From Lec 06:

Lecture 07

Structure

- Multiple Files
- More File Formats

Scope and Temporality

Faithfulness (and Missing Values)

- Demo: Mauna Loa CO2

Example

JSON: JavaScript Object Notation

A less common file format.

- Very similar to Python dictionaries
- Strict formatting ”quoting” addresses some issues in CSV/TSV
- **Self-documenting**: Can save metadata (data about the data) along with records in the same file

`To reads JSON file:`
`pd.read_json()` function, which works for most simple JSON files.

You will dive deeper into exactly how a JSON can structured in today’s notebook.

Example

JSON: JavaScript Object Notation

Berkeley covid cases by day

A less common file format.

- Very similar to Python dictionaries
- Strict formatting ”quoting” addresses some issues in CSV/TSV
- **Self-documenting**: Can save metadata (data about the data) along with records in the same file

Issues

- Not rectangular
- Each record can have different fields
- Nesting means records can contain tables – complicated

Reading a JSON into pandas often requires some EDA.

JSON File

1. **JSON** (JavaScript Object Notation) is a lightweight data-interchange format that machines can parse and generate easily.
2. **Use**: for data storage and exchange between a server and a web application, as well as for configuration files and data serialization.
3. **Syntax:** JSON data is represented as key-value pairs.

Keys are strings enclosed in double quotes ("), and values can be strings, numbers, objects, arrays, Boolean values (true or false), null, or nested JSON objects.

4. **file extension**: .json

JSON File: Example

```
{
    "name": "John",
    "age": 30,
    "isStudent": false,
    "courses": ["Math", "Science"],
    "address": {
        "street": "123 Main St",
        "city": "Cityville"
    }
}
```

- Most programming languages have libraries or built-in support for parsing and generating JSON data.
- **Compact and Efficient:** JSON is relatively compact and efficient for data transmission and storage, making it suitable for various use cases, including mobile applications.
- **Common Use Cases:** JSON is used in a wide range of applications, including web development (for AJAX requests and data storage), configuration files (e.g., package.json in Node.js projects), and as an interchange format in APIs.
- **Support for Nested Data:** JSON allows for nested data structures, which can represent complex relationships and hierarchies.

JSON File: Reading

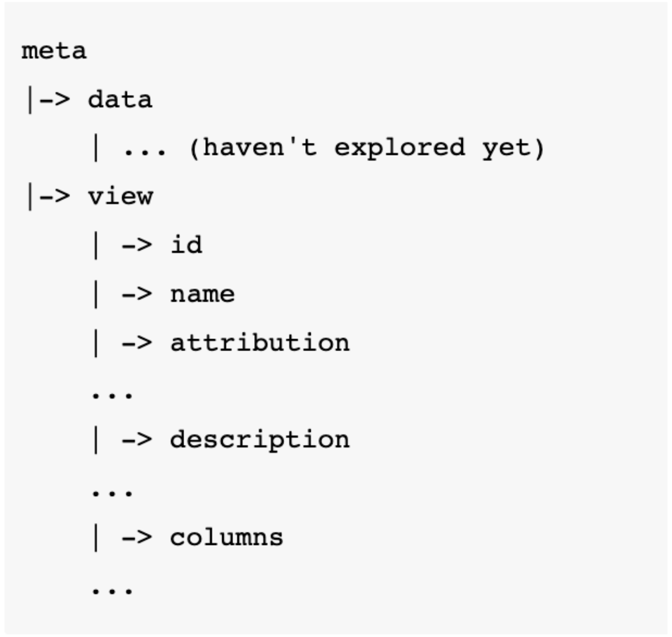
```
import json
with open(covid_file, "rb") as f:
    covid_json = json.load(f)
```

1. type(covid_json)
2. For Associated Keys in dictionary:

   covid_json.keys()

   Output: dict_keys(['meta', 'data’])
3. covid_json['meta'].keys()

   Output:dict_keys(['view'])



```
meta
|-> data
    | ... (haven't explored yet)
|-> view
    | -> id
    | -> name
    | -> attribution
    ...
    | -> description
    ...
    | -> columns
    ...
```

JSON File: Reading

```
covid_json['meta']['view'].keys()
```

```
output
dict_keys(['id', 'name', 'assetType', 'attribution',
'averageRating', 'category', 'createdAt', 'description',
'displayType', 'downloadCount', 'hideFromCatalog',
'hideFromDataJson', 'newBackend', 'numberOfComments', 'oid',
'provenance', 'publicationAppendEnabled', 'publicationDate',
'publicationGroup', 'publicationStage', 'rowsUpdatedAt',
'rowsUpdatedBy', 'tableId', 'totalTimesRated', 'viewCount',
'viewLastModified', 'viewType', 'approvals', 'clientContext',
'columns', 'grants', 'metadata', 'owner', 'query', 'rights',
'tableAuthor', 'tags', 'flags'])
```

JSON File: Reading [1/2]

```
covid_json['meta']['view']['columns’]
```

```
{'id': -1, 'name': 'sid', 'dataTypeName': 'meta_data',
'fieldName': ':sid', 'position': 0, 'renderTypeName':
'meta_data', 'format': {}, 'flags': ['hidden']}
```

JSON File: Reading [2/2]

```
covid_json['meta']['view']['columns’]
```

```
{'id': 542388893, 'name': 'New Cases', 'dataTypeName': 'number',
'description': 'Total number of new cases reported by date created in
CalREDIE. ', 'fieldName': 'bklhj_newcases', 'position': 2,
'renderTypeName': 'number', 'tableColumnId': 98765830,
'cachedContents': {'non_null': '1387', 'largest': '326', 'null': '0',
'top': [{'item': '0', 'count': '144'}, {'item': '1', 'count': '99'},
{'item': '2', 'count': '88'}, {'item': '4', 'count': '87'}, {'item':
'3', 'count': '86'}, {'item': '5', 'count': '65'}, {'item': '6',
'count': '62'}, {'item': '7', 'count': '54'}, {'item': '8', 'count':
'45'}, {'item': '11', 'count': '40'}, {'item': '9', 'count': '40'},
{'item': '12', 'count': '36'}, {'item': '13', 'count': '34'}, {'item':
'10', 'count': '34'}, {'item': '16', 'count': '24'}, {'item': '17',
'count': '23'}, {'item': '14', 'count': '23'}, {'item': '19', 'count':
'22'}, {'item': '18', 'count': '21'}, {'item': '15', 'count': '21'}],
'smallest': '0', 'count': '1387', 'cardinality': '114'}, 'format': {}}
```

Example: Calls data

- Looks like there are three columns with dates/times: EVENTDT, EVENTTM, and InDbDate.
- Most likely, EVENTDT stands for the date when the event took place
- EVENTTM stands for the time of day the event took place (in 24-hr format)
- InDbDate is the date this call is recorded on the database.

```
calls["EVENTDT"] = pd.to_datetime(calls["EVENTDT"])
calls["EVENTDT"].dt.month
calls["EVENTDT"].dt.dayofweek
```

Demo: Mauna Loa CO2

Lecture 07

Structure

- Multiple Files
- More File Formats

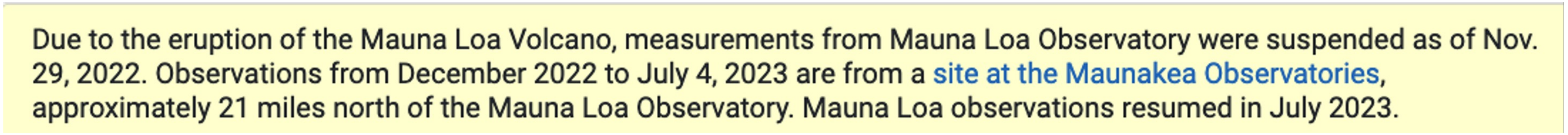
Scope and Temporality

Faithfulness (and Missing Values)

- **Example: Mauna Loa CO2**

Aside: An update to the Mauna Loa Dataset

https://gml.noaa.gov/ccgg/trends/data.html


Due to the eruption of the Mauna Loa Volcano, measurements from Mauna Loa Observatory were suspended as of Nov. 29, 2022. Observations from December 2022 to July 4, 2023 are from a site at the Maunakea Observatories, approximately 21 miles north of the Mauna Loa Observatory. Mauna Loa observations resumed in July 2023.

NPS

Example

What Are Our Variable Feature Types?

EDA step:
Understand what each record, each feature represents

First, **read file description**:

- All measurement variables (**`average`**, **`interpolated`**, **`trend`**) are monthly mean CO2 monthly mean mole fraction
    - i.e. monthly average CO2 ppm (parts per million)
    - Computed from daily means
- **`#days`**: Number of daily means in a month (i.e., # days equipment worked)

What variables define the first three columns?

- Year, month, and date in decimal

Example

The Search for the Missing Values

EDA step:
Hypothesize why these values were missing, then use that knowledge to decide whether to drop or impute missing values

From file description:

- `-99.99`: missing monthly average **`Avg`**
- `-1`: missing value for **`# days`** that the equipment was in operation that month.

Which approach?

- Drop missing values
- Keep missing values as NaN
- Impute

How should we address the missing Avg data?

Summary: Dealing with Missing Values

Mauna Loa Observatory CO2 levels (NOAA)

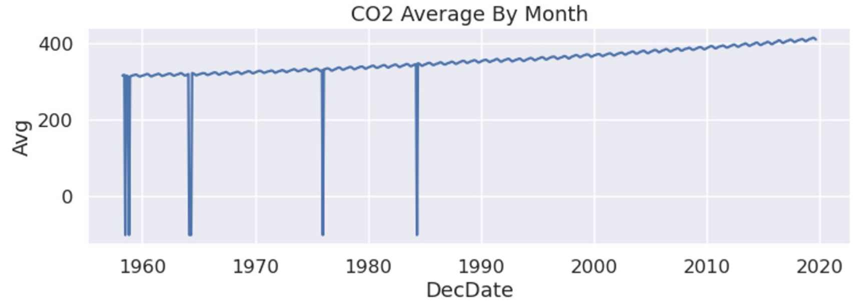
`-99.99`: missing monthly average `Avg`

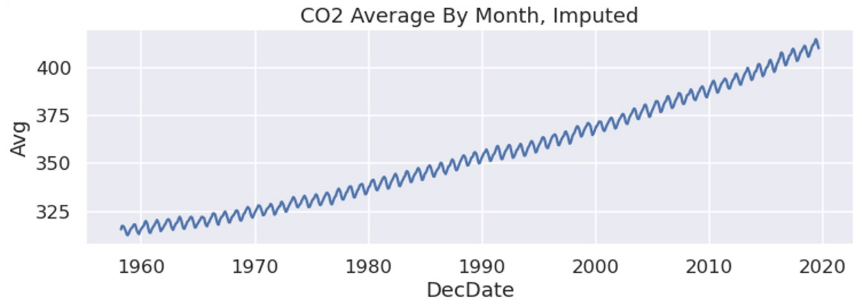
Option A: Drop records

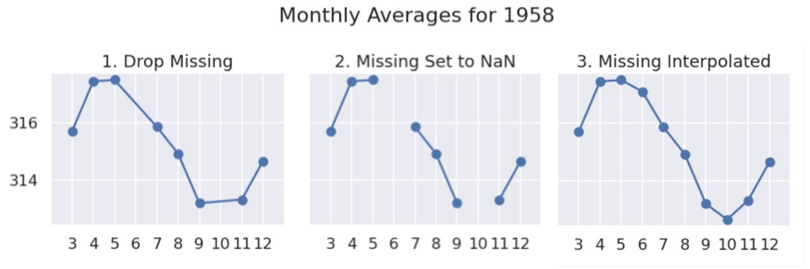
Option B: NaN missing values

Option C: **Impute** using interpolated column `Int`

All 3 are probably fine since few missing values, but we chose Option 3 based on our EDA.







With **numeric data**, you generally wrangle as you do EDA.

With **text data**, **wrangling is upfront** and requires new tools: **Python string manipulation** and **regular expressions**.

Txt File

- Note Mauna Loa CO2 data is a .txt file
- Use same pd.read_csv to read file
- Use skiprows parameter to skip rows
- Use sep = r'\s+' #delimiter for continuous whitespace (stay tuned for regex)

In this given example

- You need to visualize the monthly average CO2 concentration using sns.lineplot to check the missing values.
- Verify that all records are listed correctly using .shape
- Check the distribution for days using sns.displot
- Check the connection between missingness and the year of the recording using sns.scatterplot

Start Work on Notebook

LECTURE 7

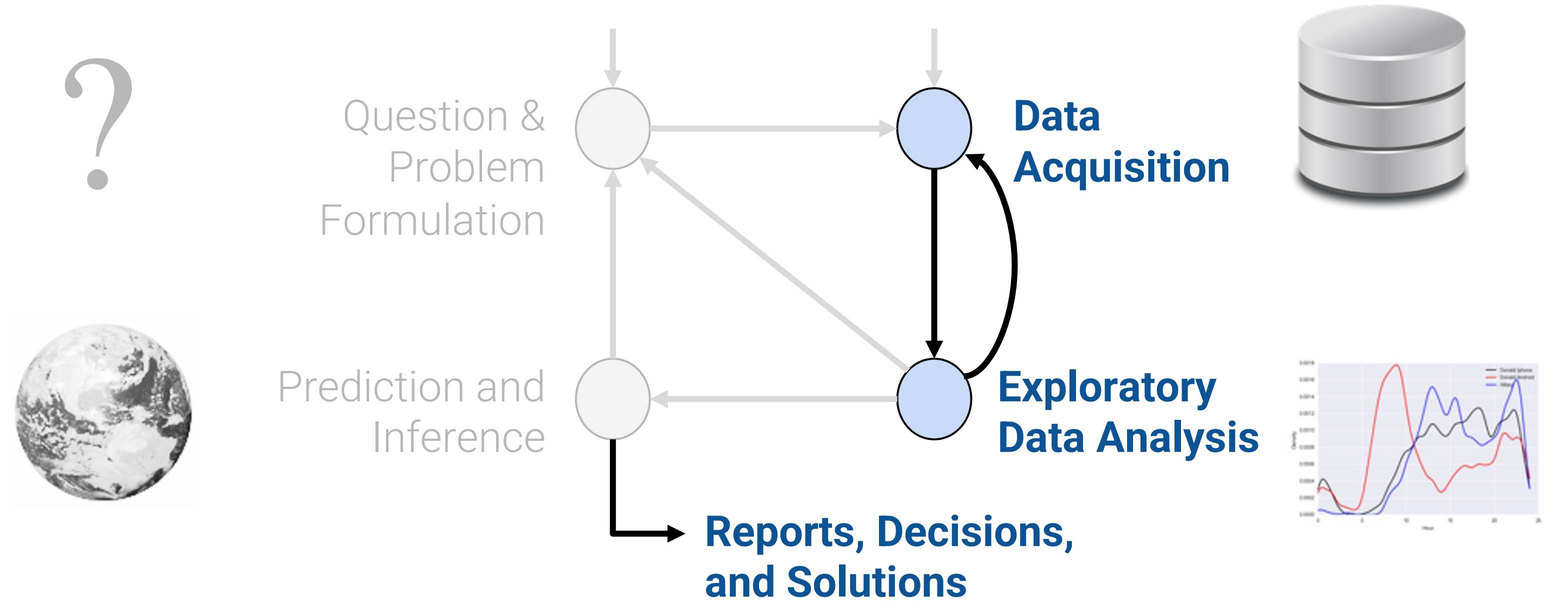
Text Wrangling and Regex

Using string methods and regular expressions (regex) to work with textual data

Data Science@ Knowledge Stream

Sana Jabbar

This Week


Question & Problem Formulation
Data Acquisition
Exploratory Data Analysis
Prediction and Inference
Reports, Decisions, and Solutions

(Last weeks)

Data Wrangling
Intro to EDA

(Today)

Working with Text Data
Regular Expressions

(Next)

Visualization
Code for plotting data

Goals for this Lecture

Lecture 07

Deal with a major challenge of EDA: cleaning text

- Operate on text data using `str` methods
- Apply regex to identify patterns in strings

Agenda

Lecture 07

- Why work with text?
- `pandas str` methods
- Why regex?
- Regex basics
- Regex functions

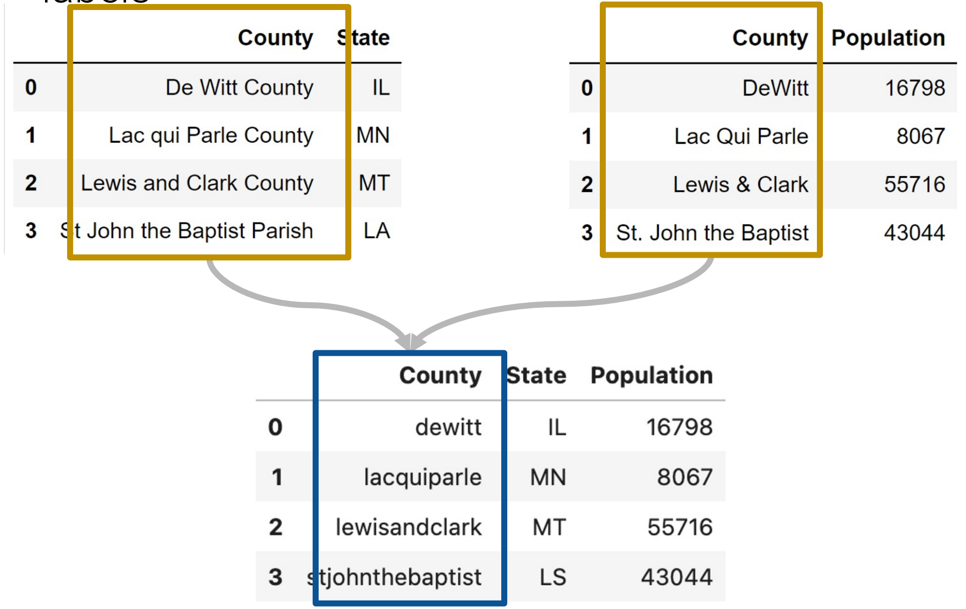
Why Work With Text?

Lecture 07

Why Work With Text? Two Common Goals

1. **Canonicalization**: Convert data that has more than one possible presentation into a standard form.

Ex   Join tables with mismatched labels



	County	State
0	De Witt County	IL
1	Lac qui Parle County	MN
2	Lewis and Clark County	MT
3	St John the Baptist Parish	LA

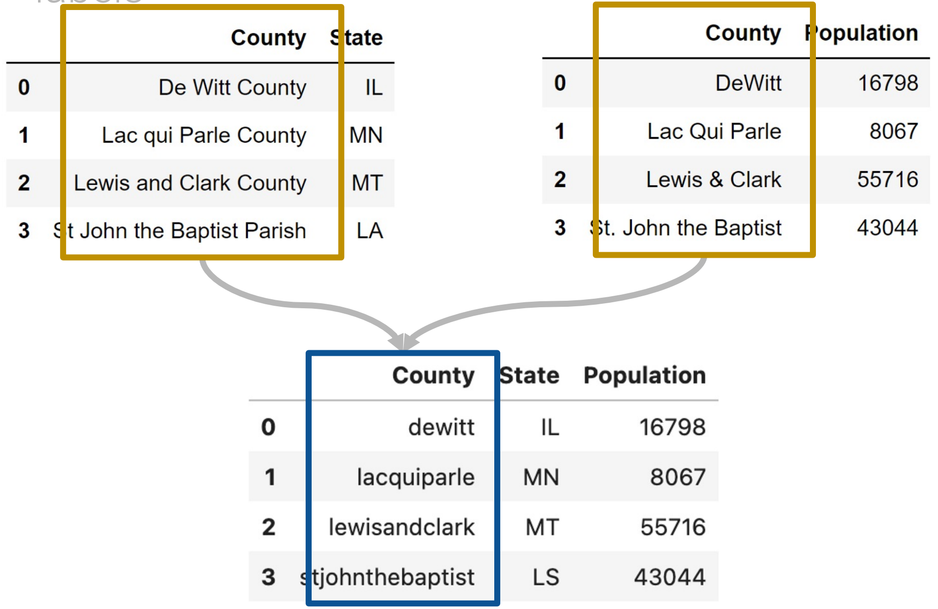
	County	Population
0	DeWitt	16798
1	Lac Qui Parle	8067
2	Lewis & Clark	55716
3	St. John the Baptist	43044

	County	State	Population
0	dewitt	IL	16798
1	lacquiparle	MN	8067
2	lewisandclark	MT	55716
3	stjohnthebaptist	LS	43044

Why Work With Text? Two Common Goals

1. **Canonicalization**: Convert data that has more than one possible presentation into a standard form.

Ex Join tables with mismatched labels



	County	State
0	De Witt County	IL
1	Lac qui Parle County	MN
2	Lewis and Clark County	MT
3	St John the Baptist Parish	LA

	County	Population
0	DeWitt	16798
1	Lac Qui Parle	8067
2	Lewis & Clark	55716
3	St. John the Baptist	43044

	County	State	Population
0	dewitt	IL	16798
1	lacquiparle	MN	8067
2	lewisandclark	MT	55716
3	stjohnthebaptist	LS	43044

2. **Extract** information into a new feature.

Ex Extract dates and times from log files

```
169.237.46.168 - -
[26/Jan/2014:10:47:58 -0800] "GET
/stat141/Winter04/ HTTP/1.1" 200 2585
"http://anson.ucdavis.edu/courses/"
```

↓

```
day, month, year = "26", "Jan", "2014"
hour, minute, seconds = "10", "47", "58"
```

`pandas str` Methods

Lecture 07

From String to `str`

In “base” Python, we have various string operations to work with text data.

Recall:

transformation	`s.lower()`
	`s.upper()`

split	`s.split(…)`
membership	`'ab' in s`

replacement/ deletion	`s.replace(…)`
substring	`s[1:4]`
length	`len(s)`

Problem: Python assumes we are working with one string at a time
Need to loop over each entry – slow in large datasets!

`str` Methods

Fortunately, pandas offers a method of **vectorizing** text operations: the `.str` operator

```
Series.str.string_operation()
```

Apply the function `string_operation` to *every* string contained in the `Series`

```
populations[“County”].str.lower()

0                   dewitt
1             lac qui parle
2             lewis & clark
3     st. john the baptist
Name: County, dtype: object
```

```
populations[“County”].str.replace('&', 'and')

0                   DeWitt
1             Lac Qui Parle
2           Lewis and Clark
3     St. John the Baptist
Name: County, dtype: object
```

`.str` Methods

Most base Python string operations have a `pandas str` equivalent

Operation	Python (single string)	pandas (`Series` of strings)

transformation	`s.lower()`
	`s.upper()`
		`ser.str.lower()`
		`ser.str.upper()`

replacement/ deletion	`s.replace(…)`	`ser.str.replace(…)`
split	`s.split(…)`	`ser.str.split(…)`
substring	`s[1:4]`	`ser.str[1:4]`
membership	`'ab' in s`	`ser.str.contains(…)`
length	`len(s)`	`ser.str.len()`

Example

Demo 1: Canonicalization

	County	State
0	De Witt County	IL
1	Lac qui Parle County	MN
2	Lewis and Clark County	MT

	County	Population
0	DeWitt	16798
1	Lac Qui Parle	8067
2	Lewis & Clark	55716
3	St. John the Baptist	43044

	County	State	Population
0	dewitt	IL	16798
1	lacquiparle	MN	8067
2	lewisandclark	MT	55716
3	stjohnthebaptist	LS	43044

```
def canonicalize_county(county_series):
 return (county_series
               .str.lower()               # lowercase
                              .str.replace(' ', '')
# remove space
                              .str.replace('&', 'and')
# replace &
                              .str.replace('.', '')
# remove dot
                          .str.replace('county', '')
                           .str.replace('parish', '')
```