Numbers. And Pythons.

A couple digressions

- iPython interpreter

A couple digressions

- Try/Except

A couple digressions

- Try/Except
    - How can I check input for errors without halting my script?

A couple digressions

- Try/Except
    - How can I check input for errors without halting my script?

```
with open('homework.csv') as f:
    line_list = [line.strip().split(',') for line in f]
    print line_list
```

A couple digressions

```
for line in line_list[1:]:
    try:
        int(line[2])
        total += float(line[2])
    except:
        print "not a number, skipping", line[2]
```

A couple digressions

```
for line in line_list[1:]:
    try:        #Attempt the below operation
        int(line[2])
        total += float(line[2])
    except: #If operation is impossible, do below
        print "not a number, skipping", line[2]
```

Handling big data

- Big data is a total buzz word

Handling big data

- Big data is a total buzz word
- But many of our basic python approaches don’t scale well

Numpy, Scipy and Pandas

- **Numpy:** Primarily for large amounts of mathematical calculations
- **Scipy:** Many statistical functions of interest to scientists
- **Pandas:** Built on Numpy, a library contextualizing Numpy functions more helpfully

Pandas

- Pandas allows you to access key pieces of Numpy functionality
- While retaining the user-friendliness of python
    - Excel support
    - Row and column names

Loading in Data

```
xl = pandas.ExcelFile('spreadsheet.xls')

no_xl = pandas.read_csv('homework.csv')
```

Loading in Data

```
xl = pandas.ExcelFile('spreadsheet.xls')
```

```
no_xl = pandas.read_csv('homework.csv')
```

Familiar: We are creating a file object, not interacting with the data.

Pandas DataFrame objects

- 2D
- Labelled!
- Name:entry pairs
- read_csv imports as DataFrame

Pandas DataFrame objects

- 2D
- Labelled!
- Name:entry pairs

Pandas DataFrame objects

- We have to coerce Python to do read in Excel appropriately

```
 xl.sheet_names
df = xl.parse('Sheet1', index_col=0)
```

What’s cool about DataFrames?

- Slicing and dicing
- Viewing data
- Finding object types

Viewing your data

```
df.head()


#Show the first five entries
```

Viewing your data

```
df.head()

#Show the first five entries

df.tail()
#Show the last 5
```

Data Types

- Remember when we checked if each entry in a column was the right type?
- How do you know what type to expect?

Data Types

```
df.index

#Tells you what the different row names and their types are
```

Data Types

```
df.describe()

#Get a quick look at the stats of our numeric columns
```

Data Types

```
df.dtypes

#Return a list of the data types of each column
```

Data Types

```
df.ix[:,'Observations']

#Return the column in question and its type
```

Data Types

- These two are odd, yes?
- The first returns that the dtype is “object”
    - Porque?

Data Types

- These two are odd, yes?
- The first returns that the dtype is “object”
  - Porque?
  - Mixed type - string + int

Data Types

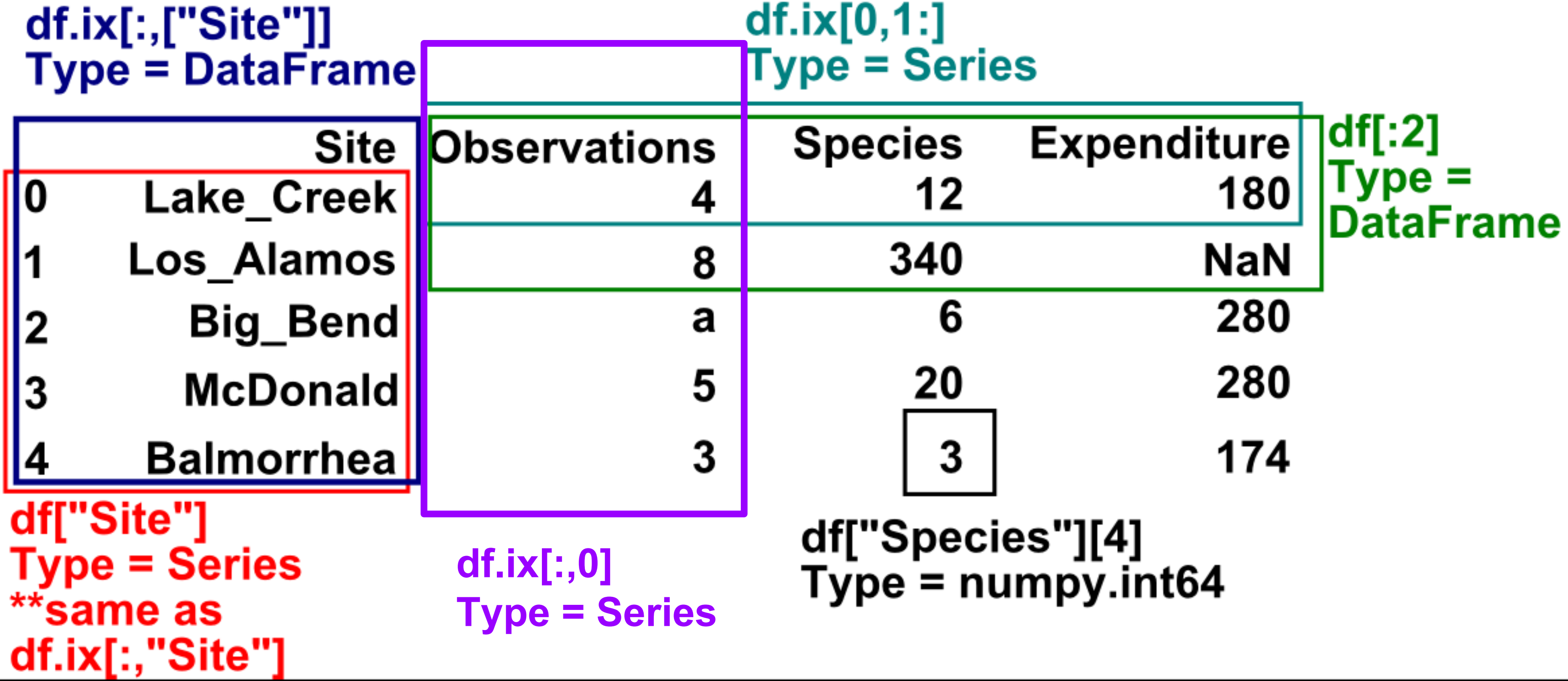
- These two are odd, yes?
- The first returns that the dtype is “object”
  - Porque?
  - Mixed type - string + int
- But what is ‘int64’?

Data Types

- These two are odd, yes?
- The first returns that the dtype is “object”
  - Porque?
  - Mixed type - string + int
- But what is ‘int64’?
- And why is ‘Expenditure’ a float64 with missing data?

More slicing

```
df.ix[1,['Observations','Species']]
```



	Site	Observations	Species	Expenditure
0	Lake_Creek	4	12	180
1	Los_Alamos	8	340	NaN
2	Big_Bend	a	6	280
3	McDonald	5	20	280
4	Balmorrhea	3	3	174

df.ix[:,["Site"]]
Type = DataFrame

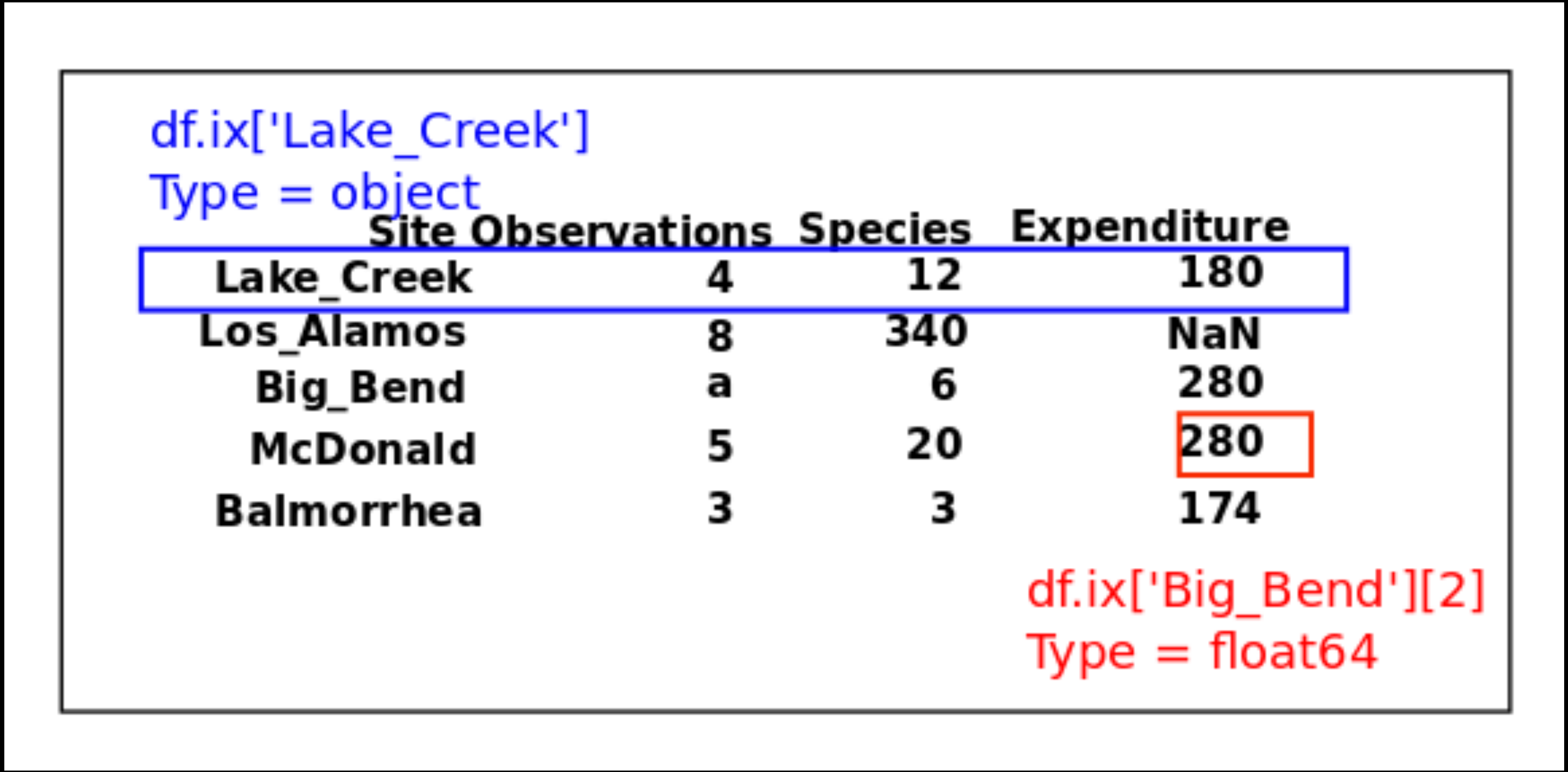
df.ix[0,1:]
Type = Series

df[:2]
Type = DataFrame

df["Site"]
Type = Series
**same as
df.ix[:,"Site"]

df.ix[:,0]
Type = Series

df["Species"][4]
Type = numpy.int64



df.ix['Lake_Creek']
Type = object

```
                Site Observations  Species  Expenditure
  Lake_Creek                    4       12          180
 Los_Alamos                     8      340          NaN
    Big_Bend                    a        6          280
    McDonald                    5       20          280
  Balmorrhea                    3        3          174
```

df.ix['Big_Bend'][2]
Type = float64

Coordinate Slicing

- df[start:stop]

Take five. Try slicing your data in different ways.

Try assigning different slices to variables and doing math or error checking with them

- Also, try sum() or df.mean() with a list or series

Nota Bene

- That’s all a bit tough
- And really the only way to get good at that sort of indexing is to practice

The Pandas-Numpy interface

- Numpy has a lot of really smart numerical functions.
- But the interface that makes those operations possible also makes interaction hard

The Pandas-Numpy interface

```
b = df.ix[:,'Expenditure'] #bind column to b
numpy.unique(b) #Distill rapidly to unique values
```

Only works with numbers!

The NaN

- NaN is “Not a Number”
- This is a formulation that *does not* store your NaN value
- These are stored as boolean values

NaN methods

```
for x in b:
    if numpy.isnan(x):
        print "This ain't a number!"

#Real quick test if something is NaN
```

NaN Methods

```
df.fillna(0)
#Fill missing values with a zero
```

NaN Methods

```
df.fillna(df.mean())
#Fill missing values with the mean of the row
```

NaN Methods

```
df['Observations'].convert_objects
(convert_numeric=True)
#Change non-numeric characters to NaN
df.fillna(df.mean())
# And fill them with the average
```

Random Subsets

```
import random
cols = random.sample(df.columns, 3)
#Get 3 random columns
rando_df =  df.ix[:, cols]
#Extract all rows associated with those columns
```

```
a = np.random.randint(5, size=(85, 300))
#create a 85 row by 300 column data set made up of random values between 0 and 5
a[1:10,1:3]
#Since we don’t have labels, we index by location
```

Output

DataFrame.to_csv(filename)

Pandas has *many* fancy output options. See here for more:

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_csv.html

Challenge Problem

Using the random matrix we just made, the commands we’ve learned and the table on the wiki, create a short pipeline to do some subsetting and a mathematical operation

Docs

Pandas Documentation

Some Stats in Scipy

Numpy Tutorial made by the Scipy people

Example

R, Python and Climatology

A side-by-side comparison for data crunching in R and Python. Helpful example!

Again …

These libraries are *huge!*

And kind of grab-baggish

Really, the best way to learn is practice and to look in pandas, scipy and numpy and see if the function you want is there.

Homework

- On your own data:
    - Make three different subsets of the data
    - Replace missing values in two ways
    - Do a little math on your subsets with replaced data
    - What are the strengths and weaknesses of each way you replaced missing data?
        - Save subsets with replacements in a file and send to us

Homework

- On a random matrix:
  - Try all the operations that you did with your own personal data.
  - Do they all work? If not, why? Google your error messages or check with us about why.

Homework

Lastly:

https://docs.google.com/spreadsheet/ccc?key=0And3uqWBFJNxdFlYeVR5NHNRdGRtZlBfQ2VYc1lNNVE&usp=sharing

Add some nonsense to this spreadsheet!

Homework

Fill this spreadsheet with garbage:

Spreadsheet