Data Integration and Analysis

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics

Portions of this lecture have been contributed to the OpenDS4All project, piloted by Penn, IBM, and the Linux Foundation

https://tinyurl.com/cis545-lecture-01-26-22

Ensuring Data is Analyzable

Last time:

We found issues with our data – so we had very

little integrated data to analyze!

https://tinyurl.com/cis545-lecture-01-26-22

How Do We Know Data Is Good?

- For our purposes, we "eyeballed it"…

  e.g., we saw that the names had underscores

- Looking at data is *always* encouraged!

- But:
  - It's best to not count on humans to look at data *at scale*!
  - Our best effort won't catch everything
- Often, we want *pipelines* that run periodically

https://tinyurl.com/cis545-lecture-01-26-22

Detecting Data Errors: Potential Validation Rules

- “Names should not have non-alpha characters”
- “Birthdays should be non-null”
- “Product names must appear in the table **master_product_list**”

Beware: nearly always have **false positives** and **false negatives**

https://tinyurl.com/cis545-lecture-01-26-22

Simple Example of a Validation Rule

https://tinyurl.com/cis545-notebook-01
"Part 3: Validating and Cleaning Data"

- Perhaps names should be all alphabetic, except possibly:
  - Spaces (compound names)
  - Hyphenation (e.g., "Jean-Luc")
  - Apostrophes (e.g., "O'Malley")
  - Periods (for initials)

- So we can create a validation rule for this…

https://tinyurl.com/cis545-lecture-01-26-22

Validation Rule: Show All Bad Names...

```
replace_item = ''

failed = False
for name in exec_df['clean_name']:
  if not name.replace(' ', replace_item).\
          replace('.', replace_item).\
          replace('\'',replace_item).\
          replace('-',replace_item).isalpha
    print ("Illegal name %s" %name)
    failed = True

if failed:
  print('Found illegal names!')
```

```
Illegal name Harald Kr%C3%BCger
Illegal name Ola K%C3%A4llenius
Illegal name B%C3%B6rje Ekholm
Illegal name Michael O%27Leary
Found illegal names!
```

https://tinyurl.com/cis545-lecture-01-26-22

Cleaning the Data

We may know how to fix the errors – in this case we can map the character codes back from URL encoding:

```
from urllib.parse import unquote

exec_df['clean_name'].apply(unquote)
```

```
19                   Warren Buffett
20                      Hubert Joly
21              Sunil Bharti Mittal
22             Stephen A. Schwarzman
23                 Andrew Mackenzie
24                    Harald Krüger
```

https://tinyurl.com/cis545-lecture-01-26-22

Validation and Cleaning is Best-Effort

Lots of data errors will be impossible to spot

- e.g., a birthday that’s wrong by a year

Even knowing data is bad / incomplete doesn’t always suggest a fix

- e.g., how do I recover the full name of “J SMITH”?

https://tinyurl.com/cis545-lecture-01-26-22

The Path to Data Suitable for Analysis

- Our example required us to think carefully about the data and what it should look like
- Let's discuss:
  - Libraries and standard approaches for data validation
  - Linking data in the presence of irregularity
  - Automating data wrangling
- Then we'll take a look at simple analysis of integrated data!

https://tinyurl.com/cis545-lecture-01-26-22

General Techniques for Data Validation

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics

Penn Engineering

ODPi OpenDS4All

Portions of this lecture have been contributed to the OpenDS4All project, piloted by Penn, IBM, and the Linux Foundation

https://tinyurl.com/cis545-lecture-01-26-22

More General Data Validation

Validation typically uses rules, but we should beware exceptions:

e.g., what about celebrity names like “50 Cent”, “X Æ A-12”!

Given this caveat, let’s look at 2 general validation approaches:

- Libraries allow you to test for values, e.g., IP address, URL, email
- Or you can compare against a master list of values

https://tinyurl.com/cis545-lecture-01-26-22

Checking Values:
Python **validator** Library (One of *Many*)

```python
import validators.url

# Are all of the URLs valid?
exec_df['page'].apply(validators.url)
```

```
0        True
1        True
2        True
3        True
4        True
5        True
6        True
7        True
```

URLs, email, minimum-length strings, bank accounts, …

https://tinyurl.com/cis545-lecture-01-26-22

Validation Rules

- Lots of domain-specific rules can be captured, and are included in various data cleaning + quality tools
- Phone numbers, credit card numbers, IP addresses, street addresses, zip codes, …
- But these typically look at patterns, and are vulnerable to invalid values that look plausible

https://tinyurl.com/cis545-lecture-01-26-22

In Our Example: Company Data – Are Country Codes Correct?

```
company_data_df[['name','country_code']]
```

	name	country_code
0	#waywire	USA
1	&TV Communications	USA
2	'Rock' Your Paper	EST
3	(In)Touch Network	GBR
4	+n (PlusN)	USA
...	...	...
47753	Zzish	GBR
47754	ZZNode Science and Technology	CHN

https://tinyurl.com/cis545-lecture-01-26-22

An Alternative: Validation Against a Master List

We can che … list of all known c…

```
countries_df … .com
/lukes/ISO-3 … all.
csv")

validated = …
    left_on= …

validated[validated['_merge'] != 'both']
```

…ategory_list	market	funding_total_usd	status	country_code
|E-Commerce|	SaaS	-	operating	ROM
…elopment|Web …gn|Advertising|	Brand Marketing	4 500 000	operating	ROM
|Advertising|	Advertising	142 000 000	operating	ROM
…king|Software|	Linux	2 940 000	operating	ROM
|Software|	Software	7 000 000	operating	ROM
|Curated Web|	Software	70 000	operating	ROM

Should be ROU

https://tinyurl.com/cis545-lecture-01-26-22

Data Validation in Business

What's different about a business: CIO typically "owns" most of the data and can make it consistent

- Data *governance* – formal processes for standardizing, cleaning, archiving, auditing, etc.
- *Master data management* – building a complete warehouse of concepts and entities
- These can be useful for validation!

We'll revisit the enterprise near the end of the semester!

https://tinyurl.com/cis545-lecture-01-26-22

Recapping Data Validation and Cleaning

- Two main components:
  - Validation – detecting when there is bad data
  - Cleaning – transforming the data if it's dirty

- Validation has standard approaches and tools – though beware exceptions to every rule

Log, and periodically check, the bad (?) data!

- Cleaning requires domain expertise and can bias the data – we'll revisit this later in the semester!

https://tinyurl.com/cis545-lecture-01-26-22

Review

https://canvas.upenn.edu/courses/1636888/quizzes/2771545 (04B)

- What is the most important thing to do when you run data validation rules?
  a. Make our rules very restrictive
  b. Log and periodically inspect the data that violates the rules
  c. Make our rules very permissive
  d. Focus on performance over all else
- Why would we prefer validation against a master (data) list, vs using a data validation function?
  a. Checking against the master list is always faster
  b. The master list captures all possible domain values rather than patterns
  c. The function captures all possible domain values rather than patterns
  d. We would never want this

https://tinyurl.com/cis545-lecture-01-26-22

Record Linking

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics

Portions of this lecture have been contributed to the OpenDS4All project, piloted by Penn, IBM, and the Linux Foundation

https://tinyurl.com/cis545-lecture-01-26-22

Linking Data with Tolerance: Record Linking

- Combining data from different sources:
  (1) figure out how to clean the data into matching values, or (2) make a more tolerant version of the join

- A general problem called "record linking": looking at rows in different tables and figuring out if they should join
  - Is "SMITH, J" the same as "JON SMITH"?
  - Many tools: **Tamr, Data Ladder, QualityStage, Magellan, etc.**

- Instead of cleaning, could we do this with `Executive` vs `clean_name`?
  - We count the rows as matching *if they are similar*

https://tinyurl.com/cis545-lecture-01-26-22

Entity Matching, Deduplication, Record Linking

Variations of the *entity matching* problem:

Deduplication

- Given t_1, t_2 in table T
- Merge t_1, t_2 if they represent the same in

executives

What's hard:

How do we know when two tuples "represent the same instance"

or "are semantically linked"?

Record linking

- Given r, s from tables R, S
- Join r, s if there is a semantic link between them

tives

JF Smith	Widgets 'R Us
J SMITH	IBM

	birthday	employer
J SMITH	1/23/45	Widgets R Us

https://tinyurl.com/cis545-lecture-01-26-22

A Simple Scheme

- Let's look at the similarity of (some of) the columns
  - And use that as *evidence* to predict if tuples should be linked

- For example: CEO names in the tuples

https://tinyurl.com/cis545-lecture-01-26-22

A Deep Dive: String Similarity

String equality is really easy to test:

```
def str_equal(x,y):
  if len(x) != len(y):
      return False
  for i in range(0, len(x)):
      if x[i] != y[i]:
          return False
  return True
```

How might we generalize to "approximately equal"?

https://tinyurl.com/cis545-lecture-01-26-22

Two Main Approaches to String Similarity

String **edit distance**:

- How many edits (char insert, delete, replace) do we need to make string x into string y?

  robot -> bot with 2 edits

- Requires *dynamic programming*; we may revisit later in the term

String **overlap**:

- How much is **in common** between the two strings?
- Take all **substrings of length q**, sometimes called **q-grams** or occasionally **n-grams**
- Compute **Jaccard similarity:** $|qgrams(x) \cap qgrams(y)| / |qgrams(x) \cup qgrams(y)|$

https://tinyurl.com/cis545-lecture-01-26-22

What's a Q-Gram, Exactly?

Typically we will "pad" the ends of the string by (q-1) characters, e.g., for 3-grams on "robot" and "bot":

`robot □ $$robot$$` `bot □ $$bot$$`

$$robot$$	$$bot$$
$$r	$$b
$ro	$bo
rob	bot
obo	ot$
bot	t$$
ot$	
t$$	

3 q-grams in common
9 total q-grams

Jaccard score = 3/9 = 0.333

https://tinyurl.com/cis545-lecture-01-26-22

String Similarity in Record Linking

- From the **Magellan** Python record linking tool set:

```
tok = sm.QgramTokenizer(qval=5,return_set=True)
output_pairs = ssj.jaccard_join(company_ceos_df, exec_df,
              'index', 'page', 'Executive', 'clean_name', tok, 0.35,
              l_out_attrs=['Executive'], r_out_attrs=['name'])
```

	_id	l_index	r_page	l_Executive	r_name	_sim_score
24	24	24	https://en.wikipedia.org/wiki/Harald_Kr%C3%BCger	Harald Krüger	Harald_Kr%C3%BCger	0.444444
41	41	41	https://en.wikipedia.org/wiki/Ola_K%C3%A4llenius	Ola Källenius	Ola_K%C3%A4llenius	0.444444
51	51	51	https://en.wikipedia.org/wiki/B%C3%B6rje_Ekholm	Börje Ekholm	B%C3%B6rje_Ekholm	0.423077
127	127	127	https://en.wikipedia.org/wiki/Michael_O%27Leary	Michael O'Leary	Michael_O%27Leary	0.538462

https://tinyurl.com/cis545-lecture-01-26-22

Recap of Record Linking

- Record linking and deduplication are similar
- Both require *similarity measures* to determine if tuples are likely about the same instance
- A common approach is *q-grams* and *Jaccard similarity*

https://tinyurl.com/cis545-lecture-01-26-22

Quick Review

https://canvas.upenn.edu/courses/1636888/quizzes/2771572 (04C)

- Which of the following are effective general approaches to computing string similarity? Pick the best answer.
  a. string containment
  b. string edit distance only
  c. string edit distance and string overlap
  d. string similarity only
- How many $ characters would we use to pad the start and end of a 5-gram?
  a. 5
  b. 4
  c. 3
  d. 1

https://tinyurl.com/cis545-lecture-01-26-22

ETL: Automated Data Wrangling

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics

Portions of this lecture have been contributed to the OpenDS4All project, piloted by Penn, IBM, and the Linux Foundation

https://tinyurl.com/cis545-lecture-01-26-22

Automating Data Wrangling

- This class focuses on *interactive* data analysis – our wrangling is often one-off

- In practice, we often take our wrangling code and *automate it*!
  - Every day, import the latest data
  - When an email comes in with an attachment, load it

- This involves "ETL pipelines" or "ETL workflows"

https://tinyurl.com/cis545-lecture-01-26-22

Extract-Transform-Load

- Big data-scale tools for wrangling are often called ETL tools:
  - Extract (and acquire) data
  - Transform the data
  - Load the data into a database

- All of the big DBMS companies, and many others (e.g., Trifacta), work in this space

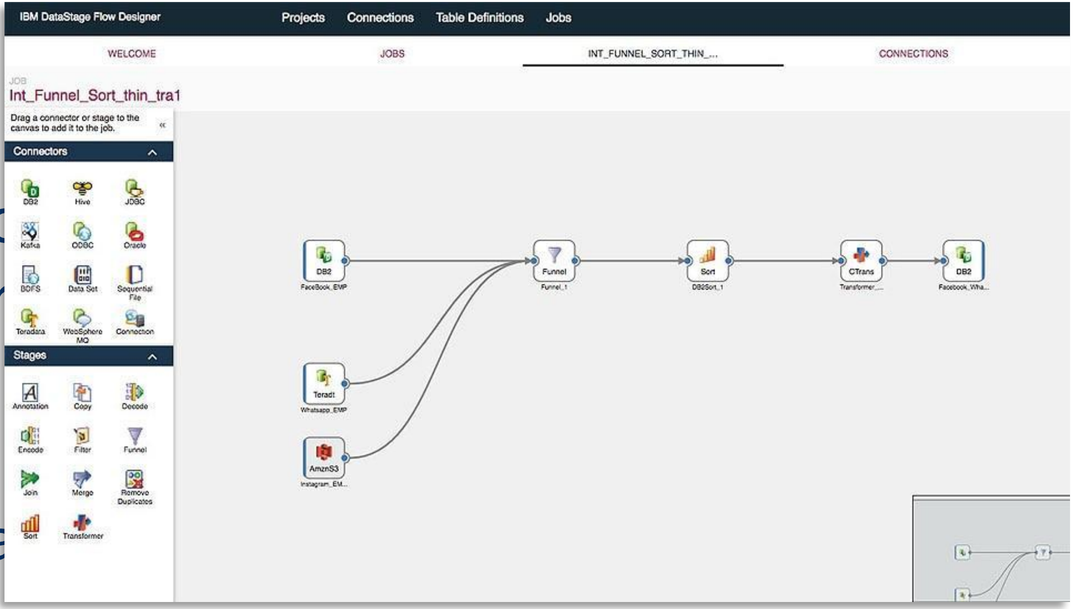
https://tinyurl.com/cis545-lecture-01-26-22

What Exactly Is ETL?

https://mp.s81c.com/pwb-production/032153a69496bc39b44bd8bb8da3de6f/offering_f90cd5e4-7502-442c-807c-e914baf5fdca.jpg

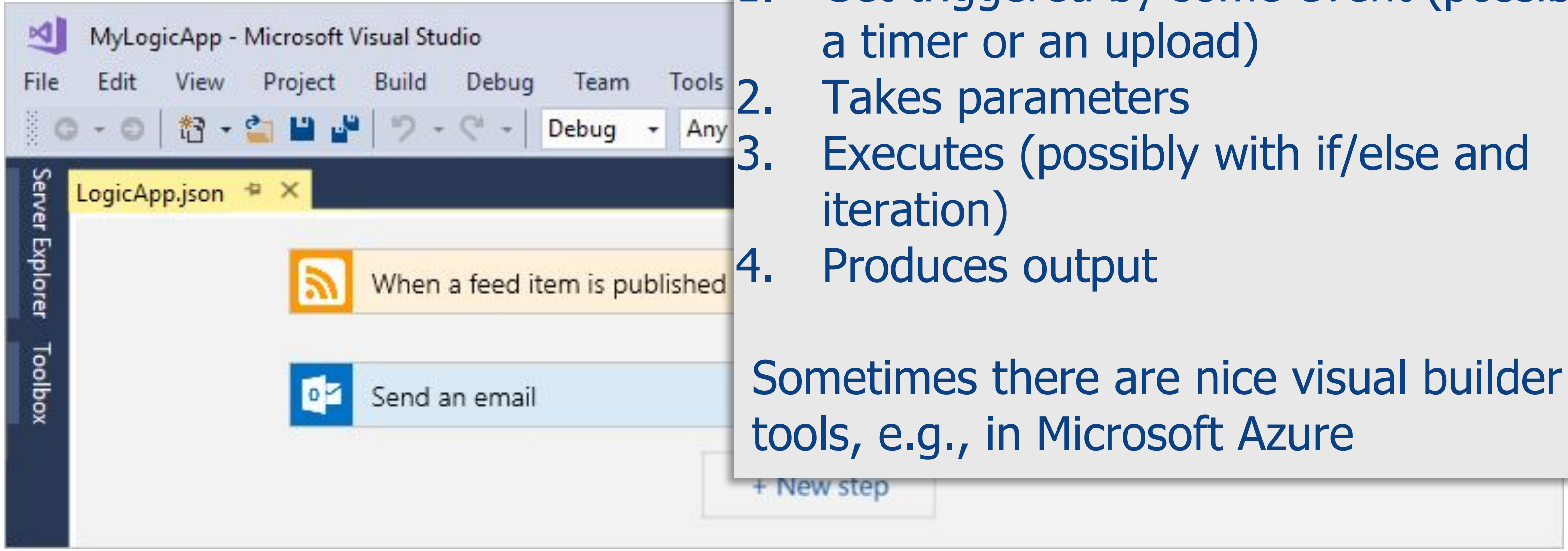
A sequence of data-c
a script, which perfor

- In Pandas, or SQL,
- They're a special ca



https://tinyurl.com/cis545-lecture-01-26-22

Workflows



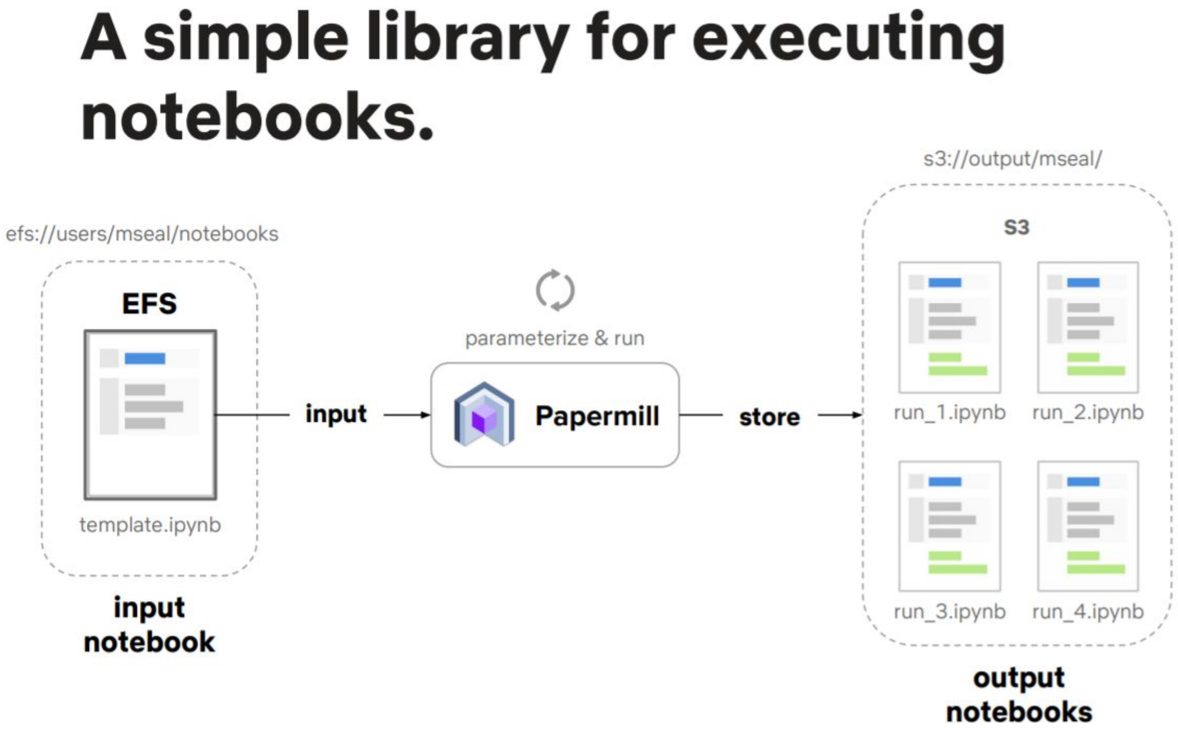
Some notion of scripts, which:

1. Get triggered by some *event* (possibly a timer or an upload)
2. Takes parameters
3. Executes (possibly with if/else and iteration)
4. Produces output

Sometimes there are nice visual builder tools, e.g., in Microsoft Azure

https://tinyurl.com/cis545-lecture-01-26-22

Papermill: Jupyter Notebooks as Workflows!

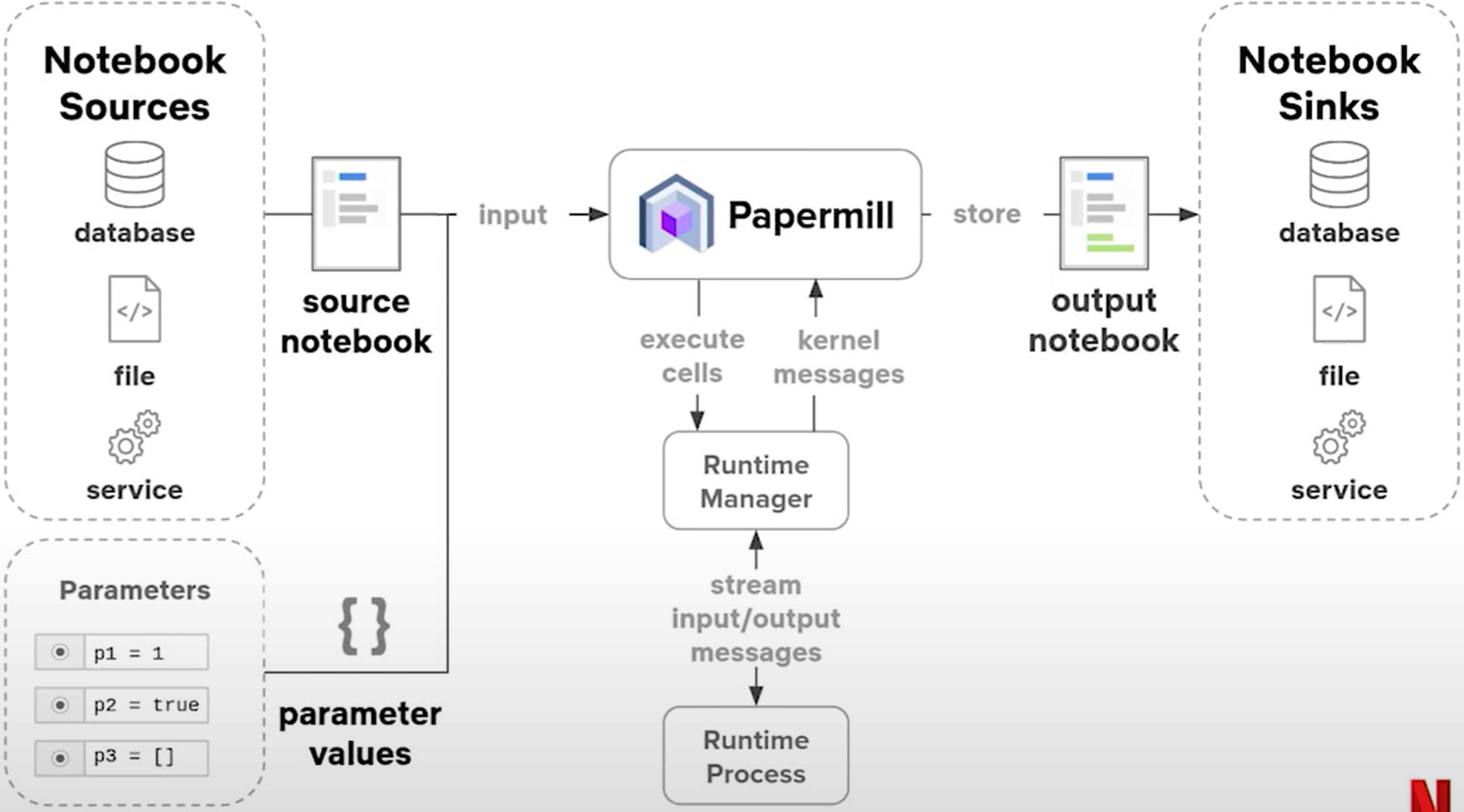


https://www.datacouncil.ai/hubfs/DataEngConf/Data%20Council/Slides%20SF%2019/Notebooks%20as%20Functions%20with%20papermill.pdf

https://tinyurl.com/cis545-lecture-01-26-22

How it works a bit more.

- Reads from a source
- Injects parameters
- Launches a runtime manager + kernel
- Sends / Receives messages
- Outputs to a destination



Review

https://canvas.upenn.edu/courses/1636888/quizzes/2771508 (04D)

- Which of the following could be used to implement the T stage in ETL?
  a. all of these answers
  b. an SQL script
  c. a Perl script
  d. a call to a web service
- An ETL workflow is most useful for
  a. automating data wrangling
  b. automating interactive discovery
  c. finding relevant data sources
  d. enforcing firewall rules

https://tinyurl.com/cis545-lecture-01-26-22

Summary of Data Wrangling and Integration

- Data *wrangling* involves getting the data into structured form
- Data *validation and cleaning* are essential to fixing errors, missing data, etc.
- Even with extensive cleaning, we may still need to use approximate matching techniques to do *record linking*
- All of these steps can be automated in workflows

https://tinyurl.com/cis545-lecture-01-26-22

ETL, Summarized

- A system of tools to extract, transform, and load data in an automated way
- Typically run as part of *workflows* that have a *trigger* and a set of parameters
- Allows us to "script" the wrangling process

https://tinyurl.com/cis545-lecture-01-26-22

Simple Data Analysis

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics

Portions of this lecture have been contributed to the OpenDS4All project, piloted by Penn, IBM, and the Linux Foundation

https://tinyurl.com/cis545-lecture-01-26-22

A Long Journey towards Data Analysis!

- Transform tabular data (select, project, apply)
- Join tables via exact-match
- Clean and link tables

- Let's look at data analysis in Pandas and SQL – using grouping and aggregation!

https://tinyurl.com/cis545-lecture-01-26-22

Grouping

Grouping: "bins" subsets of rows based on common values

born	Company	Executive
1923-05-27 00:00:00	National Amusements	Sumner Redstone
1924-02-12 00:00:00	Accenture	David Rowland

```
total.groupby(by='born').get_group\
     (datetime.datetime.strptime('1945-08-24', '%Y-%m-%d'))
```

	Company	Executive	born
180	WWE	Vince McMahon	1945-08-24 00:00:00

(for many datasets there would be more rows here)

https://tinyurl.com/cis545-lecture-01-26-22

Grouping and Aggregation

- Most commonly, we want to apply a function to (some of) the columns in a group, e.g., a count:

```
total.groupby(by='born').count()
```

Groups by 'born,' excluding the NaNs

born	index	Company	Executive	Title	Since	Notes
1923-05-27	1	1	1	1	1	1
1924-02-12	1	1	1	1	1	1
1929-03-07	1	1	1	1	1	1
1930-08-30	1	1	1	1	1	1

```
pd.read_sql('select born,
count(Company) from total_info ' +\
            'where born is not null
group by born', conn)
```

https://tinyurl.com/cis545-lecture-01-26-22

Aggregating and Counting by Decade

```
bdays = total[['born']].dropna()
bdays = bdays.applymap(lambda bday: str(int(bday.year / 10) * 10) + ‘s’)
exec_df[['born_decade']] = bdays
```

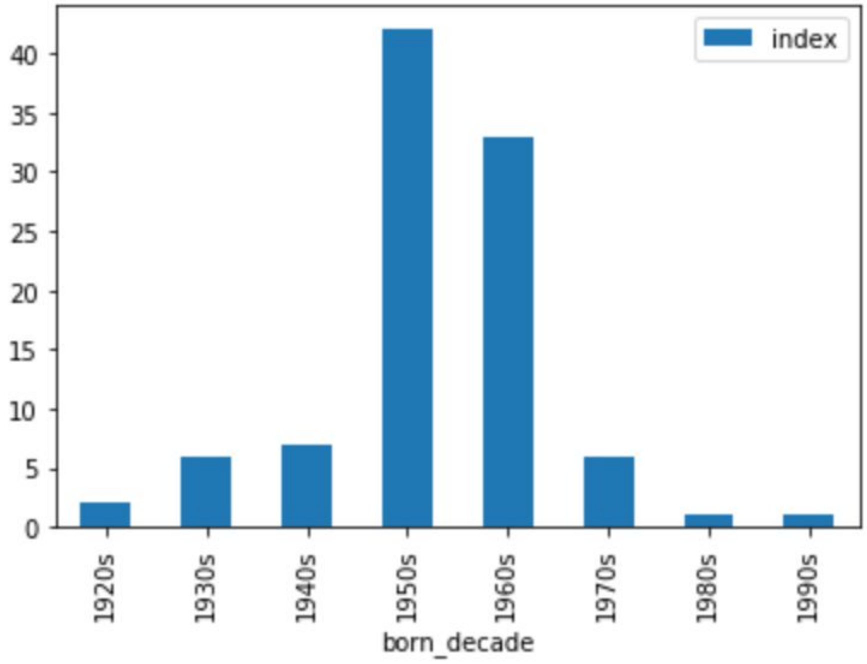
- Now let’s get a sense of the age of CEOs by calculating the decade in which they were born.
- Then we will plot a histogram, showing how many were born in each decade.

https://tinyurl.com/cis545-lecture-01-26-22

Plotting by Decade

```
# index and born_decade
df = exec_df[['born_decade’]].
  reset_index()

df.groupby('born_decade’).
  count().plot(kind='bar')
```



https://tinyurl.com/cis545-lecture-01-26-22

Summary of Grouping

- Grouping “bins” sets of tuples
- You can apply aggregate functions such as count or sum to the columns
- You can use **plot** with an x and y to visualize the results

- Beware of how null values are treated!

https://tinyurl.com/cis545-lecture-01-26-22

Review

https://canvas.upenn.edu/courses/1636888/quizzes/2771589 (04E)

- Suppose you have a dataframe df with columns name, street, city. If you use Pandas to group on city, and Philadelphia has 10 entries -- what will the output be for the Philadelphia row, in the column street, if you do a count() on the group?
  a. A street address
  b. NaN
  c. The value 10
  d. The value Philadelphia
- If you want to plot a dataframe, it needs a minimum of
  a. three columns
  b. the dataframe index and a column with values
  c. four columns
  d. the dataframe index without columns

https://tinyurl.com/cis545-lecture-01-26-22

Module Recap

We've gone on a journey from data acquisition

- to transformation
- to cleaning and linking
- to simple analysis

and learned the basic operations in both Pandas and SQL

Before we wrap up our data integration segment of the semester, we should talk about what happens when our data is *text* – next time!

https://tinyurl.com/cis545-lecture-01-26-22