Advanced Data Management (CSCI 490/680)

Data Cleaning

Dr. David Koop
Comma-separated values (CSV) Format

- Comma is a field separator, newlines denote records
  - a,b,c,d,message
  - 1,2,3,4,hello
  - 5,6,7,8,world
  - 9,10,11,12,foo

- May have a header (a,b,c,d,message), but not required

- No type information: we do not know what the columns are (numbers, strings, floating point, etc.)
  - Default: just keep everything as a string
  - Type inference: Figure out the type to make each column based on values

- What about commas in a value? ➔ double quotes
# Reading & Writing Data in Pandas

<table>
<thead>
<tr>
<th>Format</th>
<th>Data Description</th>
<th>Reader</th>
<th>Writer</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>CSV</td>
<td>read_csv</td>
<td>to_csv</td>
</tr>
<tr>
<td>text</td>
<td>Fixed-Width Text File</td>
<td>read_fwf</td>
<td></td>
</tr>
<tr>
<td>text</td>
<td>JSON</td>
<td>read_json</td>
<td>to_json</td>
</tr>
<tr>
<td>text</td>
<td>HTML</td>
<td>read_html</td>
<td>to_html</td>
</tr>
<tr>
<td>text</td>
<td>Local clipboard</td>
<td>read_clipboard</td>
<td>to_clipboard</td>
</tr>
<tr>
<td>text</td>
<td>MS Excel</td>
<td>read_excel</td>
<td>to_excel</td>
</tr>
<tr>
<td>binary</td>
<td>OpenDocument</td>
<td>read_excel</td>
<td></td>
</tr>
<tr>
<td>binary</td>
<td>HDF5 Format</td>
<td>read_hdf</td>
<td>to_hdf</td>
</tr>
<tr>
<td>binary</td>
<td>Feather Format</td>
<td>read_feather</td>
<td>to_feather</td>
</tr>
<tr>
<td>binary</td>
<td>Parquet Format</td>
<td>read_parquet</td>
<td>to_parquet</td>
</tr>
<tr>
<td>binary</td>
<td>ORC Format</td>
<td>read_orc</td>
<td></td>
</tr>
<tr>
<td>binary</td>
<td>Msgpack</td>
<td>read_msgpack</td>
<td>to_msgpack</td>
</tr>
<tr>
<td>binary</td>
<td>Stata</td>
<td>read_stata</td>
<td>to_stata</td>
</tr>
<tr>
<td>binary</td>
<td>SAS</td>
<td>read_sas</td>
<td></td>
</tr>
<tr>
<td>binary</td>
<td>SPSS</td>
<td>read_spss</td>
<td></td>
</tr>
<tr>
<td>binary</td>
<td>Python Pickle Format</td>
<td>read_pickle</td>
<td>to_pickle</td>
</tr>
<tr>
<td>SQL</td>
<td>SQL</td>
<td>read_sql</td>
<td>to_sql</td>
</tr>
<tr>
<td>SQL</td>
<td>Google BigQuery</td>
<td>read_gbq</td>
<td>to_gbq</td>
</tr>
</tbody>
</table>

read_csv

- Convenient method to read csv files
- Lots of different options to help get data into the desired format
- Basic: `df = pd.read_csv(fname)`
- Parameters:
  - `path`: where to read the data from
  - `sep` (or `delimiter`): the delimiter (',', ' ', '	', '\s+')
  - `header`: if None, no header
  - `index_col`: which column to use as the row index
  - `names`: list of header names (e.g. if the file has no header)
  - `skiprows`: number of list of lines to skip
Writing CSV data with pandas

- **Basic:** `df.to_csv(<fname>)`
- **Change delimiter with sep kwarg:**
  - `df.to_csv('example.dsv', sep='|')`
- **Change missing value representation**
  - `df.to_csv('example.dsv', na_rep='NULL')`
- **Don't write row or column labels:**
  - `df.to_csv('example.csv', index=False, header=False)`
- **Series may also be written to csv**
JavaScript Object Notation (JSON)

- A format for web data
- Looks very similar to python dictionaries and lists
- Example:

  ```
  {
  "name": "Wes",
  "places_lived": ["United States", "Spain", "Germany"],
  "pet": null,
  "siblings": [
  {
  "name": "Scott", "age": 25, "pet": "Zuko"},
  {
  "name": "Katie", "age": 33, "pet": "Cisco"
  }
  ]
  }
  ```

- Only contains literals (no variables) but allows null
- Values: strings, arrays, dictionaries, numbers, booleans, or null
  - Dictionary keys must be strings
  - Quotation marks help differentiate string or numeric values
JSON Orientation

• Indication of expected JSON string format. Compatible JSON strings can be produced by `to_json()` with a corresponding orient value. The set of possible orients is:

- **split**: dict like `{index -> [index],
  columns -> [columns],
  data -> [values]}`
- **records**: list like `[{column -> value, ... , column -> value}]`
- **index**: dict like `{index -> {column -> value}}`
- **columns**: dict like `{column -> {index -> value}}`
- **values**: just the values array
Binary Formats

- CSV, JSON, and XML are all text formats
- What is a binary format?
- Pickle: Python's built-in serialization
- HDF5: Library for storing large scientific data
  - Hierarchical Data Format, supports **compression**
  - Interfaces in C, Java, MATLAB, etc.
  - Use `pd.HDFStore` to access, shortcuts: `read_hdf/to_hdf`
- Excel: need to specify sheet when a spreadsheet has multiple sheets
  - `pd.ExcelFile` or `pd.read_excel`
- Parquet: big data format, can use compression
Databases

- **Dim_Date**
  - Id
  - Date
  - Day
  - Day_of_Week
  - Month
  - Month_Name
  - Quarter
  - Quarter_Name
  - Year

- **Fact_Sales**
  - Date_Id
  - Store_Id
  - Product_Id
  - Units_Sold

- **Dim_Store**
  - Id
  - Store_Number
  - State_Province
  - Country

- **Dim_Product**
  - Id
  - EAN_Code
  - Product_Name
  - Brand
  - Product_Category

[Wikipedia]
Types of Dirty Data Problems

• Separator Issues: e.g. CSV without respecting double quotes
  - 12, 13, "Doe, John", 45

• Naming Conventions: NYC vs. New York

• Missing required fields, e.g. key

• Different representations: 2 vs. two

• Truncated data: "Janice Keihanaikukauakahihiuliheekahaunaele" becomes "Janice Keihanaikukauakahihiuliheek" on Hawaii license

• Redundant records: may be exactly the same or have some overlap

• Formatting issues: 2017-11-07 vs. 07/11/2017 vs. 11/07/2017

[J. Canny et al.]
Dirty Data: Data Scientist's View

- Combination of:
  - Statistician's View: data has non-ideal samples for model
  - Database Expert's View: missing data, corrupted data
  - Domain Expert's View: data doesn't pass the smell test
- All of the views present problems with the data
- The goal may dictate the solutions:
  - Median value: don't worry too much about crazy outliers
  - Generally, aggregation is less susceptible by numeric errors
  - Be careful, the data may be correct…
Be careful how you detect dirty data

• The appearance of a hole in the earth’s ozone layer over Antarctica, first detected in 1976, was so unexpected that scientists didn’t pay attention to what their instruments were telling them; they thought their instruments were malfunctioning.

– National Center for Atmospheric Research

[Wikimedia]
Assignment 2

- Same data as A1, different version of the dataset
- Dealing with the raw data now
- Same questions as A1, but use pandas
- CS680 students + some questions about problems with the data
Wrangler

- Data cleaning takes a lot of **time** and **human effort**
- "Tedium is the message"
- Repeating this process on multiple data sets is even worse!
- Solution:
  - interactive interface (mixed-initiative)
  - transformation language with natural language "translations"
  - suggestions + "programming by demonstration"
Previous Work: Potter's Wheel

- V. Raman and J. Hellerstein, 2001
- Defines structure extractions for identifying fields
- Defines transformations on the data
- Allows user interaction
## Potter's Wheel: Structure Extraction

<table>
<thead>
<tr>
<th>Example Column Value (Example erroneous values)</th>
<th># Structures Enumerated</th>
<th>Final Structure Chosen (Punc = Punctuation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-60</td>
<td>5</td>
<td>Integer</td>
</tr>
<tr>
<td>UNITED, DELTA, AMERICAN etc.</td>
<td>5</td>
<td>IspellWord</td>
</tr>
<tr>
<td>SFO, LAX etc. (JFK to OAK)</td>
<td>12</td>
<td>AllCapsWord</td>
</tr>
<tr>
<td>1998/01/12</td>
<td>9</td>
<td>Int Punc(/) Int Punc() Int</td>
</tr>
<tr>
<td>M, Tu, Thu etc.</td>
<td>5</td>
<td>Capitalized Word</td>
</tr>
<tr>
<td>06:22</td>
<td>5</td>
<td>Int(len 2) Punc(;) Int(len 2)</td>
</tr>
<tr>
<td>12.8.15.147 (ferret03.webtop.com)</td>
<td>9</td>
<td>Double Punc(’’) Double</td>
</tr>
<tr>
<td>”GET\b (\b)</td>
<td>5</td>
<td>Punc(”) IspellWord Punc()</td>
</tr>
<tr>
<td>/postmodern/lecs/xia/sld013.htm</td>
<td>4</td>
<td>ξ*</td>
</tr>
<tr>
<td>HTTP</td>
<td>3</td>
<td>AllCapsWord(HTTP)</td>
</tr>
<tr>
<td>/1.0</td>
<td>6</td>
<td>Punc() Double(1.0)</td>
</tr>
</tbody>
</table>

[V. Raman and J. Hellerstein, 2001]
Potter's Wheel: Transforms

<table>
<thead>
<tr>
<th>Transform</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>( \phi(R, i, f) = {(a_1, \ldots, a_{i-1}, a_i+, \ldots, a_n, f(a_i)) \mid (a_1, \ldots, a_n) \in R} )</td>
</tr>
<tr>
<td>Add</td>
<td>( \alpha(R, x) = {(a_1, \ldots, a_n, x) \mid (a_1, \ldots, a_n) \in R} )</td>
</tr>
<tr>
<td>Drop</td>
<td>( \pi(R, i) = {(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n) \mid (a_1, \ldots, a_n) \in R} )</td>
</tr>
<tr>
<td>Copy</td>
<td>( \kappa((a_1, \ldots, a_n), i) = {(a_1, \ldots, a_n, a_i) \mid (a_1, \ldots, a_n) \in R} )</td>
</tr>
<tr>
<td>Merge</td>
<td>( \mu((a_1, \ldots, a_n), i, j, \text{glue}) = {(a_1, \ldots, a_{i-1}, a_i+, \ldots, a_{j-1}, a_j+, \ldots, a_n, a_i \oplus \text{glue} \oplus a_j) \mid (a_1, \ldots, a_n) \in R} )</td>
</tr>
<tr>
<td>Split</td>
<td>( \omega((a_1, \ldots, a_n), i, \text{splitter}) = {(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, a_i, \text{null}) \mid (a_1, \ldots, a_n) \in R \land \text{pred}(a_i)} ) \cup {(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, a_i) \mid (a_1, \ldots, a_n) \in R \land \neg\text{pred}(a_i)} )</td>
</tr>
<tr>
<td>Divide</td>
<td>( \delta((a_1, \ldots, a_n), i, \text{pred}) = {(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, \text{null}, a_i) \mid (a_1, \ldots, a_n) \in R \land \text{pred}(a_i)} )</td>
</tr>
<tr>
<td>Fold</td>
<td>( \lambda(R, i_1, i_2, \ldots, i_k) = {(a_1, \ldots, a_{i_1-1}, a_{i_1+1}, \ldots, a_{i_2-1}, a_{i_2+1}, \ldots, a_{i_k-1}, a_{i_k+1}, \ldots, a_n, a_i) \mid (a_1, \ldots, a_n) \in R \land 1 \leq l \leq k} )</td>
</tr>
<tr>
<td>Select</td>
<td>( \sigma(R, \text{pred}) = {(a_1, \ldots, a_n) \mid (a_1, \ldots, a_n) \in R \land \text{pred}(a_1, \ldots, a_n)} )</td>
</tr>
</tbody>
</table>

**Notation:** \( R \) is a relation with \( n \) columns. \( i, j \) are column indices and \( a_i \) represents the value of a column in a row. \( x \) and \( y \) are values. \( f \) is a function mapping values to values. \( x \oplus y \) concatenates \( x \) and \( y \). \( \text{glue} \) is a position in a string or a regular expression. \( \text{left}(x, \text{splitter}) \) is the left part of \( x \) after splitting by \( \text{splitter} \). \( \text{pred} \) is a function returning a boolean.

[V. Raman and J. Hellerstein, 2001]
Potter's Wheel: Example

Format 
'(.*), (.*)' to '2 \1'

2 Merges

Bob
Anna
Jerry
Joan
Stewart
Davis
Dole
Marsh

Bob
Anna
Jerry
Joan
Stewart
Davis
Dole
Marsh

Split at '

Bob Stewart
Anna Davis
Jerry Dole
Joan Marsh

Bob
Anna
Jerry
Joan
Stewart
Davis
Dole
Marsh

[V. Raman and J. Hellerstein, 2001]
Potter's Wheel: Inferring Structure from Examples

| Example Values Split By User (| is user specified split position) | Inferred Structure | Comments |
|--------------------------------|---------------------|-----------|
| Taylor, Jane | $52,072  
   Blair, John | $73,238  
   Tony Smith | $1,00,533 | $<\xi^*<', 'Money'>$) | Parsing is doable despite no good delimiter. A *regular expression* domain can infer a structure of $[0-9,]*$ for last component. |
| MAA | to | SIN  
   JFK | to | SFO  
   LAX | – | ORD  
   SEA | / | OAK | $<\text{len 3 identifier}><\xi^*>$
   $<\text{len 3 identifier}>$ | Parsing is possible despite multiple delimiters. |
| 321 Blake #7 | , | Berkeley | , | CA 94720  
   719 MLK Road | , | Fremont | , | CA 95743 | $<\text{number} \xi^*<', 'word'>$  
   $<', (2 \text{ letter word}) (5 \text{ letter integer})>$ | Parsing is easy because of consistent delimiter. |

[V. Raman and J. Hellerstein, 2001]
Wrangler Transformation Language

• Based on Potter's Wheel
• Map: Delete, Extract, Cut, Split, Update
• Lookup/join: Use external data (e.g. from zipcode→state)
• Reshape: Fold and Unfold (aka pivot)
• Positional: Fill and lag
• Sorting, aggregation, key generation, schema transforms
**Interface**

- Automated Transformation Suggestions
- Editable Natural Language Explanations
- Visual Transformation Previews
- Transformation History

![Natural Language Descriptions](image)

**Wrangler**

- **Fill Bangladesh by copying values from above**
- **Fill Bangladesh by averaging the 5 values from above**

![Table of Data](image)

![Diagram](image)

[S. Kandel et al., 2011]
Automation from past actions

- Infer parameter sets from user interaction
- Generating transforms
- Ranking and ordering transformations:
  - Based on user preferences, difficulty, and corpus frequency
  - Sort transforms by type and diversify suggestions

```
(a) Reported crime in Alabama

before:  {'in', ' '} 'Alabama' → {'Alabama', word}
selection: {'Alabama'} 'in' → {'in', word, lowercase}
after:  {} ' ' → {' '}

before:  {' '}, {'in', ' '}, (word, ' '), (lowercase, ' ')
selection: {'Alabama'}, (word)
after:  {}

{()},('Alabama'),() {}
{(''),0,0}
{(' ').('Alabama'),()}
{(' ').('Alabama'),()}
{(' ').('Alabama'),()}
{('in', ' '),(),()}
{('in', ' '),('Alabama'),()}
{('in', ' '),('Alabama'),()}
{('in', ' '),('Alabama'),()}

{lowercase, ' '},('Alabama'),() → /[a-z]+ (Alabama)/
```

[S. Kandel et al., 2011]
Evaluation

• Compare with Excel
• Tests:
  - Extract text from a single string entry
  - Fill in missing values with estimates
  - Reshape tables
• Allowed users to ask questions about Excel, not Wrangler
• Found significant effect of tool and users found previews and suggestions helpful
• Complaint: No manual fallback, make implications of user choices more obvious for users
Task Completion Times

User Study Task Completion Time (minutes)

Wrangler | Excel
---|---
T1 | 0 1 2 3 4 5 6 7 8 9 10
T2 | 0 1 2 3 4 5 6 7 8 9 10
T3 | 0 1 2 3 4 5 6 7 8 9 10

[S. Kandel et al., 2011]
Improvements in Prediction

Update suggestions when given more information

[Heer et al., 2015]
Data Wrangling Tasks

- Unboxing: Discovery & Assessment: What's in there? (types, distribution)
- Structuring: Restructure data (table, nested data, pivot tables)
- Cleaning: does data match expectations (often involves user)
- Enriching & Blending: Adding new data
- Optimizing & Publishing: Structure for storage or visualization
Differences with Extract-Transform-Load (ETL)

• ETL:
  - Who: IT Professionals
  - Why: Create static data pipeline
  - What: Structured data
  - Where: Data centers

• "Modern Data Preparation":
  - Who: Analysts
  - Why: Solve problems by designing recipes to use data
  - What: Original, custom data blended with other data
  - Where: Cloud, desktop

[J. M. Hellerstein et al., 2018]
Trifacta Wrangler
Paper Critique

- Foofah: Transforming Data By Example, Z. Jin et al., 2017
- Due Wednesday before class, submit via Blackboard
- Read the paper
- Look up references if necessary
- Keep track of things you are confused by or that seem problematic
- Write a few sentences summarizing the paper's contribution
- Write more sentences discussing the paper and what you think the paper does well or doesn't do well at
- For this response, compare/contrast with Wrangler/Trifacta
- Length: 1/2-1 page
Data Cleaning in pandas
Handling Missing Data

- Filtering out missing data:
  - Can choose rows or columns
- Filling in missing data:
  - with a default value
  - with an interpolated value
- In pandas:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dropna</td>
<td>Filter axis labels based on whether values for each label have missing data, with varying thresholds for how much missing data to tolerate.</td>
</tr>
<tr>
<td>fillna</td>
<td>Fill in missing data with some value or using an interpolation method such as 'ffill' or 'bfill'.</td>
</tr>
<tr>
<td>isnull</td>
<td>Return boolean values indicating which values are missing/NA.</td>
</tr>
<tr>
<td>notnull</td>
<td>Negation of isnull.</td>
</tr>
</tbody>
</table>

[W. McKinney, Python for Data Analysis]
Filling in missing data

- fillna arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Scalar value or dict-like object to use to fill missing values</td>
</tr>
<tr>
<td>method</td>
<td>Interpolation; by default 'ffill' if function called with no other arguments</td>
</tr>
<tr>
<td>axis</td>
<td>Axis to fill on; default axis=0</td>
</tr>
<tr>
<td>inplace</td>
<td>Modify the calling object without producing a copy</td>
</tr>
<tr>
<td>limit</td>
<td>For forward and backward filling, maximum number of consecutive periods to fill</td>
</tr>
</tbody>
</table>
Filtering and Cleaning Data

- Find duplicates
  - `duplicated`: returns boolean Series indicating whether row is a duplicate—first instance is \textbf{not marked} as a duplicate

- Remove duplicates:
  - `drop_duplicates`: drops all rows where `duplicated` is True
  - `keep`: which value to keep (first or last)

- Can pass specific columns to check for duplicates, e.g. check only key column
Changing Data

• Convert strings to upper/lower case
• Convert Fahrenheit temperatures to Celsius
• Create a new column based on another column

In [56]: lowercased

Out[56]:
0   bacon
1  pulled pork
2     bacon
3   pastrami
4  corned beef
5     bacon
6   pastrami
7   honey ham
8    nova lox
Name: food, dtype: object

meat_to_animal = {
    'bacon': 'pig',
    'pulled pork': 'pig',
    'pastrami': 'cow',
    'corned beef': 'cow',
    'honey ham': 'pig',
    'nova lox': 'salmon'
}

In [57]: data['animal'] = lowercased.map(meat_to_animal)

In [58]: data

Out[58]:
  food   ounces  animal
0  bacon     4.0     pig
1  bacon     3.0     pig
2  bacon    12.0     pig
3  Pastrami   6.0     cow
4  corned beef  7.5     cow
5     Bacon    8.0     pig
6  pastrami    3.0     cow
7   honey ham    5.0     pig
8    nova lox    6.0   salmon

[W. McKinney, Python for Data Analysis]
Replacing Values

- `fillna` is a special case
- What if `-999` in our dataset was identified as a missing value?

```python
In [61]: data
Out[61]:
0    1.0
1  -999.0
2     2.0
3  -999.0
4 -1000.0
5     3.0
dtype: float64
```

```python
In [62]: data.replace(-999, np.nan)
Out[62]:
0    1.0
1  NaN
2     2.0
3  NaN
4 -1000.0
5     3.0
dtype: float64
```

- Can pass list of values or dictionary to change different values
### Clamping Values

- Values above or below a specified thresholds are set to a max/min value.

#### Example

```python
In [93]: data.describe()
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>1000.000000</td>
<td>1000.000000</td>
<td>1000.000000</td>
<td>1000.000000</td>
</tr>
<tr>
<td>mean</td>
<td>0.049091</td>
<td>0.026112</td>
<td>-0.002544</td>
<td>-0.051827</td>
</tr>
<tr>
<td>std</td>
<td>0.996947</td>
<td>1.007458</td>
<td>0.995232</td>
<td>0.998311</td>
</tr>
<tr>
<td>min</td>
<td>-3.645860</td>
<td>-3.184377</td>
<td>-3.745356</td>
<td>-3.428254</td>
</tr>
<tr>
<td>25%</td>
<td>-0.599807</td>
<td>-0.612162</td>
<td>-0.687373</td>
<td>-0.747478</td>
</tr>
<tr>
<td>50%</td>
<td>0.047101</td>
<td>-0.013609</td>
<td>-0.022158</td>
<td>-0.088274</td>
</tr>
<tr>
<td>75%</td>
<td>0.756646</td>
<td>0.695298</td>
<td>0.699046</td>
<td>0.623331</td>
</tr>
<tr>
<td>max</td>
<td>2.653656</td>
<td>3.525865</td>
<td>2.735527</td>
<td>3.366626</td>
</tr>
</tbody>
</table>

```python
In [97]: data[np.abs(data) > 3] = np.sign(data) * 3
```

```python
In [98]: data.describe()
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>1000.000000</td>
<td>1000.000000</td>
<td>1000.000000</td>
<td>1000.000000</td>
</tr>
<tr>
<td>mean</td>
<td>0.050286</td>
<td>0.025567</td>
<td>-0.001399</td>
<td>-0.051765</td>
</tr>
<tr>
<td>std</td>
<td>0.992920</td>
<td>1.004214</td>
<td>0.991414</td>
<td>0.995761</td>
</tr>
<tr>
<td>min</td>
<td>3.000000</td>
<td>-3.000000</td>
<td>-3.000000</td>
<td>-3.000000</td>
</tr>
<tr>
<td>25%</td>
<td>0.599807</td>
<td>-0.612162</td>
<td>-0.687373</td>
<td>-0.747478</td>
</tr>
<tr>
<td>50%</td>
<td>0.047101</td>
<td>-0.013609</td>
<td>-0.022158</td>
<td>-0.088274</td>
</tr>
<tr>
<td>75%</td>
<td>0.756646</td>
<td>0.695298</td>
<td>0.699046</td>
<td>0.623331</td>
</tr>
<tr>
<td>max</td>
<td>2.653656</td>
<td>3.000000</td>
<td>2.735527</td>
<td>3.000000</td>
</tr>
</tbody>
</table>
Computing Indicator Values

- Useful for machine learning
- Want to take possible values and map them to 0-1 indicators
- Example:

```python
In [109]: df = pd.DataFrame({
                      'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                      'data1': range(6)})
......:

In [110]: pd.get_dummies(df['key'])
Out[110]:
   a  b  c
0  0  1  0
1  0  1  0
2  1  0  0
3  0  0  1
4  1  0  0
5  0  1  0
```

- Example: Genres in movies
String Transformation

- One of the reasons for Python's popularity is string/text processing.

- `split(<delimiter>):` break a string into pieces:
  - `s = "12,13, 14"
    slist = s.split(',') # ["12", "13", " 14"]`

- `<delimiter>.join([<str>]):` join several strings by a delimiter:
  - `": ".join(slist) # "12:13: 14"`

- `strip():` remove leading and trailing whitespace:
  - `[p.strip() for p in slist] # ["12", "13", "14"]`
String Transformation

• `replace(<from>,<to>):` change substrings to another substring
  - `s.replace(',', ':') # "12:13: 14"

• `upper() / lower():` casing
  - "AbCd".upper () # "ABCD"
  - "AbCd".lower() # "abcd"
String Transformations

- **index(<str>):** find where a substring first occurs (Error if not found)
  - `s = "12,13, 14"
    - `s.index(',')) # 2`
    - `s.index(':') # ValueError raised`
- **find(<str>):** same as index but -1 if not found
  - `s.find(',')) # 2`
  - `s.find(':') # -1`
- **startswith() / endswith():** boolean checks for string occurrence
  - `s.startswith("1") # True`
  - `s.endswith("5") # False`
## String Methods

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>Return the number of non-overlapping occurrences of substring in the string.</td>
</tr>
<tr>
<td>endswith</td>
<td>Returns True if string ends with suffix.</td>
</tr>
<tr>
<td>startswith</td>
<td>Returns True if string starts with prefix.</td>
</tr>
<tr>
<td>join</td>
<td>Use string as delimiter for concatenating a sequence of other strings.</td>
</tr>
<tr>
<td>index</td>
<td>Return position of first character in substring if found in the string; raises ValueError if not found.</td>
</tr>
<tr>
<td>find</td>
<td>Return position of first character of first occurrence of substring in the string; like index, but returns −1 if not found.</td>
</tr>
<tr>
<td>rfind</td>
<td>Return position of first character of last occurrence of substring in the string; returns −1 if not found.</td>
</tr>
<tr>
<td>replace</td>
<td>Replace occurrences of string with another string.</td>
</tr>
<tr>
<td>strip, rstrip, lstrip</td>
<td>Trim whitespace, including newlines; equivalent to x.strip() (and rstrip, lstrip, respectively) for each element.</td>
</tr>
<tr>
<td>split</td>
<td>Break string into list of substrings using passed delimiter.</td>
</tr>
<tr>
<td>lower</td>
<td>Convert alphabet characters to lowercase.</td>
</tr>
<tr>
<td>upper</td>
<td>Convert alphabet characters to uppercase.</td>
</tr>
<tr>
<td>casefold</td>
<td>Convert characters to lowercase, and convert any region-specific variable character combinations to a common comparable form.</td>
</tr>
<tr>
<td>ljust, rjust</td>
<td>Left justify or right justify, respectively; pad opposite side of string with spaces (or some other fill character) to return a string with a minimum width.</td>
</tr>
</tbody>
</table>

Regular expressions provide a flexible way to search or match (often more complex) string patterns in text. A single expression, commonly called a regex, is a string formed according to the regular expression language. Python's built-in `re` module is responsible for applying regular expressions to strings; I'll give a number of examples of its use here.

The art of writing regular expressions could be a chapter of its own and thus is outside the book's scope. There are many excellent tutorials and references available on the internet and in other books.
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 12/31/2020."
• May work for some questions but now suppose I have other lines like: "The last day of September is 9/30/2020."
• …and I want to find dates that look like:
  • <numbers>/<numbers>/<numbers>
• Cannot search for every combination!
  • \d+//\d+//\d+
Regular Expressions

- Character classes:
  - \d = digits
  - \s = spaces
  - \w = word character [a-zA-Z0-9_]
  - [a-z] = lowercase letters (square brackets indicate a set of chars)

- Repeating characters or patterns
  - + = one or more (any number)
  - * = zero or more (any number)
  - ? = zero or one
  - {<number>} = a specific number (or range) of occurrences
Regular Expressions in Python

• import re
• re.search(<pattern>, <str_to_check>)
  - Returns None if no match, information about the match otherwise
• Capturing information about what is in a string → parentheses
• (\d+)/\d+/\d+ will capture information about the month
• match = re.search('(^\d+)/\d+/\d+','12/31/2016')
  if match:
    match.group() # 12
• re.findall(<pattern>, <str_to_check>)
  - Finds all matches in the string, search only finds the first match
• Can pass in flags to alter methods: e.g. re.IGNORECASE
Pandas String Methods

• Any column or series can have the string methods (e.g. replace, split) applied to the entire series
• Fast (vectorized) on whole columns or datasets
• use .str.<method_name>
• .str is important!

- data = pd.Series({'Dave': 'dave@google.com',
                   'Steve': 'steve@gmail.com',
                   'Rob': 'rob@gmail.com',
                   'Wes': np.nan})

data.str.contains('gmail')
data.str.split('@').str[1]
data.str[-3:]
Paper Critique

- Foofah: Transforming Data By Example, Z. Jin et al., 2017
- Due Wednesday before class, submit via Blackboard
- Read the paper
- Look up references if necessary
- Keep track of things you are confused by or that seem problematic
- Write a few sentences summarizing the paper's contribution
- Write more sentences discussing the paper and what you think the paper does well or doesn't do well at
- For this response, compare/contrast with Wrangler/Trifacta
- Length: 1/2-1 page