Programming Principles in Python (CSCI 503/490)

Review

Dr. David Koop
Tasks Machine Learning can Help With

• Identifying the zip code from handwritten digits on an envelope

• Detecting fraudulent activity in credit card transactions
• Identifying topics in a set of blog posts
• Grouping customers with similar preferences

[A. Müller & S. Guido, Introduction to Machine Learning with Python, J. Steppan (MNIST image)]
Machine Learning

- Traditional Programming

  Data → Computer → Output
  Program → Computer

- Machine Learning

  Data → Computer → Program
  Output → Computer
Types of Learning

• Supervised (inductive) learning
  - Training data includes desired outputs

• Unsupervised learning
  - Training data does not include desired outputs

• Semi-supervised learning
  - Training data includes a few desired outputs

• Reinforcement learning
  - Rewards from sequence of actions
Supervised Learning
Supervised Learning: Learned Algorithm (Fit)
Supervised Learning: Prediction
Unsupervised Learning: Input
Unsupervised Learning: Output
scikit-learn entities

- Data: numpy matrices (also pandas series, data frames), process batches
- Estimator: all supervised & unsupervised algs implement **common** interface
  - estimator initialization does not do learning, only attaches parameters
  - **fit** does the learning, learned parameters exposed with trailing underscore
- Predictor: extends estimator with **predict** method
  - also provides **score** method to return value indicating prediction quality
- Transformer: help modify or filter data before learning
  - Preprocessing, feature selection, feature extraction, and dimensionality reduction via **transform** method
  - Can combine **fit** and **transform** via **fit_transform**

[L. Buitinck et al.]
scikit-learn Template

1. Choose model class
2. Instantiate model
3. Fit model to data
4. Predict on new data

```python
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(Xtrain, ytrain)
y_model = model.predict(Xtest)
```
5. (Check accuracy)

```python
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
```
Deep Learning

- Deep learning is tied to neural networks, attempting to mimic how human neurons work together
- Hierarchical with multiple layers
- Usually takes advantage of GPUs
- Frameworks:
  - pytorch
  - TensorFlow
  - keras
  - theano
Assignment 8

- Energy Data
- Data Manipulation using pandas
- Visualization using matplotlib and altair
Final Exam

- Tuesday, December 7 at **12:00pm-1:50pm** in PM 153
- **More** comprehensive than Test 2
- Expect questions from topics covered on Test 1 and 2
- Expect questions from the last four weeks of class (concurrency, data, visualization, machine learning)
- Similar format
Questions?
Why Python?

• High-level, readable
• Productivity
• Large standard library
• Libraries, Libraries, Libraries
• What about Speed?
  - What speed are we measuring?
  - Time to code vs. time to execute
JupyterLab and Jupyter Notebooks

In this Notebook we explore the Lorenz system of differential equations:

\[
\begin{align*}
\dot{x} &= \sigma(y - x) \\
\dot{y} &= px - y - xz \\
\dot{z} &= -rz + xy
\end{align*}
\]

Let’s call the function once to view the solutions. For this set of parameters, we see the trajectories swirling around two points, called attractors.

```python
In [4]: from lorenz import solve_lorenz
t, x, z = solve_lorenz(N=10)
```

![Output View](JupyterLab Documentation)
Principles: Explicit Code

• Complex code isn't necessarily bad, but make sure you can't make it clearer

• Bad:

```python
def make_complex(*args):
    x, y = args
    return dict(**locals())
```

• Good

```python
def make_complex(x, y):
    return {'x': x, 'y': y}
```
Principles: Don't Repeat Yourself

• "Two or more, use a for" [Dijkstra]

• Rule of Three: [Roberts]
  - Don't copy-and-paste more than once
  - Refactor into methods

• Repeated code is harder to maintain

• Bad

```python
f1 = load_file('f1.dat')
r1 = get_cost(f1)
f2 = load_file('f2.dat')
r2 = get_cost(f2)
f3 = load_file('f3.dat')
r3 = get_cost(f3)
```

• Good

```python
for i in range(1, 4):
f = load_file(f'f{i}.dat')
r = get_cost(f)
```
Expression Rules

• Involve
  - Literals \( 1, "abc" \),
  - Variables \( a, \text{my\_height} \), and
  - Operators \( +, -, *, /, //, ** \)

• Spaces are irrelevant within an expression
  - \( a + \ 34 \ # \ ok \)

• Standard precedence rules
  - Parentheses, exponentiation, mult/div, add/sub
  - Left to right at each level

• Also boolean expressions
Identifiers

• A sequence of letters, digits, or underscores, but…
• Also includes unicode "letters", spacing marks, and decimals (e.g. Σ)
• Must begin with a letter or underscore (_)
• Why not a number?
• Case sensitive (a is different from A)
• Conventions:
  - Identifiers beginning with an underscore (_) are reserved for system use
  - Use underscores (a_long_variable), not camel-case (aLongVariable)
  - Keep identifier names less than 80 characters
• Cannot be reserved words
Types

• Don't worry about types, but think about types
• Variables can "change types"
  - \( a = 0 \)
  - \( a = "abc" \)
  - \( a = 3.14159 \)
• Actually, the name is being moved to a different value
• You can find out the type of the value stored at a variable \( v \) using \( \text{type}(v) \)
• Some literal types are determined by subtle differences
  - 1 vs 1. (integer vs. float)
  - 1.43 vs 1.43j (float vs. imaginary)
  - '234' vs b'234' (string vs. byte string)
Assignment

• The = operator: 
  \[ a = 34; c = (a + b) ** 2 \]

• Python variables are actually **pointers** to objects

• Also, augmented assignment: 
  \[ +=, -=, *=, /=, //=, **= \]

\[
\begin{align*}
  x &= 42 \\
  x &= x + 1 \\
  y &= x
\end{align*}
\]
Boolean Expressions

• Type `bool`: True or False

• Note `capitalization`!

• Comparison Operators: `<`, `<=`, `>`, `>=`, `==`, `!=`
  - Double equals (==) checks for equal values,
  - Assignment ( assignment of values to variables

• Boolean operators: `not`, `and`, `or`
  - Different from many other languages (`!`, `&&`, `||`)

• More:
  - `is`: exact same object (usually `a_variable is None`)
  - `in`: checks if a value is in a collection (`34 in my_list`)
if, else, elif, pass

- if a < 10:
  print("Small")
else:
  if a < 100:
    print("Medium")
  else:
    if a < 1000:
      print("Large")
    else:
      print("X-Large")

- if a < 10:
  print("Small")
elif a < 100:
  print("Medium")
elif a < 1000:
  print("Large")
else:
  print("X-Large")

• Indentation is critical so else-if branches can become unwieldy (elif helps)
• Remember colons and indentation
• pass can be used for an empty block
Loop Styles

• Loop-and-a-Half

```python
d = get_data() # priming rd
while check(d):
    # do stuff
    d = get_data()
```

• Infinite-Loop-Break

```python
while True:
    d = get_data()
    if check(d):
        break
    # do stuff
```

• Assignment Expression (Walrus)

```python
while check(d := get_data):
    # do stuff
```
Functions

• Use `return` to return a value

```python
def <function-name>(<parameter-names>):
    # do stuff
    return res
```

• Can return more than one value using commas

```python
def <function-name>(<parameter-names>):
    # do stuff
    return res1, res2
```

• Use **simultaneous assignment** when calling:
  - `a, b = do_something(1,2,5)`

• If there is no return value, the function returns `None` (a special value)
Positional & Keyword Arguments

• Generally, any argument can be passed as a keyword argument
• def f(alpha, beta, gamma=1, delta=7, epsilon=8, zeta=2, eta=0.3, theta=0.5, iota=0.24, kappa=0.134):
  # ...
• f(5, 6)
• f(alpha=7, beta=12, iota=0.7)
Pass by object reference

- AKA passing object references by value
- Python doesn't allocate space for a variable, it just links identifier to a value
- **Mutability** of the object determines whether other references see the change
- Any immutable object will act like pass by value
- Any mutable object acts like pass by reference unless it is reassigned to a new value
Sequences

- Strings "abcde", Lists [1, 2, 3, 4, 5], and Tuples (1, 2, 3, 4, 5)

- Defining a list: my_list = [0, 1, 2, 3, 4]

- But lists can store different types:
  - my_list = [0, "a", 1.34]

- Including other lists:
  - my_list = [0, "a", 1.34, [1, 2, 3]]

- Others are similar: tuples use parenthesis, strings are delineated by quotes (single or double)
Indexing & Slicing

• Positive or negative indices can be used at any step
• my_str = "abcde"; my_str[1] + my_str[-4] # "bb"
• my_list = [1, 2, 3, 4, 5]; my_list[3:-1] # [4]

• Implicit indices
  - my_tuple = (1, 2, 3, 4, 5); my_tuple[-2:] # (4, 5)
  - my_tuple[:3] # (1, 2, 3)
Tuples

• Tuples are immutable sequences
• We've actually seen tuples a few times already
  - Simultaneous Assignment
  - Returning Multiple Values from a Function
• Python allows us to omit parentheses when it's clear
  - \( b, a = a, b \) \# same as \((b, a) = (a, b)\)
  - \( t1 = a, b \) \# don't normally do this
  - \( c, d = f(2, 5, 8) \) \# same as \((c, d) = f(2, 5, 8)\)
  - \( t2 = f(2, 5, 8) \) \# don't normally do this
Dictionary

- AKA associative array or map
- Collection of key-value pairs
  - Keys must be unique
  - Values need not be unique
- Syntax:
  - Curly brackets {} delineate start and end
  - Colons separate keys from values, commas separate pairs
    - `d = {'DeKalb': 783, 'Kane': 134, 'Cook': 1274, 'Will': 546}`
- No type constraints
  - `d = {'abc': 25, 12: 'abc', ('Kane', 'IL'): 123.54}`
Collections

- A dictionary is **not** a sequence
- Sequences are **ordered**
- Conceptually, dictionaries need no order
- A dictionary is a **collection**
- Sequences are also collections
- All collections have length (`len`), membership (`in`), and iteration (loop over values)
- Length for dictionaries counts number of key-value **pairs**
  - Pass dictionary to the `len` function
  - `d = {'abc': 25, 12: 'abc', ('Kane', 'IL'): 123.54}
  ```
  len(d) # 3
  ```
List Comprehension

- output = []
  for d in range(5):
    output.append(d ** 2 - 1)

- Rewrite as a map:
  output = [d ** 2 - 1 for d in range(5)]

- Can also filter:
  output = [d for d in range(5) if d % 2 == 1]

- Combine map & filter:
  output = [d ** 2 - 1 for d in range(5) if d % 2 == 1]
Short-Circuit Evaluation

• Automatic, works left to right according to order of operations (and before or)
• Works for and and or

• and:
  - if any value is False, stop and return False
    - a, b = 2, 3
      a > 3 and b < 5

• or:
  - if any value is True, stop and return True
    - a, b, c = 2, 3, 7
      a > 3 or b < 5 or c > 8
Strings

• Remember strings are sequences of characters
• Strings are collections so have `len`, `in`, and iteration
  
  ```python
  s = "Huskies"
  len(s); "usk" in s; [c for c in s if c == 's']
  ```

• Strings are sequences so have
  
  - indexing and slicing: `s[0]`, `s[1:]`
  - concatenation and repetition: `s + " at NIU"`; `s * 2`

• Single or double quotes `string1`, "string2"

• Triple double-quotes: """A string over many lines""

• Escaped characters: \n (newline) \t (tab)
Regular Expressions

- AKA regex
- A syntax to better specify how to decompose strings
- Look for patterns rather than specific characters
- "31" in "The last day of December is 12/31/2016."
- May work for some questions but now suppose I have other lines like: "The last day of September is 9/30/2016."
- ...and I want to find dates that look like:
  - {digits}/{digits}/{digits}
- Cannot search for every combination!
  - \d+/%d+/%d+ # \d is a character class
Reading & Writing Files

• Can iterate through the file (think of the file as a collection of lines):
  
  ```python
  f = open('huck-finn.txt', 'r')
  for line in f:
    if 'Huckleberry' in line:
      print(line.strip())
  ```

• For writing, with statement does "enter" and "exit": don't need to call outf.close()

  ```python
  with open('output.txt', 'w') as outf:
    for k, v in counts.items():
      outf.write(k + ': ' + v + '\n')
  ```
Command Line Interfaces (CLIs)

• Prompt:
  - $ 

  - 

• Commands
  - $ cat <filename>
  - $ git init

• Arguments/Flags: (options)
  - $ python -h
  - $ head -n 5 <filename>
  - $ git branch fix-parsing-bug
Modules and Packages

- Python allows you to import code from other files, even your own
- A **module** is a collection of definitions
- A **package** is an organized collection of modules
- Modules can be
  - a separate python file
  - a separate C library that is written to be used with Python
  - a built-in module contained in the interpreter
  - a module installed by the user (via conda or pip)
- All types use the same import syntax
Namespaces

• Namespace is basically a dictionary with names and their values

• Accessing namespaces
  - __builtins__, globals(), locals()

• Examine contents of a namespace:
  dir(<namespace>)

• Python checks for a name in the sequence:
  local, enclosing, global, builtins

• To access names in outer scopes, use
  global (global) and nonlocal (enclosing) declarations
Array Operations

- \( a = \text{np.array([1, 2, 3])} \)
  \( b = \text{np.array([6, 4, 3])} \)

- (Array, Array) Operations (Element-wise)
  - Addition, Subtraction, Multiplication
  - \( a + b \) # array([7, 6, 6])

- (Scalar, Array) Operations (Broadcasting):
  - Addition, Subtraction, Multiplication, Division, Exponentiation
  - \( a ^ 2 \) # array([1, 4, 9])
  - \( b + 3 \) # array([9, 7, 6])
Array Slicing

• 2D+: comma separated indices as shorthand:
  - arr2 = np.array([[1.5, 2, 3, 4], [5, 6, 7, 8]])
  - a[1:2, 1:3]
  - a[1:2, :] # works like in single-dimensional lists

• Can combine index and slice in different dimensions
  - a[1, :] # gives a row
  - a[:, 1] # gives a column

• Slicing vs. indexing produces different shapes!
  - a[1, :] # 1-dimensional
  - a[1:2, :] # 2-dimensional
Object-Oriented Programming Concepts

• Abstraction: simplify, hide implementation details, don't repeat yourself
• Encapsulation: represent an entity fully, keep attributes and methods together
• Inheritance: reuse (don't reinvent the wheel), specialization
• Polymorphism: methods are handled by a single interface with different implementations (overriding)
Classes and Instances in Python

• Class Definition:
  - class Vehicle:
    def __init__(self, make, model, year, color):
      self.make = make
      self.model = model
      self.year = year
      self.color = color
    
    def age(self):
      return 2021 - self.year

• Instances:
  - car1 = Vehicle('Toyota', 'Camry', 2000, 'red')
  - car2 = Vehicle('Dodge', 'Caravan', 2015, 'gray')
Subclass

- Just put superclass(-es) in parentheses after the class declaration
- class Car(Vehicle):
  
def __init__(self, make, model, year, color, num_doors):
    super().__init__(make, model, year, color)
    self.num_doors = num_doors

    def open_door(self):
      ...

- super() is a special method that locates the base class
  - Constructor should call superclass constructor
  - Extra arguments should be initialized and extra instance methods
Typing

- Dynamic Typing: variable's type can change (what Python does)
- Static Typing: compiler enforces types, variable types generally don't change
- Duck Typing: check method/attribute existence, not type
- Python is a dynamically-typed language (and plans to remain so)
- …but it has recently added more support for type hinting/annotations that allow **static type checking**
- Type annotations change **nothing** at runtime!
Dealing with Errors

• Can explicitly check for errors at each step
  - Check for division by zero
  - Check for invalid parameter value (e.g. string instead of int)

• Sometimes all of this gets in the way and can't be addressed succinctly
  - Too many potential errors to check
  - Cannot handle groups of the same type of errors together

• Allow programmer to determine when and how to handle issues
  - Allow things to go wrong and handle them instead
  - Allow errors to be propagated and addressed once
Try, Except, Else, and Finally

- \( b = 3 \)
- \( a = 0 \)

```python
try:
    c = b / a
except ZeroDivisionError:
    print("Division failed")
    c = 0
else:
    print("Division succeeded", c)
finally:
    print("This always runs")
```
Debugging

• print statements
• logging library
• pdb
• Extensions for IDEs (e.g. PyCharm)
• JupyterLab Debugger Support
Testing

- If statements
- Assert statements
- Unit Testing
- Integration Testing
Python Modules for Working with the Filesystem

- In general, cross-platform! (Linux, Mac, Windows)
- `os`: translations of operating system commands
- `shutil`: better support for file and directory management
- `fnmatch`, `glob`: match filenames, paths
- `os.path`: path manipulations
- `pathlib`: object-oriented approach to path manipulations, also includes some support for matching paths
Concurrency: CPU-Bound vs. I/O-Bound

- **CPU Processing**
  - Compute Problem 1
  - Compute Problem 2

- **I/O Waiting**
  - Request 1
  - Request 2
  - Request 3

- **CPU Processing**

Time

[J. Anderson]
```python
import pandas as pd

df = pd.read_csv('penguins_lter.csv')
```

<table>
<thead>
<tr>
<th>studyName</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Culmen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PAL0708</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A1</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.1</td>
</tr>
<tr>
<td>1</td>
<td>PAL0708</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A2</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.5</td>
</tr>
<tr>
<td>2</td>
<td>PAL0708</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>40.3</td>
</tr>
<tr>
<td>3</td>
<td>PAL0708</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A2</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>PAL0708</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N3A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>36.7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>339</td>
<td>PAL0910</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N38A2</td>
<td>No</td>
<td>12/1/09</td>
<td>NaN</td>
</tr>
<tr>
<td>340</td>
<td>PAL0910</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>46.8</td>
</tr>
<tr>
<td>341</td>
<td>PAL0910</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>50.4</td>
</tr>
<tr>
<td>342</td>
<td>PAL0910</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>45.2</td>
</tr>
<tr>
<td>343</td>
<td>PAL0910</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>49.9</td>
</tr>
</tbody>
</table>

344 rows x 17 columns
```python
df = pd.read_csv('penguins_lter.csv')
```

**Column Names**

<table>
<thead>
<tr>
<th>studyName</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Culmen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PAL0708</td>
<td>1 Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A1</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.1</td>
</tr>
<tr>
<td>1</td>
<td>PAL0708</td>
<td>2 Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A2</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.5</td>
</tr>
<tr>
<td>2</td>
<td>PAL0708</td>
<td>3 Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>40.3</td>
</tr>
<tr>
<td>3</td>
<td>PAL0708</td>
<td>4 Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A2</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>PAL0708</td>
<td>5 Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N3A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>36.7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>339</td>
<td>PAL0910</td>
<td>120 Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N38A2</td>
<td>No</td>
<td>12/1/09</td>
<td>NaN</td>
</tr>
<tr>
<td>340</td>
<td>PAL0910</td>
<td>121 Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>46.8</td>
</tr>
<tr>
<td>341</td>
<td>PAL0910</td>
<td>122 Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>50.4</td>
</tr>
<tr>
<td>342</td>
<td>PAL0910</td>
<td>123 Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>45.2</td>
</tr>
<tr>
<td>343</td>
<td>PAL0910</td>
<td>124 Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>49.9</td>
</tr>
</tbody>
</table>

344 rows x 17 columns
```python
df = pd.read_csv('penguins_lter.csv')
```

### Column Names

<table>
<thead>
<tr>
<th>studyName</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Culmen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAL0708</td>
<td>1</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A1</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.1</td>
</tr>
<tr>
<td>PAL0708</td>
<td>2</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A2</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.5</td>
</tr>
<tr>
<td>PAL0708</td>
<td>3</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>40.3</td>
</tr>
<tr>
<td>PAL0708</td>
<td>4</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A2</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
</tr>
<tr>
<td>PAL0708</td>
<td>5</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N3A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>36.7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>PAL0910</td>
<td>120</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N38A2</td>
<td>No</td>
<td>12/1/09</td>
<td>NaN</td>
</tr>
<tr>
<td>PAL0910</td>
<td>121</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>46.8</td>
</tr>
<tr>
<td>PAL0910</td>
<td>122</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>50.4</td>
</tr>
<tr>
<td>PAL0910</td>
<td>123</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>45.2</td>
</tr>
<tr>
<td>PAL0910</td>
<td>124</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>49.9</td>
</tr>
</tbody>
</table>

344 rows x 17 columns
Data Frame

```
df = pd.read_csv('penguins_lter.csv')
```

<table>
<thead>
<tr>
<th>studyName</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Culmen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAL0708</td>
<td>1</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A1</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.1</td>
</tr>
<tr>
<td>PAL0708</td>
<td>2</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A2</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.5</td>
</tr>
<tr>
<td>PAL0708</td>
<td>3</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>40.3</td>
</tr>
<tr>
<td>PAL0708</td>
<td>4</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A2</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
</tr>
<tr>
<td>PAL0708</td>
<td>5</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N3A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>36.7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>PAL0910</td>
<td>120</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N38A2</td>
<td>No</td>
<td>12/1/09</td>
<td>NaN</td>
</tr>
<tr>
<td>PAL0910</td>
<td>121</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>46.8</td>
</tr>
<tr>
<td>PAL0910</td>
<td>122</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>50.4</td>
</tr>
<tr>
<td>PAL0910</td>
<td>123</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>45.2</td>
</tr>
<tr>
<td>PAL0910</td>
<td>124</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>49.9</td>
</tr>
</tbody>
</table>

344 rows x 17 columns

Column: df['Island']
## Data Frame

```python
df = pd.read_csv('penguins_lter.csv')
```

### Column Names

<table>
<thead>
<tr>
<th>studyName</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Culmen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAL0708</td>
<td>1</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A1</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.1</td>
</tr>
<tr>
<td>PAL0708</td>
<td>2</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A2</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.5</td>
</tr>
<tr>
<td>PAL0708</td>
<td>3</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>40.3</td>
</tr>
<tr>
<td>PAL0708</td>
<td>4</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A2</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
</tr>
<tr>
<td>PAL0708</td>
<td>5</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N3A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>36.7</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>PAL0910</td>
<td>120</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N38A2</td>
<td>No</td>
<td>12/1/09</td>
<td>NaN</td>
</tr>
<tr>
<td>PAL0910</td>
<td>121</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>46.8</td>
</tr>
<tr>
<td>PAL0910</td>
<td>122</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>50.4</td>
</tr>
<tr>
<td>PAL0910</td>
<td>123</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>45.2</td>
</tr>
<tr>
<td>PAL0910</td>
<td>124</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>49.9</td>
</tr>
</tbody>
</table>

344 rows x 17 columns

### Row: `df.loc[2]`

```
df.loc[2]  # Row 2
```

### Index

The index is not shown in the table above. It is likely to be a continuous numerical index, starting from 0.

### Column: `df['Island']`

```
df['Island']
```

The `Island` column contains values for the different islands where the penguins are located.
### Data Frame

```python
df = pd.read_csv('penguins_lter.csv')
```

#### Column Names

<table>
<thead>
<tr>
<th>Index</th>
<th>Study Name</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Culmen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PAL0708</td>
<td>1</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A1</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.1</td>
</tr>
<tr>
<td>1</td>
<td>PAL0708</td>
<td>2</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A2</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.5</td>
</tr>
<tr>
<td>2</td>
<td>PAL0708</td>
<td>3</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>40.3</td>
</tr>
<tr>
<td>3</td>
<td>PAL0708</td>
<td>4</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A2</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>PAL0708</td>
<td>5</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N3A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>36.7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>339</td>
<td>PAL0910</td>
<td>120</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N38A2</td>
<td>No</td>
<td>12/1/09</td>
<td>NaN</td>
</tr>
<tr>
<td>340</td>
<td>PAL0910</td>
<td>121</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>46.8</td>
</tr>
<tr>
<td>342</td>
<td>PAL0910</td>
<td>123</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>50.4</td>
</tr>
<tr>
<td>343</td>
<td>PAL0910</td>
<td>124</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>45.2</td>
</tr>
</tbody>
</table>

#### Row: `df.loc[2]`

#### Index

#### Cell: `df.loc[341, 'Species']`

Gentoo penguin (Pygoscelis papua)

#### Column: `df['Island']`

344 rows x 17 columns
## Data Frame

```python
df = pd.read_csv('penguins_lter.csv')
```

### Column Names

<table>
<thead>
<tr>
<th>studyName</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Culmen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAL0708</td>
<td>1</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A1</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.1</td>
</tr>
<tr>
<td>PAL0708</td>
<td>2</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A2</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.5</td>
</tr>
<tr>
<td>PAL0708</td>
<td>3</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>40.3</td>
</tr>
<tr>
<td>PAL0708</td>
<td>4</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A2</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
</tr>
<tr>
<td>PAL0708</td>
<td>5</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N3A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>PAL0910</td>
<td>120</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N38A2</td>
<td>No</td>
<td>12/1/09</td>
<td>NaN</td>
</tr>
<tr>
<td>PAL0910</td>
<td>121</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>46.8</td>
</tr>
<tr>
<td>PAL0910</td>
<td>123</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>50.4</td>
</tr>
<tr>
<td>PAL0910</td>
<td>124</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>45.2</td>
</tr>
<tr>
<td>PAL0910</td>
<td>125</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>49.9</td>
</tr>
</tbody>
</table>

344 rows x 17 columns

### Row: `df.loc[2]`

#### Index

- Row 2: `PAL0708` 3

#### Cell: `df.loc[341,'Species']`

- `Gentoo penguin (Pygoscelis papua)`

### Missing Data

- Row 3: `NaN`
- Row 2: `NaN`

D. Koop, CSCI 503/490, Fall 2021
Aggregation of time series data, a special use case of groupby, is referred to as resampling in this book and will receive separate treatment in Chapter 10.

GroupBy Mechanics

Hadley Wickham, an author of many popular packages for the R programming language, coined the term split-apply-combine for talking about group operations, and I think that's a good description of the process. In the first stage of the process, data contained in a pandas object, whether a Series, DataFrame, or otherwise, is split into groups based on one or more keys that you provide. The splitting is performed on a particular axis of an object. For example, a DataFrame can be grouped on its rows (axis=0) or its columns (axis=1). Once this is done, a function is applied to each group, producing a new value. Finally, the results of all those function applications are combined into a result object. The form of the resulting object will usually depend on what's being done to the data. See Figure 9-1 for a mockup of a simple group aggregation.

Figure 9-1. Illustration of a group aggregation

Each grouping key can take many forms, and the keys do not have to be all of the same type:

- A list or array of values that is the same length as the axis being grouped
- A value indicating a column name in a DataFrame

[W. McKinney, Python for Data Analysis]
Tidy Data: Melt

- Want to keep each observation separate (tidy), aka pivot_longer

```python
df.melt(id_vars=['location', 'Temperature'],
       var_name='Date', value_name='Value')
```

<table>
<thead>
<tr>
<th>location</th>
<th>Temperature</th>
<th>Jan-2010</th>
<th>Feb-2010</th>
<th>Mar-2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CityA</td>
<td>30</td>
<td>45</td>
<td>24</td>
</tr>
<tr>
<td>1</td>
<td>CityB</td>
<td>32</td>
<td>43</td>
<td>22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>location</th>
<th>Temperature</th>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CityA</td>
<td>Jan-2010</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>CityB</td>
<td>Jan-2010</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>CityA</td>
<td>Feb-2010</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>CityB</td>
<td>Feb-2010</td>
<td>43</td>
</tr>
<tr>
<td>4</td>
<td>CityA</td>
<td>Mar-2010</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>CityB</td>
<td>Mar-2010</td>
<td>22</td>
</tr>
</tbody>
</table>

[AB Abhi]
Tidy Data: Pivot

• Sometimes, we have data that is given in "long" format and we would like "wide" format (aka pivot_wider)

• Long format: column names are data values...

• Wide format: more like spreadsheet format

• Example:

<table>
<thead>
<tr>
<th>date</th>
<th>item</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1959-03-31</td>
<td>realgdp</td>
<td>2710.349</td>
</tr>
<tr>
<td>1959-03-31</td>
<td>infl</td>
<td>0.000</td>
</tr>
<tr>
<td>1959-03-31</td>
<td>unemp</td>
<td>5.800</td>
</tr>
<tr>
<td>1959-06-30</td>
<td>realgdp</td>
<td>2778.801</td>
</tr>
<tr>
<td>1959-06-30</td>
<td>infl</td>
<td>2.340</td>
</tr>
<tr>
<td>1959-06-30</td>
<td>unemp</td>
<td>5.100</td>
</tr>
<tr>
<td>1959-09-30</td>
<td>realgdp</td>
<td>2775.488</td>
</tr>
<tr>
<td>1959-09-30</td>
<td>infl</td>
<td>2.740</td>
</tr>
<tr>
<td>1959-09-30</td>
<td>unemp</td>
<td>5.300</td>
</tr>
<tr>
<td>1959-12-31</td>
<td>realgdp</td>
<td>2785.204</td>
</tr>
</tbody>
</table>

```python
.data = pd.read_csv('ch07/macrodata.csv')
.periods = pd.PeriodIndex(year=data.year, quarter=data.quarter, name='date')
data = DataFrame(data.to_records(),
columns=pd.Index(['realgdp', 'infl', 'unemp'], name='item'),
index=periods.to_timestamp('D', 'end'))
.ldata = data.stack().reset_index().rename(columns={0: 'value'})
```

```
In [116]: ldata[:10]
Out[116]:
date     item     value
0 1959-03-31  realgdp  2710.349
1 1959-03-31     infl     0.000
2 1959-03-31    unemp     5.800
3 1959-06-30  realgdp  2778.801
4 1959-06-30     infl     2.340
5 1959-06-30    unemp     5.100
6 1959-09-30  realgdp  2775.488
7 1959-09-30     infl     2.740
8 1959-09-30    unemp     5.300
9 1959-12-31  realgdp  2785.204
```

```
In [117]: pivoted = ldata.pivot('date', 'item', 'value')
In [118]: pivoted.head()
Out[118]:
item        infl   realgdp  unemp
date
1959-03-31  0.00  2710.349    5.8
1959-06-30  2.34  2778.801    5.1
1959-09-30  2.74  2775.488    5.3
1959-12-31  0.27  2785.204    5.6
1960-03-31  2.31  2847.699    5.2
```

Pivoting “long” to “wide” Format

A common way to store multiple time series in databases and CSV is in so-called long or stacked format:

```python
data = pd.read_csv('ch07/macrodata.csv')
.periods = pd.PeriodIndex(year=data.year, quarter=data.quarter, name='date')
data = DataFrame(data.to_records(),
columns=pd.Index(['realgdp', 'infl', 'unemp'], name='item'),
index=periods.to_timestamp('D', 'end'))
.ldata = data.stack().reset_index().rename(columns={0: 'value'})
```

```
In [116]: ldata[:10]
Out[116]:
date     item     value
0 1959-03-31  realgdp  2710.349
1 1959-03-31     infl     0.000
2 1959-03-31    unemp     5.800
3 1959-06-30  realgdp  2778.801
4 1959-06-30     infl     2.340
5 1959-06-30    unemp     5.100
6 1959-09-30  realgdp  2775.488
7 1959-09-30     infl     2.740
8 1959-09-30    unemp     5.300
9 1959-12-31  realgdp  2785.204
```

Data is frequently stored this way in relational databases like MySQL as a fixed schema (column names and data types) allows the number of distinct values in the item column to increase or decrease as data is added or deleted in the table. In the above example date and item would usually be the primary keys (in relational database parlance), offering both relational integrity and easier joins and programmatic queries in many cases. The downside, of course, is that the data may not be easy to work with in long format; you might prefer to have a DataFrame containing one column per distinct item value indexed by timestamps in the date column. DataFrame’s pivot method performs exactly this transformation:

```python
In [117]: pivoted = ldata.pivot('date', 'item', 'value')
In [118]: pivoted.head()
Out[118]:
item        infl   realgdp  unemp
date
1959-03-31  0.00  2710.349    5.8
1959-06-30  2.34  2778.801    5.1
1959-09-30  2.74  2775.488    5.3
1959-12-31  0.27  2785.204    5.6
1960-03-31  2.31  2847.699    5.2
```

The first two values passed are the columns to be used as the row and column index, and finally an optional value column to fill the DataFrame. Suppose you had two value columns that you wanted to reshape simultaneously:

```python
In [119]: ldata['value2'] = np.random.randn(len(ldata))
In [120]: ldata[:10]
Out[120]:
date     item     value    value2
0 1959-03-31  realgdp  2710.349  1.669025
1 1959-03-31     infl     0.000 -0.438570
2 1959-03-31    unemp     5.800 -0.539741
3 1959-06-30  realgdp  2778.801  0.476985
4 1959-06-30     infl     2.340  3.248944
5 1959-06-30    unemp     5.100 -1.021228
6 1959-09-30  realgdp  2775.488 -0.577087
7 1959-09-30     infl     2.740  0.124121
8 1959-09-30    unemp     5.300  0.302614
9 1959-12-31  realgdp  2785.204  0.523772
```

By omitting the last argument, you obtain a DataFrame with hierarchical columns:

```python
In [121]: pivoted = ldata.pivot('date', 'item')
In [122]: pivoted[:5]
Out[122]:
value                     value2
item         infl   realgdp  unemp      infl   realgdp     unemp
date
1959-03-31   0.00  2710.349    5.8 -0.438570  1.669025 -0.539741
1959-06-30   2.34  2778.801    5.1  3.248944  0.476985 -1.021228
1959-09-30   2.74  2775.488    5.3  0.124121 -0.577087  0.302614
1959-12-31   0.27  2785.204    5.6  0.000940  0.523772  1.343810
1960-03-31   2.31  2847.699    5.2 -0.831154 -0.713544 -2.370232
```

Reshaping and Pivoting | 191

[In [123]: pivoted['value'][:5]]
Out[123]:
item        infl   realgdp  unemp
date
1959-03-31  0.00  2710.349    5.8
1959-06-30  2.34  2778.801    5.1
1959-09-30  2.74  2775.488    5.3
1959-12-31  0.27  2785.204    5.6
1960-03-31  2.31  2847.699    5.2

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

[F. J. Anscombe]
Visualizing Data

Mean of x  9
Variance of x  11
Mean of y  7.50
Variance of y  4.122
Correlation  0.816
matplotlib

• Strengths:
  - Designed like Matlab
  - Many rendering backends
  - Can reproduce almost any plot
  - Proven, well-tested

• Weaknesses:
  - API is imperative
  - Not originally designed for the web
  - Dated styles
Altair

- Declarative Visualization
  - Specify **what** instead of how
  - Separate specification from execution
- Based on VegaLite which is browser-based
- Strengths:
  - Declarative visualization
  - Web technologies
- Drawbacks:
  - Moving data between Python and JS
  - Sometimes longer specifications
Visual Marks

- **Marks** are the basic graphical elements in a visualization
- Marks classified by dimensionality:
  - Points
  - Lines
  - Areas
- Also can have surfaces, volumes
- Think of marks as a mathematical definition, or if familiar with tools like Adobe Illustrator or Inkscape, the path & point definitions
- Altair: area, bar, circle, geoshape, image, line, point, rect, rule, square, text, tick
  - Also compound marks: boxplot, errorband, errorbar
Data is Encoded via Visual Channels

- **Position**
  - Horizontal
  - Vertical
  - Both

- **Color**

- **Shape**

- **Tilt**

- **Size**
  - Length
  - Area
  - Volume

[Muñzner (ill. Maguire), 2014]
Multiple Views

[Improvise, Weaver, 2004]
Questions?