DSC 201: Data Analysis & Visualization

Data Cleaning

Dr. David Koop
Sorting by Value (sort_values)

- **sort_values** method on series
  - `obj.sort_values()`

- Missing values (NaN) are at the end by default (na_position controls, can be first)

- **sort_values** on DataFrame:
  - `df.sort_values(<list-of-columns>)`
  - `df.sort_values(by=['a', 'b'])`
  - Can also use `axis=1` to sort by index labels
Statistics

- **sum**: column sums (\texttt{axis=1} gives sums over rows)
- missing values are excluded unless the whole slice is \texttt{NaN}
- \texttt{idxmax}, \texttt{idxmin} are like \texttt{argmax}, \texttt{argmin} (return index)
- \texttt{describe}: shortcut for easy stats!

```python
In [204]: df.describe()
Out[204]:
          one       two
count  3.000000  2.000000
mean   3.083333 -2.900000
std    3.493685  2.262742
min    0.750000 -4.500000
25%    1.075000 -3.700000
50%    1.400000 -2.900000
75%    4.250000 -2.100000
max    7.100000 -1.300000
```

On non-numeric data, `describe` produces alternate summary statistics:

```python
In [205]: obj = Series(['a', 'a', 'b', 'c'] * 4)
In [206]: obj.describe()
Out[206]:
count     16
unique     3
top        a
type freq  8
dtype: object
```

See Table 5-10 for a full list of summary statistics and related methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>count</strong></td>
<td>Number of non-NA values</td>
</tr>
<tr>
<td><strong>describe</strong></td>
<td>Compute set of summary statistics for Series or each DataFrame column</td>
</tr>
<tr>
<td><strong>min, max</strong></td>
<td>Compute minimum and maximum values</td>
</tr>
<tr>
<td><strong>argmin, argmax</strong></td>
<td>Compute index locations (integers) at which minimum or maximum value obtained, respectively</td>
</tr>
<tr>
<td><strong>idxmin, idxmax</strong></td>
<td>Compute index values at which minimum or maximum value obtained, respectively</td>
</tr>
<tr>
<td><strong>quantile</strong></td>
<td>Compute sample quantile ranging from 0 to 1</td>
</tr>
<tr>
<td><strong>sum</strong></td>
<td>Sum of values</td>
</tr>
<tr>
<td><strong>mean</strong></td>
<td>Mean of values</td>
</tr>
<tr>
<td><strong>median</strong></td>
<td>Arithmetic median (50% quantile) of values</td>
</tr>
<tr>
<td><strong>mad</strong></td>
<td>Mean absolute deviation from mean value</td>
</tr>
<tr>
<td><strong>var</strong></td>
<td>Sample variance of values</td>
</tr>
<tr>
<td><strong>std</strong></td>
<td>Sample standard deviation of values</td>
</tr>
</tbody>
</table>
Reading and Writing Data with Python

• With statement allows "enter" and "exit" handling (kind of like 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 + '\n')
  ```

• Without `with`, we need `f.close()`
## Reading & Writing Data in Pandas

<table>
<thead>
<tr>
<th>Format Type</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>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>binary</td>
<td>MS Excel</td>
<td>read_excel</td>
<td>to_excel</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>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>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 Big Query</td>
<td>read_gbq</td>
<td>to_gbq</td>
</tr>
</tbody>
</table>

[https://pandas.pydata.org/pandas-docs/stable/io.html]
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
Assignment 4

• Soon
• Focus on Data Cleaning
• Parse the raw hurdat2.txt file using pandas
Vote Today! Be part of the next visualization!

District totals by category

189

[FiveThirtyEight, 2018]
What if your JSON/XML doesn't match a specific orientation/format?
Write your own code to create the DataFrame

• Use json library to read in the data and organize the pieces as needed

• Create the DataFrame from a list of dictionaries, etc.
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]
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
  - Interfaces in C, Java, MATLAB, etc.
  - Supports **compression**
  - 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

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

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

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

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

• sqlite3 is a simple database with built-in support in python

• Python has a database API which lets you access most database systems through a common API.
import sqlite3
query = """CREATE TABLE test(a VARCHAR(20), b VARCHAR(20), c REAL, d INTEGER);"""
con = sqlite3.connect('mydata.sqlite')
con.execute(query)
con.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(?, ?, ?, ?)"
con.executemany(stmt, data)
con.commit()
Databases

• Similar syntax from other database systems (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)
  ```
Pandas Analysis

• Lots of easy analysis:
  - `df.describe()`
  - `df["column"].sum()`

• Can plot the data from pandas:
  - `df.plot.scatter(x="price", y="numSold")`
  - `alt.Chart(df).mark_point().encode(
    x='price:Q',
    y='numSold:Q')`

• Can pass data to machine learning tools like scikit-learn
... but what if data isn't correct/trustworthy/in the right format?
Dirty Data
Geolocation Errors

• Maxmind helps companies determine where users are located based on IP address
• "How a quiet Kansas home wound up with 600 million IP addresses and a world of trouble" [Washington Post, 2016]
Numeric Outliers

ages of employees (US)

- Median: 37
- Mean: 58.52632
- Variance: 9252.041

[J. Hellerstein via J. Canny et al.]
This takes a lot of time!

What data scientists spend the most time doing

- Building training sets: 3%
- Cleaning and organizing data: 60%
- Collecting data sets: 19%
- Mining data for patterns: 9%
- Refining algorithms: 4%
- Other: 5%

[CrowdFlower Data Science Report, 2016]
Why That’s a Problem

Simply put, data wrangling isn’t fun. It takes forever. In fact, a few years back, the New York Times estimated that up to 80% of a data scientist’s time is spent doing this sort of work.

Here, it’s necessary to point out that data cleaning is incredibly important. You can’t do the sort of work data scientists truly enjoy doing with messy data. It needs to be cleaned, labeled, and enriched before you can trust the output.

The problem here is two-fold. One: data scientists simply don’t like doing this kind of work, and, as mentioned, this kind of work takes up most of their time. We asked our respondents what was the least enjoyable part of their job.

They had this to say:

Note how those last two charts mirror each other. The things data scientists do most are the things they enjoy least. Last year, we found that respondents far prefer doing the more creative, interesting parts of their job, things like predictive analysis and mining data for patterns. That’s where the real value comes. But again, you simply can’t do that work unless the data is properly labeled. And nobody likes labeling data.

Do Data Scientists Have What They Need?

With a shortage of data scientists out there in the world, we wanted to find out if they thought they were properly supported in their job. After all, when you need more data scientists, you’ll often find a single person doing the work of several.

What’s the least enjoyable part of data science?

- Building training sets: 10%
- Cleaning and organizing data: 57%
- Collecting data sets: 21%
- Mining data for patterns: 3%
- Refining algorithms: 4%
- Other: 5%

[CrowdFlower Data Science Report, 2016]
Dirty Data: Statistician's View

• Some process produces the data
• Want a model but have non-ideal samples:
  - Distortion: some samples corrupted by a process
  - Selection bias: likelihood of a sample depends on its value
  - Left and right censorship: users come and go from scrutiny
  - Dependence: samples are not independent (e.g. social networks)
• You can add/augment models for different problems, but cannot model everything
• Trade-off between accuracy and simplicity
Dirty Data: Database Expert's View

• Got a dataset
• Some values are missing, corrupted, wrong, duplicated
• Results are absolute (relational model)
• Better answers come from improving the quality of values in the dataset
Dirty Data: Domain Expert's View

• Data doesn't look right
• Answer doesn't look right
• What happened?
• Domain experts carry an implicit model of the data they test against
• You don't always need to be a domain expert to do this
  - Can a person run 50 miles an hour?
  - Can a mountain on Earth be 50,000 feet above sea level?
  - Use common sense
Dirty Data: Data Scientist's View

• Combination of the previous three views
• 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
Where does dirty data originate?

- Source data is bad, e.g. person entered it incorrectly
- Transformations corrupt the data, e.g. certain values processed incorrectly due to a software bug
- Integration of different datasets causes problems
- Error propagation: one error is magnified
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 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

[J. Canny et al.]
Data Wrangling

- Data wrangling: transform raw data to a more meaningful format that can be better analyzed
- Data cleaning: getting rid of inaccurate data
- Data transformations: changing the data from one representation to another
- Data reshaping: reorganizing the data
- Data merging: combining two datasets
Data Cleaning
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>