DataFrameIndexingAndLoading_ed

July 21, 2021

In this course, we'll be largely using smaller or moderate-sized datasets. A common workflow is to read the dataset in, usually from some external file, then begin to clean and manipulate the dataset for analysis. In this lecture I'm going to demonstrate how you can load data from a comma separated file into a DataFrame.

[1]:
```
# Lets just jump right in and talk about comma separated values (csv) files.␣
 ↪You've undoubtedly used these -
# any spreadsheet software like excel or google sheets can save output in CSV␣
 ↪format. It's pretty loose as a
# format, and incredibly lightweight. And totally ubiquitous.

# Now, I'm going to make a quick aside because it's convenient here. The␣
 ↪Jupyter notebooks use ipython as the
# kernel underneath, which provides convenient ways to integrate lower level␣
 ↪shell commands, which are
# programs run in the underlying operating system. If you're not familiar with␣
 ↪the shell don't worry too much
# about this, but if you are, this is super handy for integration of your data␣
 ↪science workflows. I want to
# use one shell command here called "cat", for "concatenate", which just␣
 ↪outputs the contents of a file. In
# ipython if we prepend the line with an exclamation mark it will execute the␣
 ↪remainder of the line as a shell
# command.  So lets look at the content of a CSV file
!cat datasets/Admission_Predict.csv
```

[2]:
```
# We see from the output that there is a list of columns, and the column␣
↪identifiers are listed as strings on
# the first line of the file. Then we have rows of data, all columns separated␣
↪by commas. Now, there are lots
# of oddities with the CSV file format, and there is no one agreed upon␣
↪specification. So you should be
# prepared to do a bit of work when you pull down CSV files to explore. But␣
↪this lecture isn't focused on CSV
# files, and is more about pandas DataFrames. So lets jump into that.

# Let's bring in pandas to work with
import pandas as pd

# Pandas mades it easy to turn a CSV into a dataframe, we just call read_csv()
df = pd.read_csv('datasets/Admission_Predict.csv')

# And let's look at the first few rows
df.head()
```

[2]:
```
   Serial No.  GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  \
0           1        337          118                  4  4.5   4.5  9.65
1           2        324          107                  4  4.0   4.5  8.87
2           3        316          104                  3  3.0   3.5  8.00
3           4        322          110                  3  3.5   2.5  8.67
4           5        314          103                  2  2.0   3.0  8.21

   Research  Chance of Admit
0         1             0.92
1         1             0.76
2         1             0.72
3         1             0.80
4         0             0.65
```

[3]:
```
# We notice that by default index starts with 0 while the students' serial␣
↪number starts from 1. If you jump
# back to the CSV output you'll deduce that pandas has create a new index.␣
↪Instead, we can set the serial no.
# as the index if we want to by using the index_col.
df = pd.read_csv('datasets/Admission_Predict.csv', index_col=0)
df.head()
```

[3]:
```
            GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  \
Serial No.
1                 337          118                  4  4.5   4.5  9.65
2                 324          107                  4  4.0   4.5  8.87
3                 316          104                  3  3.0   3.5  8.00
4                 322          110                  3  3.5   2.5  8.67
5                 314          103                  2  2.0   3.0  8.21

            Research  Chance of Admit
Serial No.
1                  1             0.92
2                  1             0.76
3                  1             0.72
4                  1             0.80
5                  0             0.65
```

[4]:
```
# Notice that we have two columns "SOP" and "LOR" and probably not everyone␣
 ↪knows what they mean So let's
# change our column names to make it more clear. In Pandas, we can use the␣
 ↪rename() function It takes a
# parameter called columns, and we need to pass into a dictionary which the␣
 ↪keys are the old column name and
# the value is the corresponding new column name
new_df=df.rename(columns={'GRE Score':'GRE Score', 'TOEFL Score':'TOEFL Score',
                   'University Rating':'University Rating',
                   'SOP': 'Statement of Purpose','LOR': 'Letter of␣
 ↪Recommendation',
                   'CGPA':'CGPA', 'Research':'Research',
                   'Chance of Admit':'Chance of Admit'})
new_df.head()
```

[4]:
```
            GRE Score  TOEFL Score  University Rating  Statement of Purpose  \
Serial No.
1                 337          118                  4                   4.5
2                 324          107                  4                   4.0
3                 316          104                  3                   3.0
4                 322          110                  3                   3.5
5                 314          103                  2                   2.0

            LOR   CGPA  Research  Chance of Admit
Serial No.
1            4.5  9.65         1             0.92
2            4.5  8.87         1             0.76
3            3.5  8.00         1             0.72
4            2.5  8.67         1             0.80
5            3.0  8.21         0             0.65
```

[5]:
```
# From the output, we can see that only "SOP" is changed but not "LOR" Why is␣
↪that? Let's investigate this a
# bit. First we need to make sure we got all the column names correct We can␣
↪use the columns attribute of
# dataframe to get a list.
new_df.columns
```

[5]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'Statement of Purpose',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

[6]:
```
# If we look at the output closely, we can see that there is actually a space␣
↪right after "LOR" and a space
# right after "Chance of Admit. Sneaky, huh? So this is why our rename␣
↪dictionary does not work for LOR,
# because the key we used was just three characters, instead of "LOR "

# There are a couple of ways we could address this. One way would be to change␣
↪a column by including the space
# in the name
new_df=new_df.rename(columns={'LOR ': 'Letter of Recommendation'})
new_df.head()
```

[6]:
```
            GRE Score  TOEFL Score  University Rating  Statement of Purpose  \
Serial No.
1                 337          118                  4                   4.5
2                 324          107                  4                   4.0
3                 316          104                  3                   3.0
4                 322          110                  3                   3.5
5                 314          103                  2                   2.0

            Letter of Recommendation  CGPA  Research  Chance of Admit
Serial No.
1                                4.5  9.65         1             0.92
2                                4.5  8.87         1             0.76
3                                3.5  8.00         1             0.72
4                                2.5  8.67         1             0.80
5                                3.0  8.21         0             0.65
```

[7]:
```
# So that works well, but it's a bit fragile. What if that was a tab instead of␣
↪a space? Or two spaces?
# Another way is to create some function that does the cleaning and then tell␣
↪renamed to apply that function
# across all of the data. Python comes with a handy string function to strip␣
↪white space called "strip()".
# When we pass this in to rename we pass the function as the mapper parameter,␣
↪and then indicate whether the
# axis should be columns or index (row labels)
```

```
new_df=new_df.rename(mapper=str.strip, axis='columns')
# Let's take a look at results
new_df.head()
```

[7]:
```
            GRE Score  TOEFL Score  University Rating  Statement of Purpose  \
Serial No.
1                 337          118                  4                   4.5
2                 324          107                  4                   4.0
3                 316          104                  3                   3.0
4                 322          110                  3                   3.5
5                 314          103                  2                   2.0

            Letter of Recommendation  CGPA  Research  Chance of Admit
Serial No.
1                                4.5  9.65         1             0.92
2                                4.5  8.87         1             0.76
3                                3.5  8.00         1             0.72
4                                2.5  8.67         1             0.80
5                                3.0  8.21         0             0.65
```

[8]:
```
# Now we've got it - both SOP and LOR have been renamed and Chance of Admit has␣
 ↪been trimmed up. Remember
# though that the rename function isn't modifying the dataframe. In this case,␣
 ↪df is the same as it always
# was, there's just a copy in new_df with the changed names.
df.columns
```

[8]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

[9]:
```
# We can also use the df.columns attribute by assigning to it a list of column␣
 ↪names which will directly
# rename the columns. This will directly modify the original dataframe and is␣
 ↪very efficient especially when
# you have a lot of columns and you only want to change a few. This technique␣
 ↪is also not affected by subtle
# errors in the column names, a problem that we just encountered. With a list,␣
 ↪you can use the list index to
# change a certain value or use list comprehension to change all of the values

# As an example, lets change all of the column names to lower case. First we␣
 ↪need to get our list
cols = list(df.columns)
# Then a little list comprehenshion
cols = [x.lower().strip() for x in cols]
# Then we just overwrite what is already in the .columns attribute
df.columns=cols
```

```
# And take a look at our results
df.head()
```

[9]:
```
            gre score  toefl score  university rating  sop  lor  cgpa  \
Serial No.
1                 337          118                  4  4.5  4.5  9.65
2                 324          107                  4  4.0  4.5  8.87
3                 316          104                  3  3.0  3.5  8.00
4                 322          110                  3  3.5  2.5  8.67
5                 314          103                  2  2.0  3.0  8.21

            research  chance of admit
Serial No.
1                  1             0.92
2                  1             0.76
3                  1             0.72
4                  1             0.80
5                  0             0.65
```

In this lecture, you've learned how to import a CSV file into a pandas DataFrame object, and how to do some basic data cleaning to the column names. The CSV file import mechanisms in pandas have lots of different options, and you really need to learn these in order to be proficient at data manipulation. Once you have set up the format and shape of a DataFrame, you have a solid start to further actions such as conducting data analysis and modeling.

Now, there are other data sources you can load directly into dataframes as well, including HTML web pages, databases, and other file formats. But the CSV is by far the most common data format you'll run into, and an important one to know how to manipulate in pandas.