Advanced Data Management (CSCI 490/680)

Data & Pandas

Dr. David Koop
Arrays

- Usually a fixed size—lists are meant to change size
- Are mutable—tuples are not
- Store only one type of data—lists and tuples can store anything
- Are faster to access and manipulate than lists or tuples
- Can be multidimensional:
  - Can have list of lists or tuple of tuples but no guarantee on shape
  - Multidimensional arrays are rectangles, cubes, etc.
Why NumPy?

- Fast **vectorized** array operations for data munging and cleaning, subsetting and filtering, transformation, and any other kinds of computations
- Common array algorithms like sorting, unique, and set operations
- Efficient descriptive statistics and aggregating/summarizing data
- Data alignment and relational data manipulations for merging and joining together heterogeneous data sets
- Expressing conditional logic as array expressions instead of loops with `if-elif-else` branches
- Group-wise data manipulations (aggregation, transformation, function application).

[W. McKinney, Python for Data Analysis]
NumPy Arrays

- `data1 = [6, 7.5, 8, 0, 1]`
  `arr1 = np.array(data1)`

- **Zeros**: `np.zeros(10)`, **Ones**: `np.ones((4,5))`, **Empty**: `np.empty((2,2))`

- # of dimensions: `arr2.ndim`, **Shape**: `arr2.shape`, **Type**: `arr2.dtype`

- Types: Each array has a fixed type unlike other variables in python
2D Array Slicing

Suppose each name corresponds to a row in the data array and we wanted to select all the rows with corresponding name 'Bob'. Like arithmetic operations, comparisons (such as ==) with arrays are also vectorized. Thus, comparing names with the string 'Bob' yields a boolean array:

In [87]: names == 'Bob'
Out[87]: array([ True, False, False, True, False, False, False], dtype=bool)

This boolean array can be passed when indexing the array:

In [88]: data[names == 'Bob']
Out[88]:

array([[ -0.048 ,  0.5433, -0.2349,  1.2792],
       [ 2.1452,  0.8799, -0.0523,  0.0672]])

The boolean array must be of the same length as the axis it's indexing. You can even mix and match boolean arrays with slices or integers (or sequences of integers, more on this later):

In [89]: data[names == 'Bob', 2:]
Out[89]:

array([[-0.2349,  1.2792]])

How to obtain the blue slice from array arr?

<table>
<thead>
<tr>
<th>Expression</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>arr[:2, 1:]</td>
<td>(2, 2)</td>
</tr>
</tbody>
</table>
2D Array Slicing

Figure 4-2. Two-dimensional array slicing

Suppose each name corresponds to a row in the data array and we wanted to select all the rows with corresponding name 'Bob'. Like arithmetic operations, comparisons (such as ==) with arrays are also vectorized. Thus, comparing names with the string 'Bob' yields a boolean array:

In [87]: names == 'Bob'
Out[87]: array([ True, False, False, True, False, False, False], dtype=bool)

This boolean array can be passed when indexing the array:

In [88]: data[names == 'Bob']
Out[88]:
array([[-0.048 ,  0.5433, -0.2349,  1.2792],
       [ 2.1452,  0.8799, -0.0523,  0.0672]])

The boolean array must be of the same length as the axis it's indexing. You can even mix and match boolean arrays with slices or integers (or sequences of integers, more on this later):

In [89]: data[names == 'Bob', 2:]
Out[89]:
array([[-0.2349,  1.2792]])

How to obtain the blue slice from array arr?

Expression  Shape
arr[:2, 1:]  (2, 2)

[W. McKinney, Python for Data Analysis]
2D Array Slicing

Suppose each name corresponds to a row in the data array and we wanted to select all the rows with corresponding name 'Bob'. Like arithmetic operations, comparisons (such as ==) with arrays are also vectorized. Thus, comparing names with the string 'Bob' yields a boolean array:

```
In [87]: names == 'Bob'
Out[87]: array([ True, False, False, True, False, False, False], dtype=bool)
```

This boolean array can be passed when indexing the array:

```
In [88]: data[names == 'Bob']
Out[88]:
array(
    [[-0.048,  0.5433, -0.2349,  1.2792],
     [ 2.1452,  0.8799, -0.0523,  0.0672]],
  dtype=float64)
```

The boolean array must be of the same length as the axis it's indexing. You can even mix and match boolean arrays with slices or integers (or sequences of integers, more on this later):

```
In [89]: data[names == 'Bob', 2:]
Out[89]:
array(
    [[-0.2349,  1.2792]],
  dtype=float64)
```

How to obtain the blue slice from array \texttt{arr}?

<table>
<thead>
<tr>
<th>Expression</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{arr[:2, 1:]}, \texttt{arr[2]}, \texttt{arr[2, :]}, \texttt{arr[2:, :]}</td>
<td>(2, 2), (3,), (3,), (1, 3)</td>
</tr>
</tbody>
</table>

[W. McKinney, *Python for Data Analysis*]
Boolean Indexing

- names == 'Bob' gives back booleans that represent the element-wise comparison with the array names.

- Boolean arrays can be used to index into another array:
  - data[names == 'Bob']

- Can even mix and match with integer slicing.

- Can do boolean operations (\&, |) between arrays (just like addition, subtraction):
  - data[(names == 'Bob') | (names == 'Will')]

- Note: or and and do not work with arrays.

- We can set values too! data[data < 0] = 0
Assignment 1

- Due Monday, Feb. 1 at 11:59pm
- Using Python for data analysis on Info Wanted ads
- Provided a1.ipynb file (right-click and download)
- Use basic python for now to demonstrate language knowledge
  - No pandas (for now)
- Use Anaconda or hosted Python environment
- Turn .ipynb file in via Blackboard
- Notes:
  - Bug in URL (https instead of http),
  - Bug in Problem 1 solution
Other Operations

- Fancy Indexing: `arr[[1, 2, 3]]`
- Transposing arrays: `arr.T`
- Reshaping arrays: `arr.reshape((3, 5))`
- Unary universal functions (ufuncs): `np.sqrt`, `np.exp`
- Binary universal functions: `np.add`, `np.maximum`
Unary Universal Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs, fabs</td>
<td>Compute the absolute value element-wise for integer, floating-point, or complex values</td>
</tr>
<tr>
<td>sqrt</td>
<td>Compute the square root of each element (equivalent to arr ** 0.5)</td>
</tr>
<tr>
<td>square</td>
<td>Compute the square of each element (equivalent to arr ** 2)</td>
</tr>
<tr>
<td>exp</td>
<td>Compute the exponent e(^x) of each element</td>
</tr>
<tr>
<td>log, log10,</td>
<td>Natural logarithm (base e), log base 10, log base 2, and log(1 + x), respectively</td>
</tr>
<tr>
<td>log2, log1p</td>
<td></td>
</tr>
<tr>
<td>sign</td>
<td>Compute the sign of each element: 1 (positive), 0 (zero), or –1 (negative)</td>
</tr>
<tr>
<td>ceil</td>
<td>Compute the ceiling of each element (i.e., the smallest integer greater than or equal to that number)</td>
</tr>
<tr>
<td>floor</td>
<td>Compute the floor of each element (i.e., the largest integer less than or equal to each element)</td>
</tr>
<tr>
<td>rint</td>
<td>Round elements to the nearest integer, preserving the dtype</td>
</tr>
<tr>
<td>modf</td>
<td>Return fractional and integral parts of array as a separate array</td>
</tr>
<tr>
<td>isnan</td>
<td>Return boolean array indicating whether each value is NaN (Not a Number)</td>
</tr>
<tr>
<td>isfinite, isinf</td>
<td>Return boolean array indicating whether each element is finite (non-inf, non-NaN) or infinite, respectively</td>
</tr>
<tr>
<td>cos, cosh, sin, sinh, tan, tanh</td>
<td>Regular and hyperbolic trigonometric functions</td>
</tr>
<tr>
<td>arccos, arccosh, arcsin, arccsinh, arctan, arctanh</td>
<td>Inverse trigonometric functions</td>
</tr>
<tr>
<td>logical_not</td>
<td>Compute truth value of not ( x ) element-wise (equivalent to (-arr)).</td>
</tr>
</tbody>
</table>

Table 4-3. Unary ufuncs

Table 4-4. Binary universal functions
Now, evaluating the function is a matter of writing the same expression you would replace explicit loops with array expressions is commonly referred to as powerful method for vectorizing computations.

As a simple example, suppose we wished to evaluate the function replacing explicit loops with array expressions is commonly referred to as

In general, vectorized array operations will often be one or two (or more) orders

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out

In

Out
Here, \( \text{arr.mean(1)} \) means "compute mean across the columns" where \( \text{arr.sum(0)} \) means "compute sum down the rows." Other methods like \( \text{cumsum} \) and \( \text{cumprod} \) do not aggregate, instead producing an array of the intermediate results:

In \[\text{184}\]:
\[
\text{arr} = \text{np.array([0, 1, 2, 3, 4, 5, 6, 7])}
\]

In \[\text{185}\]:
\[
\text{arr.cumsum}()
\]

Out \[\text{185}\]:
\[
\text{array([0, 1, 3, 6, 10, 15, 21, 28])}
\]

In multidimensional arrays, accumulation functions like \( \text{cumsum} \) return an array of the same size, but with the partial aggregates computed along the indicated axis according to each lower dimensional slice:

In \[\text{186}\]:
\[
\text{arr} = \text{np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])}
\]

In \[\text{187}\]:
\[
\text{arr.cumsum}()
\]

Out \[\text{187}\]:
\[
\text{array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])}
\]

In \[\text{188}\]:
\[
\text{arr.cumsum}() \text{ axis}=0
\]

Out \[\text{188}\]:
\[
\text{array([[0, 1, 2], [3, 5, 7], [6, 12, 15]])}
\]

In \[\text{189}\]:
\[
\text{arr.cumprod}() \text{ axis}=1
\]

Out \[\text{189}\]:
\[
\text{array([[0, 0, 0], [3, 12, 60], [6, 42, 336]])}
\]

See Table 4-5 for a full listing. We'll see many examples of these methods in action in later chapters.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>Sum of all the elements in the array or along an axis; zero-length arrays have sum 0</td>
</tr>
<tr>
<td>mean</td>
<td>Arithmetic mean; zero-length arrays have NaN mean</td>
</tr>
<tr>
<td>std, var</td>
<td>Standard deviation and variance, respectively, with optional degrees of freedom adjustment (default denominator n)</td>
</tr>
<tr>
<td>min, max</td>
<td>Minimum and maximum</td>
</tr>
<tr>
<td>argmin, argmax</td>
<td>Indices of minimum and maximum elements, respectively</td>
</tr>
<tr>
<td>cumsum</td>
<td>Cumulative sum of elements starting from 0</td>
</tr>
<tr>
<td>cumprod</td>
<td>Cumulative product of elements starting from 1</td>
</tr>
</tbody>
</table>
More

- Other methods:
  - any and all
  - sort
  - unique
- Linear Algebra (numpy.linalg)
- Pseudorandom Number Generation (numpy.random)
Data

• What is data?
  - Types
  - Semantics
• How is data structured?
  - Tables (Data Frames)
  - Databases
  - Data Cubes
• What formats is data stored in?
• Raw versus derived data
Data

• What is this data?

<table>
<thead>
<tr>
<th></th>
<th>42ND STREET &amp; 8TH AVENUE</th>
<th>00228985</th>
<th>00008471</th>
<th>00000441</th>
<th>00001455</th>
<th>00000134</th>
<th>00033341</th>
<th>00071255</th>
</tr>
</thead>
<tbody>
<tr>
<td>R011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R170</td>
<td>14TH STREET-UNION SQUARE</td>
<td>00224603</td>
<td>00011051</td>
<td>00000827</td>
<td>00003026</td>
<td>00000660</td>
<td>00089367</td>
<td>00199841</td>
</tr>
<tr>
<td>R046</td>
<td>42ND STREET &amp; GRAND CENTRAL</td>
<td>00207758</td>
<td>00007908</td>
<td>00000323</td>
<td>00001183</td>
<td>00003001</td>
<td>00040759</td>
<td>00096613</td>
</tr>
</tbody>
</table>

• Semantics: real-world meaning of the data
• Type: structural or mathematical interpretation
• Both often require metadata
  - Sometimes we can infer some of this information
  - Line between data and metadata isn’t always clear
## Data

<table>
<thead>
<tr>
<th>REMOTE</th>
<th>STATION</th>
<th>FF</th>
<th>Y</th>
<th>SEN/DIX</th>
<th>7-D AFAS UNL</th>
<th>D AFAS/RMF</th>
<th>JOINT RR TKT</th>
<th>7-D UNL</th>
<th>30-D UNL</th>
</tr>
</thead>
<tbody>
<tr>
<td>R011</td>
<td>42ND STREET &amp; 8TH AVENUE</td>
<td>00228985</td>
<td>00008471</td>
<td>00000441</td>
<td>00001455</td>
<td>00000134</td>
<td>00033341</td>
<td>00071255</td>
<td></td>
</tr>
<tr>
<td>R170</td>
<td>14TH STREET-UNION SQUARE</td>
<td>00224603</td>
<td>00011051</td>
<td>00000627</td>
<td>00003026</td>
<td>00000660</td>
<td>00089367</td>
<td>00199841</td>
<td></td>
</tr>
<tr>
<td>R046</td>
<td>42ND STREET &amp; GRAND CENTRAL</td>
<td>00207758</td>
<td>00007908</td>
<td>00000323</td>
<td>0001183</td>
<td>00003001</td>
<td>00040759</td>
<td>00096613</td>
<td></td>
</tr>
<tr>
<td>R012</td>
<td>34TH STREET &amp; 8TH AVENUE</td>
<td>00188811</td>
<td>00006490</td>
<td>00000498</td>
<td>0001279</td>
<td>00003622</td>
<td>00035527</td>
<td>00067483</td>
<td></td>
</tr>
<tr>
<td>R293</td>
<td>34TH STREET - PENN STATION</td>
<td>00168768</td>
<td>00006155</td>
<td>00000523</td>
<td>0001065</td>
<td>00005031</td>
<td>00030645</td>
<td>00054376</td>
<td></td>
</tr>
<tr>
<td>R033</td>
<td>42ND STREET/TIMES SQUARE</td>
<td>00159322</td>
<td>00005743</td>
<td>00000317</td>
<td>0001205</td>
<td>00006900</td>
<td>00058931</td>
<td>00078644</td>
<td></td>
</tr>
<tr>
<td>R022</td>
<td>34TH STREET &amp; 6TH AVENUE</td>
<td>00156008</td>
<td>00006276</td>
<td>00000487</td>
<td>0001543</td>
<td>00000712</td>
<td>00058910</td>
<td>00110466</td>
<td></td>
</tr>
<tr>
<td>R084</td>
<td>59TH STREET/COLUMBUS CIRCLE</td>
<td>00155262</td>
<td>00000948</td>
<td>00000589</td>
<td>0002071</td>
<td>00000542</td>
<td>00053397</td>
<td>00113966</td>
<td></td>
</tr>
<tr>
<td>R020</td>
<td>47-50 STREETS/ROCKEFELLER</td>
<td>00143500</td>
<td>00006402</td>
<td>00000384</td>
<td>0001159</td>
<td>00000723</td>
<td>00037928</td>
<td>00097045</td>
<td></td>
</tr>
<tr>
<td>R179</td>
<td>86TH STREET-LEXINGTON AVE</td>
<td>00142169</td>
<td>00010367</td>
<td>00000470</td>
<td>0001839</td>
<td>00000271</td>
<td>00050328</td>
<td>00125250</td>
<td></td>
</tr>
<tr>
<td>R023</td>
<td>34TH STREET &amp; 6TH AVENUE</td>
<td>00134052</td>
<td>00005005</td>
<td>00000348</td>
<td>0001112</td>
<td>00000649</td>
<td>00031531</td>
<td>00075040</td>
<td></td>
</tr>
<tr>
<td>R029</td>
<td>PARK PLACE</td>
<td>00121614</td>
<td>00004311</td>
<td>00000287</td>
<td>0000931</td>
<td>00000792</td>
<td>00025404</td>
<td>00065362</td>
<td></td>
</tr>
<tr>
<td>R047</td>
<td>42ND STREET &amp; GRAND CENTRAL</td>
<td>00100742</td>
<td>00004273</td>
<td>00000185</td>
<td>0000704</td>
<td>00001241</td>
<td>00022808</td>
<td>00068216</td>
<td></td>
</tr>
<tr>
<td>R031</td>
<td>34TH STREET &amp; 7TH AVENUE</td>
<td>00095076</td>
<td>00003990</td>
<td>00000232</td>
<td>0000727</td>
<td>00001459</td>
<td>00024284</td>
<td>00038671</td>
<td></td>
</tr>
<tr>
<td>R017</td>
<td>LEXINGTON AVENUE</td>
<td>00094655</td>
<td>00004688</td>
<td>00000190</td>
<td>0000833</td>
<td>00000754</td>
<td>00020018</td>
<td>00055066</td>
<td></td>
</tr>
<tr>
<td>R175</td>
<td>8TH AVENUE-14TH STREET</td>
<td>00094313</td>
<td>00003907</td>
<td>00000286</td>
<td>0001144</td>
<td>00000256</td>
<td>00038272</td>
<td>00074661</td>
<td></td>
</tr>
<tr>
<td>R057</td>
<td>BARCLAYS CENTER</td>
<td>00093804</td>
<td>00004204</td>
<td>00000454</td>
<td>0001386</td>
<td>00001491</td>
<td>00030113</td>
<td>00068119</td>
<td></td>
</tr>
<tr>
<td>R138</td>
<td>WEST 4TH ST-WASHINGTON SO</td>
<td>00093562</td>
<td>00004677</td>
<td>00000251</td>
<td>0000965</td>
<td>00000127</td>
<td>00031628</td>
<td>00074458</td>
<td></td>
</tr>
</tbody>
</table>
Dataset Types

→ Tables

Attributes (columns)
Items (rows)
Cell containing value

→ Networks

Attributes (columns)
Node (item)
Link

→ Fields (Continuous)

Attributes (columns)
Grid of positions
Cell
Value in cell

→ Geometry (Spatial)

Position

→ Multidimensional Table

Key 1
Key 2
Attributes
Value in cell

→ Trees

Value in cell

[Munzner (ill. Maguire), 2014]
Data Terminology

• Items
  - An **item** is an individual discrete entity
  - e.g., a row in a table

• Attributes
  - An **attribute** is some specific property that can be measured, observed, or logged
  - a.k.a. variable, (data) dimension
  - e.g., a column in a table
# Tables

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S</th>
<th>T</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Order ID</td>
<td>Order Date</td>
<td>Order Priority</td>
<td>Product Container</td>
<td>Product Base Margin</td>
</tr>
<tr>
<td>3</td>
<td>10/14/06</td>
<td>S-Low</td>
<td>Large Container</td>
<td>0.8</td>
<td>10/21/06</td>
</tr>
<tr>
<td>6</td>
<td>2/21/08</td>
<td>S-Not Specified</td>
<td>Small Box</td>
<td>0.55</td>
<td>2/22/08</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Small Pack</td>
<td>0.79</td>
<td>7/17/07</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Jumbo Box</td>
<td>0.79</td>
<td>7/17/07</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.00</td>
<td>7/17/07</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.00</td>
<td>7/17/07</td>
</tr>
<tr>
<td>35</td>
<td>10/23/07</td>
<td>S-Not Specified</td>
<td>Wrap Bag</td>
<td>0.52</td>
<td>10/24/07</td>
</tr>
<tr>
<td>35</td>
<td>10/23/07</td>
<td>S-Not Specified</td>
<td>Small Box</td>
<td>0.58</td>
<td>10/25/07</td>
</tr>
<tr>
<td>36</td>
<td>11/3/07</td>
<td>1-Urgent</td>
<td>Small Box</td>
<td>0.55</td>
<td>11/3/07</td>
</tr>
<tr>
<td>65</td>
<td>3/18/07</td>
<td>1-Urgent</td>
<td>Small Pack</td>
<td>0.49</td>
<td>3/19/07</td>
</tr>
<tr>
<td>66</td>
<td>1/20/05</td>
<td>S-Low</td>
<td>Wrap Bag</td>
<td>0.56</td>
<td>1/20/05</td>
</tr>
<tr>
<td>69</td>
<td>S-Not Specified</td>
<td>S-Not Specified</td>
<td>Small Pack</td>
<td>0.56</td>
<td>6/6/05</td>
</tr>
<tr>
<td>69</td>
<td>S-Not Specified</td>
<td>S-Not Specified</td>
<td>Wrap Bag</td>
<td>0.56</td>
<td>6/6/05</td>
</tr>
<tr>
<td>70</td>
<td>12/18/06</td>
<td>S-Low</td>
<td>Small Box</td>
<td>0.59</td>
<td>12/23/06</td>
</tr>
<tr>
<td>70</td>
<td>12/18/06</td>
<td>S-Low</td>
<td>Wrap Bag</td>
<td>0.82</td>
<td>12/23/06</td>
</tr>
<tr>
<td>96</td>
<td>4/17/05</td>
<td>2-High</td>
<td>Small Box</td>
<td>0.55</td>
<td>4/19/05</td>
</tr>
<tr>
<td>97</td>
<td>1/29/06</td>
<td>3-Medium</td>
<td>Small Box</td>
<td>0.38</td>
<td>1/30/06</td>
</tr>
<tr>
<td>129</td>
<td>11/19/08</td>
<td>5-Low</td>
<td>Small Box</td>
<td>0.37</td>
<td>11/28/08</td>
</tr>
<tr>
<td>130</td>
<td>5/8/08</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.37</td>
<td>5/9/08</td>
</tr>
<tr>
<td>130</td>
<td>5/8/08</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.38</td>
<td>5/10/08</td>
</tr>
<tr>
<td>130</td>
<td>5/8/08</td>
<td>2-High</td>
<td>Small Box</td>
<td>0.6</td>
<td>5/11/08</td>
</tr>
<tr>
<td>132</td>
<td>6/1/06</td>
<td>3-Medium</td>
<td>Medium Box</td>
<td>0.6</td>
<td>6/12/06</td>
</tr>
<tr>
<td>132</td>
<td>6/1/06</td>
<td>3-Medium</td>
<td>Jumbo Box</td>
<td>0.69</td>
<td>6/14/06</td>
</tr>
<tr>
<td>134</td>
<td>5/1/08</td>
<td>4-Not Specified</td>
<td>Large Box</td>
<td>0.82</td>
<td>5/3/08</td>
</tr>
<tr>
<td>134</td>
<td>5/1/08</td>
<td>4-Not Specified</td>
<td>Small Pack</td>
<td>0.64</td>
<td>10/23/07</td>
</tr>
<tr>
<td>166</td>
<td>9/12/07</td>
<td>2-High</td>
<td>Small Box</td>
<td>0.55</td>
<td>9/14/07</td>
</tr>
<tr>
<td>193</td>
<td>8/8/06</td>
<td>1-Urgent</td>
<td>Medium Box</td>
<td>0.57</td>
<td>8/10/06</td>
</tr>
<tr>
<td>194</td>
<td>4/5/08</td>
<td>3-Medium</td>
<td>Wrap Bag</td>
<td>0.42</td>
<td>4/7/08</td>
</tr>
</tbody>
</table>
### Tables

**Flat**
- Data organized by rows & columns
  - row ~ item (usually)
  - column ~ attribute
  - label ~ attribute name
- Key: identifies each item (row)
  - Usually **unique**
  - Allows **join** of data from 2+ tables
- Compound key: key split among multiple columns, e.g. (state, year) for population

**Multidimensional**
- Split compound key

---

[Munzner (ill. Maguire), 2014]
Attribute Types

- **Categorical**
  - +
  - ●
  - ■
  - △

- **Ordered**
  - **Ordinal**
  - Quantitative

[Munzner (ill. Maguire), 2014]
# Categorial, Ordinal, and Quantitative

<table>
<thead>
<tr>
<th>Order ID</th>
<th>Order Date</th>
<th>Order Priority</th>
<th>Product Container</th>
<th>Product Base Margin</th>
<th>Ship Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10/14/06</td>
<td>5-Low</td>
<td>Large Box</td>
<td>0.8</td>
<td>10/21/06</td>
</tr>
<tr>
<td>6</td>
<td>2/21/08</td>
<td>4-Not Specified</td>
<td>Small Pack</td>
<td>0.55</td>
<td>2/22/08</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Small Pack</td>
<td>0.79</td>
<td>7/17/07</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Jumbo Box</td>
<td>0.72</td>
<td>7/17/07</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.6</td>
<td>7/18/07</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.65</td>
<td>7/18/07</td>
</tr>
<tr>
<td>35</td>
<td>10/23/07</td>
<td>4-Not Specified</td>
<td>Wrap Bag</td>
<td>0.52</td>
<td>10/24/07</td>
</tr>
<tr>
<td>35</td>
<td>10/23/07</td>
<td>4-Not Specified</td>
<td>Small Box</td>
<td>0.58</td>
<td>10/25/07</td>
</tr>
<tr>
<td>36</td>
<td>11/3/07</td>
<td>1-Urgent</td>
<td>Small Box</td>
<td>0.55</td>
<td>11/3/07</td>
</tr>
<tr>
<td>65</td>
<td>3/18/07</td>
<td>1-Urgent</td>
<td>Small Pack</td>
<td>0.49</td>
<td>3/19/07</td>
</tr>
<tr>
<td>66</td>
<td>1/20/05</td>
<td>5-Low</td>
<td>Wrap Bag</td>
<td>0.56</td>
<td>1/20/05</td>
</tr>
<tr>
<td>69</td>
<td>6/4/05</td>
<td>4-Not Specified</td>
<td>Small Pack</td>
<td>0.44</td>
<td>6/6/05</td>
</tr>
<tr>
<td>69</td>
<td>6/4/05</td>
<td>4-Not Specified</td>
<td>Wrapt Bag</td>
<td>0.6</td>
<td>6/6/05</td>
</tr>
<tr>
<td>70</td>
<td>12/18/06</td>
<td>5-Low</td>
<td>0.59</td>
<td>12/23/06</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>12/18/06</td>
<td>5-Low</td>
<td>0.82</td>
<td>12/23/06</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>4/17/05</td>
<td>2-High</td>
<td>0.55</td>
<td>4/19/05</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>1/29/06</td>
<td>3-Medium</td>
<td>0.38</td>
<td>1/30/06</td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>11/19/08</td>
<td>5-Low</td>
<td>0.37</td>
<td>11/28/08</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>5/8/08</td>
<td>2-High</td>
<td>Small Box</td>
<td>0.37</td>
<td>5/9/08</td>
</tr>
<tr>
<td>130</td>
<td>5/8/08</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.38</td>
<td>5/10/08</td>
</tr>
<tr>
<td>130</td>
<td>5/8/08</td>
<td>2-High</td>
<td>Small Box</td>
<td>0.6</td>
<td>5/11/08</td>
</tr>
<tr>
<td>132</td>
<td>6/11/06</td>
<td>3-Medium</td>
<td>Medium Box</td>
<td>0.6</td>
<td>6/12/06</td>
</tr>
<tr>
<td>132</td>
<td>6/11/06</td>
<td>3-Medium</td>
<td>Jumbo Box</td>
<td>0.69</td>
<td>6/14/06</td>
</tr>
<tr>
<td>134</td>
<td>5/1/08</td>
<td>4-Not Specified</td>
<td>Large Box</td>
<td>0.82</td>
<td>5/3/08</td>
</tr>
<tr>
<td>135</td>
<td>10/21/07</td>
<td>4-Not Specified</td>
<td>Small Pack</td>
<td>0.64</td>
<td>10/23/07</td>
</tr>
<tr>
<td>166</td>
<td>9/12/07</td>
<td>2-High</td>
<td>Small Box</td>
<td>0.55</td>
<td>9/14/07</td>
</tr>
<tr>
<td>193</td>
<td>8/8/06</td>
<td>1-Urgent</td>
<td>Medium Box</td>
<td>0.57</td>
<td>8/10/06</td>
</tr>
<tr>
<td>194</td>
<td>4/5/08</td>
<td>3-Medium</td>
<td>Wrap Bag</td>
<td>0.42</td>
<td>4/7/08</td>
</tr>
</tbody>
</table>

**Quantitative**

**Ordinal**

**Categorical**
# Categorial, Ordinal, and Quantitative

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S</th>
<th>T</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order ID</td>
<td>Order Date</td>
<td>Order Priority</td>
<td>Product Container</td>
<td>Product Base Margin</td>
<td>Ship Date</td>
</tr>
<tr>
<td>3</td>
<td>10/14/06</td>
<td>5-Low</td>
<td>Large Box</td>
<td>0.8</td>
<td>10/21/06</td>
</tr>
<tr>
<td>6</td>
<td>2/21/08</td>
<td>4-Not Specified</td>
<td>Small Pack</td>
<td>0.55</td>
<td>2/22/08</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Small Pack</td>
<td>0.79</td>
<td>7/17/07</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Jumbo Box</td>
<td>0.72</td>
<td>7/17/07</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.6</td>
<td>7/18/07</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.65</td>
<td>7/18/07</td>
</tr>
<tr>
<td>35</td>
<td>10/23/07</td>
<td>4-Not Specified</td>
<td>Wrap Bag</td>
<td>0.52</td>
<td>10/24/07</td>
</tr>
<tr>
<td>35</td>
<td>10/23/07</td>
<td>4-Not Specified</td>
<td>Small Box</td>
<td>0.58</td>
<td>10/25/07</td>
</tr>
<tr>
<td>36</td>
<td>11/3/07</td>
<td>1-Urgent</td>
<td>Small Box</td>
<td>0.55</td>
<td>11/3/07</td>
</tr>
<tr>
<td>65</td>
<td>3/18/07</td>
<td>1-Urgent</td>
<td>Small Pack</td>
<td>0.49</td>
<td>3/19/07</td>
</tr>
<tr>
<td>66</td>
<td>1/20/05</td>
<td>5-Low</td>
<td>Wrap Bag</td>
<td>0.56</td>
<td>1/20/05</td>
</tr>
<tr>
<td>69</td>
<td>6/4/05</td>
<td>4-Not Specified</td>
<td>Small Pack</td>
<td>0.44</td>
<td>6/6/05</td>
</tr>
<tr>
<td>69</td>
<td>6/4/05</td>
<td>4-Not Specified</td>
<td>Small Pack</td>
<td>0.6</td>
<td>6/6/05</td>
</tr>
<tr>
<td>70</td>
<td>12/18/06</td>
<td>5-Low</td>
<td>0.59</td>
<td>12/23/06</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>12/18/06</td>
<td>5-Low</td>
<td>0.82</td>
<td>12/23/06</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>4/17/05</td>
<td>2-High</td>
<td>0.55</td>
<td>4/19/05</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>1/29/06</td>
<td>3-Medium</td>
<td>0.38</td>
<td>1/30/06</td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>11/19/08</td>
<td>5-Low</td>
<td>0.37</td>
<td>11/28/08</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>5/8/08</td>
<td>2-High</td>
<td>Small Box</td>
<td>0.37</td>
<td>5/9/08</td>
</tr>
<tr>
<td>130</td>
<td>5/8/08</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.38</td>
<td>5/10/08</td>
</tr>
<tr>
<td>130</td>
<td>5/8/08</td>
<td>2-High</td>
<td>Small Box</td>
<td>0.6</td>
<td>5/11/08</td>
</tr>
<tr>
<td>132</td>
<td>6/11/06</td>
<td>3-Medium</td>
<td>Medium Box</td>
<td>0.6</td>
<td>6/12/06</td>
</tr>
<tr>
<td>132</td>
<td>6/11/06</td>
<td>3-Medium</td>
<td>Jumbo Box</td>
<td>0.69</td>
<td>6/14/06</td>
</tr>
<tr>
<td>134</td>
<td>5/1/08</td>
<td>4-Not Specified</td>
<td>Large Box</td>
<td>0.82</td>
<td>5/3/08</td>
</tr>
<tr>
<td>135</td>
<td>10/21/07</td>
<td>4-Not Specified</td>
<td>Small Pack</td>
<td>0.64</td>
<td>10/23/07</td>
</tr>
<tr>
<td>166</td>
<td>9/12/07</td>
<td>2-High</td>
<td>Small Box</td>
<td>0.55</td>
<td>9/14/07</td>
</tr>
<tr>
<td>193</td>
<td>8/8/06</td>
<td>1-Urgent</td>
<td>Medium Box</td>
<td>0.57</td>
<td>8/10/06</td>
</tr>
<tr>
<td>194</td>
<td>4/5/08</td>
<td>3-Medium</td>
<td>Wrap Bag</td>
<td>0.42</td>
<td>4/7/08</td>
</tr>
</tbody>
</table>

- **Quantitative**
- **Ordinal**
- **Categorical**
Attribute Types

• May be further specified for computational storage/processing
  - **Categorical**: string, boolean, blood type
  - **Ordered**: enumeration, t-shirt size
  - **Quantitative**: integer, float, fixed decimal, datetime

• Sometimes, types can be *inferred* from the data
  - e.g. numbers and none have decimal points → integer
  - could be incorrect (data doesn't have floats, but could be)
Ordering Direction

- **Sequential**

- **Diverging**

- **Cyclic**
Sequential and Diverging Data

- Sequential: homogenous range from a minimum to a maximum
  - Examples: Land elevations, ocean depths
- Diverging: can be deconstructed into two sequences pointing in opposite directions
  - Has a zero point (not necessary 0)
  - Example: Map of both land elevation and ocean depth

[Rogowitz & Treinish, 1998]
Sequential and Diverging Data

- **Sequential**: homogenous range from a minimum to a maximum
  - Examples: Land elevations, ocean depths

- **Diverging**: can be deconstructed into two sequences pointing in opposite directions
  - Has a **zero point** (not necessary 0)
  - Example: Map of both land elevation and ocean depth

[Rogowitz & Treinish, 1998]
Cyclic Data
Cyclic Data
Semantics

• The meaning of the data
• Example: 94023, 90210, 02747, 60115
Semantics

• The meaning of the data
• Example: 94023, 90210, 02747, 60115
  - Attendance at college football games?
Semantics

• The meaning of the data
• Example: 94023, 90210, 02747, 60115
  - Attendance at college football games?
  - Salaries?
Semantics

• The meaning of the data
• Example: 94023, 90210, 02747, 60115
  - Attendance at college football games?
  - Salaries?
  - Zip codes?
• Cannot always infer based on what the data looks like
• Often require semantics to better understand data, column names help
• May also include rules about data: a zip code is part of an address that uniquely identifies a residence
• Useful for asking good questions about the data
Data Model vs. Conceptual Model

- Data Model: raw data that has a specific data type (e.g. floats):
  - Temperature Example: [32.5, 54.0, -17.3] (floats)
- Conceptual Model: how we think about the data
  - Includes semantics, reasoning
  - Temperature Example:
    - Quantitative: [32.50, 54.00, -17.30]
Data Model vs. Conceptual Model

• Data Model: raw data that has a specific data type (e.g. floats):
  - Temperature Example: \([32.5, 54.0, -17.3]\) (floats)

• Conceptual Model: how we think about the data
  - Includes semantics, reasoning
  - Temperature Example:
    • Quantitative: \([32.50, 54.00, -17.30]\)
    • Ordered: \([\text{warm, hot, cold}]\)

[via A. Lex, 2015]
Data Model vs. Conceptual Model

• Data Model: raw data that has a specific data type (e.g. floats):
  - Temperature Example: \([32.5, 54.0, -17.3]\) (floats)

• Conceptual Model: how we think about the data
  - Includes semantics, reasoning
  - Temperature Example:
    • Quantitative: \([32.50, 54.00, -17.30]\)
    • Ordered: \([\text{warm}, \text{hot}, \text{cold}]\)
    • Categorical: \([\text{not burned}, \text{burned}, \text{not burned}]\)

[via A. Lex, 2015]
Derived Data
Derived Data

• Often, data in its original form isn't as useful as we would like
• Examples: Data about a basketball team's games
Derived Data

• Often, data in its original form isn't as useful as we would like
• Examples: Data about a basketball team's games
• Example 1: 1stHalfPoints, 2ndHalfPoints
  - More useful to know total number of points
  - Points = 1stHalfPoints + 2ndHalfPoints
Derived Data

- Often, data in its original form isn't as useful as we would like.
- Examples: Data about a basketball team's games.
- Example 1: `1stHalfPoints, 2ndHalfPoints`
  - More useful to know total number of points.
  - Points = `1stHalfPoints + 2ndHalfPoints`
- Example 2: `Points, OpponentPoints`
  - Want to have a column indicating win/loss.
  - Win = True if (Points > OpponentPoints) else False.
Derived Data

- Often, data in its original form isn't as useful as we would like.
- Examples: Data about a basketball team's games.
- Example 1: `1stHalfPoints, 2ndHalfPoints`
  - More useful to know total number of points.
  - Points = `1stHalfPoints + 2ndHalfPoints`
- Example 2: `Points, OpponentPoints`
  - Want to have a column indicating win/loss.
  - Win = `True if (Points > OpponentPoints) else False`
- Example 3: `Points`
  - Want to have a column indicating how that point total ranks.
  - Rank = index in sorted list of all Point values.
pandas

• Contains high-level data structures and manipulation tools designed to make data analysis fast and easy in Python
• Built on top of NumPy
• Requirements:
  - Data structures with labeled axes (aligning data)
  - Time series data
  - Arithmetic operations that include metadata (labels)
  - Handle missing data
  - Merge and relational operations
Pandas Code Conventions

• Universal:
  - import pandas as pd

• Also used:
  - from pandas import Series, DataFrame
Series

- A one-dimensional array (with a type) with an **index**
- Index defaults to numbers but can also be text (like a dictionary)
- Allows easier reference to specific items
- `obj = pd.Series([7, 14, -2, 1])`
- Basically two arrays: `obj.values` and `obj.index`
- Can specify the index explicitly and use strings
- `obj2 = pd.Series([4, 7, -5, 3],
                  index=['d', 'b', 'a', 'c'])`
- Kind of like fixed-length, ordered dictionary + can create from a dictionary
- `obj3 = pd.Series({'Ohio': 35000, 'Texas': 71000,
                   'Oregon': 16000, 'Utah': 5000})`
Series

- Indexing: s[1] or s['Oregon']
- Can check for missing data: pd.isnull(s) or pd.notnull(s)
- Both index and values can have an associated name:
  - s.name = 'population'; s.index.name = 'state'
- Addition and NumPy ops work as expected and preserve the index-value link
- These operations **align**: 

<table>
<thead>
<tr>
<th>In [28]: obj3</th>
<th>In [29]: obj4</th>
<th>In [30]: obj3 + obj4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ohio</td>
<td>California</td>
<td>California</td>
</tr>
<tr>
<td>35000</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>Oregon</td>
<td>Ohio</td>
<td>Ohio</td>
</tr>
<tr>
<td>16000</td>
<td>35000</td>
<td>35000</td>
</tr>
<tr>
<td>Texas</td>
<td>Oregon</td>
<td>Oregon</td>
</tr>
<tr>
<td>71000</td>
<td>16000</td>
<td>16000</td>
</tr>
<tr>
<td>Utah</td>
<td>Texas</td>
<td>Texas</td>
</tr>
<tr>
<td>5000</td>
<td>71000</td>
<td>71000</td>
</tr>
<tr>
<td>dtype: int64</td>
<td>dtype: float64</td>
<td>dtype: float64</td>
</tr>
</tbody>
</table>

[W. McKinney, Python for Data Analysis]
Data Frame

- A dictionary of Series (labels for each series)
- A spreadsheet with column headers
- Has an index shared with each series
- Allows easy reference to any cell
- `df = DataFrame({'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada'],
                 'pop': [1.5, 1.7, 3.6, 2.4]})`

- Index is automatically assigned just as with a series but can be passed in as well via index kwarg
- Can reassign column names by passing columns kwarg
Chicago Food Inspections Exploration

• Based on David Beazley's PyData Chicago talk
• Data
• YouTube video: https://www.youtube.com/watch?v=j6VSAsKAj98
• Our in-class exploration:
  - Python can give answers fairly quickly
  - Data analysis questions:
    • What is information is available
    • **Questions** are interesting about this dataset
    • How to decide on good follow-up questions
    • What the computations mean
Chicago Food Inspections Exploration