DSC 201: Data Analysis & Visualization

numpy Arrays

Dr. David Koop
Flattening the Sphere?

- Central Meridian (selected by mapmaker)
- Great distortion at high latitudes
- Examples of two rhumb lines (direction true between any two points)
- Equator touches cylinder if cylinder is tangent
- Reasonably true shapes and distances within 15 degrees of Equator
Map Projection Tradeoffs

World Mercator projection with country going to true size

@neilrkaye
Map Projection Tradeoffs

World Mercator projection with country going to true size

@neilrkaye
Choropleth Map: 2016 Election by County

[Washington Post, 2018]
Aggregation: 2016 Election by State

[D. Koop, DSC 201, Fall 2018]

[Washington Post, 2018]
Other Encodings: Glyphs

2016 ELECTION MAP

Each figure represents 250,000 votes

- Trump
- Clinton
- Other

Votes are distributed by state as accurately as possible while keeping national totals correct. Location within each state is approximate.
Maps Aren't Always Best: Close House Races

<table>
<thead>
<tr>
<th>12 Lean Democratic</th>
<th>31 Tossups</th>
<th>25 Lean Republican</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ-02 Open (McSally)</td>
<td>CA-10 Denham</td>
<td>AR-02 Hill</td>
</tr>
<tr>
<td>CA-49 Open (Issa)</td>
<td>CA-25 Knight</td>
<td>CA-50 Hunter</td>
</tr>
<tr>
<td>CO-06 Coffman</td>
<td>CA-39 Open (Royce)</td>
<td>FL-15 Open (Ross)</td>
</tr>
<tr>
<td>IA-01 Blum</td>
<td>CA-45 Walters</td>
<td>FL-16 Buchanan</td>
</tr>
<tr>
<td>KS-03 Yoder</td>
<td>CA-48 Rohrabacher</td>
<td>GA-06 Handel</td>
</tr>
<tr>
<td>MI-11 Open (Trott)</td>
<td>FL-26 Curbelo</td>
<td>GA-07 Woodall</td>
</tr>
<tr>
<td>MN-02 Lewis</td>
<td>FL-27 Open (Ros-Lehtinen)</td>
<td>IL-13 Davis</td>
</tr>
<tr>
<td>MN-03 Paulsen</td>
<td>IL-06 Roskam</td>
<td>IL-14 Hultgren</td>
</tr>
<tr>
<td>NV-03 Open (Rosen)</td>
<td>IL-12 Bost</td>
<td>MO-02 Wagner</td>
</tr>
<tr>
<td>NJ-11 Open (Frelinghuysen)</td>
<td>IA-03 Young</td>
<td>MT-AL Gianforte</td>
</tr>
<tr>
<td>PA-07 Vacant (formerly Dent)</td>
<td>KS-02 Open (Jenkins)</td>
<td>NE-02 Bacon</td>
</tr>
<tr>
<td>VA-10 Comstock</td>
<td>KY-06 Barr</td>
<td>NY-24 Katsko</td>
</tr>
<tr>
<td></td>
<td>ME-02 Poliquin</td>
<td>NY-27 Collins</td>
</tr>
<tr>
<td></td>
<td>MI-08 Bishop</td>
<td>NC-02 Holding</td>
</tr>
<tr>
<td></td>
<td>MN-01 Open (Walz)</td>
<td>OH-12 Balderson</td>
</tr>
<tr>
<td></td>
<td>MN-08 Open (Nolan)</td>
<td>PA-10 Perry</td>
</tr>
<tr>
<td></td>
<td>NJ-03 MacArthur</td>
<td>PA-16 Kelly</td>
</tr>
<tr>
<td></td>
<td>NJ-07 Lance</td>
<td>SC-01 Open (Sanford)</td>
</tr>
<tr>
<td></td>
<td>NM-02 Open (Pearce)</td>
<td>TX-23 Hurd</td>
</tr>
<tr>
<td></td>
<td>NY-19 Faso</td>
<td>TX-31 Carter</td>
</tr>
<tr>
<td></td>
<td>NY-22 Tenney</td>
<td>VA-05 Open (Garrett)</td>
</tr>
<tr>
<td></td>
<td>NC-09 Open (Pittenger)</td>
<td>WA-03 Herrera Beutler</td>
</tr>
<tr>
<td></td>
<td>NC-13 Budd</td>
<td>WA-05 McMorris Rodgers</td>
</tr>
<tr>
<td></td>
<td>OH-01 Chabot</td>
<td>WV-03 Vacant (formerly Jenkins)</td>
</tr>
<tr>
<td></td>
<td>PA-01 Fitzpatrick</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TX-07 Culberson</td>
<td>WI-01 Open (Ryan)</td>
</tr>
<tr>
<td></td>
<td>TX-32 Sessions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UT-04 Love</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VA-02 Taylor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VA-07 Brat</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WA-08 Open (Reichert)</td>
<td></td>
</tr>
</tbody>
</table>

[New York Times, 2018]
Sets

- Sets are like dictionaries but without any values:
  - \( s = \{ 'MA', 'RI', 'CT', 'NH' \}; \quad t = \{ 'MA', 'NY', 'NH' \} \)
  - \( \{ \} \) is an empty dictionary, \( \texttt{set()} \) is an empty set
- Adding values: \( s.\text{add}( 'ME' ) \)
- Removing values: \( s.\text{discard}( 'CT' ) \)
- Exists: "CT" in \( s \)
- Union: \( s \mid t \Rightarrow \{ 'MA', 'RI', 'CT', 'NH', 'NY' \} \)
- Intersection: \( s \& t \Rightarrow \{ 'MA', 'NH' \} \)
- Exclusive-or (xor): \( s ^ t \Rightarrow \{ 'RI', 'CT', 'NY' \} \)
- Difference: \( s - t \Rightarrow \{ 'RI', 'CT' \} \)
None

• Like null in other languages
• Used as a placeholder when no value exists
• The value returned from a function that doesn't return a value
  
  ```python
  def f(name):
      print("Hello,", name)
  v = f("Patricia") # v will have the value None
  ```

• Also used when you need to create a new list or dictionary:
  
  ```python
  def add_letters(s, d=None):
      if d is None:
          d = {}
      d.update(count_letters(s))
  ```

• Looks like `d={}` would make more sense, but that causes issues
• *None* serves as a **sentinel** value in `add_letters`
is and ==

• == does a normal equality comparison
• is checks to see if the object is the exact same object
• Common style to write statements like `if d is None: ...`
• Weird behavior:
  - `a = 4 - 3`
    - `a is 1` # True
  - `a = 10 ** 3`
    - `a is 1000` # False
  - `a = 10 ** 3`
    - `a == 1000` # True

• Python caches common integer objects
• Generally, avoid is unless writing `is None`
Python Modules

• Python module: a file containing definitions and statements

• Import statement: like Java, get a module that isn't a Python builtin

```python
import collections

d = collections.defaultdict(list)
d[3].append(1)
```

• From...import...: don't need to refer to the module

```python
from collections import defaultdict

d = defaultdict(list)
d[3].append(1)
```

• Import as:

```python
import collections as c

d = c.defaultdict(list)
d[3].append(1)
```
Comprehensions

- Shorthand for transformative or filtering for loops

```python
squares = []
for i in range(10):
    if i % 3 != 1:
        squares.append(i ** 2)
```

- `squares = [i**2 for i in range(10) if i % 3 != 1]`

- Equivalent code, just moved the loop inside of list definition

- Advantages: concise, readable

- Also works for dictionaries

```python
names = {"Al": ["Smith", "Brown"], "Beth": ["Jones"]}
first_counts = {k: len(v) for k, v in names.items()}
```
Midterm

- Tuesday, October 23, during class (2-3:15pm, entire class period)
- Material:
  - Everything since the beginning of class
  - Ch. 1-4 in book, Visualization, Exploratory Data Analysis, Python
- Format: (like Quiz 1)
  - Multiple Choice
  - Free Response
- No class on Thursday, October 25
- No office hours next week
Exceptions

• errors but potentially something that can be addressed
• try-except-else-finally:
  - except clause runs if exactly the error(s) you wish to address happen
  - else clause will run if no exceptions are encountered
  - finally: always runs (even if the program is about to crash)
• Can have multiple except clauses
• can also raise exceptions using the raise keyword
• (and define your own)
Exception Example

- def divide(mylist, x, y):
  newlist = []
  try:
    z = x // y
    below, mid, above = \
    mylist[:z], mylist[z], mylist[z+1:]
  except ZeroDivisionError:
    below, mid, above = mylist, -999, []
  except IndexError:
    below, mid, above = mylist, -999, []
  else:
    newlist = below + above
  finally:
    newlist.append(-999)
Exception Example

```
• def divide(mylist, x, y):
    newlist = []
    try:
        z = x // y
        below, mid, above = \
        mylist[:z], mylist[z], mylist[z+1:]
    except (ZeroDivisionError, IndexError):
        below, mid, above = mylist, -999, []
    else:
        newlist = below + above
    finally:
        newlist.append(-999)
```
Object-Oriented Programming: Why?
Object-Oriented Programming

• Encapsulation
• Inheritance
• Polymorphism
• Nesting/Composition

• Components:
  - Instance variables/methods
  - Class variables/methods
Classes

- `class ClassName:
  ...

- Everything in the class should be indented until the declaration ends

- `self: this` in Java or C++ is `self` in Python

- Every instance method has `self` as its first parameter

- Instance variables are defined in methods (usually constructor)

- `__init__`: the constructor

- `__init__` should initialize instance variables

- `def __init__(self):
  
  self.a = 12
  self.b = 'abc'

- `def __init__(self, a, b):
  
  self.a = a
  self.b = b`
Class Example

- class Rectangle:
  
  def __init__(self, x, y, w, h):
    self.x = x
    self.y = y
    self.w = w
    self.h = h

  def set_corner(self, x, y):
    self.x = x
    self.y = y

  def set_width(self, w):
    self.w = w

  def set_height(self, h):
    self.h = h

  def area(self):
    return self.w * self.h
Advanced Classes

- Can have class variables (defined in the class body)
- Can have class methods (use `@classmethod`)
- Can have static methods (use `@staticmethod`)
- Class methods are passed the class as an object, static methods take no instance or class arguments
- Can have properties (use `@property`, `@<name>.setter`)
- `@` directives are called **decorators** and precede the class/method they decorate
Advanced Classes

• A class variables and instance variable can have the same name…

• ```
class A:
    b = 6
    def set_b(self):
        self.b = 3

    @classmethod
    def set_class_b(cls):
        cls.b = 7

a = A()
a.b # 6
a.set_b()
a.b # 3
a.set_class_b()
a.b # 3
A.b # 7
```
Properties

• class Rectangle:
  
  def __init__(self, x, y, w, h):
    self.x = x
    self.y = y
    self.w = w
    self.h = h

  @property
  def width(self):
    return self.w

  @width.setter
  def width(self, w):
    if w > 0: self.w = w

r = Rectangle(0,0,12,3)
print(r.width)
r.width = 4
Inheritance

• Parentheses after the class name indicate superclass

• class Rectangle:
  
  def __init__(self, x, y, w, h):
  self.x = x; self.y = y
  self.w = w; self.h = h

  
  class Square(Rectangle):
  def __init__(self, x, y, s):
  self.x = x; self.y = y
  self.w = s; self.h = s

• super() can be used to call the superclass method:

• class Square(Rectangle):
  def __init__(self, x, y, s):
  super().__init__(x,y,s,s)

• Python allows multiple inheritance (multiple classes separated by commas in the parentheses)
Overriding and Overloading Methods

• class Square(Rectangle):
  
  def __init__(self, x, y, s):
    super().__init__(x, y, s, s)

  def set_width(self, w):
    super().set_width(w)
    super().set_height(w)

  def set_height(self, h):
    super().set_width(h)
    super().set_height(h)

• Overriding: use the same name

• No overloading, but with keyword parameters (or by checking types) can accomplish similar results if needed
Exercise: Queue Class

• Write a class to encapsulate queue behavior. It should have five methods:
  - **constructor**: should allow a list of initial elements (in order)
  - **size**: should return the number of elements
  - **is_empty**: returns `True` if the queue is empty, `False` otherwise
  - **enqueue**: adds an item to the queue
  - **dequeue**: removes an item from the queue and returns it
Exercise: Stack Class

• How do we modify this for a stack?
  - constructor: should allow a list of initial elements (in order)
  - size: should return the number of elements
  - is_empty: returns True if the stack is empty, False otherwise
  - push instead of enqueue: adds an item to the stack
  - pop instead of dequeue: removes an item from the stack

• Could we use inheritance?
What is the difference between an array and a list (or a tuple)?
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]
import numpy as np
Textbook's Notebooks

- ch04.ipynb
- Click the raw button and save that file to disk
- ...or download/clone the entire repository
Creating arrays

- `data1 = [6, 7.5, 8, 0, 1]`
  `arr1 = np.array(data1)`
- `data2 = [[1,2,3,4],[5,6,7,8]]`
  `arr2 = np.array(data2)`
- **Number of dimensions**: `arr2.ndim`
- **Shape**: `arr2.shape`
- **Types**: `arr1.dtype, arr2.dtype, can specify explicitly (np.float64)`
- **Zeros**: `np.zeros(10)`
- **Ones**: `np.ones((4,5))`
- **Empty**: `np.empty((2,2))`
- _like versions: pass an existing array and matches shape with specified contents_
- **Range**: `np.arange(15)`