Advanced Data Management (CSCI 680/490)

Data Cleaning

Dr. David Koop
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 Keihanaikukauakahihuliheekahaunaele" becomes "Janice Keihanaikukauakahihuliheek" 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
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…

[J. Canny et al.]
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
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"
Potter's Wheel: Example

<table>
<thead>
<tr>
<th></th>
<th>Stewart, Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anna</td>
<td>Davis</td>
</tr>
<tr>
<td></td>
<td>Dole, Jerry</td>
</tr>
<tr>
<td>Joan</td>
<td>Marsh</td>
</tr>
</tbody>
</table>

Format 

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

<table>
<thead>
<tr>
<th></th>
<th>Bob Stewart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anna</td>
<td>Davis</td>
</tr>
<tr>
<td>Jerry</td>
<td>Dole</td>
</tr>
<tr>
<td>Joan</td>
<td>Marsh</td>
</tr>
</tbody>
</table>

2 Merges

<table>
<thead>
<tr>
<th></th>
<th>Bob</th>
<th>Stewart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anna</td>
<td>Anna</td>
<td>Davis</td>
</tr>
<tr>
<td>Jerry</td>
<td>Jerry</td>
<td>Dole</td>
</tr>
<tr>
<td>Joan</td>
<td>Joan</td>
<td>Marsh</td>
</tr>
</tbody>
</table>

Split at '

[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: $\phi(R, i, f)$</td>
<td>${(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, f(a_i)) \mid (a_1, \ldots, a_n) \in R}$</td>
</tr>
<tr>
<td>Add: $\alpha(R, x)$</td>
<td>${(a_1, \ldots, a_n, x) \mid (a_1, \ldots, a_n) \in R}$</td>
</tr>
<tr>
<td>Drop: $\pi(R, i)$</td>
<td>${(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n) \mid (a_1, \ldots, a_n) \in R}$</td>
</tr>
<tr>
<td>Copy: $\kappa((a_1, \ldots, a_n), i)$</td>
<td>${(a_1, \ldots, a_n, a_i) \mid (a_1, \ldots, a_n) \in R}$</td>
</tr>
<tr>
<td>Merge: $\mu((a_1, \ldots, a_n), i, j, \text{glue})$</td>
<td>${(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_j-1, a_j+1, \ldots, a_n, a_i \oplus \text{glue} \oplus a_j) \mid (a_1, \ldots, a_n) \in R}$</td>
</tr>
<tr>
<td>Split: $\omega((a_1, \ldots, a_n), i, \text{splitter})$</td>
<td>${(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, \text{left}(a_i, \text{splitter}), \text{right}(a_i, \text{splitter})) \mid (a_1, \ldots, a_n) \in R}$</td>
</tr>
<tr>
<td>Divide: $\delta((a_1, \ldots, a_n), i, \text{pred})$</td>
<td>${(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, a_i, \text{null}) \mid (a_1, \ldots, a_n) \in R \land \text{pred}(a_i)} \cup {(a_1, \ldots, a_n) \in R \land \neg\text{pred}(a_i)}$</td>
</tr>
<tr>
<td>Fold: $\lambda(R, i_1, i_2, \ldots i_k)$</td>
<td>${(a_1, \ldots, a_{i_1-1}, a_{i_1+1}, \ldots, a_{i_2-1}, a_{i_2+1}, \ldots, a_{i_k-1}, a_{i_k+1}, \ldots, a_n, a_i) \mid (a_1, \ldots, a_n) \in R \land 1 \leq l \leq k}$</td>
</tr>
<tr>
<td>Select: $\sigma(R, \text{pred})$</td>
<td>${(a_1, \ldots, a_n) \mid (a_1, \ldots, a_n) \in R \land \text{pred}((a_1, \ldots, a_n))}$</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 glue are values. $f$ is a function mapping values to values. $x \oplus y$ concatenates $x$ and $y$. splitter is a position in a string or a regular expression, left($x$, splitter) is the left part of $x$ after splitting by splitter. pred is a function returning a boolean.

[V. Raman and J. Hellerstein, 2001]
Interface

- Automated Transformation Suggestions
- Editable Natural Language Explanations

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

- Visual Transformation Previews
- Transformation History

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

Update suggestions when given more information

[Heer et al., 2015]
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]
Test 1

• This Wednesday, Feb. 23
• In-class, 3:30-4:45pm in PM 153
• Format:
  - Multiple Choice
  - Free Response
• Information posted online
Paper Critique

- Foofah: Transforming Data By Example, Z. Jin et al., 2017
- Due Monday **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 Formats
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
Delimiter-separated Values

- Comma is a **delimiter**, specifies boundary between fields
- Could be a tab, pipe (|), or perhaps spaces instead
- All of these follow similar styles to CSV
## Fixed-width Format

- Old school
- Each field gets a certain number of spots in the file
- Example:

<table>
<thead>
<tr>
<th>id</th>
<th>Value1</th>
<th>Value2</th>
<th>Value3</th>
</tr>
</thead>
<tbody>
<tr>
<td>id8141</td>
<td>360.242940</td>
<td>149.910199</td>
<td>11950.7</td>
</tr>
<tr>
<td>id1594</td>
<td>444.953632</td>
<td>166.985655</td>
<td>11788.4</td>
</tr>
<tr>
<td>id1849</td>
<td>364.136849</td>
<td>183.628767</td>
<td>11806.2</td>
</tr>
<tr>
<td>id1230</td>
<td>413.836124</td>
<td>184.375703</td>
<td>11916.8</td>
</tr>
<tr>
<td>id1948</td>
<td>502.953953</td>
<td>173.237159</td>
<td>12468.3</td>
</tr>
</tbody>
</table>

- Specify exact character ranges for each field, e.g. 0-6 is the id
Reading & Writing Data
Reading Data in Python

- Use the `open()` method to open a file for reading
  
  ```python
  f = open('huck-finn.txt')
  ```

- Usually, add an `'r'` as the second parameter to indicate "read"

- 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())
  ```

- Using `line.strip()` because the read includes the newline, and print writes a newline so we would have double-spaced text

- Closing the file: `f.close()`
With Statement: Improved File Handling

• With statement does "enter" and "exit" handling (similar to the finally clause):

• In the previous example, we need to remember to call `f.close()`

• Using a with statement, this is done automatically:

  ```python
  with open('huck-finn.txt', 'r') as f:
      for line in f:
          if 'Huckleberry' in line:
              print(line.strip())
  ```

• This is more important for writing files!

  ```python
  with open('output.txt', 'w') as f:
      for k, v in counts.items():
          f.write(k + ': ' + v + '
')
  ```

• Without `with`, we need `f.close()`
## 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></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>

[https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html]
Types of arguments for readers

- Indexing: choose a column to index the data, get column names from file or user
- Type inference and data conversion: automatic or user-defined
- Datetime parsing: can combine information from multiple columns
- Iterating: deal with very large files
- Unclean Data: skip rows (e.g. comments) or deal with formatted numbers (e.g. 1,000,345)
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 (',', ' ', '\t', '\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
## More read_csv/read_tables arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>skiprows</td>
<td>Number of rows at beginning of file to ignore or list of row numbers (starting from 0) to skip.</td>
</tr>
<tr>
<td>na_values</td>
<td>Sequence of values to replace with NA.</td>
</tr>
<tr>
<td>comment</td>
<td>Character(s) to split comments off the end of lines.</td>
</tr>
<tr>
<td>parse_dates</td>
<td>Attempt to parse data to datetime; False by default. If True, will attempt to parse all columns. Otherwise can specify a list of column numbers or name to parse. If element of list is tuple or list, will combine multiple columns together and parse to date (e.g., if date/time split across two columns).</td>
</tr>
<tr>
<td>keep_date_col</td>
<td>If joining columns to parse date, keep the joined columns; False by default.</td>
</tr>
<tr>
<td>converters</td>
<td>Dict containing column number of name mapping to functions (e.g., {'foo': f} would apply the function f to all values in the 'foo' column).</td>
</tr>
<tr>
<td>dayfirst</td>
<td>When parsing potentially ambiguous dates, treat as international format (e.g., 7/6/2012 -&gt; June 7, 2012); False by default.</td>
</tr>
<tr>
<td>date_parser</td>
<td>Function to use to parse dates.</td>
</tr>
<tr>
<td>nrows</td>
<td>Number of rows to read from beginning of file.</td>
</tr>
<tr>
<td>iterator</td>
<td>Return a TextParser object for reading file piecemeal.</td>
</tr>
<tr>
<td>chunksize</td>
<td>For iteration, size of file chunks.</td>
</tr>
</tbody>
</table>
Chunked Reads

• With very large files, we may not want to read the entire file
• Why?
  - Time
  - Want to understand part of data before processing all of it
• Reading only a few rows:
  - `df = pd.read_csv('example.csv', nrows=5)`
• Reading chunks:
  - Get an iterator that returns the next chunk of the file
  - `chunker = pd.read_csv('example.csv', chunksize=1000)`
  - `for piece in chunker:
    process_data(piece)`
Python csv module

- Also, can read csv files outside of pandas using csv module

```python
import csv
with open('persons_of_concern.csv', 'r') as f:
    for i in range(3):
        next(f)
    reader = csv.reader(f)
    records = [r for r in reader]  # r is a list
```

- or

```python
import csv
with open('persons_of_concern.csv', 'r') as f:
    for i in range(3):
        next(f)
    reader = csv.DictReader(f)
    records = [r for r in reader]  # r is a dict
```
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
eXtensible Markup Language (XML)

- Older, self-describing format with nesting; each field has tags
- Example:
  - <INDICATOR>
    - <INDICATOR_SEQ>373889</INDICATOR_SEQ>
    - <PARENT_SEQ></PARENT_SEQ>
    - <AGENCY_NAME>Metro-North Railroad</AGENCY_NAME>
    - <INDICATOR_NAME>Escalator Avail.</INDICATOR_NAME>
    - <PERIOD_YEAR>2011</PERIOD_YEAR>
    - <PERIOD_MONTH>12</PERIOD_MONTH>
    - <CATEGORY>Service Indicators</CATEGORY>
    - <FREQUENCY>M</FREQUENCY>
    - <YTD_TARGET>97.00</YTD_TARGET>
  - </INDICATOR>
- Top element is the root
XML

- No built-in method
- Use lxml library (also can use ElementTree)

```python
from lxml import objectify
path = 'datasets/mta_perf/Performance_MNR.xml'
parsed = objectify.parse(open(path))
root = parsed.getroot()
data = []
skip_fields = ['PARENT_SEQ', 'INDICATOR_SEQ', 'DESIRED_CHANGE', 'DECIMAL_PLACES']

for elt in root.INDICATOR:
    el_data = {}
    for child in elt.getchildren():
        if child.tag in skip_fields:
            continue
        el_data[child.tag] = child.pyval
    data.append(el_data)

perf = pd.DataFrame(data)
```

[W. McKinney, Python for Data Analysis]
JavaScript Object Notation (JSON)

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

```json
{
  "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
What is the problem with reading this data?

- [{"name": "Wes",
  "places_lived": ["United States", "Spain", "Germany"],
  "pet": null,
  "siblings": [
    {"name": "Scott", "age": 25, "pet": "Zuko"},
    {"name": "Katie", "age": 33, "pet": "Cisco"}
  ]},
  {"name": "Nia",
  "address": {"street": "143 Main",
               "city": "New York",
               "state": "New York"},
  "pet": "Fido",
  "siblings": [
    {"name": "Jacques", "age": 15, "pet": "Fido"}
  ]},
...
Reading JSON data

• Python has a built-in json module
  - with open('example.json') as f:
    data = json.load(f)
  - Can also load/dump to strings:
    • json.loads, json.dumps
• Pandas has read_json, to_json methods
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`, need to specify object
- Excel: need to specify sheet when a spreadsheet has multiple sheets
  - `pd.ExcelFile` or `pd.read_excel`
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]
Databases

• Relational databases are similar to multiple data frames but have more features
  - Links between tables via foreign keys
  - SQL to create, store, and query data

• duckdb is an OLAP database with support for python and pandas

• Python has a database API which lets you access most database systems through a common API.
import duckdb
query = """CREATE TABLE test(a VARCHAR(20), b VARCHAR(20),
c REAL, d INTEGER);"""
conn = duckdb.connect('mydata.sqlite')
conn.execute(query)
conn.commit()
# Insert some data
data = [('Atlanta', 'Georgia', 1.25, 6),
       ('Tallahassee', 'Florida', 2.6, 3),
       ('Sacramento', 'California', 1.7, 5)]
stmt = "INSERT INTO test VALUES(?, ?, ?, ?)"
conn.executemany(stmt, data)
conn.commit()
Databases

• Similar syntax from other database systems (sqlite, MySQL, Microsoft SQL Server, Oracle, etc.)

• SQLAlchemy: Python package that abstracts away differences between different database systems

• SQLAlchemy gives support for reading queries to data frame:
  
  ```python
  import sqlalchemy as sqla
  db = sqla.create_engine('sqlite:///mydata.sqlite')
  pd.read_sql('select * from test', db)
  ```
Data Cleaning
Classifying Data Quality Problems

Data Quality Problems

Single-Source Problems

Schema Level
(Lack of integrity constraints, poor schema design)
- Uniqueness
- Referential integrity...

Instance Level
(Data entry errors)
- Misspellings
- Redundancy/duplicates
- Contradictory values...

Multi-Source Problems

Schema Level
(Heterogeneous data models and schema designs)
- Naming conflicts
- Structural conflicts...

Instance Level
(Overlapping, contradicting and inconsistent data)
- Inconsistent aggregating
- Inconsistent timing...

Source type
Record
Record
Attribute
Scope/Problem Dirty Data Reasons/Remarks

Single-source problems
- Referential integrity
- Uniqueness

Examples for single-source problems at schema level (violated integrity constraints)

<table>
<thead>
<tr>
<th>Source</th>
<th>Attribute</th>
<th>Scope/Problem</th>
<th>Dirty Data</th>
<th>Reasons/Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples for single-source problems at instance level

<table>
<thead>
<tr>
<th>Source</th>
<th>Attribute</th>
<th>Scope/Problem</th>
<th>Dirty Data</th>
<th>Reasons/Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Data Quality Problems]

Classification of data quality problems in data sources

Referential constraints controlling permissible data values. For sources without schema, such as files, there are few constraints, e.g., due to data model limitations or poor integrity constraints. Schema-related data quality problems occur because of the lack of appropriate referential integrity, etc., as well as application-specific model-specific or application-specific integrity constraints.

The data quality of a source largely depends on the degree to which it is governed by schema and integrity constraints (Lack of integrity violation and/or Poor data model).

The relational approach requires simple attribute values, referential integrity, etc., as well as application-specific constraints.

Single-source problems
- Referential integrity
- Uniqueness

Examples for single-source problems at schema level (violated integrity constraints)

- Contradictory values
- Redundancy/duplicates

Instances of data quality problems occur because of the lack of appropriate uniqueness constraints at the schema level.

Database systems, on the other hand, enforce restrictions of a specific data model (e.g., the restrictions on what data can be entered and stored, giving rise to a high probability of errors and inconsistencies.}

Multi-source problems
- Inconsistent timing
- Inconsistent aggregating

Examples for multi-source problems
- Structural conflicts

Data entry errors

Misspellings

Misfielded values

Wrong references

Cryptic values, experience=
- Overlapping, contradicting and inconsistent data
- Inconsistent aggregating
- Inconsistent timing

Table 1.

<table>
<thead>
<tr>
<th>Source</th>
<th>Attribute</th>
<th>Scope/Problem</th>
<th>Dirty Data</th>
<th>Reasons/Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.

Examples for multi-source problems

- Structural conflicts

Inconsistent timing

Inconsistent aggregating

[Data Quality Problems]

Classification of data quality problems in data sources

Referential constraints controlling permissible data values. For sources without schema, such as files, there are few constraints, e.g., due to data model limitations or poor integrity constraints. Schema-related data quality problems occur because of the lack of appropriate referential integrity, etc., as well as application-specific model-specific or application-specific integrity constraints.
### Single-Source Schema Problems

<table>
<thead>
<tr>
<th>Scope/Problem</th>
<th>Dirty Data</th>
<th>Reasons/Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>bdate=30.13.70</td>
<td>values outside of domain range</td>
</tr>
<tr>
<td>Record</td>
<td>age=22, bdate=12.02.70</td>
<td>age = (current date – birth date) should hold</td>
</tr>
<tr>
<td>Record type</td>
<td>emp₁=(name=&quot;John Smith&quot;, SSN=&quot;123456&quot;)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>emp₂=(name=&quot;Peter Miller&quot;, SSN=&quot;123456&quot;)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uniqueness for SSN (social security number) violated</td>
<td></td>
</tr>
<tr>
<td>Source</td>
<td>emp=(name=&quot;John Smith&quot;, deptno=127)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>referenced department (127) not defined</td>
<td></td>
</tr>
</tbody>
</table>
## Single-Source Instance Problems

### Scope/Problem

<table>
<thead>
<tr>
<th>Scope/Problem</th>
<th>Dirty Data</th>
<th>Reasons/Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attribute</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Missing values</td>
<td>phone=9999-99999</td>
<td>unavailable values during data entry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(dummy values or null)</td>
</tr>
<tr>
<td>Misspellings</td>
<td>city=&quot;Liipzig&quot;</td>
<td>usually typos, phonetic errors</td>
</tr>
<tr>
<td>Cryptic values,</td>
<td>experience=&quot;B&quot;; occupation=&quot;DB Prog.&quot;</td>
<td></td>
</tr>
<tr>
<td>Abbreviations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Embedded values</td>
<td>name=&quot;J. Smith 12.02.70 New York&quot;</td>
<td>multiple values entered in one attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(e.g. in a free-form field)</td>
</tr>
<tr>
<td>Misfielded</td>
<td>city=&quot;Germany&quot;</td>
<td></td>
</tr>
<tr>
<td>values</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Record</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Violated attribute dependencies</td>
<td>city=&quot;Redmond&quot;, zip=77777</td>
<td>city and zip code should correspond</td>
</tr>
<tr>
<td><strong>Record type</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word transpositions</td>
<td>name$_1$=&quot;J. Smith&quot;, name$_2$=&quot;Miller P.&quot;</td>
<td>usually in a free-form field</td>
</tr>
<tr>
<td>Duplicated records</td>
<td>emp$_1=$(name=&quot;John Smith&quot;,...);</td>
<td>same employee represented twice due to</td>
</tr>
<tr>
<td></td>
<td>emp$_2=$(name=&quot;J. Smith&quot;,...)</td>
<td>some data entry errors</td>
</tr>
<tr>
<td>Contradicting records</td>
<td>emp$_1=$(name=&quot;John Smith&quot;, bdate=12.02.70);</td>
<td>the same real world entity is described by</td>
</tr>
<tr>
<td></td>
<td>emp$_2=$(name=&quot;John Smith&quot;, bdate=12.12.70)</td>
<td>different values</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wrong references</td>
<td>emp=$(name=&quot;John Smith&quot;, deptno=17)</td>
<td>referenced department (17) is defined but</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wrong</td>
</tr>
</tbody>
</table>

Given that cleaning data sources is an expensive process, preventing dirty data to be entered is obviously an important step to reduce the cleaning problem. This requires an appropriate design of the database schema and integrity constraints as well as of data entry applications. Also, the discovery of data cleaning rules during warehouse design can suggest improvements to the constraints enforced by existing schemas.

### 2.2 Multi-source problems

The problems present in single sources are aggravated when multiple sources need to be integrated. Each source may contain dirty data and the data in the sources may be represented differently, overlap or contradict. This is because the sources are typically developed, deployed and maintained independently to serve specific needs. This results in a large degree of heterogeneity w.r.t. data management systems, data models, schema designs and the actual data.

At the schema level, data model and schema design differences are to be addressed by the steps of schema translation and schema integration, respectively. The main problems w.r.t. schema design are naming and structural conflicts [2][24][17]. Naming conflicts arise when the same name is used for different objects (homonyms) or different names are used for the same object (synonyms). Structural conflicts occur in many variations and refer to different representations of the same object in different sources, e.g., attribute vs. table representation, different component structure, different data types, different integrity constraints, etc.

In addition to schema-level conflicts, many conflicts appear only at the instance level (data conflicts). All problems from the single-source case can occur with different representations in different sources (e.g., duplicated records, contradicting records,...). Furthermore, even when there are the same attribute names and data types, there may be different value representations (e.g., for marital status) or different interpretation of the values (e.g., measurement units Dollar vs. Euro) across sources. Moreover, information in the sources may be provided at different aggregation levels (e.g., sales per product vs. sales per product group) or refer to different points in time (e.g. current sales as of yesterday for source 1 vs. as of last week for source 2).

A main problem for cleaning data from multiple sources is to identify overlapping data, in particular matching records referring to the same real-world entity (e.g., customer). This problem is also referred to as the object identity problem [11], duplicate elimination or the merge/purge problem [15]. Frequently, the information is only partially redundant and the sources may complement each other by providing additional information about an entity. Thus duplicate information should be purged out and complementing information should be consolidated and merged in order to achieve a consistent view of real world entities.

#### Customer (source 1)

<table>
<thead>
<tr>
<th>CID</th>
<th>Name</th>
<th>Street</th>
<th>City</th>
<th>Sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Kristen Smith</td>
<td>2 Hurley Pl</td>
<td>South Fork, MN 48503</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>Christian Smith</td>
<td>Hurley St 2</td>
<td>S Fork MN</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Client (source 2)

<table>
<thead>
<tr>
<th>Cno</th>
<th>LastName</th>
<th>FirstName</th>
<th>Gender</th>
<th>Address</th>
<th>Phone/Fax</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>Smith</td>
<td>Christoph</td>
<td>M</td>
<td>23 Harley St, Chicago IL, 60633-2394</td>
<td>333-222-6542 / 333-222-6599</td>
</tr>
<tr>
<td>493</td>
<td>Smith</td>
<td>Kris L.</td>
<td>F</td>
<td>2 Hurley Place, South Fork MN, 48503-5998</td>
<td>444-555-6666</td>
</tr>
</tbody>
</table>

#### Customers (integrated target with cleaned data)

<table>
<thead>
<tr>
<th>No</th>
<th>LName</th>
<th>FName</th>
<th>Gender</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>ZIP</th>
<th>Phone</th>
<th>Fax</th>
<th>CID</th>
<th>Cno</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith</td>
<td>Kristen L.</td>
<td>F</td>
<td>2 Hurley Place</td>
<td>South Fork</td>
<td>MN</td>
<td>48503-5998</td>
<td>444-555-6666</td>
<td></td>
<td>11</td>
<td>493</td>
</tr>
<tr>
<td>2</td>
<td>Smith</td>
<td>Christian</td>
<td>M</td>
<td>2 Hurley Place</td>
<td>South Fork</td>
<td>MN</td>
<td>48503-5998</td>
<td></td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Smith</td>
<td>Christoph</td>
<td>M</td>
<td>23 Harley Street</td>
<td>Chicago IL</td>
<td>60633-2394</td>
<td>333-222-6542</td>
<td>333-222-6599</td>
<td>24</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

HoloClean

- A holistic data cleaning framework that combines qualitative methods with quantitative methods:
  - Qualitative: use integrity constraints or external data sources
  - Quantitative: use statistics of the data
- Driven by probabilistic inference. Users only need to provide a dataset to be cleaned and describe high-level domain specific signals.
- Can scale to large real-world dirty datasets and perform automatic repairs with high accuracy

[T. Rekatsinas et al., 2017]
Example: Input Data

(A) Input Database External Information
(Chicago food inspections)

<table>
<thead>
<tr>
<th>DBAName</th>
<th>AKAName</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1 John Veliotis Sr.</td>
<td>Johnnyo's</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
<tr>
<td>t2 John Veliotis Sr.</td>
<td>Johnnyo's</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t3 John Veliotis Sr.</td>
<td>Johnnyo's</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t4 Johnnyo's</td>
<td>Johnnyo's</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
</tbody>
</table>

(B) Functional Dependencies

c1: DBAName → Zip
c2: Zip → City, State
c3: City, State, Address → Zip

[Rekatsinas et al., 2017]
Example: Fixing via Minimality

(A) Input Database External Information
(Chicago food inspections)

<table>
<thead>
<tr>
<th>DBAName</th>
<th>AKAName</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
<tr>
<td>t2</td>
<td>John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t3</td>
<td>John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t4</td>
<td>Johnnyo's</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
</tbody>
</table>

(B) Functional Dependencies

- c1: DBAName → Zip
- c2: Zip → City, State
- c3: City, State, Address → Zip

(E) Repair using Minimality w.r.t FDs

<table>
<thead>
<tr>
<th>DBAName</th>
<th>AKAName</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t2</td>
<td>John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t3</td>
<td>John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t4</td>
<td>Johnnyo's</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
</tbody>
</table>

[T. Rekatsinas et al., 2017]
Example: Fixing via External Matches

(C) Matching Dependencies

m1: Zip = Ext_Zip \rightarrow \text{City} = \text{Ext_City}
m2: Zip = Ext_Zip \rightarrow \text{State} = \text{Ext_State}
m3: \text{City} = \text{Ext_City} \land \text{State} = \text{Ext_State} \land \text{Address} = \text{Ext_Address} \rightarrow \text{Zip} = \text{Ext_Zip}

(D) External Information
(Address listings in Chicago)

<table>
<thead>
<tr>
<th>Ext_Address</th>
<th>Ext_City</th>
<th>Ext_State</th>
<th>Ext_Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
<tr>
<td>1208 N Wells ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60610</td>
</tr>
<tr>
<td>259 E Erie ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60611</td>
</tr>
<tr>
<td>2806 W Cermak Rd</td>
<td>Chicago</td>
<td>IL</td>
<td>60623</td>
</tr>
</tbody>
</table>

(F) Repair using Matching Dependencies

<table>
<thead>
<tr>
<th>DBName</th>
<th>AKAName</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1 John Veliotis Sr.</td>
<td>Johnnyo's</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
<tr>
<td>t2 John Veliotis Sr.</td>
<td>Johnnyo's</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t3 John Veliotis Sr.</td>
<td>Johnnyo's</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t4 Johnnyo's</td>
<td>Johnnyo's</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
</tbody>
</table>

T. Rekatsinas et al., 2017
Example: Fixing via Statistics

(A) Input Database External Information
(Chicago food inspections)

<table>
<thead>
<tr>
<th>DBAName</th>
<th>AKAName</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
<tr>
<td>t2</td>
<td>John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t3</td>
<td>John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t4</td>
<td>Johnnyo's</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
</tbody>
</table>

Conflicts due to c2

(G) Repair that leverages Quantitative Statistics

<table>
<thead>
<tr>
<th>DBAName</th>
<th>AKAName</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
<tr>
<td>t2</td>
<td>John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
<tr>
<td>t3</td>
<td>John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
<tr>
<td>t4</td>
<td>John Veliotis Sr.</td>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
</tbody>
</table>

| D. Koop, CSCI 680/490, Spring 2022 | [T. Rekatsinas et al., 2017] | Conflicts due to c2 |

Does not obey data distribution
HoloClean

Input

Dataset to be cleaned

<table>
<thead>
<tr>
<th>DBAName</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>John Veliotis Sr.</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
<tr>
<td>t2</td>
<td>John Veliotis Sr.</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t3</td>
<td>John Veliotis Sr.</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t4</td>
<td>Johnny’s</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
</tbody>
</table>

Denial Constraints

1: DBAName → Zip
2: Zip → City, State
3: City, State, Address → Zip

Matching Dependencies

x1: Zip = Ext.Zip → City = Ext.City
x2: Zip = Ext.Zip → State = Ext.State
x3: City = Ext.City ∧ State = Ext.State
x4: Address = Ext.Address → Zip = Ext.Zip

External Information

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3465 S Morgan ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
<tr>
<td>1366 N Wells ST</td>
<td>Chicago</td>
<td>IL</td>
<td>60610</td>
</tr>
<tr>
<td>2900 W Grand Ave</td>
<td>Chicago</td>
<td>IL</td>
<td>60637</td>
</tr>
<tr>
<td>2626 N Carmen St</td>
<td>Chicago</td>
<td>IL</td>
<td>60659</td>
</tr>
</tbody>
</table>

The HoloClean Framework

1. Error detection module

2. Automatic compilation to a probabilistic graphical model

3. Repair via statistical learning and inference

Output

Proposed Cleaned Dataset

<table>
<thead>
<tr>
<th>DBAName</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>John Veliotis Sr.</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
<tr>
<td>t2</td>
<td>John Veliotis Sr.</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t3</td>
<td>John Veliotis Sr.</td>
<td>Chicago</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>t4</td>
<td>Johnny’s</td>
<td>Chicago</td>
<td>IL</td>
<td>60608</td>
</tr>
</tbody>
</table>

Marginal Distribution of Cell Assignments

<table>
<thead>
<tr>
<th>Cell</th>
<th>Possible Values</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2.Zip</td>
<td>60608</td>
<td>0.84</td>
</tr>
<tr>
<td></td>
<td>60609</td>
<td>0.16</td>
</tr>
<tr>
<td>t4.City</td>
<td>Chicago</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>Cicago</td>
<td>0.05</td>
</tr>
<tr>
<td>t4.DBAName</td>
<td>John Veliotis Sr.</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>Johnny’s</td>
<td>0.01</td>
</tr>
</tbody>
</table>

[T. Rekatsinas et al., 2017]
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 **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

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

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

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

In [58]: data
Out[58]:
   food   ounces  animal
0  bacon      4.0     pig
1  pulled pork  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?

```
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
```
```
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

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

<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()
Out[98]:

<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: Genres in movies

```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)
  ```python
  s = "12,13, 14"
  s.index(',',') # 2
  s.index(':') # ValueError raised
  ```

• `find(<str>)`: same as `index` but -1 if not found
  ```python
  s.find(',') # 2
  s.find(':') # -1
  ```

• `startswith() / endswith()`: boolean checks for string occurrence
  ```python
  s.startswith("1") # True
  s.endswtih("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

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.

The re module functions fall into three categories: pattern matching, substitution, and splitting. Naturally these are all related; a regex describes a pattern to locate in the text, which can then be used for many purposes. Let's look at a simple example:
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_]\n  - [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:]
```