions of the index/column   df.iloc[index]   * Method 3: Filters by the labels/positions of the index along with column list   df.loc[index\_label, column\_label] or df.loc[index\_label, [‘column1’,’column2’]]  df.loc[index\_value, column\_no] or df.loc[index\_value, [column1,column3]]  *Before we proceed, notice that the default index of the dataset is a numeric range from 0 to 194. This makes it difficult to do a query by a specific country. For example, to search for data on Japan, we need to know the corresponding index value.*  # This can be fixed very easily by setting the 'Country' column as the index using set\_index() method.  df\_can.set\_index('Country', inplace=True) # The opposite of set is reset. So, to reset the index use df\_can.reset\_index()  df\_can.head(3)  # In case you need to remove the name of the index  df\_can.index.name = None  # Let's view the number of immigrants from Japan (row 87)for the following scenarios:   * Scenario 1: The full row data (all columns)   print(df\_can.loc['Japan']) # Here 'Japan' is the label of Index that we set earlier  # Alternate methods  print(df\_can.iloc[87]) # Here 87 is the Index value (row 87) i.e. 'Japan'  print(df\_can[df\_can.index == 'Japan'].T.squeeze()) # “.T” method used here is to transpose the result   * Scenario 2: For year 2013   print(df\_can.loc['Japan', 2013])  # Alternate method  print(df\_can.iloc[87, 36]) # year 2013 is the last column, with a positional index of 36   * Scenario 3: For years 1980 to 1985   print(df\_can.loc['Japan', [1980, 1981, 1982, 1983, 1984, 1984]])  print(df\_can.iloc[87, [3, 4, 5, 6, 7, 8]])  *Note: Column names that are integers (such as the years) might introduce some confusion. For example, when we are referencing the year 2013, one might confuse that when the 2013th positional index.*  # To avoid this ambiguity, let's convert the column names into strings: '1980' to '2013'.  df\_can.columns = list(map(str, df\_can.columns))  # [print (type(x)) for x in df\_can.columns.values] #<-- uncomment to check type of column headers  # Since we have the years column label as string, let's declare a variable to easily call upon the full range of years:  years = list(map(str, range(1980, 2014))) # We will Use it for plotting later on  years  # Filtering based on a criterion:  *To filter the dataframe based on a condition, we simply pass the condition as a Boolean vector. For example, let's filter the dataframe to show the data on Asian countries (AreaName = Asia).*   * # Create the condition Boolean series   condition = df\_can['Continent'] == 'Asia'   * # Pass this condition into the dataFrame   df\_can[condition]  *We can also pass multiple criteria in the same line.*  # Let's filter for AreaNAme = Asia and RegName = Southern Asia  ​df\_can[(df\_can['Continent']=='Asia') & (df\_can['Region']=='Southern Asia')]  ​  *Note: When using 'and' and 'or' operators, pandas requires we use '&' and '|' instead of 'and' and 'or', and don't forget to enclose the two conditions in parentheses.*  *Before we proceed, Let's review the changes we have made to our dataframe.*  print('data dimensions:', df\_can.shape)  print(df\_can.columns)  df\_can.head(2) |