Data Analysis in Pandas - Introduction

(https://github.com/learn-co-curriculum/dsc-data-analysis-pandas-intro) (https://github.com/learn-co-curriculum/dsc-data-analysis-pandas-intro/issues/new/choose)

Introduction

In this section, you'll be introduced to pandas, one of the most powerful and widely used libraries for data analysis in Python.

Python Libraries

In some cases, such as opening a JSON file to extract a single value, base Python has all of the functionality you need. Other times when the task is more complex, writing your own code to do it can get overwhelming pretty quickly.

Luckily one of the great benefits of the Python language is that it has a very active open-source community, which means tons of great libraries and frameworks we can use to do the heavy lifting. One of the main reasons that Python is such a great choice for data science is that the scientific community has written plenty of great packages for various advanced purposes. This means that when we use Python, we have access to a wealth of robust, effective tools written and maintained by an army of volunteers and professionals.

Pandas

The pandas library will likely be your most-used library over the course of this program. It is a flexible, powerful data analysis library that is especially well-suited to handling tabular data (meaning data that is represented as rows and columns). The name is reminiscent of the term "panel data" as well as "Python data analysis".

Help

eader to Pandas

The central pandas library feature we will use is the **dataframe** (also styled DataFrame). A dataframe represents tabular data with an integrated index, so data can be selected an manipulated using rows or columns. For some tasks, this means that the syntax can be significantly simpler and more efficient than the equivalent base Python syntax.

Let's go over a quick comparison of using pandas vs. the built-in `csv` module.

We'll use this dataset about Olympic track and field medal-winners from [kaggle (https://www.kaggle.com/jayrav13/olympic-track-field-results)](https://www.kaggle.com/jayrav13/olympic-track-field-results). The first five rows look like this:

Gender	Event	Location	Year	Medal	Name	Nationality	Result
M	10000M Men	Rio	2016	G	Mohamed FARAH	GBR	25:05.17
M	10000M Men	Rio	2016	S	Paul Kipngetich TANUI	KEN	27:05.64
M	10000M Men	Rio	2016	B	Tamirat TOLA	ETH	27:06.26
M	10000M Men	Beijing	2008	G	Kenenisa BEKELE	ETH	27:01.17
M	10000M Men	Beijing	2008	S	Sileshi SIHINE	ETH	27:02.77

To open and read the first 5 lines with the `csv` module, that would look like this:

```
import csv

with open("olympic_medals.csv") as f:
    reader = csv.DictReader(f)
    olympics_data = list(reader)

# Print the first 5 rows of data
for index in range(5):
    print(olympics_data[index])
```

```
{'Gender': 'M', 'Event': '10000M Men', 'Location': 'Rio', 'Year': '2016', 'Medal': 'G', 'I
{'Gender': 'M', 'Event': '10000M Men', 'Location': 'Rio', 'Year': '2016', 'Medal': 'S', 'I
{'Gender': 'M', 'Event': '10000M Men', 'Location': 'Rio', 'Year': '2016', 'Medal': 'B', 'I
{'Gender': 'M', 'Event': '10000M Men', 'Location': 'Beijing', 'Year': '2008', 'Medal': 'G
{'Gender': 'M', 'Event': '10000M Men', 'Location': 'Beijing', 'Year': '2008', 'Medal': 'S
```

W Help f dictionaries, where every dictionary has the same keys.

Let's say we want to select all data for the 3rd **row** (record). That's simple enough — we can just use list indexing:

```
olympics_data[2]
```

```
{'Gender': 'M',
 'Event': '10000M Men',
 'Location': 'Rio',
 'Year': '2016',
 'Medal': 'B',
 'Name': 'Tamirat TOLA',
 'Nationality': 'ETH',
 'Result': '27:06.26'}
```

Now, what if we want to select all data for the 3rd **column** (i.e. the values associated with the `'Location'` keys)? That's not impossible, but it will require some kind of loop or list comprehension. Something like this:

```
# Cutting it off at 100 for readability
print([row['Location'] for row in olympics_data][:100])
```

```
['Rio', 'Rio', 'Rio', 'Beijing', 'Beijing', 'Beijing', 'Sydney', 'Sydney', 'Sydney', 'Bar
```

With pandas, accessing columns is just as simple as accessing rows. For example, if we convert `olympics_data` (a list of dictionaries) in a dataframe, then view the first five rows:

```
import pandas as pd

df = pd.DataFrame(olympics_data)
df.head()
```

	Gender	Event	Location	Year	Medal	Name	Nationality	Result
0	M	10000M Men	Rio	2016	G	Mohamed FARAH	GBR	25:05.17
1	M	10000M Men	Rio	2016	S	Paul Kipngetich TANUI	KEN	27:05.64
2	M	10000M Men	Rio	2016	B	Tamirat TOLA	ETH	27:06.26
[illegible]	[illegible]	10000M Men	Beijing	2008	G	Kenenisa BEKELE	ETH	27:01.17

Help

	Gender	Event	Location	Year	Medal	Name	Nationality	Result
4	M	10000M Men	Beijing	2008	S	Sileshi SIHINE	ETH	27:02.77

We just imported the pandas library as `pd` , the standard alias, then used the `DataFrame` constructor to make a dataframe out of our existing list of dictionaries.

Now we can extract all of the information from the 3rd column with a simpler syntax:

```
df['Location']
```

```
0           Rio
1           Rio
2           Rio
3       Beijing
4       Beijing
         ...
2389     Athens
2390     Athens
2391    Atlanta
2392    Atlanta
2393    Atlanta
Name: Location, Length: 2394, dtype: object
```

But it's also very easy to extract information by row, just like it was with the list of dictionaries:

```
df.iloc[2]
```

```
Gender                    M
Event             10000M Men
Location                Rio
Year                   2016
Medal                     B
Name           Tamirat TOLA
Nationality             ETH
Result             27:06.26
Name: 2, dtype: object
```

We can also skip the `csv` module and `olympics_data` variable altogether, and just read the data from the CSV file directly. All you need to do is specify the file path:

```
d[illegible] csv("olympic_medals.csv")
```

Help

	Gender	Event	Location	Year	Medal	Name	Nationality	Result
0	M	10000M Men	Rio	2016	G	Mohamed FARAH	GBR	25:05.17
1	M	10000M Men	Rio	2016	S	Paul Kipngetich TANUI	KEN	27:05.64
2	M	10000M Men	Rio	2016	B	Tamirat TOLA	ETH	27:06.26
3	M	10000M Men	Beijing	2008	G	Kenenisa BEKELE	ETH	27:01.17
4	M	10000M Men	Beijing	2008	S	Sileshi SIHINE	ETH	27:02.77
...	...	...	...	...	...	...	...	...
2389	W	Triple Jump Women	Athens	2004	S	Hrysopiyi DEVETZI	GRE	15.25
2390	W	Triple Jump Women	Athens	2004	B	Tatyana LEBEDEVA	RUS	15.14
2391	W	Triple Jump Women	Atlanta	1996	G	Inessa KRAVETS	UKR	15.33
2392	W	Triple Jump Women	Atlanta	1996	S	Inna LASOVSKAYA	RUS	14.98
2393	W	Triple Jump Women	Atlanta	1996	B	Sarka KASPARKOVA	CZE	14.98

2394 rows × 8 columns

Features of Pandas

The code snippets above demonstrate two of the library highlights from the pandas [about page (https://pandas.pydata.org/about/)](https://pandas.pydata.org/about/) :

- A fast and efficient **DataFrame** object for data manipulation with integrated indexing;
- **ding and writing data between in-memory data structures and different** Help V and text files, Microsoft Excel, SQL databases, and the fast HDF5 format;

Other highlights include:

- Intelligent data alignment and integrated **handling of missing data**: gain automatic label-based alignment in computations and easily manipulate messy data into an orderly form;
- Flexible **reshaping and pivoting** of data sets;
- Intelligent label-based **slicing**, fancy indexing, and **subsetting** of large data sets;
- Aggregating or transforming data with a powerful **group by** engine allowing split-apply-combine operations on data sets;
- High performance **merging and joining** of data sets;
- **Time series**-functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data;
- Highly **optimized for performance**, with critical code paths written in Cython or C.

We will begin to dive into these features in this section!