PheWAS-ME in-depth exploration

Appendix A for *PheWAS-ME: A web-app for interactive exploration of multimorbidity patterns in PheWAS*

This article is a longer-form and less-formal companion to the manuscript *PheWAS-ME: A web-app for interactive exploration of multimorbidity patterns in PheWAS* and accompanying application.

The PheWAS-ME (short for Phenome Wide Association Study Multimorbidity Explorer) project represents a significant collaborative effort bringing together Electronic Health Records (EHR) and Biobank data using R and Shiny.

Figure 0.1: The main dashboard view of PheWAS-ME.

0.1 Format

The following is organized into three primary sections: **what** the app does, **why** it does it, and **how** it works.

1 What

1.1 The product

The paper introduces PheWAS-Multimorbidity Explorer (PheWAS-ME): a Shiny application that we have been producing over the past two years in collaboration with the Vanderbilt Drug Repurposing team. It is an interactive data visualization and exploration tool for digging into PheWAS results and the subject-level data that generated those results.

1.2 PheWAS

PheWAS is a statistical method for finding associations between a given genetic mutation (often a Single Nucleotide Polymorphism, or SNP) and phenotypes. It is a sibling to the GWAS (genome-wide association study). However, whereas in GWAS, you fix your desired phenotype and scan the genome for associations, in PheWAS, you fix a genetic mutation, and scan the 'phenome.'

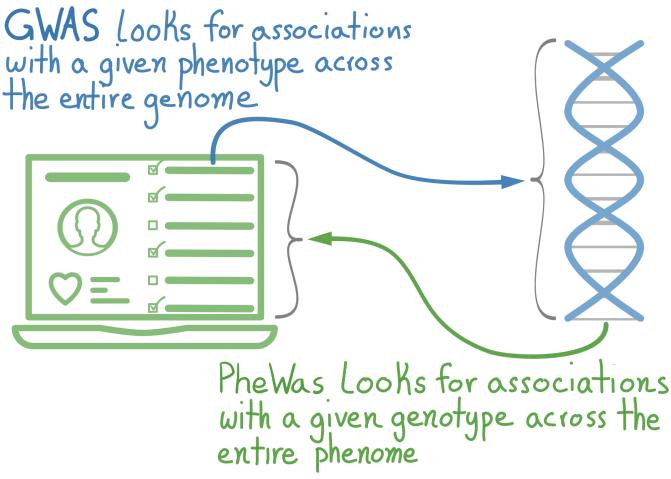

Figure 1.1: Slide from [15]

1.3 The data

Both PheWAS analyses and PheWAS-ME take two types of subject-level data: First is information on a given genetic mutation. In this case, 0, 1, or 2 copies of the minor allele of an SNP. The second is the subject's 'phenome.' In this case, a list of every phenotype that they had. In the paper and this blog post, we have used the Vanderbilt developed 'Phecode,'[7] but any binary phenotype information works.

Note: In this post, the words 'codes' and 'phecodes' will be used interchangeably.

2 Why

2.1 The need for PheWAS-ME

PheWAS-ME came out of the need for subject-experts to have a fast and intuitive way to dig through the very high-dimensional results of PheWAS analyses and the individual-level data that lead to those results. Unlike a typical statistical analysis, which may return just a handful of p-values or effect-sizes of interest, a PheWAS analysis returns a single p-value and effect-size for every phenotype in the scanned phenome. In the case of Phecodes, this is around 1,800 (and goes into the many thousands for more traditional ICD-based encodings).

2.2 Previous methods

In the past, communicating these results was accomplished using two different tools:

First is a **manhattan plot**: This is simply a plot of every phenotype investigated on the x-axis and their significance (as encoded by the negative log of the p-value) on the y-axis. The manhattan plot allows the reader to pick out which codes are significant in the results while maintaining context spatially on the x-axis. (In the case of PheWAS the context is phenotype categories, in GWAS it is genetic location.)

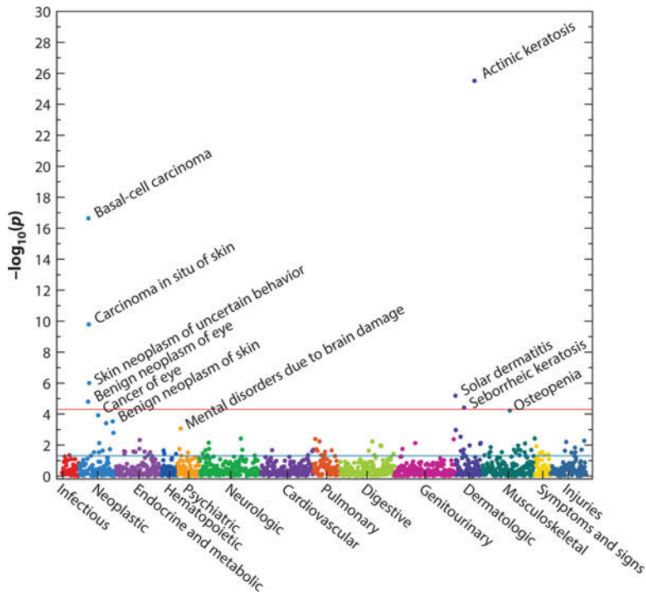

Figure 2.1: Manhattan plot for PheWAS results. Figure from **???**

The second is the ever-present results table. A PheWAS table has columns on all sorts of values of interest such as p-value, effect-size, phenotype description, etc. Due to a large number of phenotypes typically present, these tables frequently get filtered to only includes codes that fit some criteria, such as significance level.

Code	OR	P-Value	Description	Category
627.21	1.55	0.00805	Symptomatic artificial menopause	genitourinary
586.10	2.11	0.0089	Anatomical abnormatilies of kidney and ureters	genitourinary
599.90	1.76	0.00914	Other abnormality of urination	genitourinary
172.00	0.773	0.0102	Skin cancer	neoplasms
801.00	1.6	0.0102	Fracture of ankle and foot	injuries & poisonings
285.10	1.3	0.0108	Acute posthemorrhagic anemia	hematopoietic
627.20	1.76	0.0114	Symptomatic menopause	genitourinary
172.20	0.929	0.0118	Other non-epithelial cancer of skin	neoplasms
276.60	0.858	0.0118	Fluid overload	endocrine/metabolic
288.00	0.786	0.0121	Diseases of white blood cells	hematopoietic

Figure 2.2: Table of PheWAS results from simulated data provided with app.

2.3 Problems with previous methods

While looking at the most significantly associated phenotypes is straightforward, it can mask important aspects of the data. Different phenotypes across the phenome correlate with each other in complex ways. These correlations manifest themselves in common patterns of phenotypes called multi-morbidities (also referred to as comorbidities). Since PheWAS looks at single SNP and Phenotype associations, these correlations can be extremely hard to discover and reason about with a traditional manhattan plot and table report it's not clear if the same individuals are contributing to two phenotypes being significant or the populations generating "significance" are distinct, hinting at potentially more interesting biological phenomena.

2.4 How does PheWAS-ME fix these problems?

PheWAS-ME helps subject-matter experts parse through the results of a PheWAS analysis by letting them isolate and explore specific subsets of phenotypes by looking not only at their p-values and effects-sizes but also by the subject-level data that generated those results.

The analyst selects a set of phenotypes to explore by dragging a selection box around a region in a manhattan plot, choosing directly from a table, or using text-search. Once a desired set of phenotypes is selected,

the application displays the subject-level data using an interactive force-directed network plot and an upset plot.

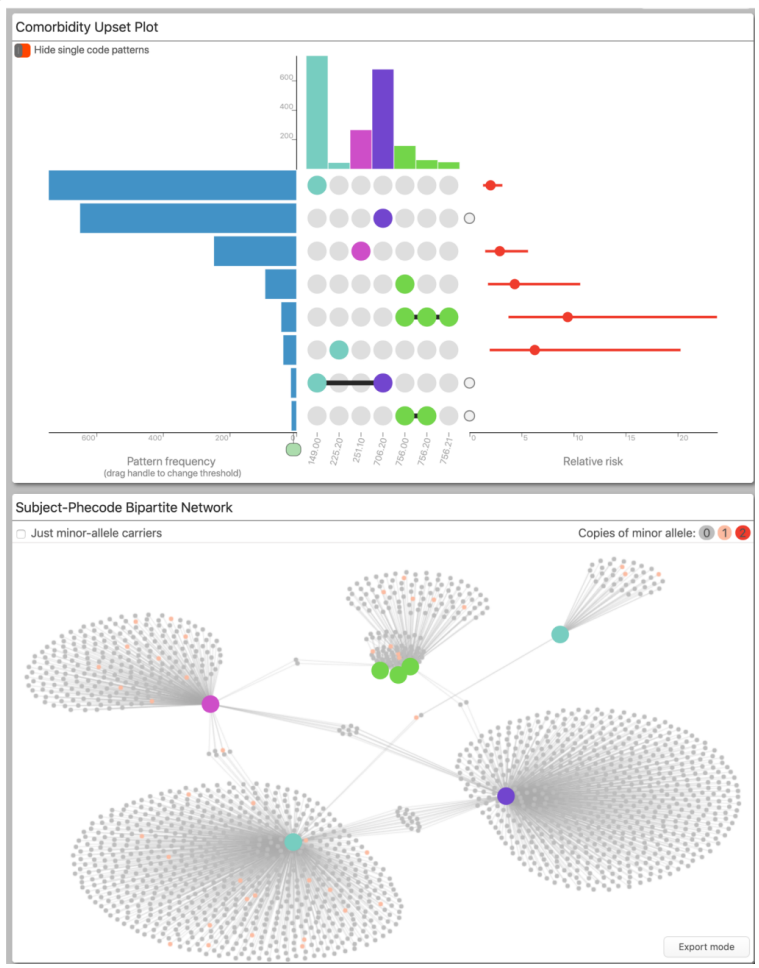

Figure 2.3: Subject-Level plots in app. Note the cluster of phenotypes in light green. This cluster is caused by the affomentioned phenotype hierarchy, with each refering to the same broad-level phenotype with differing levels of specificity.

These subject-level visualizations allow the analyst to see patterns of co-morbid phenotypes directly and to interrogate their potential causes. For instance, often, a group of codes are highly correlated because they

are more or less specific definitions of the same phenotype, e.g., cancer -> lung cancer -> stage 2 lung cancer. Another possibility is multimorbidity caused by drug side-effects; e.g., patients taking a drug to treat the cancer phenotype likely also have the nausea phenotype.

3 How

3.1 Interactivity

Interaction in PheWAS-ME centers around one primary thing: a list of currently selected phenotypes. The typical way to choose these phenotypes is by using the PheWAS results panel.

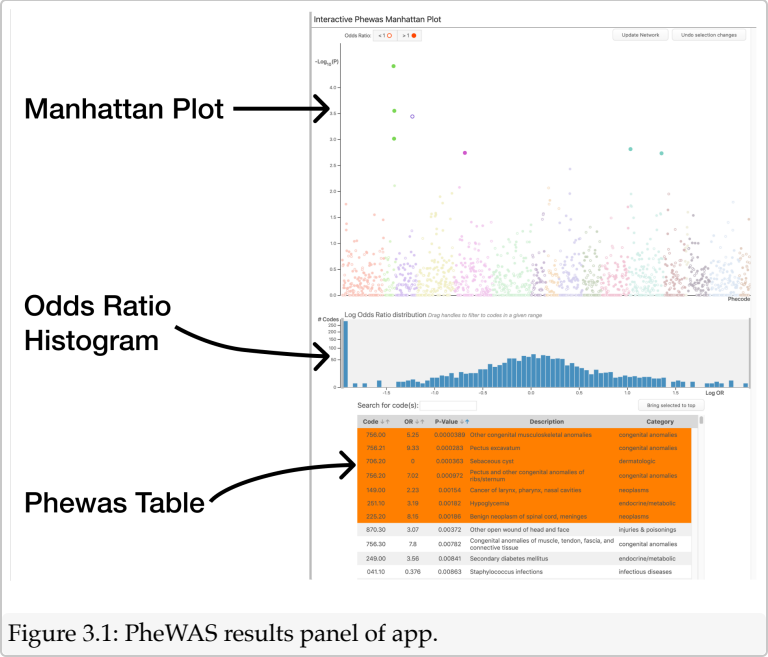

Figure 3.1: PheWAS results panel of app.

This panel includes the traditional manhattan plot and results table; however, unlike normal results plots and tables, these are now interactive with linked state. Codes selected in the manhattan plot are reflected in the table and vis-versa.

3.2 Code selection

3.2.1 Region dragging

The user can drag a box around a region in the manhattan plot, selecting the codes within. Adding another region to the selection is accomplished by holding down the 'a' key (for add) and dragging another box. Conversely, removing a region of codes from the selection can be done by holding down the 'd' key and dragging a box.

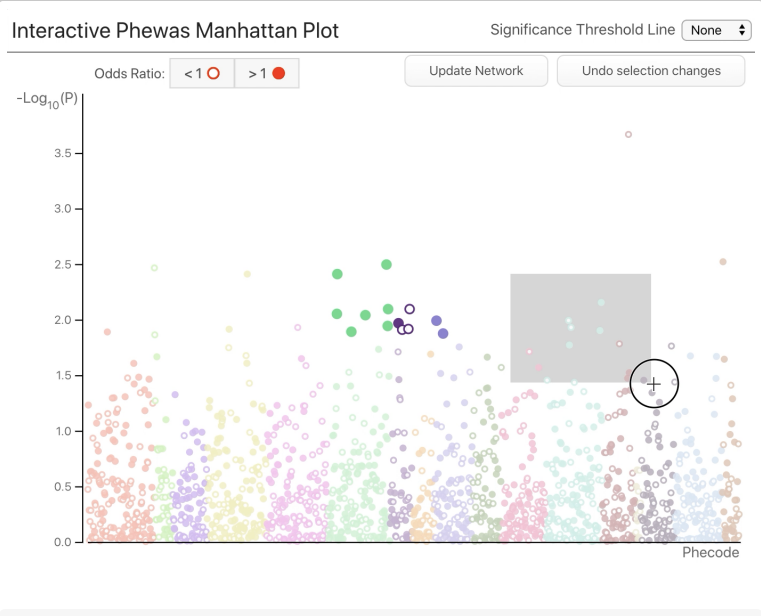

Figure 3.2: Codes in manhattan plot can be selected by dragging a box over a region.

3.2.2 Code clicking

Clicking individual phenotypes in either the plot or the table toggles their selected-ness. This slower but more precise method allows for fine-grained tuning of the selected codes.

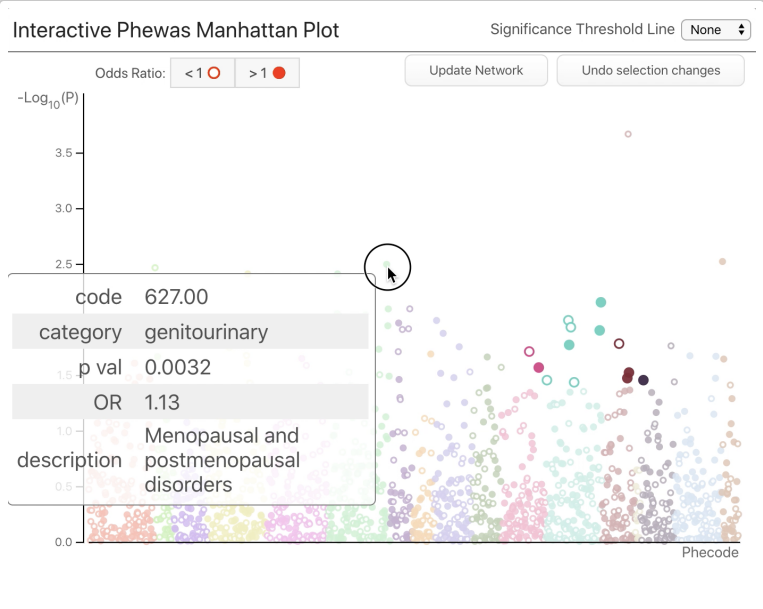

Figure 3.3: Fine-grain selection of individuals codes is possible by clicking code in manhattan plot or PheWAS table.

3.2.3 Search

Above the results table is a search bar. The user can search for phenotypes by name or description. Codes that match a supplied search query are raised to the top of the results table and highlighted. The user can then select codes as needed.

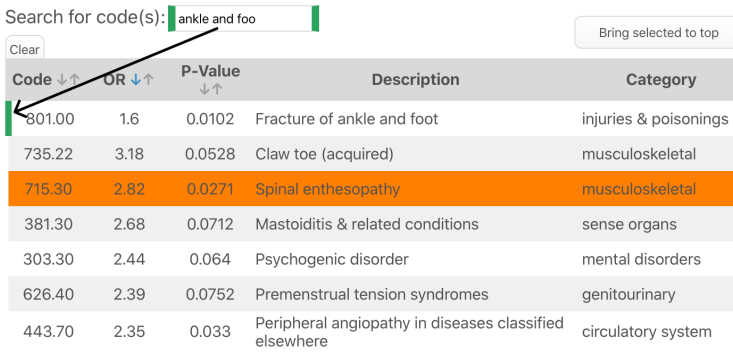

Figure 3.4: Keyword searches can be used to find specific codes by id, description, or category. Here a user has searched for 'ankle and foo' and the top result, phecode 801.00 is highlighted and placed at the top of the table for easy selection.

3.2.4 Network filtering

In the subject-level network plot, nodes corresponding to phenotypes can be used to filter the selection further. After selecting codes in the network plot by clicking, users can delete from the current selection, isolate a subset, or invert (this flips the definition of a connection to a phenotype in the network to the lack of that phenotype in a subject's phenome).

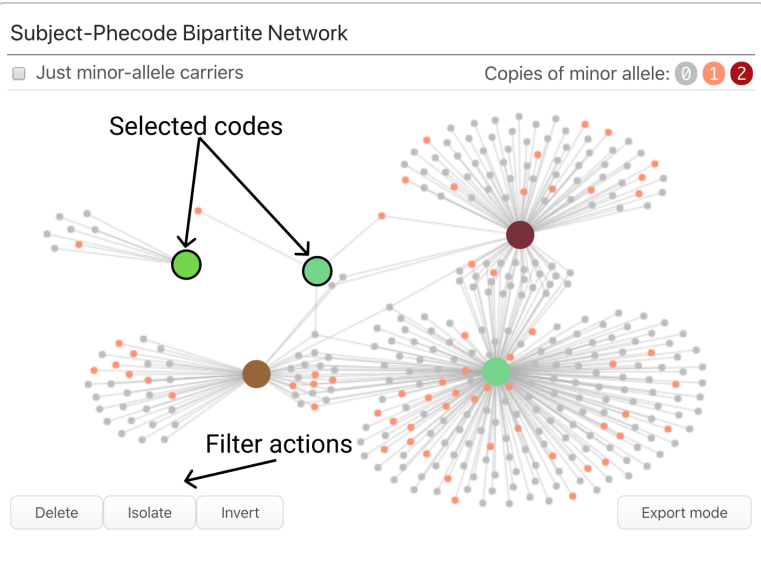

Figure 3.5: The network plot can be used to filter selected codes, allowing quick deletion of unwanted codes or isolation of interesting ones. Here, two phecodes have been selected. The app then prompts the user for filtering methods. The codes can be removed from current selection, or the selection can be isolated to just those codes.

3.3 Other interaction

Outside of phenotype selection, there are a few other forms of interaction.

3.3.1 Info tooltips

In both the manhattan plot and the network plot, when a given phenotype's point is moused-over, a tooltip with all the supplied information for that phenotype appears. This allows quick reference to metadata about the phenotype without having to refer to the results table or external resources.

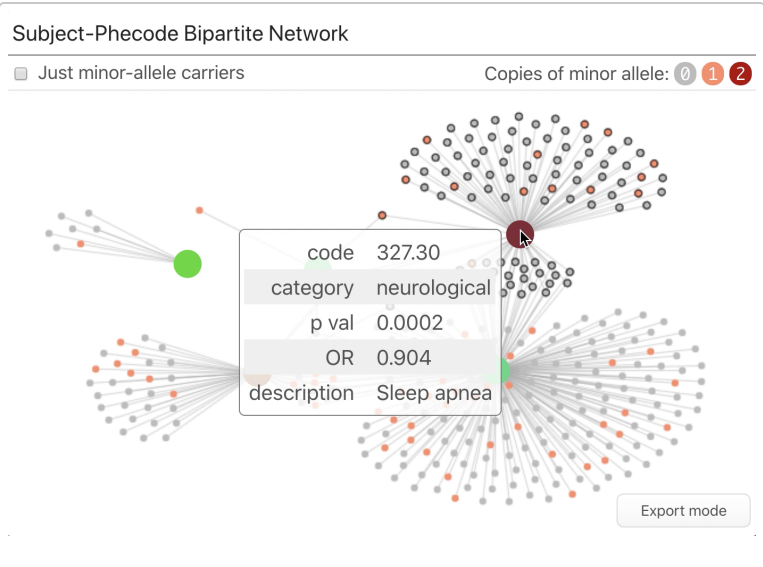

Figure 3.6: A tooltip with information about a given phecode is displayed when the mouse is hovered over that phecode's point in both the manhattan and network plots.

3.3.2 SNP filtering

A check-box by the network plot allows the user to show only carriers of the minor allele of interest in the network plot. This helps differentiate between genetics-driven network patterns and simply population trends.

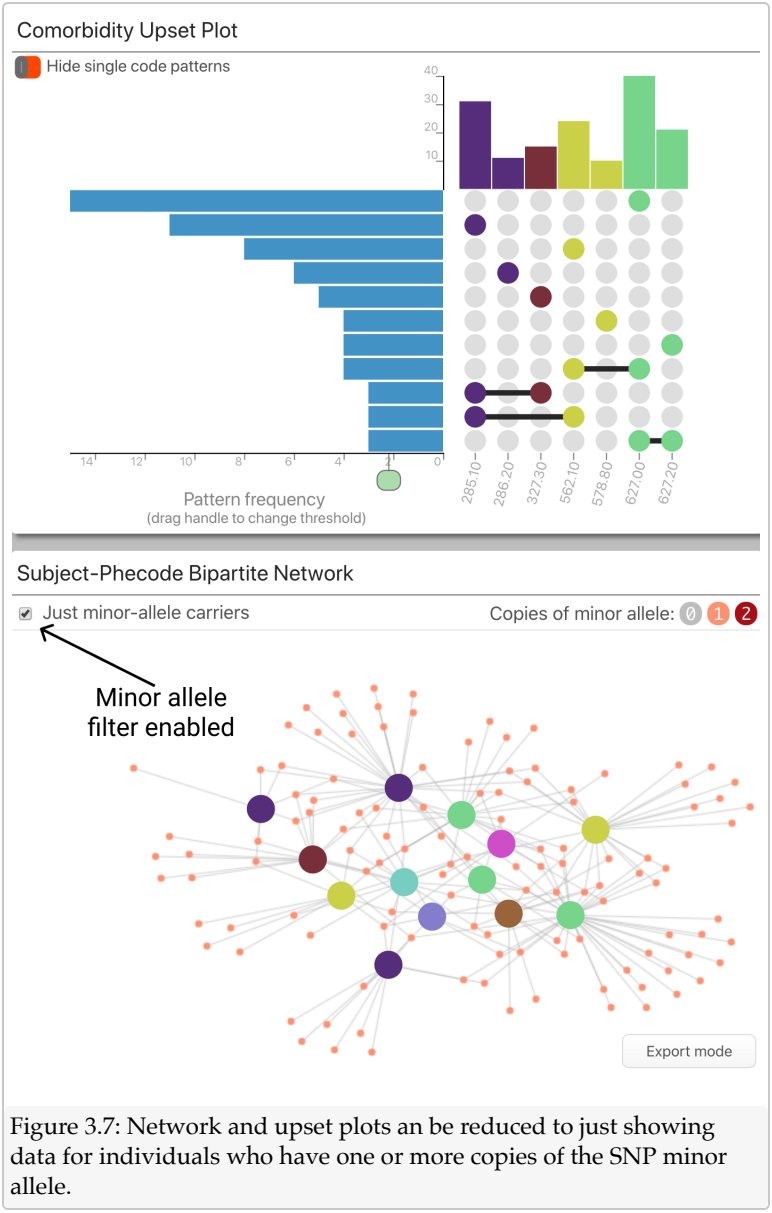

Figure 3.7: Network and upset plots an be reduced to just showing data for individuals who have one or more copies of the SNP minor allele.

3.3.3 Upset pattern highlighting

When a user clicks on a given pattern in the Upset Plot panel, the network plot updates to highlight the subjects who have that pattern. Similarly, if a phenotype node in the network plot itself is moused over, the subjects connected to that phenotype are highlighted.

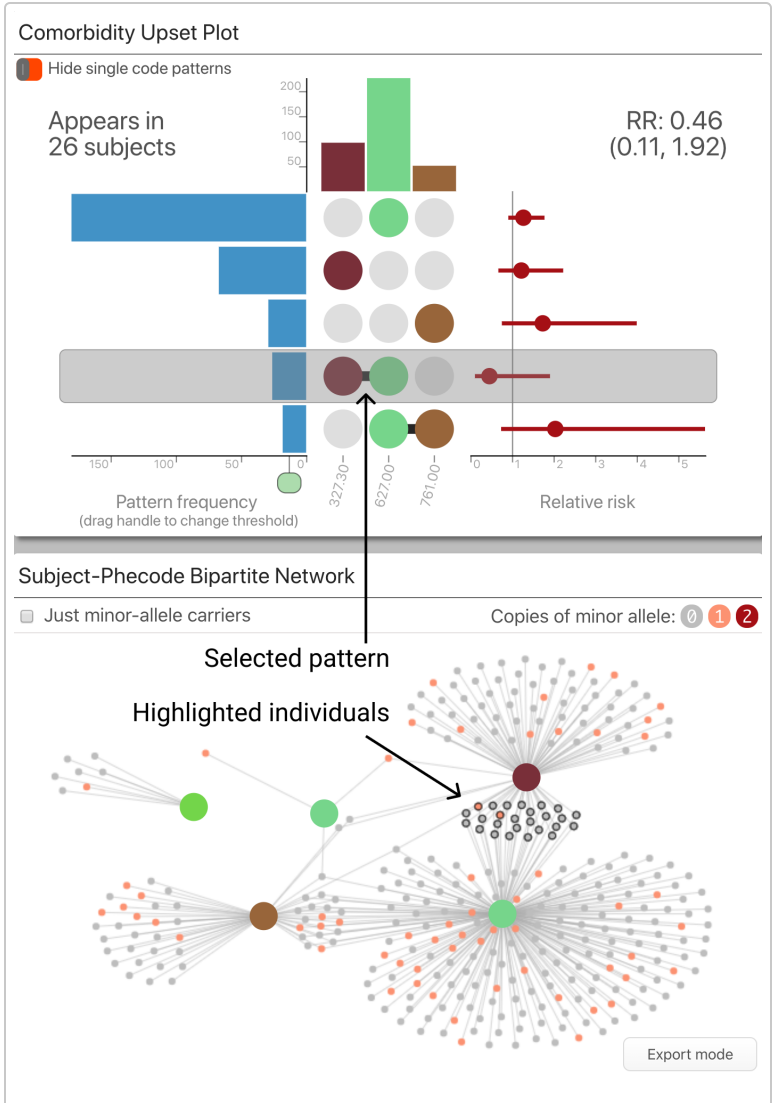

Figure 3.8: When a comorbidity pattern is selected in the upset plot, individuals with a given pattern are highligted in the network plot. Here the pattern of phecodes 327.30 and 627.00 is selected and all patients with those two phenotypes are shown below.

3.3.4 Upset singleton toggle

A toggle in the upset plot allows the hiding of single-phenotype patterns (i.e., subjects who had just a single phenotype). Often these singletons can crowd out the more interesting comorbidity patterns, so by hiding them, the user can focus on patterns.

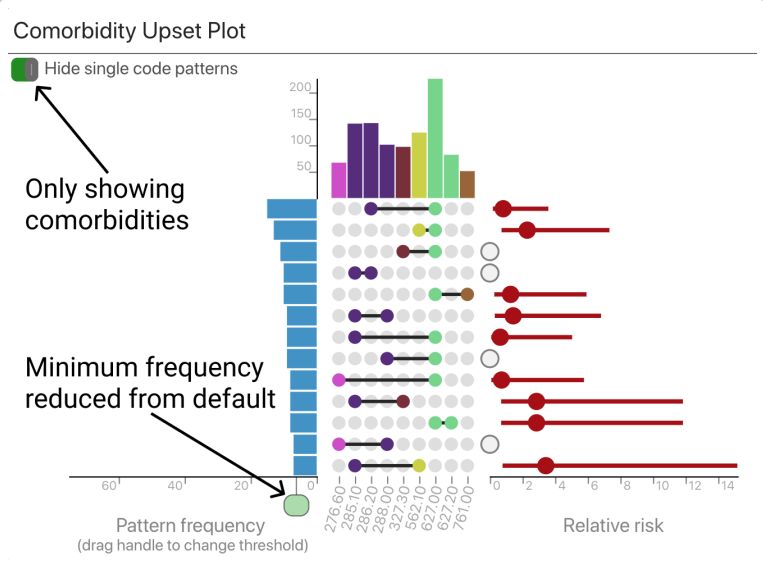

Figure 3.9: The upset plot display can be filtered to only show comorbidities of two or more phecodes, and the minimum frequency to show a pattern can be adjusted.

3.3.5 Upset minimum pattern frequency slider

Second, a slider allows the user to set the threshold for inclusion in the plot based on the number of times a pattern appears. Often there will be patterns of phenotypes that are only had by one or two subjects. Because of their tiny sample-sizes, these are usually not of interest, so filtering them out can again improve the plot's effectiveness.

3.4 How it's made

There is a lot of exciting tech going on behind the scenes of PheWAS-ME.

3.4.1 Framework/meToolkit package

The main app is built using the Shiny [4] package/framework in R. An R package was created to ease deployment and customization. There are two main functions in the package: `run_me()` which takes dataframes of all necessary information as input and starts at the main dashboard; and also `build_me_app()` which returns an app that starts at a data-loading screen where the user can either load data by uploading spreadsheets, or by picking from a list of pre-loaded datasets. All individual plots and panels exported by the package are exposed functions producing shiny modules [3]. This modular format eases the creation of customized versions of PheWAS-ME by allowing apps to be built by composing the desired modules.

3.4.2 State management

The reactivity protocol of Shiny works great if you have a single component of your app that listens to and modifies unique parts of your apps state. However, in PheWAS-ME, multiple components listen to the same state variables (for instance, both the PheWAS results table and the network plot modify the currently selected codes). A state management system inspired by Javascript's Redux [1] was built to keep each component pure and modular.

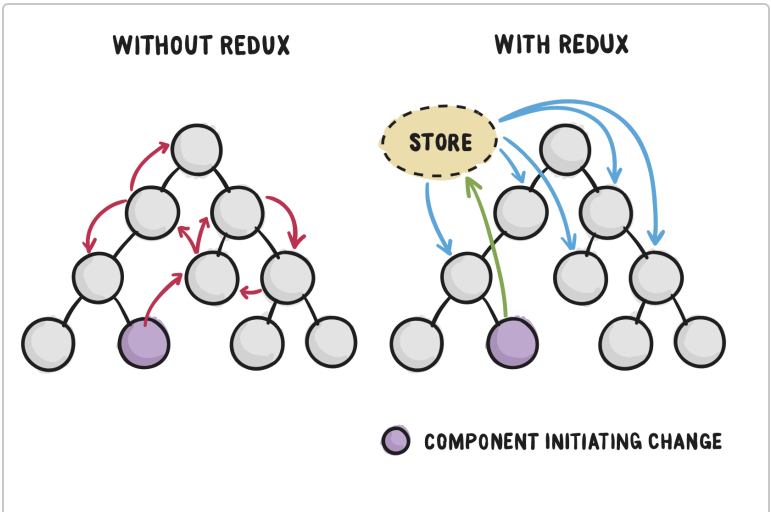

Figure 3.10: Broad level idea of how Redux works, via [12].

In this system, a single message-passing reactive value is supplied to all of the separate components. When a component needs to modify the state, it sends a message through that reactive value to the main app, which then passes the new state down to all components of the application that depend on it.

As an example, if the user removes a code using the network plot. The network plot module sends a list to the message passing reactive variable containing an action type: `'delete code'`, and then a payload: `'008.12'`. Then at the main app level, an `observeEvent()` chunk that watches that message passing reactive reads the message and modifies the appropriate state variables, in this case, the selected codes list.

By isolating all state modification in a single observable chunk, the primary app state is only ever modified in a single place. This isolation reduces the amount of overhead required to reason about state changes, both for Shiny and the programmer.

3.4.3 R to Javascript communications

Every visual in the app is built using custom javascript and HTML. The fantastic package `R2D3`[14] is used to facilitate the hand-off of data between R/Shiny and the javascript visualization code.

3.4.3.1 Manhattan plot

The manhattan plot is built using `d3js` [2], the popular javascript visualization library. With such a large number of points plotted, finding points within a selected area can be very slow when naively searching through every point.

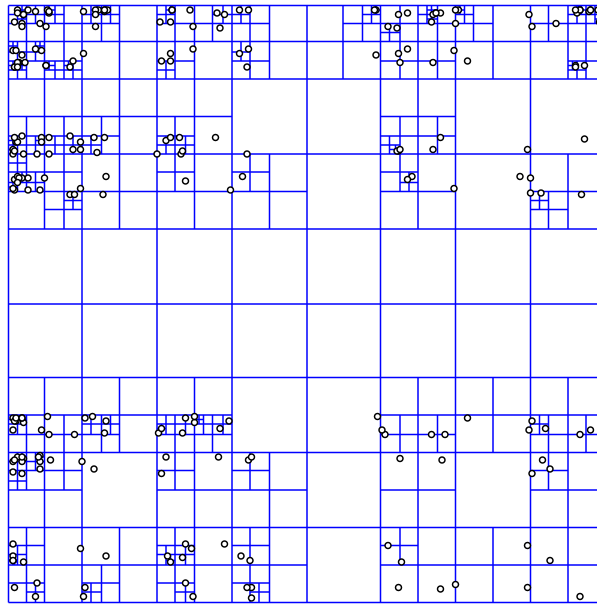

Figure 3.11: A 2d quadree storing datapoints. These subdividing boxes are stored as in a tree-structure, which is always a big-O help. Figure from [18]

To provide a responsive interface, the javascript code utilizes a special data structure known as a quad-tree [18] to speed up retrieving codes within a region. By storing data in a hierarchical location-aware tree, the quad-tree data-structure only searches through points near the selection, cutting down on the computation needed and speeding up app reaction times.

3.4.3.2 PheWAS table

Like the manhattan plot, the PheWAS table has tricks to help keep the app responsive while also showing large amounts of data. The browser stores all the elements on a webpage in a text-based format called the DOM (or Document Object Model). (What you get when you right-click and say 'view source' on a web page.) When a browser renders the page, it parses through the DOM and figures out how to place that element on the screen. Because the browser has to parse through all present elements every time it wants to update the screen, there are limits to how many elements can be stored in the DOM before things start

slowing down. Displaying a table of more than 1,800 phenotypes, along with 5+ columns, quickly hits that limit.

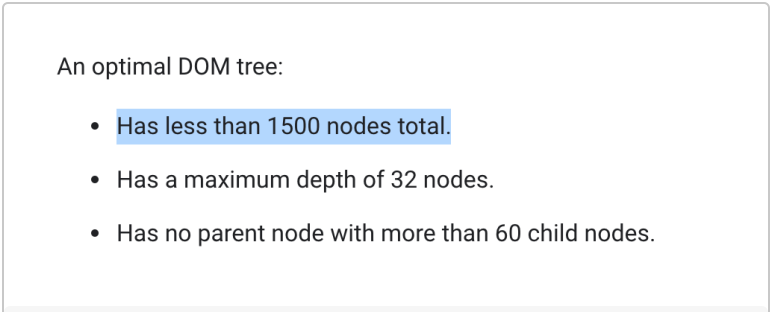

Figure 3.12: Recomendations from Google lighthouse [8] for optimal DOM performance.

The trick used to display a large table while not overloading the DOM is only rendering a small portion of the table at a time. At any moment, only 50 rows of the table are actually in the DOM. As the user scrolls down, the javascript writes new rows at the bottom and removes old rows from the top. To maintain a scroll bar for fast scrolling and visual feedback on where the user is in the table, hidden elements are placed above and below the visible rows that expand and contract to simulate a table with every row filled in. The user sees a table just like it was full of 1,800 rows, but the browser only has to render 50. Implementing this optimization resulted in interaction latency dropping by ~4 times.

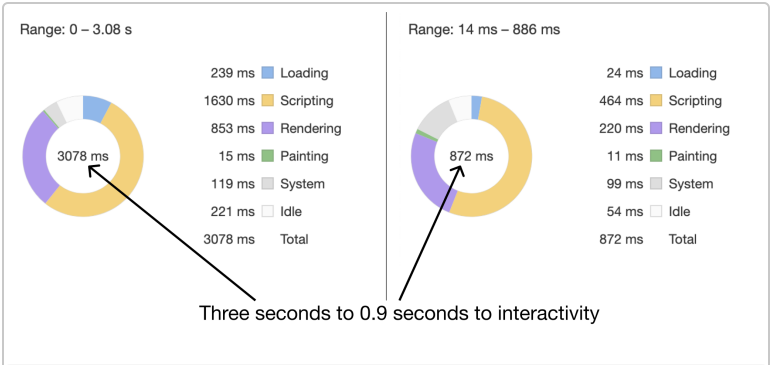

Figure 3.13: Performance improvements in results table after implementing smart-table. Via the Chrome devtool's lighthose audits [8].

3.4.3.3 UpSet plot

Like the manhattan plot, the upset plot is drawn using d3. Unlike the results table, there are not too many elements to draw here, so no fancy tricks are needed to speed it up.

3.4.3.4 Network plot

The network plot was where a majority of optimization effort was spent in PheWAS-ME. The result is a plot that can show very large subsets of individual-level data and calculate layout simulations without slowing down the app and hindering interactivity.

Drawing lots of points and lines

Often the displayed individual-level data for the app state contains tens of thousands of individuals along with many times that number of connections to the selected phecodes. To plot this amount of data often requires upwards of 100 thousand points and lines to be drawn.

Plotting lots of points and lines using javascript has a relatively simple solution: The Canvas element [5,19]. Unlike SVG[6,9] - which uses the DOM - canvas allows the user to draw to the screen at the pixel-level, basically coding an image. Because the Canvas element is just a raster image, it only takes up a single entry in the DOM and has very little performance impact.

Adding interactivity

Interactivity with the phenotypes (e.g., mouse-over and selection) is a very important part of PheWAS-ME. A limitation of the canvas element is there's no programming interface provided to detect if a component of the chart is interacted with by the user. To solve this, interactivity for the network plot is provided by overlaying an SVG element on top of the Canvas element. The SVG element draws only the phenotype nodes. By rendering this small portion of the plot in SVG, the application can use the built-in intersection observers to detect interaction.

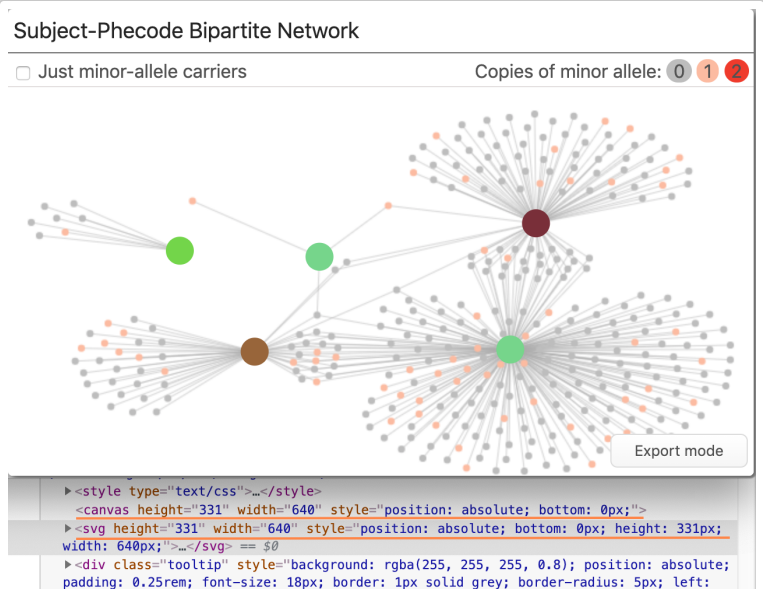

Figure 3.14: Both the Canvas and the SVG are absolutely positioned within the parent to make sure they overlap properly.

Exporting publication-ready plots

One goal of PheWAS-ME is to assist the processing of PheWAS data for publications. To better enable this, the ability to export a high-resolution version of network plot for use in a publication was added. Since canvas elements are just raster-images, they don't scale well when exported. To allow for clean high-res exports, an 'export mode' that switches all rendering to the SVG (which is vector-based) and adds a download button is provided. When in export mode, annotations of present phenotypes is available via a "callout" mode. When a user presses the download button, the SVG element is converted to the SVG file format and downloaded the user's computer. Adobe Illustrator or other vector-based tools can be used to insert the plot into a figure or PDF.

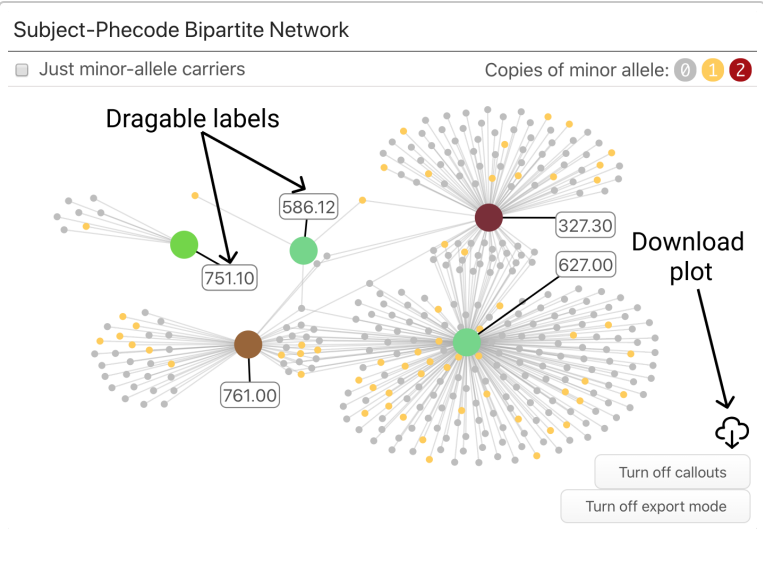

Figure 3.15: An export mode for the network plot allows the export of vector rendered version of current network plot along with optional adjustable labels for phecode ids.

Calculating a force-layout in real time

As the data displayed in the network plot is dependent upon user interaction, the computationally intensive task of generating a force-directed network layout [10] must be calculated on-the-fly as new data is displayed. Javascript uses the same thread for both computation and page rendering. Due to the single-threaded nature of the javascript event-loop, computationally intensive tasks like force-layout simulations can majorly impact the responsiveness of application.

PheWAS-MEs runs force-layout simulations for the network plot in real-time while maintaining a responsive user-experience by offloading the layout calculation to another thread using the web technology web workers[11,13].

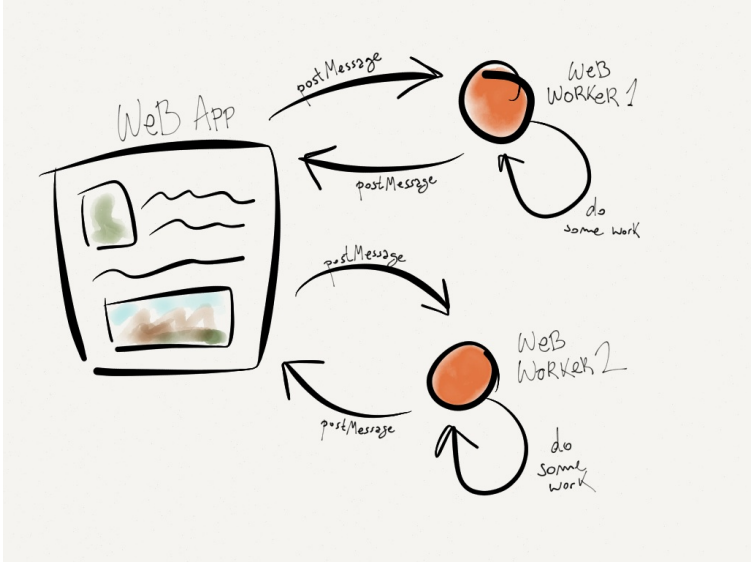

Figure 3.16: Web workers are there own threads with their own event loops, so they don't slow down your main page. Source [11]

Web workers allow javascript code to run on another thread and communicate with the main thread through a series of asynchronous callbacks. PheWAS-ME calculates the layout for the network plot on this separate thread using the network sub-module of D3.js and the following API:

When the network plot receives new data, it spins up one of these web worker threads, sends the data to it to run the layout simulation. The simulation web worker sends back occasional updates on the node positions to the main thread to update the plot, assuring the user it is doing something and not frozen. This adaptation meant that massive networks could be laid out and rendered without causing any perceivable slowdowns on the main app.

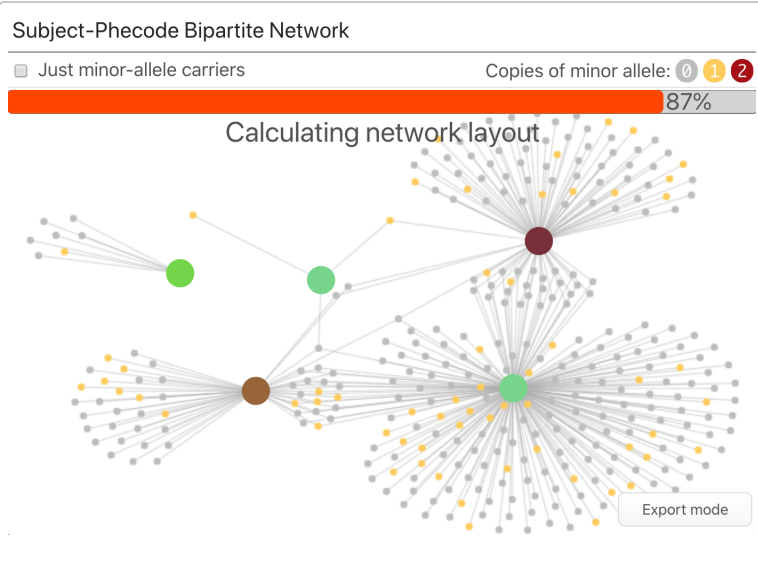

Figure 3.17: As the computationaly intensive layout simulation is run on a seperate thread, the network plot let's the user know the progress of simulation.

3.4.4 Performance

As with any web application performance is not straightforward to report for PheWAS-ME. This is because the performance of the server hosting the Shiny app, the performance of the client viewing the app, and speed of the networks for both the server and client all play important roles in the "speed" of the app.

Some of the optimizations used in constructing the app were discussed in the previous sections, but here we provide a basic anecdotal benchmark of the performance of the app.

3.4.4.1 The setup:

For these tests a PheWAS-ME application was hosted locally using the function `meToolkit::build_me_dashboard` running on the included simulated dataset. A Macbook Pro (2019 16 inch) with a 2.6 GHz 6-Core Intel Core i7 was used to both serve and view the application. The RStudio IDE (v1.3.820) was used to serve the app and Google Chrome (v81.0.4044.138) was used for viewing. All runtime results were obtained via the performance panel in Chrome's developers tools.

3.4.4.2 Results:

Starting the application from page-load to full interaction with all plots, took **2.6 seconds** (with 1.8 of those seconds used for the transfer of data between the server and client.)

Making a large selection of 40 codes (resulting in the transfer of 6,024 individual's data) from the initial load state took **4.18 seconds** (3.82 for network transfer).

Peak memory usage for the 40-code selection was **384 megabytes** for the server and **19 megabytes** for the client.

3.4.4.3 Optimizing the app:

The largest performance bottleneck comes from data transfer between the server and the client. To optimize an application for lower-power machines, a PheWAS-ME administrator can set the maximum allowed selection via the argument `max_allowed_codes` in the application creation function `build_me_dashboard()`. For more details run `?build_me_dashboard`.

3.5 How can I use the app myself?

There are two main methods for using PheWAS-ME.

3.5.1 Hosted version

A version of Phewas-ME is available at https://prod.tbilab.org/phewas_me/. Anyone can use this to load their data and use the application. There is a simulated dataset for exploration along with a data loading screen where new data can be uploaded.

3.5.2 An R package

If the data to be explored, a more secure method of running PheWAS-ME is doing it locally by downloading the R-Package, `meToolkit` [16]. Setting up a local version of the application is as easy as installing the package from GitHub using…

…and then starting an app up with the function `run_me()`.

If you want to learn how to make an app that loads directly to a given dataset or pre-load data, check out the online documentation for the package at https://prod.tbilab.org/phewas_me_manual.

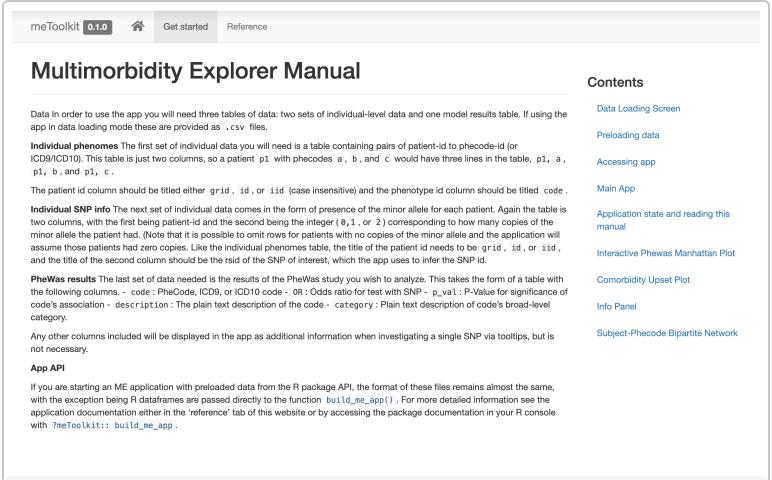

Figure 3.18: meToolkit package website, generated with [17]

References

[1] Dan Abramov and contributors. 2020. Redux: Predictable state container for javascript apps. *GitHub repository*.

[2] Mike Bostock. 2019. D3.js - data-driven documents. Retrieved September 20, 2019 from https://d3js.org/

[3] Winston Chang. 2019. *Modularizing shiny app code*. Retrieved from https://shiny.rstudio.com/articles/modules.html

[4] Winston Chang, Joe Cheng, JJ Allaire, Yihui Xie, and Jonathan McPherson. 2019. *Shiny: Web application framework for r*. Retrieved from https://CRAN.R-project.org/package=shiny

[5] MDN contributors. 2020. *Canvas api*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

[6] MDN contributors. 2020. *SVG: Scalable vector graphics*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/SVG

[7] Joshua C Denny, Lisa Bastarache, and Dan M Roden. 2016. Phenome-wide association studies as a tool to advance precision medicine. *Annual review of genomics and human genetics* 17, (2016), 353–373.

[8] Chrome Developers. 2020. *Lighthouse*. Retrieved from https://developers.google.com/web/tools/lighthouse/

[9] Jon Ferraiolo. 2001. *Scalable vector graphics (SVG) 1.0 specification*. W3C.

[10] Thomas MJ Fruchterman and Edward M Reingold. 1991. Graph drawing by force-directed placement. *Software: Practice and experience* 21, 11 (1991), 1129–1164.

[11] Jaime González García. 2015. *Barbaric basics: Web workers*. Retrieved from https://www.barbarianmeetscoding.com/blog/2015/02/13/barbaric-basics-web-workers

[12] Thomas Greco. 2017. *Becoming familiar with redux*. Retrieved from https://blog.jscrambler.com/becoming-familiar-with-redux/

[13] Ian Hickson. 2015. *Web workers*. W3C.

[14] Javier Luraschi and JJ Allaire. 2018. *R2d3: Interface to 'd3' visualizations*. Retrieved from https://CRAN.R-project.org/package=r2d3

[15] Nick Strayer. 2019. *Taking a network view of ehr and biobank data to find explainable multivariate patterns*. Retrieved from http://nickstrayer.me/biostat_seminar/

[16] Nick Strayer. 2020. *MeToolkit: Build and customize phewas multimorbidity explorer apps*. Retrieved from https://github.com/tbilab/meToolkit

[17] Hadley Wickham and Jay Hesselberth. 2019. *Pkgdown: Make static html documentation for a package*. Retrieved from https://CRAN.R-project.org/package=pkgdown

[18] Wikipedia. 2020. Quadtree, wikipedia, the free encyclopedia.

[19] Tom Wiltzius, Jay Munro, Jatinder Mann, Ian Hickson, and Rik Cabanier. 2015. *HTML canvas 2D context*. W3C.