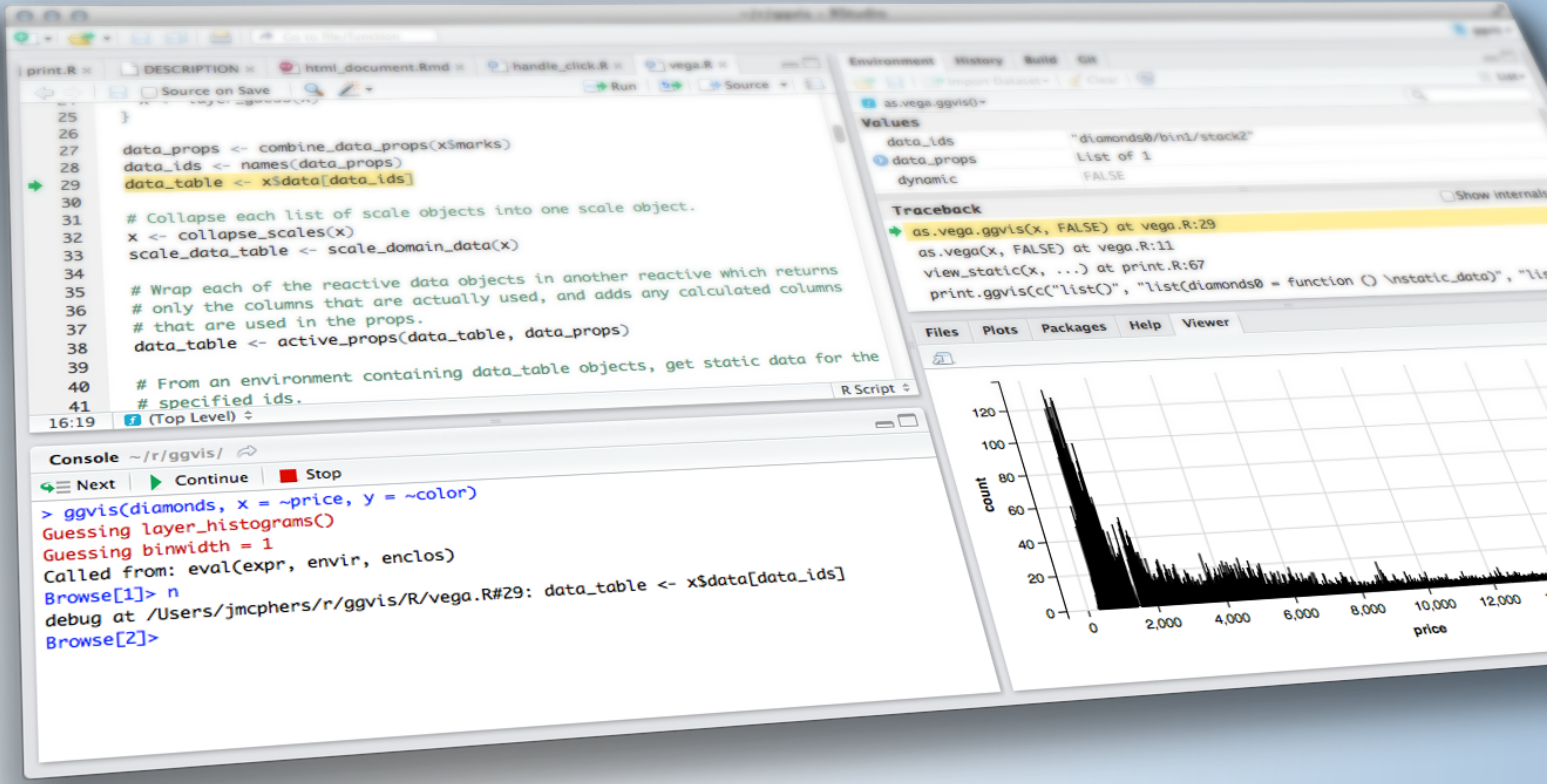


```r
}

data_props <- combine_data_props(x$marks)
data_ids <- names(data_props)
data_table <- x$data[data_ids]

# Collapse each list of scale objects into one scale object.
x <- collapse_scales(x)
scale_data_table <- scale_domain_data(x)

# Wrap each of the reactive data objects in another reactive which returns
# only the columns that are actually used, and adds any calculated columns
# that are used in the props.
data_table <- active_props(data_table, data_props)

# From an environment containing data_table objects, get static data for the
# specified ids.
```

Values

data_ids	"diamonds0/bin1/stack2"
data_props	List of 1
dynamic	FALSE

Traceback

```
as.vega.ggvis(x, FALSE) at vega.R:29
as.vega(x, FALSE) at vega.R:11
view_static(x, ...) at print.R:67
print.ggvis(c("list()", "list(diamonds0 = function () \nstatic_data)", "li
```

Console ~/r/ggvis/

Next Continue Stop

```
> ggvis(diamonds, x = ~price, y = ~color)
Guessing layer_histograms()
Guessing binwidth = 1
Called from: eval(expr, envir, enclos)
Browse[1]> n
debug at /Users/jmcphers/r/ggvis/R/vega.R#29: data_table <- x$data[data_ids]
Browse[2]>
```

API'S

Shiny from RStudio

OUTLINE

- Course review
- What are API's and what do they do?
- Making an API Call
  1. Build your URL
  2. Encode the URL
  3. Process the content
  - Spatial Data with Esri
- Geocode Example
- Shiny Example

Survey Monkey

Whats an API,

and what does it do?

API EXAMPLES

- WPRDC
- Census
- Geocoders
- Esri Online Datasets
- Online Weather APIs
- Sport Score API
- And more!

API'S

- Stands for: Application Programming Interface

- There are many kinds of API's
  - *Web service*
    - SOAP, XML-RPC, JSON-RPC, and **REST**
  - *WebSocket*
  - *Library-based*
  - *Class-based*
  - *OS functions and routines*
  - *Object remoting*
  - *Hardware*

REST API'S

- End points - different URL's that tell the webserver what data you would like
- It's essentially a website where you request different "end points"
- There are 5 types of Requests you can make
  - `GET` (what we will use the most in this course)
  - `POST` *(sometimes necessary for authentication, if you're trying to write data somewhere)*
  - `PUT`
  - `PATCH`
  - `DELETE`

Making an API Call

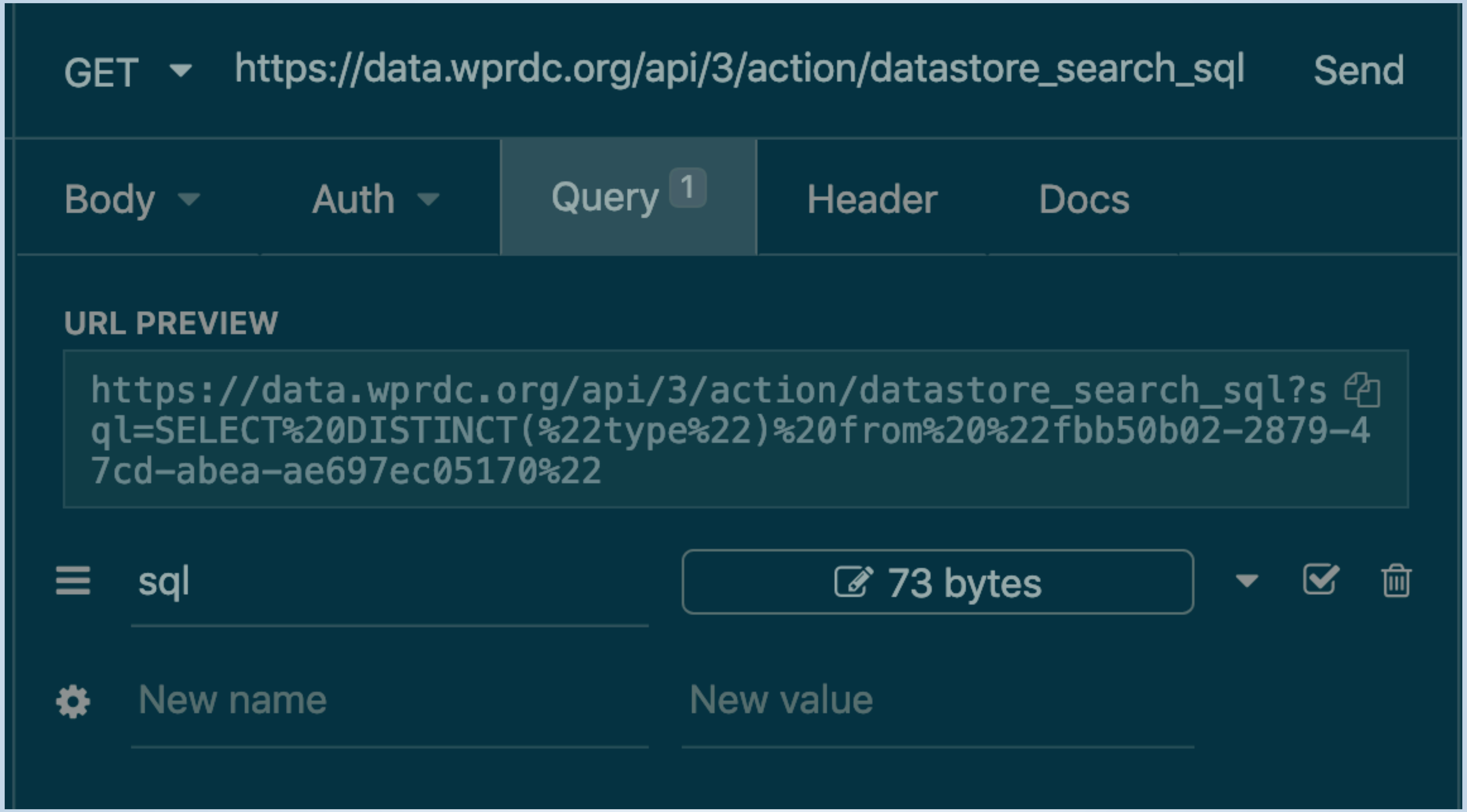
THE STEPS

1. Build your URL
2. Encode the URL
3. Process the content
4. Transform to a usable format

1. BUILDING YOUR QUERY

Many tools that make life easier:

- *Insomnia*
- *Advanced REST Client*
- *PostMan*
- *And others…*



```
https://data.wprdc.org/api/3/action/datastore_search_sql
```

```
https://data.wprdc.org/api/3/action/datastore_search_sql?sql=SELECT%20DISTINCT(%22type%22)%20from%20%22fbb50b02-2879-47cd-abea-ae697ec05170%22
```

sql | 73 bytes

WPRDC API Call in Insomnia

THE STEPS

1. Build your URL
2. Encode the URL
3. Process the content
4. Transform to a usable format

DEMO

```
URLencode("someString", repeated = TRUE)
```

THE STEPS

1. Build your URL
2. Encode the URL
3. Process the content
4. Transform to a usable format

CONTENT

- Any API call will have multiple portions of it.
- 2 most important are:
  - Content
  - status_code

GETTING TO THE CONTENT

- Most API calls you will be making are `GET` requests.

```
get <- httr::GET(“encodedURL”)
c <- jsonlite::fromJSON(content(get, "text"))
```

- Arguments you may need:
  - `$something` after `fromJSON` function
  - `flatten=TRUE`

ERRORS

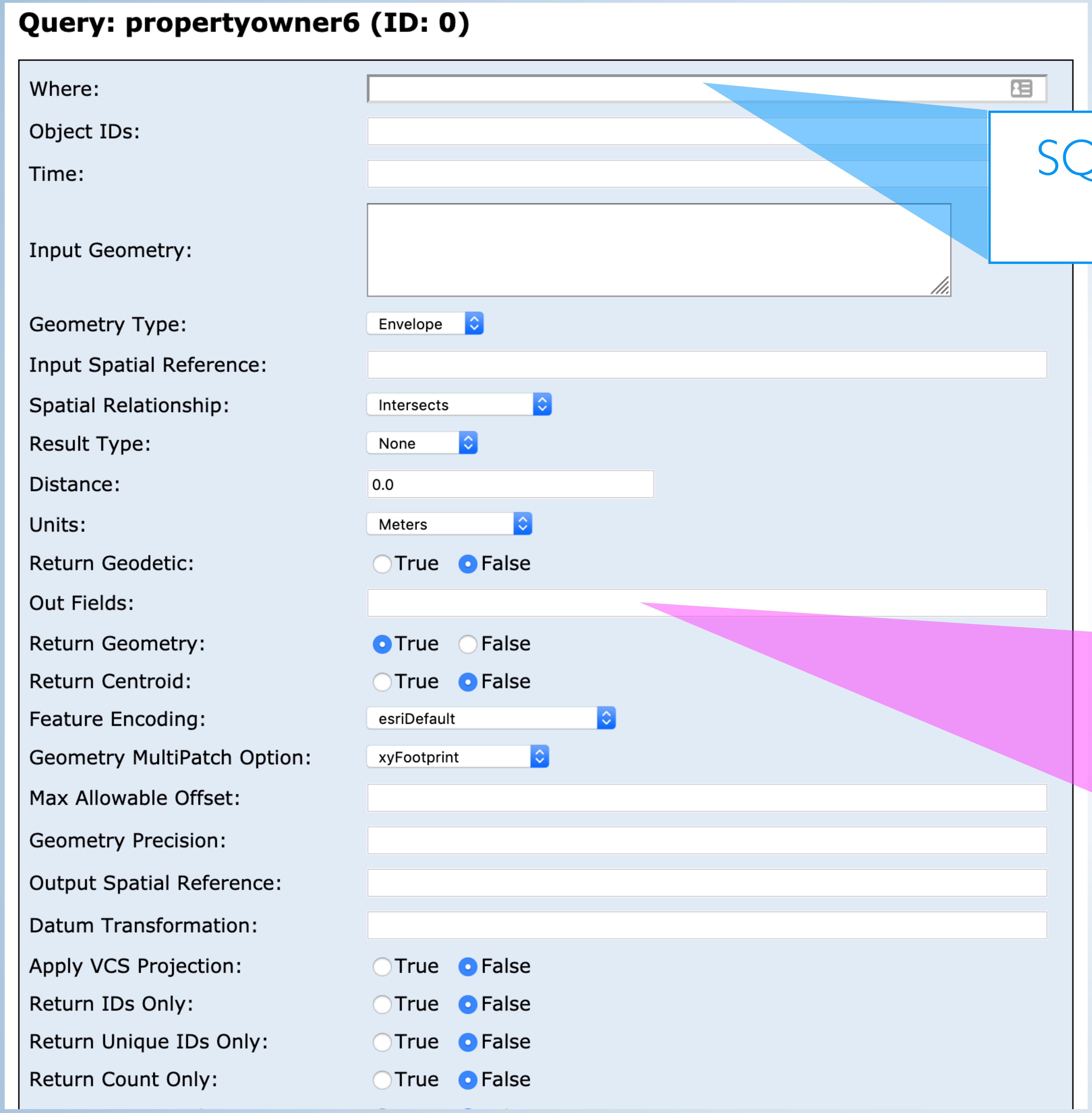
- Status codes indicate the result of the HTTP request.
  - ***100's*** - *info*
  - ***200's*** - *success*
  - ***300's*** - *redirection*
  - ***400's*** - *client error (you messed up)*
  - ***500*** *'s- server error (something went wrong on their end, but you still could have messed up)*

EXERCISE

- Open exercises/api_practice.Rmd and use the chunk labeled “Blotter”
  - Like last class generate an API call that downloads all of the data from the City of Pittsburgh Police Blotter
    - It might be easier to build the query in Insomnia or something else first
    - Stretch: After you have built a query that calls all of the data, add a group by or filter of some kind

10m 00s

Spatial Data



Query: propertyowner6 (ID: 0)

Where:	
Object IDs:	
Time:	
Input Geometry:	
Geometry Type:	Envelope
Input Spatial Reference:	
Spatial Relationship:	Intersects
Result Type:	None
Distance:	0.0
Units:	Meters
Return Geodetic:	True [✓ False]
Out Fields:	
Return Geometry:	[✓ True] False
Return Centroid:	True [✓ False]
Feature Encoding:	esriDefault
Geometry MultiPatch Option:	xyFootprint
Max Allowable Offset:	
Geometry Precision:	
Output Spatial Reference:	
Datum Transformation:	
Apply VCS Projection:	True [✓ False]
Return IDs Only:	True [✓ False]
Return Unique IDs Only:	True [✓ False]
Return Count Only:	True [✓ False]

SQL like where statement to get one the data you want

Same as the select portion of a SQL query

Source: Allegheny County Esri API

GETTING SPATIAL DATA

- For ESRI API's so long as your format is set to GEOJSON…

```
data <- readOGR("encodedURL")
```

- Its that easy

EXERCISE

- Open exercises/api_practice.Rmd and go to the chunk labeled “Esri”
  - Look at the fields on the May 2019 Election layer from the Allegheny County Esri API: https://services1.arcgis.com/vdNDkVykv9vEWFX4/ArcGIS/rest/services/Allegheny_County_Polling_Places_May2019/FeatureServer/0
    - Get all of the polling places in just the City of Pittsburgh and load it into R from the URL

10m 00s

SOLUTION

Solutions to both of todays exercises are in:
api_practice_solutions.R

THE STEPS

1. Build your URL
2. Encode the URL
3. Process the content
4. Transform to a usable format

Geocode

Example

DEMO

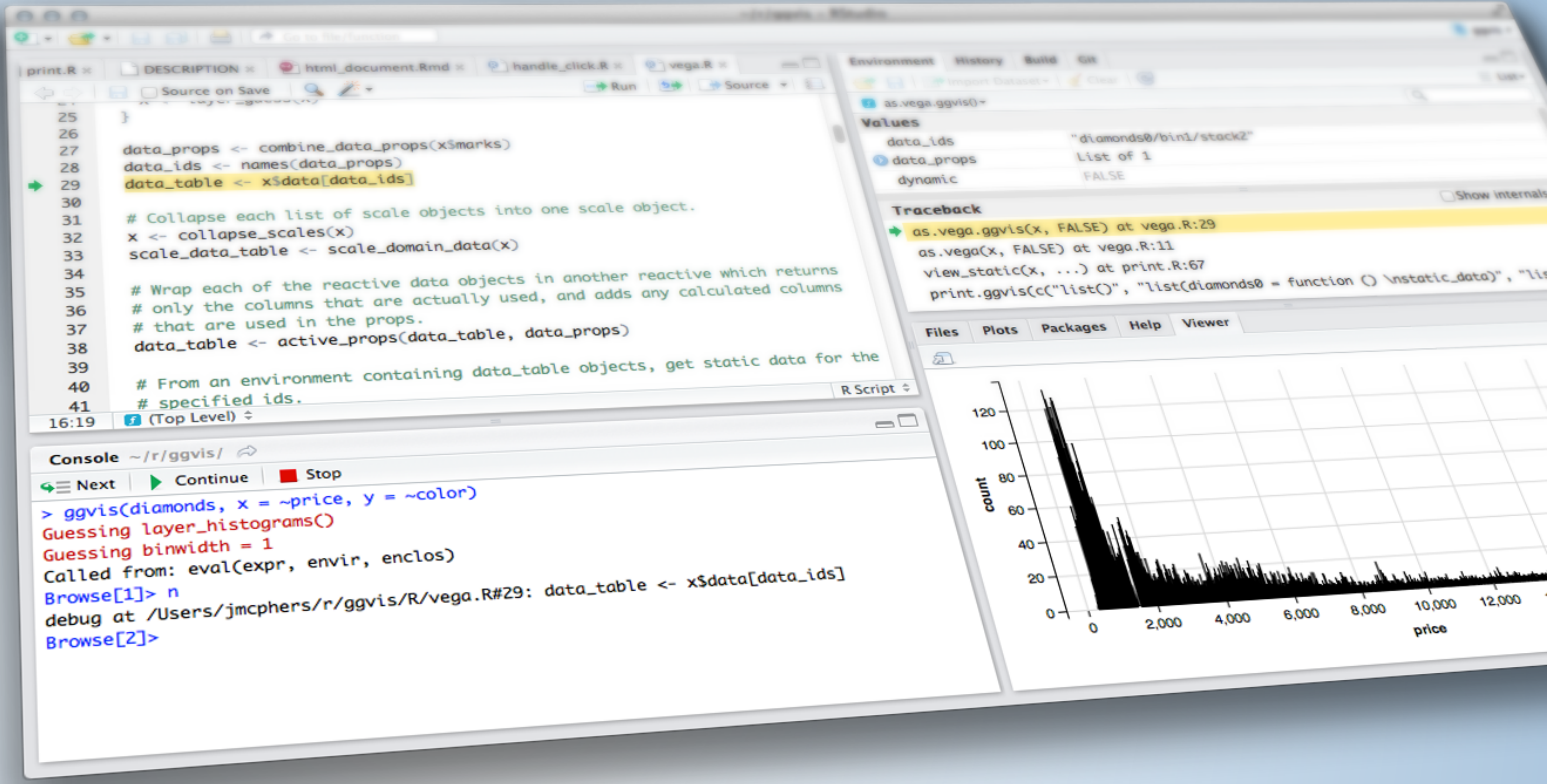
alco_geocode.R

Shiny

Example

DEMO

`app/311_dashboard.R`



```r
  }

  data_props <- combine_data_props(x$marks)
  data_ids <- names(data_props)
  data_table <- x$data[data_ids]

  # Collapse each list of scale objects into one scale object.
  x <- collapse_scales(x)
  scale_data_table <- scale_domain_data(x)

  # Wrap each of the reactive data objects in another reactive which returns
  # only the columns that are actually used, and adds any calculated columns
  # that are used in the props.
  data_table <- active_props(data_table, data_props)

  # From an environment containing data_table objects, get static data for the
  # specified ids.
```

```
> ggvis(diamonds, x = ~price, y = ~color)
Guessing layer_histograms()
Guessing binwidth = 1
Called from: eval(expr, envir, enclos)
Browse[1]> n
debug at /Users/jmcphers/r/ggvis/R/vega.R#29: data_table <- x$data[data_ids]
Browse[2]>
```

Values

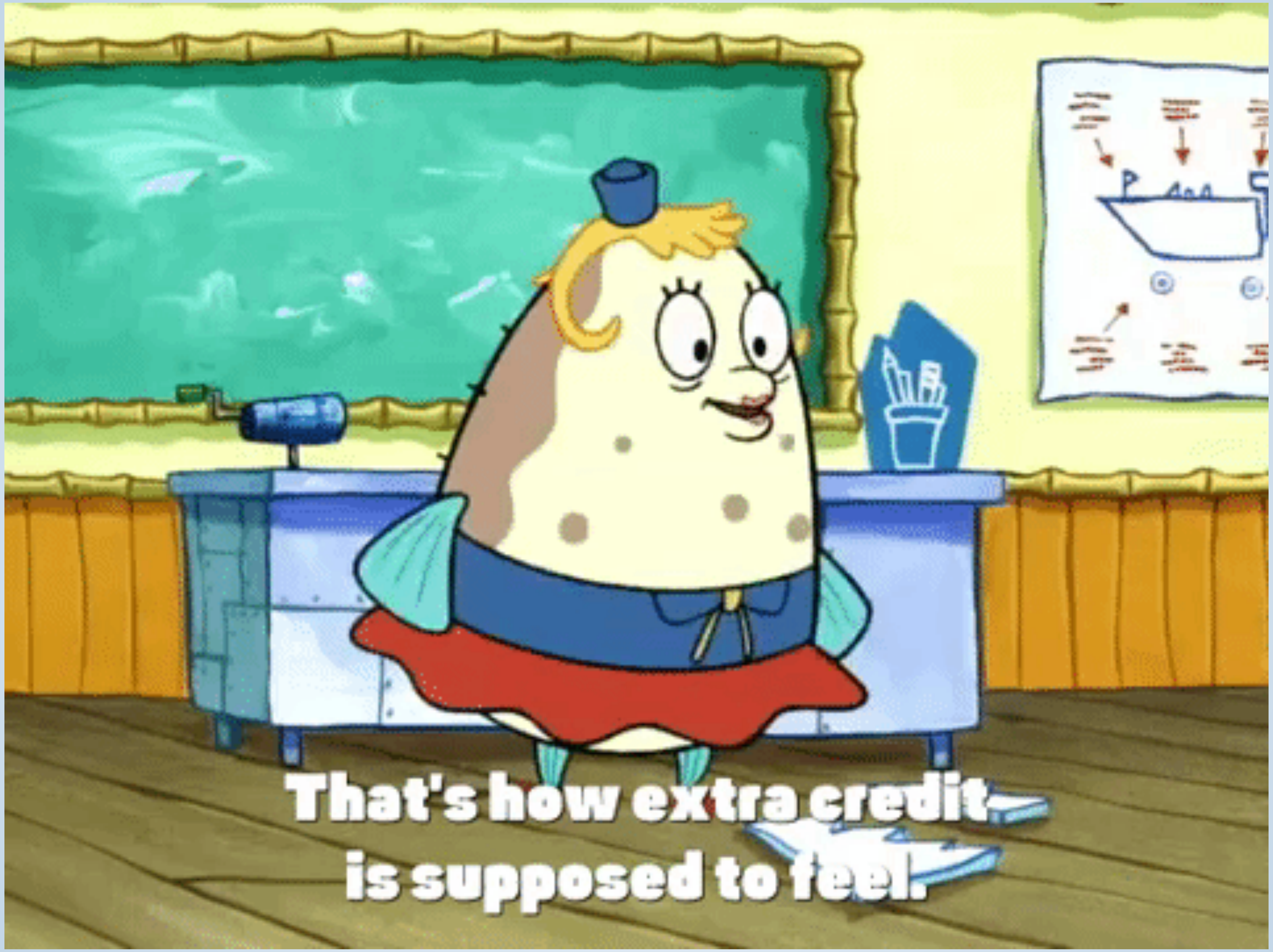
data_ids	"diamonds0/bin1/stack2"
data_props	List of 1
dynamic	FALSE

Traceback

```
as.vega.ggvis(x, FALSE) at vega.R:29
as.vega(x, FALSE) at vega.R:11
view_static(x, ...) at print.R:67
print.ggvis(c("list()", "list(diamonds0 = function () \nstatic_data)", "li
```

API'S

Shiny from RStudio™


That's how extra credit
is supposed to feel.