CMPU 101 § 2 · Computer Science I

Table Sanitizing and Processing

21 September 2022

Where are we?

Last class, we downloaded a real data set – the London Fire Brigade's animal rescues – as a CSV file.

We imported it to Google Sheets and pruned it to just the columns we were interested in.

Then we loaded the data in Pyret and began to investigate what we had and what issues we'd need to fix.

ward	borough
some("Crystal Palace & Upper Norwood")	some("Croydon")
some("Woodside")	some("Croydon")
some("Carshalton Central")	some("Sutton")
some("Harefield")	some("Hillingdon")
some("Gooshays")	some("Havering")
some("Alibon")	some("Barking and Dagenham")
some("Cathall")	some("Waltham Forest")
some("Wanstead")	some("Redbridge")
some("New Addington North")	some("Croydon")
some("Lea Bridge")	some("Hackney")

We'll sanitize the "borough" and "ward" fields, which contain blanks:

```
include gdrive-sheets
include data-source

include shared-gdrive("dcic-2021", "1wyQZj_L0qqV9Ekgr9au6RX2iqt2Ga8Ep")

ssid = "1iohS5voB-Y-CezClAphyIdc90t_Me1z3tA0TACemOa4"
spreadsheet = load-spreadsheet(ssid)

rescue-data =
  load-table:
    datetime, year, description, animal-group,
    property-type, property-category, service-kind,
    service-type, ward, borough, station-name, street
    source: spreadsheet.sheet-by-name("rescues-simple", true)
    sanitize borough using string-sanitizer
    sanitize ward using string-sanitizer
  end
```

This provides the sanitizer functions

ward	borough
"Crystal Palace & Upper Norwood"	"Croydon"
"Woodside"	"Croydon"
"Carshalton Central"	"Sutton"
"Harefield"	"Hillingdon"
"Gooshays"	"Havering"
"Alibon"	"Barking and Dagenham"
"Cathall"	"Waltham Forest"
"Wanstead"	"Redbridge"
"New Addington North"	"Croydon"
"Lea Bridge"	"Hackney"

Looking for problems

```
>>> count(rescue_data_clean, "borough")
```

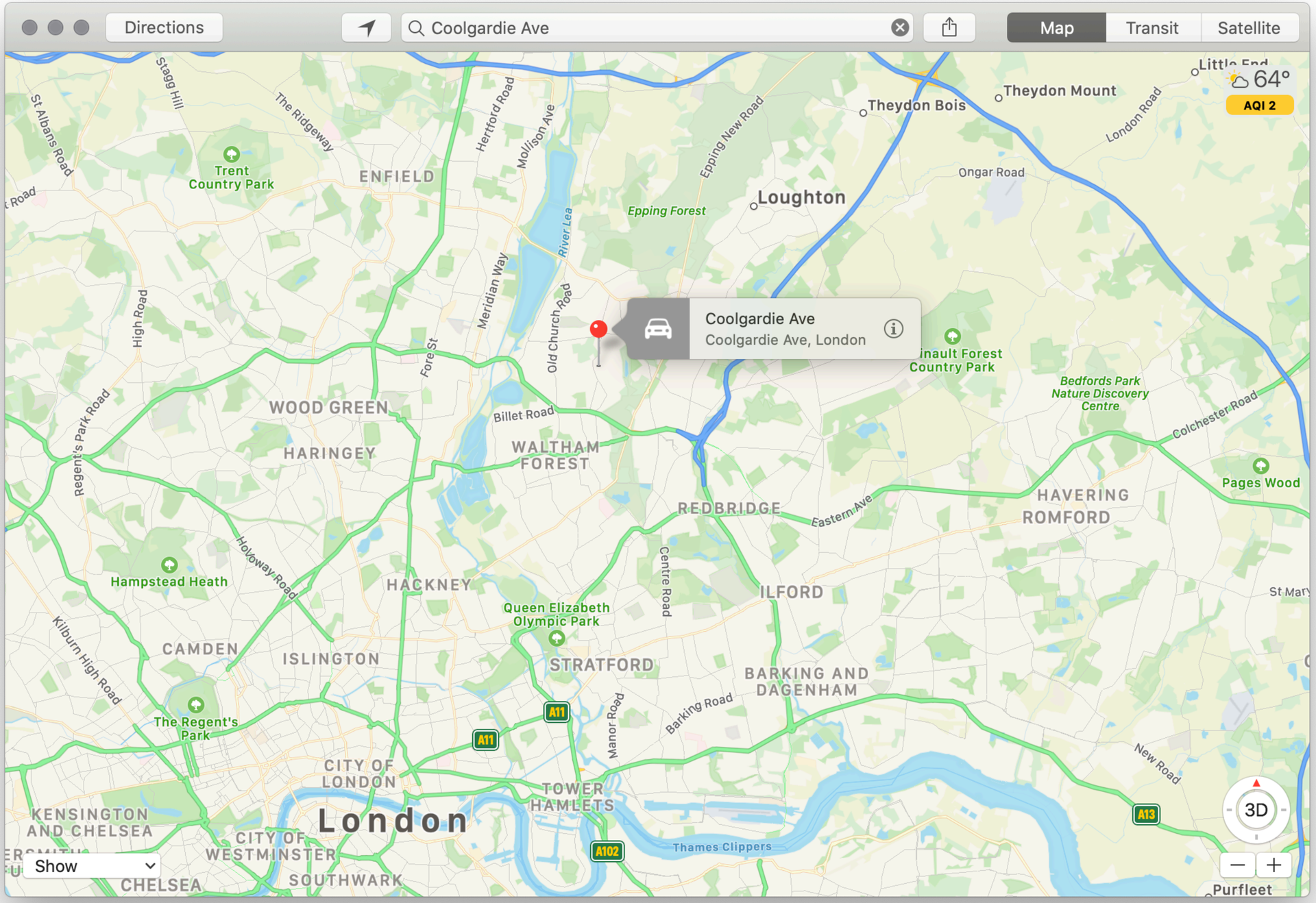
value	count
""	11
"WESTMINSTER"	152
"HAMMERSMITH AND FULHAM"	119
"CITY OF LONDON"	5
"RICHMOND UPON THAMES"	130
"HARROW"	101
"MERTON"	93
"SOUTHWARK"	187

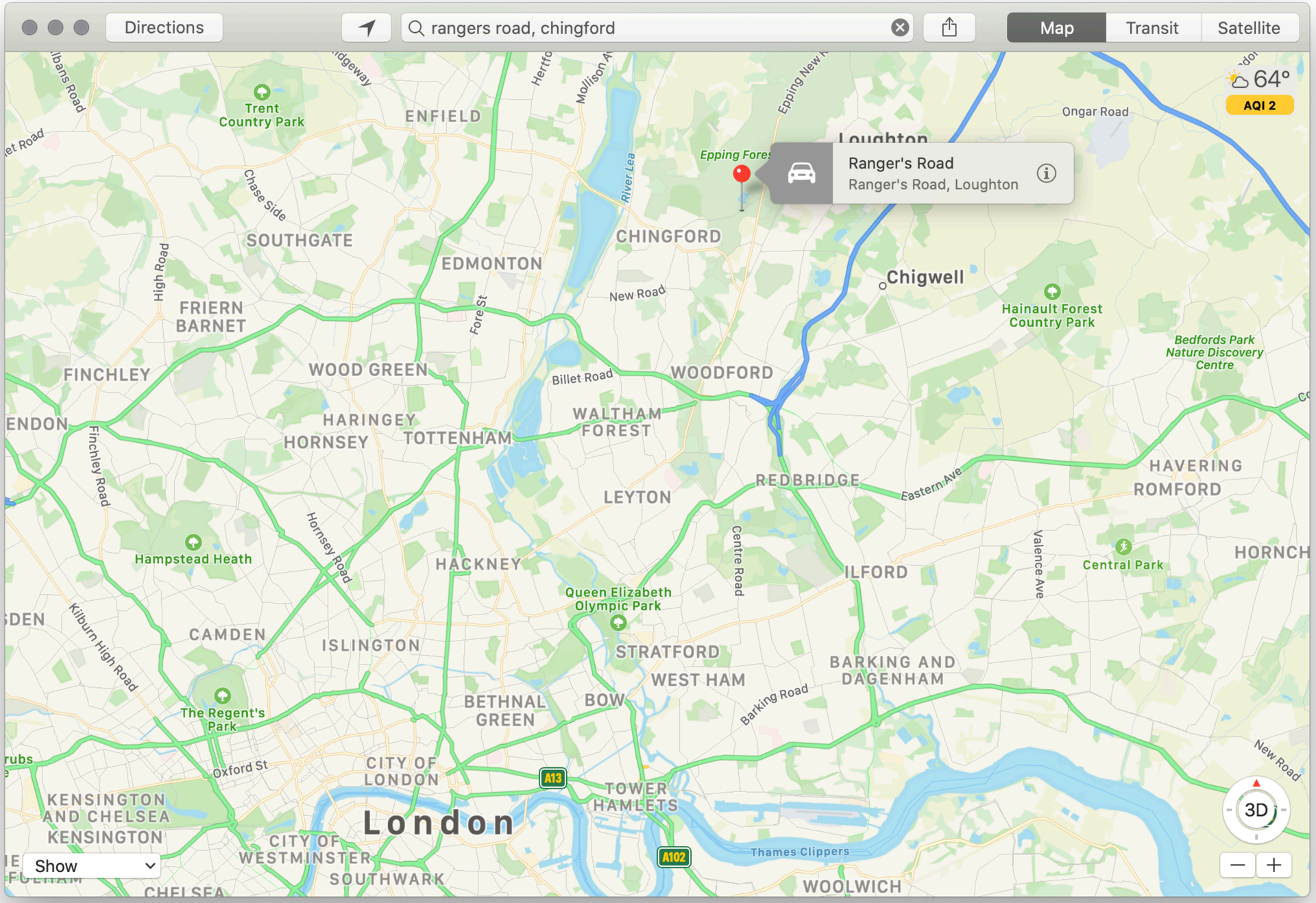
What are these rows? What should we do with them?

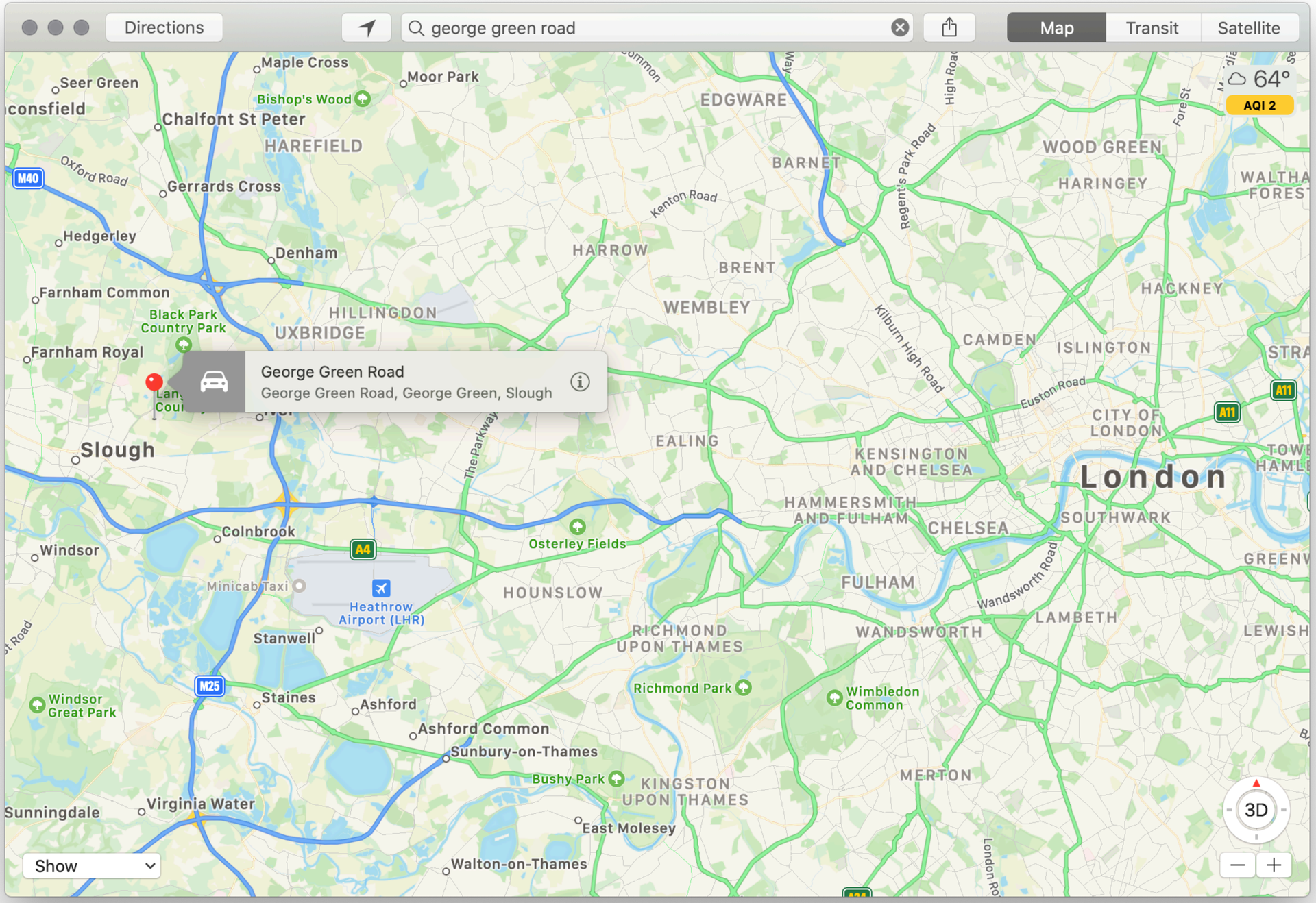
```
fun no-borough(r :: Row) -> Boolean:
  doc: "Return true if the borough column is empty for
a given row"
  r["borough"] == ""
end
```

```
››› filter-with(rescue-data, no-borough)
```

ward	borough	station-name	street
""	""	"Essex"	"COOLGARDIE AVENUE"
""	""	"Essex"	"PALMERSTON ROAD"
""	""	"Essex"	"BRADWELL ROAD"
""	""	"Chingford"	"RANGERS ROAD"
""	""	"Kent"	"MEAD WALL"
""	""	"Buckinghamshire"	"GEORGE GREEN ROAD"


Directions
Coolgardie Ave
Map
Transit
Satellite
64°
AQI 2
Coolgardie Ave
Coolgardie Ave, London
Theydon Mount
Theydon Bois
Loughton
Epping Forest
ENFIELD
Trent Country Park
WOOD GREEN
HARINGEY
WALTHAM FOREST
REDBRIDGE
HAVERING ROMFORD
HACKNEY
ILFORD
CAMDEN
ISLINGTON
STRATFORD
BARKING AND DAGENHAM
CITY OF LONDON
TOWER HAMLETS
KENSINGTON AND CHELSEA
CITY OF WESTMINSTER
SOUTHWARK
CHELSEA
London
Hampstead Heath
Queen Elizabeth Olympic Park
The Regent's Park
Bedfords Park Nature Discovery Centre
Pages Wood
Thames Clippers
Purfleet
Show
3D


Directions
rangers road, chingford
Map
Transit
Satellite
64°
AQI 2
Ranger's Road
Ranger's Road, Loughton
Trent Country Park
ENFIELD
Ongar Road
Epping Forest
River Lea
Chase Side
SOUTHGATE
CHINGFORD
EDMONTON
High Road
Chigwell
New Road
FRIERN BARNET
Fore St
Hainault Forest Country Park
Bedfords Park Nature Discovery Centre
FINCHLEY
WOOD GREEN
Billet Road
WOODFORD
HARINGEY
HORNSEY
TOTTENHAM
WALTHAM FOREST
Finchley Road
REDBRIDGE
Eastern Ave
HAVERING
ROMFORD
LEYTON
Hampstead Heath
Hornsey Road
HACKNEY
Centre Road
ILFORD
Valence Ave
Central Park
Queen Elizabeth Olympic Park
Kilburn High Road
CAMDEN
ISLINGTON
STRATFORD
BARKING AND DAGENHAM
WEST HAM
The Regent's Park
BETHNAL GREEN
BOW
Barking Road
Oxford St
CITY OF LONDON
A13
TOWER HAMLETS
KENSINGTON AND CHELSEA
KENSINGTON
London
CITY OF WESTMINSTER
SOUTHWARK
A102
Thames Clippers
WOOLWICH
New Road
3D
Show


Directions
george green road
Map
Transit
Satellite
64°
AQI 2
George Green Road
George Green Road, George Green, Slough
Slough
London
Heathrow Airport (LHR)
Minicab Taxi
Osterley Fields
Richmond Park
Wimbledon Common
Bushy Park
Windsor Great Park
Black Park Country Park
Show
3D
Seer Green
Maple Cross
Moor Park
Bishop's Wood
Chalfont St Peter
HAREFIELD
Oxford Road
Gerrards Cross
Hedgerley
Denham
Farnham Common
Farnham Royal
HILLINGDON
UXBRIDGE
HARROW
BRENT
WEMBLEY
EDGWARE
BARNET
Kenton Road
Regent's Park Road
WOOD GREEN
HARINGEY
HACKNEY
CAMDEN
ISLINGTON
Kilburn High Road
Euston Road
CITY OF LONDON
EALING
KENSINGTON AND CHELSEA
HAMMERSMITH AND FULHAM
CHELSEA
SOUTHWARK
FULHAM
LAMBETH
WANDSWORTH
RICHMOND UPON THAMES
HOUNSLOW
Wandsworth Road
The Parkway
Colnbrook
Windsor
Stanwell
Staines
Ashford
Ashford Common
Sunbury-on-Thames
KINGSTON UPON THAMES
MERTON
Virginia Water
Sunningdale
East Molesey
Walton-on-Thames
M40
M25
A4
A11

```
fun no-borough(r :: Row) -> Boolean:
  doc: "Return true if the borough column is empty for
a given row"
  r["borough"] == ""
end
```

```
››› filter-with(rescue-data, no-borough)
```

ward	borough	station-name	street
""	""	"Essex"	"COOLGARDIE AVENUE"
""	""	"Essex"	"PALMERSTON ROAD"
""	""	"Essex"	"BRADWELL ROAD"
""	""	"Chingford"	"RANGERS ROAD"
""	""	"Kent"	"MEAD WALL"
""	""	"Buckinghamshire"	"GEORGE GREEN ROAD"

Look like places outside of London itself or where the location is vague?

If we make a visualization that shows a count for "", people will rightly think it's a bug. We should give a more meaningful value for these rows.

The **`transform-column`** function is used to clean up or otherwise alter the data in a single column of a table.

It returns a new table by applying its function argument to each value of the given column.

```
fun empty-to-other(s :: String) -> String:
  doc: "Return the string or 'Other' if the string is
empty."
  if s == "":
    "Other"
  else:
    s
  end
end

rescue-data-clean =
  transform-column(
    transform-column(rescue-data,
      "borough", empty-to-other),
    "ward", empty-to-other)
```

```
order-by(
  count(rescue-data-clean, "borough"),
  "value", true)
```

value	count
"BARKING AND DAGENHAM"	98
"BARNET"	198
"BEXLEY"	130
"BRENT"	142
"BROMLEY"	174
"Barking and Dagenham"	116
"Barnet"	151

```
order-by(
   count(rescue-data-clean, "borough"),
   "value", true)
```

value	count
"BARKING AND DAGENHAM"	98
"BARNET"	198
"BEXLEY"	130
"BRENT"	142
"BROMLEY"	174
"Barking and Dagenham"	116
"Barnet"	151

Some of the boroughs are ALL UPPERCASE and others are Title Case.

`"Islington"` and `"ISLINGTON"` will be treated by Pyret as distinct values, which will throw off our counts!

```
rescue-data-clean =
  transform-column(
    transform-column(
      transform-column(
        transform-column(
          transform-column(rescue-data,
            "borough", empty-to-other),
          "ward", empty-to-other),
        "borough", string-to-upper),
      "ward", string-to-upper)
```

This is getting a bit deep.

```
fun clean-table(t :: Table) -> Table:
  doc: "Fix various columns from the animal rescue spreadsheet"

  # Change the blank strings to "Other"
  t1 =
    transform-column(
      transform-column(t,
        "borough", empty-to-other),
      "ward", empty-to-other)

  # Change strings to all uppercase
  t2 =
    transform-column(
      transform-column(t1,
        "borough", string-to-upper),
      "ward", string-to-upper)

  t2
end

rescue-data-clean = clean-table(rescue-data)
```

Now we'll want to use **`rescue-data-clean`** rather than **`rescue-data`** for the rest of our work.

We usually maintain separate names for the initially-loaded table, the cleaned table, and for significant variations for analysis purposes.

Data analysis

```
>>> count(rescue_data_clean, "borough")
```

value	count
"OTHER"	11
"TANDRIDGE"	1
"BROXBOURNE"	1
"BRENTWOOD"	1
"CITY OF LONDON"	13
"ENFIELD"	394
"MERTON"	172
"BRENT"	246

```
>>> order-by(
      count(rescue-data-clean, "borough"),
      "value", true)
```

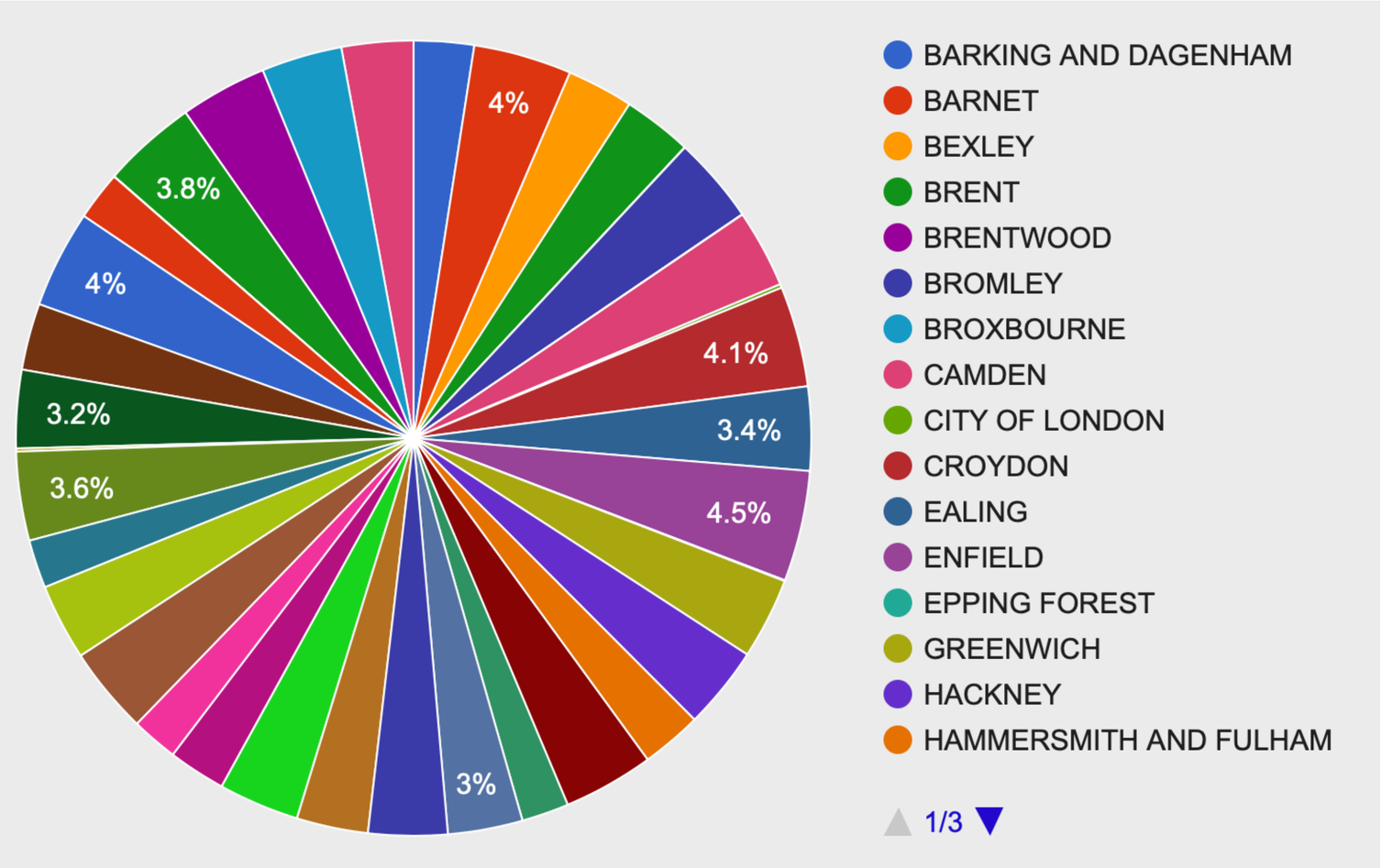
value	count
"BARKING AND DAGENHAM"	214
"BARNET"	349
"BEXLEY"	236
"BRENT"	246
"BRENTWOOD"	1
"BROMLEY"	309
"BROXBOURNE"	1

```
>>> order-by(
      count(rescue-data-clean, "borough"),
      "name", false)
```

value	count
"ENFIELD"	394
"CROYDON"	358
"BARNET"	349
"SOUTHWARK"	347
"TOWER HAMLETS"	336
"HARINGEY"	322
"NEWHAM"	316

```
borough-counts =
  order-by(
    count(rescue-data-clean, "borough"),
    "value", true)
```

```
> > > pie-chart(borough-counts, "value", "count")
```



What about getting a pie chart for the types of animals? Well, we could do

```
animal-counts =
  order-by(
    count(rescue-data-clean, "animal-group"),
    "value", true)
```

and then

```
pie-chart(animal-counts, "value", "count")
```

What about getting a pie chart for the types of animals? Well, we could do

```
animal-counts =
  order-by(
    count(rescue-data-clean, "animal-group"),
    "value", true)
```

and then

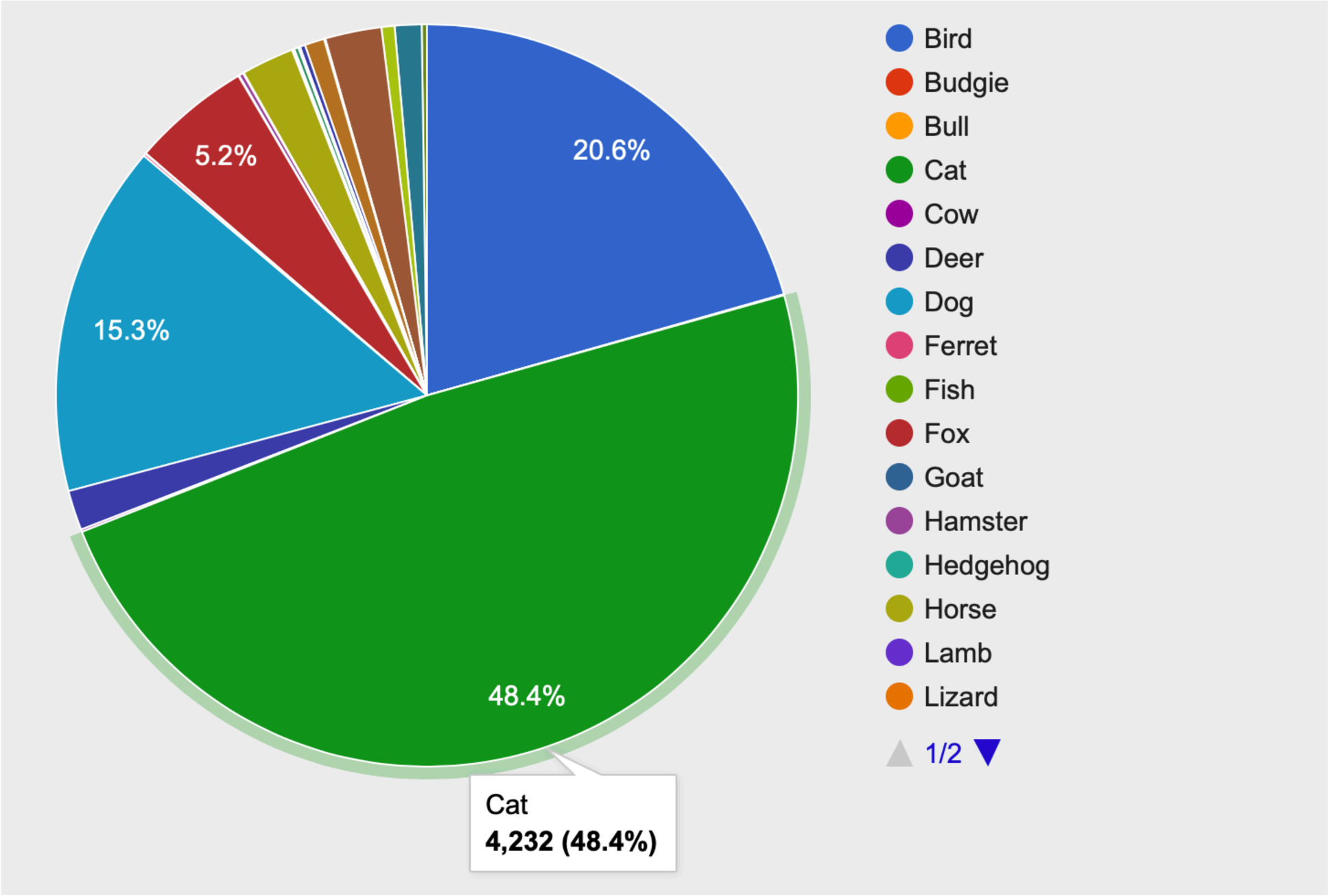
```
pie-chart(animal-counts, "value", "count")
```

Stop!

Rather than copy and paste long expressions, we should define a function:

```
fun pie-chart-counts(t :: Table, col :: String) -> Image:
  doc: "Make a pie chart of the number of times each value occurs in the
specified table column"
  counts = order-by(count(t, col), "value", true)
  pie-chart(counts, "value", "count")
end
```

```
>>> pie-chart-counts(rescue-data-clean, "animal-group")
```



The cliché may be true!

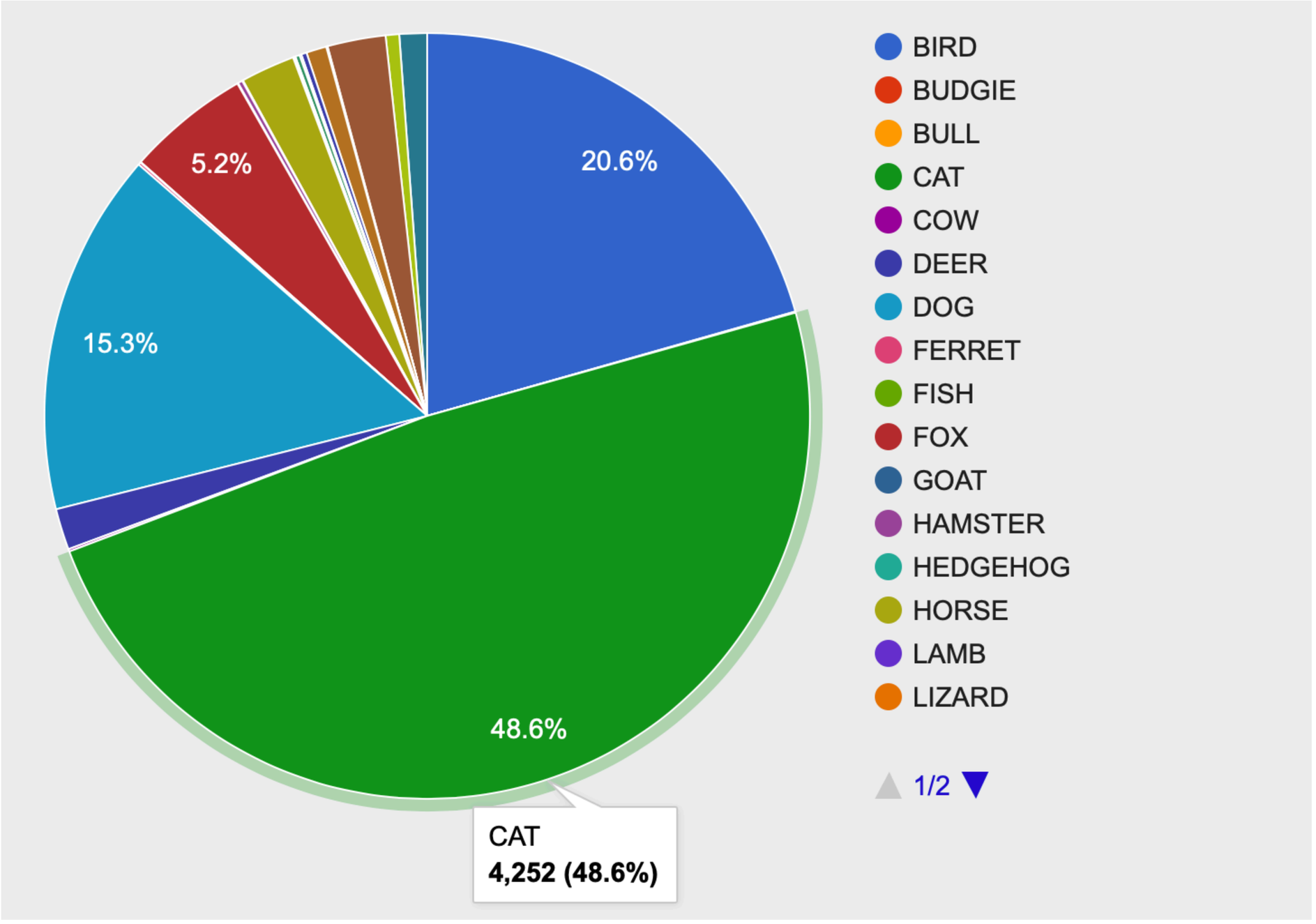
But we see a familiar problem – we have a count for `"Cat"` and for `"cat"`.

Back to cleaning data!

```
fun clean-table(t :: Table) -> Table:
  doc: "Fix various columns from the animal rescue spreadsheet"

  # Change the blank strings to "Other"
  t1 =
    transform-column(
      transform-column(t,
        "borough", empty-to-other),
      "ward", empty-to-other)

  # Change strings to all uppercase
  t2 =
    transform-column(
      transform-column(
        transform-column(t1,
          "borough", string-to-upper),
        "ward", string-to-upper),
      "animal-group", string-to-upper)

  t2
end

rescue-data-clean = clean-table(rescue-data)
```

Make the animal types uppercase just like the boroughs and wards.

```
>>> pie-chart-counts(rescue-data-clean, "animal-group")
```



Ok, I *have* to know: How many of those cats are actually stuck up trees?

About 15% of the time, the `"description"` column is `"Redacted"`, but when it isn't, this gives us a way to approximate the question.

Plan

Get just the rows about cats

Figure out for each row if it involves a tree

Make a pie chart

> Need a column to count values in, so we'd better put the up-a-tree-or-not answer into a table.

```
fun is-cat(r :: Row) -> Boolean:
  doc: "Return true if the row is about rescuing a
cat"
  r["animal-group"] == "CAT"
end

just-cats = filter-with(rescue-data-clean, is-cat)
```

Plan

✅ Get just the rows about cats

Figure out for each row if it involves a tree

Make a pie chart

Need a column to count values in, so we'd better put the up-a-tree-or-not answer into a table.

```
fun up-tree(r :: Row) -> String:
  doc: "Return string indicating whether or not cat is up
a tree"

  desc = string-to-lower(r["description"])

  if string-contains(desc, "tree"):
    "up a tree!"
  else:
    "not up a tree"
  end
end

cats-who-may-be-up-trees =
  build-column(just-cats, "up-tree", up-tree)
```

```
fun up-tree(r :: Row) -> String:
  doc: "Return string indicating whether or not cat is up
a tree"
  desc = string-to-lower(r["description"])

  if string-contains(desc, "tree"):
    "up a tree!"
  else:
    "not up a tree"
  end
end

cats-who-may-be-up-trees =
  build-column(just-cats, "up-tree", up-tree)
```

This handles descriptions with **"TREE"** *or* **"Tree"**

Plan

- ✓ Get just the rows about cats
- ✓ Figure out for each row if it involves a tree
- Make a pie chart

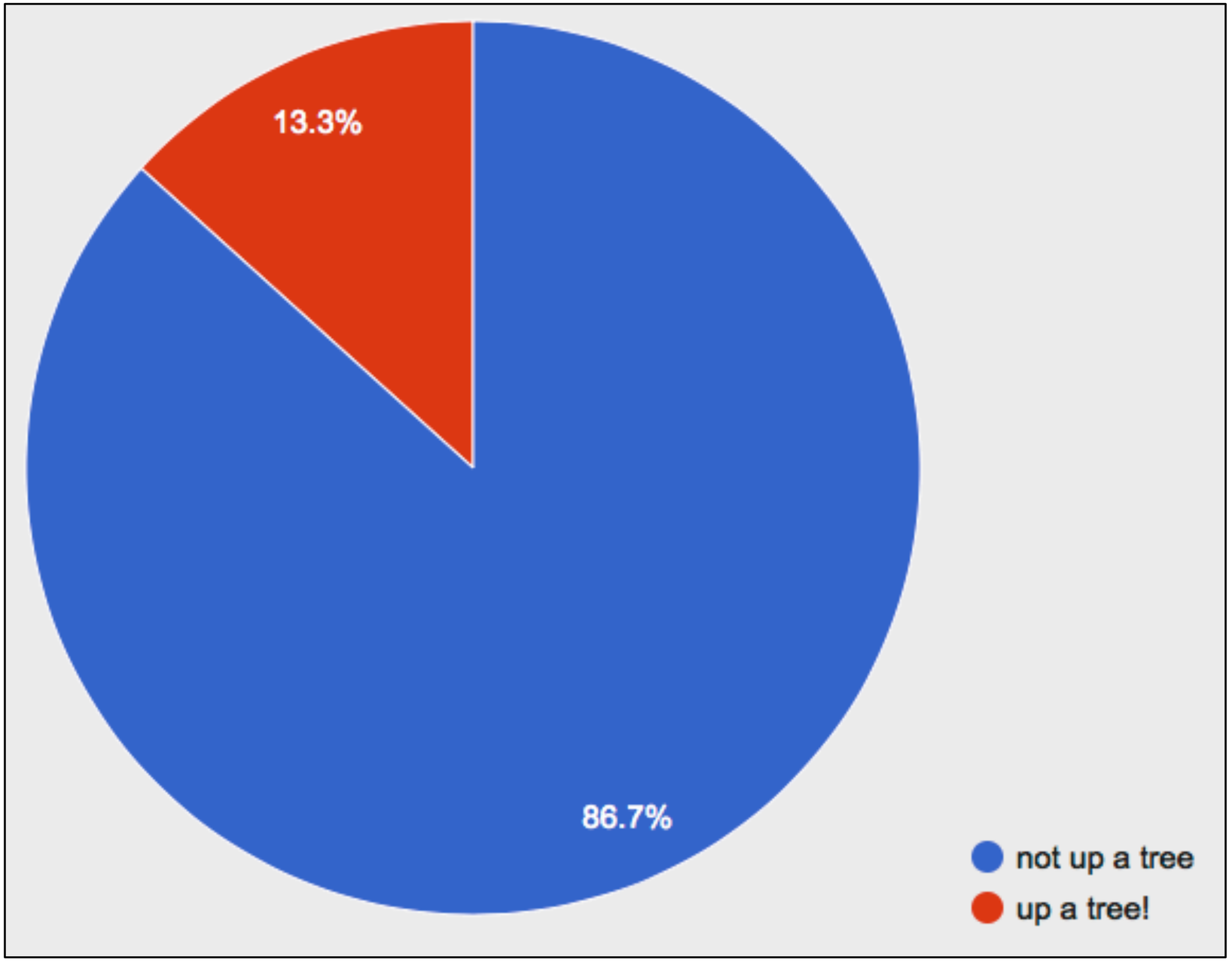
  Need a column to count values in, so we'd better put the up-a-tree-or-not answer into a table.

```
pie-chart-counts(cats-who-may-be-up-trees, "up-tree")
```

Plan

- ✓ Get just the rows about cats
- ✓ Figure out for each row if it involves a tree
- ✓ Make a pie chart

  Need a column to count values in, so we'd better put the up-a-tree-or-not answer into a table.


13.3%
86.7%
not up a tree
up a tree!

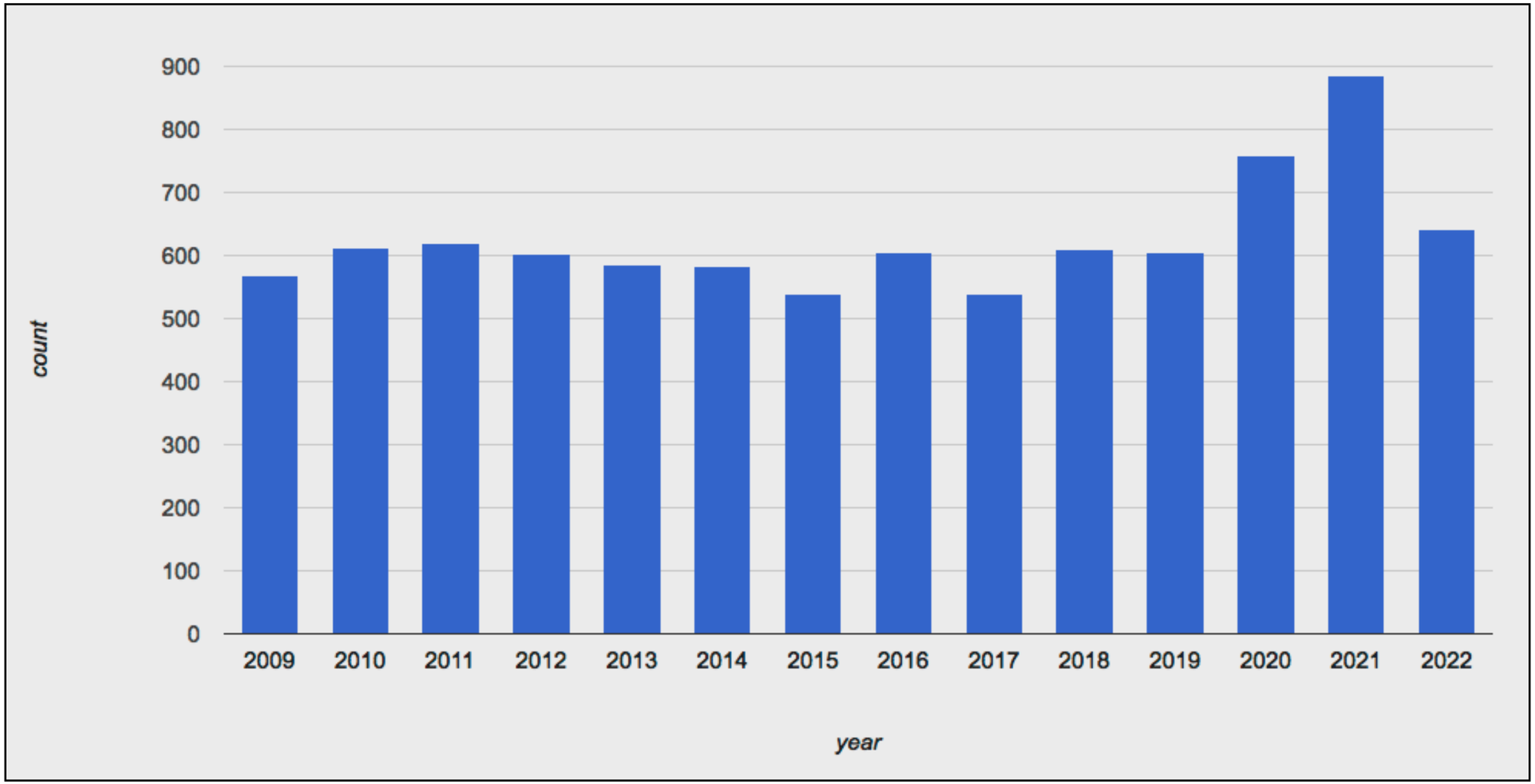
Slander!

Changes

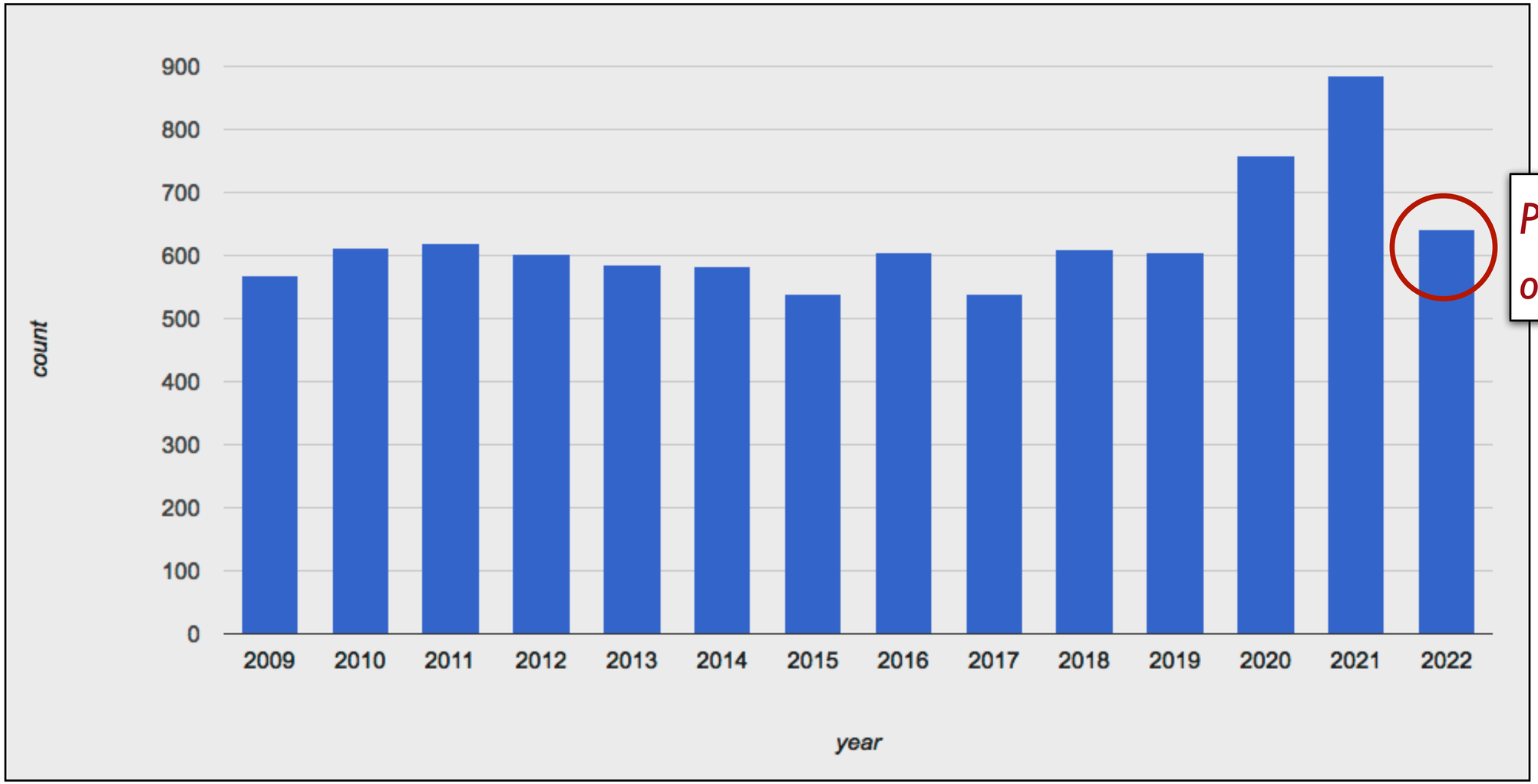
That’s enough pie for now.

Let’s make a bar chart to see the number of animal rescues per year.

```
>>> freq_bar_chart(rescue_data, "year")
```



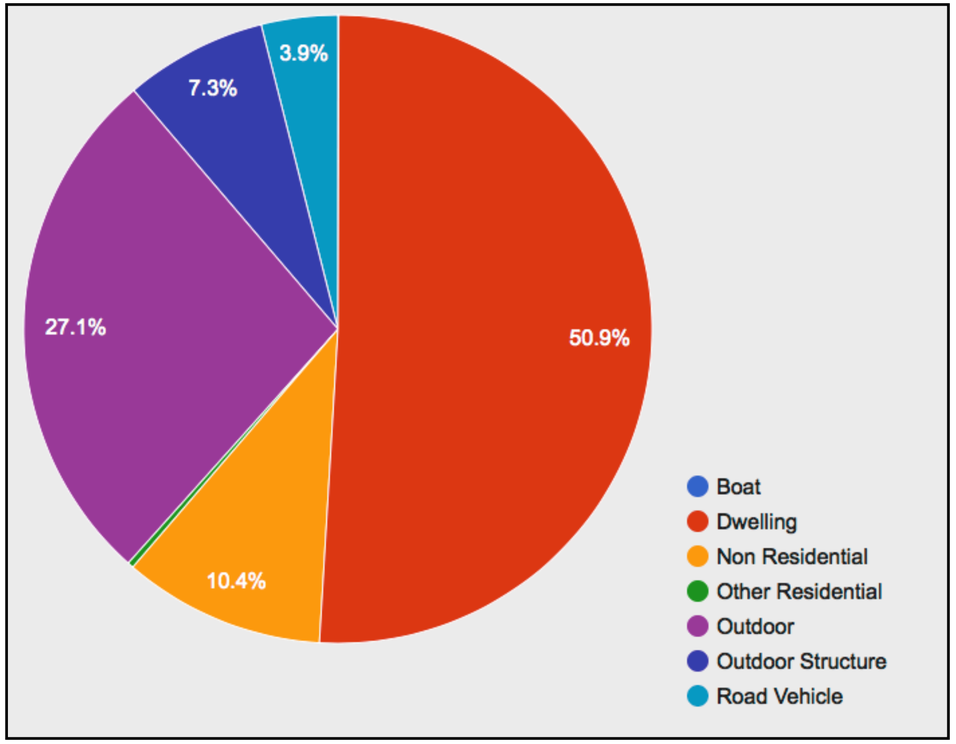
```
>>> freq_bar_chart(rescue_data, "year")
```

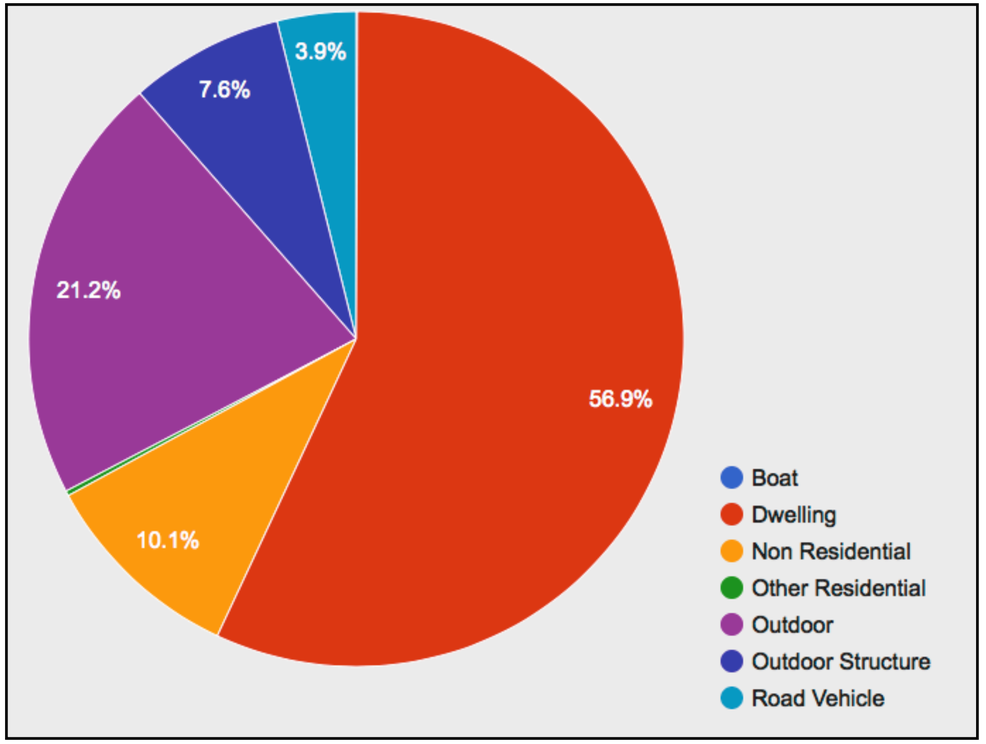


We can split the data into pre-2020 and 2020–2022 to explore this surge.

```
fun plague-year(r :: Row) -> Boolean:
  doc: "Return true if the row takes place during the COVID pandemic
(rounding up to full years)"
  (r["year"] >= 2020) and (r["year"] <= 2022)
end

rescue-data-plague = filter-with(rescue-data-clean, plague-year)
```

```
fun not-plague(r :: Row) -> Boolean:
  doc: "Return true if the row is from a year entirely before or after
the COVID pandemic"
  not(plague-year(r))
end

rescue-data-not-plague = filter-with(rescue-data-clean, not-plague)
```


3.9%
7.3%
27.1%
50.9%
10.4%
Boat
Dwelling
Non Residential
Other Residential
Outdoor
Outdoor Structure
Road Vehicle


3.9%
7.6%
21.2%
56.9%
10.1%
Boat
Dwelling
Non Residential
Other Residential
Outdoor
Outdoor Structure
Road Vehicle

What else could we compare for these time periods?

Class code:

https://tinyurl.com/101-2022-09-21

Acknowledgments

This class incorporates material from:

Kathi Fisler, Brown University