Department of Data Science - Data and Visual Analytics Lab

Lab5. Pandas Concatenate, Merge and Join

Objectives ¶

In this lab, you will learn how to

- concatenate two dataframes
- append a dataframe to another existing dataframe
- merge two dataframes
- join two dataframes using various SQL style join operations

We will play the role of a macroeconomic analyst at the Organization for Economic Cooperation and Development (OECD). The question we are trying to answer is simple but interesting: which countries have citizens putting in the longest work hours and how have these trends been changing over time?.

Unfortunately, the OECD has been collecting data for different continents and time periods separately. Our job is to first get all of the data into one place so we can run the necessary analysis.

```
In [1]: # Import necessary modules
```

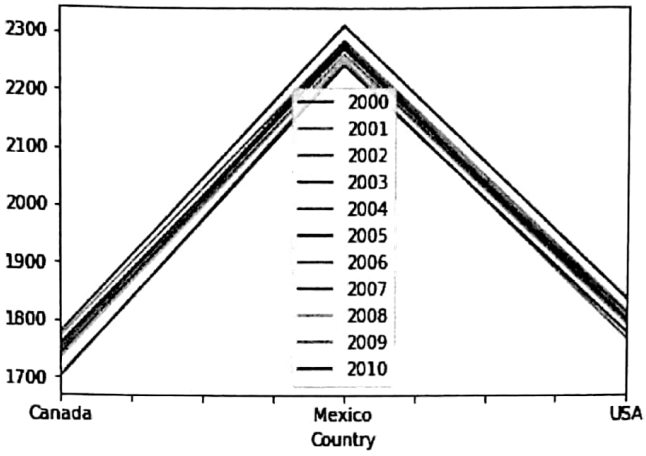
First column should be used as the row index by passing the argument index_col=0

```
In [2]: north_america = pd.read_csv('./oecd/north_america_2000_2010.csv', index_col=0)
        south_america = pd.read_csv('./oecd/south_america_2000_2010.csv', index_col=0)

In [3]: #north_america  #(UNCOMMENT AND SEE OUTPUT)

In [4]: #south_america  #(UNCOMMENT AND SEE OUTPUT)
```

Here, rows are countries, columns are years, and cell values are the average annual hours worked per employee.

Create line graphs for our yearly labor trends in north_america

In [5]: sns.lineplot(data=north_america)

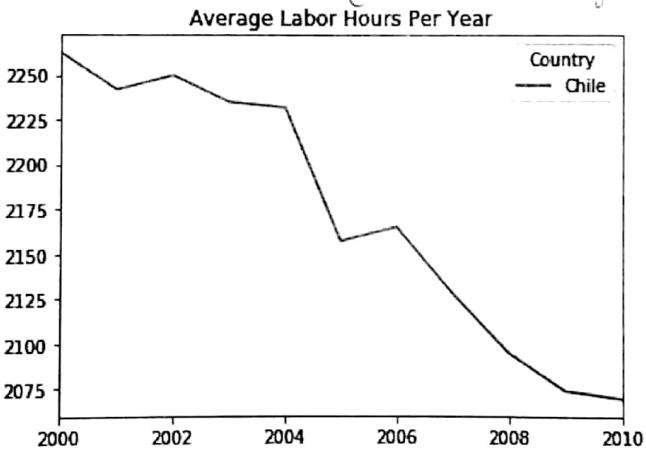


Plot transposed line graph of north_america dataframe, with title "Average Labor Hours Per Year"

In [6]: north_america.T.plot(title="Average Labor Hours Per Year")

Similarly, plot transposed south_america dataframe with title "Average Labor Hours Per Year". Output chart is shown below

In [7]: south_america.T.plot(title="Average Labor Hours Per Year")



Concatenate America Data

It's hard to compare the average labor hours in South America versus North America. If we were able to get all the countries into the same data frame, it would be much easier to do this camparison.

Concatenate north_america and south_america dataframes and store result in a dataframe, americas

In [ ]:
```
americas = pd.concat([north_america, south_america])
```

In [8]:
```
americas
```

Out[8]:

Country	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
Canada	1779.0	1771.0	1754.0	1740.0	1760.0	1747	1745.0	1741.0	1735	1701.0	1703.0
Mexico	2311.2	2285.2	2271.2	2276.5	2270.6	2281	2280.6	2261.4	2258	2250.2	2242.4
USA	1836.0	1814.0	1810.0	1800.0	1802.0	1799	1800.0	1798.0	1792	1767.0	1778.0
Chile	2263.0	2242.0	2250.0	2235.0	2232.0	2157	2165.0	2128.0	2095	2074.0	2069.6

Now, our data collection team has sent us data files for each year from 2011 to 2015 in separate CSV files. They are americas_2011.csv , americas_2012.csv, americas_2014.csv and americas_2015.csv

Load the additional files

In [9]:
```
americas_dfs = [americas]

for year in range(2011, 2016):
    filename = "./oecd/americas_{}.csv".format(year)
    df = pd.read_csv(filename, index_col=0)
    americas_dfs.append(df)
```

In [10]:
```
americas_dfs[1]
```

Out[10]:

Country	2011
Canada	1700.0
Chile	2047.4
Mexico	2250.2
USA	1786.0

In [11]:
```
#americas_dfs[2]
```

One thing you might notice is the rows in the americas_2011 DataFrame we just printed are not in the same sequence as the americas DataFrame (pandas automatically alphabetized them). Luckily, the pd.concat() function joins data on index labels (countries, in our case), not sequence, so this won't pose an issue during concatenation. If we wanted to instead concatenate the rows in the order they are currently in, we could pass the argument ignore_index=True. This would result in the indexes being assigned a sequence of integers. It's also important to keep in mind we have to create the list of DataFrames in the order we would like them concatenated, otherwise our years will be out of chronological order.

We can't use the pd.concat() function exactly the same way we did last time, because now we are adding columns instead of rows. This is where axis comes into play. By default, the argument is set to axis=0, which means we are concatenating rows. This time, we will need to pass in axis=1 to indicate we want to concatenate columns. Remember, this will only work if all the tables have the same height (number of rows).

One caveat to keep in mind when concatenating along axis 1 is the title for the row indexes, 'Country', will be dropped. This is because pandas isn't sure whether that title applies to the new row labels that have been added. We can easily fix this by assigning the DataFrame.index.names attribute.

Concatenate americas and americas_dfs dataframes and store result in americas

In [12]:
```
americas = america.join(temp)
```

```
C:\Users\Rajkumar\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Future
Warning: Sorting because non-concatenation axis is not aligned. A future vers
ion
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  """Entry point for launching an IPython kernel.
```

In [ ]:
```
americas.index.names = ['Country']
```

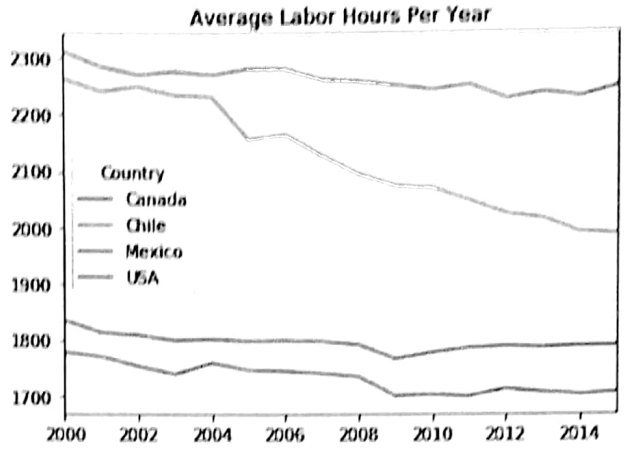
In [13]:
```
americas
```

Out[13]:

Country	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2(
Canada	1779.0	1771.0	1754.0	1740.0	1760.0	1747	1745.0	1741.0	1735	1701.0	1703.0	170
Chile	2263.0	2242.0	2250.0	2235.0	2232.0	2157	2165.0	2128.0	2095	2074.0	2069.6	204
Mexico	2311.2	2285.2	2271.2	2276.5	2270.6	2281	2280.6	2261.4	2258	2250.2	2242.4	225
USA	1836.0	1814.0	1810.0	1800.0	1802.0	1799	1800.0	1798.0	1792	1767.0	1778.0	178

Now, plot transposed americas dataframe

In [14]: americas.T.plot(title = "Average labor hours per value")



Appending data from other Continents

The data collection team has provided CSV files for Asia, Europe, and the South Pacific for 2000 through 2015. Let's load these files in and have a preview

In [15]:
```
asia = pd.read_csv('./oecd/asia_2000_2015.csv', index_col=0)
asia
```

Out[15]:

Country	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
Israel	2017	1979	1993	1974	1942	1931	1919	1931	1929	1927	1918	1920	1910	1867
Japan	1821	1809	1798	1799	1787	1775	1784	1785	1771	1714	1733	1728	1745	1734
Korea	2512	2499	2464	2424	2392	2351	2346	2306	2246	2232	2187	2090	2163	2079
Russia	1982	1980	1982	1993	1993	1989	1998	1999	1997	1974	1976	1979	1982	1980

In [16]:
```
europe = pd.read_csv('./oecd/europe_2000_2015.csv', index_col=0)
europe.head()
```

Out[16]:

Country	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	201
Austria	1807.4	1794.6	1792.2	1783.8	1786.8	1764.0	1746.2	1736.0	1728.5	1673.0	1668.
Belgium	1595.0	1588.0	1583.0	1578.0	1573.0	1565.0	1572.0	1577.0	1570.0	1548.0	1546.
Switzerland	1673.6	1635.0	1614.0	1626.8	1656.5	1651.7	1643.2	1632.7	1623.1	1614.9	1612.
Czech Republic	1896.0	1818.0	1816.0	1806.0	1817.0	1817.0	1799.0	1784.0	1790.0	1779.0	1800.
Germany	1452.0	1441.9	1430.9	1424.8	1422.2	1411.3	1424.7	1424.4	1418.4	1372.7	1389.

In [17]:
```
south_pacific = pd.read_csv('./oecd/south_pacific_2000_2015.csv', index_col=0)
south_pacific
```

Out[17]:

Country	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
Australia	1778.7	1736.7	1731.7	1735.8	1734.5	1729.2	1720.5	1712.5	1717.2	1690	1691.5	1
New Zealand	1836.0	1825.0	1826.0	1823.0	1830.0	1815.0	1795.0	1774.0	1761.0	1740	1755.0	1

If any columns were missing from the data we are trying to append, they would result in those rows having NaN values in the cells falling under the missing year columns. Let's run the append method and verify that all the countries have been sucesfully appended by printing DataFrame.index.

Append asia, europe and south_pacific to americas dataframe and assign to new dataframe world

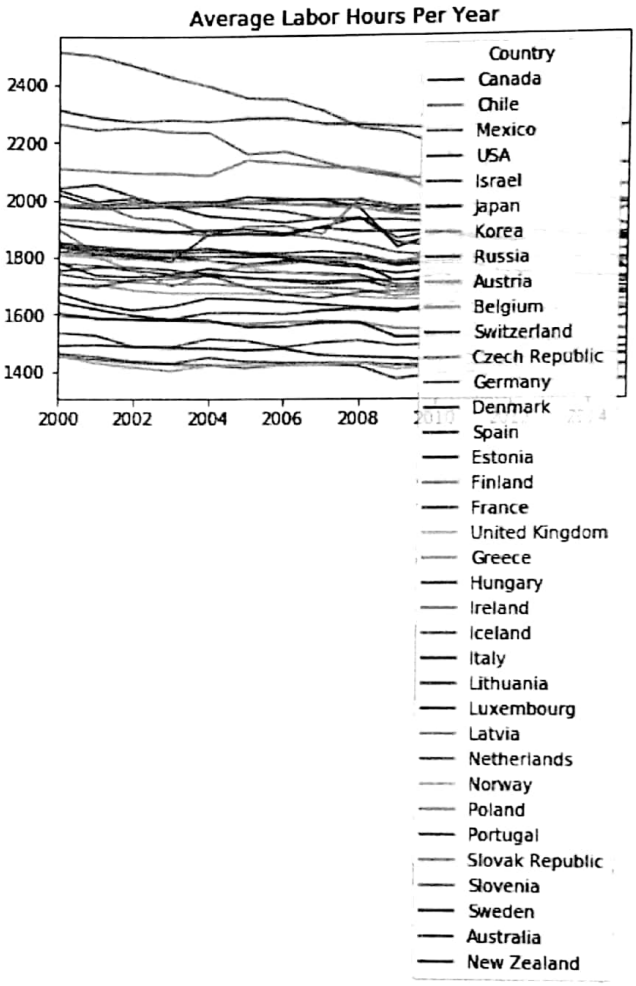
In [18]:
```
world = pd.concat([americas, europe, south_pacific, asia])
```

In [19]:
```
world.index
```

Out[19]:
```
Index(['Canada', 'Chile', 'Mexico', 'USA', 'Israel', 'Japan', 'Korea',
       'Russia', 'Austria', 'Belgium', 'Switzerland', 'Czech Republic',
       'Germany', 'Denmark', 'Spain', 'Estonia', 'Finland', 'France',
       'United Kingdom', 'Greece', 'Hungary', 'Ireland', 'Iceland', 'Italy',
       'Lithuania', 'Luxembourg', 'Latvia', 'Netherlands', 'Norway', 'Polan
d',
       'Portugal', 'Slovak Republic', 'Slovenia', 'Sweden', 'Australia',
       'New Zealand'],
      dtype='object', name='Country')
```

Plot, transposed world dataframe

In [20]: `World.T.plot(title = "Average Labor Hours per year")`

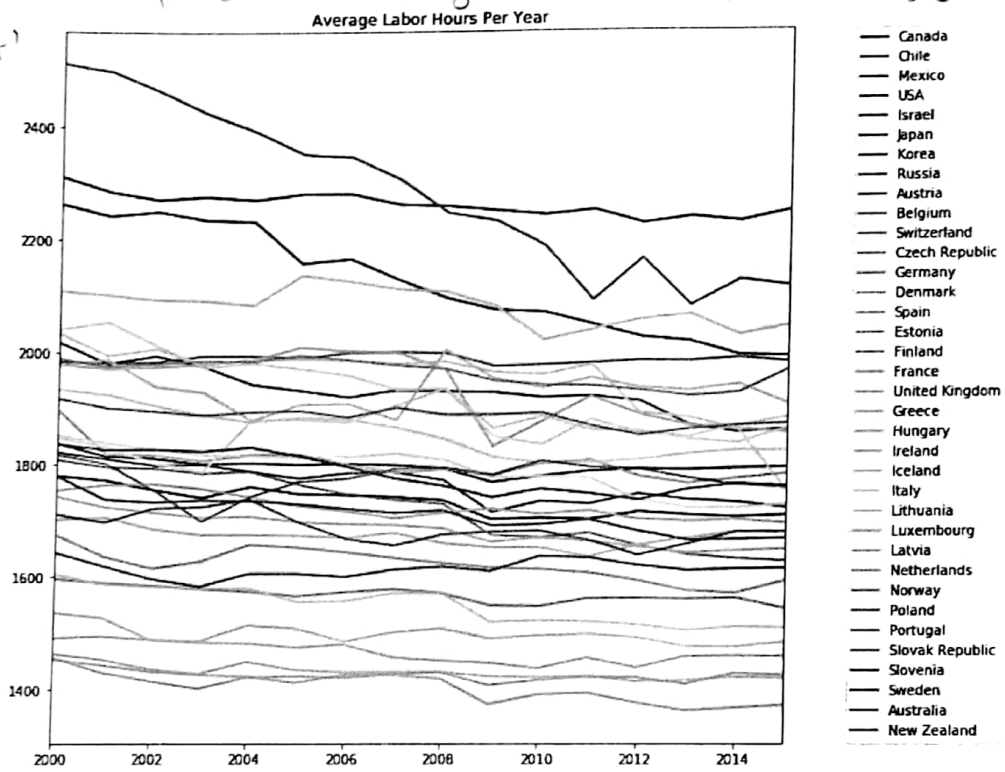


let us customize this plot, so that country names appear outside the chart

Update plot() with the following features

```
figsize=(10,10),
colormap='rainbow',
linewidth=2,
loc='right'
```

In [21]: world.T.plot(title = "Average Labour Hours per Year", figsize=(10,10)
colormap='paint')



Merging Historical Labor Data

It's nice being able to see how the labor hours have shifted since 2000, but in order to see real trends emerge, we want to be able to see as much historical data as possible. The data collection team was kind enough to send data from 1950 to 2000, let's load it in and take a look.

```
In [22]: historical = pd.read_csv('../data/historical.csv', index_col=0)
         historical.head()
```

Out[22]:

Country	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	...	1990	1991
Australia	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1779.5	1774.90
Austria	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
Belgium	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1662.9	1625.79
Canada	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1789.5	1767.50
Switzerland	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	1673.10

5 rows × 50 columns

You'll notice there are a lot of NaN values, especially in the earlier years. This simply means that there was no data collected for those countries in the earlier years. Putting a 0 in those cells would be misleading, as it would imply that no one spent any hours working that year. Instead, NaN represents a null value, meaning "not a number". Having null values will not affect our DataFrame merging since we will use the row labels (index) as our key.

When merging, it's important to keep in mind which rows will be retained from each table. I'm not sure what the full dimensions of my tables are, so instead of displaying the whole thing, we can just look at facts we're interested in. Let's print the DataFrame.shape() attribute to see a tuple containing (total rows, total columns) for both tables.

```
In [23]: print("World rows & columns: ", world.shape)
         print("Historical rows & columns: ", historical.shape)
```

```
World rows & columns:  (36, 16)
Historical rows & columns:  (39, 50)
```

Note that the historical table has 39 rows, even though we are only analyzing 36 countries in our world table. Dropping the three extra rows can be automatically taken care of with some proper DataFrame merging. We will treat world as our primary table and want this to be on the right side of the resulting DataFrame and historical on the left, so the years (columns) stay in chronological order. The columns in these two tables are all distinct, that means we will have to find a key to join on. In this case, the key will be the row indexes (countries).

We will want to do a right join using the pd.merge() function and use the indexes as keys to join on.

The right join will ensure we only keep the 36 rows from the right table and discard the extra 3 from the historical table. Let's print the shape of the resulting DataFrame and display the head to make sure everything turned out correct.

Merge historical dataframe with world dataframe and store in a new variable, world_historical

```
In [25]: world_historical = world.merge(historical, right_on = 'Country', left_on = 'Country
```

Print size of world_historical dataframe

```
In [26]: world_historical.shape
Out[26]: (36, 66)
```

Print top-5 of world_historical dataframe

```
In [27]: world_historical.head()
```

Out[27]:

Country	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	...	200
Canada	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1745
Chile	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2165
Mexico	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2280
USA	1960.0	1975.5	1978.0	1980.0	1970.5	1992.5	1990.0	1962.0	1936.5	1947.0	...	1800
Israel	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1919

5 rows × 66 columns

Joining Historical Data

Now that we've done it the hard way and understand table merging conceptually, let's try a more elegant technique. Pandas has a clean method to join on indexes which is perfect for our situation.

Use join method to join historical dataframe and world dataframe and store result in world_historical dataframe

```
In [28]: world_historical = world.join(historical)
```

In [29]:
```
# Print head of world_historical dataframe
world_historical.head()
```

Out[29]:

Country	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	...	20(
Canada	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1745
Chile	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2165
Mexico	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2280
USA	1960.0	1975.5	1978.0	1980.0	1970.5	1992.5	1990.0	1962.0	1936.5	1947.0	...	1800
Israel	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1919

5 rows × 66 columns

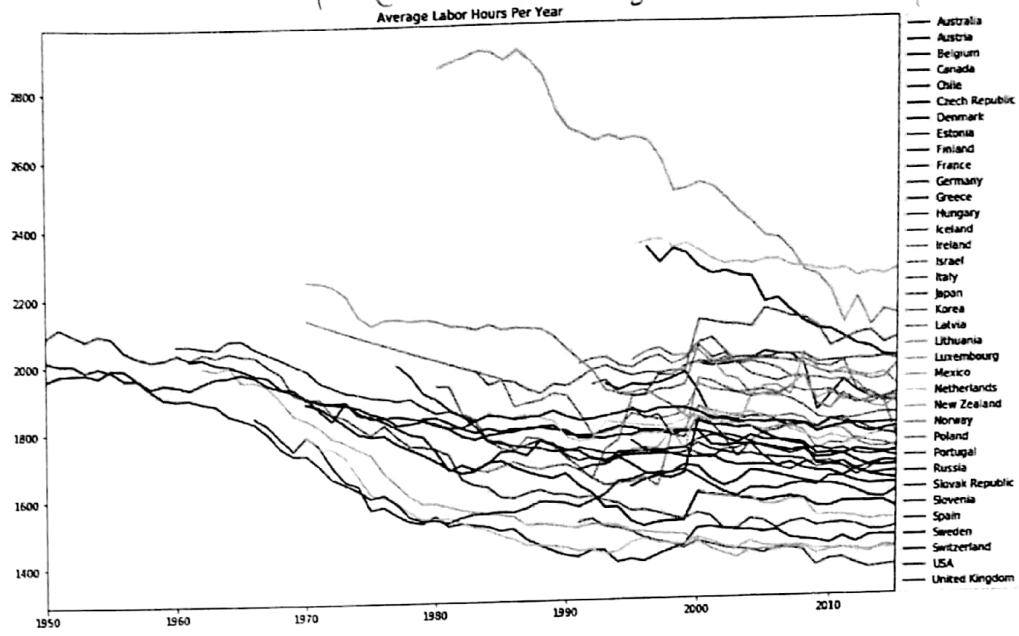
Plot our world labor data

Before plotting the final line graph, it's a good idea to sort our rows alphabetically to make the legend more easy to read for our viewers. This can be executed with the DataFrame.sort_index() method. We can pass in the parameter inplace=True to avoid having to reassign our world_historical variable.

In [30]:
```
world_historical.sort_index(inplace=True)
```

Plot, transposed world_historical dataframe

In [31]: World_historical.T.plot(title="Average Labor Hours Per Year")



Which country worked longer hours per year?

In [ ]: print("Country worked longer hours per year is", Work.[Work == long].Index[0])

Which country worked shorter hours per year?

In [ ]: print("Country worked longer hours per year is", work[work == short].Index[0])