# ExampleManipulatingDataFrames

May 12, 2022

In this lecture I'm going to walk through a basic data cleaning process with you and introduce you to a few more pandas API functions.

[12]:
```
# Let's start by bringing in pandas
import pandas as pd
# And load our dataset. We're going to be cleaning the list of presidents in␣
 ↪the US from wikipedia
df=pd.read_csv("datasets/presidents.csv")
# And lets just take a look at some of the data
df.head()
```

[12]:
```
   #          President             Born      Age atstart of presidency  \
0  1  George Washington  Feb 22, 1732[a]  57ăyears, 67ădaysApr 30, 1789
1  2         John Adams  Oct 30, 1735[a]  61ăyears, 125ădaysMar 4, 1797
2  3   Thomas Jefferson  Apr 13, 1743[a]  57ăyears, 325ădaysMar 4, 1801
3  4      James Madison  Mar 16, 1751[a]  57ăyears, 353ădaysMar 4, 1809
4  5       James Monroe     Apr 28, 1758  58ăyears, 310ădaysMar 4, 1817

        Age atend of presidency Post-presidencytimespan          Died  \
0   65ăyears, 10ădaysMar 4, 1797       2ăyears, 285ădays  Dec 14, 1799
1  65ăyears, 125ădaysMar 4, 1801      25ăyears, 122ădays   Jul 4, 1826
2  65ăyears, 325ădaysMar 4, 1809      17ăyears, 122ădays   Jul 4, 1826
3  65ăyears, 353ădaysMar 4, 1817      19ăyears, 116ădays  Jun 28, 1836
4  66ăyears, 310ădaysMar 4, 1825       6ăyears, 122ădays   Jul 4, 1831

                  Age
0  67ăyears, 295ădays
1  90ăyears, 247ădays
2   83ăyears, 82ădays
3  85ăyears, 104ădays
4   73ăyears, 67ădays
```

[19]:
```
# Ok, we have some presidents, some dates, I see a bunch of footnotes in the␣
 ↪"Born" column which might cause
# issues. Let's start with cleaning up that name into firstname and lastname.␣
 ↪I'm going to tackle this with
# a regex. So I want to create two new columns and apply a regex to the␣
 ↪projection of the "President" column.
```

```
# Here's one solution, we could make a copy of the President column
df["First"]=df['President']

# Then we can call replace() and just have a pattern that matches the last name␣
↪and set it to an empty string

df["First"]=df["First"].replace("[ ].*", "", regex=True)
# buscamos en la columna "First" un patrón que contenga un espacio([ ]) seguido␣
↪de numerosos caracteres(.*) y que lo reemplace por un string
# vacío ("") en la misma columna donde lo buscamos


# Now let's take a look
df.head()
```

[19]:
```
   #          President              Born      Age atstart of presidency  \
0  1  George Washington  Feb 22, 1732[a]  57ăyears, 67ădaysApr 30, 1789
1  2         John Adams  Oct 30, 1735[a]  61ăyears, 125ădaysMar 4, 1797
2  3   Thomas Jefferson  Apr 13, 1743[a]  57ăyears, 325ădaysMar 4, 1801
3  4      James Madison  Mar 16, 1751[a]  57ăyears, 353ădaysMar 4, 1809
4  5       James Monroe     Apr 28, 1758  58ăyears, 310ădaysMar 4, 1817

         Age atend of presidency Post-presidencytimespan          Died  \
0   65ăyears, 10ădaysMar 4, 1797       2ăyears, 285ădays  Dec 14, 1799
1  65ăyears, 125ădaysMar 4, 1801      25ăyears, 122ădays   Jul 4, 1826
2  65ăyears, 325ădaysMar 4, 1809      17ăyears, 122ădays   Jul 4, 1826
3  65ăyears, 353ădaysMar 4, 1817      19ăyears, 116ădays  Jun 28, 1836
4  66ăyears, 310ădaysMar 4, 1825       6ăyears, 122ădays   Jul 4, 1831

                  Age   First
0  67ăyears, 295ădays  George
1  90ăyears, 247ădays    John
2   83ăyears, 82ădays  Thomas
3  85ăyears, 104ădays   James
4   73ăyears, 67ădays   James
```

[20]:
```
# That works, but it's kind of gross. And it's slow, since we had to make a␣
↪full copy of a column then go
# through and update strings. There are a few other ways we can deal with this.␣
↪Let me show you the most
# general one first, and that's called the apply() function. Let's drop the␣
↪column we made first
del(df["First"])

# The apply() function on a dataframe will take some arbitrary function you␣
↪have written and apply it to
```

```
# either a Series (a single column) or DataFrame across all rows or columns.␣
↪Lets write a function which
# just splits a string into two pieces using a single row of data
def splitname(row):
    # The row is a single Series object which is a single row indexed by column␣
↪values
    # Let's extract the firstname and create a new entry in the series
    row['First']=row['President'].split(" ")[0]
    # Let's do the same with the last word in the string
    row['Last']=row['President'].split(" ")[-1]
    # Now we just return the row and the pandas .apply() will take of merging␣
↪them back into a DataFrame
    return row

# Now if we apply this to the dataframe indicating we want to apply it across␣
↪columns
df=df.apply(splitname, axis='columns')
df.head()
```

[20]:
```
   #          President              Born      Age atstart of presidency  \
0  1  George Washington  Feb 22, 1732[a]  57ăyears, 67ădaysApr 30, 1789
1  2         John Adams  Oct 30, 1735[a]  61ăyears, 125ădaysMar 4, 1797
2  3   Thomas Jefferson  Apr 13, 1743[a]  57ăyears, 325ădaysMar 4, 1801
3  4      James Madison  Mar 16, 1751[a]  57ăyears, 353ădaysMar 4, 1809
4  5       James Monroe     Apr 28, 1758  58ăyears, 310ădaysMar 4, 1817

        Age atend of presidency Post-presidencytimespan          Died  \
0   65ăyears, 10ădaysMar 4, 1797       2ăyears, 285ădays  Dec 14, 1799
1  65ăyears, 125ădaysMar 4, 1801      25ăyears, 122ădays   Jul 4, 1826
2  65ăyears, 325ădaysMar 4, 1809      17ăyears, 122ădays   Jul 4, 1826
3  65ăyears, 353ădaysMar 4, 1817      19ăyears, 116ădays  Jun 28, 1836
4  66ăyears, 310ădaysMar 4, 1825       6ăyears, 122ădays   Jul 4, 1831

                  Age   First        Last
0  67ăyears, 295ădays  George  Washington
1  90ăyears, 247ădays    John       Adams
2   83ăyears, 82ădays  Thomas   Jefferson
3  85ăyears, 104ădays   James     Madison
4   73ăyears, 67ădays   James      Monroe
```

[21]:
```
# Pretty questionable as to whether that is less gross, but it achieves the␣
↪result and I find that I use the
# apply() function regularly in my work. The pandas series has a couple of␣
↪other nice convenience functions
# though, and the next I would like to touch on is called .extract(). Lets drop␣
↪our firstname and lastname.
del(df['First'])
del(df['Last'])

# Extract takes a regular expression as input and specifically requires you to␣
 ↪set capture groups that
# correspond to the output columns you are interested in. And, this is a great␣
 ↪place for you to pause the
# video and reflect - if you were going to write a regular expression that␣
 ↪returned groups and just had the
# firstname and lastname in it, what would that look like?

# Here's my solution, where we match three groups but only return two, the␣
 ↪first and the last name
pattern="(^[\w]*)(?:.* )([\w]*$)"
# Acá el segundo grupo, al poner ?: se indica que ese grupo NO debe agregarse␣
 ↪al resultado devuelvo por REGEX
# Por eso el resultado es una matriz con 2 columnas, Nombre (grupo1) y Apellido␣
 ↪(grupo2)



# Now the extract function is built into the str attribute of the Series␣
 ↪object, so we can call it
# using Series.str.extract(pattern)
df["President"].str.extract(pattern).head()
```

[21]:
```
        0           1
0  George  Washington
1    John       Adams
2  Thomas   Jefferson
3   James     Madison
4   James      Monroe
```

[5]:
```
# So that looks pretty nice, other than the column names. But if we name the␣
 ↪groups we get named columns out

# acá es lo mismo que arriba, mismo patrón pero a los grupos1 y 3 se le␣
 ↪asingnan nombres para que aparezcan en las columnas
pattern="(?P<First>^[\w]*)(?:.* )(?P<Last>[\w]*$)"

# Now call extract

# Se crea una nueva matriz a partir de la columna de la matriz original df
names=df["President"].str.extract(pattern).head()
names
```

[5]:
```
    First        Last
0  George  Washington
1    John       Adams
```

```
2  Thomas   Jefferson
3   James     Madison
4   James      Monroe
```

[6]:
```
# And we can just copy these into our main dataframe if we want to

# Copiamos las nuevas columnas dentro de la matriz original
df["First"]=names["First"]
df["Last"]=names["Last"]
df.head()
```

[6]:
```
   #          President             Born      Age atstart of presidency  \
0  1  George Washington  Feb 22, 1732[a]  57ăyears, 67ădaysApr 30, 1789
1  2         John Adams  Oct 30, 1735[a]  61ăyears, 125ădaysMar 4, 1797
2  3   Thomas Jefferson  Apr 13, 1743[a]  57ăyears, 325ădaysMar 4, 1801
3  4      James Madison  Mar 16, 1751[a]  57ăyears, 353ădaysMar 4, 1809
4  5       James Monroe     Apr 28, 1758  58ăyears, 310ădaysMar 4, 1817

        Age atend of presidency Post-presidencytimespan          Died  \
0   65ăyears, 10ădaysMar 4, 1797       2ăyears, 285ădays  Dec 14, 1799
1  65ăyears, 125ădaysMar 4, 1801      25ăyears, 122ădays   Jul 4, 1826
2  65ăyears, 325ădaysMar 4, 1809      17ăyears, 122ădays   Jul 4, 1826
3  65ăyears, 353ădaysMar 4, 1817      19ăyears, 116ădays  Jun 28, 1836
4  66ăyears, 310ădaysMar 4, 1825       6ăyears, 122ădays   Jul 4, 1831

                Age   First        Last
0  67ăyears, 295ădays  George  Washington
1  90ăyears, 247ădays    John       Adams
2   83ăyears, 82ădays  Thomas   Jefferson
3  85ăyears, 104ădays   James     Madison
4   73ăyears, 67ădays   James      Monroe
```

[7]:
```
# It's worth looking at the pandas str module for other functions which have␣
 ↪been written specifically
# to clean up strings in DataFrames, and you can find that in the docs in the␣
 ↪Working with Text
# section: https://pandas.pydata.org/pandas-docs/stable/user_guide/text.html
```

[8]:
```
# Now lets move on to clean up that Born column. First, let's get rid of␣
 ↪anything that isn't in the
# pattern of Month Day and Year

# usamos regex para extraer únicamente las fechas compuestas por MMM DD, AAAA
# se reemplaza la columna Born por los nuevos datos

df["Born"]=df["Born"].str.extract("([\w]{3} [\w]{1,2}, [\w]{4})")
df["Born"].head()
```

[8]:
```
0    Feb 22, 1732
1    Oct 30, 1735
2    Apr 13, 1743
3    Mar 16, 1751
4    Apr 28, 1758
Name: Born, dtype: object
```

[9]:
```
# So, that cleans up the date format. But I'm going to foreshadow something␣
 ↪else here - the type of this
# column is object, and we know that's what pandas uses when it is dealing with␣
 ↪string. But pandas actually
# has really interesting date/time features - in fact, that's one of the␣
 ↪reasons Wes McKinney put his efforts
# into the library, to deal with financial transactions. So if I were building␣
 ↪this out, I would actually
# update this column to the write data type as well
df["Born"]=pd.to_datetime(df["Born"])
df["Born"].head()
```

[9]:
```
0   1732-02-22
1   1735-10-30
2   1743-04-13
3   1751-03-16
4   1758-04-28
Name: Born, dtype: datetime64[ns]
```

[10]:
```
# This would make subsequent processing on the dataframe around dates, such as␣
 ↪getting every President who
# was born in a given time span, much easier.
```

Now, most of the other columns in this dataset I would clean in a similar fashion. And this would be a good practice activity for you, so I would recommend that you pause the video, open up the notebook for the lecture if you don't already have it opened, and then finish cleaning up this dataframe. In this lecture I introduced you to the str module which has a number of important functions for cleaning pandas dataframes. You don't have to use these - I actually use apply() quite a bit myself, especially if I don't need high performance data cleaning because my dataset is small. But the str functions are incredibly useful and build on your existing knowledge of regular expressions, and because they are vectorized they are efficient to use as well.