pandas-data-cleaning-tricks

November 25, 2018

1 Tricks for cleaning your data in Python using pandas

By Christine Zhang (ychristinezhang at gmail dot com

GitHub repository for Data+Code: https://github.com/underthecurve/pandas-data-cleaning-tricks

In 2017 I gave a talk called "Tricks for cleaning your data in R" which I presented at the Data+Narrative workshop at Boston University. The repo with the code and data, https://github.com/underthecurve/r-data-cleaning-tricks, was pretty well-received, so I figured I'd try to do some of the same stuff in Python using `pandas`.

Disclaimer: when it comes to data stuff, I'm much better with R, especially the `tidyverse` set of packages, than with Python, but in my last job I used Python's `pandas` library to do a lot of data processing since Python was the dominant language there.

Anyway, here goes:

Data cleaning is a cumbersome task, and it can be hard to navigate in programming languages like Python.

The `pandas` library in Python is a powerful tool for data cleaning and analysis. By default, it leaves a trail of code that documents all the work you've done, which makes it extremely useful for creating reproducible workflows.

In this workshop, I'll show you some examples of real-life "messy" datasets, the problems they present for analysis in Python's `pandas` library, and some of the solutions to these problems.

Fittingly, I'll start the numbering system at 0.

1.1 0. Importing the `pandas` library

Here I tell Python to import the `pandas` library as `pd` (a common alias for `pandas` — more on that in the next code chunk).

```
In [1]: import pandas as pd
```

1.2 1. Finding and replacing non-numeric characters like , and $

Let's check out the city of Boston's Open Data portal, where the local government puts up datasets that are free for the public to analyze.

The Employee Earnings Report is one of the more interesting ones, because it gives payroll data for every person on the municipal payroll. It's where the *Boston Globe* gets stories like these every year:

- "64 City of Boston workers earn more than $250,000" (February 6, 2016)

- "Police detective tops Boston's payroll with a total of over $403,000" (February 14, 2017)

Let's take at the February 14 story from 2017. The story begins:

> "A veteran police detective took home more than $403,000 in earnings last year, topping the list of Boston's highest-paid employees in 2016, newly released city payroll data show."

What if we wanted to check this number using the Employee Earnings Report?

We can use the `pandas` function `pandas.read_csv()` to load the csv file into Python. We will call this DataFrame `salary`. Remember that I imported `pandas` "as pd" in the last code chunk. This saves me a bit of typing by allowing me to access `pandas` functions like `pandas.read_csv()` by typing `pd.read_csv()` instead. If I had typed `import pandas` in the code chunk under section `0` without `as pd`, the below code wouldn't work. I'd have to instead write `pandas.read_csv()` to access the function.

The `pd` alias for `pandas` is so common that the library's documentation even uses it sometimes.

Let's try to use `pd.read_csv()`:

```
In [2]: salary = pd.read_csv('employee-earnings-report-2016.csv')

        ---------------------------------------------------------------------------

        UnicodeDecodeError                        Traceback (most recent call last)

        pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._convert_tokens()


        pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._convert_with_dtype()


        pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._string_convert()


        pandas/_libs/parsers.pyx in pandas._libs.parsers._string_box_utf8()


        UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe9 in position 22: invalid contin

    During handling of the above exception, another exception occurred:


        UnicodeDecodeError                        Traceback (most recent call last)

        <ipython-input-2-188a5f5f7049> in <module>()
    ----> 1 salary = pd.read_csv('employee-earnings-report-2016.csv')
```

```
    ~/anaconda/envs/ipykernel_py3/lib/python3.6/site-packages/pandas/io/parsers.py in parse
    676                     skip_blank_lines=skip_blank_lines)
    677
--> 678         return _read(filepath_or_buffer, kwds)
    679
    680     parser_f.__name__ = name


    ~/anaconda/envs/ipykernel_py3/lib/python3.6/site-packages/pandas/io/parsers.py in _read
    444
    445     try:
--> 446         data = parser.read(nrows)
    447     finally:
    448         parser.close()


    ~/anaconda/envs/ipykernel_py3/lib/python3.6/site-packages/pandas/io/parsers.py in read
   1034                 raise ValueError('skipfooter not supported for iteration')
   1035
-> 1036         ret = self._engine.read(nrows)
   1037
   1038         # May alter columns / col_dict


    ~/anaconda/envs/ipykernel_py3/lib/python3.6/site-packages/pandas/io/parsers.py in read
   1846     def read(self, nrows=None):
   1847         try:
-> 1848             data = self._reader.read(nrows)
   1849         except StopIteration:
   1850             if self._first_chunk:


    pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.read()


    pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._read_low_memory()


    pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._read_rows()


    pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._convert_column_data()


    pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._convert_tokens()


    pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._convert_with_dtype()
```

```
pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._string_convert()

pandas/_libs/parsers.pyx in pandas._libs.parsers._string_box_utf8()

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe9 in position 22: invalid contin
```

That's a pretty long and nasty error. Usually when I run into something like this, I start from the bottom and work my way up — in this case, I typed `UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe9 in position 22: invalid continuation byte` into a search engine and came across this discussion on the Stack Overflow forum. The last response suggested that adding `encoding ='latin1'` inside the function would fix the problem on Macs (which is the type of computer I have).

```
In [3]: salary = pd.read_csv('employee-earnings-report-2016.csv', encoding = 'latin-1')
```

Great! (I don't know much about encoding, but this is something I run into from time to time so I thought it would be helpful to show here.)

We can use `head()` on the `salary` DataFrame to inspect the first five rows of `salary`. (Note I use the `print()` to display the output, but you don't need to do this in your own code if you'd prefer not to.)

```
In [4]: print(salary.head())
```

```
                      NAME           DEPARTMENT_NAME                 TITLE  \
0           Abadi,Kidani A      Assessing Department      Property Officer
1         Abasciano,Joseph  Boston Police Department        Police Officer
2   Abban,Christopher John    Boston Fire Department          Fire Fighter
3            Abbasi,Sophia             Green Academy  Manager (C) (non-ac)
4  Abbate-Vaughn,Jorgelina      BPS Ellis Elementary               Teacher

       REGULAR RETRO      OTHER    OVERTIME     INJURED     DETAIL  \
0   $46,291.98   NaN    $300.00         NaN         NaN        NaN
1    $6,933.66   NaN    $850.00     $205.92  $74,331.86        NaN
2  $103,442.22   NaN    $550.00  $15,884.53         NaN  $4,746.50
3   $18,249.83   NaN        NaN         NaN         NaN        NaN
4   $84,410.28   NaN  $1,250.00         NaN         NaN        NaN

  QUINN/EDUCATION INCENTIVE TOTAL EARNINGS POSTAL
0                       NaN     $46,591.98  02118
1               $15,258.44     $97,579.88  02132
2                       NaN    $124,623.25  02132
3                       NaN     $18,249.83  02148
4                       NaN     $85,660.28  02481
```

There are a lot of columns. Let's simplify by selecting the ones of interest: `NAME`, `DEPARTMENT_NAME`, and `TOTAL.EARNINGS`. There are a few different ways of doing this with `pandas`. The simplest way, imo, is by using the indexing operator `[]`.

For example, I could select a single column, `NAME`: (Note I also run the line `pd.options.display.max_rows = 20` in order to display a maximum of 20 rows so the output isn't too crowded.)

```
In [5]: pd.options.display.max_rows = 20

        salary['NAME']

Out[5]: 0                    Abadi,Kidani A
        1                  Abasciano,Joseph
        2             Abban,Christopher John
        3                      Abbasi,Sophia
        4            Abbate-Vaughn,Jorgelina
        5                   Abberton,James P
        6               Abbott,Erin Elizabeth
        7                     Abbott,John R.
        8                 Abbruzzese,Angela
        9                  Abbruzzese,Donna
                             ...
        22036          Zuares,David Jonathan
        22037              Zubrin,William W.
        22038                Zuccaro,John E.
        22039             Zucker,Alyse Paige
        22040          Zuckerman,Naomi Julia
        22041           Zukowski III,Charles
        22042    Zuluaga  Castro,Juan Pablo
        22043         Zwarich,Maralene Zoann
        22044                Zweig,Susanna B
        22045                Zwerdling,Laura
        Name: NAME, Length: 22046, dtype: object
```

This works for selecting one column at a time, but using `[]` returns a Series, not a DataFrame. I can confirm this using the `type()` function:

```
In [6]: type(salary['NAME'])

Out[6]: pandas.core.series.Series
```

If I want a DataFrame, I have to use double brackets:

```
In [7]: salary[['NAME']]

Out[7]:                              NAME
        0                    Abadi,Kidani A
        1                  Abasciano,Joseph
        2             Abban,Christopher John
```

```
3                    Abbasi,Sophia
4          Abbate-Vaughn,Jorgelina
5                 Abberton,James P
6            Abbott,Erin Elizabeth
7                   Abbott,John R.
8                Abbruzzese,Angela
9                 Abbruzzese,Donna
...                             ...
22036       Zuares,David Jonathan
22037           Zubrin,William W.
22038             Zuccaro,John E.
22039          Zucker,Alyse Paige
22040       Zuckerman,Naomi Julia
22041        Zukowski III,Charles
22042  Zuluaga  Castro,Juan Pablo
22043      Zwarich,Maralene Zoann
22044             Zweig,Susanna B
22045             Zwerdling,Laura

[22046 rows x 1 columns]
```

```
In [8]: type(salary[['NAME']])

Out[8]: pandas.core.frame.DataFrame
```

To select multiple columns, we can put those columns inside of the second pair of brackets. We will save this into a new DataFrame, `salary_selected`. We type `.copy()` after `salary[['NAME','DEPARTMENT_NAME', 'TOTAL EARNINGS']]` because we are making a copy of the DataFrame and assigning it to new DataFrame. Learn more about `copy()` here.

```
In [9]: salary_selected = salary[['NAME','DEPARTMENT_NAME', 'TOTAL EARNINGS']].copy()
```

We can also change the column names to lowercase names for easier typing. First, let's take a look at the columns by displaying the `columns` attribute of the `salary_selected` DataFrame.

```
In [10]: salary_selected.columns

Out[10]: Index(['NAME', 'DEPARTMENT_NAME', 'TOTAL EARNINGS'], dtype='object')

In [11]: type(salary_selected.columns)

Out[11]: pandas.core.indexes.base.Index
```

Notice how this returns something called an "Index." In `pandas`, DataFrames have both row indexes (in our case, the row number, starting from 0 and going to 22045) and column indexes. We can use the `str.lower()` function to convert the strings (aka characters) in the index to lowercase.

```
In [12]: salary_selected.columns = salary_selected.columns.str.lower()

         salary_selected.columns
```

```
Out[12]: Index(['name', 'department_name', 'total earnings'], dtype='object')
```

Another thing that will make our lives easier is if the `total earnings` column didn't have a space between `total` and `earnings`. We can use a "string replace" function, `str.replace()`, to replace the space with an underscore. The syntax is: `str.replace('thing you want to replace', 'what to replace it with')`

```
In [13]: salary_selected.columns.str.replace(' ', '_')

         salary_selected.columns

Out[13]: Index(['name', 'department_name', 'total earnings'], dtype='object')
```

We could have used both the `str.lower()` and `str.replace()` functions in one line of code by putting them one after the other (aka "chaining"):

```
In [14]: salary_selected.columns = salary_selected.columns.str.lower().str.replace(' ', '_')

         salary_selected.columns

Out[14]: Index(['name', 'department_name', 'total_earnings'], dtype='object')
```

Let's use `head()` to visually inspect the first five rows of `salary_selected`:

```
In [15]: print(salary_selected.head())

                     name           department_name total_earnings
0          Abadi,Kidani A      Assessing Department     $46,591.98
1        Abasciano,Joseph  Boston Police Department     $97,579.88
2   Abban,Christopher John    Boston Fire Department    $124,623.25
3           Abbasi,Sophia              Green Academy     $18,249.83
4  Abbate-Vaughn,Jorgelina      BPS Ellis Elementary     $85,660.28
```

Now let's try sorting the data by `total.earnings` using the `sort_values()` function in pandas:

```
In [16]: salary_sort = salary_selected.sort_values('total_earnings')
```

We can use `head()` to visually inspect `salary_sort`:

```
In [17]: print(salary_sort.head())

                     name              department_name total_earnings
11146     Lally,Bernadette           Boston City Council      $1,000.00
7104   Fowlkes,Lorraine E.           Boston City Council      $1,000.00
15058         Nolan,Andrew              Parks Department      $1,000.00
21349   White-Pilet,Yoni A  BPS Substitute Teachers/Nurs      $1,006.53
5915           Dunn,Lori D          BPS East Boston High      $1,010.05
```

At first glance, it looks okay. The employees appear to be sorted by `total_earnings` from lowest to highest. If this were the case, we'd expect the last row of the `salary_sort` DataFrame to contain the employee with the highest salary. Let's take a look at the last five rows using `tail()`.

```
In [18]: print(salary_sort.tail())
```

```
                   name               department_name total_earnings
13303   McGrath,Caitlin  BPS Substitute Teachers/Nurs        $990.61
1869   Bradshaw,John E.  BPS Substitute Teachers/Nurs        $990.62
21380   Wiggins,Lucas A  BPS Substitute Teachers/Nurs        $990.63
15036       Nixon,Chloe  BPS Substitute Teachers/Nurs        $990.64
10478   Kassa,Selamawit  BPS Substitute Teachers/Nurs        $990.64
```

What went wrong?

The problem is that there are non-numeric characters, , and $, in the `total.earnings` column. We can see with `dtypes`, which returns the data type of each column in the DataFrame, that `total_earnings` is recognized as an "object".

```
In [19]: salary_selected.dtypes
```

```
Out[19]: name               object
         department_name    object
         total_earnings     object
         dtype: object
```

Here is an overview of `pandas` data types. Basically, being labeled an "object" means that the column is not being recognized as containing numbers.

We need to find the , and $ in `total.earnings` and remove them. The `str.replace()` function, which we used above when renaming the columns, lets us do this.

Let's start by removing the comma and write the result to the original column. (The format for calling a column from a DataFrame in `pandas` is `DataFrame['column_name']`)

```
In [20]: salary_selected['total_earnings'] = salary_selected['total_earnings'].str.replace(','
```

Using `head()` to visually inspect `salary_selected`, we see that the commas are gone:

```
In [21]: print(salary_selected.head()) # this works - the commas are gone
```

```
                     name           department_name total_earnings
0           Abadi,Kidani A      Assessing Department      $46591.98
1         Abasciano,Joseph  Boston Police Department      $97579.88
2   Abban,Christopher John    Boston Fire Department     $124623.25
3            Abbasi,Sophia             Green Academy      $18249.83
4  Abbate-Vaughn,Jorgelina      BPS Ellis Elementary      $85660.28
```

Let's do the same thing, with the dollar sign $:

```
In [22]: salary_selected['total_earnings'] = salary_selected['total_earnings'].str.replace('$'
```

Using `head()` to visually inspect `salary_selected`, we see that the dollar signs are gone:

```
In [23]: salary_selected.head()
```

```
Out[23]:
                        name           department_name total_earnings
0           Abadi,Kidani A      Assessing Department       46591.98
1         Abasciano,Joseph  Boston Police Department       97579.88
2   Abban,Christopher John    Boston Fire Department      124623.25
3            Abbasi,Sophia             Green Academy       18249.83
4  Abbate-Vaughn,Jorgelina      BPS Ellis Elementary       85660.28
```

Now can we use `arrange()` to sort the data by `total_earnings`?

```
In [24]: salary_sort = salary_selected.sort_values('total_earnings')

         salary_sort.head()
```

```
Out[24]:
                            name       department_name total_earnings
3315            Charles,Yveline    BPS Transportation          10.07
9914       Jean Baptiste,Hugues    BPS Transportation          10.12
16419             Piper,Sarah A    BPS Transportation          10.47
11131        Laguerre,Yolaine M    BPS Transportation          10.94
17641  Rosario Severino,Yomayra  Food & Nutrition Svc         100.00
```

```
In [25]: salary_sort.tail()
```

```
Out[25]:
                        name               department_name total_earnings
18134       Santos,Maria C                   Curley K-8       99970.30
5999     Dyson,Margaret O.              Parks Department       99972.07
13012  McCarthy,Margaret M  BPS Substitute Teachers/Nurs        9998.47
1083   Bartholet,Carolyn V          BPS Mckay Elementary       99989.18
1960     Bresnahan,John M.      Boston Police Department       99997.38
```

Again, at first glance, the employees appear to be sorted by `total_earnings` from lowest to highest. But that would imply that John M. Bresnahan was the highest-paid employee, making 99,997.38 dollars in 2016, while the *Boston Globe* story said the highest-paid city employee made more than 403,000 dollars.

What's the problem?

Again, we can use `dtypes` to check on how the `total_earnings` variable is encoded.

```
In [26]: salary_sort.dtypes
```

```
Out[26]: name              object
         department_name   object
         total_earnings    object
         dtype: object
```

It's still an "object" now (still not numeric), because we didn't tell `pandas` that it should be numeric. We can do this with `pd.to_numeric()`:

```
In [27]: salary_sort['total_earnings'] = pd.to_numeric(salary_sort['total_earnings'])
```

Now let's run `dtypes` again:

```
In [28]: salary_sort.dtypes
```

```
Out[28]: name                object
         department_name     object
         total_earnings     float64
         dtype: object
```

"float64" means "floating point numbers" — this is what we want.
Now let's sort using `sort_values()`.

```
In [29]: salary_sort = salary_sort.sort_values('total_earnings')

         salary_sort.head() # ascending order by default
```

```
Out[29]:                          name       department_name  total_earnings
         9849        Jameau,Bernadette    BPS Transportation            2.14
         1986    Bridgewaters,Sandra J    BPS Transportation            2.50
         13853      Milian,Sonia Maria    BPS Transportation            3.85
         2346   Burke II,Myrell Nadine    BPS Transportation            4.38
         7717      Gillard Jr.,Trina F  Food & Nutrition Svc            5.00
```

One last thing: we have to specify `ascending = False` within `sort_values()` because the function by default sorts the data in ascending order.

```
In [30]: salary_sort = salary_sort.sort_values('total_earnings', ascending = False)

         salary_sort.head() # descending order
```

```
Out[30]:                    name           department_name  total_earnings
         11489        Lee,Waiman  Boston Police Department       403408.61
         10327  Josey,Windell C.  Boston Police Department       396348.50
         15716    Painten,Paul A  Boston Police Department       373959.35
         2113      Brown,Gregory  Boston Police Department       351825.50
         9446      Hosein,Haseeb  Boston Police Department       346105.17
```

We see that Waiman Lee from the Boston PD is the top earner with >403,408 per year, just as the *Boston Globe* article states.

A bonus thing: maybe it bothers you that the numbers next to each row are no longer in any numeric order. This is because these numbers are the row index of the DataFrame — basically the order that they were in prior to being sorted. In order to reset these numbers, we can use the `reset_index()` function on the `salary_sort` DataFrame. We include `drop = True` as a parameter of the function to prevent the old index from being added as a column in the DataFrame.

```
In [31]: salary_sort = salary_sort.reset_index(drop = True)

         salary_sort.head() # index is reset
```

```
Out[31]:                name           department_name  total_earnings
         0       Lee,Waiman  Boston Police Department       403408.61
         1  Josey,Windell C.  Boston Police Department       396348.50
         2    Painten,Paul A  Boston Police Department       373959.35
         3     Brown,Gregory  Boston Police Department       351825.50
         4     Hosein,Haseeb  Boston Police Department       346105.17
```

The Boston Police Department has a lot of high earners. We can figure out the average earnings by department, which we'll call `salary_average`, by using the `groupby` and `mean()` functions in `pandas`.

```
In [32]: salary_average = salary_sort.groupby('department_name').mean()

In [33]: salary_average = salary_average

         salary_average
```

```
Out[33]:                                 total_earnings
         department_name
         ASD Human Resources               67236.150755
         ASD Intergvernmtl Relations       83787.581000
         ASD Office Of Labor Relation      58899.954615
         ASD Office of Budget Mangmnt      73946.044643
         ASD Purchasing Division           72893.203750
         Accountability                   102073.280667
         Achievement Gap                   60105.522500
         Alighieri Montessori School       55160.025556
         Assessing Department              70713.327111
         Asst Superintendent-Network A    132514.885000
         ...                                        ...
         Unified Student Svc               65018.485000
         Veterans' Services                48411.606250
         WREC: Urban Science Academy       81170.398214
         Warren/Prescott K-8               66389.351341
         West Roxbury Academy              70373.066494
         West Zone ELC                     55868.384118
         Women's Advancement               63811.150000
         Workers Compensation Service      23797.119133
         Young Achievers K-8               56534.020463
         Youth Engagement & Employment     33645.202308

         [228 rows x 1 columns]
```

Notice that `pandas` by default sets the `department_name` column as the row index of the `salary_average` DataFrame. I personally don't love this and would rather have a straight-up DataFrame with the row numbers as the index, so I usually run `reset_index()` to get rid of this indexing:

```
In [34]: salary_average = salary_average.reset_index() # reset_index

         salary_average
```

```
Out[34]:                    department_name  total_earnings
         0              ASD Human Resources    67236.150755
         1       ASD Intergvernmtl Relations    83787.581000
         2      ASD Office Of Labor Relation    58899.954615
         3      ASD Office of Budget Mangmnt    73946.044643
         4           ASD Purchasing Division    72893.203750
         5                     Accountability   102073.280667
         6                   Achievement Gap    60105.522500
         7       Alighieri Montessori School    55160.025556
         8             Assessing Department    70713.327111
         9    Asst Superintendent-Network A   132514.885000
         ..                             ...             ...
         218            Unified Student Svc    65018.485000
         219             Veterans' Services    48411.606250
         220    WREC: Urban Science Academy    81170.398214
         221            Warren/Prescott K-8    66389.351341
         222           West Roxbury Academy    70373.066494
         223                  West Zone ELC    55868.384118
         224           Women's Advancement    63811.150000
         225   Workers Compensation Service    23797.119133
         226            Young Achievers K-8    56534.020463
         227  Youth Engagement & Employment    33645.202308

         [228 rows x 2 columns]
```

We should also rename the `total_earnings` column to `average_earnings` to avoid confusion. We can do this using `rename()`. The syntax for `rename()` is `DataFrame.rename(columns = {'current column name':'new column name'})`.

```
In [35]: salary_average = salary_average.rename(columns = {'total_earnings': 'dept_average'})

In [36]: salary_average

Out[36]:                    department_name   dept_average
         0              ASD Human Resources   67236.150755
         1       ASD Intergvernmtl Relations   83787.581000
         2      ASD Office Of Labor Relation   58899.954615
         3      ASD Office of Budget Mangmnt   73946.044643
         4           ASD Purchasing Division   72893.203750
         5                     Accountability  102073.280667
         6                   Achievement Gap   60105.522500
         7       Alighieri Montessori School   55160.025556
         8             Assessing Department   70713.327111
         9    Asst Superintendent-Network A  132514.885000
         ..                             ...            ...
         218            Unified Student Svc   65018.485000
         219             Veterans' Services   48411.606250
         220    WREC: Urban Science Academy   81170.398214
         221            Warren/Prescott K-8   66389.351341
```

```
222           West Roxbury Academy   70373.066494
223                  West Zone ELC   55868.384118
224            Women's Advancement   63811.150000
225   Workers Compensation Service   23797.119133
226            Young Achievers K-8   56534.020463
227  Youth Engagement & Employment   33645.202308

[228 rows x 2 columns]
```

We can find the Boston Police Department. Find out more about selecting based on attributes here.

```
In [37]: salary_average[salary_average['department_name'] == 'Boston Police Department']
```

```
Out[37]:              department_name   dept_average
         121  Boston Police Department  124787.164775
```

Now is a good time to revisit "chaining." Notice how we did three things in creating `salary_average`: 1. Grouped the `salary_sort` DataFrame by `department_name` and calculated the mean of the numeric columns (in our case, `total_earnings` using `group_by()` and `mean()`. 2. Used `reset_index()` on the resulting DataFrame so that `department_name` would no longer be the row index. 3. Renamed the `total_earnings` column to `dept_average` to avoid confusion using `rename()`.

In fact, we can do these three things all at once, by chaining the functions together:

```
In [38]: salary_sort.groupby('department_name').mean().reset_index().rename(columns = {'total_
```

```
Out[38]:                    department_name   dept_average
         0              ASD Human Resources   67236.150755
         1      ASD Intergvernmtl Relations   83787.581000
         2     ASD Office Of Labor Relation   58899.954615
         3     ASD Office of Budget Mangmnt   73946.044643
         4          ASD Purchasing Division   72893.203750
         5                   Accountability  102073.280667
         6                  Achievement Gap   60105.522500
         7      Alighieri Montessori School   55160.025556
         8             Assessing Department   70713.327111
         9    Asst Superintendent-Network A  132514.885000
         ..                             ...            ...
         218           Unified Student Svc   65018.485000
         219            Veterans' Services   48411.606250
         220    WREC: Urban Science Academy   81170.398214
         221            Warren/Prescott K-8   66389.351341
         222           West Roxbury Academy   70373.066494
         223                  West Zone ELC   55868.384118
         224            Women's Advancement   63811.150000
         225   Workers Compensation Service   23797.119133
         226            Young Achievers K-8   56534.020463
         227  Youth Engagement & Employment   33645.202308
```

```
[228 rows x 2 columns]
```

That's a pretty long line of code. To make it more readable, we can split it up into separate lines. I like to do this by putting the whole expression in parentheses and splitting it up right before each of the functions, which are delineated by the periods:

```
In [39]: (salary_sort.groupby('department_name')
          .mean()
          .reset_index()
          .rename(columns = {'total_earnings':'dept_average'}))
```

Out[39]:

	department_name	dept_average
0	ASD Human Resources	67236.150755
1	ASD Intergvernmtl Relations	83787.581000
2	ASD Office Of Labor Relation	58899.954615
3	ASD Office of Budget Mangmnt	73946.044643
4	ASD Purchasing Division	72893.203750
5	Accountability	102073.280667
6	Achievement Gap	60105.522500
7	Alighieri Montessori School	55160.025556
8	Assessing Department	70713.327111
9	Asst Superintendent-Network A	132514.885000
..	...	...
218	Unified Student Svc	65018.485000
219	Veterans' Services	48411.606250
220	WREC: Urban Science Academy	81170.398214
221	Warren/Prescott K-8	66389.351341
222	West Roxbury Academy	70373.066494
223	West Zone ELC	55868.384118
224	Women's Advancement	63811.150000
225	Workers Compensation Service	23797.119133
226	Young Achievers K-8	56534.020463
227	Youth Engagement & Employment	33645.202308

```
[228 rows x 2 columns]
```

1.3 2. Merging datasets

Now we have two main datasets, `salary_sort` (the salary for each person, sorted from high to low) and `salary_average` (the average salary for each department). What if I wanted to merge these two together, so I could see side-by-side each person's salary compared to the average for their department?

We want to join by the `department_name` variable, since that is consistent across both datasets. Let's put the merged data into a new dataframe, `salary_merged`:

```
In [40]: salary_merged = pd.merge(salary_sort, salary_average, on = 'department_name')
```

Now we can see the department average, `dept_average`, next to the individual's salary, `total_earnings`:

```
In [41]: salary_merged.head()
```

```
Out[41]:                name           department_name  total_earnings   dept_average
         0       Lee,Waiman  Boston Police Department       403408.61  124787.164775
         1  Josey,Windell C.  Boston Police Department       396348.50  124787.164775
         2    Painten,Paul A  Boston Police Department       373959.35  124787.164775
         3     Brown,Gregory  Boston Police Department       351825.50  124787.164775
         4     Hosein,Haseeb  Boston Police Department       346105.17  124787.164775
```

1.4 3. Reshaping data

Here's a dataset on unemployment rates by country from 2012 to 2016, from the International Monetary Fund's World Economic Outlook database (available here).

When you download the dataset, it comes in an Excel file. We can use the `pd.read_excel()` function from `pandas` to load the file into Python.

```
In [42]: unemployment = pd.read_excel('unemployment.xlsx')
         unemployment.head()
```

```
Out[42]:      Country    2012    2013    2014    2015    2016
         0    Albania  13.400  16.000  17.500  17.100  16.100
         1    Algeria  11.000   9.829  10.600  11.214  10.498
         2  Argentina   7.200   7.075   7.250     NaN   8.467
         3    Armenia  17.300  16.200  17.600  18.500  18.790
         4  Australia   5.217   5.650   6.058   6.058   5.733
```

You'll notice if you open the `unemployment.xlsx` file in Excel that cells that do not have data (like Argentina in 2015) are labeled with "n/a". A nice feature of `pd.read_excel()` is that it recognizes these cells as NaN ("not a number," or Python's way of encoding missing values), by default. If we wanted to, we could explicitly tell pandas that missing values were labeled "n/a" using `na_values = 'n/a'` within the `pd.read_excel()` function:

```
In [43]: unemployment = pd.read_excel('unemployment.xlsx', na_values = 'n/a')
```

Right now, the data are in what's commonly referred to as "wide" format, meaning the variables (unemployment rate for each year) are spread across rows. This might be good for presentation, but it's not great for certain calculations or graphing. "Wide" format data also becomes confusing if other variables are added.

We need to change the format from "wide" to "long," meaning that the columns (2012, 2013, 2014, 2015, 2016) will be converted into a new variable, which we'll call `Year`, with repeated values for each country. And the unemployment rates will be put into a new variable, which we'll call `Rate_Unemployed`.

To do this, we'll use the `pd.melt()` function in `pandas` to create a new DataFrame, `unemployment_long`.

```
In [44]: unemployment_long = pd.melt(unemployment, # data to reshape
                                     id_vars = 'Country', # identifier variable
                                     var_name = 'Year', # column we want to create from the ro
                                     value_name = 'Rate_Unemployed') # the values of interest
```

Inspecting `unemployment_long` using `head()` shows that we have successfully created a long dataset.

```
In [45]: unemployment_long.head()
```

```
Out[45]:      Country  Year  Rate_Unemployed
          0    Albania  2012           13.400
          1    Algeria  2012           11.000
          2  Argentina  2012            7.200
          3    Armenia  2012           17.300
          4  Australia  2012            5.217
```

1.5 4. Calculating year-over-year change in panel data

Sort the data by `Country` and `Year` using the `sort_values()` function:

```
In [46]: unemployment_long = unemployment_long.sort_values(['Country', 'Year'])

         unemployment_long.head()
```

```
Out[46]:      Country  Year  Rate_Unemployed
          0    Albania  2012             13.4
          112  Albania  2013             16.0
          224  Albania  2014             17.5
          336  Albania  2015             17.1
          448  Albania  2016             16.1
```

Again, we can use `reset_index(drop = True)` to reset the row index so that the numbers next to the rows are in sequential order.

```
In [47]: unemployment_long = unemployment_long.reset_index(drop = True)

         unemployment_long.head()
```

```
Out[47]:    Country  Year  Rate_Unemployed
          0  Albania  2012             13.4
          1  Albania  2013             16.0
          2  Albania  2014             17.5
          3  Albania  2015             17.1
          4  Albania  2016             16.1
```

This type of data is known in time-series analysis as a panel; each country is observed every year from 2012 to 2016.

For Albania, the percentage point change in unemployment rate from 2012 to 2013 would be 16 - 13.4 = 2.5 percentage points. What if I wanted the year-over-year change in unemployment rate for every country?

We can use the `diff()` function in `pandas` to do this. We can use `diff()` to calculate the difference between the `Rate_Unemployed` that year and the `Rate_Unemployed` for the year prior (the default for `lag()` is 1 period, which is good for us since we want the change from the previous year). We will save this difference into a new variable, `Change`.

```
In [48]: unemployment_long['Change'] = unemployment_long.Rate_Unemployed.diff()
```

Let's inspect the first five rows again, using `head()`:

```
In [49]: unemployment_long.head()
```

```
Out[49]:    Country  Year  Rate_Unemployed  Change
         0  Albania  2012             13.4     NaN
         1  Albania  2013             16.0     2.6
         2  Albania  2014             17.5     1.5
         3  Albania  2015             17.1    -0.4
         4  Albania  2016             16.1    -1.0
```

So far so good. It also makes sense that Albania's `Change` is `NaN` in 2012, since the dataset doesn't contain any unemployment figures before the year 2012.

But a closer inspection of the data reveals a problem. What if we used `tail()` to look at the *last* 5 rows of the data?

```
In [50]: unemployment_long.tail()
```

```
Out[50]:      Country  Year  Rate_Unemployed  Change
         555  Vietnam  2012             2.74 -18.493
         556  Vietnam  2013             2.75   0.010
         557  Vietnam  2014             2.05  -0.700
         558  Vietnam  2015             2.40   0.350
         559  Vietnam  2016             2.40   0.000
```

Why does Vietnam have a -18.493 percentage point change in 2012?
(Hint: use `tail()` to look at the last 6 rows of the data.)

```
In [51]: unemployment_long['Change'] = (unemployment_long
                                        .groupby('Country')
                                        .Rate_Unemployed.diff())

         unemployment_long.tail()
```

```
Out[51]:      Country  Year  Rate_Unemployed  Change
         555  Vietnam  2012             2.74     NaN
         556  Vietnam  2013             2.75    0.01
         557  Vietnam  2014             2.05   -0.70
         558  Vietnam  2015             2.40    0.35
         559  Vietnam  2016             2.40    0.00
```

(Also notice how I put the entire expression in parentheses and put each function on a different line for readability.)

1.6 5. Recoding numerical variables into categorical ones

Here's a list of some attendees for the 2016 workshop, with names and contact info removed.

In [52]:
```
attendees = pd.read_csv('attendees.csv')

attendees.head()
```

Out[52]:

	Occupation	Job title	Age group	Gender	State/Province	Education	Which data subject area are you most interested in working with? (Select up to three	What do you hope to get out of the workshop?	Which type of laptop will you bring?	College or University Name	Major or Concentration	College Year
0	Data Analyst	Data Quality Analyst	30-39	Male	MA	Bachelor's Degree	Retail	other	PC	NaN	NaN	NaN
1	PhD Student	Student/Research Assistant	18-29	Male	MA	Bachelor's Degree	Sports	Master Advanced R	PC	Boston University	Biostatistics	PhD
2	Education	Data Analyst	18-29	Female	Kentucky	Master's Degree	Retail	other	PC	NaN	NaN	NaN
3	Manager	BAS Manager	30-39	Male	MA	Bachelor's Degree	Education	Pick up Beginning R And SQL	PC	Boston University	PEMBA	Graduate
4	Government Finance	Performance Analyst	30 - 39	Male	MA	Master's Degree	Environment, Finance, Food and agriculture	Pick up Beginning R And SQL	MAC	NaN	NaN	NaN

```
  Which Digital Badge track best suits you?  \
0                Advanced Data Storytelling
1                Advanced Data Storytelling
2                Advanced Data Storytelling
3                Advanced Data Storytelling
4                Advanced Data Storytelling

  Which session would you like to attend?  \
0                                 June 5-9
1                                 June 5-9
2                                 June 5-9
3                                 June 5-9
4                                 June 5-9

                           Choose your status:
0             Nonprofit, Academic, Government
1                                     Student
2             Nonprofit, Academic, Government
3                                     Student
4  Nonprofit, Academic, Government Early Bird
```

What if we wanted to quickly see the age distribution of attendees?

```
In [53]: attendees['Age group'].value_counts()
```

```
Out[53]: 30-39      7
         18-29      4
         30 - 39    1
         Name: Age group, dtype: int64
```

There's an inconsistency in the labeling of the `Age group` variable here. We can fix this using `np.where()` in the `numpy` library. First, let's import the `numpy` library. Like `pandas`, `numpy` has a commonly used alias — `np`.

```
In [54]: import numpy as np
```

```
In [55]: attendees['Age group'] = np.where(attendees['Age group'] == '30 - 39', # where attend
                                           '30-39', # replace attendees['Age group'] with '30-
                                           attendees['Age group']) # otherwise, keep attendees
```

This might seem trivial for just one value, but it's useful for larger datasets.

```
In [56]: attendees['Age group'].value_counts()
```

```
Out[56]: 30-39    8
         18-29    4
         Name: Age group, dtype: int64
```

Now let's take a look at the professional status of attendees, labeled in `Choose your status:`

```
In [57]: attendees['Choose your status:'].value_counts()

Out[57]: Student                                        5
         Nonprofit, Academic, Government                3
         Professional                                   3
         Nonprofit, Academic, Government Early Bird     1
         Name: Choose your status:, dtype: int64
```

"Nonprofit, Academic, Government" and "Nonprofit, Academic, Government Early Bird" seem to be the same. We can use `np.where()` (and the Python designation `|` for "or") to combine these two categories into one big category, "Nonprofit/Gov". Let's create a new variable, `status`, for our simplified categorization.

Notice the extra sets of parentheses around the two conditions linked by the `|` symbol.

```
In [58]: attendees['status'] = np.where((attendees['Choose your status:'] == 'Nonprofit, Acade
                                        (attendees['Choose your status:'] == 'Nonprofit, Acade
                                    'Nonprofit/Gov',
                                    attendees['Choose your status:'])

In [59]: attendees['status'].value_counts()

Out[59]: Student          5
         Nonprofit/Gov    4
         Professional     3
         Name: status, dtype: int64
```

1.7 What else?

- How would you create a new variable in the `attendees` data (let's call it `status2`) that has just two categories, "Student" and "Other"?
- How would you rename the variables in the `attendees` data to make them easier to work with?
- What are some other issues with this dataset? How would you solve them using what we've learned?
- What are some other "messy" data issues you've encountered?