COMP5339: Data Engineering

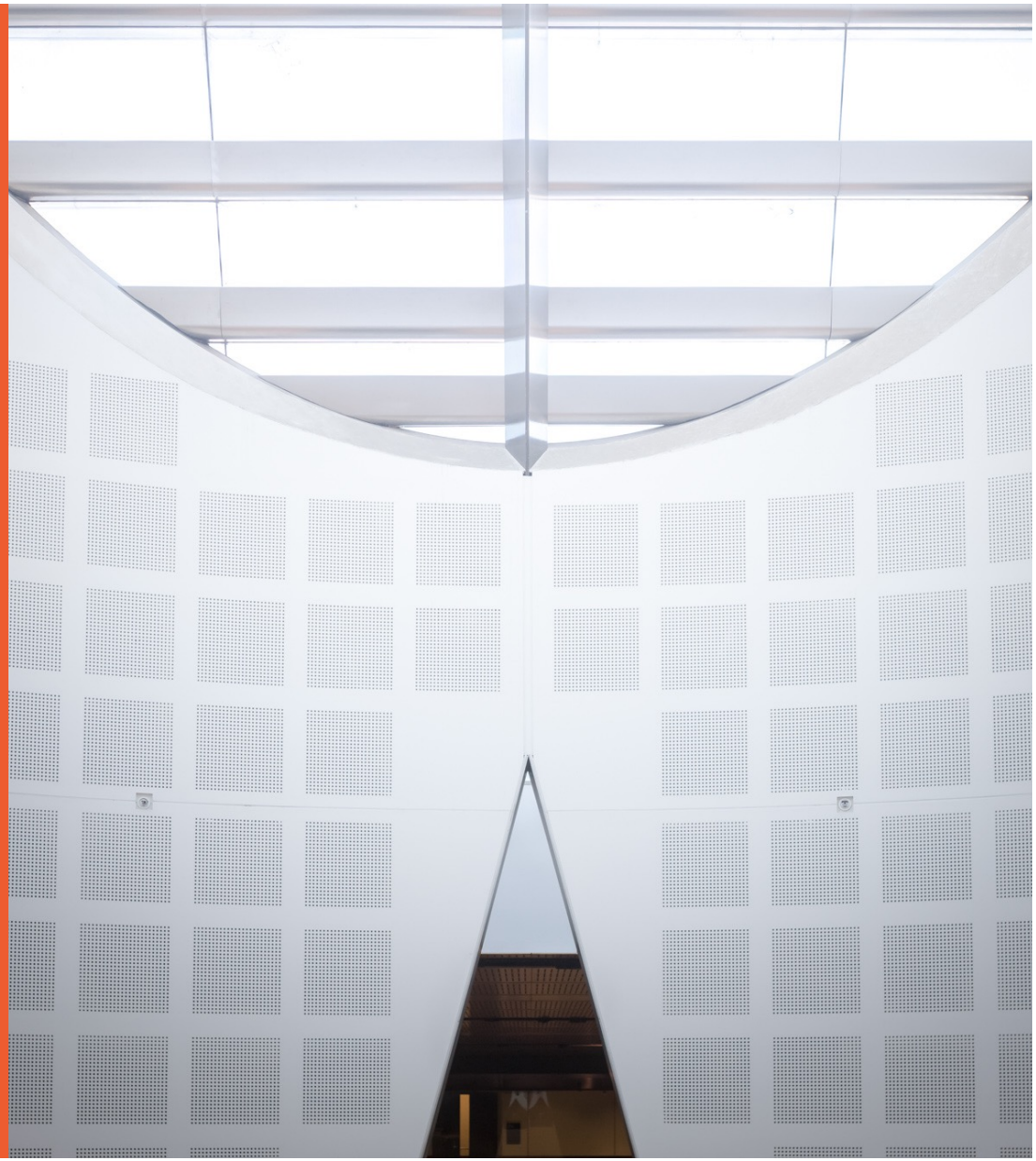
W2: Data Acquisition and Cleaning

Presented by

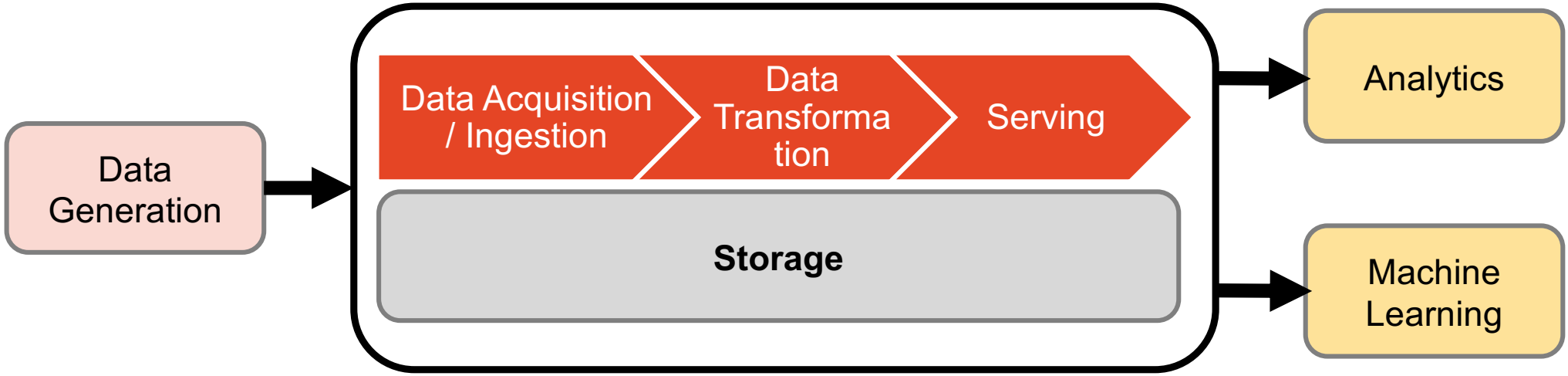
Uwe Roehm

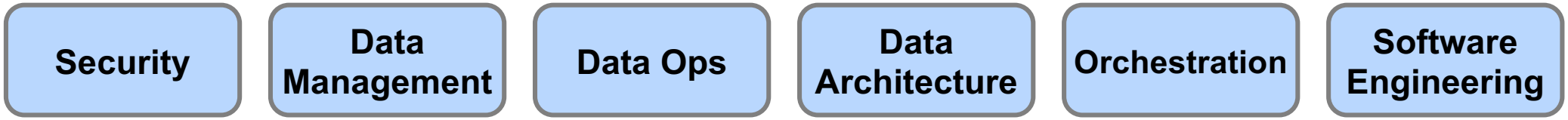
School of Computer Science

THE UNIVERSITY OF SYDNEY



Typical Data Pipeline





[J. Reis and M. Housley: Fundamentals of Data Engineering, 2022]

Data Acquisition Overview

Data Acquisition – Where does data come from?

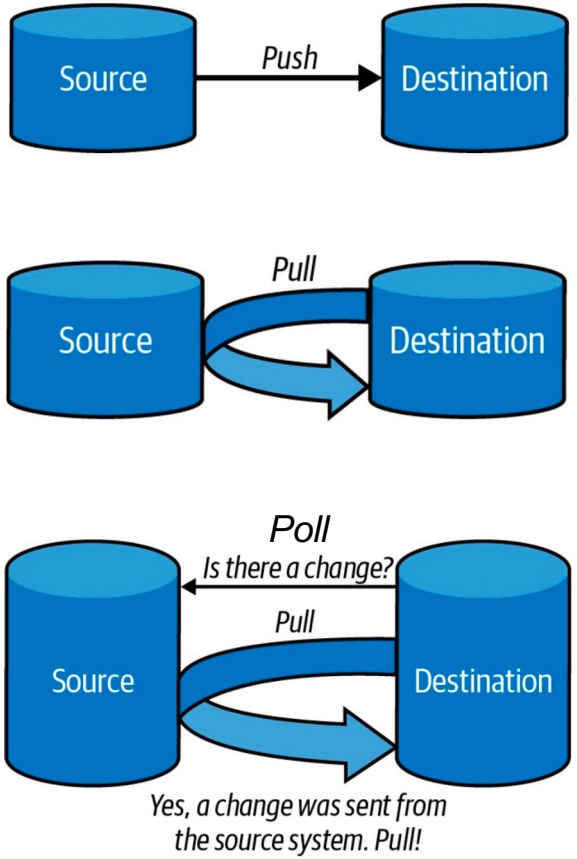
- File Access
  - Existing files / datasets of an enterprise;
    or download from an online web/data server (e.g. data.gov.au)
  - Typical exchange formats: CSV, Excel, sometimes XML or JSON
  - Also unstructured files such as text or images
- Database Access / Application Databases
- Programmatically / APIs
  - For example scraping the web (HTML)
  - or using APIs of Web Services or IoT devices (XML/JSON)
- Change Data Capture (CDC) / Logs
- Messages and streams, Publish/Subscribe

Key Engineering Considerations for Data Ingestion

- Is on-demand ingestion possible? Push vs. Pull?
- What is the expected data volume?
- In what velocity will the data typically arrive?
  Respectively, how often should the data be updated from the source?
- Batch data vs. Streaming data?
- What format is the data in?
- Is the source data in good shape for immediate downstream use or which data cleaning and transformation steps are needed?
- Data transformation and cleaning is needed immediately, or is the first destination a staging area?

Push versus Pull versus Poll Patterns

- A **push strategy** involves a source system sending data to a target
  - E.g. notifications or publish-subscribe systems such as Apache Kafka, MQTT
- **Pull strategy**: target reading data directly from a source
  - E.g. file download or database querying, web access
- **Polling** involves periodically checking a data source for any changes.
  - E.g. Web crawling



[J. Reis and M. Housley: Fundamentals of Data Engineering, 2022]

Bounded versus Unbounded Data

- Data comes in two forms: bounded and unbounded
- **Bounded data** is data in discrete datasets or batches.
  - Such as downloadable files or data pulled from an API call
- **Unbounded data** is continuous and varying in speed as events happen in reality
  - For example stream of sensor readings, log entries, messaging systems
  - But also change sets of databases over time
- All unbounded data must be converted to bounded data for processing

Throughput, Latency and Scalability

- Some data acquisition tasks are one-off and hence not time critical
- When working with unbounded data streams or change sets, ***latency*** considerations are important
  - A function of both the simplicity and performance of the data pipeline itself, and the frequency and low overhead of the data ingestion process
  - Ability to handle bursty data ingestion
    - requires fast processing and built-in buffering to prevent data from getting lost
- Data ***throughput*** and system ***scalability*** become critical as data volumes grow and requirements change.

higher throughput → deal more data faster

Cleaning and Transforming Data

deal with dirty data

- Real data is often '*dirty*'
- Important to do some data cleaning and transforming first
- Typical steps involved:
  - type and name conversion
    - eg string to number
  - filtering of missing or inconsistent data
  - unifying semantic data representations
  - matching of entries from different sources
- Optionally:
  - **Scaling** and optional dimensionality reduction

Data Acquisition: ETL versus ELT

- ETL: Extract – Transform – Load
  - **Extract** – get data from a source system; while typically use a batch-pull approach, can also use a push strategy
  - **Transform** – transform and clean data for a target system
  - **Load** – load data into the storage layer, e.g., a data warehouse
- Originally ETL was driven by resource limitations of both source and target systems, handling transformations external to both
- Evolution of ETL to ELT in two variants:
  - load extracted data into a *data lake* (e.g HDFS - Hadoop Distributed File System) and then transform for each use case
  - load extracted data into a data warehouse with minimal transformation, and then clean and transform it directly there

Undercurrents

- Security
  - Data ingestion phase exposes data pipeline to major security risks, e.g regarding unauthorized access or man-in-the-middle attacks (copying or modifying data)
  - Use a VPN or a dedicated private connection if you need to send data between the cloud and an on-premises network. If your data traverses the public internet, use encryption in-transit (e.g. https)
- Data ethics, privacy, and compliance
  - Do you really need the data you plan to ingest? Especially sensitive/personal data?
    - eg identity data, credit card data
  - If indeed needed, aim to only use it for production deployment, but otherwise build, train and test with simulated or anonymized/masked data.
  - Encryption (at rest) and access authorization are good, but no magical bullets due to human factor
- Data Management
  - How are schema-changes at the source(s) handled? How are you informed about them?

Undercurrents (cont'd)

- Data Operations
  - Data pipelines should be properly monitored to avoid analysing stale data downstream
    - E.g. errors / uptime, latency, and data volumes
  - Data Quality Tests

- Orchestration
  - As data pipeline complexity grows, "orchestration" is necessary – e.g. a system capable of scheduling complete task graphs rather than individual tasks.

deal with multiple tasks

- Software Engineering
  - Take advantage of the best available tools – managed tools and services are very useful
  - Important to use proper version control and code review processes

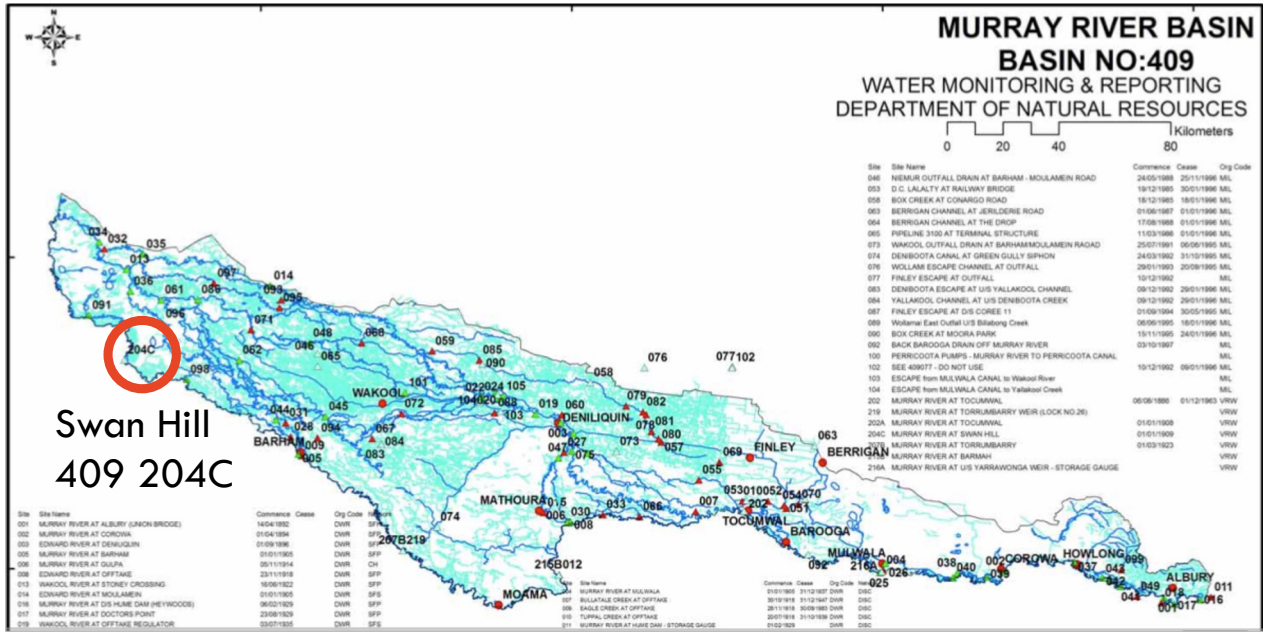
Data Acquisition Summary

- Important phase of the data pipeline
- Driven by both data needs of downstream, as well as source capabilities
- Design of data ingestion process(es):
  - Different architectures to consider (push vs pull, stream vs batch processing)
  - Performance considerations are important re throughput / latency / scalability
  - All undercurrents, such as security and data privacy, DataOps or orchestration affect the design of this phase

File-based Approach

THE UNIVERSITY OF SYDNEY

Example: Water Data about Murray River Basin in NSW

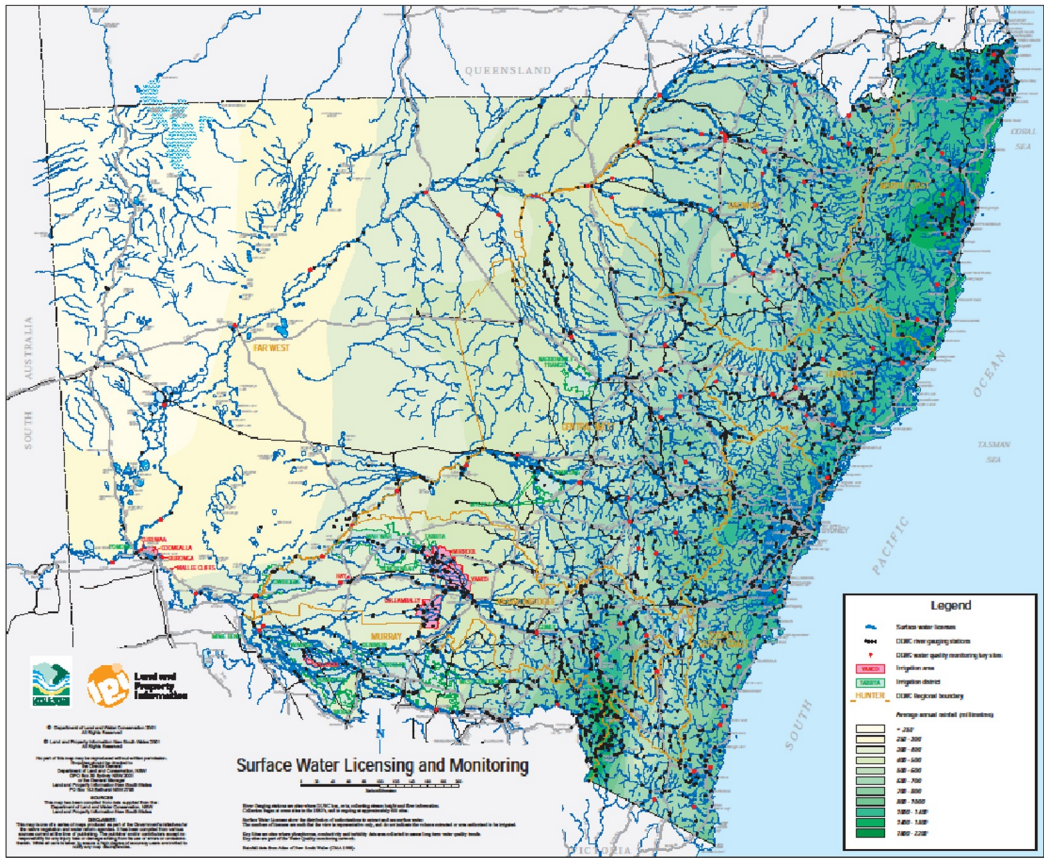


- automatic monitoring stations
- periodically send data about water level, water flow, water temperature, salinity…

[Source: mdba.gov.au]
[https://realtimedata.waternsw.com.au]

Murray River Basin Data Set

- Water measurements:
  - automatic monitoring stations
  - that are distributed over a larger area
    - (say Murray River catchment basin)
  - that periodically send their measured values to a central authority:
    - Time-series data of water level, water flow, water temperature, salinity (via measuring electric conductivity) or other hydraulic properties



File-based Approaches

- Just put the data in files
  - eg. csv text files
  - eg. spreadsheet files
- Anyone who wants to do some analysis, gets a copy of the files (eg. in email, or download from a web site)
- Analysis might be done by writing formulas or creating charts in a spreadsheet, or by writing Python code (eg. with pandas)

Example: Spreadsheet Files

Murray-waterinfo.nsw.gov.au.xls

	A	B	C	D	E	F	G	H	I
1	Station	Date	Level (m)	MeanDischarg	Discharge (ml/d)	Temp (C)	EC @ 25C (us/cm)		
272	409204C	1-Apr-09	0.713	2821.487	2773.949	21.558	54		
273	219018	1-Apr-09	-0.173	0					
274	409017	1-Apr-09	2.331	7152.066	8499.806	20.921	45		
275	409204C	2-Apr-09	0.698	2721.779	2667.749	21.833	53.5		
276	219018	2-Apr-09	-0.098	0	0				
277	409017	2-Apr-09	2.497	8972.182	10741.82	21.167	45.766		
278	409204C	3-Apr-09	0.677	2609.139	2552.696	22.194	54.458		
279	219018	3-Apr-09	0.04	0	0				
280	409017	3-Apr-09	2.638	10596.43	9263.902	21.505	47.51		
281	409204C	4-Apr-09	0.653	2470.194	2409.639	22.102	>55		
282	219018	4-Apr-09	0.166	0.198	0.371				
283	409017	4-Apr-09	2.472	8684.356	7817.866	21.569	49.823		
284	409204C	5-Apr-09	0.637	2373.525	2358.659	20.633	51.75		
285	219018	5-Apr-09							
286	409017	5-Apr-09	2.389	7734.209	7744.308	21			
287	409204C	6-Apr-09	0.637	2368.798	2390.783	20			
288	219018	6-Apr-09	--	--	0.239				
289	409017	6-Apr-09	2.403	7883.954	7861.138	20			
290	409204C	7-Apr-09	0.637	2372.584	2358.659	19			
291	219018	7-Apr-09	0.166	0.159	0.119				
292	409017	7-Apr-09	2.39	7747.374	7649.435	20			
293	409204C	8-Apr-09	0.628	2310.373	2271.601	18			
294	219018	8-Apr-09	0.162	0.105	0.091				
295	409017	8-Apr-09	2.368	7511.15	7299.264	19			
296	409204C	9-Apr-09	0.615	2221.495	2185.795	18			
297	219018	9-Apr-09	0.164	0.127	0.122				
298	409017	9-Apr-09							
299	409204C	10-Apr-09	0.601	2131.538	2101.01	18			
300	219018	10-Apr-09	0.163	0.117	0.116				
301	409017	10-Apr-09	--	--	6228.199				
302	409204C	11-Apr-09	0.591	2096.919	2086.492	19			

WaterInfo | Stations | Sensors

Ready

Murray-waterinfo.nsw.gov.au.xls

	A	B	C	D	E	F	G	H
1	Basin No	Site	Site Name	Long	Lat	Commence	Cease	Org Code
2	409	001	Murray River at Albury (Union Bridge)	146.8957 E	36.0929 S	14/04/1892		DWR
3	409	002	Murray River at Corowa	146.3954 E	36.0076 S	01/04/1894		DWR
4	409	003	Murray River at Denuquin	144.9663 E	35.5301 S	01/09/1896		DWR
5	409	005	Murray River at Barham	144.1235 E	35.6304 S	1/1/1905		DWR
6	409	204C	Murray River @ Swan Hill	143.5680 E	35.3318 S	1/1/1909		VRW
7	409	017	Murray River @ Doctors Point	146.9401 E	36.1129 S	8/23/1929		DWR
8	409	019	Wakool River at Offtake Regulator	144.8846 E	35.4985 S	7/3/1935		DWR
9	409	202	Murray River @ Tocumwal	145.5596 E	35.8151 S	06/06/1886	12/1/1963	VRW
10	219	018	Murray River @ Quaama	149.8526 E	36.4762 S	7/12/1966		DWR
11	...							
12								
13								
14	Organisation Codes							
15	Code	Organisation						
16	DNR	NSW Department of Water and Energy (and predecessors)						
17	DWR	NSW Department of Water and Energy (and predecessors)						
18	MIL	Murray Irrigation Ltd						
19	PWD	Manly Hydraulics Laboratory						
20	QWR	Qld Department of Natural Resources and Water						
21	SCA	Sydney Catchment Authority						
22	SMA	Snowy Mountains Authority						
23	SWB	Sydney Catchment Authority						
24	VRW	Vic Government						
25								

WaterInfo | Stations | Sensors

Ready

Sum=0

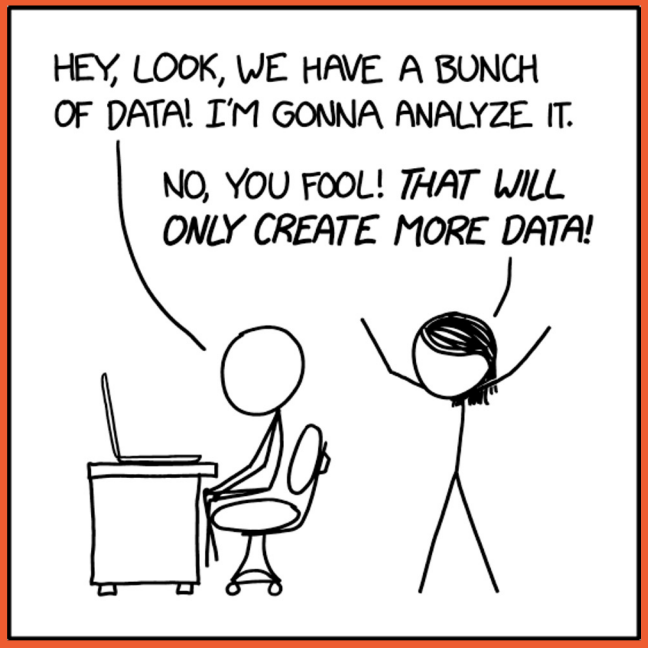
[Source: www.waterinfo.nsw.gov.au]

Quality Issues with File-based Approaches

- Data quality is left to the users doing the right thing

- **Metadata** is often stored separately from the data files; Difficult to keep metadata up-to-date and synchronized with the data using File-based system.
- Keep **backups,** manage **sharing**
  - Multiple copies ("redundancy"), hard to know which is most-up-to-date
- Prevent changes that introduce inconsistency or violate integrity properties

Data Cleaning



THE UNIVERSITY OF SYDNEY

Data Quality Overview

- The quality of our data directly impacts the quality of the conclusions we draw from it
  - "Garbage in, garbage out" premise
    (first known use in the pictured US newspaper article from 1957)

- While data cleaning is integral to the analysis workflow, the overall data quality must first be considered


Work With New Electronic 'Brains' Opens Field for Army Math Experts

WASHINGTON — BIZMAC UNIVAC, GARBAGE IN-GARBAGE OUT—all new terms in the Army

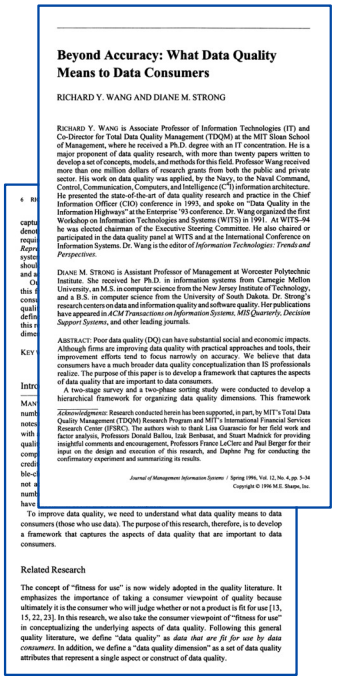
NOW is the TIME!

[Image: The Times of Northwest Indiana 10-Nov-1957, p65, digitised by Newspapers.com]

Data Quality

Though not a complete list, the top five most important attributes of data quality from an often-cited study*:

1. **Believable**: Accepted or regarded as true, real, and credible
2. **Value-added**: Beneficial and provide advantages from their use
3. **Relevant**: Applicable and helpful for the task at hand
4. **Accurate**: Correct, reliable, and certified free of error
5. **Interpretable**: Inappropriate language, with clear data definitions



Beyond Accuracy: What Data Quality Means to Data Consumers

RICHARD Y. WANG AND DIANE M. STRONG

To improve data quality, we need to understand what data quality means to data consumers (those who use data). The purpose of this research, therefore, is to develop a framework that captures the aspects of data quality that are important to data consumers.

Related Research

The concept of "fitness for use" is now widely adopted in the quality literature. It emphasizes the importance of taking a consumer viewpoint of quality because ultimately it is the consumer who will judge whether or not a product is fit for use [13, 15, 22, 23]. In this research, we also take the consumer viewpoint of "fitness for use" in conceptualizing the underlying aspects of data quality. Following this general quality literature, we define "data quality" as *data that are fit for use by data consumers*. In addition, we define a "data quality dimension" as a set of data quality attributes that represent a single aspect or construct of data quality.

*Based on study by Wang & Strong (1996), who surveyed 355 MBA alumni of a "large university" in the United States

[Image: Journal of Management Information Systems, Vol 12 No 4 p5-6]

Data Quality

Data Issue	Example	Impact
Believable?	Sales data shows an unexpected spike during non-peak periods, suggesting possible data tampering.	Misleading analysis and decision-making.
Value-Added?	Collecting web traffic data that doesn't correlate with conversion rates.	Resource waste without actionable insights.
Relevant?	Including data on discontinued products in current sales analysis.	Distracts from focusing on current inventory.
Accurate?	Customer records have mismatched names and addresses due to data entry errors.	Leads to ineffective communication and loss of trust.
Interpretable?	Financial reports contain undefined technical terms and abbreviations.	Misinterpretation by non-technical stakeholders.

Data Cleaning

- Once quality assured, data cleaning must then be conducted
- This is to ensure a consistent, "clean" representation of the information
- Most common problems occur when values are:
  - **Missing**: A specific value is simply unpopulated
  - **Default**: A placeholder has been used when the true value is unknown
  - **Incorrect**: A value exists, but is not valid in the context of that field
  - **Inconsistent**: A valid value exists, but is inconsistent with other related values

Missing Values

Causes	Often simply **not recorded.** (e.g. intentional non-response in a survey) Could instead be indicative of other issues. (e.g. device fault if capture via sensor)
Impact	If missingness is not random, then could risk **underrepresentation.** (e.g. common at a particular time of day, or amongst survey respondents of a particular demographic)
Detection	**In-built functions** to determine and quantify the extent of missing values. (e.g. Python libraries such as Pandas)
Remedy	Data points with missing values can be **removed,** or imputed. (be careful this does not sacrifice valuable data, or entrench underrepresentation)

Default Values

Causes	Like missing values, but **labelled** with a set value, rather than left blank. (e.g. "NA", though can be less obvious like –1)
Impact	Can cause **inaccuracy** in analysis, or complicate operations. (e.g. attempting an average of a column when a text value exists)
Detection	**Domain knowledge** is often critical, but can also be discerned. (check metadata definitions, or frequency distributions)
Remedy	Often best **removed** and treated as missing values.

Incorrect Values

Causes	Can be the result of a **range of issues.** (e.g. device faults in sensors, unreliable survey responses, simple data entry mistakes)
Impact	Simply distorts the **accuracy** of results.
Detection	Relies on domain knowledge to know what constitutes a **valid** value. (alternatives are proper statistical tests, see e.g. Benford's law for fraudulent accounting)
Remedy	Also often best **removed,** unless adjustments can be made. (e.g. spelling errors may be possible to replace)

Inconsistent Values

Causes	Typically the result of **free-text** input from collected data. (e.g. spelling variations like "Röhm" vs "Roehm")
Impact	Risks **underrepresentation** by dividing responses that should be considered equal.
Detection	May require **detection of clusters** to find unknown variants.
Remedy	Values best **reconciled** as a single value for the purpose of reporting.

Example

CustomerID	Name	Email	PostalCode	OrderDate	ProductCategory
101	Alice Johnson	alice@example.com	2000	12/25/2023 (DATE)	Electronics
102	Bob Smith	NULL	99999	25th Dec 2024 (DATE, TEXT)	electronics
103	Charlie Brown	charlie@example.com	3000	01-15-2023 UTC	Electronics
104	Dana White	dana@example.com	4000	12/31/2023 12:00:00 (TIMESTAMP)	electronics

- Missing: Bob Smith's Email is missing.
- Default: Bob Smith's PostalCode is set to a placeholder 99999.
- Incorrect: Bob Smith's OrderDate uses an incorrect date format.
- Inconsistent: The ProductCategory shows mixed casing for similar products ("Electronics" and "electronics").

Data Cleaning Summary

- **Data Quality** directly impacts the quality of the conclusions we draw from data
- **Data Cleaning** is important in particular for data acquired from external sources
- Typical cleaning steps:
  - **Missing**: A specific value is simply unpopulated – can be **removed**, or **imputed**.
  - **Default**: A placeholder has been used when the true value is unknown – **replace** with default.
  - **Incorrect**: A value exists, but is not valid in the context of that field – **remove** or **adjust**.
  - **Inconsistent**: A valid value exists, but is inconsistent with other related values – **reconcile**.

Data Engineering with Python

THE UNIVERSITY OF SYDNEY

Example: Analysis of Major Power Stations in Australia

- Dataset from data.gov.au

- How can we load this data into Python?
- Which data preparation steps are needed?

Source: https://digital.atlas.gov.au/search?collection=dataset&q=power%20station

Read Data into Python using csv

- Python **csv** module
  - Reads/writes comma-separated values with escaping to handle cases where comma occurs within a field
    - csv.reader reads rows into arrays
    - csv.DictReader reads rows into ***dictionaries***
- However: Not much support for further data handling or output…
  - E.g. type conversions needed

```
import csv
import pprint
data = list(csv.DictReader(open('Major_Power_Stations.csv')))
pprint.pprint(data[0])
```

Pandas – Python Data Analysis Library

- Open-source library providing data import and analysis functionality to Python
- https://pandas.pydata.org/
  - optimised data structures for data analysis
    - Tabular data (DataFrame)
    - Time series data (Series)
    - Matrixes
  - configurable input/output file
  - support for handling missing data, cleaning data, descriptive stats
- API documentation:
  https://pandas.pydata.org/pandas-docs/stable/reference/index.html

Pandas – Reading Data from a CSV file

- Pandas provides several reader functions for various file formats such as, for example, CSV
  - configurable with many options (cf. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html)

```
import pandas as pd
data = pd.read_csv('Major_Power_Stations.csv')
data.head()
```

Pandas: Fix Missing Values During Import

- Some datasets contain placeholders for missing values
  - such as ‘n/a’, ‘--’ or ‘null’
- Best to replace during import to avoid later problems

```
import pandas as pd

missing_values = [“--”,”<Null>”]
data = pd.read_csv(‘Major_Power_Stations.csv’, na_values = missing_values)
data.head()
```

Pandas – Missing Data Handling

- Pandas provides various functions for handling missing/wrong data
  - Part of this already included in the input functions (cf. **csv_read()**) where missing values are automatically replaced with NA/NaN
  - Other examples:
    - DataFrame.dropna() remove rows with any missing values
    - DataFrame.fillna() fill NA/NaN values using a specified method
    - DataFrame.replace() replace values

```
data2 = data['generatornumber'].dropna()
data.fillna({'generatornumber':0}, inplace=True)
data['generatornumber'] =
    data['generatornumber'].replace(to_replace='<Null>', value=0)
```

Cleaning Data: Convert to Correct Types

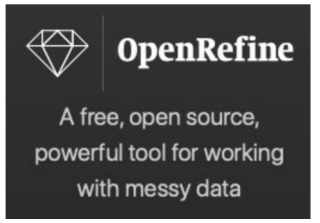
- The standard Python **csv** module reads everything as string types
- **Pandas** is a bit better, but still will fallback to string when it can't deduce the type from all values in a column
- This will give problems sooner or later with stat functions or plots
- Need to convert as appropriate (e.g., int, float, timestamp)
  - int() creates integer objects, e.g., -1, 101
  - float() creates floating point object, e.g., 3.14, 2.71
  - datetime.strptime() creates datetime objects from strings

Pandas Typing

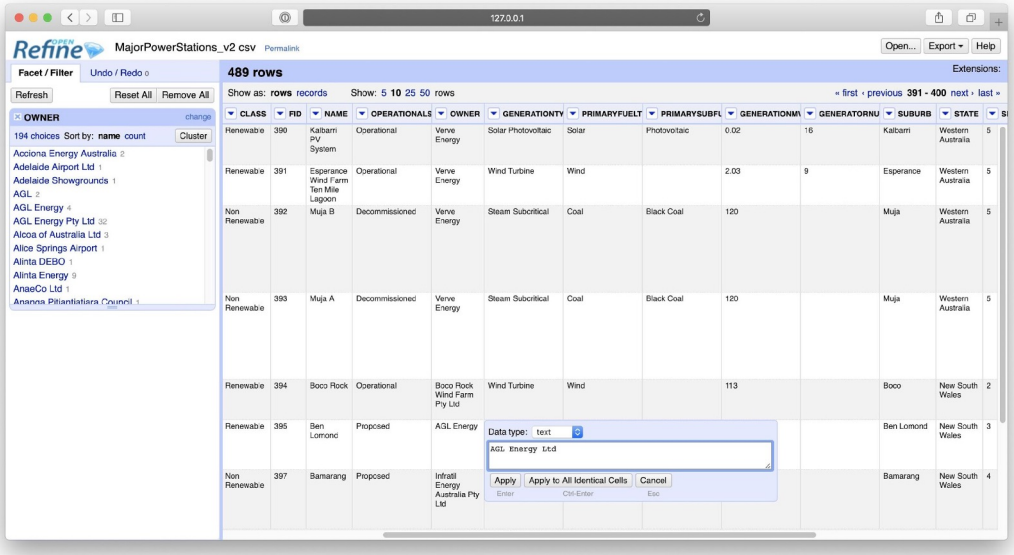
- astype() function on Data series to convert to new types
  - Careful: fails if any entry in the series violates the new type
  - For example: ints do not support NaN
  - Also: does not support special value handling

```
data['generatornumber'] = data['generatornumber'].astype(int)
data['generationmw']   = data['generationmw'].astype(float)
```

Specific Data Cleaning Tools



- Open-Source Example: **Open Refine**
  - Originally developed by Google
  - Allows to visually inspect and clean data with interactive user-interface
  - More advanced: Reconcile and match different data sets
  - Export to CSV, Excel, HTML, …
- Very helpful, especially for smaller data sets
  - But manual interaction is required



Data Visualisation with Python

THE UNIVERSITY OF SYDNEY

Visualising data with Pandas and matplotlib

- Matplotlib provides functionality for creating various plots
- Bar charts, line charts, scatter plots, etc
- Pandas offers some easy-to-use shortcut functions

- Reference page for Pandas plotting:
  https://pandas.pydata.org/pandas-docs/stable/reference/plotting.html
- Reference page for pyplot:
  http://matplotlib.org/api/pyplot_api.html
- Matplotlib Documentation:
  http://matplotlib.org/contents.html

Creating a Bar Chart

for nominal / categorial data

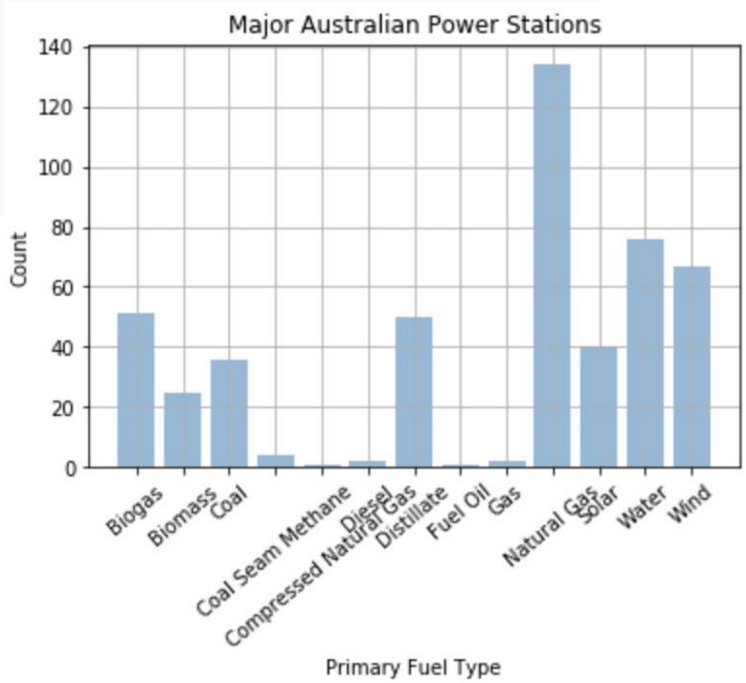
Use groupby() to get frequency distribution
Rename resulting counts as 'numStations'

```
import matplotlib.pyplot as plt
%matplotlib inline
fuelTypeDistr = data.groupby("primaryfueltype").size().reset_index(name="numStations")

# plot
plt.bar(fuelTypeDistr['primaryfueltype'], fuelTypeDistr['numStations'], alpha=0.5, align='center')
plt.xticks(rotation=40)
plt.title('Major Australia Power Stations')
plt.xlabel('Primary Fuel Type')
plt.ylabel('Count')
plt.grid()
```

Configure plot using matplotlib

Resulting plot ->



Plotting a Histogram

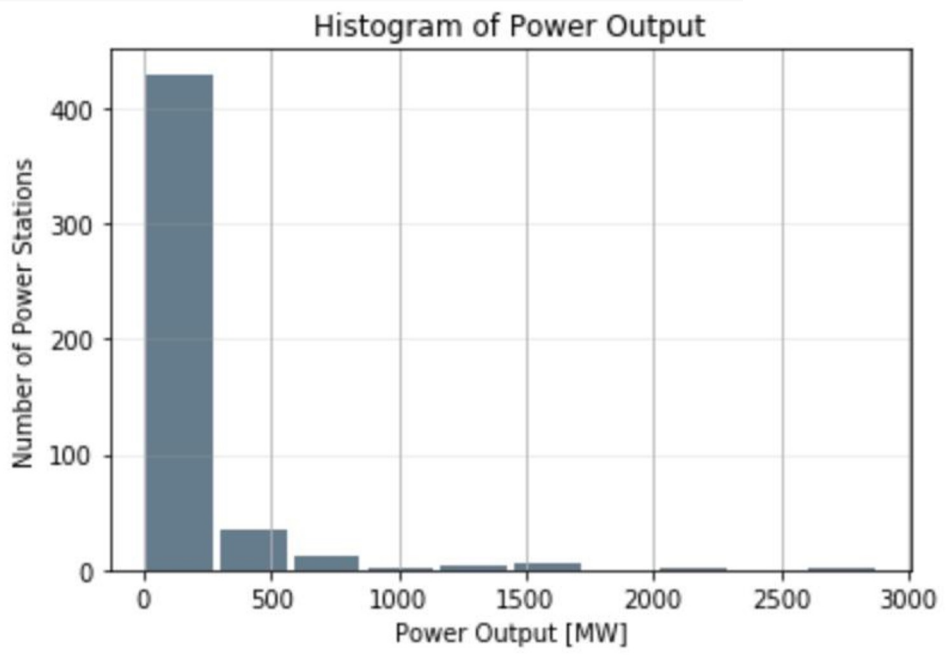
for continuous variables

Create histogram plot with 10 bins of 'power' values

```
pyExpFreq = data['generationmw'].hist(bins=10, rwidth=0.9, color='#607c8e')
plt.title('Histogram of Power Output')
plt.xlabel('Power Output [MW]')
plt.ylabel('Number of Power Stations')
plt.grid(axis='y', alpha=0.25)
```

Configure plot

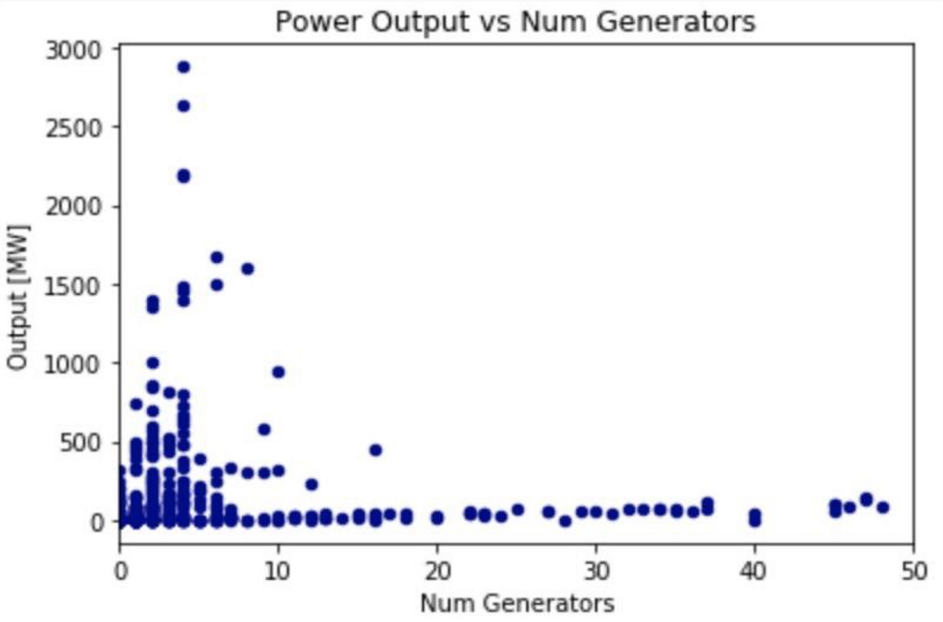
Resulting histogram ->



Creating a Scatter Plot

Create scatter plot

```
fig = plt.figure()
sub = plt.subplot()
data.plot.scatter(x='generatornumber', y='generationmw', c='DarkBlue', ax=sub)
sub.set_xlim(0,50)
plt.title('Power Output vs Num Generators')
plt.xlabel('Num Generators')
plt.ylabel('Output [MW]')
```



Creating a Scatter Plot with variable coloring/grayscale

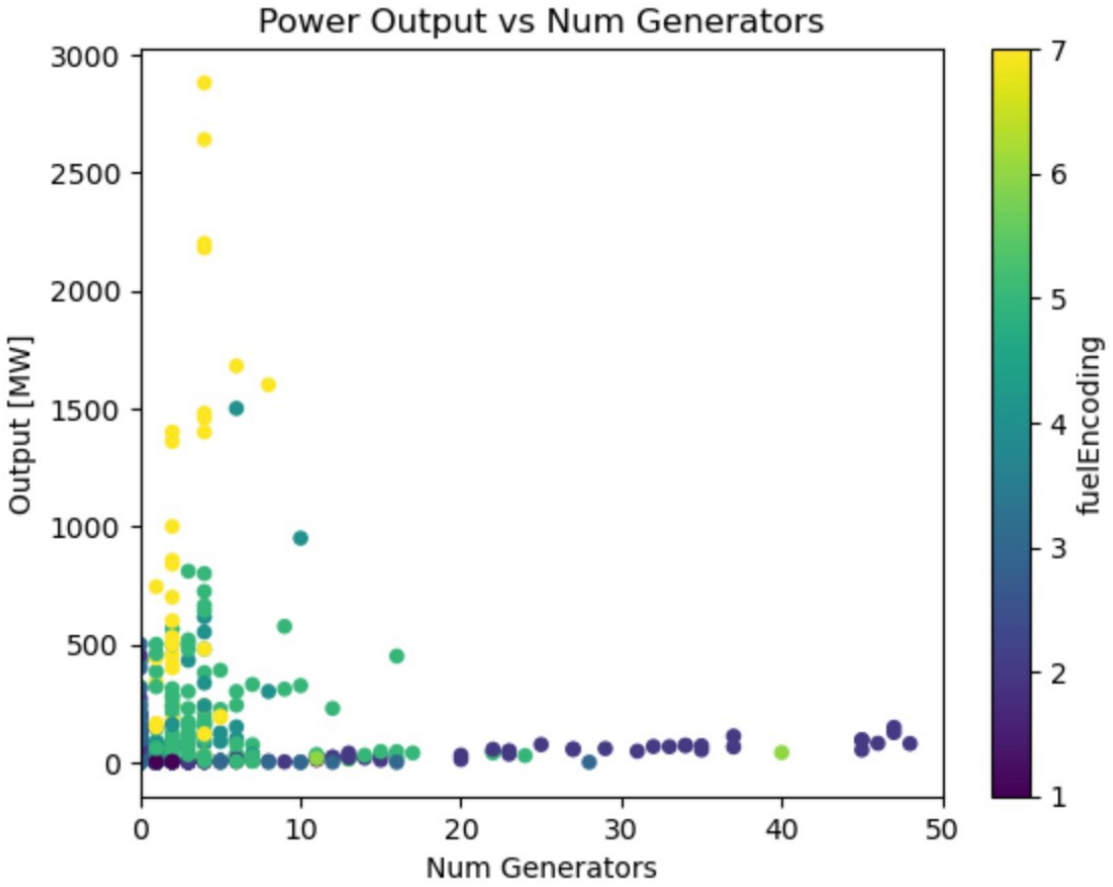
```
# assign colors to some selected fuel types
# the numbers and the order chosen are up-to you
# we have chosen an order that works well with the color schemes used in the subsequent plots
data['fuelEncoding'] = data['primaryfueltype'].map({
    'Biogas': 1,
    'Wind': 2,
    'Solar': 3,
    'Water': 4,
    'Natural Gas': 5,
    'Coal Seam Methane': 6,
    'Coal': 7
})
```

Encode fuel type into numerical values [1..7]

```
# Now we can use this encoding column to color our plot
fig = plt.figure()
sub = plt.subplot()
data.plot.scatter(x='generatornumber', y='generationmw', c='fuelEncoding', ax=sub)
sub.set_xlim(0,50)
plt.title('Power Output vs Num Generators')
plt.xlabel('Num Generators')
plt.ylabel('Output [MW]')
```

Color by encoding values

Scatter Plot2 comparing Power Output vs. Generator Size



Coal

Coal Seam

Natural Gas

Water

Solar

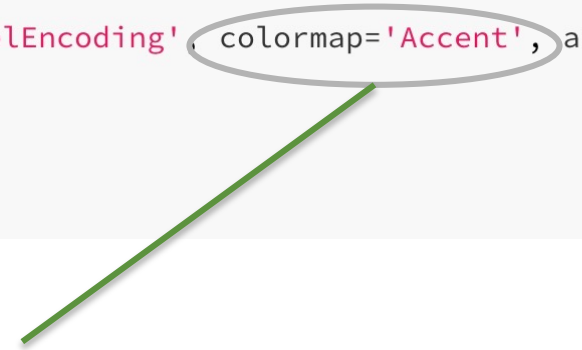
Wind

Biogas

Creating a Scatter Plot with specific colormap

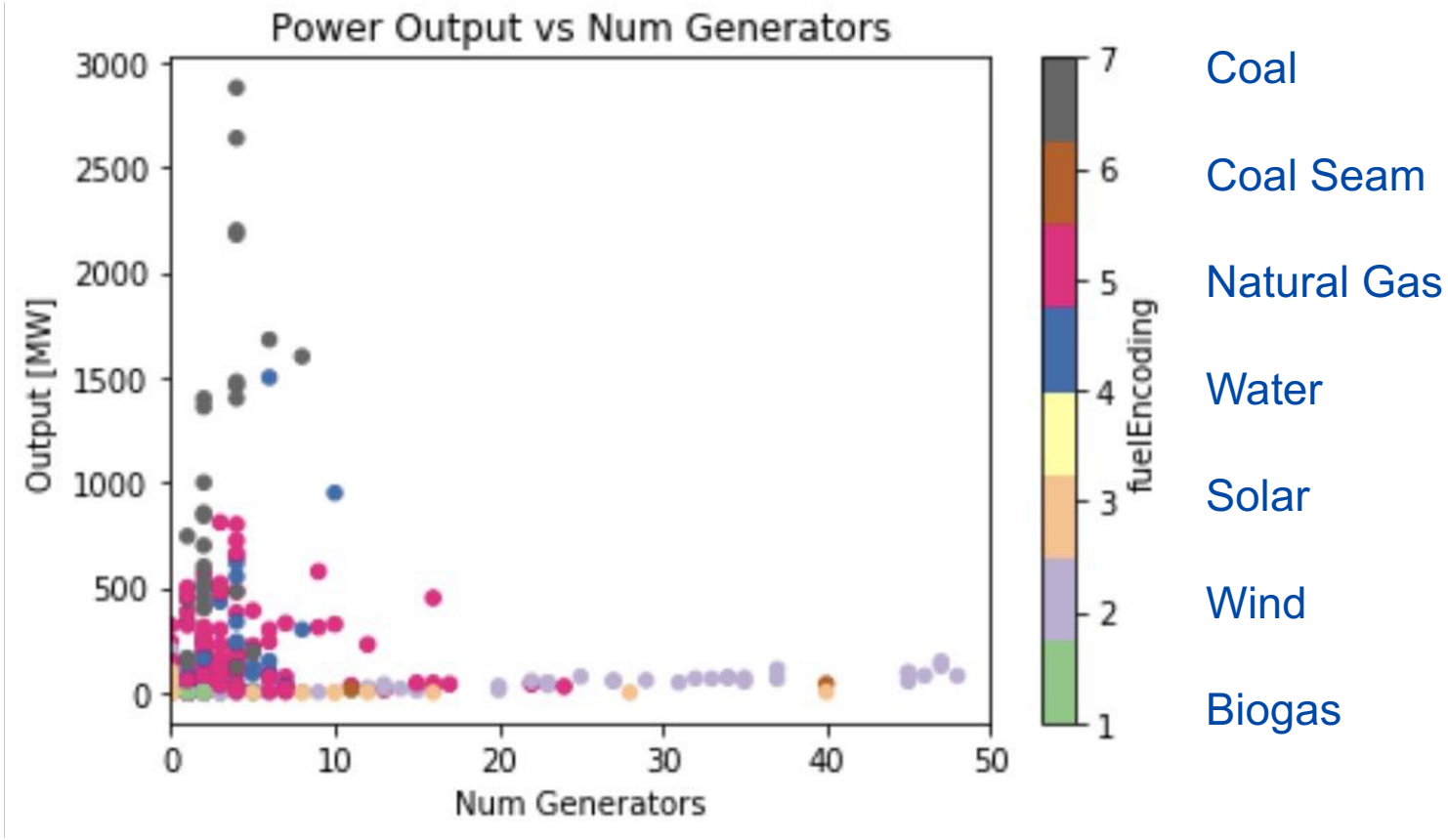
Use same encoding as before…

```
# the same plot as before, but using a more vivid color scheme (colormap='Accent')
fig = plt.figure()
sub = plt.subplot()
data.plot.scatter(x='generatornumber', y='generationmw', c='fuelEncoding', colormap='Accent', ax=sub)
sub.set_xlim(0,50)
plt.title('Power Output vs Num Generators')
plt.xlabel('Num Generators')
plt.ylabel('Output [MW]')
```



Color using matplotlib's 'Accent' colormap

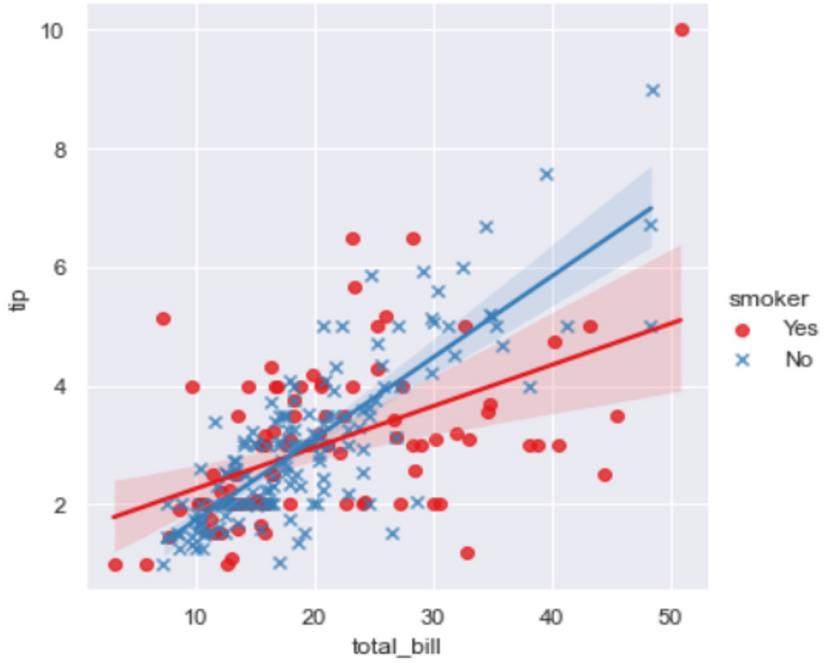
Scatter Plot2 Comparing Power Output vs. Generator Size



Other Visualisations: Seaborn library

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_theme(color_codes=True)
tips = sns.load_dataset("tips")
sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips, markers=["o", "x"], palette="Set1");
```



Data Visualisation Summary

Data visualization is important for data exploration and monitoring pipelines. Some standard examples:

- **Bar Charts** for Categorical/Nominal Data
- **Histograms** for Numerical Data
- **Scatter Plots** for comparing two continuing variables
  - Colouring (and/or shapes) to overlay categorical data

- Python + Pandas provide a lot of support
  - matplotlib, seaborn library, many more…
  - many configuration options to adjust data visualisations to your needs