# Final project report:
# An interactive spatial and longitudinal data dashboard

Faysal Shaikh

CSI 695 - Fall 2021

# Final project report: An interactive spatial and longitudinal data dashboard

**Faysal Shaikh** **CSI 695 - Fall 2021**

## Table of contents

# List of figures

# 1 Background and data sources

The purpose of this project is to create an interactive data dashboard tool consolidating longitudinal and spatial data from different sources to allow policy leaders and other relevant stakeholders to sift through and ask questions of relevant data. This project is in part a collaboration with the Justice Community Opioid Innovation Network (JCOIN) and is thus also inspired by work of the National Institutes of Health (NIH) Helping End Addiction Long-term (HEAL) Initiative (NIH HEAL Initiative, 2021). As such, this project utilizes data relevant to the United States "opioid epidemic" introduced below.

## 1.1 The United States "opioid epidemic"

### 1.1.1 Origins: The undertreatment of pain and rise of opioid drugs

Prior to the 1980s, opioid drugs were not considered commonplace treatments for chronic pain in the United States. There is actually considerable evidence of a characteristic "opiopohobia," especially following the 1914 Harrison Narcotic Tax Act (Jones et al., 2018). The addictive potential of opioid drugs was understood following the American Civil War, during which these drugs were used in field hospitals to relieve surgery pains (Provine, 2011).

As such, this time period saw opioid addiction generally viewed through a medical lens. Unbeknownst to many, buildup to the passage of the Harrison Act potentially served as early warning signs of the impending nefarious "War on Drugs." In fact, President Theodore Roosevelt's appointed Opium Commissioner in 1908 had used explicitly racial claims, blaming opium for illicit sexual relations between white women and Chinese men and blaming cocaine for violence in African American men, to push for drug control at the federal level (Provine, 2011). We now know that similar techniques were used by Henry Anslinger, who from 1930 until 1962 served as the first commissioner of the Federal Bureau of Narcotics, to incite racial fearmongering and reshape the general view of addiction towards one of criminality via what we call today the "War on Drugs" (Provine, 2011).

Views of opioid medications began to change during the 1980s, as literature emerged highlighting the undertreatment of pain in the United States. These findings coincided with the surfacing of two pieces of literature, neither of which are considered to meet today's standard for scientific rigor, regarding an apparent low addiction potential for opioid drugs (Jones et al., 2018). Prior to this time, opioid prescriptions were typically reserved for short-term pain relief following surgery or cancer patients suffering from terminal illness. However, a burgeoning interest in the utility of opioid drugs for non-cancer pain, at times driven by misconceptions of non-cancer pain by underinformed cancer pain specialists, began to take hold of the medical community (Jones et al., 2018) and would lead to a gradual increase of opioid prescriptions during this time period (DeWeerdt, 2019).

The following years saw many notable patient advocacy and regulatory organizations, such as the American Pain Society (launching their "pain as a fifth vital sign" campaign in 1995), the Veteran's Health Administration (moving to adopt "pain as a fifth vital sign" in 1999), the Joint Commission (publishing standards for pain management in 2000), the Institute of Medicine, the Federation of State Medical Boards, and even the United States Drug Enforcement Agency, synergistically pushed for a more structured approach to pain assessment and management that heavily relied upon the prescription of opioid drugs (Jones et al., 2018). It was also this time period

that saw pharmaceutical companies devote significant resources towards lobbying, sponsorships, and marketing to promote their opioid products to the greatest extent possible (DeWeerdt, 2019). Some pharmaceutical companies took advantage of these opportunistic circumstances by peddling fraudulent claims to sell their opioid products. Purdue Pharma falsely marketed OxyContin, a new sustained-release formulation of the highly-addictive opioid drug oxycodone, as less addictive than other opioid painkillers; Purdue Pharma later admitted their knowledge of OxyContin as addictive in a 2007 lawsuit (DeWeerdt, 2019). Purdue even focused their initial marketing of OxyContin towards white communities, knowing that the image of the typical drug addict painted by Anslinger and the ensuing War on Drugs would serve their message of OxyContin as a non-addictive drug (DeWeerdt, 2019). As a result of these efforts, OxyContin prescriptions rose sharply from 670,000 in 1970 to 6.2 million in 2002 (Jones et al., 2018).

Despite best efforts from the pharmaceutical industry to hide the truth about the addictive potential of opioid drugs, it was only a matter of time before the reality of the situation would reveal itself.

### 1.1.2 Today: The opioid overdose crisis

The modern opioid epidemic in the United States is often described as taking place in 3 overlapping phases (DeWeerdt, 2019). The first phase began with the overprescription and abuse of opioid pharmaceuticals described earlier. The second phase, heavily involving heroin, saw heroin overdose deaths increase nearly fivefold in the United States from 2010 to 2016 (DeWeerdt, 2019). The early days of the third (present) phase saw the involvement of cheaper yet more potent opioids, namely fentanyl, such that opioid deaths from fentanyl and similar molecules increased by 88% per year between 2013 and 2016 (DeWeerdt, 2019).

American opioid overdose deaths in 2016 surpassed 42,000, at that point in time more than any previous year on record (U.S. Department of Health and Human Services, 2021). This record was subsequently broken by over 47,000 opioid overdose deaths in 2017 (National Institute on Drug Abuse, 2021). Despite the declaration of the opioid epidemic as a public health emergency in 2017 (U.S. Department of Health and Human Services, 2021), which saw a decline in opioid overdose deaths from 2017 to 2018, the previous record was surpassed once again by nearly 50,000 opioid overdose deaths in 2019 (National Institute on Drug Abuse, 2021).

Although pandemic coronavirus disease 2019 (COVID-19) has resulted in challenges obtaining recent unbiased estimates of opioid use (Haley & Saitz, 2020), we can safely continue to assume the omnipresence of the opioid epidemic in the United States today.

## 1.2 Sources of data relevant to the opioid epidemic

While numbers of opioid overdose deaths and other opioid-use-specific measures may provide useful information in understanding the scope of the opioid epidemic, they certainly do not provide the entire picture. Considering various social determinants of health (SDoH), the importance of which was especially validated in monitoring disease spread during the COVID-19 pandemic, may be required to better understand the factors that may contribute to opioid overdose outcomes.

Additionally, statistical relationships are necessarily derived at a group level. Thus we must ensure our perspective is not of individuals but of larger units of aggregation, e.g., geospatial

neighborhoods. An additional benefit of considering these larger units is the ability to compare areas based on their differing profiles of SDoH and other environmental factors, for example by quantifying a multi-dimensional "risk environment" as described by Rhodes (2002).

As such, this work serves to highlight two distinct data sources which may prove relevant in understanding SDoH and additional environmental factor profiles of specific levels of geospatial aggregation: the University of Chicago Opioid Environmental Policy Scan (OEPS) dataset (Kolak et al., 2021), and the University of Wisconin County Health Rankings & Roadmaps (CHR) dataset (University of Wisconsin Population Health Institute & Robert Wood Johnson Foundation, n.d.).

### 1.2.1 The Opioid Environmental Policy Scan (OEPS) dataset

The OEPS dataset, developed by Kolak et al. (2021) as a collaboration between the University of Chicago Healthy Regions & Policies Lab and the University of Chicago Center for Spatial Data Science, utilizes a "risk environment framework" approach, as first described by Rhodes (2002), to consolidate data from various sources into six "spheres of influence": policy, health, demographic, economic, built environment, and COVID-19 (Kolak et al., 2021). This conceptual model for this data is shown in Figure 1.1 below (on page 6).

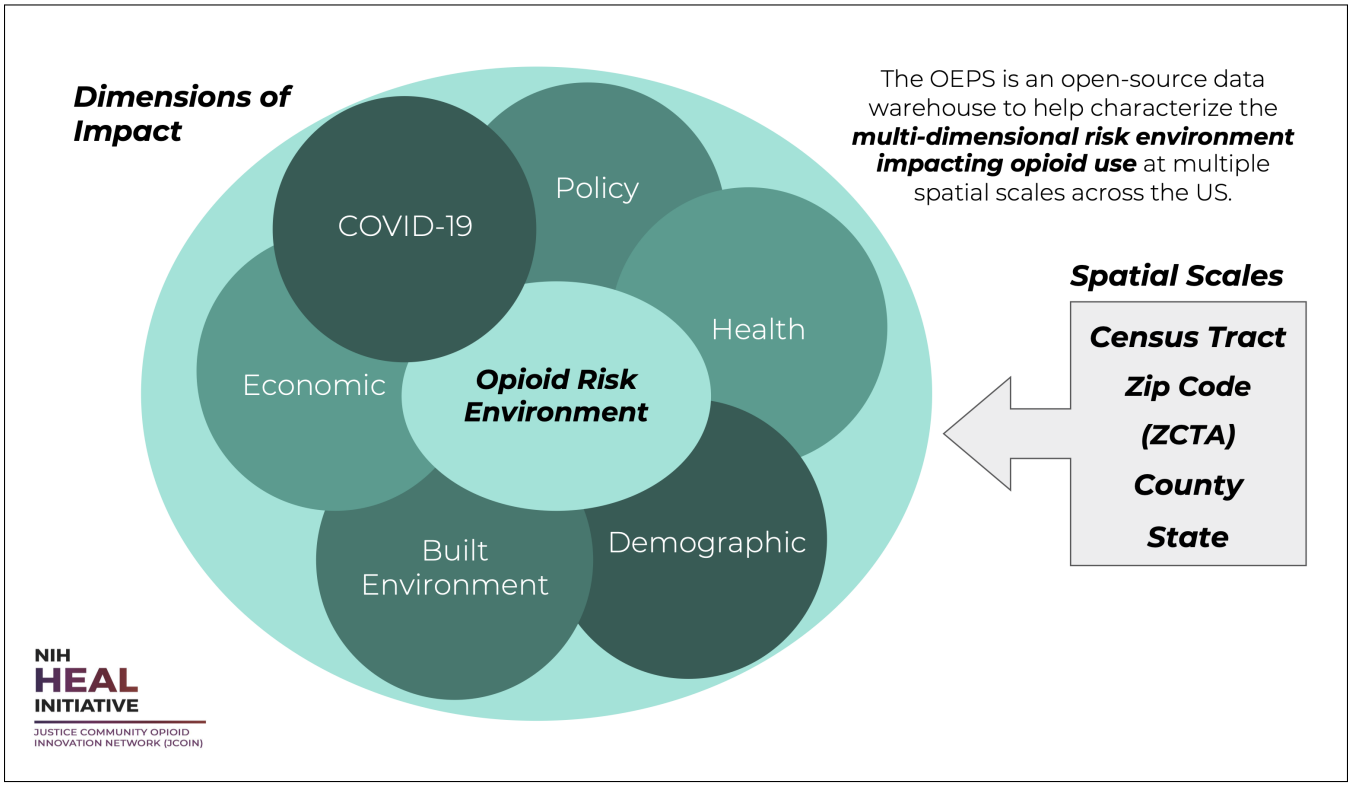


Figure 1.1: Diagram of OEPS dataset.

The OEPS dataset is a nationwide extension of previous work by M.A. Kolak et al. (2020) highlighting the utility of the risk environment approach in understanding various health outcomes, including opioid-related overdose, in rural Southern Illinois between 2015 and 2017. In appreciation

of "rural risk environment" analyses (M.A. Kolak et al., 2020) and of the expertise of the University of Chicago as the JCOIN Methodology and Advanced Analytics Resource Center (MAARC) (Kolak et al., 2021), this project hopes to utilize OEPS data to start with a multi-dimensional risk environment framework of the opioid epidemic at varying geospatial scales across the entire United States.

OEPS data were obtained via the "Filter Data and Download" section of the OEPS Explorer web application (https://oeps.netlify.app/download). "County" was selected under the "Filter by Scale" heading and downloaded to the project working directory. The data download interface is shown below in Figure 1.2 (on page 7).

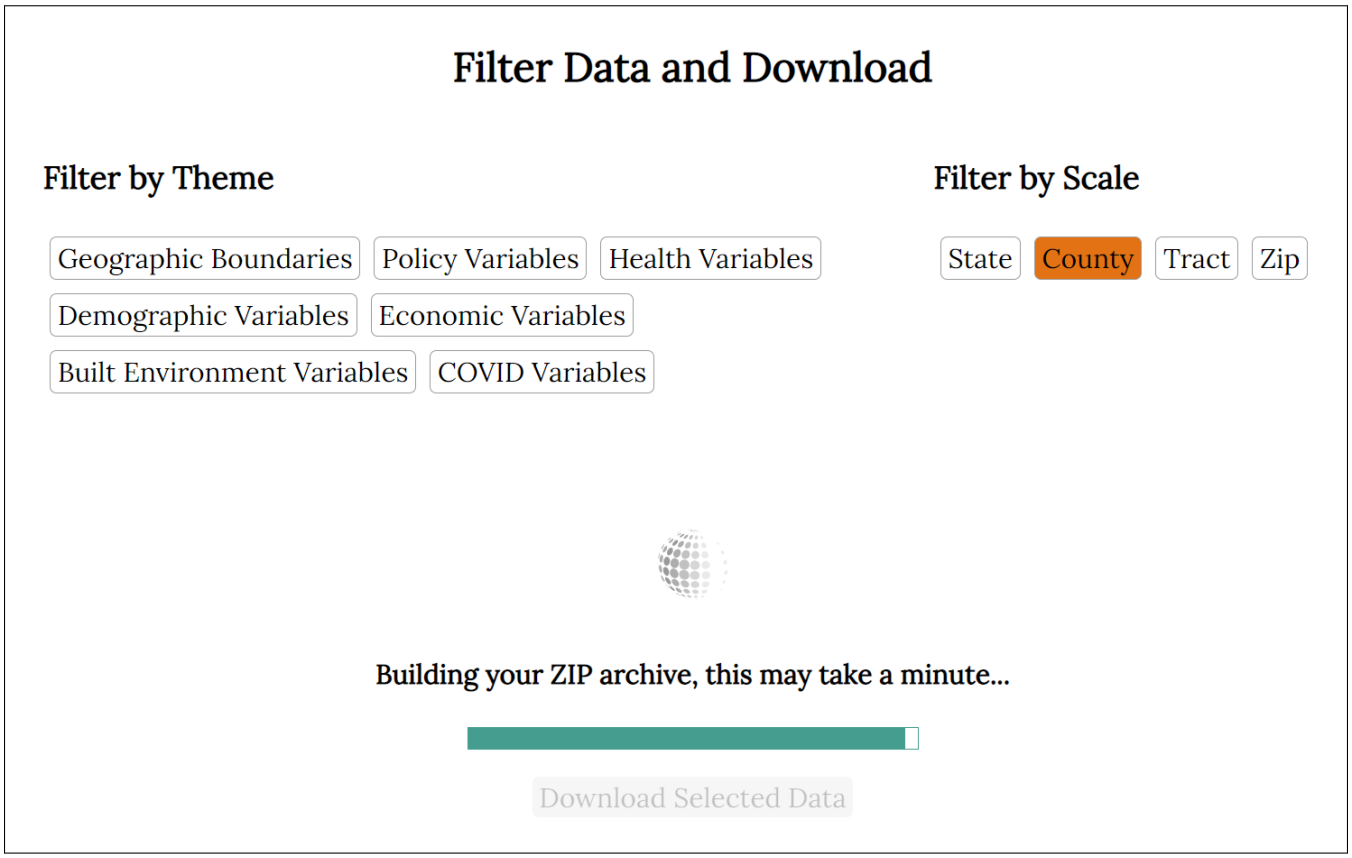


Figure 1.2: OEPS Explorer data download interface.

### 1.2.2 The County Health Rankings & Roadmaps (CHR) dataset

The CHR dataset (University of Wisconsin Population Health Institute & Robert Wood Johnson Foundation, n.d.), released annually since 2010, is a collaborative effort between the University of Wisconsin Population Health Institute and the Robert Wood Johnson Foundation to consolidate data from various sources and publish health rankings that consider both health outcomes and modifiable health factors for each of over 3,000 counties and county equivalents in the United States (Remington et al., 2015). The conceptual model for CHR data is shown in Figure 1.3 (on page 8) below.

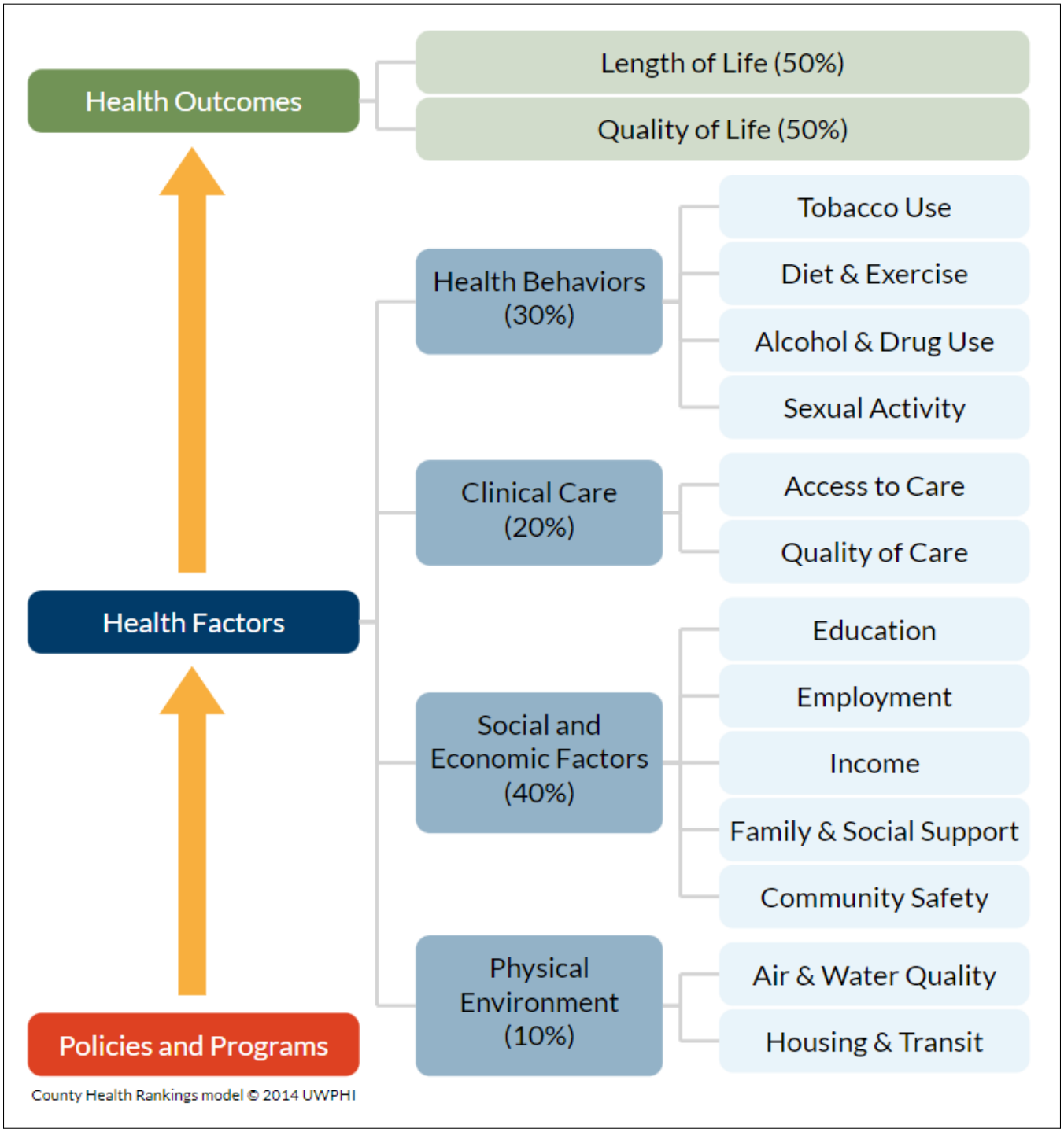


Figure 1.3: Diagram of CHR dataset.

Although CHR data provide a more general picture of health and health factors when compared to the more opioid-use-related focus of OEPS data, the utility of these data in drawing conclusions of health within a county are well-demonstrated. For example, Remington et al. (2015) analyzed 2014 CHR data and found premature death rates to be more than twice as high in bottom five

healthy counties when compared with top five healthy counties in each state.

While the dimensional richness of the multifaceted opioid risk environment is a strength of the OEPS dataset, data for each geospatial entity (without considering dis-/re- aggregation across multiple spatial scales) are only collected at a single timepoint, or "cross-sectionally" in time. As a result, face-value similiarites (or differences) seen "between subjects" (in this case, between geospatial entities, such as counties) may otherwise be characterized differently when taking into account a bigger picture that also includes "within-subject" variation over time (for example, repeated measurements of within the same geospatial entity), also known as "longitudinal" data. By nature of its yearly releases, CHR data are in fact longitudinal. This project thus attempts to consolidate cross-sectional and longitudinal data regarding SDoH and other environmental factors by merging the OEPS and CHR datasets in order to paint a more detailed multidimensional spatial and longitudinal picture of the opioid risk environment in the United States.

CHR data were obtained via either the "Rankings Data & Documentation" section (https://www.countyhealthrankings.org/explore-health-rankings/rankings-data-documentation) for 2020-2021 data, or via the "National Data & Documentation: 2010-2019" (https://www.countyhealthrankings.org/explore-health-rankings/rankings-data-documentation/national-data-documentation-2010-2019) section for historic data, previewed below in Figure 1.4 (on page 9). Relevant "County Health Rankings National Data" files were downloaded to the project working directory.


Figure 1.4: CHR historic data files download page.

# 2 Database design and creation

Following the introduction of these data sources, this work now turns to the discussion of designing a suitable database to store the data and later serve as the data source for the interactive dashboard.

## 2.1 Special considerations for spatial OEPS data

As previously mentioned OEPS data contain numerical measures of various variables in relevant risk environment framework spheres of influence. However, OEPS data also includes geographic Shapefiles for utilization with geographic information system (GIS) softwares and packages. As the end goal for this project is to create an interactive data dashboard that adequately visualizes spatial data, the primary software tools utilized in this project (including the database itself) must be able to handle and store these geographic Shapefiles appropriately.

## 2.2 Special considerations for longitudinal CHR data

Additionally, CHR data have been described previously to have been collected annually per county since 2010. While there exist specific software solutions designed as databases for longitudinal data storage, the primary issues arising with longitudinal data are less about software compatibility and more about database design. More clearly, longitudinal data are characterized as repeated measurements of the same variables so the database solution utilized for this project thus must be designed by the database developer to support repeated measures in two unique use-cases: (1) clear distinction of data from the same geospatial entity collected during different years, and (2) aggregation of the same measure across specific time period for time-averaging purposes.

## 2.3 Choice of database management system (DBMS)

After evaluating the requirements to store and manage our desired data, the time now comes to select a database management system (DBMS) software solution for our data. For its excellent compliance with structured query language (SQL), extensive documentation base, consistent release schedule, high compatibility in linking with other software tools, and myriad of features, this project has selected PostgreSQL (The PostgreSQL Global Development Group, 2021) as the DBMS for storage of OEPS and CHR data.

PostgreSQL began as `POSTGRES` in 1986 at the University of California at Berkeley and has seen around 30 years of active development since. As such, PostgreSQL is still often referred to as "Postgres" today. PostgreSQL is an open-source "object-relational database system" that is compatible with all major operating systems, and has maintained compliance with the popular "atomicity, consistency, isolation, and durability" (ACID) database transactions standard since 2001 (The PostgreSQL Global Development Group, 2021). These reasons make PostgreSQL a fine choice for most standard DBMS use-cases.

PostgreSQL is also an excellent choice for handling both spatial and longitudinal data. PostgreSQL is known for the extremely powerful PostGIS extension (Open Geospatial Consortium, 2021) that extends compatibility of PostgreSQL to handle and store geographic Shapefiles while also providing additional functionality such as geographic queries. While there exist dedicated longitudinal database management systems (e.g., TimescaleDB: https://www.timescale.com/, which

is built on top of PostgreSQL), any standard relational DBMS can be utilized to handle repeated measures in a similar manner to "long" (versus "wide") tabular data format. For the sake of brevity, additional specification of longitudinal data storage for this project will be reserved for the following sections.

## 2.4 Conceptual database design

We now shift our focus towards the task of the database developer to architecture an appropriate relational database that will meet the requirements of our data while keeping simple our downstream tasks connecting the DBMS to our data dashboard.

### 2.4.1 Development of entity-relationship (ER) models

An entity-relationship (ER) model utilizes domain knowledge alongside database design concepts to transform a combination of data and database use requirements into a model connecting relevant "entities" to one another via appropriate "relationships." This is generally an open-ended problem where any of several possible sets of entities and relationships may adequately serve the purposes of the end user. Some potential ER models may be better (or worse) suited than others, especially when considering downstream steps for the data (e.g., linking an interactive data dashboard), so domain knowledge is extremely valuable in this design process.

Despite this flexibility in conceptualization of ER models, the ER diagram generation procedure typically follows several conventions: an entity is typically represented as a square or rectangle; one of potentially multiple attributes of an entity is typically represented as an ellipse; a relationship between two entities is typically represented as rhombus (or colloquially a "diamond"); line segments typically connecting the aforementioned shapes where appropriate. Additionally, relationships between entities may be specified as "one-to-one," "one-to-many" (or vice versa), or "many to many," where "one-to-" sides of relationships contain visual arrowheads as a distinction from "many-to-" sides of relationships (which do not contain arrowheads). The aforementioned conventions were used in generating the pertinent ER diagrams included below (Figure 2.1 on page 13, Figure 2.2 on page 14, and Figure 2.3 on page 15).

**Cross-sectional and spatial OEPS dataset.** We begin this design process by considering the cross-sectional OEPS dataset. Let us consider the relevant heirarchical geographic information of a given geospatial "entity", e.g., a county, by specifying the following entities in our ER model: "county," "state," and "geography."

The "geography" entity is meant to serve as an oversimplified generalization of any Shapefile and related geometric GIS object data. We represent this in our ER diagram as an attribute called "geom." We consider the "geography" entity to have a one-to-one relationship with our "county" entity, such that every county may be described by one and only one geometry component (for lack of a better term), and vice versa.

We consider our "county" entity to showcase a many-to-one relationship with our "state" entity, such that each county may belong to only one state, but each state may contain many counties. Our "state" entity will only consist of a unique "state name" attribute. However, a "county" will exhibit a "FIPS_ID" attribute (leading to derived attributes of "county_ID" and "state_ID") as well as a "county_name" attribute.

We then turn our focus towards the measures included in the dataset. As a way of handling measures in an extremely general, topic-agnostic manner, we resort to creation of a generic "measure" entity and a corresponding "metadata" entity.

The "measure" entity is considered to exhibit a many-to-many relationship with the "county" entity, such that each measure may be taken in multiple counties and each county may have multiple measures taken. Relevant attributes of the "measure" entity include a "measure_ID," "measure_name," "value," and "unique_ID." This "unique_ID" attribute plays particular importance that will be revealed in a later ER diagram and the ensuing derived relational schema.

Our "metadata" entity serves to hold any additional information for a "measure" not worth storing as a direct attribute of the "measure" entity itself. We specify a one-to-one relationship between the "measure" and "metadata" entities, such that each measure corresponds with one specific corresponding entry of metadata (and vice versa). We shall specify the only attribute of the "metadata" entity as a "measure_descrip" to reference for each "measure."

This concludes our specification of our ER model for our cross-sectional OEPS dataset. Many of these entities and relationships will be preserved across different datasets (as especially becomes apparent upon combinind the datasets), so these descriptions will not be repeated in the ensuing descriptions of the ER model for the longitudinal CHR dataset as well as for the combined OEPS and CHR dataset.

The resulting ER diagram that corresponds with our specification of the OEPS data can be seen in Figure 2.1 below (on page 13).

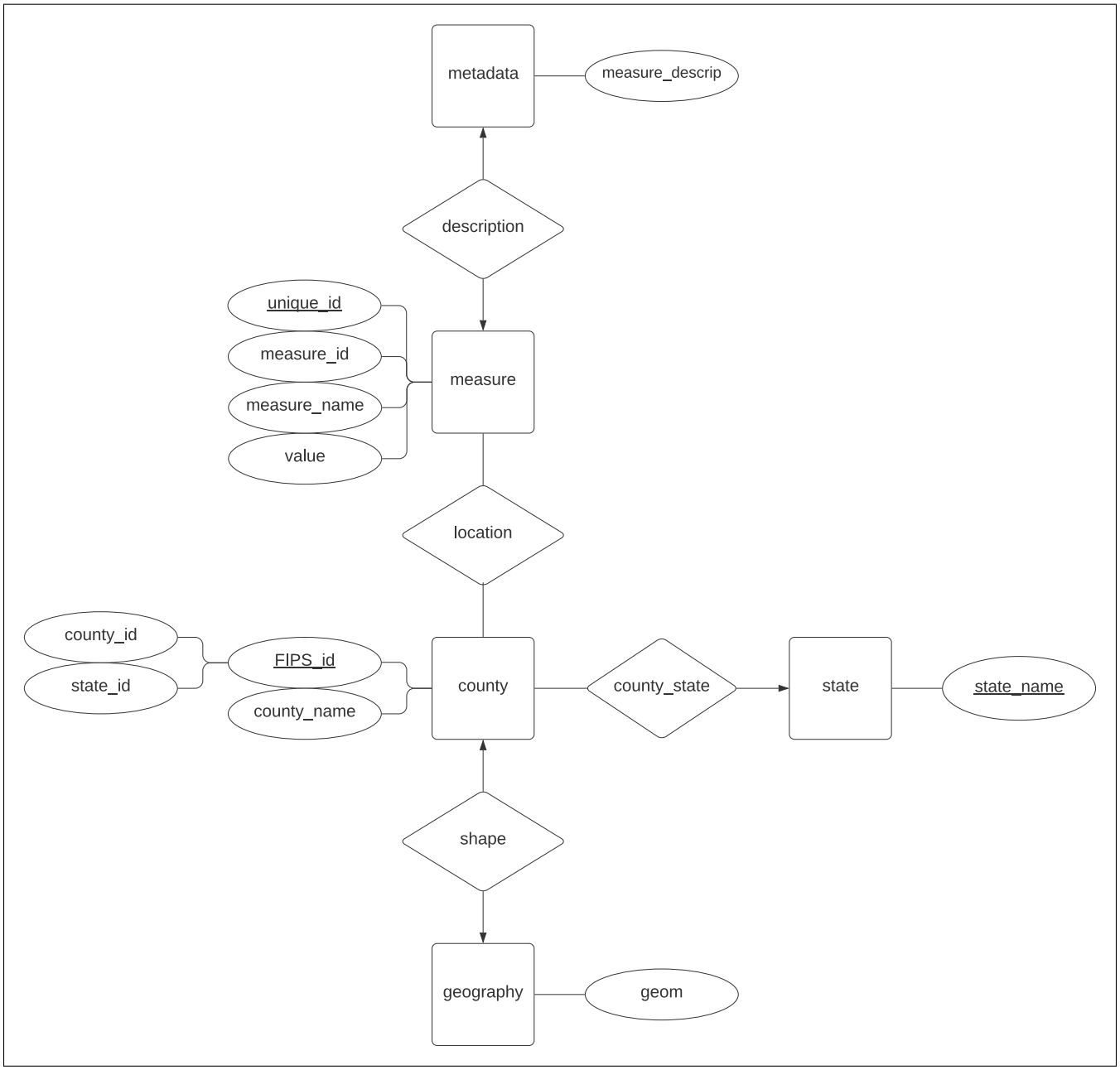


Figure 2.1: ER model for cross-sectional and spatial OEPS data.

**Longitudinal CHR dataset.** We then turn our attention to developing an ER model for the longitudinal CHR dataset. We notably do not explicitly consider a spatial component of this data at this point so the previous "geography" entity has been removed. The remainder of entities, relationships, and attributes remain the same from our OEPS ER model with the exception of an added "year_collected" attribute to our "measure" entity.

As briefly highlighted previously, inclusion of a "year_collected" attribute to our "measure" entity effectively allows us to store longitudinal data in our model in the "long" tabular data

format. Each unique pair of "measure_ID" (or equivalently "measure_ID") and "year_collected" can effectively define each unique row (or "tuple") of a data table in "long" format. As we had previously alluded to the importance of our "unique_ID" attribute in the previous ER model, this attribute effectively serves as a unique identifier for each point in measure-time space. We will see the added importance of this attribute when combining OEPS and CHR data in our combined ER model further below.

The resulting ER diagram that corresponds with our specification of the CHR data can be seen in Figure 2.2 below (on page 14).

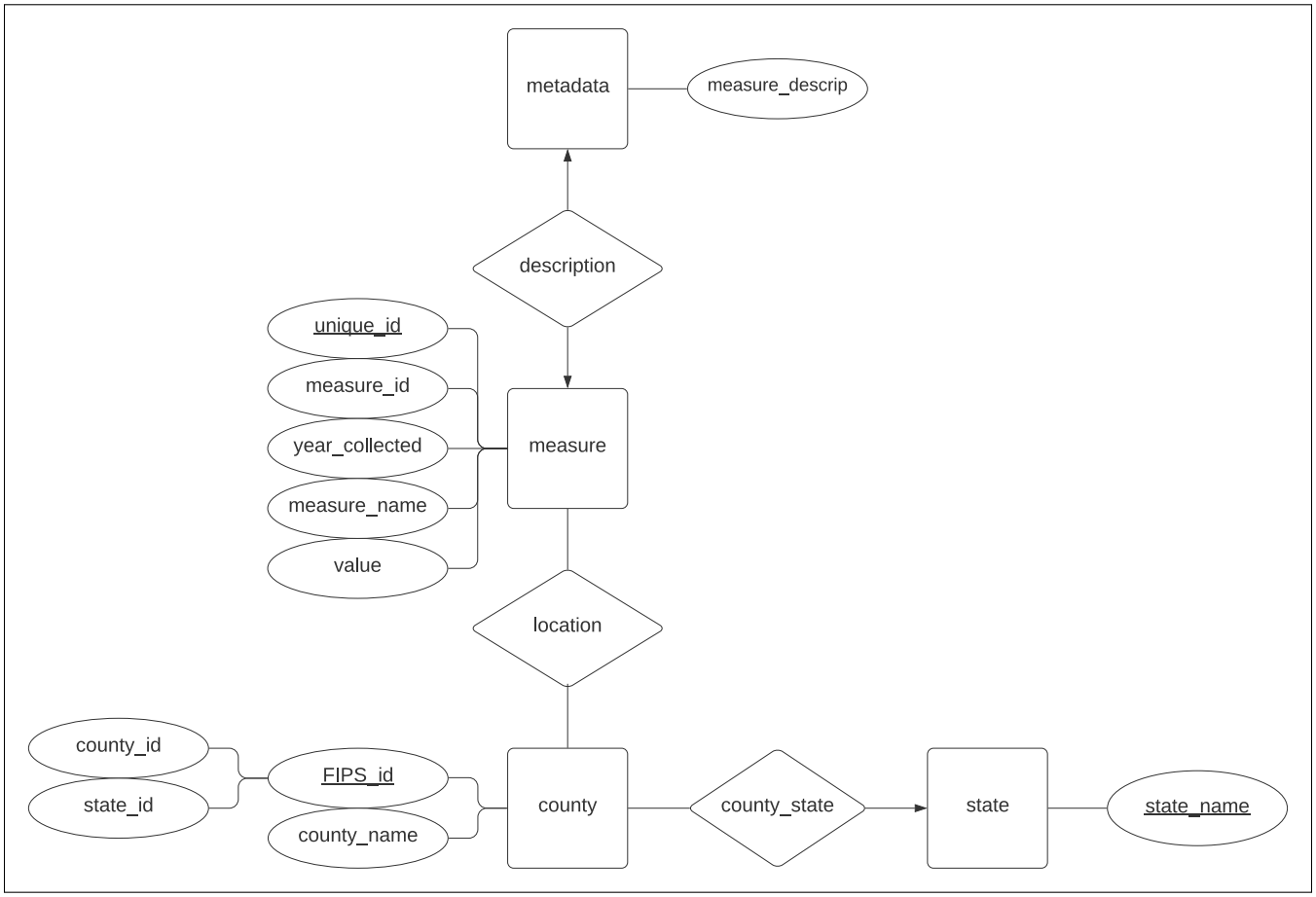


Figure 2.2: ER model for longitudinal CHR data.

**Combined OEPS and CHR dataset.** Finally, we turn to the problem of designing an ER model for a consolidated dataset combining our cross-sectional and spatial OEPS dataset with our longitudinal CHR dataset.

The resultant ER model of our combined dataset contains the union of all entities, relationships, and attributes contained in both our OEPS ER model and our CHR ER model. The visualized ER model for our combined dataset can be seen below in Figure 2.3 below (on page 15).

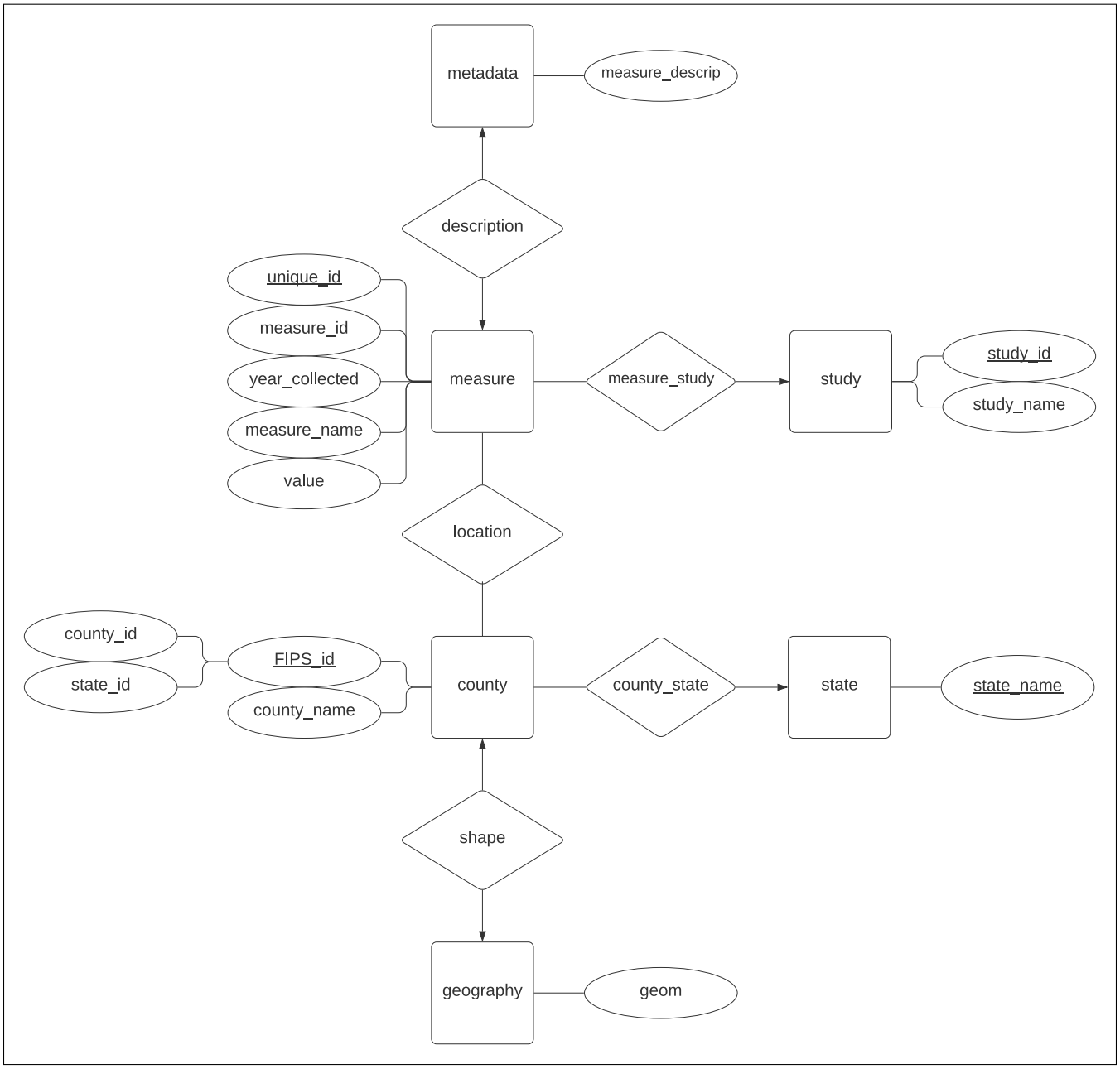


Figure 2.3: ER model for combined data.

As seen in the ER diagram, a new entity has been introduced for our combined dataset that was not present previously: "study". This entity serves to provide human-readable identifiers, in the form of "study_ID" and "study_name" attributes of the "study" entity, to distinguish between measures originally taken from the OEPS dataset versus those originally taken from from the CHR dataset. Additionally, this entity allows for data from additional studies (currently unspecified for this project) to be added-in later.

Interestingly enough, this "study" entity is to some extent redundant, as the "unique_ID" at-

tribute of the "measure" entity introduced above already does contain this information. Cross-sectional data from the OEPS dataset are to be introduced into the current ER model with a value of `NULL` for the "year_collected" attribute of the "measure" entity. This will allow for the "unique_ID" attribute of the "measure" entity to generate distinct "unique_ID" values for any measures that may be shared between the OEPS and CHR (or potentially future to-be-included) datasets! (Remember that "unique_ID" was previously described as a unique identifier for every unique pairing of "measure_name," or effectively "measure_ID," and "year_collected.")

As our ER model for our combined dataset seems to afford us enough flexibility to adequately manage our current data, and additionally has considerations for potential future data, we can feel comfortable with moving forward to reduce our ER model into relational schema and continue to design our database.

### 2.4.2 Reduction of ER models to relational schema

As one step closer from ER data model to implemented software database, relational schema serve to describe how data can be stored in the database as "relations" (also commonly referred to as tables). Relational schema share some overlap in components with ER diagrams, as previously-specified relationship cardinalities (i.e., "one-to-one," "one-to-many," or vice versa, and many-to-many") still seem to find their way into relational schema diagrams. ER model attributes, each of which are restricted to belonging exclusively to one entity in standard ER model convention, are sometimes shown in relational schema as "foreign-keys" that serve to link relations that sharing one or more common attributes.

Entities and relationships from our ER model are transformed into relations. Our previous "measure," "county," "state," "metadata," and "study" entities all become namesake tables. (For clarity's sake, we have renamed our "geography" entity from previously to "county_geography" to represent its connection to the "county" relation here, as opposed to the "state" relation.) "Foreign-key" (FK) attributes, as briefly described above, seem to be the specific way that ER relationships seem to exert themselves within the data.

In our relational schema we also notice the introduction of a "primary-key" concept that exists at the software level but was not as emphasized (though did exist in the form of underlined attribute names) in the ER model. In this case, a "primary-key" (PK) is an attribute (or more generally, a combination of multiple attributes) that serves to uniquely identify each "tuple" (or row) in a relation. For an attribute(s) that is specified as a primary-key within a relation, that attribute(s) cannot have any duplicates within that relation. This describes the relational database model.

As such, the relevant attributes and their designation within relations as primary-key (PK), foreign-key (FK), or none are shown in the relational schema diagram for the combined dataset in Figure 2.4 below (on page 17).

Additionally, relational schema are often times shown with explicit data types for each attribute as well as primary-key/foreign-key designations as one step even closer to the software implementation. A relational schema diagram for the combined dataset in this style with explicitly specified data types for each attribute is shown below in Figure 2.5 (on page 17) as well.

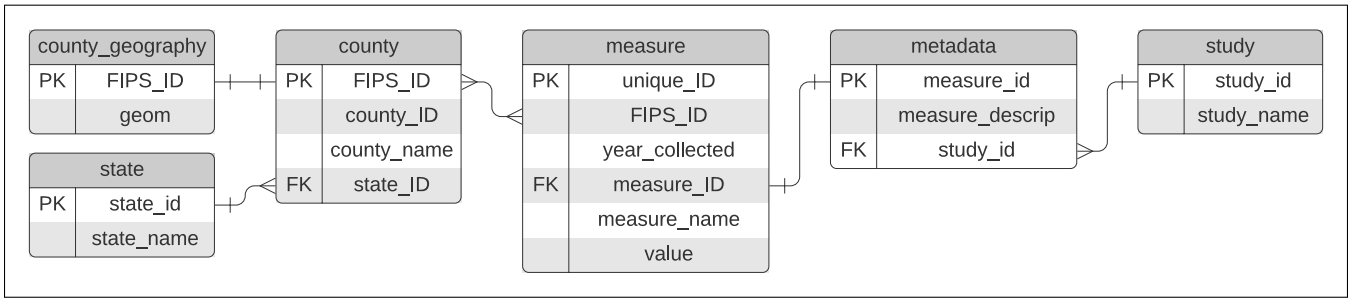


Figure 2.4: Relational schema for combined data.

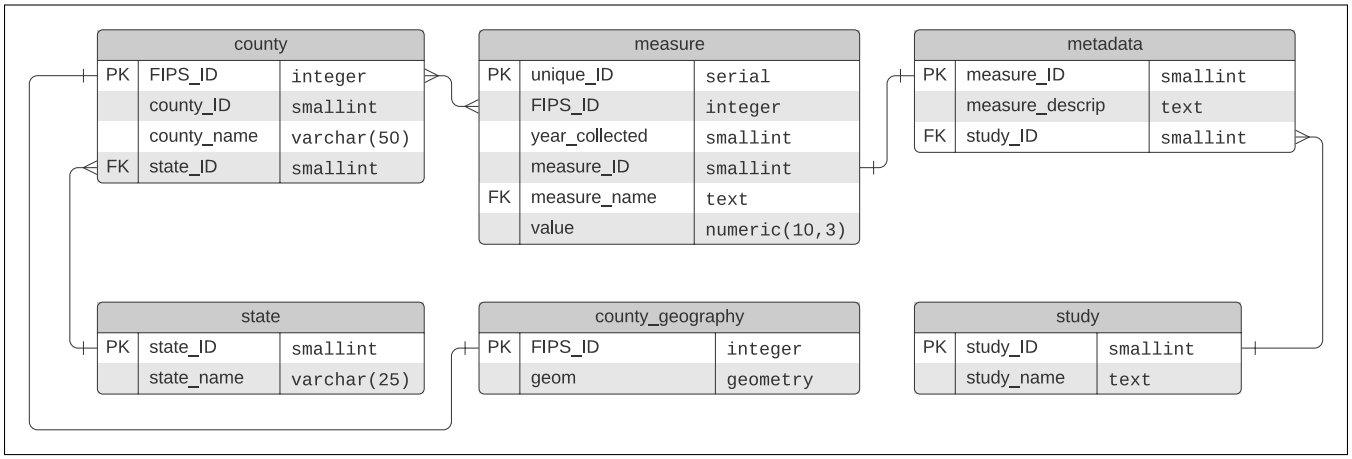


Figure 2.5: Relational schema including explicit data types for combined data.

Following the successful reduction of our ER model to relational schema, we now turn to the more concrete steps involved in creating the database from a software perspective.

## 2.5 Database creation via PostgreSQL

This project utilizes PostgreSQL version 13.4 (The PostgreSQL Global Development Group, 2021) and PostGIS version 3.1.4 (Open Geospatial Consortium, 2021). Prior to completing the below steps, these software packages were downloaded from the original websites and installed to the Windows computer used for this project. Following installation of required software, any subsequent terminal commands below were executed in Microsoft PowerShell version 7.2.0 in Windows Terminal.

### 2.5.1 Setup and preliminary steps

Following installation of PostgreSQL, the `pg_ctl` service must be started to access PostgreSQL. Figure 2.6 below (on page 18) showcases the specific commands run to start `pg_ctl` and enter PostgreSQL from the command line interface (CLI) as the `postgres` user.

```
PS project_path> pg_ctl start
waiting for server to start....done
server started

PS project_path> psql -U postgres
Password for user postgres:
psql (13.4)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=#
```

Figure 2.6: Initializing `pg_ctl` and starting PostgreSQL.

This project also specifies the storage location file path for the database using the `TABLESPACE` function in PostgreSQL. This is illustrated in Figure 2.7 (on page 18) below.

```
postgres=# CREATE TABLESPACE csi695
postgres-#   LOCATION 'D:\Codebase\CSI-695-project\combined-db_CHR-OEPS';
CREATE TABLESPACE

postgres=# CREATE DATABASE combined_db_chr_oeps
postgres-#   WITH TABLESPACE = csi695;
CREATE DATABASE
```

Figure 2.7: Specifying database storage location via `TABLESPACE`.

While the `CREATE TABLESPACE` command may have not been expressly required to achieve the goals of the project, we hoped to perform this command to ensure all data, documentation, and software were stored together in the same directory (specified by the file path above).

### 2.5.2 Data definition language (DDL)

It is now time to create our relations as tables in line with the design work we have done in creating our relational schema. As previously mentioned, PostgresQL touts excellent compliance with structured query language (SQL), the standard language used in database design and manipulation today. As such, the specific SQL used in table creation is considered "data definition language" (DDL). Figure 2.8 below (on page 19) contains the relevant SQL DDL commands executed in PostgreSQL to create empty tables with specifications given by our relational schema in Figures 2.4 and 2.5 previously.

```
postgres=# \connect combined_db_chr_oeps
You are now connected to database "combined_db_chr_oeps" as user "postgres".

combined_db_chr_oeps=# CREATE TABLE measure (
combined_db_chr_oeps(#  unique_ID smallserial PRIMARY KEY,
combined_db_chr_oeps(#  FIPS_ID integer,
combined_db_chr_oeps(#  year_collected smallint,
combined_db_chr_oeps(#  measure_ID smallint,
combined_db_chr_oeps(#  measure_name text,
combined_db_chr_oeps(#  value numeric(10,3),
combined_db_chr_oeps(# );
CREATE TABLE

combined_db_chr_oeps=# CREATE TABLE county (
combined_db_chr_oeps(#  FIPS_ID integer PRIMARY KEY,
combined_db_chr_oeps(#  county_ID smallint,
combined_db_chr_oeps(#  county_name varchar(50),
combined_db_chr_oeps(#  state_ID smallint
combined_db_chr_oeps(# );
CREATE TABLE

combined_db_chr_oeps=# CREATE TABLE state (
combined_db_chr_oeps(#   state_ID smallint PRIMARY KEY,
combined_db_chr_oeps(#   state_name varchar(25)
combined_db_chr_oeps(# );
CREATE TABLE

combined_db_chr_oeps=# CREATE TABLE metadata (
combined_db_chr_oeps(#   measure_ID serial PRIMARY KEY,
combined_db_chr_oeps(#   measure_descrip text,
combined_db_chr_oeps(#   study_ID smallint
combined_db_chr_oeps(# );
CREATE TABLE

combined_db_chr_oeps=# CREATE TABLE study (
combined_db_chr_oeps(#   study_ID smallint PRIMARY KEY,
combined_db_chr_oeps(#   study_name text
combined_db_chr_oeps(# );
CREATE TABLE
```

Figure 2.8: DDL commands in SQL to create desired relations.

Notably we have waited to define all tables above with their attributes and corresponding explicit data types prior to linking any tables to one another through foreign-keys. This next step of specifying foreign keys was done below in Figure 2.9 (on page 20).

```
combined_db_chr_oeps=# ALTER TABLE measure
combined_db_chr_oeps-#   ADD FOREIGN KEY (measure_ID)
combined_db_chr_oeps-#   REFERENCES metadata (measure_ID);
ALTER TABLE

combined_db_chr_oeps=# ALTER TABLE county
combined_db_chr_oeps-#   ADD FOREIGN KEY (state_ID)
combined_db_chr_oeps-#   REFERENCES state (state_ID);
ALTER TABLE

combined_db_chr_oeps=# ALTER TABLE metadata
combined_db_chr_oeps-#   ADD FOREIGN KEY (study_ID)
combined_db_chr_oeps-#   REFERENCES study (study_ID);
ALTER TABLE
```

Figure 2.9: DDL commands in SQL for foreign key specification.

This project has at this point not defined a table corresponding to the "county_geography" relation described earlier in Figures 2.4 and 2.5 above. This project includes hypothetical SQL commands intended to complete these steps below in Figure 2.10 (on page 20), however the actual implementation of this step was completed via the graphical user interface (GUI) rather than the CLI.

```
combined_db_chr_oeps=# CREATE TABLE county_geography (
combined_db_chr_oeps(#   FIPS_ID integer PRIMARY KEY,
combined_db_chr_oeps(#   geom geometry
combined_db_chr_oeps(# );
CREATE TABLE

combined_db_chr_oeps=# ALTER TABLE county_geography
combined_db_chr_oeps-#   ADD FOREIGN KEY (FIPS_ID)
combined_db_chr_oeps-#   REFERENCES county (FIPS_ID);
ALTER TABLE
```

Figure 2.10: Hypothetical DDL commands in SQL to generate `county_geography` relation.

Finally, in order to complete the setup of PostGIS, this project performed several `CREATE EXTENSION` SQL commands as specified in the PostGIS installation instructions (Open Geospatial Consortium, 2021); these commands are shown in Figure 2.11 (on page 21) below.

```
combined_db_chr_oeps=# CREATE EXTENSION postgis;
CREATE EXTENSION
combined_db_chr_oeps=# CREATE EXTENSION postgis_raster;
CREATE EXTENSION
combined_db_chr_oeps=# CREATE EXTENSION postgis_topology;
CREATE EXTENSION
combined_db_chr_oeps=# CREATE EXTENSION postgis_sfcgal;
CREATE EXTENSION
combined_db_chr_oeps=# CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION
combined_db_chr_oeps=# CREATE EXTENSION address_standardizer;
CREATE EXTENSION
combined_db_chr_oeps=# CREATE EXTENSION address_standardizer_data_us;
CREATE EXTENSION
combined_db_chr_oeps=# CREATE EXTENSION postgis_tiger_geocoder;
CREATE EXTENSION
```

Figure 2.11: Necessary DDL commands in SQL to complete PostGIS setup.

# 3 Database manipulation

Following design of our database via ER modeling and reduction to relational schema, as well as creation of empty database tables via SQL DDL commands, we now turn to the problem of storing our data into the database. We begin this process by performing relevant preprocessing steps.

## 3.1 Data preprocessing via R

Consolidation and loading of the extremely extensive OEPS and CHR datasets requires a significant preprocessing effort. We dedicated specific scripts for each of the separate data sources to handle the distinct directory structures, file types, and variable naming conventions utilized in each project. Relevant preprocessing scripts were created using the R language and environment for statistical computing version 4.1.2 (R Core Team, 2021) and can be found for OEPS data in Appendix A.1.1 on page 39 and for CHR data in Appendix A.1.2 on page 40, respectively. The general preprocessing methodologies for both OEPS and CHR datasets are described as follows.

OEPS data were initially separated into several spreadsheet files by original data source (the OEPS data are consolidated from several data sources themselves). The preprocessing algorithm for OEPS data involved a first pass through all data files to create separate R data frame objects for each file. This was followed by a second pass executing sequential (in alphabetical filename order) and cumulative pairwise merges of these data frames. A third (final) pass was executed on the resultant cumulative data frame object and involved dropping columns that were repeated or otherwise not needed for future analyses.

CHR data were initially separated into several spreadsheet files by year of reporting (as they were downloaded from the original data website). The preprocessing algorithm for CHR data involved a first pass through all data files to create separate R data frame objects for each file. This was followed by a second pass discovering common columns between all years by performing sequential (in increasing year order) pairwise intersections of column name lists, successively removing column names until only those columns from the earliest (2010) data that were preserved through the most recent (2021) data are remaining. A third pass was executed to drop all non-preserved columns (those not discovered in the second pass) in each of the yearly data frame objects. The fourth (final) pass was executed to perform cumulative pairwise merges of all data frames by county-level geographic columns (i.e., `FIPS`, `county`, and `state`).

Following the preprocessing phase, our OEPS and CHR datasets were each saved as individual comma-separated value (CSV) files to minimize file size and to allow for compatibility with downstream steps of the project.

## 3.2 Data manipulation language (DML)

As we had previously described PostgreSQL's compliance with SQL and typical DDL commands for generating tables from the relational schema, SQL also contains commands corresponding to a "data manipulation language" (DML) that can be used to execute database transactions to `SELECT` data, `PARTITION` data, `INSERT` data, and perform many more related tasks.

Brief excerpts of the exact DML SQL commands used to add OEPS data (Figure 3.1 on page 23) and CHR data (Figure 3.2 on page 3.2) are included below.

```
INSERT INTO measure VALUES (1001, NULL, 0, 'totPopE', 55200.0);
INSERT INTO measure VALUES (1001, NULL, 1, 'moudMinDis', 15.74711697);
INSERT INTO measure VALUES (1001, NULL, 2, 'bupMinDis', 15.74711697);
```

Figure 3.1: Excerpt of DML commands in SQL to import OEPS data into the "measure" table.

```
INSERT INTO measure VALUES (1001, 2010, 69, 'Percentage Smokers', 28.14);
INSERT INTO measure VALUES (1001, 2010, 70, 'Teen Birth Rate', 52.6);
INSERT INTO measure VALUES (1001, 2010, 71, 'Percentage Uninsured', 14.0);
```

Figure 3.2: Excerpt of DML commands in SQL to import CHR data into the "measure" table.

Even despite the extensive subsetting of the original OEPS and CHR datasets performed above in the preprocessing phase, our data are still so numerous that manually executing individual `INSERT` DML commands for each tuple would be extremely error-prone and inefficient. As such, we turn to the power of automation to accomplish the task of creating SQL scripts containing our desired DML commands for data loading.

## 3.3 Data loading via Python

In order to generate the relevant DML SQL commands, we once more dedicate individual scripts to loading the OEPS and CHR preprocessed CSV files. Relevant scripts created to generate end-result SQL DML data loading scripts were created using the Python language version 3.9.9 ("Python", 2021) and can be found for OEPS data in Appendix A.2.1 on page 42 and for CHR data in Appendix A.2.2 on page 43, respectively.

The algorithms for these two scripts were however extremely similar. Both scripts attempted to scan relevant CSV files and write-to-file preformatted SQL DML `INSERT` commands that contained the corresponding values from each row to input as a tuple. Both scripts each contained separate segments devoted to SQL DML `INSERT` commands for the previously-constructed "measure," "metadata," and "study" tables, but only the OEPS data script generated SQL DML `INSERT` commands for the "county" and "state" tables. Notably the "county_geography" table did not feature data loading in these scripts, as this was taken care of via the PostGIS GUI in the next section.

As previously alluded to, the SQL DML `INSERT` commands were written to files `oeps_loader.sql` and `chr_loader.sql` for OEPS and CHR data, respectively. However, these files were not included in this report due to their sheer size. These SQL scripts were then executed in a Windows Terminal CLI open to PostgreSQL and connected to the previously-designed database via the `\i` (or `\ir` for use with relative paths) command as demonstrated in Figure 3.3 (on page 23) below.

```
combined_db_chr_oeps=# \ir oeps_loader.sql \ir chr_loader.sql;
```

Figure 3.3: Demonstration executing data loader SQL DML scripts in PostgreSQL CLI.

## 3.4 Importing geographical data via PostGIS

As described previously, the loading of geographic Shapefile data was handled via the PostGIS GUI rather than the PostgreSQL CLI. The GUI was extremely user-friendly and simply required connection with the desired PostgreSQL database and specification of file path for the desired Shapefiles. This is shown below in Figure 3.4 (on page 24).

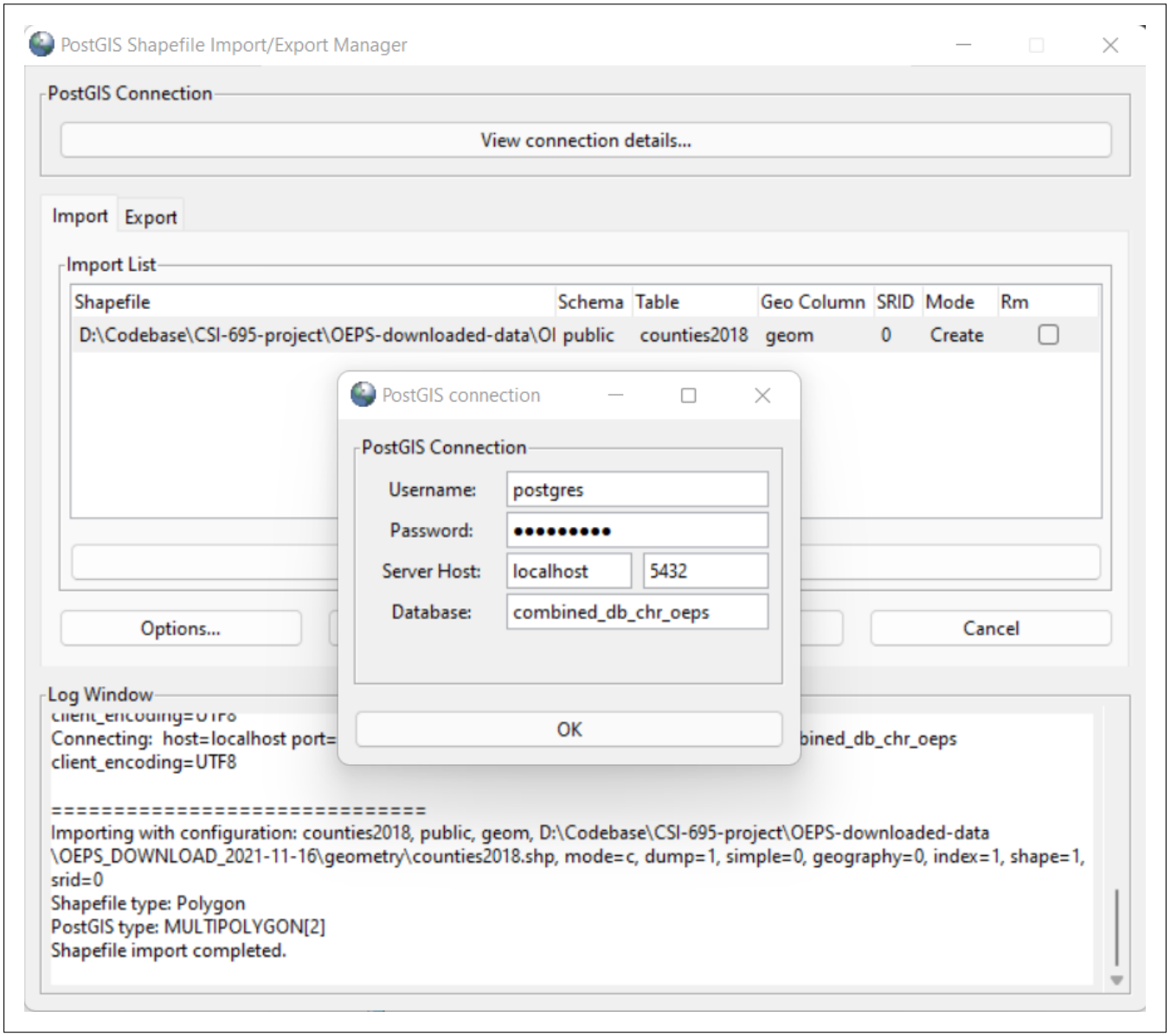


Figure 3.4: PostGIS Shapefile import/export manager screen.

Following the loading of geographic Shapefiles into the "county_geography" table, we have now fully loaded all data into our database and are ready to continue with downstream project steps.

# 4 Final combined database

Now that our data are loaded into our desired PostgreSQL (with PostGIS extension enabled), we can begin to perform several queries to inspect our work. We begin with simple queries and move towards a few more complex queries by the end of this section.

## 4.1 Simple queries

To initiate our queries, we may be interested in seeing the results of our data loading work. In order to see the number of records (in other words, number of rows or "tuples") in each table, we can use the `SELECT COUNT(*) FROM` SQL command. We utilize this command in Figure 4.1 below (on page 25). It seems that our final database contains nearly 400,000 records in the "measure" table alone!

```
combined_db_chr_oeps=# SELECT COUNT(*) FROM measure;
394790

combined_db_chr_oeps=# SELECT COUNT(*) FROM county;
2746

combined_db_chr_oeps=# SELECT COUNT(*) FROM county_geography;
3142

combined_db_chr_oeps=# SELECT COUNT(*) FROM state;
51

combined_db_chr_oeps=# SELECT COUNT(*) FROM metadata;
74

combined_db_chr_oeps=# SELECT COUNT(*) FROM study;
2
```

Figure 4.1: Queries to return row counts for each relation ("table").

Perhaps we may be interested in learning of the available data years we have in our data. We utilize the `SELECT DISTINCT` command below in Figure 4.2 (on page 26). Our count of 13 rows returned reflect the 12 years of longitudinal data we have from the CHR dataset (2010 to 2021) as well as the `NULL` value for "year_collected" we chose to assign data from the OEPS dataset.

```
combined_db_chr_oeps=# SELECT DISTINCT year_collected FROM measure;

 year_collected
----------------
           2013
           2021
           2020

           2015
           2011
           2014
           2010
           2017
           2019
           2016
           2012
           2018
(13 rows)
```

Figure 4.2: Query to return available data years.

Finally, we decide to utilize the `SELECT *` command in combination with `LIMIT 0` in Figure 4.3 (on page 26) below to list attribute names for our "measure," "county," and "state," tables (in preparation for more complex queries in the next section).

```
combined_db_chr_oeps=# SELECT * FROM measure LIMIT 0;
 fips_id | year_collected | measure_id | measure_name | value | unique_id
---------+----------------+------------+--------------+-------+-----------
(0 rows)


combined_db_chr_oeps=# SELECT * FROM county LIMIT 0;
 fips_id | county_id | county_name | state_id
---------+-----------+-------------+----------
(0 rows)


combined_db_chr_oeps=# SELECT * FROM state LIMIT 0;
 state_id | state_name
----------+------------
(0 rows)
```

Figure 4.3: Printing relation attributes in preparation for more complex queries.

## 4.2 More complex queries

At this point, we would like to ask more complex questions of our database. For example, What were all of the longitudinal (derived from the CHR dataset) "Percentage Smokers" values for Fairfax City in Virginia? We utilize the typical texttttSELECT command in combination with `INNER JOIN`

first on the "county" table and then the "state" table, followed by the `WHERE` command to specify additional filters based on our question. Notably we specify `year_collected IS NOT NULL` in order to ensure we are taking from the CHR dataset, based on our previous specification. This query is showcased below in Figure 4.4 (on page 27).

```
combined_db_chr_oeps=# SELECT year_collected, measure_name, value
combined_db_chr_oeps-#   FROM measure
combined_db_chr_oeps-#   INNER JOIN county USING(fips_id)
combined_db_chr_oeps-#   INNER JOIN state USING(state_id)
combined_db_chr_oeps-#   WHERE state_name = 'Virginia'
combined_db_chr_oeps-#     AND county_name = 'Fairfax City'
combined_db_chr_oeps-#     AND year_collected IS NOT NULL
combined_db_chr_oeps=#     AND measure_name = 'Percentage Smokers';

 year_collected |    measure_name    | value
----------------+--------------------+--------
           2010 | Percentage Smokers |  2.410
           2011 | Percentage Smokers |  3.400
           2012 | Percentage Smokers |  5.700
           2013 | Percentage Smokers |  6.000
           2016 | Percentage Smokers | 13.200
           2017 | Percentage Smokers | 10.729
           2018 | Percentage Smokers | 11.509
           2019 | Percentage Smokers | 11.509
           2020 | Percentage Smokers | 12.665
           2021 | Percentage Smokers | 11.625
(10 rows)
```

Figure 4.4: Query of yearly "Percentage Smokers" for Fairfax City.

We may be additionally be interested in the time-average of the "Percentage Smokers" variable across all timepoints available for Fairfax City, Virginia. We can answer this question with slight modifications to include `AVG(value)` in place of `value` in our original query, and to include `OVER (PARTITION BY fips_id, measure_name)` between the `SELECT` portion and `FROM` portion of our original query. The SQL `AVG()` function simply performs an average of the selection as expected, while the `OVER` window function takes the specified `PARTITION BY` clause and computes our desired quantities over the specified window. This modified query is showcased below in Figure 4.5 (on page 28). Since we chose to include the same variables in the `SELECT` portion of the command, the time-average is included at the end of the previously-seen table and repeated for each relevant entry utilized to calculate the time-average. We may have been able to avoid this problem had we utilized a `SELECT DISTINCT` statement instead, like we do in the following queries.

```
combined_db_chr_oeps=# SELECT year_collected, measure_name, AVG(value)
combined_db_chr_oeps-#   OVER (PARTITION BY fips_id, measure_name)
combined_db_chr_oeps-#   FROM measure
combined_db_chr_oeps-#   INNER JOIN county USING(fips_id)
combined_db_chr_oeps-#   INNER JOIN state USING(state_id)
combined_db_chr_oeps-#   WHERE state_name = 'Virginia'
combined_db_chr_oeps-#     AND county_name = 'Fairfax City'
combined_db_chr_oeps-#     AND year_collected IS NOT NULL
combined_db_chr_oeps-#     AND measure_name = 'Percentage Smokers';

 year_collected |    measure_name    |        avg
----------------+--------------------+--------------------
           2010 | Percentage Smokers | 8.8747000000000000
           2011 | Percentage Smokers | 8.8747000000000000
           2012 | Percentage Smokers | 8.8747000000000000
           2013 | Percentage Smokers | 8.8747000000000000
           2016 | Percentage Smokers | 8.8747000000000000
           2017 | Percentage Smokers | 8.8747000000000000
           2018 | Percentage Smokers | 8.8747000000000000
           2019 | Percentage Smokers | 8.8747000000000000
           2020 | Percentage Smokers | 8.8747000000000000
           2021 | Percentage Smokers | 8.8747000000000000
(10 rows)
```

Figure 4.5: Query of time-average "Percentage Smokers" for Fairfax City.

Perhaps we may be interested in a ranking-/sorting-based question, such as the following: which Virginia Counties (or County equivalents, e.g., Fairfax City) have the top 10 highest or top 10 lowest time-average "Percentage Smokers" values? In answering this question we would likely only want 1 row per County (or County equivalent), and thus modify our previous query to utilize `SELECT DISTINCT` command alluded to earlier, rather than the plain `SELECT` command. As a final modification, we may end our query with either `ORDER BY AVG DESC LIMIT 10` (first 10 of averages in descending order) for our "top 10" case or `ORDER BY AVG ASC LIMIT 10` (first 10 of averages in ascending order) for our "bottom 10" scenarios. These SQL queries resultant output can be seen in Figure 4.6 (on page 29) for the 10 highest time-average and in Figure 4.7 (on page 29) for the 10 lowest time-average "Percentage Smokers" Counties (and County equivalents) in the state of Virginia from 2010-2021.

(Note: Figures 4.6 and 4.7 were produced with smaller font sizes to fit together on the same page.)

```
combined_db_chr_oeps=# SELECT DISTINCT county_name, state_name, measure_name, AVG(value)
combined_db_chr_oeps-#   OVER (PARTITION BY fips_id, measure_name)
combined_db_chr_oeps-#   FROM measure
combined_db_chr_oeps-#   INNER JOIN county USING(fips_id)
combined_db_chr_oeps-#   INNER JOIN state USING(state_id)
combined_db_chr_oeps-#   WHERE state_name = 'Virginia'
combined_db_chr_oeps-#     AND year_collected IS NOT NULL
combined_db_chr_oeps-#     AND measure_name = 'Percentage Smokers'
combined_db_chr_oeps-#   ORDER BY AVG DESC LIMIT 10;

 county_name | state_name |    measure_name    |         avg
-------------+------------+--------------------+---------------------
 Wise        | Virginia   | Percentage Smokers | 27.4436666666666667
 Dickenson   | Virginia   | Percentage Smokers | 26.9171818181818182
 Carroll     | Virginia   | Percentage Smokers | 26.8813333333333333
 Buchanan    | Virginia   | Percentage Smokers | 25.9701666666666667
 Dinwiddie   | Virginia   | Percentage Smokers | 24.9875833333333333
 Pulaski     | Virginia   | Percentage Smokers | 24.8219166666666667
 Lee         | Virginia   | Percentage Smokers | 24.7314166666666667
 Henry       | Virginia   | Percentage Smokers | 24.4219166666666667
 Scott       | Virginia   | Percentage Smokers | 24.0952500000000000
 Mecklenburg | Virginia   | Percentage Smokers | 23.9395833333333333
(10 rows)
```

Figure 4.6: Query of highest 10 time-average "Percentage Smokers" counties in Virginia.

```
combined_db_chr_oeps=# SELECT DISTINCT county_name, state_name, measure_name, AVG(value)
combined_db_chr_oeps-#   OVER (PARTITION BY fips_id, measure_name)
combined_db_chr_oeps-#   FROM measure
combined_db_chr_oeps-#   INNER JOIN county USING(fips_id)
combined_db_chr_oeps-#   INNER JOIN state USING(state_id)
combined_db_chr_oeps-#   WHERE state_name = 'Virginia'
combined_db_chr_oeps-#     AND year_collected IS NOT NULL
combined_db_chr_oeps-#     AND measure_name = 'Percentage Smokers'
combined_db_chr_oeps-#   ORDER BY AVG ASC LIMIT 10;

 county_name  | state_name |    measure_name    |         avg
--------------+------------+--------------------+---------------------
 Fairfax City | Virginia   | Percentage Smokers |  8.8747000000000000
 York         | Virginia   | Percentage Smokers | 11.4976666666666667
 Fairfax      | Virginia   | Percentage Smokers | 11.5245833333333333
 Loudoun      | Virginia   | Percentage Smokers | 11.6141666666666667
 Arlington    | Virginia   | Percentage Smokers | 11.6650000000000000
 James City   | Virginia   | Percentage Smokers | 12.3024166666666667
 Fluvanna     | Virginia   | Percentage Smokers | 12.9787000000000000
 Albemarle    | Virginia   | Percentage Smokers | 13.6415000000000000
 Botetourt    | Virginia   | Percentage Smokers | 14.4337500000000000
 Rockingham   | Virginia   | Percentage Smokers | 14.6154166666666667
(10 rows)
```

Figure 4.7: Query of lowest 10 time-average "Percentage Smokers" counties in Virginia.

# 5 Linking database with interactive data dashboard

Now that we have verified our data were loaded successfully into our DBMS and asked a few preliminary questions of our data, it is time for us to select a software solution for our interactive data dashboard and connect our data source.

## 5.1 Choice of interactive data dashboard software

The original intention of this project was to utilize the RStudio Shiny project (https://shiny.rstudio.com/) team to create the interactive data due to its open source nature. However, initial efforts in learning the Shiny framework proved excessively time-consuming to the point that this project changed directions to utilize a more familiar product for this goal.

As such, this project began to explore software options developed by Tableau Software LLC, such as Tableau Public and Tableau Desktop (Tableau Software, LLC, 2021). A combination of prior experience with developing a geospatial data dashboard in Tableau Public alongside the availability of free temporary educational licenses for Tableau Desktop led to this project's selection of Tableau Desktop as the software tool for development of an interactive data dashboard.

### 5.1.1 Free educational licenses for Tableau Desktop

The developers of Tableau have a few different software offerings. Tableau Public (https://public.tableau.com/en-us/s/) is a free software with many of the same functionalities as Tableau desktop. However, the ability to link PostgreSQL and additional DBMS software solutions to a Tableau visualization is a feature restricted to the paid Tableau Desktop software.

However, Tableau seems to offer free one-year software licenses for Tableau Desktop via their "Tableau for Students" program (https://www.tableau.com/academic/students). The process to obtain an educational license is relatively straightforward, involving a simple form submission using a university.edu email address and clicking an email confirmation link. Following this process, this project utilizes Tableau Desktop version 2021.3.3 (Tableau Software, LLC, 2021) as the software solution for the interactive data dashboard.

Additionally of note, Tableau seems to offer renewable 1-year licenses for instructors and non-commercial academic researchers via their Tableau for Teaching program (https://www.tableau.com/academic/teaching).

### 5.1.2 Data linkage interface

As previously described, Tableau Desktop has an excellent selection of connections to various data sources, including PostgreSQL. The wide variety of available options for data source connections is showcased in Figure 5.1 (on page 31) below.

The process for linking Tableau with PostgreSQL was similar to that with linking PostGIS with PostgreSQL. After providing the relevant database login information, Tableau Desktop showed the appropriate tables and links on the screen reproduced in Figure 5.2 (on page 31) below.

Our PostgreSQL database is now successfully linked to Tableau Desktop.

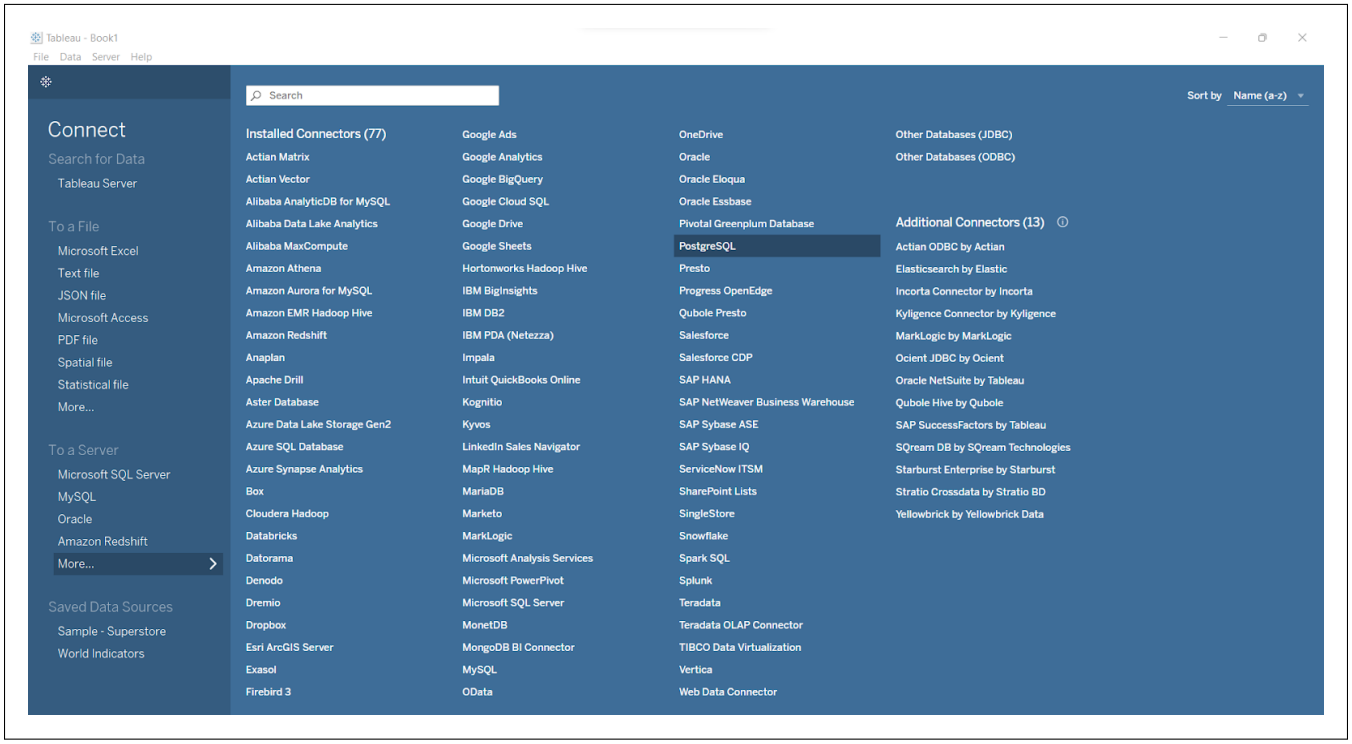


Figure 5.1: Initial DBMS linkage screen on Tableau Desktop.

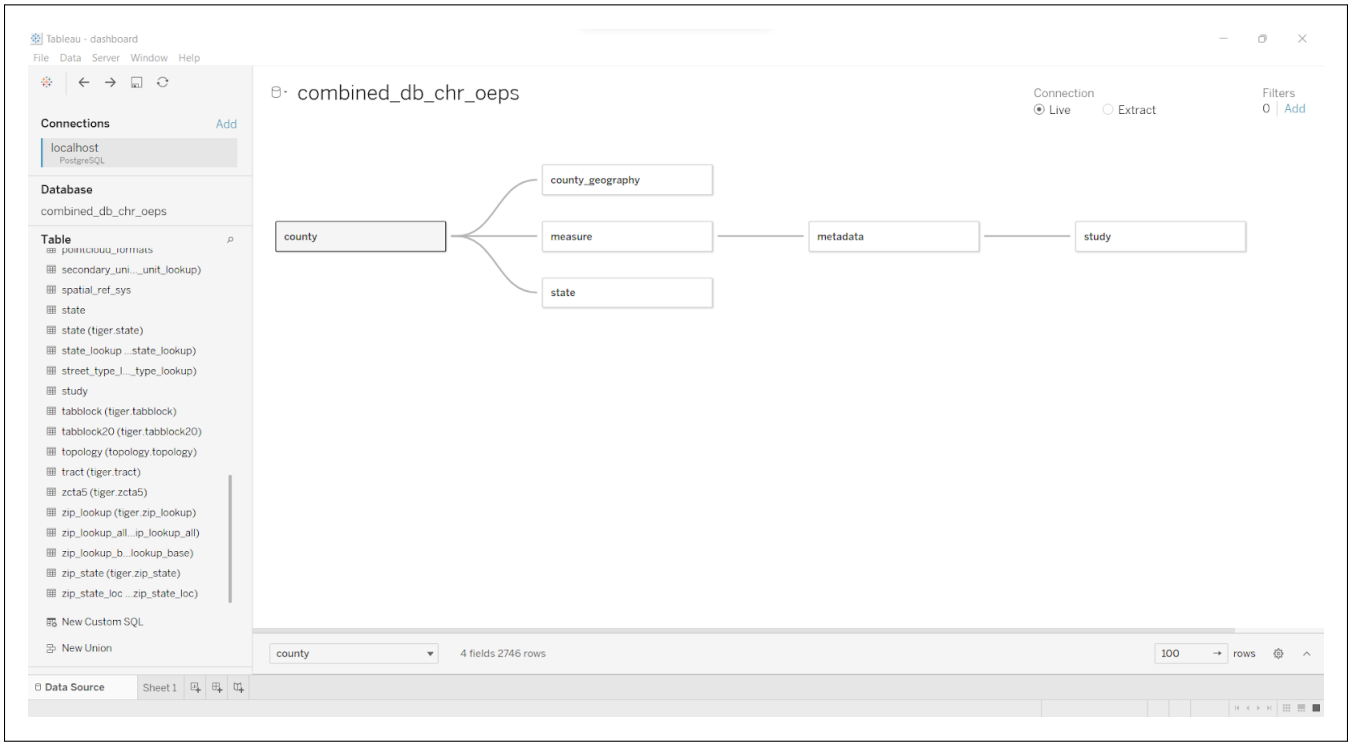


Figure 5.2: Successful DBMS linkage screen on Tableau Desktop.

## 5.2 Special considerations for Tableau Public

As mentioned previously, Tableau Public is the name of the free software tool created by the Tableau developers with restricted functionality from Tableau Desktop. However, Tableau Public is also somehow tied to free web hosting of Tableau dashboards.

As such, exporting the interactive data dashboard from Tableau Desktop onto Tableau Public (for web hosting) runs into previously-described issues regarding data source linkage. However, Tableau Desktop provides functionality to save a data extract to allow for compatibility with Tableau Public. More information on generating Tableau data extracts can be found on the Tableau website (https://help.tableau.com/current/pro/desktop/en-us/extracting_data.htm).

# 6 Interactive data dashboard

We now shift our focus from the design phase to sharing the results of our efforts. A persistent link to the web-hosted interactive data dashboard on Tableau Public is as follows: https://public.tableau.com/views/oeps-chr-combined-data-dashboard/OEPSCHRcombineddatadashboard?:language=en-US&:display_count=n&:origin=viz_share_link. Screenshots of the longitudinal and spatial components of our data dashboard can be seen below in Figures 6.1 (on page 33) and 6.2 (on page 34), respectively. We turn now to provide explanation of each component of the interactive data dashboard separately.

## 6.1 Longitudinal CHR trends data visualization

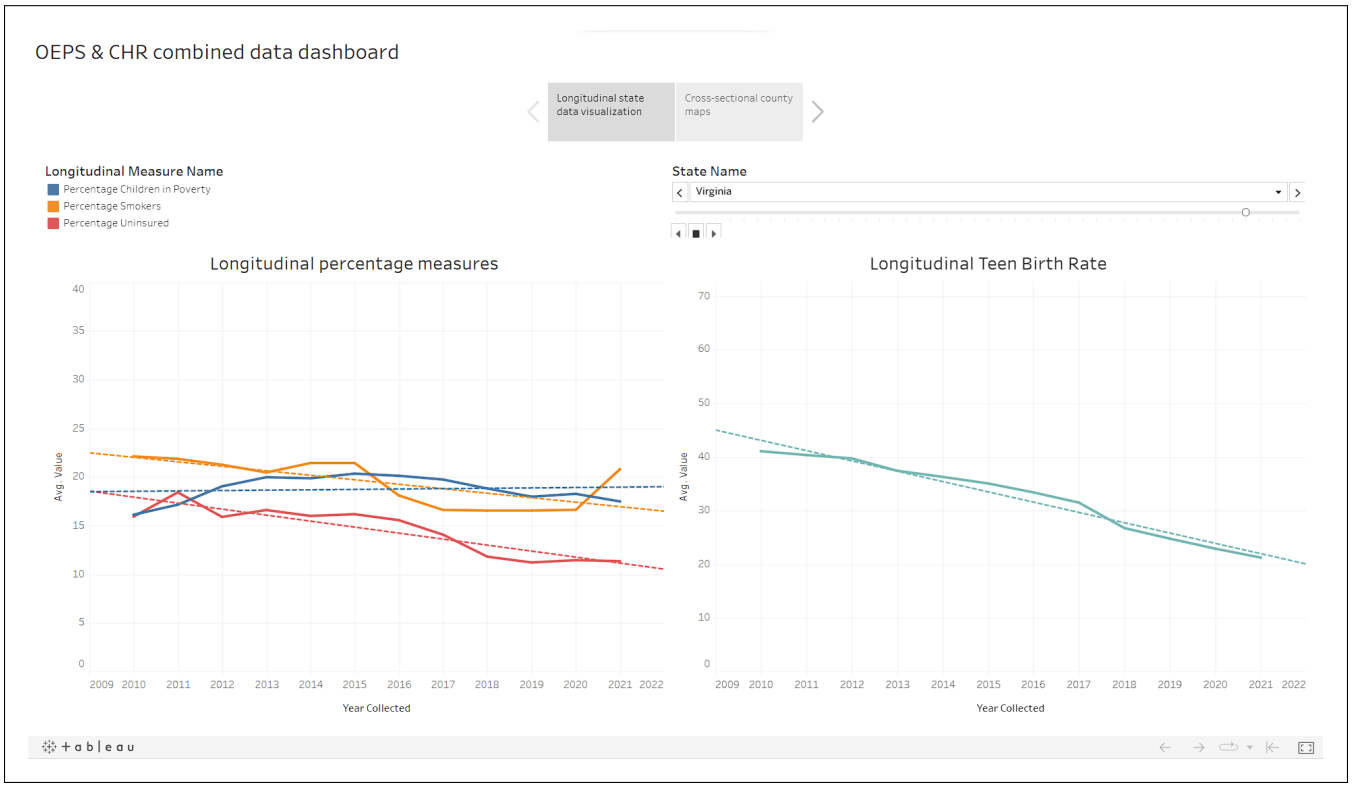


Figure 6.1: Longitudinal component of interactive data dashboard.

As seen in Figure 6.1 above, our longitudinal data are visualized in 2 columns. On the left side we have longitudinal trajectories for 3 percentage measures (percentage children in poverty, percentage smokers, and percentage uninsured) visualized throughout the CHR data collection period (from 2010 to 2021). On the right side we have the longitudinal trajectory for a non-percentage measure (in this case, teen birth rate) also visualized throughout the CHR data collection period. On both trajectory visualizations we see solid lines representing the actual data trajectories and dashed lines representing simple autogenerated lines of best fit. Near the top of the right side we see a dropdown box (along with a linked slider) that allows a user to select a state to visualize.

Although not yet implemented, we would like to implement an additional dropdown box to select

between all-County (and County equivalents) average trajectories for a given State or County-level trajectories for a given County or County equivalent. We would also like to implement an additional dropdown box to allow for the right column measure to be selected from a list of potential measures.

This visualization provides an at-a-glance view of how different longitudinal measures have changed throughout the past decade. This would hopefully serve as a quick way for policymakers or researchers to understand the longitudinal trends in various social determinants of health in a given geospatial entity to have a more nuanced understanding of environmental factors faced by a given population than is possible with a static cross-sectional snapshot.

## 6.2 Cross-Sectional OEPS spatial data visualization

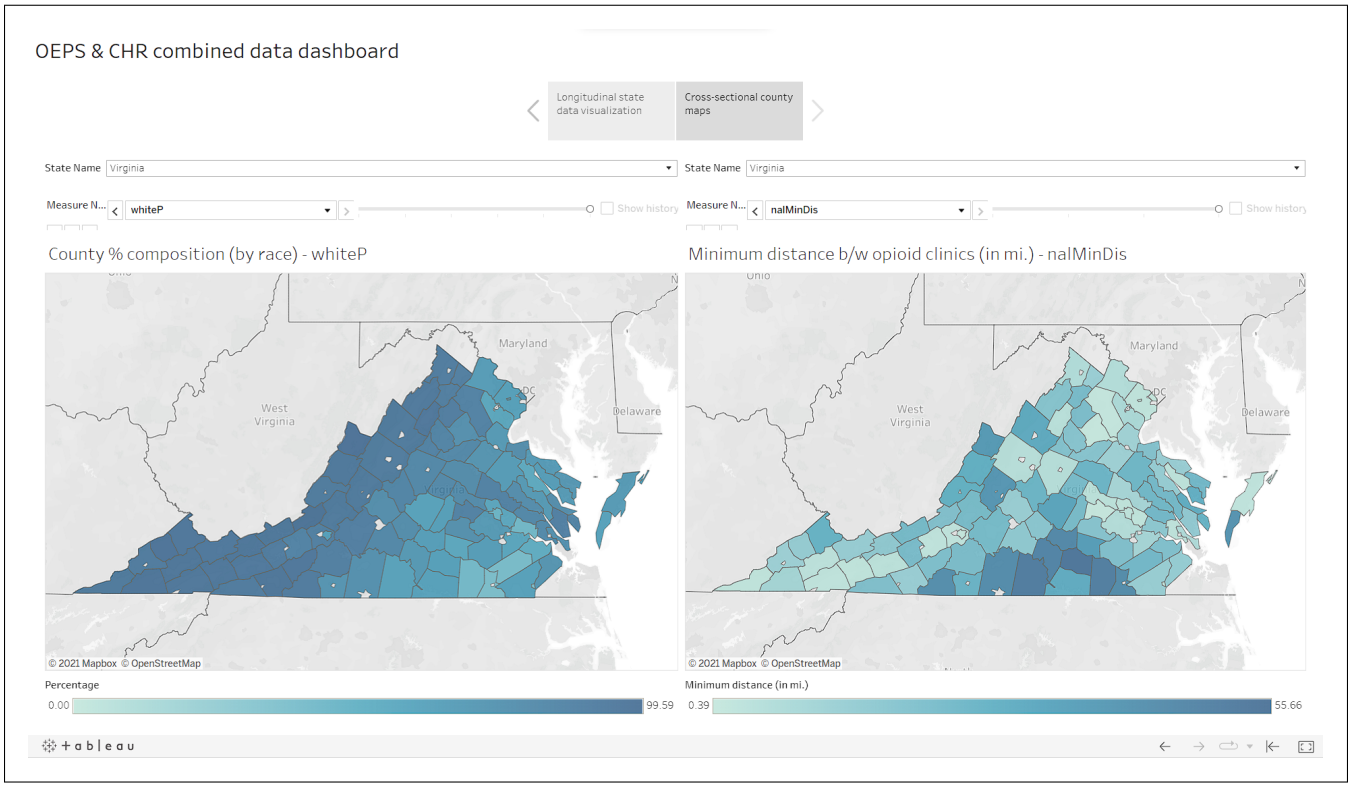


Figure 6.2: Spatial component of interactive data dashboard.

As seen in Figure 6.2 above, our cross-sectional spatial data are also visualized in 2 columns. In the left column we see a spatial visualization of the County (and County equivalent) percentage composition by race. In the right column we see a spatial visualization of the minimum distance between opioid clinics. Both columns feature dropdown boxes that allow for selection of either different race categories for percent composition (left column) or minimum distance between different types of opioid treatment clinics (right column). Both columns also have a dropdown at the top to select the State viewed, but the intention is for both maps to show the same State.

Although not yet implemented, we would like to implement a single dropdown box that controls the State selected for both columns. We would additionally like to implement novel composite

measures (such as a composite risk score created from several socioeconomic variables on the left and a composite access score created from several minimum distance variables on the right) and have those serve as the default views of this visualization as they would offer the most information to the user at once. We would ideally like to also implement the longitudinal dashboard as a "hover-over" window to pop up when the user's mouse is on a specific County or County equivalent.

This visualization provides an at-a-glance view of potential spatial correlation as well as "hotspot" sighting that can allow policymakers a quick look into which Counties (or County equivalents) in a given State are facing harsher socioeconomic or access scenarios than others. The additional context provided by a spatial dashboard may serve a policymaker in understanding more than they would otherwise by considering these factors outside of a spatial context.

# 7 Acknowledgments

This material is based upon work supported by the National Science Foundation grant IIS: 1945764 *(EAGER: An Open Data Sharing Platform for Substance Use Disorders)*, and was also supported in part by the NSF grant DGE: 1922598 *(NRT-HDR: Transdisciplinary Graduate Training Program in Data-Driven Adaptive Systems of Brain-Body Interactions)*. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

# 8 References

DeWeerdt, S. (2019). Tracing the us opioid crisis to its roots. *Nature (London)*, *573*(7773), S10–S12. https://doi.org/10.1038/d41586-019-02686-2

Haley, D. F., & Saitz, R. (2020). The Opioid Epidemic During the COVID-19 Pandemic. *JAMA*, *324*(16), 1615–1617. https://doi.org/10.1001/jama.2020.18543

Jones, M. R., Viswanath, O., Peck, J., Kaye, A. D., Gill, J. S., & Simopoulos, T. T. (2018). A brief history of the opioid epidemic and strategies for pain medicine. *Pain and Therapy*, *7*(1), 13–21. https://doi.org/10.1007/s40122-018-0097-6

Kolak, Lin, Paykin, Menghaney, & Li. (2021). *Geodacenter/opioid-policy-scan: Opioid environment policy scan data warehouse* (comp. software; Version v0.1-beta). Zenodo. https://doi.org/10.5281/zenodo.4747876

Kolak, M.A., Chen, Y.T., Joyce, S., Ellis, K., Defever, K., McLuckie, C., Friedman, S., & Pho, M.T. (2020). Rural risk environments, opioid-related overdose, and infectious diseases: A multidimensional, spatial perspective. *International Journal of Drug Policy*, *85*. https://doi.org/10.1016/j.drugpo.2020.102727

National Institute on Drug Abuse. (2021, March 11). *Opioid overdose crisis*. Retrieved November 16, 2021, from https://www.drugabuse.gov/drug-topics/opioids/opioid-overdose-crisis

NIH HEAL Initiative. (2021, November 2). *Justice community opioid innovation network*. Retrieved December 7, 2021, from https://heal.nih.gov/research/research-to-practice/jcoin

Open Geospatial Consortium. (2021, September 4). *Postgis* (comp. software; Version 3.1.4). https://postgis.net/

Provine, D. M. (2011). Race and inequality in the war on drugs. *Annual review of law and social science*, *7*(1), 41–60. https://doi.org/10.1146/annurev-lawsocsci-102510-105445

*Python* (comp. software; Version 3.9.9). (2021, November 15). https://www.python.org/

R Core Team. (2021, November 1). *R: A language and environment for statistical computing* (comp. software; Version 4.1.2). Vienna, Austria. https://www.R-project.org/

Remington, P. L., Catlin, B. B., & Gennuso, K. P. (2015). The county health rankings: Rationale and methods. *Population Health Metrics*, *13*(1). https://doi.org/10.1186/s12963-015-0044-2

Rhodes, T. (2002). The 'risk environment': A framework for understanding and reducing drug-related harm. *International Journal of Drug Policy*, *13*(2), 85–94. https://doi.org/https://doi.org/10.1016/S0955-3959(02)00007-5

RStudio Team. (2021). *Rstudio: Integrated development environment for r* (Version 2021.9.1.372). Boston, MA. http://www.rstudio.com/

Tableau Software, LLC. (2021). *Tableau desktop professional edition* (comp. software; Version 2021.3.3). https://www.tableau.com

The PostgreSQL Global Development Group. (2021, August 12). *Postgresql* (comp. software; Version 13.4). https://www.postgresql.org/

University of Wisconsin Population Health Institute, & Robert Wood Johnson Foundation. (n.d.). *County health rankings*. Retrieved November 16, 2021, from https://www.countyhealthrankings.org

U.S. Department of Health and Human Services. (2021, October 27). *About the epidemic*. Retrieved November 16, 2021, from https://www.hhs.gov/opioids/about-the-epidemic/index.html

# A Appendix of relevant code

This section serves to provide complete code used in this project. Earlier text may reference this section to retain focus on the outcomes of the work rather than on the code.

## A.1 Data preprocessing scripts in R

Preprocessing scripts were created using R version 4.1.2 (R Core Team, 2021) in RStudio version 2021.9.1.372. (RStudio Team, 2021)

### A.1.1 preprocessing_oeps.R

```
# preprocessing_oeps.R
# Faysal Shaikh
# fshaikh4@gmu.edu
# Nov. 16, 2021
#
# This program serves to preprocess OEPS data files.
# For access to OEPS data see
#
library(git2r)
library(readr)
library(dplyr)
library(stringr)

# # establish top of git repository for file path purposes
# repo <- repository('.') # if placed in same repo as listed above
# cwd <- workdir(repo)

# # specify file paths relative to top of git repository
# OEPS_data_path <- file.path(cwd, 'data_final')
# all_files <- list.files(OEPS_data_path)

# TEMPORARY: set cwd as file path of specifically-downloaded OEPS data for project
cwd <- file.path('./OEPS-downloaded-data/OEPS_DOWNLOAD_2021-11-16/data/') # 23 files, less than 92 from above
data_fnames <- list.files(cwd)

# initialize empty lists for file names and variable names to add to later
OEPS_fnames_list <- list()
OEPS_df_list <- list()

# first pass: loop through all data files and create R objects
for (fname in data_fnames) {
  # use fname to name data frame
  name_stem <- str_split(fname, '_')[[1]][1] # pull text before underscore

  OEPS_fnames_list[as.character(name_stem)] = file.path(cwd, fname) # add fnames to empty OEPS_fnames_list created above
  OEPS_df_list

  assign(paste0('OEPS_data_', name_stem), # add data frames to appropriately-named R objects
```

```
        read_csv(file.path(cwd, fname),
                 col_types = cols(COUNTYFP = col_integer(), STATEFP = col_integer())
    )  %>% rename_with(toupper, ends_with('ear')) # handle issue with Year vs year vs YEAR
  )

  OEPS_df_list[name_stem] = paste0('OEPS_data_', name_stem) # tie R object names to name_stems in empty OEPS_df_list created above
}

# second pass: loop through all R objects and execute sequential (in order) cumulative pairwise merges
OEPS_data_combined <- get(OEPS_df_list[[1]]) # start with first
# for (df_name in OEPS_df_list[2:length(OEPS_df_list)]) {
for (df_name in OEPS_df_list[2:length(OEPS_df_list)]) {
  OEPS_data_combined <- OEPS_data_combined %>% merge(get(df_name))
}

# drop columns we don't want
OEPS_data_combined <- OEPS_data_combined %>%
  subset(select = -c(YEAR, STATEFP, state, name, note, county))

# save preprocessed data as CSV file
OEPS_data_combined %>% write.csv(file.path(cwd, 'OEPS_data_combined.csv'), row.names = FALSE)
```

### A.1.2 preprocessing_chr.R

```
# preprocessing_chr.R
# Faysal Shaikh
# fshaikh4@gmu.edu
# Nov. 15, 2021
#
# This program serves to preprocess CHR data files.
# For more information see https://github.com/fshaikh4/CHR-data-repo
#
library(git2r)
library(readxl)
library(dplyr)

# establish top of git repository for file path purposes
repo <- repository('.')
cwd <- workdir(repo)

# specify file paths relative to top of git repository
CHR_national_data_path <- file.path(cwd, 'data_raw', 'national-data-excel-files')
all_national_files <- list.files(CHR_national_data_path)

# initialize empty lists for file names and variable names to add to later
CHR_fnames_list <- list()
CHR_varnames_list <- list()

```

```
# first pass: loop through all national data files and create R objects
for (national_file in all_national_files) {
  # pull year and save filenames for each year into empty CHR_fnames_list created above
  year <- substr(national_file, 1, 4) # slice year from filename
  CHR_fnames_list[year] = file.path(CHR_national_data_path, national_file) # add fnames to fname list

  # create separate data frames for each excel file
  assign(paste0('CHR_data_', year), # assign each data frame to 'CHR_data_year' variable
         read_excel(
           file.path(CHR_national_data_path, national_file),
           sheet = 'Ranked Measure Data',
           skip = 1
           ) %>% select(-matches(c('...[0-9]', 'Unreliable'))) # ignore numbered (duplicate) or "unreliable" variables
         )

  # add each variable name to empty CHR_df_list created above
  CHR_varnames_list[year] = paste0('CHR_data_', year)
}

# second pass: loop through created R objects and discover columns common between all years
keep_cols <- CHR_varnames_list[[1]] %>% get() %>% names() # start with first set of variables
for (varname in CHR_varnames_list){
  # each pass, successively remove columns until only those found in all years are left
  keep_cols <- keep_cols %>% intersect(varname %>% get() %>% names())
}

# third pass: loop through objects and only keep the common columns
for (varname in CHR_varnames_list){
  assign(varname, varname %>% get() %>% select(keep_cols) %>%
    # also add year suffix to changing variables (not FIPS, state, county)
    rename_with(~paste(., varname %>% substr(10,14)), -c(FIPS, State, County)) # add year to end
  )
}

# final pass: merge all data by FIPS, State, County
CHR_data_combined <- CHR_varnames_list[[1]] %>% get() # start with first set as combined
for (varname in CHR_varnames_list[2:length(CHR_varnames_list)]) { # since above uses 1st variable, loop range starts from 2nd variable and beyond
  CHR_data_combined <- CHR_data_combined %>% merge(varname %>% get())
}

CHR_varnames_list['combined'] = 'CHR_data_combined' # add to list after loop

# save preprocessed data as CSV file
```

```
CHR_data_combined %>% write.csv(file.path(cwd, 'data_processed', 'CHR_data_combined.csv'), row.names=FALSE)
```

## A.2 Data loading scripts in Python

Data loader scripts were created using Python version 3.9.9 ("Python", 2021).

### A.2.1 oeps_sql_loader.py

```
# oeps_sql_loader.py
# Faysal Shaikh
# fshaikh4@gmu.edu
# Nov. 17, 2021
#
# This program creates an SQL loading script for OEPS_data_combined.csv
#

from pathlib import Path
import pandas as pd

# current file location: D:/Codebase/CSI-695-project
# target data file (OEPS_combined_data.csv) is located in:
# D:\Codebase\CSI-695-project\OEPS-downloaded-data\OEPS_DOWNLOAD_2021-11-16\data
source_path = Path('.').resolve()
data_path = source_path / 'OEPS-downloaded-data' / 'OEPS_DOWNLOAD_2021-11-16' / 'data'
fp = data_path / 'OEPS_data_combined.csv'



### read data into pandas data frame
oeps_df = pd.read_csv(fp)



### create empty list to hold all SQL queries
sql_list = []



### populate relevant tables: measure, metadata, study, county, state
## INSERT INTO measure VALUES (FIPS_ID, year_collected, measure_ID, measure_name, value)
# keep list of measures
measures = oeps_df.columns[1:] # do not include "FIPS_ID" as a measure
measures_dict = {measure_ID:measure_name for measure_ID,measure_name in enumerate(measures)} # generate incremental IDs for non-FIPS measures
next_measure_ID = max(measures_dict.keys()) + 1 # needed to pickup measure_IDs later

measure_DML_list = [] # empty list for this set of DML queries

# define static variables
measure_prefix = 'INSERT INTO measure VALUES ('
```

```
year_collected = 'NULL' # for cross-sectional OEPS data

for row_index in range(oeps_df.shape[0]): # every row gets inserted
    for meas_ID in measures_dict.keys(): # every column (except FIPS) should be inserted before moving to the next row
        FIPS_ID = int(oeps_df.loc[row_index]['COUNTYFP'])
        measure_ID = int(meas_ID)
        measure_name =  measures_dict[meas_ID]
        value = oeps_df.loc[row_index][measure_name]
        measure_DML_string = measure_prefix+str(FIPS_ID)+', '+year_collected+', '+ \
                        str(measure_ID)+', \''+measure_name+'\', '+str(value)+');'
        measure_DML_list.append(measure_DML_string)


## INSERT INTO metadata VALUES (measure_ID, measure_descrip, study_ID)
metadata_DML_list = [] # empty list for this set of DML queries

# define static variables
metadata_prefix = 'INSERT INTO metadata VALUES ('
metadata_suffix = ', 0);' # study_ID hardcoded for OEPS as 0
for meas_ID in measures_dict.keys():
    measure_ID = meas_ID
    measure_descrip = '\'This is measure: '+measures_dict[measure_ID]+'. Description is pending.\''
    metadata_DML_string = metadata_prefix+str(measure_ID)+', '+measure_descrip+\
                    metadata_suffix
    metadata_DML_list.append(metadata_DML_string)


## INSERT INTO study VALUES (study_ID, study_name)
study_DML_string = 'INSERT INTO study VALUES (0, \'Opioid Environmental Policy Scan (OEPS)\');' # study_ID hardcoded for OEPS as 0


### write queries to SQL script file
# order statements to avoid issues with foreign-key
sql_list.extend([study_DML_string]) # add study_DML_string to sql_list
sql_list.extend(metadata_DML_list) # add study_DML_string to sql_list
sql_list.extend(measure_DML_list) # add measure_DML_list to sql_list

# write to file
with open(source_path / 'oeps_loader.sql', 'w') as f:
    for DML_query in sql_list:
        f.write(DML_query+'\n')
```

### A.2.2 chr_sql_loader.py

```
# chr_sql_loader.py
# Faysal Shaikh
# fshaikh4@gmu.edu
# Nov. 17, 2021
```

```
#
# This program creates an SQL loading script for CHR_data_combined.csv
#

from pathlib import Path
import pandas as pd

# current file location: D:/Codebase/CSI-695-project
# target data file (CHR_combined_data.csv) is located in:
# D:\Codebase\CSI-695-project\CHR-data\data_processed
source_path = Path('.').resolve()
data_path = source_path / 'CHR-data' / 'data_processed'
fp = data_path / 'CHR_data_combined.csv'



### read data into pandas data frame
chr_df = pd.read_csv(fp)



### create empty list to hold all SQL queries
sql_list = []



### populate relevant tables: measure, metadata, study, county, state
## INSERT INTO measure VALUES (FIPS_ID, year_collected, measure_ID, measure_name, value)
# keep list of measures
measures = chr_df.columns[3:] # do not include "FIPS_ID", "state_name", "county_name" as  relevant measures
measures = pd.Index([measure_name.replace('%', 'Percentage') for measure_name in measures]) # replace '%' symbol with 'Percentage' word

chr_df.columns = list(chr_df.columns[:3]) + list(measures) # replace columns according to above name changes

unique_measures = pd.Index(pd.unique([x[:-5] for x in measures])) # take unique measures after removing year (last 5 string characters)

previous_next_measure_ID = 69 # hardcoded (from OEPS data loading)
measures_dict = {measure_name : measure_ID+previous_next_measure_ID for measure_ID,measure_name in enumerate(unique_measures)} # generate incremental IDs for non-FIPS measures [NOTE: THIS IS BACKWARDS FROM OEPS VERSION]
next_measure_ID = max(measures_dict.values()) + 1 # needed to pickup measure_IDs later

measure_DML_list = [] # empty list for this set of DML queries

# define static variables
measure_prefix = 'INSERT INTO measure VALUES ('

for row_index in range(chr_df.shape[0]): # every row gets inserted
```

```
    for measure in measures: # every column (except first 3 non-measure cols) should be inserted before moving to the next row
        FIPS_ID = int(chr_df.loc[row_index]['FIPS'])
        year_collected = int(measure[-4:]) # extracted from measure name for longitudinal data
        measure_name =  measure[:-5]
        measure_ID = int(measures_dict[measure_name])
        value = chr_df.loc[row_index][measure]
        measure_DML_string = measure_prefix+str(FIPS_ID)+', '+str(year_collected)+', '+ \
                                str(measure_ID)+', \''+measure_name+'\', '+str(value)+');'
        measure_DML_list.append(measure_DML_string)


## INSERT INTO metadata VALUES (measure_ID, measure_descrip, study_ID)
metadata_DML_list = [] # empty list for this set of DML queries

# define static variables
metadata_prefix = 'INSERT INTO metadata VALUES ('
metadata_suffix = ', 1);' # study_ID hardcoded for CHR as 0
for measure_name in unique_measures:
    measure_ID = measures_dict[measure_name]
    measure_descrip = '\'This is measure: '+measure_name+'. Description is pending.\''
    metadata_DML_string = metadata_prefix+str(measure_ID)+', '+measure_descrip+\
                            metadata_suffix
    metadata_DML_list.append(metadata_DML_string)


## INSERT INTO study VALUES (study_ID, study_name)
study_DML_string = 'INSERT INTO study VALUES (1, \'County Health Rankings & Roadmaps (CHR)\');' # study_ID hardcoded for OEPS as 0


## prepare for county and state DML queries
geo_df = chr_df[['FIPS','County','State']]
# pull state and county IDs from full FIPS code
geo_df['state_ID'] = pd.Index([int(str(x)[:-3]) for x in geo_df['FIPS']])
geo_df['county_ID'] = pd.Index([int(str(x)[-3:]) for x in geo_df['FIPS']])


## INSERT INTO county VALUES (FIPS_ID, county_ID, county_name, state_ID)
county_df = geo_df[['FIPS', 'county_ID', 'County', 'state_ID']]

county_DML_list = []
county_prefix = 'INSERT INTO county VALUES ('

for row_index in range(county_df.shape[0]): # for all rows
    FIPS_ID = county_df.loc[row_index]['FIPS']
    county_ID = county_df.loc[row_index]['county_ID']
    county_name = county_df.loc[row_index]['County']
    state_ID = county_df.loc[row_index]['state_ID']
```

```
    county_DML_string = county_prefix+str(FIPS_ID)+', '+str(county_ID)+', '+ \
                        '\''+county_name+'\', '+str(state_ID)+');'
    county_DML_list.append(county_DML_string)

## INSERT INTO state VALUES (state_ID, state_name)
state_df = geo_df[['state_ID', 'State']].drop_duplicates() # remove extra rows

state_DML_list = []
state_prefix = 'INSERT INTO state VALUES ('

for row_index in range(state_df.shape[0]): # for all rows
    state_ID = state_df.iloc[row_index]['state_ID']
    state_name = state_df.iloc[row_index]['State']
    state_DML_string = state_prefix+str(state_ID)+', \''+state_name+'\');'
    state_DML_list.append(state_DML_string)


### write queries to SQL script file
# order statements to avoid foreign-key issues
sql_list.extend(state_DML_list)
sql_list.extend(county_DML_list)
sql_list.extend([study_DML_string])
sql_list.extend(metadata_DML_list)
sql_list.extend(measure_DML_list)

# write to file
with open(source_path / 'chr_loader.sql', 'w') as f:
    for DML_query in sql_list:
        f.write(DML_query+'\n')
```