CS4624:

Multimedia, Hypertext, and Information Access

Final Report

Chorobesity:

Modern Insight Into An Enduring Epidemic

Sarah Burnett, Van Ha Tran Nguyen, Bharathi Ganesan, Bradley Freedman, Roshan Ravindran

Instructor: Mohamed Farag

Clients: Professor Lynn Abbott and Chenyu Mao

November 30, 2023

Virginia Tech, Blacksburg, Virginia 24061

Table of Contents

Table of Figures

1. Abstract

Currently, research is needed into two prevalent health conditions that exist in the United States: obesity and diabetes. Health researchers are looking for all possible relationships between the two conditions, which includes on a geographical level. In order to investigate the issue robustly, create detailed experimentation, and develop lasting solutions, the Chorobesity project presents a visual tool of the geographical relationship between obesity and diabetes for our clients to utilize in their studies. Making use of different geographical levels of maps, as well as different color "keys", the user is able to study different regions' health condition statuses.

Chorobesity, a portmanteau of obesity and choropleth, creates a play on words based on two essential components of our project. Obesity is one of the health conditions visualized within the project, and choropleth is the type of map that we use to represent our health data. A choropleth map uses different shadings or colorings to represent different averages within a certain boundary. In other words, our project shades different regions of the United States depending on their overall obesity and diabetes rates. The user can interact with this map through a mouse hover, which displays additional information, as well as view different maps and color keys.

In summary, the Chorobesity project aims to be a visual and dynamic tool that researchers can use to further their understanding of the geographical correlation between obesity and diabetes. It provides relevant data and tools with which the user can easily interpret and tweak this data for their best understanding. This interactive map, in providing a snapshot of the current health profile of the United States, seeks to be an indispensable tool for policymakers, health professionals, and the general public to understand how obesity and diabetes correlate as the clients see fit.

2. Introduction

2.1. Problem

According to the CDC, around 41.9% of the American population is obese (2017–March 2020)[4]. Obesity, according to Mayo Clinic, is described as "a medical problem that increases the risk of many other diseases and health problems…includ[ing] heart disease, diabetes, high blood pressure, high cholesterol, liver disease, sleep apnea, and certain cancers"[7]. The prevalence of obesity among the overall US population, along with its links to other serious health conditions, means that the need to study and collect data about these correlations is extremely important. The collection of data helps researchers not only continue to support their specialized expertise in the problem, but can also lead to more effective treatment options.

Our clients, Dr. Abbot and Chenyu Mao, approached our team with the following problem: Is there a correlation between diabetes and obesity diagnoses that can be seen on a geographical level? Furthermore, can this geographical correlation be visualized? From us, they are looking for an interactive map that can display the correlation that they are looking for. Our team's main problem was constructing a visualization tool that could display health data to researchers in an effective way. While we were responsible for the "behind the scenes" of this project, the clients plan to utilize the frontend display of this tool to continue their own research. Therefore, besides building this tool, we planned to make its client-end easy to navigate and interact with, clearly visualize their correlation, and make any data easy to access.

Our tool, as per the specifications of our clients, was looking to display obesity and diabetes information for Virginia, but as we continued to develop our project, we were able to display this type of information for all states in the United States and on the national level. Our map displays this information for each city/county on the single-state level and utilizes color shading to show

how the health conditions are either more or less prevalent in certain regions. The map also displays either the health conditions on their own or together, depending on the user's preference. This is done in order to see possible correlations between the data, as well as possible ways in which the data differs across the map.

This tool allows parsing through a given CSV file, automatically converting it into an internal JSON format used to represent the states/counties and displaying the data effectively to the user. The clients relied on our team to search for, present, and utilize viable datasets within the tool we are creating, incorporating it into our application to display real-life correlations.

2.2. Motivation

As touched on above, the motivation behind the creation of such a tool is to ultimately aid in the furtherance of health research in the United States. Not only do visualization tools show where there are health correlations to researchers, but they can also smoothly and effectively present findings to people not in the healthcare field or other specialized health professions. Perhaps these findings can contribute to the appropriate treatment funding of geographical areas in which obesity and diabetes are prevalent or illustrate to a lawmaker how certain initiatives should be passed to lower the rates of these health conditions.

On a smaller scale, we are motivated to create a visualization tool that conforms to the specifications that our clients are seeking for their research. First and foremost, the tool we developed will be used by our client, and creating something that is effectively usable was our top priority. In order to be usable, we needed this tool to display the data effectively, and the datasets we pulled from were clear and reputable for the clients.

2.3. Technical Approach

To construct a visualization tool that will depict rates of obesity and diabetes at the county, state, and national level for all 50 states, we will employ a technological stack comprising an Angular front end, as well as a Django REST framework for the backend. The initial step involves defining a Django data schema to efficiently store information on all US counties, as well as state-level data and national-level data, alongside their respective rates of obesity and diabetes. We used the "2023 County Health Rankings National Data" file from "County Health Rankings & Roadmaps", which was compiled by the University of Wisconsin Population Health Institute. This source documents a wide range of data for each state and county in the United States and compiles them into a large Excel document[1]. The application will extract the relevant data from the original CSV file and will then convert it into an internal JSON format using a Python script. From this extracted information, calculations will be performed in the backend to normalize the data and determine each of the frontend color key ranges. All of this extracted and calculated information will be imported into Django and used by Angular to store and display the data accurately.

A backend API will be established using the Django REST framework, enabling seamless communication between the front end and the Django database. Within the Angular framework, distinct components will be constructed, including a data table, filters, and interactive elements, to facilitate comprehensive data visualization. Integration with the backend API will be facilitated through HTTP requests, enabling dynamic retrieval of country-specific data. Data visualization will be achieved by robustly implementing Angular and taking advantage of Cartography Vectors found online, allowing for the creation of interactive, visually appealing, and detailed maps[3]. To enhance user experience, features enabling data filtration by various

parameters, such as map display level, color key type, and data normalization width factor, will be integrated, ensuring a dynamic dataset exploration.

Performance optimization measures, including lazy loading, segmentation, and caching, will be implemented to ensure optimal functionality, particularly when dealing with extensive datasets. If required, user authentication mechanisms can be introduced in later semesters to regulate access to the visualization tool. Rigorous testing, encompassing both unit and integration testing, will be conducted to validate the functionality and integrity of individual components and the application as a whole.

In the future, a goal of this web application would be to ingest various datasets containing different types of information about diseases other than obesity and diabetes. With more work from future students, this application should seamlessly adapt and provide an intuitive interface for exploring this diverse data, enabling users to uncover emerging trends with ease. Furthermore, comprehensive documentation will be provided to guide users in navigating and utilizing the visualization tool effectively.

2.4. Team Roles

Below are the roles that each team member has assumed in order to complete our project. While each member has their own specific roles, all persons plan to work together and work across both the frontend and backend of the project as needed.

- Bharathi Ganesan, Leader, Front End Design
- Sarah Burnett, Note Taker, Presentation Lead, BackEnd
- Van Ha Tran Nguyen, Frontend Design, Database Input
- Bradley Freedman, Communicator, BackEnd

- Roshan Ravindran, Research Development, Requirements Lead
- All: Prototyping and General Implementation

3. Requirements

The requirements for this project can be split into two parts: the requirements of the client and the requirements for the successful implementation of the project. In the sections below, we will discuss what the client finds to be a “minimum viable product,” as well as what is necessary to create this dynamic and useful application successfully. Our requirements for the project include the consistent formatting of data, robust analysis to create an accurate visualization, and clear and concise documentation that the client can reference when interacting with the application.

3.1. Client Requirements

In the team’s first meeting with the clients, Professor Abbot and Chenyu Mao discussed the characteristics and the purpose of the application that they were looking for. There are four core requirements in the application: a color-coded map comparing obesity and diabetes, a mouse hover that displays more information to the user, real-life and reputable datasets that can be utilized in the application, and documentation on how to use the application.The beginning of the discussion starts with the requirements of the visualization of the map from the clients’ inspired images. Images 1 and 2 show two different choropleth maps that Professor Abbott was inspired by: a “Diabetes and Obesity Map” from the CDC and “2020 Election Results” from the New York Times. The two maps show color comparison between two different data points.

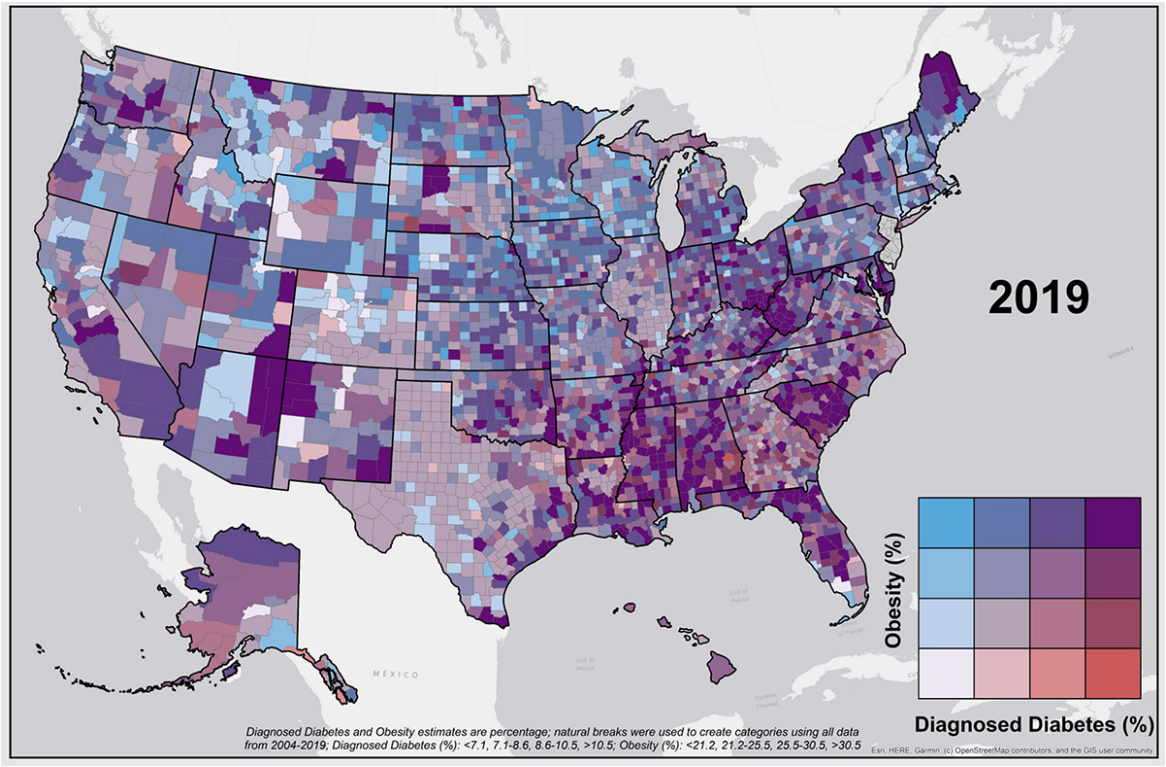

Image 1: CDC “Diabetes and Obesity Map”[6]

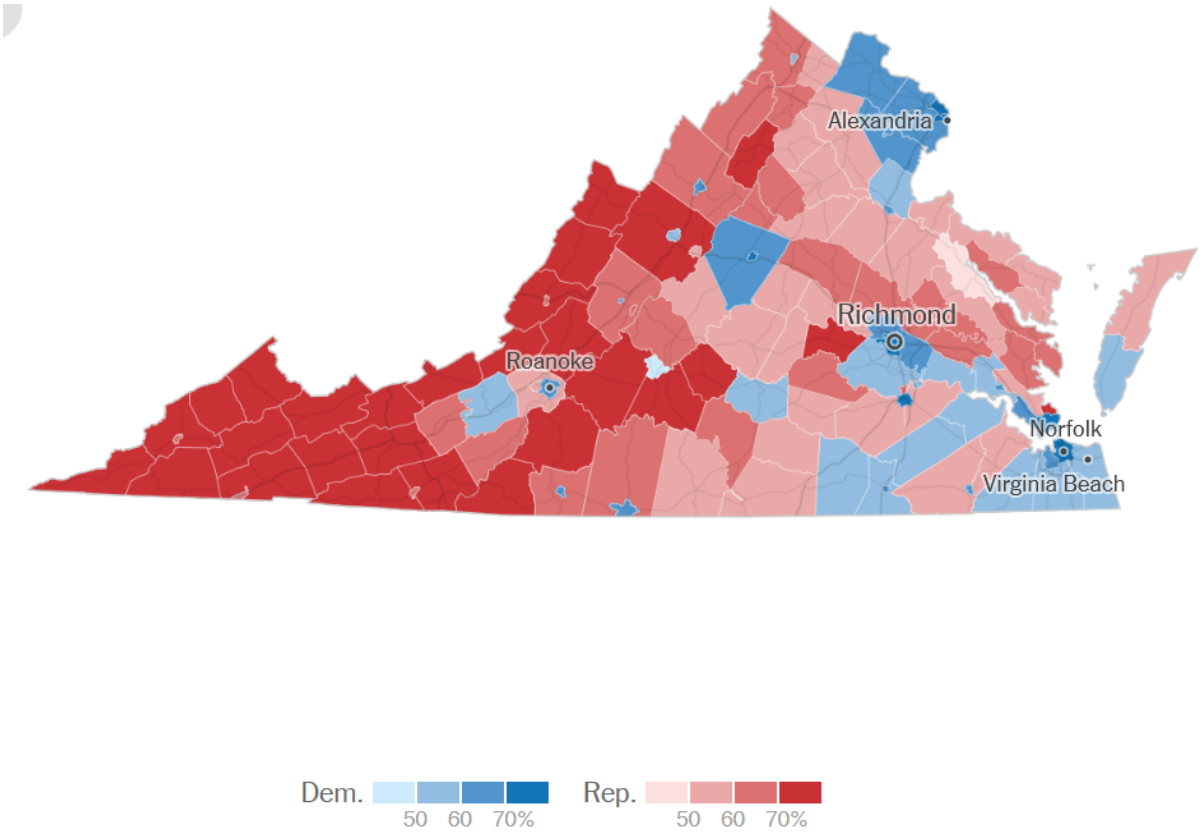

Image 2: The New York Times “2020 Virginia Election Results”[5]

The clients also detailed how real-life datasets for the project would need to be researched. Our group found the “2023 County Health Rankings” dataset on “County Health Rankings & Roadmaps” to be the most sturdy and contain all of the necessary information needed for the development of this application[1]. The following columns are used directly from this source: County (or State) Name, Obesity Percentage, Diabetes Percentage, and Population.

The clients wanted to see our group incorporate their four requirements first on a state level, starting with Virginia and listing its counties. In other words, implementing all four requirements for Virginia and its counties was considered the “minimum viable product”. In order for the

group to reach this stage and eventually surpass it, we needed to formulate our own requirements for completing the application, while staying aware of the clients' ultimate goals.

3.2. Formatting Data

The efficacy of the visualization tool is predicated on the systematic procurement and structuration of health-related data. Our clients entrust us to identify, extract, and refine datasets that will be instrumental in populating the visualization framework. The "County Health Rankings & Roadmaps" are the foundational data repository from which we have distilled obesity and diabetes data relevant to our investigative parameters. We have meticulously assembled a dataset that will underpin the visualization tool, which is adeptly constructed to assimilate and display data from files formatted to specified standards.

3.3. Website/Analysis

The visualization tool is the centerpiece of this project, encapsulating the principal interest of our clientele. It boasts a robust and intuitive interface, exhibiting a geographically precise map with functionalities that include zoom and detailed demarcation of Virginia's county boundaries. The intent is to extend these capabilities to portray each state nationwide. The integration of obesity and diabetes data necessitates a highly interactive interface. Thus, implementing hoverable features on the map unveils health data correlated to each county. The map's coloration will visually represent the prevalence of the health conditions based on the dataset in focus, facilitating the addition of further data by users as required.

The tool will also facilitate the integration and interpretation of obesity and diabetes data sourced from CSV or JSON files. Our mandate is to guarantee the precision and fidelity of these datasets, along with their seamless incorporation into the tool for straightforward access and analysis by the research community. This will be achieved through the deployment of a technological suite

that includes Angular for frontend architecture, Django REST framework for backend operations, and the OpenStreetMap API for map visualizations. This holistic approach encompasses the careful curation of data, the establishment of a resilient backend API, the development of dynamic frontend elements, and the enhancement of performance to efficiently manage large datasets.

3.4. Documentation

The pressing issue of obesity in the United States, with prevalence rates approaching 41.9%, and its correlation with various health risks, as articulated by the CDC, underscores the significance of this tool [4]. It is designed not merely as a research aid but also as a means to identify effective treatment strategies by visualizing geographic patterns in obesity and diabetes diagnoses.

In adherence to client specifications, we are committed to producing detailed documentation that elucidates every aspect of the visualization software's setup, deployment, and operation. This guide aims to empower our clients with the knowledge to confidently navigate the system, thus enhancing the utility of the tool in their research pursuits. The manual will cover the full gamut of operational facets, ensuring ease of use and a seamless user experience.

4. Design

In this section, a straightforward explanation of the design of the choropleth map in the application that exploits the above key insights and perform the four important task smoothly and efficiently: (1) color component of the map: represent the data of two prevalent health factors in the map (obesity and diabetes), (2) data indicators via color matching: identify the value of the two prevalent health factors, (3) observation through data processing: formatted data in backend, (4) user interaction: fast responsive to user's request. In the application, the frontend and backend development support the goal of efficient run and real-time speed to provide data to users. There is a direct, symbiotic relationship between the data and the color representation seen on the final map, and the process by which we calculate the color key and assign values will be expanded on below.

4.1 Preprocessing Data and Database Structure


```
{
    "name": "colorado",
    "valid_data": true,
    "population": 5812069,
    "obesity_percentage_afflicted": 24,
    "obesity_population_afflicted": 140652069,
    "diabetes_percentage_afflicted": 7,
    "diabetes_population_afflicted": 40684483,
    "mean_obesity_percentage": "24.00",
    "mean_diabetes_percentage": "7.00",
    "std_obesity_percentage": "3.58",
    "std_diabetes_percentage": "1.57"
},
{
    "name": "connecticut",
    "valid_data": true,
    "population": 3605597,
    "obesity_percentage_afflicted": 29,
    "obesity_population_afflicted": 106365111,
    "diabetes_percentage_afflicted": 8,
    "diabetes_population_afflicted": 29565895,
    "mean_obesity_percentage": "29.00",
    "mean_diabetes_percentage": "8.00",
    "std_obesity_percentage": "2.91",
    "std_diabetes_percentage": "0.82"
},
```


Image 3: Sample of JSON Items Stored in the Backend

To ensure the accuracy and clarity of our visual representation to the clients, the data preprocessing stage is a crucial phase. The biggest and most comprehensive dataset we found came from “County Health Rankings & Roadmaps", and provides our clients with a starting point for well-rounded data[1]. Through the use of a Python script, the relevant data is directly pulled from the Excel document (County/State name, Obesity Percentage, Diabetes percentage, and Population) and appended onto the JSON object representing the specific county so it can be efficiently used by the system for storage and retrieval. The “cleaning,” formatting, and storage

of our data are imperative so that both our clients and the application's frontend can easily access the relevant data points that power our visualization tool.

4.2. Interactive Map Design in the Backend

While we want the frontend of our application to display both the choropleth map and additional mouse hover data, we want the backend to do most of the legwork. The backend will be responsible for normalizing the data and storing it. The only thing the frontend is responsible for is fetching the data from the backend. However, since there are large amounts of information, we designed the frontend to either fetch a single state and its counties at a time, or display the national-level information (state only). This way, our application remains efficient and the user can see the data they need instantly.

4.3 Interactive Map Design in the Frontend

The core of our system's visualization power is the generated choropleth map, which is the culmination of backend data processing and normalization. The front-end visualization can have three different options. A user can choose to view an obesity-only color scale, a diabetes-only scale, and a bivariate scale. The different colors signify the density and prevalence of obesity and diabetes within the cities and counties of a certain state, leveling up to the national level. Lighter shades represent lower rates, progressing to darker shades that indicate higher percentages of obesity, diabetes, or both.

Images 4,5, and 6 explain each of the color keys that are used within our application. When the user is examining either obesity or diabetes, the key uses three distinct colors to represent the data. When the user is examining obesity and diabetes together, the key expands out to nine

distinct colors to represent bivariate data. Thought was given to each of the color choices, and how the colors for obesity and diabetes could “blend” with each other. We ultimately decided on red and blue because these are primary colors that blend together to create purple, an easy visual cue that a region will contain both obesity and diabetes data. The gradation of colors provides not only an aesthetic dimension to the data, but also serves as a key analytical feature, enabling the discernment of patterns, correlations, or notable differences in health data across regions.

Image 4: Bivariate Representation

Image 5: Diabetes-Only Representation

Image 6: Obesity-Only Representation

Below are the color codes for each square in the overall key and a description of what each square represents (e.g. the Middle Middle square in the bivariate key represents higher obesity and higher diabetes). These same red/blue colors are also used for the “mini” keys for obesity and diabetes.

- Bottom Left - "#e6e2e2" - Low obesity and low diabetes
- Bottom Middle - "#a0b3d3" - Low obesity and higher diabetes
- Bottom Right - "#5c85c5" - Low obesity and high diabetes
- Middle Left - "#d6a1a2" - Higher obesity and low diabetes
- Middle Middle - "#967f97" - Higher obesity and higher diabetes
- Middle Right - "#565f8d" - Higher obesity and high diabetes
- Top Left - "#c65b5d" - High obesity and low diabetes
- Top Middle - "#8b4857" - High obesity and higher diabetes
- Top Right - "#503651" - High obesity and high diabetes

For the bivariate key, the X-axis represents diabetes, while the Y-axis represents obesity. The color allocation corresponds to different ranges of obesity and diabetes percentages:

For the “mini” keys, there is either no X-axis or Y-axis; the entire key is simply for either obesity or diabetes. By visualizing such critical health data, the tool will play an instrumental role in supporting public health initiatives and resource allocation. It is designed to be intuitive and user-friendly, ensuring that users from diverse backgrounds can navigate the data with ease and extract meaningful insights.

5. Implementation

While end users will only ever interact with the chorobesity project in a high-level context, understanding the manner in which the project was created helps to provide even further insight into the design goals of its implementers.

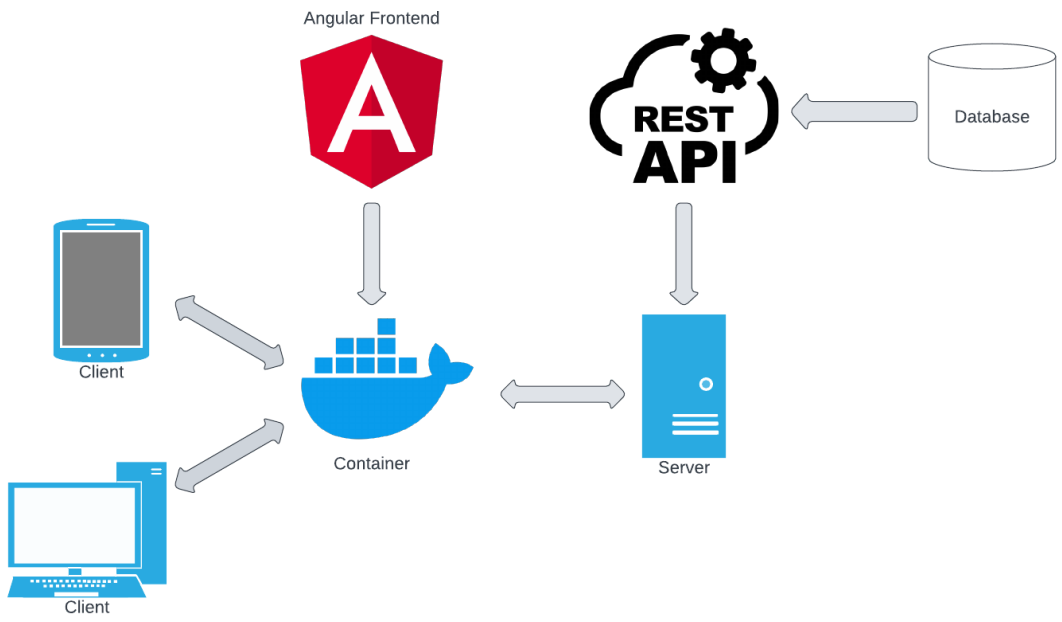

Image 7: Original Application Architecture with Inclusion of Dockerization

5.1. Overall Implementation

The inner workings of the software can be separated into three distinct components, each pertaining to a unique role within the overall implementation. The first of these components is the frontend, which serves as the interface used by the end-user. It's designed to be as minimal as logistically possible so that the end-user can only interact with elements relevant to the data at

hand. It should also be clean, modern, and well-designed, with a well-tested user interface and user experience workflow.

The next component is the backend API. This is the side of the application opaque to end-users. Accessible only through HTTP(S) requests made by the frontend, the backend is connected to the application database and is able to set, get, or manipulate data within it as necessary. It serves as the interface between the "code" of the application and the data of the database. The raw data is pulled in from CSV files, which is then appended to JSON objects representing each county/state. The numbers are then normalized by calculating the standard deviation of the dataset by state and assigning each county an appropriate color key based on how many factors away the county's statistic is from the standard deviation.

The application's final component is the database, which simply stores the data relevant to the application in an easily accessible format. While, in a full-stack application such as this, this is typically done with a structured query language (MySQL, PostgreSQL, etc.), all that is truly necessary is for the data to be readable via code - even a .txt file could be considered a "database." Below are detailed explanations of the implementations of each of these components, as well as their purpose and usage within this application.

5.2. Frontend

The Chorobesity project application will utilize an Angular 2.0 framework for its frontend implementation. Angular, like other frameworks, is used to compile HTML, (S)CSS, and TypeScript files into a deployable package. The use of a framework allows for simple implementation of HTTP(S) requests through its provided library, as well as other features expected of a modern web application.

5.3. Backend

For the backend of our application, we have decided to utilize the Django REST framework to serve as our API and database communication service. Interestingly, Django, like Angular, is in fact also a web framework that can be used to construct a frontend application environment. However, when used in tandem with the Django REST framework library extension, the framework can instead serve as a powerful API service, with the Django framework hosting the REST API.

This is precisely how we've utilized these tools in the Chorobesity application. Using the Django REST framework, we will create a comprehensive API from which the front end can access data essential to the core facets of the application. This includes but is not limited to, getting county boundaries as readable JSON files and retrieving obesity data related to given counties. By making data accessible through the backend rather than directly through files accessible by the frontend, we introduce the opportunity for administrators to update the datasets used in the future without disrupting the application flow.

5.4. Database

Our comprehensive database system, hosted on PostgreSQL, serves as a robust repository that integrates GeoJSON files and CSV files to create the interactive visual tool the users see. The GeoJSON files play a pivotal role in housing geospatial data, specifically identifying state and county boundaries[3]. Through a systematic process, our backend efficiently retrieves and assimilates these files' data, enabling our application to dynamically render geographic boundaries and present visually compelling representations to end-users in real-time.

Complementing the GeoJSON, our system fetches pertinent data from CSV files which holds various statistics, such as obesity rates, diabetes statistics, population figures, and percentage distributions within respective counties[1]. After the data is fetched from the backend, the data is then normalized on a consistent scale. Normalizing the data allows us to maintain data integrity and coherence, ensuring uniformity across datasets for each county. Notably, this uniformity allows our system to effectively apply color scales to each county based on normalized values. By employing this standardized approach, our application's visual representation of health-related statistics across counties becomes intelligible and meaningful, enabling users to compare data seamlessly across different states and regions.

6. Testing/Evaluation/Assessment

The testing process started with a focus on the front end, specifically the interactive map functionality. Front-end testing involved validating the map's responsiveness, and ensuring accurate display and user interaction functionality, like hovering across various states and counties. Next, we tested the database and backend integration. Rigorous testing procedures were executed to confirm the proper retrieval and handling of data. This stage involved inspecting the backend's ability to fetch and process GeoJSON and CSV files accurately. Emphasis was placed on validating the integration between the backend and PostgreSQL database, verifying that the data ingestion process was error-free and that data integrity was maintained throughout. We also focused on the normalization process, with comprehensive testing to ensure that normalization algorithms accurately calculated the correct statistical figures for obesity and diabetes percentages. Validation procedures were conducted to confirm that these normalized figures were stored correctly within each county's respective data object in the database. Concluding the testing cycle, the frontend and backend integration were tested to cross-check that the frontend visualization aligned with the backend data retrieval and processing. Special emphasis was placed on confirming that the colors representing obesity and diabetes percentages were displayed accurately for each county, to allow users to intuitively compare health statistics across the United States at the county level.

7. User's Manual

This section is committed to providing information on how to interact with our application. It's split into two broad sections: setting up the application, as well as how to interact with the application itself.

7.1. Application Setup

The application setup is covered extensively in the Developer's Manual section.

7.2. Understanding the Map Hierarchy and Hover Feature

The biggest feature of this project is its map display. Upon launching the application, the user can immediately view the map and interact with its features, including the hover capability. Each of the map levels will be discussed in detail below.

7.2.1. National-Level Map

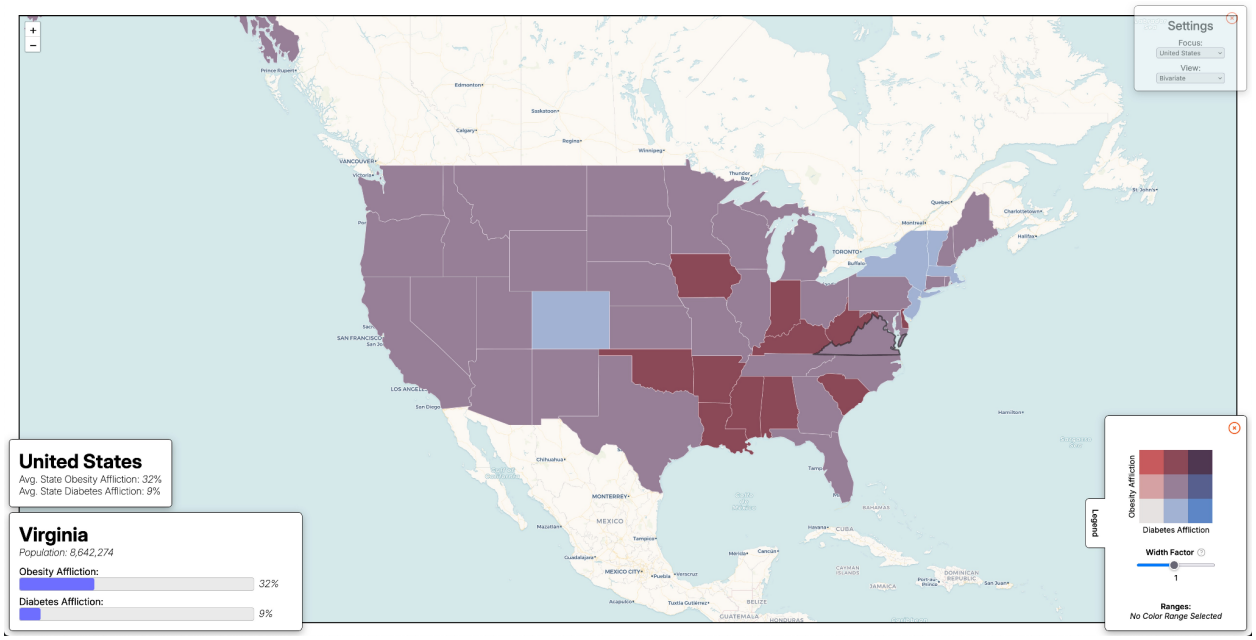

Image 8: Image of the Application on the National Level

On a national level, the obesity and diabetes data for each state is displayed through the map, which includes Alaska and Hawaii. As shown in Image 8, the bottom part of Alaska can be seen in the top left corner of the image. The user can click and drag with the mouse to navigate to other sections of the map, revealing the state's color and data. In the image above, you can see that the border around Virginia is bolded, indicating that the mouse is hovering over this specific state. In the bottom left corner, there are two boxes; one displays the name "United States", and the other displays the name "Virginia". Both contain percentage information about the state that is currently being hovered over with the mouse, and the bottom box also lists the population of the current state.

7.2.2. Switching Focus Level

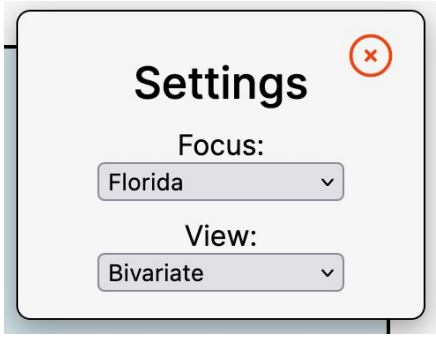

Image 9: Close-Up Image of the Settings Menu

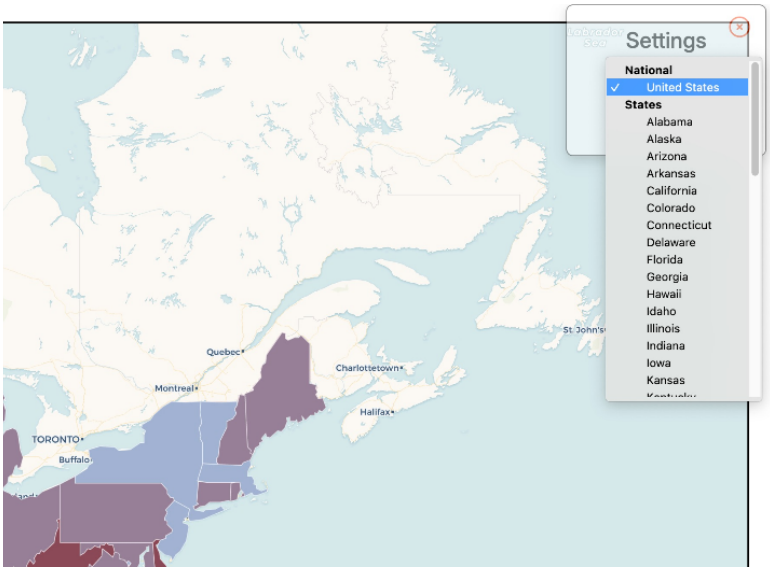

Image 10: Cropped Image of the Settings Menu (Focus)

If a user wants to navigate to a new map level, they can look to the Settings Menu in the upper right-hand corner of the application (it can be clicked on to expand out). The first dropdown menu, seen in Image 9 and titled Focus, is where the user can select a new map to view. In addition to the national level, the user can click on any of the 50 states to view them at a more detailed scope, as seen in image 10. This finer focus views each state separately and adds county

boundaries to the map, making it easier for the user to see the geographical correlation of the health data within the selected state itself.

7.2.3. State-Level Map

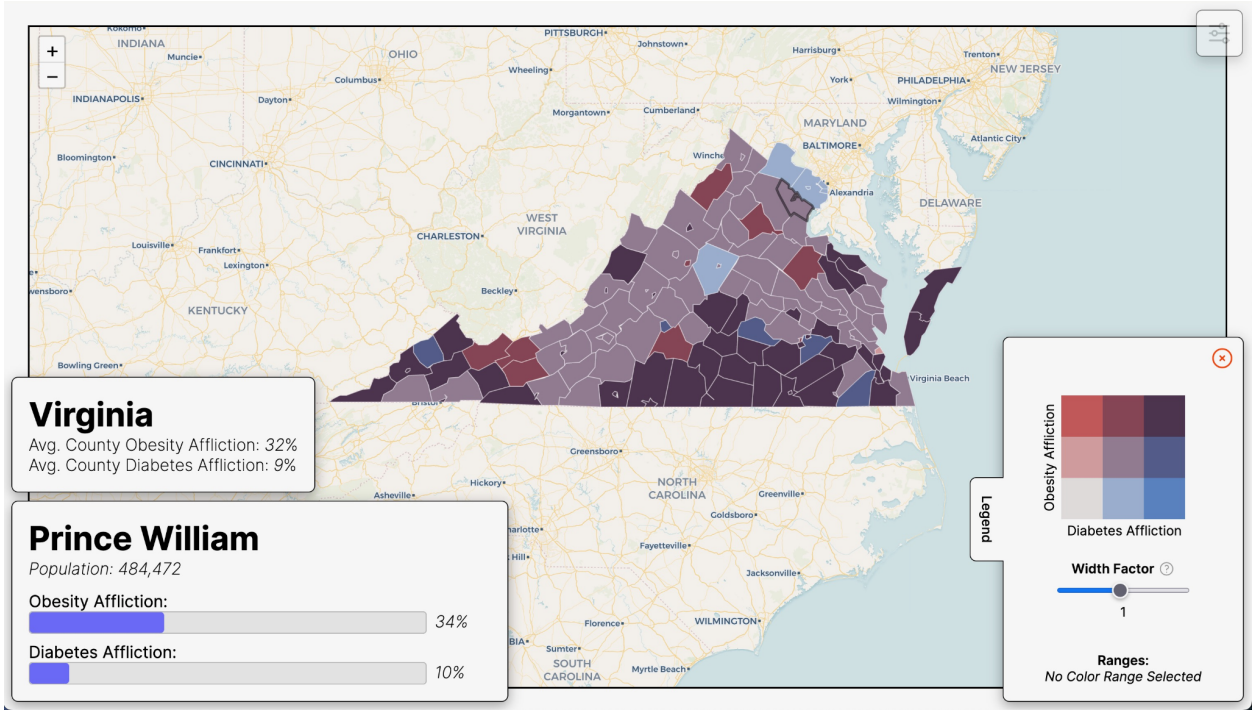

Image 11: Image of the Application on a State Level with a county selected

Once a user has chosen which state they want to view, the application updates the map accordingly. This map level contains similar information as the national level, but instead views one state at a time and includes its county boundaries. The user can hover over different counties and major cities and display their corresponding data, as seen in Image 11. The top box in the lower left-hand corner displays the name of the state and its affliction percentages, and the bottom box displays the name of the county that is currently being hovered over, along with its corresponding information.

7.3. Color Keys

7.3.1. Color Key Features

There are three main color key features present on the legend: the color key itself, the “Width Factor”, and “Ranges”. Each of these will be expanded on in more detail below.

7.3.1.1. Color Key Box

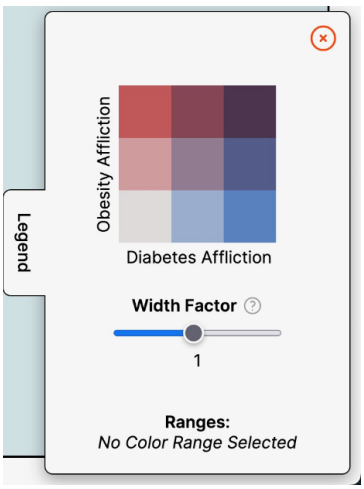

Image 12: Close-Up Image of the Color Legend (Bivariate)

In addition to the choropleth map itself, the application utilizes a color key that is located in the lower right-hand corner of the screen. This legend is expandable, so clicking on the Legend tab on the side will show the full key. At the top, depending on the color key that is selected, will be a square key that is labeled with obesity as red and diabetes as blue. If only obesity or diabetes is selected for the key, then the key will be only red or blue, respectively. The bivariate key combines the red and blue values to create purple values for overlapping data, which can be seen in Image 12.

7.3.1.2. Width Factor

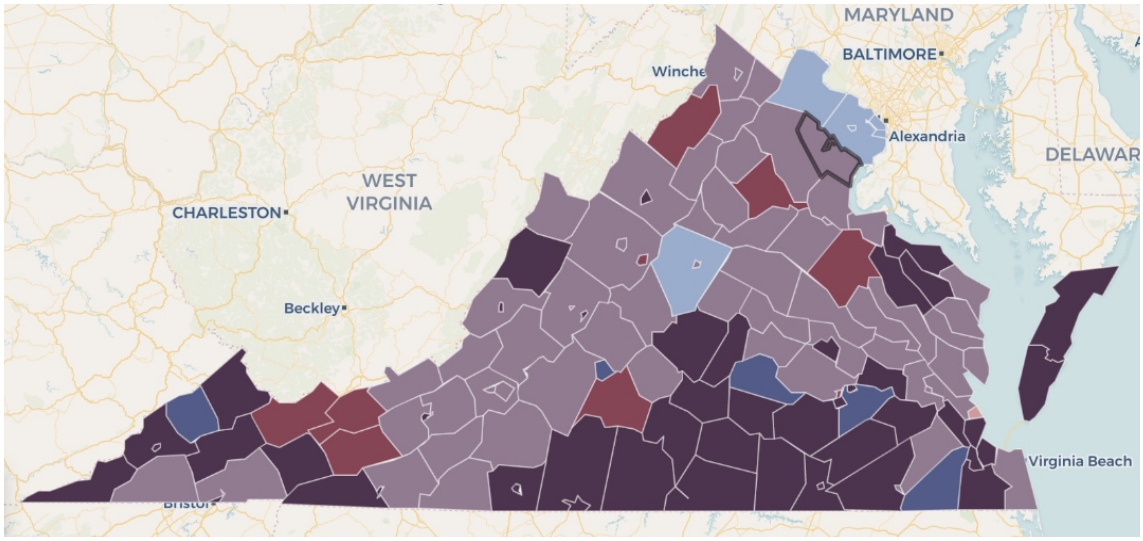

Image 13: State Map with a Width Factor of 1

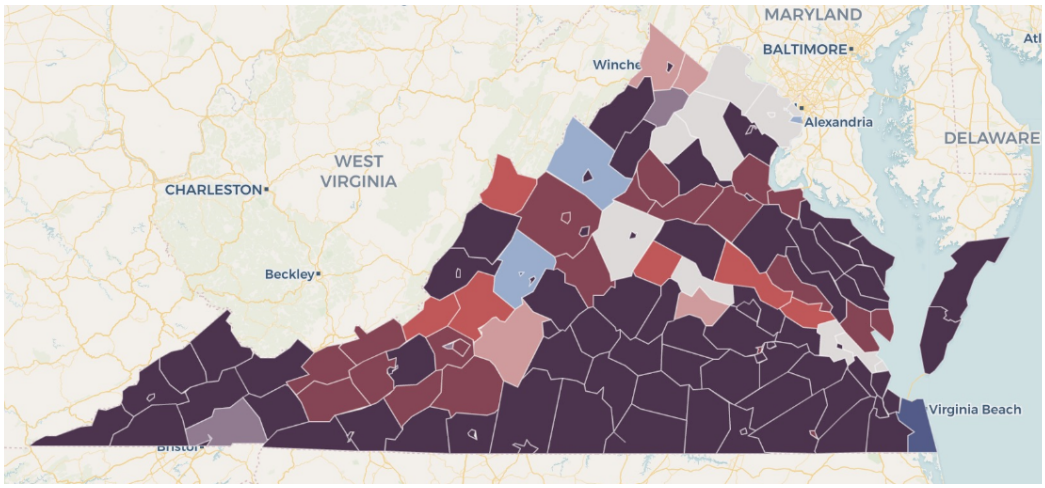

Image 14: State Map with a Width Factor of 0.1

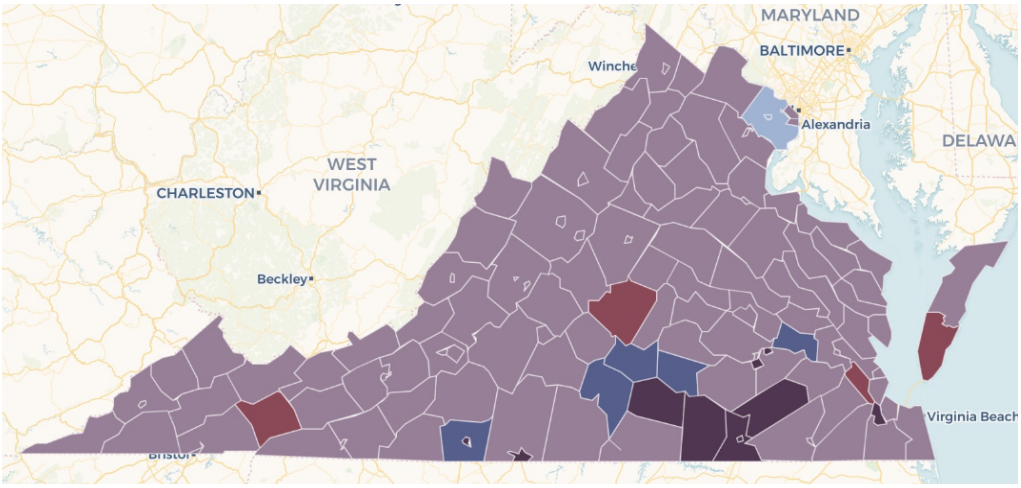

Image 15: State Map with a Width Factor of 2

Below the key is a slider for “width factor,” which adjusts the standard deviation values that were calculated from normalizing the data. This is provided to the user so that they can adjust to what degree they wish to view the data, allowing for the user to have flexibility in the display of their data. For example, a high-width factor would create a choropleth map with the outlier states or counties clearly defined. Images 13, 14, and 15 display how the width factor value can affect the map display. If more information is needed about this tool, the user can click on the “?” beside its title for an explanation.

7.3.1.3. Ranges

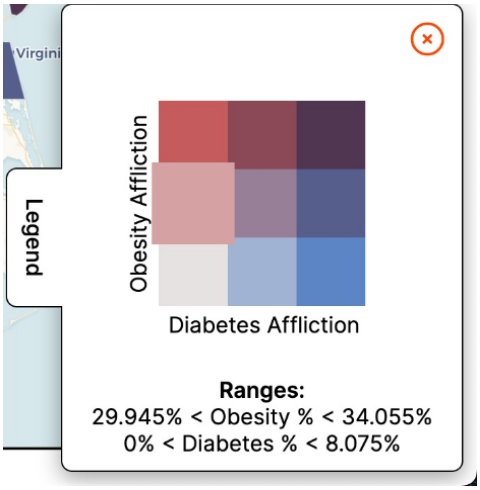

Image 16: The Ranges for the Left Middle Box in the Color Key

Finally, the Ranges section at the bottom of the Legend works in tandem with the color key. As seen with Image 16, each individual box on the key is clickable, and clicking on a box will display the low and high percentage values for the health affliction(s) for which a data point must fall between to be that specific color.

7.3.2. Changing the Color Key

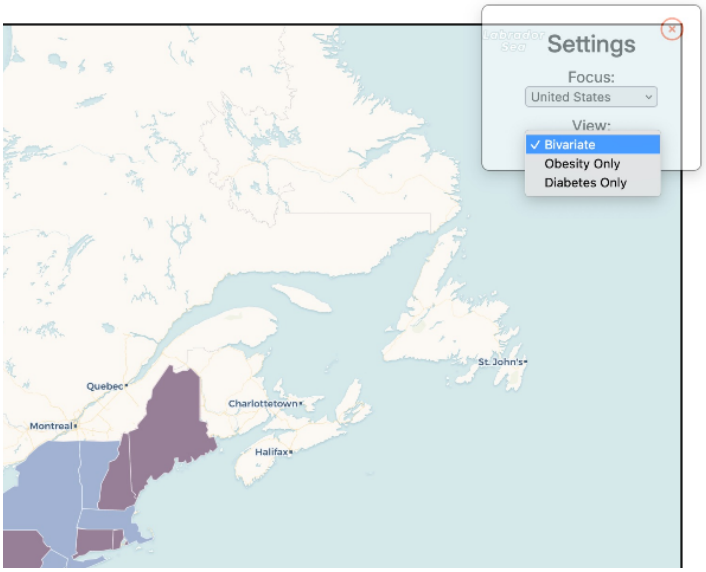

Image 17: Cropped Image of the Settings Menu (View)

Similar to changing the Focus in the Settings menu, the user can also change the color key view within this menu. There are three options to choose from: a bivariate key, an obesity-only key, and a diabetes-only key. Clicking on the preferred key in the Settings menu will also update the Legend with corresponding information.

8. Developer's Manual

The Chorobesity application has two distinct components: the Angular frontend and the Django REST backend. Both components rely heavily upon a variety of external software and libraries. Thus, in this section, you will be guided through the process required to set up the development environment locally.

8.1. Prerequisites

Alongside the utilized libraries (which we will install later), Chorobesity makes use of a multitude of full-fledged applications. To successfully set up the development environment, these must be installed prior to beginning the setup process.

1.) Git (https://git-scm.com/downloads) [Used to download code to machine]
2.) Postgres (https://www.postgresql.org/download/) [Used as the application database]
3.) Python3 (https://www.python.org/downloads/) (Ensure downloaded version is 3.X.X and not 2.X.X) [Used to run the backend and upload file]
4.) Node.js (aka Node) (https://nodejs.org/en/download)

Additionally, the most up-to-date "County Health Rankings National Data" dataset should be downloaded from the following URL:
https://www.countyhealthrankings.org/explore-health-rankings/rankings-data-documentation
This will be used as the foundational dataset from which the Chorobesity application draws. In the event that this dataset proves faulty later in the setup process (made apparent by failure during the upload step), previous versions of the dataset should be used. Notably, the 2023 dataset is known to be working.

If the above applications have already been instead installed prior to this step, ensure you are on the latest version of each prior to proceeding. Otherwise, follow through with the installation steps presented. When installing Postgres, if prompted, download any and all related applications (such as pgAdmin). Upon completion, ensure Postgres was properly installed by opening the application in the selected directory. Additionally ensure Git was properly installed by opening the Windows Command Prompt (which can be done by pressing the Windows key and typing "cmd") or MacOS terminal (accessible by pressing ⌘+Space and typing "terminal"), and entering the version command

COMMAND: git --version

If installed successfully, Command Prompt/Terminal should output the version of the installed Git application.

Next, ensure Python3 has installed properly with the following version command:

COMMAND: python3 --version

Similarly to Git, installed successfully, Command Prompt/Terminal should output the version of the installed Python3 application.

Finally, ensure Python3 has installed properly with the following version command:

COMMAND: node --version

Similarly to Git, installed successfully, Command Prompt/Terminal should output the version of the installed Node application.

If all four applications have installed successfully, you may proceed to section 8.2

8.2. Cloning the Repositories

The code for both the front and back ends are hosted in what is known as a *repository*, or an online storage center for code. If you wish to replicate the development environment, the following subsections will provide step-by-step instructions for downloading the code onto your computer:

8.2.1. Clone the Repositories

To download the codebases to your machine, you must use the URLs which point to the code repositories in which they are stored (provided below) alongside the Git application. This can be done as follows:

1. Open Command Prompt/Terminal on your local machine
2. Take note of the current working directory of the Command Prompt/Terminal (discoverable by entering the command "cd" on Windows and "pwd" on MacOS). This is the folder where the code will be cloned. Note that the frontend and backend will automatically download into two separate folders: *chorobesity*, and *chorobesity-backend*, respectively.
3. Clone the frontend codebase with the following command:

COMMAND: git clone https://github.com/freteck/chorobesity.git

1. Clone the backend codebase with the following command:

COMMAND: git clone https://github.com/freteck/chorobesity-backend.git

2. Next, clone the separate data uploader codebase (containing two files: a Python3 script which will upload the data to the backend database, and a simple text file explaining its usage) from its repository using the following command:

COMMAND: git clone https://github.com/freteck/chorobesity-data-uploader

3. Finally, open a File Explorer/Finder window to the location discovered earlier in this step and move the three folders (chorobesity, chorobesity-backend, and chorobesity-data-uploader) to a location of your choosing. This is where the development environment will be located from here on out.

8.3 Installing Angular

With the codebases downloaded, we can now begin the actual setup process, beginning with the frontend. The frontend runs using the Angular web development framework and thus, to run the project locally, Angular must be installed. This can be done as follows:

8.3.1. Check Angular Installation

First, we can check to see if Angular has already been installed as follows:

1. Open your terminal or command prompt.
2. Type the following command to check if Angular is already installed:

COMMAND: ng version

If Command Prompt/Terminal outputs a version, Angular is already installed on your machine. If it fails, the application must be installed.

8.3.2. Install Angular (Skip if already installed)

If Angular is not installed, run the following command to install it:

COMMAND: npm install -g @angular/cli

8.4. (Windows Only) Set Execution Policy

For Windows users, the scripts upon which Angular relies to run by default lack the permissions necessary for execution. To fix this, we must set the execution policy to allow script execution. Open a PowerShell window with administrator privileges (which can be done by pressing the Windows key and typing "Powershell", right clicking, and clicking "Run as Administrator) and run the following command:

COMMAND: Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned

Following completion, open Command Prompt and follow the steps found in section 8.3.1 above to ensure installation finished successfully.

8.5. Install Project Dependencies

The Chorobesity application frontend relies upon a variety of external libraries in execution, most notably Leaflet.js. Thus, prior to running the application, these dependencies must be installed on your machine. Luckily, Angular has built-in functionality to make this process simple.

1. Open Command Prompt/Terminal
2. Use the "cd" command to navigate to the chorobesity folder cloned earlier:

   COMMAND: cd <path/to/chorobesity>
3. Once in the chorobesity folder, all necessary dependencies can be installed with a single command:

   COMMAND: npm install

8.6. Run the Frontend Locally

Once all dependencies have been installed, the frontend can be run locally using Angular's "serve" command. Note, however, that in production/once deployed, the application utilizes a file structure entirely separate from that used with the "serve" command. Thus it cannot always be assumed that development behavior is always replicable in production. This aside, the frontend should now be runnable using the Command Prompt/Terminal window used in the above subsection 8.5 with the following command:

COMMAND: ng serve

By running this command, the development environment automatically compiles and "serves" a live version of the application to the user. If any changes were to be made within the frontend codebase's live files (.html, .ts, and .scss), the application would automatically be recompiled and re-served. By default, the served application can be accessed at http://localhost:4200. If an application is already running on port 4200, Angular may change the port being used. Additionally note that localhost is not a "real" website. Rather, it is a reserved URL used by your computer to access applications running locally. Thus (without additional setup not covered within this manual), it is only accessible by the computer on which it is run.

Once you have navigated to http://localhost:4200, you will likely notice that nothing has been rendered. This is because we have not yet set up the backend, which is required for the application to run.

8.7. Set Up PostgreSQL

PostgreSQL, known also as Postgres, is a database management system which the Chorobesity application uses to store data related to obesity and diabetes affliction and serves as the primary long-term storage method. Unlike previously installed dependencies, Postgres is an independent application which must itself be set up. This can be done as follows:

- Windows:
    a) We must first disable the Postgres password system so that we may access our database. To do this, perform the following:
        i) Press the Windows key + “R”
        ii) Type “Notepad” and hit “Enter”
        iii) Click “File” in the top left
        iv) Click “Open.” This will open a File Explorer window
        v) From the file explorer window, you should see a file address bar to the left of the search bar. Click within it to reveal the address of the current working directory
        vi) Delete all of the contents of the address bar and insert the following:

            ADDRESS: C:/Program Files
        vii) Scroll down and open the folder titled “PostgreSQL”
        viii) You will be presented with a list of folders with numbers for names. Each number corresponds to a version of Postgres installed on your machine. Open the folder with the highest number
        ix) Open the folder labeled “data”

x) In the bottom right of the file explorer, there should be a dropdown menu above the “Open” button with the default value “Text Documents (.txt).” Click this drop dropdown and select the “All Files” option.

xi) Scroll down to the bottom of the folder and double-click the file labeled “pg_hba.conf”. This will open the file in Notepad

xii) Scroll to the bottom of the file. Note the six pieces of text with the value “scram-sha-256”

xiii) One-by-one, delete these pieces of text (being careful not to delete the spaces before them and not to add or remove lines to the file) and replace them with the word “trust” (case-sensitive)

xiv) Once all instances of “scram-sha-256” have been replaced with “trust”, click “File” in the top left and click “Save”

xv) The Notepad editor may now be closed

xvi) Again, press the Windows key + “R”

xvii) Type “services.msc” and click “OK”. This will open the Windows Services menu.

xviii) Observe that the services are listed in alphabetical order. Given this, scroll down to the service beginning with “postgresql-x64-{version}”, where “{version}” is the version of Postgres you have installed

xix) Right click the service and click “Restart” and wait for the service to restart.

xx) You have now removed password requirements!

b) Press the Windows key and type “pgadmin” and open the application “pgAdmin 4”, which should have been installed alongside Postgres.

c) Once pgAdmin 4 opens, right click the tab “Servers”, visible on the left-hand side

d) Select Register > Server

e) Name the Server “Chorobesity”

f) Click the “Connections” tab at the top of the popup

g) In the “Host name/address” field, enter “localhost” (case-sensitive)

h) Change the value in the “Port” field to 4132. This is *essential* for the backend to run properly

i) Click “Save” in the bottom right to create the server. By default, pgAdmin creates a database within the new server named “postgres”. This is the database used by the chorobesity backend.

- MacOS:

a) From Spotlight Search (⌘ + Space) type “postgres” and select the application to open it

b) Click the + in the bottom left to create a new server

c) Name the server “Chorobesity”

d) Select the most up-to-date Postgres version

e) ***ENSURE THE PORT IS SET TO 4132***. If set otherwise, the application will not run properly

f) Click “Create Server”

g) Click “Initialize” in the top right

Your Postgres database is now running and ready to be connected to the application backend! By default, Postgres disables your server/database upon restarting your machine. Thus, upon reboot, it may be necessary to open the Postgres application and start the server up again when you want to use the Chorobesity application locally.

8.8. Running the Backend

Once Postgres has been successfully set up and a server has been initialized and run, you may progress to setting up the backend itself. This can be done as follows:

1. Using Command Prompt/Terminal and the “cd” command, navigate to the *chorobesity-backend* folder.
2. (MacOS Only) Typically the process of setting up the backend requires multiple commands to be run. On MacOS have simplified this process however, by creating a single command which in turn runs these necessary commands in sequence. This initialization command can be run as follows:

   COMMAND: make initialize
3. (Windows only) Unfortunately, Windows lacks the ability to run the “make” command specified in the step above without the installation of additional software. For this reason, we will run each command individually, as is typical. These commands are as follows:

   a. Install all of the backend dependencies (also known as requirements) with:

      COMMAND: python3 -m pip install -r requirements.txt

b. Convert our python3 database code to a format readable by Postgres:

   COMMAND: python3 manage.py makemigrations

c. Apply our converted code to the database:

   COMMAND: python3 manage.py migrate

Upon the completion of this command, the backend has been successfully set up and is ready to be run. This can be done with the following command:

COMMAND: python3 manage.py runserver

This will run the backend server and, assuming both Postgres and the application frontend are both running, automatically connect the three components. Congratulations! The Chorobesity application has been successfully set up on your local machine!

8.9. Uploading Health Data

At this point, we have successfully cloned, installed, and set up all of the necessary components for the Chorobesity application - all of which should be currently running. However, although we have the database set up and prepared to accept health data, we have yet to actually submit any such information. Thus, for the final step, we must run one final program to upload all of the relevant, up-to-date health data to the Chorobesity database. This can be done as follows.

1. Open two File Explorer/Finder windows
2. Navigate the first to the location where you downloaded the "County Health Rankings National Data" .xlsx file/dataset in the earliest steps of this set up process
3. Navigate the second window to the *chorobesity-data-uploader* folder we cloned from earlier
4. Either move or copy the .xlsx file/dataset to the *chorobesity-data-uploader* folder
5. Open a Command Prompt/Terminal window
6. Using the "cd" command, navigate to *chorobesity-data-uploader* folder

7. To execute the upload script, we must run it using Python3 and provide it with the necessary fields as arguments. These arguments, in this order, are as follows:
    a. base_url: the url at which your backend is running (By default this should be "http://localhost:8000"
    b. filename: the exact filename (including ".xlsx") of your" County Health Rankings National Data" file
8. Knowing this, we can run the data uploader with the following command:

   COMMAND: python3 python_data_uploader.py <base_url> <filename>

   EXAMPLE: python3 python_data_uploader.py http://localhost:8000 data.xlsx

   Note that, if your .xlsx file/dataset has a filename which contains spaces, then the argument should be wrapped in quotation marks, as follows:

   EXAMPLE: python3 python_data_uploader.py http://localhost:8000 "my data set.xlsx"
9. The program should output as it executes, with detailed error messages should something lead it to fail
10. Verify that the data was properly uploaded by visiting the following url:

    {base_url}/api/states

    EXAMPLE: http://localhost:8000/api/states

    If successful, you should see a list of all 50 states, with relevant health data attached. Additionally note, from this view, county-level data can be viewed by appending /{state} to the URL

    EXAMPLE: http://localhost:8000/api/states/virginia

Congratulations! Following the upload of the dataset, you have successfully set up the full Chorobesity application! From here you are able to work as you please on the front or back end and, as you are using the development environment, changes will be reflected live.

8.10. Project Hierarchy

8.10.1. Frontend

The following is the directory tree structure of the src folder, where code relevant for the development and upkeep of the project frontend is stored. Note that the src/ directory is *not* the root directory of the frontend. However files outside of this folder are considered high-level and should generally not need to be modified if one wishes to further develop the project. A brief description of the tree structure below is provided after it.

```
src/
│
├── favicon.ico
├── index.html
├── main.ts
├── polyfills.ts
├── styles.scss
├── test.ts
│
├── app/
│   ├── app-routing.module.ts
│   ├── app.component.html
│   ├── app.component.scss
│   ├── app.component.ts
│   ├── app.module.ts
│   │
│   ├── legend/
│   │   ├── legend.component.html
│   │   ├── legend.component.scss
│   │   └── legend.component.ts
│   │
│   ├── map/
│   │   ├── map.component.html
│   │   ├── map.component.scss
```

```
│   │   └── map.component.ts
│   │
│   └── services/
│       ├── backend.service.ts
│       ├── color.service.ts
│       ├── map.service.ts
│       ├── shape.service.ts
│       └── utils.service.ts
│
├── assets/
│   ├── .gitkeep
│   │
│   └── geojson/
│       ├── counties.geojson
│       └── us-states.json
│
└── environments/
    ├── environment.prod.ts
    └── environment.ts
```

The first files in the src/ directory, *favicon.ico, index.html, main.ts, polyfills.ts, styles.scss,* and *test.ts* are global to the project and with the exception of *favicon.ico* and *styles.scss* should *not* be altered. As for these files, however, there purpose is as follows:

- favicon.ico: A small image representing the application which displays next to the page title within it's browser tab
- styles.scss: A standard SCSS file containing global styles. That is, any SCSS classes defined within this file can be used across the entire application, without the need to explicitly import or redefine such classes.

Moving on, next is the app directory, which contains, alongside information regarding the applications itself, a collection of what are known in Angular as Components - denoted by folders with their corresponding names. These Components represent either modules present within a page on the application or the entire page itself and serve as the foundational building blocks of the application. Each Component is separated into three distinct files, each with an

essential purpose respective to the component. These files are as follows, where {name} denotes the name of the component:

- {name}.component.html: A standard HTML file providing the virtual structure of the Component. Designates the way in which inner elements should be arranged, if/how they should interact with scripting methods, and assigns SCSS classes to elements
- {name}.component.scss: A standard SCSS file which provides styling to the component's elements
- {name}.component.ts: A standard TypeScript file which is capable of interacting with the elements of the Component

Given that the file structure is identical across every Component, we will instead describe below the different components present along with their purpose in the application:

- Legend: The module visible in the bottom right of the application denoting the scale upon which colorations are made, as well as which allows for alteration of Width Factor values
- Map: The main application page. Makes use of Leaflet.js and the OpenMapBox API to display the visualization.

Next is the services folder. In Angular, services are TypeScript files with utility TypeScript functions shared across multiple components. Services cannot be displayed with the application and have no function aside from providing Components with these utility functions. This process is known as *injection*. By separating these functions to their own files, we minimize code duplication that would have occurred otherwise. The services utilized within this project and their purposes are as follows:

- Backend: Provides functions necessary for interacting via HTTP protocols with the application backend.
- Color: Provides functions allowing the Map and Legend to receive color info depending on current settings values
- Map: Provides setter and getter for the current map view
- Shape: Provides getters for the county and state boundaries
- Utils: Provides a variety of general utility functions relevant to the project - nonspecific to any particular category

Following this, the *assets*/ directory stores static asset files needed by the application. The .gitkeep file within is used by the Git application, while the geojson/ directory contains two .json/.geojson files, each providing geojson data respective to the region described in their name Finally, the *environments/* directory contains files allowing for the creation of "secret data." When developing an application, it's not unusual to need to access/manipulate a variable containing sensitive data. However, since the source code of an application is easily available to its users (this is especially true for web applications, where the inspector can be opened by pressing F12), we want to make sure that this sensitive data is not visible to those that view such code. This is where *environment variables* come into play. Environment variables can be defined in a file that is not accessible to the user from the client and read without ever exposing itself. These variables are defined within the *environment.ts* and *environment.prod.ts* files, with the former being used automatically within development environments and the latter being used automatically within production environments. In the case of this application, the only defined environment variable is "backendUrl", which by default is set to "http://127.0.0.1:8000" and

which should point to the base url of the backend, without a slash appended to the end. Should the address of the backend be changed, this value should be altered as well.

8.10.2. Backend

Similarly to the frontend, the root directory of the backend largely consists of files which comprise the structure and general functionality of the Django REST Framework (DRF) API. As such, during the development, they should almost always be left untouched. This is with the exception of the root-level file *.env*. Similarly to the *environment.ts* file in the frontend, Python3 also supports the usage of environment variables. The only such variable defined in this file is "FRONTEND_URL", which by default has a value of "http://127.0.0.1:8000" (note 127.0.0.1 is the exact same as the word "localhost") and which should point to the URL of the frontend. If the frontend URL or port were to ever change, this environment variable should be changed to reflect that.

Unlike Angular, which makes available to the user a wide variety of files for the purpose of modification, developing a DRF application only requires the modification of a handful of files - all present within the root-level directories *chorobesity/* and *chorobesity_api/*, the structures and contents of which have been transcribed below:

```
chorobesity/
│
├── __init__.py
├── admin.py
├── apps.py
├── models.py
├── serializers.py
├── tests.py
├── views.py
│
└── migrations/
    └── __init__.py
```

For the purpose of further development of this application, it's best that the files *__init__.py, admin.py, apps.py,* and *tests.py* remain unchanged. The remaining files, however, should be modified in the event further changes to the backend are desired. These files and their purposes are as follows:

- models.py: Contains a series of classes known as *Models* which can store specific data fields - similar to an object in an object-oriented programming language. Our team has already implemented the State and County models within this file, which store data relevant to the obesity and diabetes rates of these respective region-levels. When new objects are created in the backend, they are created as instances of the models defined here.
- serializers.py: Contains the method in which models should be converted from database SQL code to human-readable JSON. Each new model requires a corresponding serializer to be created alongside it. By default, serializers simply output each field of a model as a key-value pair, however this can be altered as desired.
- views.py: The primary interface for the API, this is where new API endpoints are created. By default, DRF creates the endpoints *list(), create(), retrieve(), update()*, and *destroy()* - although these are not defined explicitly. They can be overridden within this file, or additional endpoints can be created using the action decorator, as has been done with *flush()*

```
chorobesity_api/
│
├── __init__.py
├── asgi.py
├── settings.py
├── urls.py
└── wsgi.py
```

Similarly to above, the contents of this folder should largely remain unaltered. However, there are two files enclosed within which may require alteration - albeit the need to do so is much less common than with the files in the prior folder. These files and their purpose are as follows:

- settings.py: Contains settings relevant to the backend. While these should almost all remain as-is, fields commonly in need of change are *DATABASES,* which contains the username and password to your Postgres database, and *CORS_ORIGIN_WHITELIST*, which contains a list of all URLs allowed to access the API. URLs other than that specified here will be rejected and have an HTTP error returned to them upon request. We have inserted a reference to the environment variable *FRONTEND_URL* as the only permitted address.
- urls.py: Contains a router, used to direct a user to a set of API endpoints upon navigating to a specified URL. The only endpoint we've established is "states", which can be navigated to as follows:

  ADDRESS: {base_url}/states

  Additional urls can be added by adding a line along the following structure following the existing *router.register()* call on line 7:

  CODE: router.register(r'{location}', views.{viewset})

  In this code block, {location} is the desired addition to the url (such as "/states" in the example above), and {viewset} is the name of the ViewSet class being directed to, as defined in views.py (StateListViewSet, in the example above).

9. Results

After finalizing the project, we have successfully incorporated all of the client's requirements and expanded Chorobesity's scope from only Virginia to the national level. We were able to retrieve a comprehensive dataset from a valid source, which aided us in creating a comprehensive and multi-level representation tool. We also provided dynamic representation aids that the user can utilize to understand the geographical correlation of diabetes and obesity rates throughout the country. Our team believes that this work expands the understanding of health issues of United States' citizens, and that our clients will be able to utilize this application to further their own research.

10. Lessons Learned

10.1. Timeline

Table 1 is an overview of the timeline for our work on the project:

Date	Milestone
September 15, 2023	Go through the requirements of the Project
September 29, 2023	Establish a management framework from the inside
October 5, 2023	Collect diabetes and obesity data
October 23, 2023	Finish the initial iteration of the project (ready to be viewed by clients)
November 1, 2023	Finish design and implementation work
November 10, 2023	Final touch and testing & containerization work
November 14, 2023	Finish report and presentation
November 30, 2023	Submit all paperwork

Table 1: Project Timeline

10.2. Problems and Solutions

Throughout the development process, our team encountered many difficulties in various components of the project. Below, we discuss some problems that the group encountered as well as their solutions, which are expanded on and discussed.

10.2.1. Preprocessing

The raw data sets we researched from a variety of sources were often inconsistent, which posed a significant challenge in creating a comprehensive dataset without gaps that could be processed properly. Before finalizing the dataset referenced in “References”, finding reliable data was a

difficult task. Initially, our focus was on successfully setting up Virginia on the map, and then gradually expanding to a national level, integrating the data we found step by step. As the scope of the project grew, so did the need for more and more comprehensive data. Truly, finding a fully comprehensive dataset was the foundation for the achieved functionality of the application.

10.2.2. Related Database

Given the extensive amount of information available on obesity and diabetes, establishing a database that performed well and maintained its integrity was challenging. The volume of data needed to generate the different choropleth maps influenced how to display all of this information. Careful attention was required in implementing this part of the project, and the ultimate solution was to create a dropdown menu that displays states and their corresponding county data one at a time, to improve data retrieval times and keep the application fast and efficient.

10.2.3. GeoJSON file acquisition

A significant challenge we encountered was sourcing an accurate geoJSON file that included all outlines of both the states and counties in the country. This file was critical for our project's functionality, as clearly defined boundaries were the foundation to containing the different color shadings for different regions. The group was able to utilize coordinates found on Cartography Vectors[3].

10.2.4. Finding a color to represent both the parameters

Another hurdle was deciding how to represent each county visually on the map. This involved choosing appropriate colors to depict different data points related to obesity and diabetes, and

was discussed extensively in the Design section of this report. Ultimately, the group chose red for obesity, blue for diabetes, and purple for the overlap of data. From there, different shadings were developed.

10.3. Future Work

Below, we will discuss aspects of the application that can be developed further in the future.

10.3.1. Containerization

The next phase of this project could greatly benefit from focusing on containerization. This involves building and bundling the application to ensure it can run efficiently in various environments. The implementation of containerization would provide enhanced portability and scalability, essential for maintaining the robustness and versatility of the application. By using this method, the program and its dependencies might be contained inside a container, enabling consistent and reliable performance across a range of infrastructures. Containerization would ultimately make it easier for users of both Mac and Windows software to use the application and reduce the need for separate instructions for the execution of the application.

10.3.2. Mobile Integration

Another advancement recommended for future teams is the integration of the application with mobile platforms. It is essential to optimize the application for these platforms given the increasing reliance on mobile devices. This phase would include modifying its features and interface to make them completely compatible and easy to use on a variety of mobile devices. The goal of mobile integration is to increase accessibility and make sure the application runs well across a range of smartphones and tablets.

10.3.3. File Upload (Frontend Support)

An important feature for future development is implementing the frontend infrastructure for file uploads. This improvement would enhance the interactive and customized components of the user experience by enabling users to submit their datasets directly into the program. Ensuring compatibility with numerous file formats and designing an easy-to-use interface for file uploads should be the main priorities. This addition would significantly enhance the application's capabilities in data handling and visualization, making it a more comprehensive tool for research and analysis.

11. Acknowledgements

Professor Lynn Abbott

Contact Email: abbott@vt.edu

Dr. Lynn Abbott is a professor at Virginia Tech and is the sponsoring client of the Chorobesity project. He received his B.S. at Rutgers University before earning his M.S. at Stanford and finally his Ph.D in Computer Vision and Electrical Engineering. His research interests lie in his field of doctoral studies, as well as in biometrics and autonomous vehicle systems. His publications in this work have received over 3,000 citations. Guidance and expertise from Professor Abbott were instrumental in steering the project towards its successful culmination. Always available to address queries, he offered essential information that enriched the overall direction of the initiative.

Chenyu Mao

Contact Email: mchenyu@vt.edu

Chenyu Mao is a sponsoring client of the Chorobesity project. He has contributed significantly by providing a visual depiction of the anticipated outcome. This visual guide has served to unify the team's direction, ensuring all members are aligned with the intended objectives.

Professor Mohamed Farag

Contact Email: mmagdy@vt.edu

Essential data samples provided by Professor Farag laid the foundation for this initiative. We express our gratitude to Professor Farag for his guidance and support throughout the semester.

12. References

[1] "Explore Health Rankings | Rankings Data & Documentation." County Health Rankings & Roadmaps, www.countyhealthrankings.org/explore-health-rankings/rankings-data-documentation.

[2] BVG Software (@benjaminadk). (2019, July 17). Bivariate Choropleth color generator. Observable. https://observablehq.com/@benjaminadk/bivariate-choropleth-color-generator

[3] Cartography vectors. (n.d.). Cartography Vectors. https://cartographyvectors.com/map/1129-virginia-withcounty-boundaries

[4] "Obesity is a Common, Serious, and Costly Disease." Centers for Disease Control and Prevention, 20 July 2022, www.cdc.gov/obesity/data/adult.html.

[5]"2020 Virginia Election Results (Published 2021)." The New York Times - Breaking News, US News, World News and Videos, 17 Feb. 2023, www.nytimes.com/interactive/2020/11/03/us/elections/results-virginia.html.

[6]"Diabetes and Obesity Maps." Centers for Disease Control and Prevention, 21 Sept. 2022, www.cdc.gov/diabetes/data/center/slides.html.

[7]"Obesity - Symptoms and Causes." Mayo Clinic, 22 July 2023, www.mayoclinic.org/diseases-conditions/obesity/symptoms-causes/syc-20375742.