Data 301
Tabular Data and Pandas

Dennis Sun

April 5, 2016

1 Tabular Data

2 Introduction to Pandas

Tabular Data

Data often comes in "tabular," or spreadsheet-like, form. Tabular data is also called relational data.

Bsal	Sal77	Sex	Senior	Age	Educ	Exper
5220	8340	Female	70	468	12	127
5040	12420	Male	96	329	15	14
6300	12060	Male	82	357	15	72
6000	15120	Male	67	315	15	35.5
6000	16320	Male	97	354	12	24
⋮						⋮

This data is from a sex discrimination lawsuit against Harris Bank of Chicago. It shows salary information for all entry-level employees hired between 1969 and 1971.

What Data Structures Should We Use?

Bsal	Sal77	Sex	Senior	Age	Educ	Exper
5220	8340	Female	70	468	12	127
5040	12420	Male	96	329	15	14
6300	12060	Male	82	357	15	72
6000	15120	Male	67	315	15	35.5
6000	16320	Male	97	354	12	24
⋮						⋮

How about a list of lists / tuples?

```
data = [
  (5220, 8340, "Female", 70, 468, 12, 127),
  (5040, 12420, "Male", 96, 329, 15, 14),
  (6300, 12060, "Male", 82, 357, 15, 72),
  ...
]
```

Advantage: easy to access rows (e.g., `data[2]`)
Disadvantage: difficult to access columns, columns unlabeled, no way to enforce consistency between rows

Numpy Arrays

Numpy is a Python package that provides an n-dimensional array structure, along with methods to manipulate them efficiently.

```
import numpy as np
data = np.array([
  (5220, 8340, "Female", 70, 468, 12, 127),
  (5040, 12420, "Male", 96, 329, 15, 14),
  (6300, 12060, "Male", 82, 357, 15, 72),
  ...
])
```

`data[2]` still returns row 3. But now we can also access column 3 using `data[:, 2]`. You can also pass in a slice, like `1:4`, or a boolean array that indicates which rows you want.

Try it!

- How would you get the 1977 salary for the third employee?
- How would you get a data set with only the male employees?

Numpy Arrays

Do you notice any problems with the way Numpy stored this array?

All the entries are strings, even the numbers! Can you modify this data frame so that it stores the numerical entries as ints or floats?

Numpy will try to store all elements of an array as the same type because this is most efficient. But a data set may have columns with different types.

Variable Types

I don't mean data types (e.g., int, float, str, etc.)!

We will classify the variables into two main types:

- **Categorical:** the possible values of the variable are categories.
- **Quantitative:** the possible values of the variable are numbers.

What are the types of the variables in the Harris Bank data?

Bsal	Sal77	Sex	Senior	Age	Educ	Exper
5220	8340	Female	70	468	12	127
5040	12420	Male	96	329	15	14
6300	12060	Male	82	357	15	72
6000	15120	Male	67	315	15	35.5
6000	16320	Male	97	354	12	24
⋮						⋮

Structured Arrays

Numpy provides structured arrays that allow different columns to have different types. Structured arrays also allow us to set indexes (i.e., names) for columns.

```
data = np.array([
  (5220, 8340, "Female", 70, 468, 12, 127),
  (5040, 12420, "Male", 96, 329, 15, 14),
  (6300, 12060, "Male", 82, 357, 15, 72),
  ...
], dtype={
  "names": ('Bsal', 'Sal77', 'Sex', 'Senior',
            'Age', 'Educ', 'Exper'),
  "formats": ('i4', 'i4', 'S8', 'i4', 'i4', 'S2', 'f8')
})
```

Now notice that you can access columns as you would a dict, e.g., `data['Sal77']`.

Check that the columns have the right types.

Vectorized Computations

Computations in Numpy are **vectorized**. What does this mean? Let's learn by example.

Suppose we have two lists `a = [1, 2, 5]` and `b = [2, 4, 6]`, and we want to add them together elementwise. (In other words, we want `[3, 6, 11]`.) How would you write code to do this?

If we think of `a` and `b` as vectors, then the elementwise sum is just `a + b`. But this code doesn't work! You have to do something like:

```
[i + j for i, j in zip(a, b)]
```

But if `a` and `b` are Numpy arrays instead of lists, then `a + b` works!

```
a = np.array([1, 2, 5])
b = np.array([2, 4, 6])
a + b
```

Try it! For the Harris Bank data, write code that computes for each employee how much his/her salary increased from when he/she started.

Vectorized Computations

Another Example: How would you square every element in:

- `a = [1, 2, 5]`?

```
[i**2 for i in a]
```

- `a = np.array([1, 2, 5])`?

```
a ** 2
```

Try it! Because salaries are often right-skewed, we often log-transform the salaries. Write code that computes the log 1977 salary of each employee.

Use vectorized computations wherever possible. They are not only prettier, but they are also faster!

Are Numpy Structured Arrays All We Need?

With Numpy's structured arrays, we can now:

- index by row
- index by column
- give names to columns (i.e., set column indexes)
- store different data types in each column
- use vectorized operations

However, structured arrays are not perfect:

- They do not allow us to set the row indexes.
- They do not infer the types automatically.
- It is difficult to add new columns to the data.

This is where another library, Pandas, comes in handy!

1 Tabular Data

2 Introduction to Pandas

What is Pandas?

Pandas stands for "Panel data" or "Python data analysis".

Pandas provides a DataFrame structure that allows both rows and columns to be indexed:

	Bsal	Sal77	Sex	Senior	Age	Educ	Exper
0	5220	8340	Female	70	468	12	127
1	5040	12420	Male	96	329	15	14
2	6300	12060	Male	82	357	15	72
3	6000	15120	Male	67	315	15	35.5
4	6000	16320	Male	97	354	12	24
	⋮						⋮

Pandas is built on top of Numpy, so you can take advantage of vectorization and all the other Numpy goodies.

It also provides functions that reads data from a file and infers the type of each column intelligently.

Wait, doesn't R's `data.frame` do all the same things? Yes, Pandas brings the elegance of R's **`data.frame`** to Python.

Reading Data Into a Pandas DataFrame

```
import pandas as pd
data = pd.read_csv("/data/harris.csv")
```

Try it!

1. Try `data.head()` and `data.tail()`. What do these do?
2. Try `data.describe()`. Notice anything interesting?
3. Try `data['Age']`. What is the data structure?
4. Try `data[['Sex', 'Age']]`. What is the data structure?
5. Suppose you wanted a DataFrame with only one column, `Age`. How would you do that?
6. Try describing the series `data['Age']` and `data['Sex']`. What do you notice?

We can (usually) also access columns using attribute notation `data.Age` instead of dictionary notation `data['Age']`.

More About Series

Like Numpy arrays, Pandas Series support vectorized computations.

Unlike Numpy arrays, it is easy to create new columns in Pandas DataFrame. Simply assign a series to a new key in the DataFrame, as if it were a dict:

```
data['AgeInYears'] = data['Age'] / 12
```

Try it! Create a new column, `SalaryIncrease` that shows how much each employee's salary increased between when he/she started and 1977.

Indexing Rows

We've seen how to access columns in a DataFrame. How about rows?

Does `data[2]` work, like it did with Numpy?

No, we have to be explicit that we are accessing rows instead of columns. Pandas' way of being explicit about this is the `.ix` method of the data frame. Try `data.ix[2]`.

Why do you think Pandas was designed this way?

You can also use `.ix` to index both rows and columns, e.g., `data.ix[2, "Age"]` or `data.ix[:, "Age"]`.

You can access (and modify) the column indexes using `data.columns` and the row indexes using `data.index`. Try changing the row index to go from 1 to 93, instead of 0 to 92. Does this change `data.ix[2, "Age"]`?

Analyzing the Harris Data

In-Class Exercise

Compute a 95% confidence interval for the average beginning salary of male employees. Then, compute a 95% confidence interval for the beginning salary of female employees. What do you conclude?

Remember, a 95% confidence interval is defined as:

$$\overline{X} \pm t_{.975,n-1} \frac{\sigma}{\sqrt{n}}.$$

You can look up quantiles of the t (and other distributions) in the Scipy package. Scipy contains many useful functions for statistics and scientific computing. Here's some code to get you started:

```
from scipy.stats import t
```

Missing Data and String Handling

Pandas is also very good at dealing with missing data and handling strings, which are necessary in today's messy data world. You'll explore these features in the lab for today.