Paul Bradshaw

What do you need to know?

- Common info problems: wrong format, inconsistent case, HTML, spaces, etc.
- Python: Converting, combining, import-and-clean
- Python: Use algorithms to correct variant spelling

Problems to watch out for:

- Numbers/dates treated as strings
- Vice-versa: e.g. company ‘numbers’
- Combined data (addresses)
- Different data in one column (country, region, city)
- Variant spellings
- Mistypings - missing decimals etc.

Help us. Oh god no. (Part 1)

Merged cells
Empty rows
Headings across multiple rows
Different information in same column
Different terms for same thing

```
[ ]  overview21mardf = pd.read_excel(overview21mar,
                                     engine="odf",
                                     sheet_name="Q5_1",
                                     skiprows=4)
```

`sheet_name=` **name (string) or index**
`skip_rows=` **how many rows before header row?**

https://github.com/paulbradshaw/python5sessions/blob/main/session4/04cleaning_criminaljusticeODS.ipynb

Help us. Oh god no.

Multiple files? The os library
Multiple sheets? Pandas's .ExcelFile() and .sheet_names

```
xls = pd.ExcelFile('excel_file_path.xls')

# Now you can list all sheets in the file
xls.sheet_names
# ['house', 'house_extra', ...]

# to read just one sheet to dataframe:
df = pd.read_excel(file_name, sheetname="house")
```

https://github.com/paulbradshaw/python5sessions/blob/main/session4/os_files.ipynb
https://github.com/paulbradshaw/python5sessions/blob/main/session1/01pythonFirstStepsColab.ipynb

Fill down using `ffill`

The `ffill` function will fill down when given `axis=0` (to fill across use `axis=1`).

Note that this fills row index 1 with the values from row 0, too. So it's a good thing we removed that row first.

```
#fill down into empty cells (axis 0 means columns)
overview21mardf = overview21mardf.ffill(axis=0)
#show the results
overview21mardf
```

`df.ffill(axis=0)` **fill down empty cells with whatever is above**

https://github.com/paulbradshaw/python5sessions/blob/main/session4/04cleaning_criminaljusticeODS.ipynb

```python
#https://pandas.pydata.org/docs/reference/api/pandas.melt.html
longdf = pd.melt(df,id_vars='lad21cd', value_name='value', var_name="original_col")
print(longdf)
```

```
         lad21cd original_col value
0      E06000001            A    20
1      E06000002            A   NaN
2      E06000003            A   NaN
3      E06000004            A    12
4      E06000005            A   NaN
...          ...          ...   ...
28792  W06000020         G.10    93
28793  W06000021         G.10   416
28794  W06000022         G.10   177
28795  W06000023         G.10  1623
28796  W06000024         G.10   109

[28797 rows x 3 columns]
```

`pd.melt()` wide to long (columns become values)

https://github.com/paulbradshaw/python5sessions/blob/main/session4/04cleaning_criminaljusticeODS.ipynb

```
[ ] #use the parse function to interpret a string as a datetime object
    parse('September 18, 2020 at 11:05AM')

    datetime.datetime(2020, 9, 18, 11, 5)
```

`from dateutil.parser import parse`
`parse('September 18, 2020 at 11:05AM')` **convert strings to datetime objects**

https://github.com/paulbradshaw/python5sessions/blob/main/session4/DatetimeMPexes.ipynb

Clustering

An intelligent helper

Navigation

Project description

Release history

Download files

Project links

Homepage

Statistics

GitHub statistics:

Stars: 10

Forks: 0

Open issues/PRs: 0

View statistics for this project via [Libraries.io](), or by using [our public dataset on Google BigQuery]()

Project description

Fuzz Up [W.I.P.]

build passing | codecov 89% | pypi v0.0.15 | downloads 46/month | license MIT

`fuzzup` offers a simple approach for clustering strings based on [Levenshtein Distance]() using [Fuzzy Matching]() in conjunction with [Hierarchical Clustering]().

Installation guide

`fuzzup` can be installed from the Python Package Index (PyPI) by:

```
pip install fuzzup
```

If you want the development version then install directly from [Github]().

Workflow

`fuzzup` organizes strings by forming clusters from them. It does so in 3 steps:

1. Compute all of the mutual string distances (Levensteihn Distances/fuzzy ratios) between the strings

https://pypi.org/project/fuzzup/

```python
# strings we want to cluster
person_names = ['Donald Trump', 'Donald Trump',
                    'J. biden', 'joe biden', 'Biden',
                    'Bide', 'mark esper', 'Christopher c . miller',
                    'jim mattis', 'Nancy Pelosi', 'trumps',
                    'Trump', 'Donald', 'miller']

from fuzzup.gear import form_clusters_and_rank
form_clusters_and_rank(person_names)
```

```
/usr/local/lib/python3.7/dist-packages/fuzzywuzzy/fuzz.py:11: UserWarning
  warnings.warn('Using slow pure-python SequenceMatcher. Install python-L
[{'COUNT': 4,
  'PROMOTED_STRING': 'joe biden',
  'RANK': 2,
  'STRINGS': ['Bide', 'Biden', 'J. biden', 'joe biden']},
 {'COUNT': 2,
  'PROMOTED_STRING': 'Christopher c . miller',
  'RANK': 3,
  'STRINGS': ['Christopher c . miller', 'miller']},
 {'COUNT': 5,
  'PROMOTED_STRING': 'Donald Trump',
  'RANK': 1,
  'STRINGS': ['Donald', 'Donald Trump', 'Trump', 'trumps']},
 {'COUNT': 1,
  'PROMOTED_STRING': 'Nancy Pelosi',
  'RANK': 6,
  'STRINGS': ['Nancy Pelosi']},
 {'COUNT': 1,
  'PROMOTED_STRING': 'jim mattis',
```

```python
#create empty list
prefstrings = []

#loop through original names
for i in person_names:
  #loop through dicts of clusters of names
  for d in rankdict:
    #if the original name is in the list of strings
    if i in d['STRINGS']:
      #print that and the 'promoted' (preferred) one
      print(i, "=", d['PROMOTED_STRING'])
      #add promoted one to list
      prefstrings.append(d['PROMOTED_STRING'])
```

```
Donald Trump = Donald Trump
Donald Trump = Donald Trump
J. biden = joe biden
joe biden = joe biden
Biden = joe biden
Bide = joe biden
mark esper = mark esper
Christopher c . miller = Christopher c . miller
jim mattis = jim mattis
Nancy Pelosi = Nancy Pelosi
trumps = Donald Trump
Trump = Donald Trump
```

Algorithms (from Open Refine)

Fingerprint: looks for items with identical characters, e.g. “John Smith,” and “Smith, John”
metaphone3: looks for similar sounds, e.g. “Horowitz” and “Horowicz”
PPM: partial matches - try increasing radius to increase

Nearest neighbor: looks for shared clusters of characters, e.g. “Johnson” and “Johnsons”

Levenshtein: looks for number of edits needed to change one to another, e.g. “New York” -> “newyork” = 3 edits

Gender API Home API Docs Clients FAQ Blog Contact Privacy Policy Terms of Use My A

LOCALIZATION

Add more accuracy to your request by localizing your query:

Request **Example Italy**

```
GET https://gender-api.com/get?name=Andrea&country=IT&key=<your private server key>
```

Request **Example Germany**

```
GET https://gender-api.com/get?name=Andrea&country=DE&key=<your private server key>
```

Field	Type	Description
name	String	Name to query
country	String	ISO 3166 ALPHA-2 Country Code
key	String	Your private server key

Response **Example Italy**

```
//In Italy, Andrea is male.
{"name":"andrea","name_sanitized":"Andrea","country":"IT","gender":"male","samples":1068,"accuracy":95,"c
```

Hooray!

gender-api.com/
genderize.io (R package)

Parserator Parse addresses, names & more

Home API About

Parse addresses, names or any unstructured text into useful components

`123 Main St. Suite 100 Chicago, IL` →

Address part	Tag
`123`	AddressNumber
`Main`	StreetName
`St.`	StreetNamePostType
`Suite`	OccupancyType
`100`	OccupancyIdentifier
`Chicago`	PlaceName
`IL`	StateName

Hooray!

https://parserator.datamade.us/

Internal Validity

- row counts (e.g. correct number of provinces?)
- histograms
- do the numbers add up?

External Validity

- alternate data sources
- expert knowledge
- previous versions
- common sense!

Adapted from Jonathan Stray

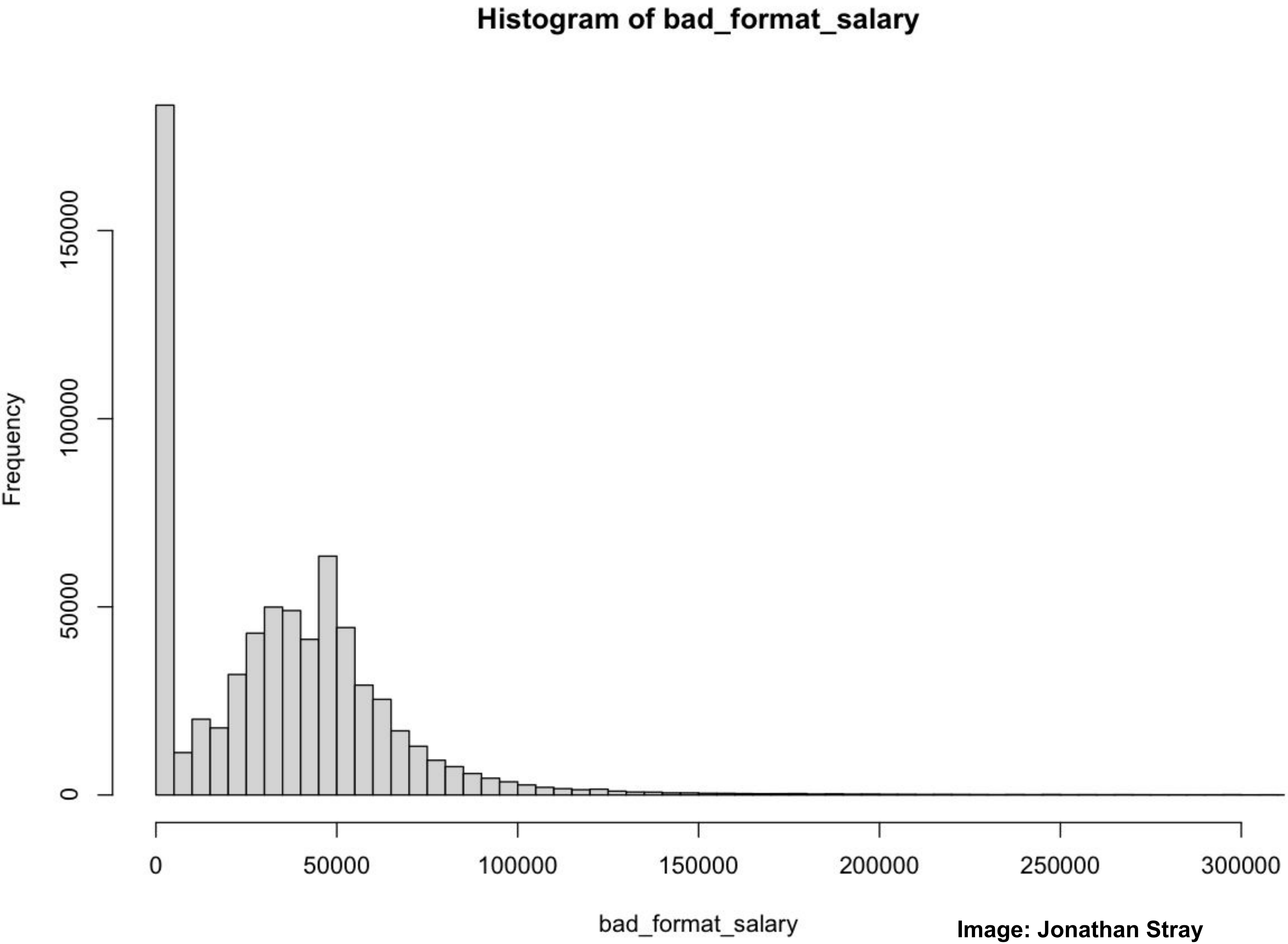

Image: Jonathan Stray

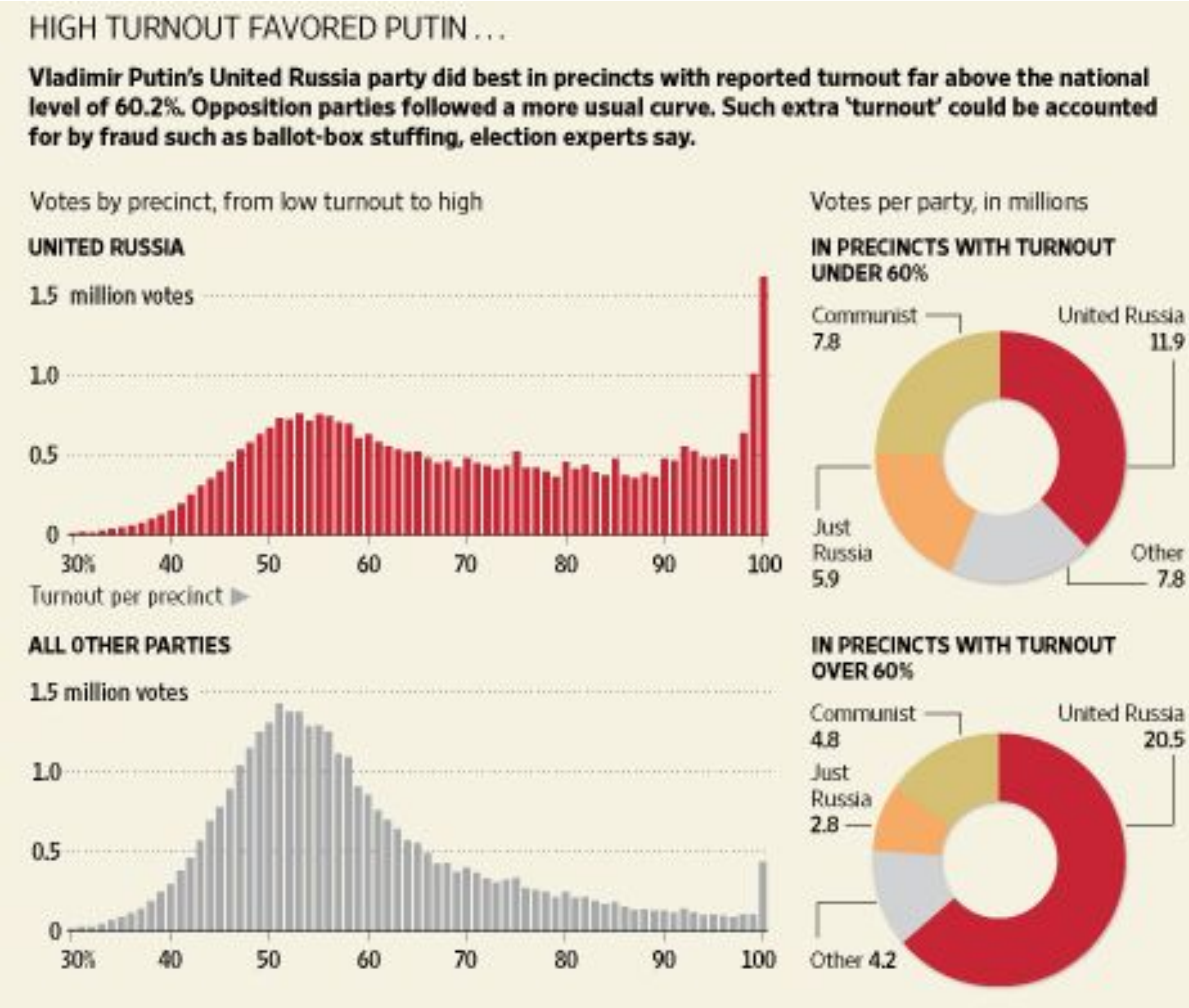

Image: Jonathan Stray

What to remember

- If you have a problem, someone has probably already solved it before (APIs and libraries)
- Look out for outliers
- Break down the problem, and solve each part separately