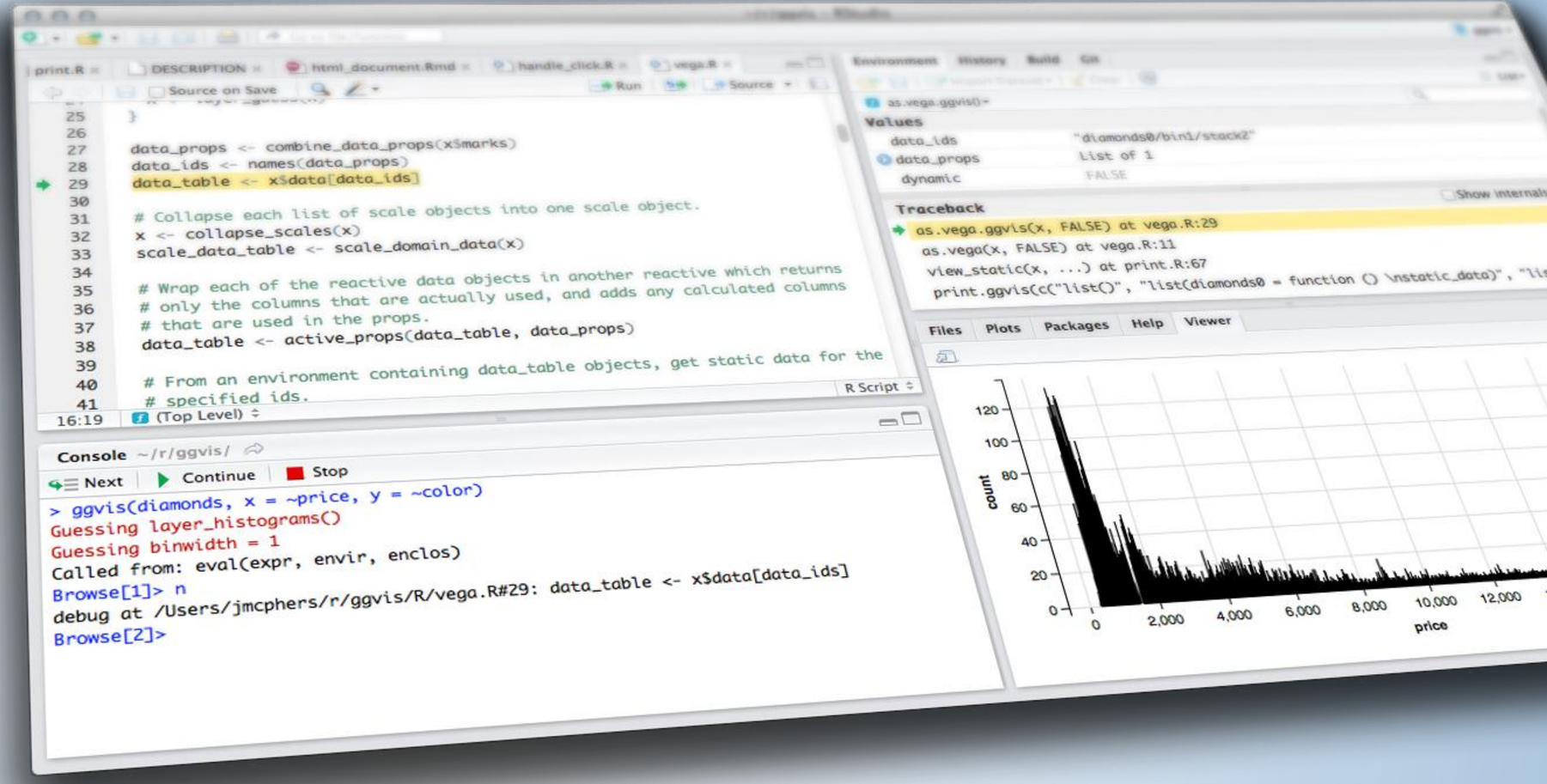


CONNECTING TO DATABASES & API'S

Shiny from RStudio™

EXERCISE

- Course Evaluation: https://cmu.smartevals.com/

SmartEvals!

5m 00s

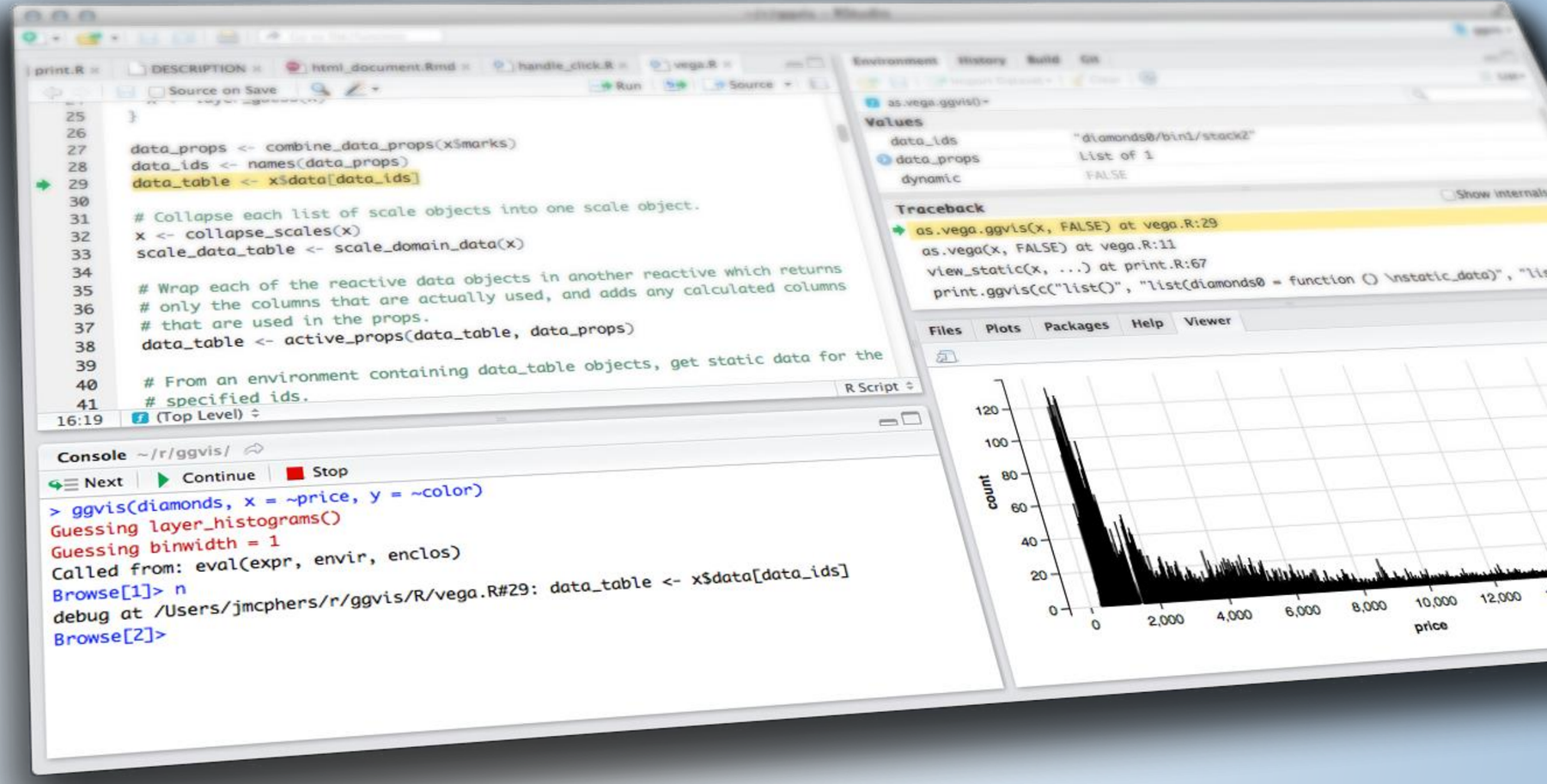
HOMEWORK 2 FEEDBACK

- In-code COMMENTS!
  - I want to know what you think you're doing.
- Thousands separators!
  - Value/info box: prettyNum()
  - DT: formatCurrency()
  - ggplot2: library(scales)
- **Reminder:** Next week we are in HBH 1208 for Homework Help and a *very* short lecture on bookmarking

OUTLINE

- SQL
  - SQL Basics
    - Constructing a query
    - Functions
  - Where should I write SQL queries?
  - Building Connections

- API’s
  - What are API’s and what do they do?
  - Making an API Call
    - Build your URL
    - Encode the URL
    - Process the content
    - Spatial Data with Esri
  - Geocode Example
  - Shiny Example



SQL DATABASES

Shiny from RStudio™

“I ~~rob banks~~ use databases because its where the ~~money~~ data is.”

–Willie Sutton

SQL

the structured language

“SQL is a domain specific language used in programming and [accessing]… data held in a relational database management system”

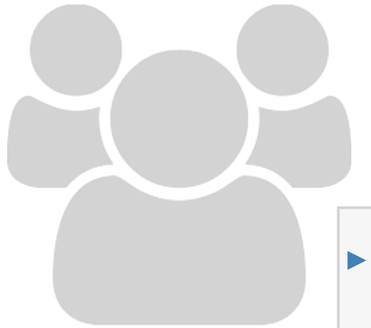
–Wikipedia

Structuring a query

QUERIES

ORDER	CLAUSE	FUNCTION
1	from	Choose and join tables to get base data.
2	where	Filters the base data.
3	group by	Aggregates the base data.
4	having	Filters the aggregated data.
5	select	Returns the final data.
6	order by	Sorts the final data.
7	limit	Limits the returned data to a row count.

Source: periscope data



EXERCISE

- Run apps/wprdc_sql.R
- Build a query that selects all of the crimes from the City of Pittsburgh Police Blotter
  - Hint 1: FROM would be the resource ID (1797ead8-8262-41cc-9099-cbc8a161924b)
  - Hint 2: The WPRDC uses a PostgreSQL backend
    - This means that anything that tables or columns that contain numbers or capital letters must be wrapped in double quotes

5m 00s

SOLUTION

```
SELECT * FROM "1797ead8-8262-41cc-9099-cbc8a161924b"
```

WHERE

BETWEEN … AND

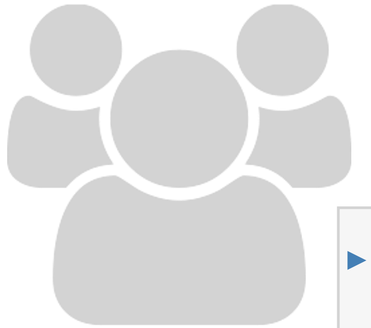
- BETWEEN
  - *Grab Values between two other values, like IN but for numeric values*
  - *Works like < and >*

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

IN STATEMENTS

- Useful for when you have an input that returns multiple
- This works the same way %in% does in R
- Checks to see if the value in the column matches *any* of the values in your list

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...)
```



EXERCISE

- Run apps/wprdc_sql.R
  - This time let's target 311 requests: 76fda9d0-69be-4dd5-8108-0de7907fc5a4
    - Use the BETWEEN function as a WHERE filter to get 311 requests from from the last week.
      - Stretch goal: Use the IN Filter to only get requests of the Potholes, Weeds/Debris and Overgrowth call types.

5m 00s

SOLUTION

```
SELECT * FROM  "76fda9d0-69be-4dd5-8108-0de7907fc5a4"
WHERE "CREATED_ON" BETWEEN (‘2022-12-01') AND ('2021-12-
07')
```

```
SELECT * FROM  "76fda9d0-69be-4dd5-8108-0de7907fc5a4"
WHERE "CREATED_ON" BETWEEN (‘2022-12-01') AND (‘2022-12-07')
AND "REQUEST_TYPE" IN (‘Potholes', 'Weeds/Debris', 'Overgrowth')
```

SELECT Functions and GROUP BY

SQL FUNCTIONS

- Sometimes you don’t just want the raw data
- You want to aggregate the data in SQL before you load it into R
  - Use another server to do the heavy lifting so you don’t have to!

DISTINCT

- DISTINCT()
  - Every unique value of a column.
  - Placing TWO columns inside will return unique instances of both columns:

```
DISTINCT(“REQUEST_TYPE”, “DEPARTMENT”)
```

MIN & MAX FUNCTIONS

- MIN()
  - Returns minimum value in a column(s)
- MAX()
  - Return maximum value in a column(s)

COUNT, AVERAGE, SUM

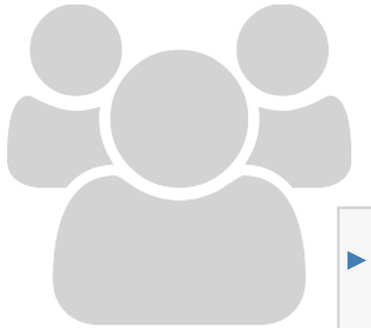
- COUNT() - returns the number of rows that your query returns
  - SELECT COUNT(*column_name)*
    FROM *table_name*
- AVG() - returns the average value of a numeric column.
  - SELECT AVG(*column_name*)
    FROM *table_name*
- SUM() - function returns the total sum - numeric columns only
  - SELECT SUM(column_name)
    FROM table_name

GROUP BY

- This is helpful for when you are doing any of the summary functions mentioned in the previous slides. (COUNT, SUM, MAX etc)

- Any column that isn't handled with a function should be included in your GROUP BY

```
SELECT column_name(s), max(column_name)
FROM table_name
WHERE condition
GROUP BY column_name(s)
```



EXERCISE

- Run apps/wprdc_sql.R
  - Build a query that counts the number crimes by neighborhood from the City of Pittsburgh Police Blotter

5m 00s

SOLUTION

```
SELECT
"INCIDENTNEIGHBORHOOD",
COUNT("CCR")
FROM "1797ead8-8262-41cc-9099-cbc8a161924b"
GROUP BY "INCIDENTNEIGHBORHOOD"
```

Where should I…

write my DB queries

SQL IDE’S

- There are a bunch of SQL IDE’s each database provider has their own
- If you’re in a workplace like mine with no standard then I suggest something like DBeaver because it connects to pretty much everything
- If not, then use whatever comes standard with the platform

DB connections in R

Not always easy

CONNECTING

- Database connectors require that your computer has the necessary software.
  - This will depend on what database type you are trying to connect to

ALLOWING HANDSHAKES

- To setup database connections you will need to install the proper drivers.
  - The steps for this can be found here: https://solutions.posit.co/connections/db/best-practices/drivers/
  - In general setup on Windows is a little bit easier since ODBC Data Source Administrator can be used
- Your machine may already have drives installed if you've already installed SQL IDE's such as: Microsoft SQL Server Management Studio, pgAdmin, DBeaver, or the MySQL Workbench

Storing credentials

FILE OR ENVIRONMENTAL VARIABLE

- You should never “hard code” your credentials into an app.
- Instead you should store them as environmental variables, or in a hidden file that you ignore in the Git Repository
  - Why?

If something requires that you to login, we can assume that not just anybody should be able to access it. Think of your credentials like your debit card and pin number

TWO OPTIONS

.env files

- Use the dotenv library
- File must be in the root directory of your app
- See .env example file in apps folder for formatting
- Include filename in .gitignore

JSON, CSV or other document

- Read the file in using the applicable file reader
- File location is more flexible
- Format dependent on file type
- Include filename in .gitignore

ESTABLISHING CONNECTIONS

- Each data base type has a different connection string and list of requirements.

```
conn <- dbConnect(odbc::odbc(), driver = "FreeTDS", server = "IP_or_HOST_ADDRESS",
port = 1433, database = "DBName", uid = un, pwd = pw, TDS_Version = "8.0")
```

- More on connection strings: https://db.rstudio.com/best-practices/drivers/#connecting-to-a-database-in-r

DB connections in Shiny

DATABASE POOLS

- Its not nice to have a bunch of active connections to your databases
- Pools are a great way to keep the connections ready
  - Keeps the connection ready to be checked out when needed
  - Closes the connection
    - Read more on using pools in Shiny: https://shiny.rstudio.com/articles/pool-basics.html

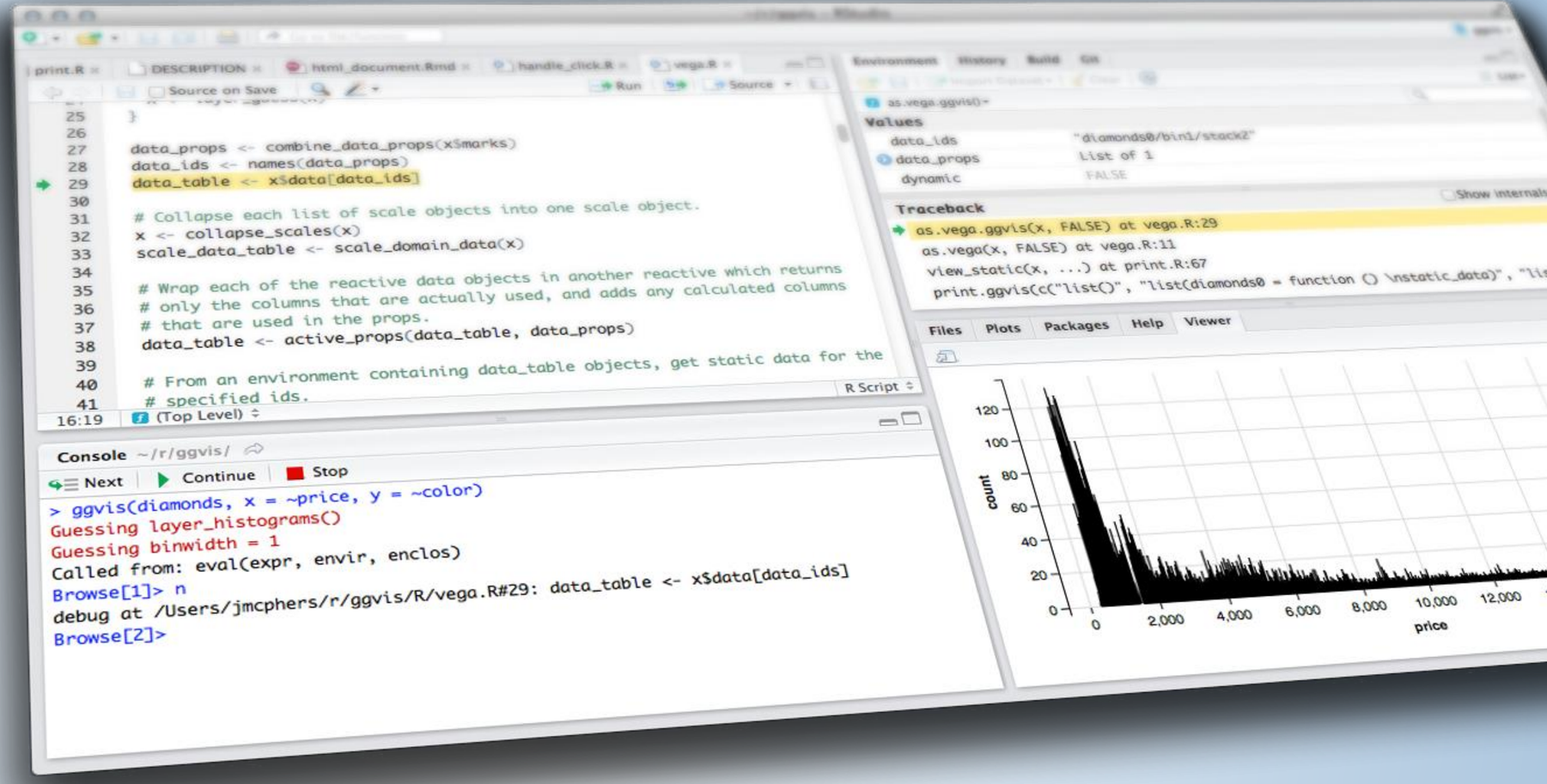
REACTIVEPOLL REVIEW

- **Check function:** is executed periodically and should always return a consistent value until the data changes
  - Note doesn't return `TRUE` or `FALSE`, instead it indicates change by returning a different value from the previous time it was called
  - **Value retrieval function:** is used to re-populate the data when the check function returns a different value
- We can use this in our apps to see if there's new data, and if not simply keep what the user has been using, and if not, load the updated data

DEMO

example_dbi.R

API'S



Shiny from RStudio™

Whats an API,

and what does it do?

API EXAMPLES

- WPRDC
- Census
- Geocoders
- Esri Online Datasets
- Twitter
- Spotify
- 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

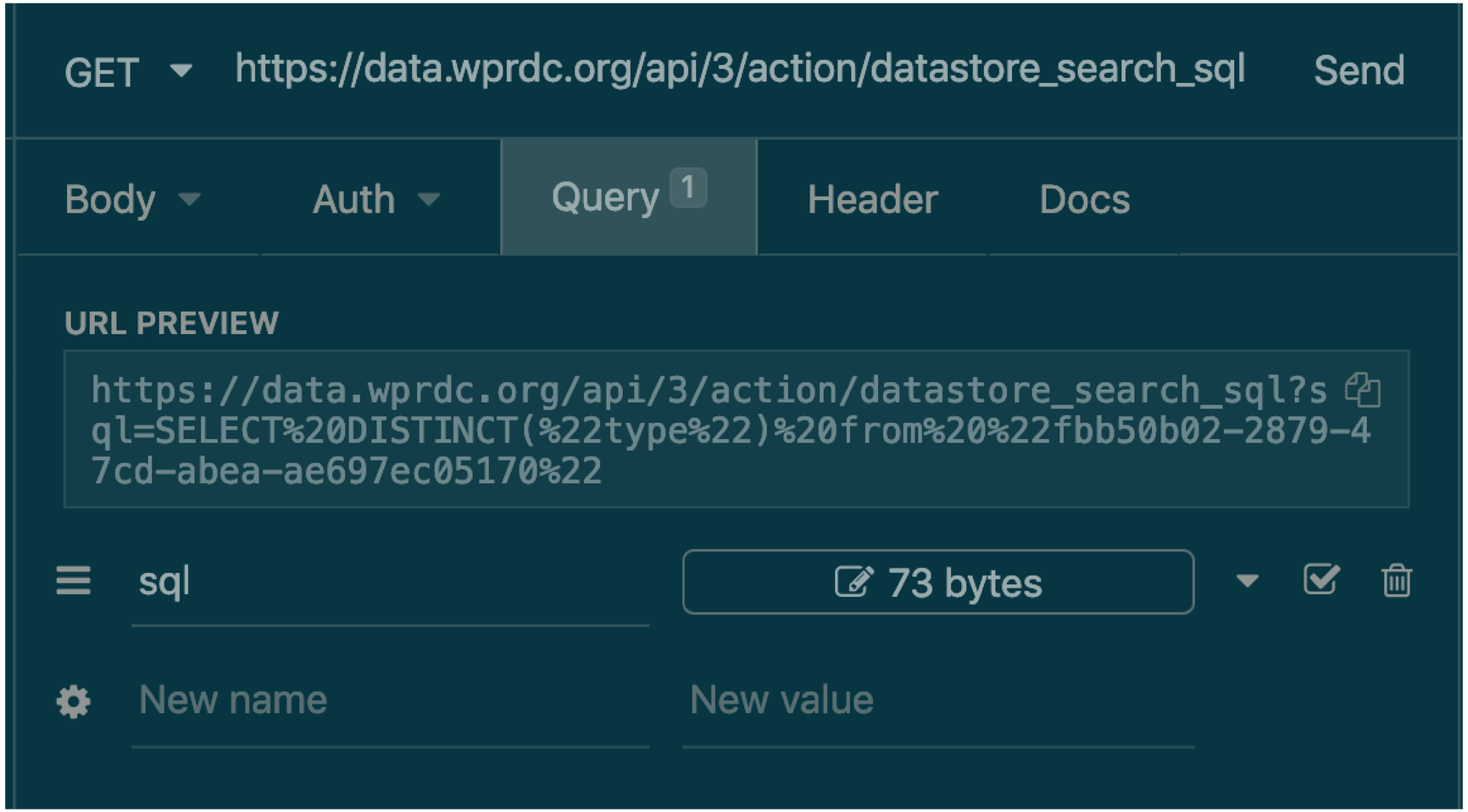
THE STEPS

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*
- *Your internet browser*
- *And others...*



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)*

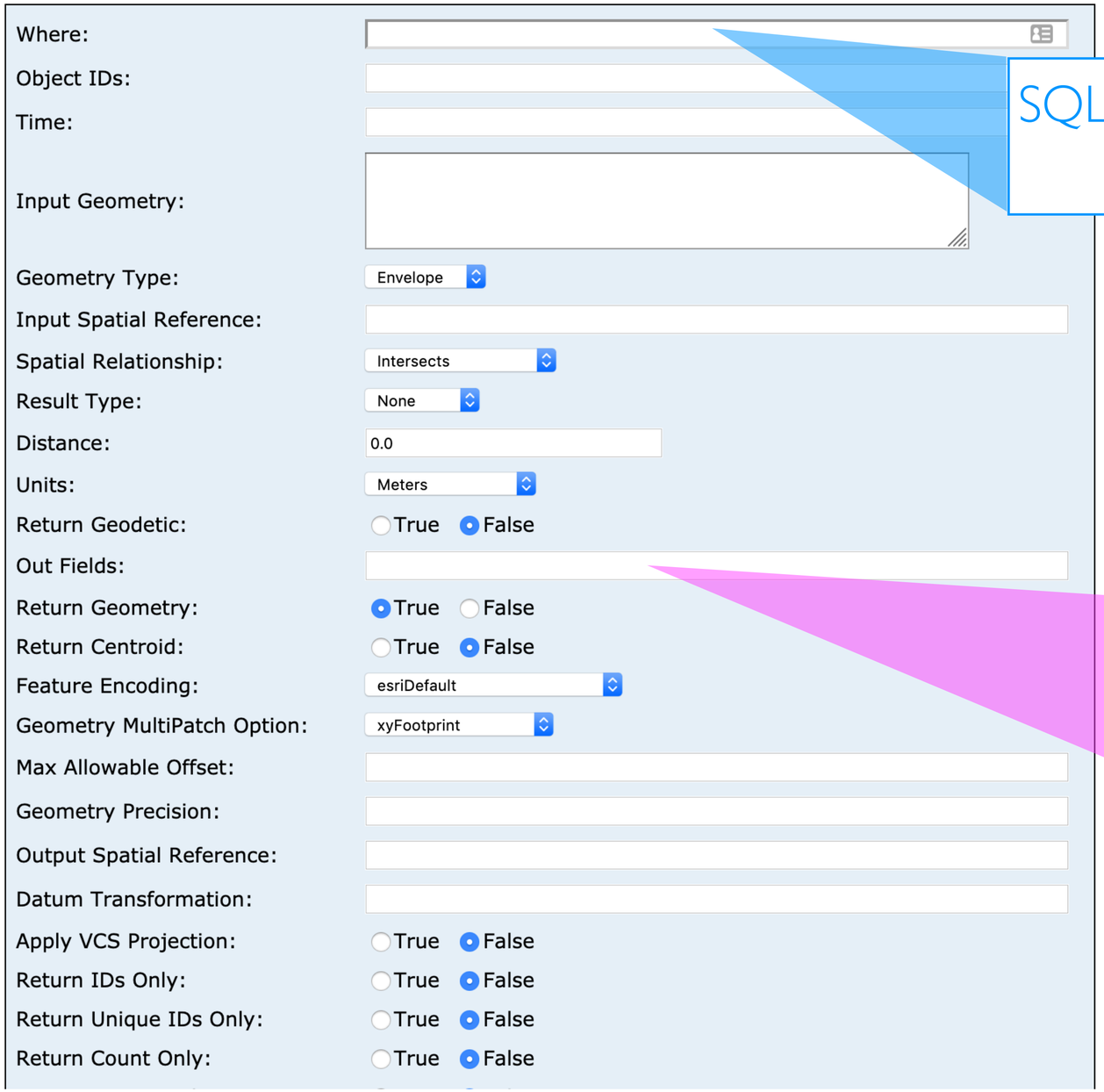
EXERCISE

- 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

Esri API Example

Query: propertyowner6 (ID: 0)



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 <- st_read("encodedURL")
```

- Its that easy

EXERCISE

- Open exercises/api_practice.Rmd and go to the chunk labeled “Esri”
  - Look at the fields on the Polling Places Election layer from the Allegheny County Esri API:
    - https://services1.arcgis.com/vdNDkVykv9vEWFX4/ArcGIS/rest/services/Polling_Places_2020/FeatureServer/0/query
    - 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

Other API’s

OTHER API'S

- Not all (most) API's will require you to do filtering or sql in them
  - This is mostly how Data Portal's API's work
  - Socrata Portals have a weird endpoint sql hybrid model using their SoQL framework: https://dev.socrata.com/docs/queries/
- Typically API's will have endpoints
- You will need to read up on the Documentation on the API you are attempting to use.

Geocode Example

DEMO

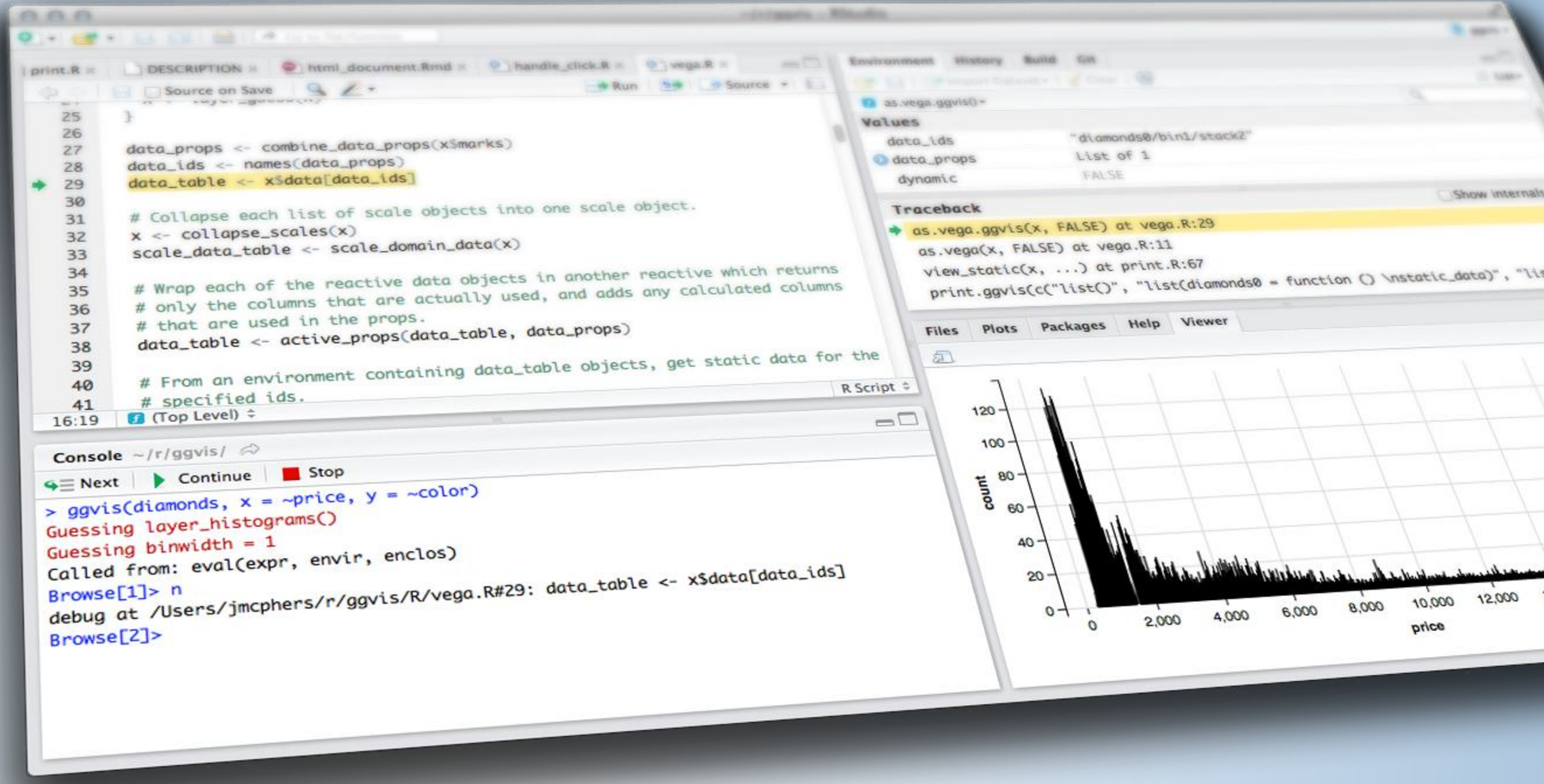
alco_geocode.R

Shiny

Example

DEMO

app/311_dashboard.R



API WRAPPERS

Shiny from RStudio™

API WRAPPERS

- There are a ton of packages which interact with API's in a more familiar "R" format.
  - These are mostly built on the httr package in some way
- Examples:
  - Tidy Census
  - tigris
  - spotifyR


That's how extra credit
is supposed to feel.