Title

A Suite of Tools for Biologists That Improve Accessibility and Visualization of Large Systems Genetics Datasets: Applications to the Hybrid Mouse Diversity Panel

Authors

Rau, Christoph D
Civelek, Mete
Pan, Calvin
et al.

Publication Date

2017

DOI

10.1007/978-1-4939-6427-7_7

Peer reviewed

HHS Public Access

Author manuscript

Methods Mol Biol. Author manuscript; available in PMC 2019 July 09.

Published in final edited form as:

Methods Mol Biol. 2017 ; 1488: 153–188. doi:10.1007/978-1-4939-6427-7_7.

A Suite of Tools for Biologists That Improve Accessibility and Visualization of Large Systems Genetics Datasets: Applications to the Hybrid Mouse Diversity Panel

Christoph D. Rau[1], **Mete Civelek**[2], **Calvin Pan**[1], and **Aldons J. Lusis**[3]

[1]Department of Medicine/Division of Cardiology, University of California, Campus – 167917, BH-307 CHS, 10833 Le Conte Ave., Los Angeles, CA, USA.

[2]Center for Public Health Genomics, Department of Biomedical Engineering, University of Virginia, Charlottesville, VA 22908, USA.

[3]Department of Medicine/Division of Cardiology, University of California, Campus – 167917, BH-307 CHS, 10833 Le Conte Ave., Los Angeles, CA, USA.

Abstract

In this chapter we address the recent explosion in large multilevel population studies such as the METSIM study in humans as well as large panels of animal models such as the Hybrid Mouse Diversity Panel or the BXD set of recombinant inbred strains. These studies have harnessed the increasing affordability of large-scale high-throughput profiling to gather massive quantities of data. These datasets, spread across different -omics levels (genome, transcriptome, etc.), different tissues (e.g. heart, plasma, bone) and different environmental factors (e.g. diet, drugs) each individually have led to a number of novel findings relevant to a variety of complex diseases and other phenotypes. The analysis of these results, however, is often limited to individuals with a comprehensive understanding of database languages such as SQL. In this chapter, we describe the development of a GUI-based database analysis suite, using the Hybrid Mouse Diversity Panel as an example to lay out a series of methods for visualization and integration of large systems genetics datasets. The database is based on the Shiny suite of tools in R, and is transferrable to other SQL-based datasets.

Keywords

Analysis tools in systems genetics; GUI-based database analysis suite; Multilevel population studies; Hybrid mouse diversity panel; BxD recombinant inbred strains; METSIM in humans

1 Introduction and Background

Systems genetics deals with the analysis of massive datasets typically gathered by large, multicenter studies. In order to overcome hurdles ranging from the need to amass large sample sizes to performing data analysis to dealing with administrative issues such as

Electronic supplementary material The online version of this chapter (doi: 10.1007/978-1-4939-6427-7_7) contains supplementary material, which is available to authorized users.

informed consent or obtaining funding, these studies often require teams of individuals who, working together, are able to generate data which must subsequently be sifted through to find notable results. As the field has matured, increasingly ambitious studies have been designed and implemented by building off of the framework of other studies. For instance, a recent study, the METabolic Syndrome in Men (METSIM) [1], is collecting multiple -omics levels of data from over 10,000 men and intends to do follow-up studies with these same -omes over the next several decades. Similarly, large panels of inbred animal models, such as the Hybrid Mouse Diversity Panel (HMDP) [2], the BxD set of recombinant inbred strains, or the Collaborative Cross [3] have been constructed which allow for a continually expanding set of diseases and phenotypes to be explored on genetically identical individuals. These animal models are able to avoid the issues of environmental confounders and informed consent, while still providing powerful insights into the underlying mechanisms of complex phenotypes and diseases.

These large studies have had considerable successes in discovering genes and pathways which are implicated in the regulation of phenotypes and disease progression. In the process, they have generated massive datasets which span different tissue types, environmental conditions and -omics levels. The analysis of these large datasets, however, can prove challenging. On the one hand, individuals without a sufficient background in programming may have difficulty navigating the (often SQL) databases in which these data are typically deposited and/or be unable to visualize and interpret these results even if they were able to access them. On the other hand, the scale of the data generated means that a comprehensive analysis is often beyond the capabilities of a few members of a team who do possess the ability to access, visualize and understand these data after they are generated. This chapter lays out the implementation of a suite of point-and-click tools for the visualization and interpretation of large systems genetics datasets, designed to allow researchers who do not deal with computational techniques on a regular basis to access and interpret these data in an intuitive way. The database accessibility suite has been coded using the Shiny R package [4], therefore each tool can be modified to interface with any SQL-based database by a programmer knowledgable about the struture of the database.

2 Methods

2.1 Type of Data and Principle Analysis Tools

2.1.1 The Hybrid Mouse Diversity Panel—The dataset we will be using throughout this chapter is the Hybrid Mouse Diversity Panel, a set of over 150 unique inbred mouse lines. These lines have been extensively studied, and at present the database consists of mice studied under five different environmental conditions or genetic stressors (Table 1) and ten different tissues from which transcriptomes and other -omics studies have been performed. In total, over 300 clinical traits, 40,000 transcripts, 350 metabolites and 3500 protein fragments have been queried in one or more HMDP study (Figs. 1 and S1). Through a variety of systems biology techniques, many candidate genes and pathways [5, 9, 14, 18] have been identified using this panel. While past studies have generally focused on data from an individual experimental model/condition, integration across models can also be fruitful. Because the mice studied under various conditions are identical in terms of their

genetic backgrounds, a variety of combined analyses across multiple tissues or studies are possible; for example, the data can be used for the discovery of novel cross-tissue or cross-conditional relationships.

Below, we describe the suite of accessibility tools we have developed to assist in these sorts of HMDP analyses. Broadly speaking, our database implements two categories of analysis: The Visualization of previously generated data (e.g., the creation of a Manhattan plot) and the Discovery of new relationships between these data (e.g., identifying correlations of genes and phenotypes across multiple tissues/studies).

2.1.2 Overall Design—It was important that the tools we created for accessing the database operated similarly to one another. Each tool (Figs. S2, S3, S4, S5, S6, S7, and S8) asks users for inputs on the left side of the screen using drop-down menus, checkboxes and places for users to input text. As many of our studies examine similar measurements, each drop-down menu is dependent on the selection of the menus above it. For instance, in Fig. S2, the plotted Manhattan plot was generated from clinical trait data from the atherosclerosis study using female mice and the adiposity phenotype. Similarly, in Fig. S3, the displayed beeswarm plot was generated from clinical data from one of the Chow studies, using males from the first chow study and visualizing the effects on HDL of the SNP rs31423553. On the right hand side of each tool is the output, with the figure or table at the top followed by a button to download the results from the database to one's own computer.

2.2 Visualization Tools

2.2.1 Generating a Manhattan Plot—A classic tool for the analysis of GWAS results is the Manhattan plot, which visualizes genome-wide association. The locations of single nucleotide polymorphisms (SNPs) are plotted on the *X*-axis and the strength of their association with the trait of interest as $-\log_{10}$ (*p* value) on the *Y*-axis. The HMDP contains over 400,000 individual quantitative traits. For each trait, the database allows for a Manhattan plot (using the qqman package [22]) to be generated either over the entire genome (Fig. 2a S2a) or in greater detail at an individual chromosome (Fig. S2b) or at an even narrower scope to look at a more localized region. At distances of less than 10 Mb, the UCSC genome browser is linked to the output (Fig. S2c), allowing for direct identification of possible candidate genes. Additionally, at any point the data displayed may be downloaded for later analysis.

2.2.2 Generating a Beeswarm Plot—Whereas a Manhattan plot displays the results of an association study across a range of SNPs, it is often desirable to examine the distribution of individual samples across a single polymorphism. In this case, a "beeswarm" plot is often used. Our database is capable of quickly generating such a plot for any phenotype/SNP pairing (Figs. 2b and S3) through the use of the Beeswarm package [23].

2.2.3 Visualizing Values Across Strains and Tissues—A frequently asked question in animal research is whether a given animal model is the best model to explore a particular phenotype of interest. Since the strains which comprise the HMDP are publically available, one of the benefits of the HMDP as a model is that it can answer these questions

and provide researchers with the ideal mouse strain for further research. By visualizing the phenotype or gene expression value across all or a subset of strains and all or a subset of studies/tissues (Figs. 2c and S4), our database allows for quick analysis and subsequent download of any gene expression or phenotypic value it contains. For example, we can see in Fig. 2c that the gene *Abcc6* is highly expressed in the liver and moderately expressed in the intestine, but weakly expressed in other tissues.

2.2.4 Linkage Disequilibrium—

While linkage disequilibrium (LD) in humans is typically quite small, the LD structure in mouse panels is broader, ranging from 1 Mb to up to 10 Mb in the HMDP. Consequently, identifying candidate genes near significantly associated SNPs involves examining all genes which lie within the LD of the peak SNP, rather than simply examining the one or two nearest genes to the peak SNP as is often done in human studies. Our database allows researchers to either specify an individual SNP, in which case a proposed LD block around the SNP will be provided, or provide a particular window to examine, in which case the LD structure between each pair of SNPs in the window will be displayed (Figs. 2d and S5).

2.3 Discovery Tools

2.3.1 Identifying Nonsynonymous SNPs Within a Gene—

Prioritizing candidate genes at a locus can be difficult, as some genes in a locus can be poorly annotated or described. One common technique used to identify genes with a greater likelihood to be implicated in the phenotype of interest is to examine the gene for nonsynonymous and splice mutations which may act to disrupt the structure and function of a gene at the locus without affecting its expression. Our database makes use of the Wellcome Trust Mouse Genomes Project [24] which contains the full sequences of 18 of the inbred lines of the HMDP and, therefore, the vast majority of all sequence variations within the panel. The output allows researchers to identify which strains have mutations within them, what sort of mutations they are and where in the gene they are located (Figs. 3a and S6).

2.3.2 Identifying Local-eQTLs Within Tissues—

If a candidate gene's physical amino acid sequence is not altered, another means by which a SNP may affect a gene's function is by altering its expression. SNPs residing near a gene whose expression is associated with that SNP are commonly termed local or *cis*-eQTLs. Our database allows users to search across all HDMP studies and tissues to identify *cis*-eQTLs by either probe ID or gene symbol, returning the best *p*-value and location for a SNP within a user-defined window (default 2 Mb) (Figs. 3b and S7). The table can be searched quickly within the GUI or downloaded for more detailed analysis.

2.3.3 Find Correlations Within and Across Studies, Tissues and Conditions

—The HMDP is an expanding resource, where each subsequent study builds on prior research. Finding relationships between genes of interest in one dataset and genes or phenotypes of interest in another, however, can be challenging. We have implemented a searchable unified correlation table, allowing users to examine how genes and phenotypes might correlate to one another across tissues and experimental conditions. For example, we can observe that *Adamts2* expression in the heart is linked to changes in heart weight after

isoproterenol stimulation (Figs. 3c and S8). However, we can also see that *Adamts2* expression in the aorta and intestine are linked to plasma platelet counts, suggesting a role for the gene in clotting or wound repair. It is notable that while both aortic and intestinal expression of *Adamts2* is linked to plasma platelets, they are linked in opposite directions: Higher expression in the intestines in mice with a high-fat diet is linked to lower platelets, while higher expression in the aorta in mice induced to develop atherosclerosis is linked to increased platelet counts. Such inter-study observations can provide additional clues to the roles genes play in physiologic/disease traits.

2.3.4 Identify Overlapping Loci—In addition to exploring how genes and phenotypes are correlated to one another across different tissues and conditions in the HMDP, we can use the large number of associations available in the panel to look for loci which are shared between multiple phenotypes. This could have a number of applications, from the validation of *cis* or *trans*-eQTLs in multiple tissues to exploring how a complex phenotype or set of phenotypes, identified across multiple tissues, may be regulated by a single locus or set of loci. Our database allows researchers to look at and download either all phenotypes which are associated with a SNP at a specific *p*-value, or to examine all phenotypes which have a *p*-value of a given significance within a user-defined window. For example, a locus on chromosome 5 near the gene *Mospd3* has been linked to increased right ventricular weight after isoproterenol stimulation [18]. By examining this same locus across all of our tissues, we can see that we also have links to food intake on a high-fat diet, levels of the IL-1b cytokine, levels of the metabolite hexanoyl-carnitine, and the abundance of several proteins in the liver (Figs. 3d and S9).

2.3.5 Availability—The HMDP database may be accessed at systems.genetics.ucla.edu. The most up-to-date version of the code for the implementation of the database can be found at https://github.com/ChristophRau/HMDPDatabase.

3 Further Considerations and Limitations

Other databases do exist for the analysis of mouse data, the most notable of which is the Mouse Genome Informatics (MGI) database provided by Jackson Labs (informatics.jax.org). There it is possible to download phenotypes and genotypes of a number of strains from a variety of different studies as well as visualize the results of each individual study in terms of the phenotypes observed as well as information on a gene of interest. One strength of the HMDP over the studies typically found at this database is the scale of the study and the comparability across studies. While most HMDP studies involve over 100 strains of mice, studies in the MGI repository are typically much smaller. At the same time, few studies involve the same strain of mice as others, making it difficult for researchers to compare results between individual studies. Additionally, few studies involve the study of multiple -omics layers, and differences in housing, diet and other environmental factors make it difficult to directly compare studies to one another. Moreover, the tools made available in the MGI database to query these studies are generally designed for the smaller, less systems-wide studies that make up the bulk of the repository.

It is always possible to perform more nuanced and complicated analyses of the HDMP database using direct SQL queries. While results gathered by these sorts of analyses may be stronger and more meaningful than the results obtainable by our analysis suite, it requires the ability to navigate the SQL databases directly. As mentioned in the introduction, we believe that an ultimately more fruitful approach is to open up access to such databases to anyone with a grasp of the concepts involved, but perhaps not the technical skill to access the data directly.

4 Outlook

The HMDP is a constantly evolving tool with ongoing studies in several laboratories and an expanding database. Our graphical interface is designed to be able to access any data which is added to the database without the need to manually update a number of tables. We plan to continue to add modules to the database, for instance allowing users to select a series of phenotypes, genes, metabolites and proteins and receive in return graphical and tabular outputs of SNPs on the genome which are associated with one or more of these inputs, a tool which will complement our currently implemented ability to study a given genomic location for association overlaps. Additionally, we plan to improve the relationships of the modules to one another, allowing a user to start in one module, get a result, click on that result and then be taken to another module for additional analyses. Finally, we plan to expand our results to interface with some of the gene-centric databases which currently exist (e.g. NCBI Gene), to allow researchers to seamlessly travel from phenotype to locus to gene. The code for each of these updates will be made available and like the rest of the interface, will be designed to be easily modifiable to interface with other researchers' SQL databases.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgments

The HMDP database was developed and is currently maintained through support from NIH grants HL30568, HL28481.

5: Appendix: Code for Database Algorithm v0.7

The Shiny package, developed by Rstudio allows users to implement a graphical user interface with an R-based backend. A GUI created with Shiny requires two R scripts to properly function. The first, Server.R, is the part of the code which actually performs the various analyses, interfaces with the SQL database and does all the things that a standard R script would do. The second, ui.R, controls the layout and structure of the GUI itself and is responsible for sending inputs to and receiving outputs from Server.R. Updates to this code may be found on GitHub at: https://github.com/ChristophRau/HMDPDatabase.

6 Server.R

```
# A Graphical User Interface for querying a genetics SQL Database using
```

```
Shiny in R
# Version: 0.7
# Last Modified: 12/9/15
#
# The following is an implementation of a GUI using the Shiny package in
Rstudio. Shiny programs have two scripts associated with them. This one,
server.R acts as the “brains”
# of the code and contains all of the functions which actually compute
results. The other script, ui.R, controls the appearance of the GUI and
provides inputs and displays outputs from
# Server.R
#####Startup Stuff#####
options(shiny.maxRequestSize = 50*1024^2)
options(stringsAsFactors=FALSE)
#Scripts and packages required for operation
source(“manhattan.R”)
if (!require(“beeswarm”)) install.packages(“beeswarm”)
if (!require(“gplots”)) install.packages(“gplots”)
library(beeswarm)
library(“gplots”)
#Initialize the database reader and get a list of all relevant tables
print(“Initializing Database”)
library(RODBC)
dbhandle <- odbcDriverConnect(‘driver={SQL
Server};server=JLUSISDB;database=HM
DP;trusted_connection=true’)
FullTables=sqlTables(dbhandle)
#A section to define limited Tables if you want to password protect the full
data.
limitedTables=FullTables #no password protection
limitedTables=FullTables[grep(“Chow”, FullTables[,3]),] #limit to a subset
of data
allTables = limitedTables
#The following section goes through all of the tables in the database and
creates a master list which maps gene symbols to probe IDs. This takes some
time and it is often
# easier to generate this file separately and read it in with read.csv
#####Create the Master table of Gene names and IDs#####
print(“Creating Genes Table... this could take up to 5 minutes…”)
# AFFY=“Affymetrix_HT_MG-430A_v33”
# ILMN=“Illumina_MouseRef8_v2_R3”
#
# #Load Affy Data
#
```

```
# query <- paste("Select gene_symbol, probesetID from [TranscriptAnnotation].
["",AFFY,"]",sep="")
# temp=sqlQuery(dbhandle, query)
#
# #Load Illumina Data
#
# query <- paste("Select Symbol, probesetID from [TranscriptAnnotation].
["",ILMN,"]",sep="")
# temp2=sqlQuery(dbhandle, query)
# colnames(temp)=c("Symbol", "probesetID")
# all_genes=rbind(temp,temp2)
# all_genes[,1]=toupper(all_genes[,1])
#
#
# #Creating the Search_List. This can take a moment...
# geneSearchList<-do.call(rbind,
# by(all_genes,all_genes$Symbol, function(x)
# with (x,
# data.frame(
# Symbol=unique(Symbol),
# probeIDs=paste(probesetID,collapse=",")
# )
# )
# )
# )
# geneSearchList=as.matrix(geneSearchList)
# geneSearchList=paste(geneSearchList[,1],geneSearchList[,2],sep=",")
#or just read it in.
geneSearchList=as.matrix(read.delim(file="GeneSearchList.csv"))
allStrains=as.matrix(read.csv(file="Strains.csv")) #reads in all the strains
(individuals) used in the study
#####At this point the server is initialized and ready to launch#####
print("Launching!")
shinyServer(function(input, output,session) { #basic implementation of a
shinyServer
#####Suggestions#####
#This first part of the code implements a simple suggestion .csv for
recording bugs and/or suggestions
Suggest <- reactive({
outfile=file("Suggestions.csv","a")
name=input$Suggestion_Name
value=input$Suggestion_Report
outrow=paste(name,value,sep=",")
cat(outrow,file=outfile,sep="\n")
```

```
close(outfile)
})
output$Suggestion_Text <- renderText({
if(input$Suggestion_Button==0){
return (NULL)} else{
isolate(Suggest())
val="Thanks!"
return(val)
}
})
#####Login#####
#A way to implement some degree of password protection on your data, say if
you have some public and some private information.
Password="Password"
Login <- reactive({
if(input$Password_Go==0)
{return("")} else {
isolate({
if(input$Password == Password){
allTables <<- FullTables
return("Login Successful")
} else {
return("Login Failed")
}
})
}
})
output$PassOK <- renderText({
LogVal=Login()
print(LogVal)})
#####ProbeID_Lookup#####
#This section allows for rapid conversion of Probe_IDs to Gene Symbols and
vice versa. Works in batch mode.
Lookup <- reactive({
inFile <- input$Lookup_Batch
if(is.null(inFile)){ #If no batch file uploaded
if(is.null(input$Lookup_One)){ return(NULL)} else{ #if nothing entered,
return nothing
val=input$Lookup_One
print(val)
entry=geneSearchList[grep(paste0(val,"(,|$)"),geneSearchList, ignore.
case= TRUE)] #find gene in master table
print(entry)
entry=strsplit(entry,",")[[1]]
```

```
print(entry)
out=c()
for(i in 2:length(entry)){
out=rbind(out,c(entry[1],entry[i])) #generate output for gene
}
colnames(out)=c("Gene Name","Probe ID")
print(out)
return(out)
} else { #If batch file uploaded
vals=read.csv(inFile$datapath,header=F) #read the file
out=c()
for(i in 1:nrow(vals)){ #for each gene
val=vals[i,1]
entry=geneSearchList[grep(paste0(val,"(,|$)"),geneSearchList, ignore.
case= TRUE)] #find in master table
entry=strsplit(entry,",")[[1]]
for(j in 2:length(entry)){ #append results to output
out=rbind(out,c(entry[1],entry[j]))
}
}
colnames(out)=c("Gene Name","Probe ID")
return(out)
}
})
output$Lookup_Table <- renderDataTable({ #This tells the ui how to output
the data.
if(input$Lookup_Button==0){
return (NULL)} else{
isolate(Lookup())
}
})
#####Data Visualization Section####
#This section creates Manhattan Plots for any study in the database
#This function populates the possible studies to be drawn from based on the
type of data being mapped
output$DataViz_StudyUI<- renderUI({
if(input$DataViz_DataType=="Clinical"){
DV_StudyChoices=allTables[allTables[,2]=="ClinicalTraitAnnotation",3]
}
if(input$DataViz_DataType=="Expression"){
DV_temp=allTables[allTables[,4]=="TABLE",]
DV_StudyChoices=DV_temp[DV_temp[,2]=="TranscriptAbundance",3]
}
if(input$DataViz_DataType=="Metabolite"){
```

```
DV_StudyChoices=allTables[allTables[,2]==“MetaboliteAnnotation”,3]
}
if(input$DataViz_DataType==“Protein”){
DV_StudyChoices=allTables[allTables[,2]==“ProteinAnnotation”,3]
}
selectInput(“DataViz_Study”, “Select Study”, DV_StudyChoices ) #the select
input to be placed into the UI
})
#After a study has been selected, this function populates the possible
phenotypes to select from output$DataViz_PhenotypeUI <- renderUI({
cur_table=input$DataViz_Study #Get which study is being examined
if(input$DataViz_DataType==“Clinical”){
query=paste(“SELECT distinct trait_name FROM HMDP.ClinicalTraitAnnotati
on.”,input$DataViz_Study,””,paste=””)
DV_PhenoChoices=as.vector(sqlQuery(dbhandle, query)[[1]])
}
if(input$DataViz_DataType==“Expression”){ #My original idea was to do a drop-
down menu for all phenotypes, including genes/probes. This proved too taxing
and instead I’ve implemented a simple text entry box.
# The code for the drop-down menu is below.
# query=paste(“Select Top 1 HMDP.TranscriptAbundance.”,input$DataViz_
Study,”.* FROM HMDP.TranscriptAbundance.”,input$DataViz_Study,sep=””)
# DV_TempExpressionQuery=sqlQuery(dbhandle,query)
# query=paste(“SELECT distinct probesetID “,colnames(DV_TempExpressionQuery)
[2],” FROM HMDP.TranscriptAbundance.”,input$DataViz_Study,
# “ WHERE “,colnames(DV_TempExpressionQuery)
[2],”=‘“,DV_TempExpressionQuery[2][[1]],”‘“,sep=””)
# DV_PhenoChoices=as.vector(sqlQuery(dbhandle, query)[[1]])
}
if(input$DataViz_DataType==“Metabolite”){
query=paste(“SELECT distinct metabolite_name FROM HMDP.MetaboliteAnnota
tion.”,input$DataViz_Study,””,paste=””)
DV_PhenoChoices=as.vector(sqlQuery(dbhandle, query)[[1]])
}
if(input$DataViz_DataType==“Protein”){
query=paste(“SELECT distinct gene_symbol FROM HMDP.ProteinAnnotation.”, input
$DataViz_Study,””,paste=””)
DV_PhenoChoices=as.vector(sqlQuery(dbhandle, query)[[1]])
}
if(input$DataViz_DataType==“Expression”){ #Text entry for expression
textInput(“DataViz_Pheno”, “Please Enter your probesetID”)
} else{ # Select Input for phenotype
selectInput(“DataViz_Pheno”, “Select Phenotype”, DV_PhenoChoices )}
})
```

```
#This function differentiates between sub-studies (for instance, male vs
female mice)
output$DataViz_FinalTableSelectUI <-renderUI({
if(input$DataViz_DataType==“Clinical”){
DV_temp=allTables[allTables[,4]==“VIEW”,]
DV_FinalTableChoices=DV_temp[DV_temp[,2]==“ClinicalQTL”,3]
}
if(input$DataViz_DataType==“Expression”){
DV_temp=allTables[allTables[,4]==“VIEW”,]
DV_FinalTableChoices=DV_temp[DV_temp[,2]==“expressionQTL”,3]
}
if(input$DataViz_DataType==“Metabolite”){
DV_temp=allTables[allTables[,4]==“VIEW”,]
DV_FinalTableChoices=DV_temp[DV_temp[,2]==“MetaboliteQTL”,3]
}
if(input$DataViz_DataType==“Protein”){
DV_temp=allTables[allTables[,4]==“VIEW”,]
DV_FinalTableChoices=DV_temp[DV_temp[,2]==“ProteinQTL”,3]
}
DV_FinalTableContenders=DV_FinalTableChoices[grep(input
$DataViz_Study,DV_FinalTableChoices)]
selectInput(“DataViz_ExactView”, “Select Table”, DV_FinalTableContenders )
})
#Finally, now that we have pinpointed the exact phenotype/study combination
desired, we get the data from the server
DV_GetData <- reactive({ #A reactive function only triggers if a variable
within it changes (in this case, if the button ‘DataViz_Calulate’ is
pressed) if(input$DataViz_Calculate==0)
{return(“NULL”)} else {
isolate({ #Nothing within the isolate function “counts” for the reactive
function above. This allows the user to modify what they are looking for
without constantly telling the program to start interacting with the database
if(input$DataViz_DataType==“Clinical”){
Group=“ClinicalQTL”
}
if(input$DataViz_DataType==“Expression”){
Group=“expressionQTL”
}
if(input$DataViz_DataType==“Metabolite”){
Group=“MetaboliteQTL”
}
if(input$DataViz_DataType==“Protein”){
Group=“ProteinQTL”
}
```

```
#We are now going to construct the query to the SQL server.
query <- paste("SELECT trait_name,rsID,snp_chr,snp_bp_mm10,pvalue FROM
HMDP.",Group,".",input$DataViz_ExactView," WHERE trait_name='",input
$DataViz_Pheno,"'",sep="")
print("Query Constructed")
#and here we actually run the query
DV_Data=sqlQuery(dbhandle, query)
})
}
})
#This function makes the Manhattan Plot. It is separate from the above
function to allow users to modify their Manhattan plot parameters (eg look
at specific chromosomes)
#without having to re-download the data
DV_MakeManhattan <- reactive({
withProgress(message="Constructing Query…",value=0,{ #withProgress allows
for the creation of a progress bar in the GUI. In this case, its reporting
that the query is being constructed
DV_Data=DV_GetData()}) #and then getting the data
print("Data Aquired")
#print(dim(DV_Data))
if(DV_Data[1]!="NULL"){ #If there is data…
print("Running")
withProgress(value=.5, message="Processing Results…",{ #another update on
the current progress (50% complete)
subset=DV_Data[,c(3,4,5)] #get relevant values from the output (chromosome,
position, p-value)
if(input$DataViz_Chromosome!="All"){ #if we are NOT looking at all
chromosomes, we need to filter our data
subset=subset[subset[,1]==input$DataViz_Chromosome,] #Limit to just the
chromosome of interest
positions=as.numeric(subset[,2])
#And Limit to the region on the chromosome of interest
subset=subset[positions>as.numeric(input$DataViz_Lower_Bound)*1000000,]
positions=as.numeric(subset[,2])
subset=subset[positions<as.numeric(input$DataViz_Upper_Bound)*1000000,]
}
#Tweak the X and Y chromosome to Chromosomes '20' and '21' respectively
levels(subset[,1])[levels(subset[,1])=="X"]="20"
levels(subset[,1])[levels(subset[,1])=="Y"]="21"
subset=apply(subset,2,as.numeric)
colnames(subset)=c("CHR","BP","P")
subset=as.data.frame(subset)
#At this point we have our final data table.
```

```
})
withProgress(value=.9,message="Constructing Plot...",{ #Finally, construct
the plot
print("Constructing Plot")
DV_Button_Value=DV_Button_Value+1
#This function is provided in Manhattan.R from http://
GettingGeneticsDone.blogspot.com. In this case, we are alternating color on
each chromosome and have set the genome-wide
#singificance like to 10^-5.387, which is the accepted significance line of
the HMDP. manhattan(subset, colors=c("black","#666666","#CC6600"),
main=DV_Data[1,1], pch=20, genomewideline=5.387, suggestiveline=F)
})
}
})
#This function is a simple wrapper which takes the output of the above
functions (which are reactive and cannot be directly interfaced with the UI)
and makes a continually updating plot for the UI
DV_Button_Value=0
output$DataViz_Manhattan <-renderPlot({
DV_MakeManhattan()
})
#This function takes the GWAS results for the phenotype of interest and
packages it for download for later off-line analysis.
output$DataViz_Download<-downloadHandler(
filename = "results.gwas", #the default filename
content = function(file) {
write.table(DV_GetData(), file,row.names=F,sep="\t")
})
#Finally, this function displays the UCSC genome browser in a frame for
regions of less than 10MB in length
output$DV_GenomeBrowser <-renderUI({
if(input$DataViz_Chromosome=="All" || input$DataViz_Upper_Bound-input
$DataViz_Lower_Bound>10 ) #If we are looking at more than 10 MB of genome
{return(tags$iframe(src="",seamless=T,height=800,width="100%"))} else
{ #Return an empty frame. Otherwise…
options(scipen=999) #turn off scientific notation
#This creates a link to the Mouse UCSC Genome broser for the chromosome and
region of interest. It will need to be changed for other species, of course.
temp=paste("http://genome.ucsc.edu/cgi-bin/hgTracks?hgHubConnect.destUrl",
"=..%2Fcgi-bin%2FhgTracks&clade=mammal&org=Mouse&db=mm10&position=chr",
input$DataViz_Chromosome,"%3A",input$DataViz_Lower_Bound*1000000,
"-",input$DataViz_Upper_Bound*1000000,
"&hgt.positionInput=enter+position%2C+gene+symbol+or+search
+terms&knownGene=pack",
```

```
“&ensGene=hide&xenoRefGene=hide&refGene=hide&ucscRetroAli6=hi de&mrna=hide”,
“&intronEst=hide&snp138Common=dense&rmsk=dense&Submit=submit”,sep=“”)
options(scipen=0) #Turn back on Scientific Notation
tags$iframe(src=temp,seamless=T,height=800,width=“100%”) #create a frame
which displays the link above
}
})
#The following section allows users to create a ‘beeswarm’ plot to examine
the effect of a single SNP on a phenotype
#####Beeswarm Section#####
#these first sections are identical to those for the Manhattan Plots above,
but due to the weirdness of Shiny, are simply repeated here.
output$Beeswarm_StudyUI<- renderUI({
if(input$Beeswarm_DataType==“Clinical”){
BS_StudyChoices=allTables[allTables[,2]==“ClinicalTraitAnnotation”,3]
}
if(input$Beeswarm_DataType==“Expression”){
BS_temp=allTables[allTables[,4]==“TABLE”,]
BS_StudyChoices=BS_temp[BS_temp[,2]==“TranscriptAbundance”,3]
}
if(input$Beeswarm_DataType==“Metabolite”){
BS_StudyChoices=allTables[allTables[,2]==“MetaboliteAnnotation”,3]
}
if(input$Beeswarm_DataType==“Protein”){
BS_StudyChoices=allTables[allTables[,2]==“ProteinAnnotation”,3]
}
selectInput(“Beeswarm_Study”, “Select Study”, BS_StudyChoices )
})
output$Beeswarm_FinalTableSelectUI <-renderUI({
if(input$Beeswarm_DataType==“Clinical”){
BS_temp=allTables[allTables[,4]==“VIEW”,]
BS_FinalTableChoices=BS_temp[BS_temp[,2]==“ClinicalTraits”,3]
}
if(input$Beeswarm_DataType==“Expression”){
BS_temp=allTables[allTables[,4]==“VIEW”,]
BS_FinalTableChoices=BS_temp[BS_temp[,2]==“TranscriptAbundance”,3]
}
if(input$Beeswarm_DataType==“Metabolite”){
BS_temp=allTables[allTables[,4]==“VIEW”,]
BS_FinalTableChoices=BS_temp[BS_temp[,2]==“MetaboliteAbundance”,3]
}
if(input$Beeswarm_DataType==“Protein”){
BS_temp=allTables[allTables[,4]==“VIEW”,]
BS_FinalTableChoices=BS_temp[BS_temp[,2]==“ProteinQTL”,3]
```

```
}
# print(input$Beeswarm_Study)
BS_FinalTableContenders=BS_FinalTableChoices[grep(input
$Beeswarm_Study,BS_FinalTableChoices)]
# print("Done")
selectInput("Beeswarm_ExactView", "Select Table", BS_FinalTableContenders )
})
output$Beeswarm_PhenotypeUI <- renderUI({
cur_table=input$Beeswarm_Study
if(input$Beeswarm_DataType=="Clinical"){
query=paste("SELECT distinct trait_name FROM HMDP.ClinicalTraitAnnotati
on.",input$Beeswarm_Study,"",paste="")
DV_PhenoChoices=as.vector(sqlQuery(dbhandle, query)[[1]])
}
if(input$Beeswarm_DataType=="Expression"){ #once again, a drop-down menu
isn't practical here.
#query=paste("Select probesetID FROM HMDP.TranscriptAbundance.",input$Be
eswarm_ExactView,sep="")
#DV_TempExpressionQuery=sqlQuery(dbhandle,query)
#DV_PhenoChoices=as.vector(sqlQuery(dbhandle, query)[[1]])
}
if(input$Beeswarm_DataType=="Metabolite"){
query=paste("SELECT distinct metabolite_name FROM HMDP.MetaboliteAnnota
tion.",input$Beeswarm_Study,"",paste="")
DV_PhenoChoices=as.vector(sqlQuery(dbhandle, query)[[1]])
}
if(input$Beeswarm_DataType=="Protein"){
query=paste("SELECT distinct gene_symbol FROM HMDP.ProteinAnnotation.", input
$Beeswarm_Study,"",paste="")
DV_PhenoChoices=as.vector(sqlQuery(dbhandle, query)[[1]])
}
if(input$Beeswarm_DataType=="Expression"){
textInput("Beeswarm_Pheno", "Please Enter your probesetID")
} else{
selectInput("Beeswarm_Pheno", "Select Phenotype", DV_PhenoChoices )}
})
#With all the inputs set, this actually creates the output once the button
'Beeswarm_Calculate' is pressed.
BS_GetData <- reactive({
if(input$Beeswarm_Calculate==0)
{return("NULL")} else {
isolate({
if(input$Beeswarm_DataType=="Clinical"){
Group="ClinicalTraits"
```

```
}
if(input$Beeswarm_DataType==“Expression”){
Group=“TranscriptAbundance”
}
if(input$Beeswarm_DataType==“Metabolite”){
Group=“MetaboliteAbundance”
}
if(input$Beeswarm_DataType==“Protein”){
Group=“ProteinAbundance”
}
if(input$Beeswarm_DataType==“Expression”){
#Expression Data is a little bit different from the other datatypes in terms
of how it is stored, so it is collected in its own conditonal phrase.
query<- paste(“SELECT * FROM HMDP.”,Group,”.”,input$Beeswarm_ExactView,”
WHERE probesetID=‘“,input$Beeswarm_Pheno,”‘“,sep=““)
print(“Query Constructed”)
print(query)
#Get Expression Data
Pheno_Data=sqlQuery(dbhandle, query)
query <- paste(“SELECT * FROM
HMDP.genotypes.MouseDivArray_genotype_calls_emma_format WHERE rsID=‘“,input
$Beeswarm_rsID,”‘“, sep=““)
print(“Query Constructed”)
#Get SNP Info
Geno_Data=sqlQuery(dbhandle,query)
Mapping=match(colnames(Pheno_Data),colnames(Geno_Data))
temp=is.na(Mapping)
Mapping[temp]=1
BS_Data=cbind(t(Geno_Data[,Mapping]),t(as.matrix(Pheno_Data)))
}
else {
#The Althersclerosis data is slightly different from the other data, so this
corrects for a difference in column names. (Strain vs Maternal_strain)
if(length(grep(“Atherosclerosis”,input$Beeswarm_ExactView))>0)
{query <- paste(“SELECT Maternal_strain, \““,input$Beeswarm_Pheno,”\” FROM
HMDP.”,Group,”.”,input$Beeswarm_ExactView,sep=““)}
else{query <- paste(“SELECT Strain, \““,input$Beeswarm_Pheno,”\”
FROM HMDP.”,Group,”.”,input$Beeswarm_ExactView,sep=““)}
print(“Query Constructed”)
#Run the query and get the Phenotype
Pheno_Data=sqlQuery(dbhandle, query)
#get the SNP data at that particular RSID
query <- paste(“SELECT * FROM
HMDP.genotypes.MouseDivArray_genotype_calls_emma_format WHERE rsID=‘“,input
```

```
$Beeswarm_rsID,"'", sep="")
print("Query Constructed")
Geno_Data=sqlQuery(dbhandle,query)
Mapping=match(Pheno_Data[,1],colnames(Geno_Data))
temp=is.na(Mapping)
Mapping[temp]=1
BS_Data=cbind(t(Geno_Data[,Mapping]),as.numeric(as.matrix(Pheno_Data)[,2]))
}
})
}
})
#This section actually creates the Beeswarm Plot from the data gathered above
output$BS_Plot <- renderPlot({
if(input$Beeswarm_Calculate==0){
return ("NULL")} else{
isolate({ #Nothing after this matters for updating, so it will only update
if the button is pressed.
BS_Data=BS_GetData()
print("Data Aquired")
if(BS_Data[1]!="NULL"){
print("Running")
#Remove missing values…
temp0=!is.na(BS_Data[,1])
BS_Data=BS_Data[temp0,]
#Identify which strains are WT or SNP for that rsID
temp1=BS_Data[,1]==0
temp2=BS_Data[,1]==1
BS_Data=rbind(BS_Data[temp1,],BS_Data[temp2,])
BS_Data=apply(BS_Data,2,as.numeric)
#calculate a p-value for that rsID and phenotype
p_val=t.test(as.numeric(BS_Data[BS_Data[,
1]==0,2]),as.numeric(BS_Data[BS_Data[,1]==1,2]))$p.value
#Create a simple boxplot of data
boxplot(BS_Data[,2]~BS_Data[,1],xlab=input$Beeswarm_rsID,ylab=input
$Beeswarm_Pheno,main="Genotype x Phenotype",outline=FALSE)
#overlay it with the beeswarm
beeswarm(BS_Data[,2]~BS_Data[,1],col = 4, pch = 16,add=TRUE)
}
})
}
})
#The following section identifies the most significant eQTL within a
specific user-defined window of the gene in question. Mostly useful for
answering 'does this gene have a cis-eQTL?'
```

```
#####ciseQTL section#####
#While this function isn't specifically needed (the outputed UI could have
just been included in the main UI script, it is here so it can be expanded
if needed) output$ciseQTL_PhenotypeUI <- renderUI({
# Needs to be probesetID
textInput("ciseQTL_Pheno", "Please Enter your probesetID")
})
#Once again, a workhorse function to actually get the data only when the
button is pushed.
CE_GetData <- reactive({
if(input$ciseQTL_Calculate==0)
{return("NULL")} else {
isolate({
Group="expressionQTL"
Gene=input$ciseQTL_Pheno
#form the SQL Query. This first one simply gets the information about the
gene, namely its location on the genome
query <-paste("SELECT TOP 1 [dataset],[probesetID],[gene_chr],
[gene_start_bp],[gene_end_bp],[snp_chr],[snp_bp_mm10],[pvalue] from
HMDP.Unified.QTL_AllInfo WHERE probesetID='",Gene,"'",sep="")
Pheno_Data=sqlQuery(dbhandle,query)
Gene_chr=Pheno_Data[,3]
SNP_Lower=Pheno_Data[,4]-(input$ciseQTL_Window*1000000)
SNP_Upper=Pheno_Data[,5]+(input$ciseQTL_Window*1000000)
#This next query grabs all SNPs for that particular gene within that window
from all studies
query <-paste("SELECT [dataset],[probesetID],[snp_chr],[snp_bp_mm10],
[pvalue] from HMDP.Unified.QTL_AllInfo WHERE probesetID='",Gene,"' and
snp_chr='",Gene_chr,
"' and snp_bp_mm10>",SNP_Lower," and snp_bp_mm10<",SNP_Upper,sep="")
#A progress bar to get the data
withProgress(value=0,message="Getting Data",{
Pheno_Data=sqlQuery(dbhandle,query)
})
#Now that we have the data, we need to find the best eQTL for each study
withProgress(value=.2,message="Generating Table",{
table_names=names(table(Pheno_Data[,1])) #get study names
output=c()
temp_count=0
for(i in table_names){ #for each study
temp_count=temp_count+1
incProgress(amount=temp_count/length(table_names)*.8) #slowly increment the
% complete from 0% to 80%
temp=grep(i,Pheno_Data[,1])
```

```
temp_data=Pheno_Data[temp,c(1,3,4,5)] #extract study from master table
temp=which.min(as.numeric(temp_data[,4])) #find minimum value
output=rbind(output,temp_data[temp,]) #add minimum row to total output
}
rownames(output)=NULL
})
output #return output.
})
}
})
#Creates the actual output table for the cis_eQTL data
output$ciseQTL_Table <- renderDataTable({
if(input$ciseQTL_Calculate==0){
return (NULL)} else{
isolate(CE_GetData())
}
})
#Creates a downloadable table with the cis_eQTL data
output$ciseQTL_Download<-downloadHandler(
filename = “results.txt”,
content = function(file) {
write.table(isolate(CE_GetData()), file,row.names=F,sep=“\t”)
})
#This section looks for all loci (for different phenotypes/tissues/studies)
which overlap a particular region. Useful for identifying relationships
between different phenotypes
#####Overlapping Loci section#####
#There are two means of looking for overlap which are supported. The first
is to provide a genomic interval of interest, the second to provide a
specific rsID.
#The following two lines are only rendered in the UI if it is in “window”
format.
output$Overlap_chr <-renderUI({if(input$Overlap_window_or_rsID==“window”)
{selectInput(“Overlap_Chr”,”Select Chromosome”,c(1:19,”X”))}})
output$Overlap_LB <-renderUI({if(input$Overlap_window_or_rsID==“window”)
{numericInput(“Overlap_LB”,”Lower Bound (in MB)”,value=10,min=0)}})
#This last one is *either* the upper bound OR the rsID.
output$Overlap_additional <-renderUI({
if(input$Overlap_window_or_rsID==“window”){
numericInput(“Overlap_UB”,”Upper Bound (in MB)”,value=15,min=0)
} else {
textInput(“Overlap_rsID”,”Please enter a SNP rsID”)
}
})
```

```
#The Workhorse function for this section
OL_GetData <- reactive({
if(input$Overlap_Calculate==0)
{return("NULL")} else { #if the button is pushed…
isolate({
print("Generating Overlap Table, Please Wait…")
if(input$Overlap_window_or_rsID=="window"){ #If we are in Window format
withProgress(value=0,message="Obtaining Results…",{
#There is an option to either include eQTLs or exclude them. It is
significantly faster to NOT include eQTLs.
if(input$Overlap_includeGenes){ #if we are including eQTLs, we draw from the
entire unified QTL table.
query <- paste("Select [dataset],[probesetID],[gene_symbol],[rsID],[snp_chr],
[snp_bp_mm10],[LD_block_start_mm10],",
"[LD_block_end_mm10],[pvalue] from Unified.QTL_AllInfo WHERE snp_chr='",input
$Overlap_Chr,"' and snp_bp_mm10>",
input$Overlap_LB*1000000," and snp_bp_mm10<",input$Overlap_UB*1000000," and
pvalue<",input$Overlap_threshold,sep="")
Pheno_Data=sqlQuery(dbhandle, query)
} else { #if we are excluding eQTLs, we filter out the eQTLs as part of the
query
query <- paste("Select [dataset],[category],[probesetID],[gene_symbol],
[rsID],[snp_chr],[snp_bp_mm10],[LD_block_start_mm10],",
"[LD_block_end_mm10],[pvalue] from Unified.QTL_AllInfo WHERE snp_chr='",input
$Overlap_Chr,"' and snp_bp_mm10>",
input$Overlap_LB*1000000," and snp_bp_mm10<",input$Overlap_UB*1000000," and
pvalue<",input$Overlap_threshold, " and category!='expression QTL'",sep="")
Pheno_Data=sqlQuery(dbhandle, query)
Pheno_Data=Pheno_Data[,-2] #To make the rest of the code work, we remove the
"category" column from the data.
}
print("Data Aquired, Analyzing. Please Wait...")
all_names=apply(Pheno_Data[,c(1:2)],1,paste,collapse="") #We wish to find
all of the unique peaks in the returned data
unique_names=names(table(all_names))
})
print(paste("There are ",length(unique_names)," peaks!",sep=""))
#We now process the results to find the minimum p-value for each phenotype
peak.
output=c()
withProgress(value=0,message="Processing Peaks...",{
for(i in 1:length(unique_names)){
# print(i)
if(i%%50==0){incProgress(50/length(unique_names),detail=paste(i," Peaks
```

```
Processed",sep=""))}
cur_name=unique_names[i] #for the ith unique name
temp=grep(cur_name,all_names) #find those names in the main output (non-
unique)
temp_array=Pheno_Data[temp,] #form a subset from just those rows
temp=which.min(as.numeric(temp_array[,9])) #find the minimum pvalue
winner=temp_array[temp,] #declare THAT particular SNP the 'winner'
winner=winner[,c(1:4,9)] #extract the important information (study, probeID/
identifier, gene symbol, rsID, p value)
output=rbind(output,winner) #add the winner to the final output
}
})
} else { #if we are looking at a specific rsID
withProgress(value=0,message="Obtaining Results…",{
#We first find the precomputed linkage disequilibrium block for that SNP
query <- paste("Select top 1 [dataset],[probesetID],[gene_symbol],[rsID],
[snp_chr],[snp_bp_mm10],[LD_block_start_mm10],",
"[LD_block_end_mm10],[pvalue] from Unified.QTL_AllInfo WHERE rsID='",input
$Overlap_rsID,"'",sep="")
#print(query)
Pheno_Data=sqlQuery(dbhandle, query)
print(Pheno_Data)
incProgress(.5,detail="part 2")
snp_chr=Pheno_Data[5] #chr of LD block
lower=Pheno_Data[7] #lower bound of LD block
upper=Pheno_Data[8] #upper bound of LD block
#And now we basically do the same thing as when we were doing the window.
query <- paste("Select [dataset],[category],[probesetID],[gene_symbol],
[rsID],[snp_chr],[snp_bp_mm10],[LD_block_start_mm10],",
"[LD_block_end_mm10],[pvalue] from Unified.QTL_AllInfo WHERE
snp_chr='",snp_chr,"' and snp_bp_mm10>",
lower," and snp_bp_mm10<",upper," and pvalue<",input
$Overlap_threshold,sep="")
#print(query)
Pheno_Data=sqlQuery(dbhandle, query)
})
if(input$Overlap_includeGenes){
Pheno_Data=Pheno_Data[,-2]
} else {
tokeep=Pheno_Data[,2]!="expression QTL"
Pheno_Data=Pheno_Data[tokeep,]
Pheno_Data=Pheno_Data[,-2]
}
print("Data Aquired, Analyzing. Please Wait...")
```

```
withProgress(value=0,message="Processing Peaks...",{
all_names=apply(Pheno_Data[,c(1:2)],1,paste,collapse="")
unique_names=names(table(all_names))
print(paste("There are ",length(unique_names)," peaks!",sep=""))
output=c()
for(i in 1:length(unique_names)){
# print(i)
if(i%%50==0){incProgress(50/length(unique_names),detail=paste(i," Peaks
Processed",sep=""))}
cur_name=unique_names[i]
temp=grep(cur_name,all_names)
temp_array=Pheno_Data[temp,]
temp=which.min(as.numeric(temp_array[,9]))
winner=temp_array[temp,]
winner=winner[,c(1:5,9)]
output=rbind(output,winner)
}
})
}
#Finally, we prepare the data for output by tweaking the significant digits
and scientific notation of the p values.
withProgress(value=.9,message="Outputing...",{
vals=as.matrix(output[,5])
vals=as.numeric(vals)
format(vals,digits=3,scientific=T)
vals=as.character(vals)
output[,5]=vals
})
#and we return that final output
output
})}
})
#This function simply takes the output of the workhorse function above and
repackages it in a form acessible to the GUI.
output$Overlap_Table <- renderDataTable({
if(input$Overlap_Calculate==0){
return (NULL)} else{
isolate(OL_GetData())
}
})
#This function creates a downloadable file of the table generated above
output$Overlap_Download<-downloadHandler(
filename = "results.txt",
content = function(file) {
```

```
write.table(isolate(OL_GetData()), file,row.names=F,sep="\t")
})
#This section creates a visual depiction of the Linkage Disequilibrium of a
specific region of the genome or, alternately, provides the precomputed LD
block around a provided rsID.
#####LD Block Section#####
#Like the section above, this is designed to take either a genomic interval
of interest OR a specific rsID.
output$LD_chr <-renderUI({if(input$LD_window_or_rsID=="window")
{selectInput("LD_Chr","Select Chromosome",c(1:19,"X"))}})
output$LD_LB <-renderUI({if(input$LD_window_or_rsID=="window")
{numericInput("LD_LB","Lower Bound (in MB)",value=10,min=0)}})
output$LD_additional <-renderUI({
if(input$LD_window_or_rsID=="window"){
numericInput("LD_UB","Upper Bound (in MB)",value=15,min=0)
} else {
textInput("LD_rsID","Please enter a SNP rsID")
}
})
output$LD_MAFCutoff <-renderUI({if(input$LD_window_or_rsID=="window")
{numericInput("LD_MAF","Minor Allele Frequency Cutoff",value=.05,min=0)}})
#First function for the 'simple' case where we are dealing with a rsID.
LD_GetData_rsID <- reactive({
if(input$LD_Calculate==0 || input$LD_window_or_rsID=="window") #if we are
looking for the LD within an interval, return nothing.
{return ("NULL")} else { #otherwise… all we need to do is grab that specific
SNP from our database…
query <- paste("Select top 1 [dataset],[rsID],[snp_chr],[snp_bp_mm10],[LD_
block_start_mm10],",
"[LD_block_end_mm10] from Unified.QTL_AllInfo WHERE rsID='",input$LD_rsID,"
'",sep="")
#print(query)
withProgress(value=0,message="Getting Data",{
Pheno_Data=sqlQuery(dbhandle, query)
val=Pheno_Data[2]
print(val)
val=val[[1]]
})
#and print out specific values found within its entry
outtext=paste("The SNP ",val," Located at Chr",Pheno_Data[3],":",Pheno_
Data[4]," Has a proposed LD window of ",
(as.numeric(Pheno_Data[6])-as.numeric(Pheno_Data[5]))," bp spanning from
",Pheno_Data[5]," to ",Pheno_Data[6],sep="")
outtext
```

```
}
})
#The more complicated situation is where instead of looking at a particular
rsID we are interested in visualizing the LD structure within a particular
region of the genome.
LD_GetData_Window <- reactive({
if(input$LD_Calculate==0 || input$LD_window_or_rsID==“rsID”) #if we are
looking at just one rsID... output nothing.
{return (“NULL”)} else {
print(“Beginning”)
options(scipen=999) #turn off scientific notation
#construct our query to extract out all the SNPs (but not the genotypes!)
within the region
query<- paste(“Select [snp_chr],[rsID],[snp_bp_mm10] from genotypes.
MouseDivArray_genotype_calls_plink_format where snp_chr=‘“,
input$LD_Chr,”‘ and snp_bp_mm10>‘“,input$LD_LB*1000000,”‘ and
snp_bp_mm10<‘“,input$LD_UB*1000000,”‘“,sep=““)
Pheno_Data=sqlQuery(dbhandle,query)
options(scipen=0) #return scientific notation to normal
SNPs=Pheno_Data[,2] #get SNP names
SNPs=paste(SNPs,collapse=“‘ OR rsID=‘“) #create a master search entry
which looks like ‘rsID=“SNPA” OR rsID=“SNPB” OR…’
print(“Getting SNPs”)
withProgress(value=0,message=“Getting SNPS”,{
#Here we actually get out the genotypes for the SNPs identified above (this
is done to save a significant amount of time)
query<- paste(“Select genotypes.MouseDivArray_genotype_calls_emma_format.*,
rsID AS Expr1 from genotypes.MouseDivArray_genotype_calls_emma_format WHERE
“, “rsID=’“,SNPs,”’“,sep=““)
#massage the data into the right format.
Pheno_Data=sqlQuery(dbhandle,query)
Pheno_Data=Pheno_Data[,−1]
Pheno_Data=Pheno_Data[,-ncol(Pheno_Data)]
positions=Pheno_Data[,1]
Pheno_Data=Pheno_Data[,−1]
#calculate the minor allele frequency of each SNP and remove those who whose
MAFs are less than the predefinited cutoff
sums=apply(Pheno_Data,1,sum,na.rm=TRUE)
ngoodcol=table(is.na(Pheno_Data[1,]))[1]
MAFS=sums/ngoodcol
tokeep=MAFS>input$LD_MAF
Pheno_Data=Pheno_Data[tokeep,]
positions=positions[tokeep]
})
```

```
#Now that we have the exact phenotypes we care about, we can calulate the
relationship of each of these SNPs to one another withProgress(value=.
5,message="Calculating Correlations",{
print("Calculating Correlations")
cortable=corFast(t(Pheno_Data),use="pairwise.complete.obs")
cortable[which(is.na(cortable))]=0
cortable2=cortablê2
rownames(cortable2)=positions
colnames(cortable2)=positions
#cortable2=1-cortable2
})
#and finally create the output PDF, which is simply a heatmap of the
correlations of each SNP to each other SNP within the window.
print("Creating Plot")
title=paste("LD Block Structure: Chr ",input$LD_Chr," ",input$LD_LB, " to
",input$LD_UB," Mb",sep="")
heatmap.2(cortable2, Rowv=FALSE,Colv=FALSE, dendrogram="none", col=heat.
colors(75), scale="none",
key=FALSE, symkey=FALSE, density.info="none", trace="none", ce
xRow=0.5,cexCol=.15,main=title)
}
})
#Two output functions, one for the rsID version and one for the window
version. output$LD_rsIDOut <- renderText({
if(input$LD_Calculate==0 || input$LD_window_or_rsID=="window"){return("")}
else{isolate(LD_GetData_rsID())}
})
output$LD_windowOut <- renderPlot({
if(input$LD_Calculate==0 || input$LD_window_or_rsID=="rsID"){return(NULL)}
else {isolate(LD_GetData_Window())}
})
#A very simple section which takes a gene name as input and opens up a
browser window to the Wellcome Trust Mouse Genomes SNP Query site for that
gene and the strains in the HMDP
#Originally, this actually opened in a frame, but recent changes to the
Wellcome Trust site mean that a new tab/window is now necessary.
#####NONSYNNONYMOUS SNP SECTION#####
output$NonSynnon_Result <-renderUI({
if(input$NonSynnon_Calculate==0){return(NULL)} else {
isolate({
temp=paste("http://www.sanger.ac.uk/sanger/Mouse_SnpViewer/rel-1505?gene=",
input
$NonSynnon_Gene,"&context=0&loc=&release=rel-1505&sn=frameshift_variant&sn=mi
ssense_variant&",
```

```
"sn=splice_region_variant&sn=stop_gained&sn=stop_lost&sv=complex_events&sv=co
py_number_gain&sv=",
"deletion&sv=insertion&sv=inversion&st=129s1_svimj&st=a_j&st=akr_j&st=balb_cj
&st=btbr_titpr3tf_j",
"&st=bub_bnj&st=c3h_hej&st=c57l_j&st=c58_j&st=cba_j&st=dba_2j&st=fvb_nj&st=i_
lnj&st=kk_hij&st=lp_j",
"&st=nod_shiltj&st=nzb_b1nj&st=sea_gnj", sep="")
browseURL(temp)
#tags$iframe(src=temp,seamless=F,height=1024,width=1600)
})
}
})
#This section creates a heatmap of all the values for a particular gene
across all the studies/strains of the HMDP
#####Vizualize Values Across Strains Section##### output$VVAS_StudyUI<-
renderUI({
if(input$VVAS_DataType=="Phenotype"){
VVAS_StudyChoices=c("Not", "Implemented","Yet") #There are some significiant
challenges here. See systems.genetics.ucla.edu for an eventual update...
selectInput("VVAS_Pheno", "Select Study", VVAS_StudyChoices )
}
if(input$VVAS_DataType=="Gene"){
textInput("VVAS_Gene", "Please Enter your probesetID OR Gene Name")
}
})
#This section of the UI lets users select a subset of all of the studies to
examine output$VVAS_SelectExperimentsUI <-renderUI({
selectors=allTables[allTables[,2]=="TranscriptAbundance",]
selectors=selectors[selectors[,4]=="VIEW",]
selectors=selectors[,3]
checkboxGroupInput("VVAS_Experiments", "Select Experiments to Include",
selectors, selected=selectors)
})
#Workhorse function for this section
VVAS_Output <-reactive({
if(input$VVAS_Calculate==0 || input$VVAS_DataType=="Phenotype")
{return ("NULL")} else {
withProgress(value=0,message="Setting up...",{
#prepare the specific studies we are interested in…
Table_subset=input$VVAS_Experiments
Table_subset=paste0("HMDP.TranscriptAbundance.",Table_subset)
Table_subset=as.matrix(Table_subset)
})
withProgress(value=0,message="Generating Table...",{
```

```
#here we are creating the first query of our output, to which the rest will
be added.
gene_query=paste("(ProbesetID='",input$VVAS_Gene,"' OR gene_symbol='",input
$VVAS_Gene,"')",sep="")
query <- paste("SELECT * FROM ",Table_subset[1]," WHERE ",
gene_query,paste="" )
})
print(paste0("Fetching Data From ",Table_subset[1]))
withProgress(value=1/length(Table_subset),message=paste0("Fetching Data From
",Table_subset[1]),{
#Actually get the data for the first row and process
#The eventual result should have the gene name as a row and all strains of
interest as columns. If more than one probeset is returned (if using gene
name instead of probesetID)
#then the most highly expressed value will be added.
Pheno_Data=as.matrix(sqlQuery(dbhandle, query))
Pheno_Data=Pheno_Data[,-c(1:2)]
Pheno_Data=as.matrix(Pheno_Data)
if(ncol(Pheno_Data)==1){
Pheno_Data=t(Pheno_Data)
}
if(nrow(Pheno_Data)>1){
Pheno_Data=apply(Pheno_Data,2,as.numeric)
averages=apply(Pheno_Data[,-c(1:2)],1,mean,na.rm=TRUE)
Pheno_Data=Pheno_Data[which.max(averages),]
Pheno_Data=t(as.matrix(Pheno_Data))
}
rownames(Pheno_Data)=strsplit(strsplit(Table_subset[1],".",fixed=T) [[1]]
[3],"_",fixed=T)[[1]][1]
gene_data=Pheno_Data
})
#now we do the same thing for every other study of interest, merging the
results with the growing master output table
for(i in 2:length(Table_subset)){
query <- paste("SELECT * FROM ",Table_subset[i]," WHERE ",
gene_query,paste="" )
print(paste0("Fetching Data From ",Table_subset[i]))
withProgress(value=i/length(Table_subset),message=paste0("Fetching Data From
",Table_subset[i]),{
Pheno_Data=as.matrix(sqlQuery(dbhandle, query))
Pheno_Data=Pheno_Data[,-c(1:2)]
Pheno_Data=as.matrix(Pheno_Data)
if(ncol(Pheno_Data)==1){Pheno_Data=t(Pheno_Data)}
if(nrow(Pheno_Data)==0){ #we need a special case if the gene/probe isn't
```

```
found in the study's array. In this case, we just add a row of NAs.
new_val= strsplit(Table_subset[i],".", fixed=T)[[1]][3]
temp=c(rownames(gene_data),new_val)
new_row=rep(NA,ncol(gene_data))
gene_data=rbind(gene_data,new_row)
rownames(gene_data)=temp
} else {
if(nrow(Pheno_Data)>1){
Pheno_Data=apply(Pheno_Data,2,as.numeric)
averages=apply(Pheno_Data[,-c(1:2)],1,mean,na.rm=TRUE)
Pheno_Data=Pheno_Data[which.max(averages),]
Pheno_Data=t(as.matrix(Pheno_Data))
}
rownames(Pheno_Data)=strsplit(Table_subset[i],".",fixed=T)[[1]][3]
temp=c(rownames(gene_data),rownames(Pheno_Data))
gene_data=merge(gene_data,Pheno_Data,all=TRUE,sort=FALSE)
rownames(gene_data)=temp
}
})
}
#and finally, we return the combined data gene_data
}
})
#Unused in the final code, this allows for testing of which samples will be
included for the strain select portion of the UI.
output$TEST_Checkbox <-renderText({
res=input$VVAS_SelectStrains
strain_classes=input$VVAS_SelectStrains
strains=c()
for(q in 1:length(strain_classes)){
temp=allStrains[allStrains[,2]==strain_classes[q],1]
strains=c(strains,temp)
}
print(strains)
})
#This section actually outputs the heatmap of all the expression values
output$VVAS_Plot <- renderPlot({
if(input$VVAS_Calculate==0){
return ("NULL")} else}
isolate({
VVAS_Data=VVAS_Output() #get the data...
print("Data Aquired")
if(VVAS_Data[1]!="NULL"){ #if there is data…
print("Running")
```

```
strain_classes=input$VVAS_SelectStrains #get which strain classes (mouse
panels) we are interested in from the GUI.
strains=c()
for(q in 1:length(strain_classes)){ #create a master list of strains of
interest.
temp=allStrains[allStrains[,2]==strain_classes[q],1]
strains=c(strains,temp)
}
#Filter output for only the strains of interest
temp=match(strains,colnames(VVAS_Data))
temp=temp[!is.na(temp)]
print(temp)
VVAS_Data=VVAS_Data[,temp]
#and now actually generate the heatmap.
withProgress(value=0,message="Generating Figure...",{
gene_data=VVAS_Data
pheno_names=rownames(gene_data)
strain_names=colnames(gene_data)
gene_data=gene_data[,order(strain_names)]
strain_names=strain_names[order(strain_names)]
gene_data=apply(gene_data,2,as.numeric)
rownames(gene_data)=pheno_names
heatmap.2(gene_data,Rowv=FALSE,Colv=FALSE,dendrogram="none",trace="none",
col=greenred(100),na.color="grey",keysize=1.2,density.
info="none",margins=c(5,9))
})
}
})
}
})
#This function allows users to download the values plotted in the heatmap.
output$VVAS_Download<-downloadHandler(
filename = "results.txt",
content = function(file) {
write.table(isolate(VVAS_Output()), file,row.names=T,sep="\t")
})
#This section examines the entirety (or a subset) of the data currently
available to find significant/suggestive correlations between a phenotype of
intetrest and other phenotypes, studies, tissues, etc.
#####Find Correlations#####
#This first function allows the user to select a subset of the entire data
to look for correlations in. Obviously, the fewer experiments, the faster it
goes.
output$FC_SelectExperimentsUI <-renderUI({
```

```
selectors=allTables[allTables[,2]==“Correlations”,] #find all correlations
in allTables
selectors=selectors[selectors[,4]==“VIEW”,] #find all of those correlations
which are views (to avoid duplicates)
selectors=selectors[,3] #get names
#We have to clean up the names a little bit (namely remove anything after
the first “_”), so here is a quick function to do so.
retElement <- function(x,num){
temp=strsplit(x,”_”)
temp=temp[[1]][num]
return(temp)
}
selectors=sapply(selectors,retElement,1)
selectors=names(table(selectors))
checkboxGroupInput(“FC_Experiments”, “Select Experiments to Include”,
selectors, selected=selectors)
})
#The workhorse function which actually finds the correlations
FC_GetResults<- reactive({
experiments=allTables[allTables[,2]==“Correlations”,] #get all correlations
experiments=experiments[experiments[,4]==“VIEW”,]
experiments=experiments[,3]
if(!input$FC_Include_Probes){ #we have the option to keep or remove all the
eQTLs.
temp=sapply(experiments,retElement,3)!=“trx” #if we don’t want the eQTLs, we
filter them out.
countElement <-function(x){
temp=strsplit(x,”_”)
temp=length(temp[[1]])
return(temp)
}
t2=sapply(experiments,countElement)!=4
temp= temp | t2
experiments=experiments[temp]
}
all_experiments=experiments
#We now filter ALL experiments by the ones we selected above that we wish to
keep experiments=c()
for(i in 1:length(input$FC_Experiments)){
temp=input$FC_Experiments[i]
temp=paste(temp,”_”,sep=“”)
temp=grep(temp,all_experiments,fixed=T)
temp=all_experiments[temp]
experiments=c(experiments,temp)
```

```
}
#Trim off the 'AllInfo' part.
for(i in 1:length(experiments)){
temp=strsplit(experiments[i],"_")[[1]]
temp=temp[-length(temp)]
temp=paste(temp,collapse="_")
experiments[i]=temp
}
#Make the MASSIVE experiment filter for the eventual query
exp_filter="(dataset='"
for(i in 1:(length(experiments)-1)){
temp=experiments[i]
exp_filter=paste0(exp_filter,temp,"' OR dataset='")
}
exp_filter=paste0(exp_filter,experiments[length(experiments)],"')")
# print(exp_filter)
#The much smaller phenotype_filter
pheno_query=paste("(ProbesetID_1='",input$FC_Input,"' OR gene_
symbol='",input$FC_Input,"' OR clinical_trait_1='",input$FC_Input,"' OR
metabolite_1='",input$FC_Input,"' OR protein_1='",input$FC_Input,"')",sep="")
# print(pheno_query)
#The tiny pvalue filter
pval_filter=paste0("pvalue<='",input$FC_threshold,"'")
# print(pval_filter)
#and finally we combine everything together to create our master SQL query.
final_query=paste0("SELECT * FROM Unified.Correlations_AllInfo WHERE
",exp_filter," AND ",pheno_query," AND ",pval_filter)
print(final_query)
withProgress(value=0,message="Generating Results… this may take some
time.",! #It really might. Working on a way to improve speed now.
FC_Data=sqlQuery(dbhandle, final_query) #and here we actually are getting
the results
})
outdata=c()
withProgress(value=0,message="Processing Results…",{
#our correlations can be with all sorts of different things. a gene, a
phenotype, a metabolite, a protein, etc, etc. Our initial input can be any
of those things as wel
#as a result, we have to figure out which entries in our unified correlation
database is actually filled
for(i in 1:nrow(FC_Data)){ #for each row of the correlations we've
downloaded incProgress(1/nrow(FC_Data))
cur_row=FC_Data[i,] #extract that row
tokeep=c(2:4) #keep a few columns that are always needed (type of
```

```
correlation, tissue of interest, study) and then look to see which other
columns are filled
if(!is.na(cur_row[5])){ tokeep=c(tokeep,5,6)} #gene 1
if(!is.na(cur_row[11])){ tokeep=c(tokeep,11,12)} #gene 2
if(!is.na(cur_row[17])){ tokeep=c(tokeep,17,18)} #phenotype 1
if(!is.na(cur_row[20])){ tokeep=c(tokeep,20,21)} #phenotype 2
if(!is.na(cur_row[23])){ tokeep=c(tokeep,23,24)} #metabolite 1
if(!is.na(cur_row[25])){ tokeep=c(tokeep,25,26)} #metabolite 2
if(!is.na(cur_row[27])){ tokeep=c(tokeep,27,27)} #protein 1
if(!is.na(cur_row[28])){ tokeep=c(tokeep,28,28)} #protein 2
tokeep=c(tokeep,29,30) #and we want to keep the last two values as well
(correlation score and pvalue)
cur_row=cur_row[tokeep] #and now we actually filter the row to the values we
care about
names(cur_row)=c("class","tissue","study","Pheno1_ID","Pheno1_
Info","Pheno2_ID","Pheno2_Info","bicor","pvalue") #add names to those values
outdata=rbind(outdata,cur_row) #and add it to our master output.
}
})
#this output is a condensed form of the SQL query which removes empty spaces
and is better for visualization
outdata
})
#the output function for the data above.
output$FC_Output <-renderDataTable({
if(input$FC_Calculate==0){
return (NULL)} else{
isolate(FC_GetResults())
}
})
#and a downloader to allow downloading of all correlations.
output$FC_Download<-downloadHandler(
filename = "results.txt",
content = function(file) {
write.table(isolate(FC_GetResults()), file,row.names=F,sep="\t")
})
})
#####EXTRAS#####
#Takes a string x, splits it and reurns the num-th element.
retElement <- function(x,num){
temp=strsplit(x,"_")
temp=temp[[1]][num]
return(temp)
}
```

7 ui.R

```
# A Graphical User Interface for querying a genetics SQL Database using
Shiny in R
# Version: 0.7
# Last Modified: 12/9/15
#
# The following is an implementation of a GUI using the Shiny package in
Rstudio. Shiny programs have two scripts associated with them. This script,
ui.R, controls the appearance of
# The GUI and provides inputs to and displays outputs from Server.R which
contains the actual functions.
#for details on how this page's layout works, please see http://
shiny.rstudio.com/tutorial/ and http://shiny.rstudio.com/reference/shiny/
latest/shinyUI(fluidPage(
titlePanel("Welcome to the HMDP Database Shiny Server v0.7"), tabsetPanel(
tabPanel("Start Here/Login",
h6("Welcome to the first iteration of the searchable HMDP Database. Please
click a relevant tab to begin."),
textInput("Password","Enter password for full access"),
actionButton("Password_Go","Login"),
textOutput("PassOK")),
tabPanel("Visualize GWAS Result",
sidebarLayout(
sidebarPanel(
selectInput("DataViz_DataType",label=h3("Select a type of data"),choices = c
("Clinical","Expression","Metabolite","Protein")), #a selectInput is a
dropdown menu htmlOutput("DataViz_StudyUI"), #an 'htmlOutput' is actually a
way to create dynamic inputs. In this case, Server.R is taking the selection
from above and creating a new
#selectInput populated with all the studies which have that type of data
htmlOutput("DataViz_FinalTableSelectUI"), #then this one is allowing for
fine tuning of the selection (typically selecting which gender of mice to
examine)
htmlOutput("DataViz_PhenotypeUI"), #and finally this one gives you a list of
all possible phenotypes that can be used.
actionButton("DataViz_Calculate","Create Manhattan Plot"), #This is a button
which, when clicked, tells Server.R to start calculating.
selectInput("DataViz_Chromosome",label="Which Chromosome?",choices=c("Al
l",c(1:19),"X"),selected="All"), #another select input
numericInput("DataViz_Lower_Bound","Lower Bound (In MB)",1,min=0),
#a numeric input which will take any number
numericInput("DataViz_Upper_Bound","Upper Bound (In MB)",999,min=0)
),
```

```
mainPanel(downloadButton('DataViz_Download', 'Download These Results'),
#creates a download button which takes a created file from Server.R
plotOutput('DataViz_Manhattan'), #creates a plot
h5("At distances of less than 10Mb, the UCSC Genome Browser will Appear
Below."),
htmlOutput("DV_GenomeBrowser"))) #once again an htmlOutput, but in this case
it really is an output, namely a visualization of the UCSC genome browser
),
tabPanel("Create Beeswarm Plot",
sidebarLayout(
sidebarPanel(
selectInput("Beeswarm_DataType",label=h3("Select a type of data"),choices =
c("Clinical","Expression","Metabolite","Protein")),
htmlOutput("Beeswarm_StudyUI"),
htmlOutput("Beeswarm_FinalTableSelectUI"),
htmlOutput("Beeswarm_PhenotypeUI"),
textInput("Beeswarm_rsID","Enter your SNP of choice",value=""), #will take
any string as an input
actionButton("Beeswarm_Calculate","Create Plot") ),
mainPanel(plotOutput('BS_Plot'))
)
),
tabPanel("Visualize Values Across Strains and Tissues",
sidebarLayout(
sidebarPanel(
selectInput("VVAS_DataType",label=h3("Select a type of data"),choices =
c("Phenotype","Gene")),
htmlOutput("VVAS_StudyUI"),
htmlOutput("VVAS_SelectExperimentsUI"),
checkboxGroupInput("VVAS_SelectStrains",label="Strain Groups",choices=c("In
bred","AxB","BxA","BxD","BxH","CxB"),selected=c("Inbred","AxB","BxA","BxD","B
xH","CxB")), actionButton("VVAS_Calculate","Create Plot") ),
mainPanel(downloadButton('VVAS_Download', 'Download These Results'),
plotOutput('VVAS_Plot')
#,textOutput("TEST_Checkbox")
)
) ),
tabPanel("Nonsynnonymous SNPs",
sidebarLayout(
sidebarPanel(
textInput("NonSynnon_Gene","Enter Gene (SYMBOL FOR NOW)"),
actionButton("NonSynnon_Calculate","Run")
),
mainPanel(htmlOutput("NonSynnon_Result")))),
```

```
tabPanel("cis-eQTLs",
sidebarLayout(
sidebarPanel(
htmlOutput("ciseQTL_PhenotypeUI"),
numericInput("ciseQTL_Window","Size of cis-eQTL window in MB",min=0,value=2),
actionButton("ciseQTL_Calculate","Create Table")
),
mainPanel(dataTableOutput('ciseQTL_Table'),
downloadButton('ciseQTL_Download', 'Download These Results')))
),
tabPanel("Gene/Phenotype Correlations",sidebarLayout(
sidebarPanel(
textInput("FC_Input","Enter your gene or phenotype name"),
htmlOutput("FC_SelectExperimentsUI"),
numericInput("FC_threshold","P-value
threshold",min=0,value=.0000042,max=1),
checkboxInput("FC_Include_Probes","Include Genes?",value=TRUE), #a simple
checkbox for TRUE/FALSE statments. In this case, should genes be included
when calculating correlations?
actionButton("FC_Calculate","Create Table")
),
mainPanel(dataTableOutput("FC_Output"),
downloadButton('FC_Download', 'Download These Results'))
)),
tabPanel("Overlapping Loci",
sidebarLayout(
sidebarPanel(
selectInput("Overlap_window_or_rsID","Please select to
begin",c("window","rsID"))
,htmlOutput("Overlap_chr"),
htmlOutput("Overlap_LB"),
htmlOutput("Overlap_additional"),
numericInput("Overlap_threshold","P-value threshold",min=0,val ue=.
0000042,max=1),
checkboxInput("Overlap_includeGenes","Include eQTLs?",value=FALSE),
actionButton("Overlap_Calculate","Create Table")
),
mainPanel(dataTableOutput('Overlap_Table'),
downloadButton('Overlap_Download', 'Download These Results'))
)),
tabPanel("Generate LD Plot",
sidebarLayout(
sidebarPanel(
selectInput("LD_window_or_rsID","Please select to begin",c("window","rsID"))
```

```
,htmlOutput("LD_chr"),
htmlOutput("LD_LB"),
htmlOutput("LD_additional"),
htmlOutput("LD_MAFCutoff"),
actionButton("LD_Calculate","Calculate!")
),
mainPanel(textOutput("LD_rsIDOut"),plotOutput("LD_windowOut"))
)),
tabPanel("Gene Name Conversions",
sidebarLayout(
sidebarPanel(
textInput("Lookup_One","Please enter a gene name or probesetID"),
fileInput("Lookup_Batch","Or upload a file for batch conversion"),
actionButton("Lookup_Button", "Convert!")
),
mainPanel(
dataTableOutput("Lookup_Table")
)
)),
tabPanel("More Tools To Come!",h3("Soon...")),
tabPanel("Bugs/Suggestions",
textInput("Suggestion_Name","Name"),
textInput("Suggestion_Report","Suggestion/Bug"),
tags$style(type='text/css', "#Suggestion_Report { height: 300px; width:
600px; }"),
actionButton("Suggestion_Button","Suggest!"),
textOutput("Suggestion_Text"),
h3("Planned Changes:"),
h4("Make it Faster (Especially correlations)"),
h4("Eliminate Bugs"),
h4("Make it Look Nice")
)
)))
```

References

1. Stancakova A, Javorsky M, Kuulasmaa T, Haffner SM, Kuusisto J, Laakso M (2009) Changes in insulin sensitivity and insulin release in relation to glycemia and glucose tolerance in 6,414 Finnish men. Diabetes 58(5):1212–1221. doi:10.2337/db08-1607 [PubMed: 19223598]
2. Ghazalpour A, Rau CD, Farber CR, Bennett BJ, Orozco lD, van Nas A, Pan C, Allayee H, Beaven SW, Civelek M, Davis RC, Drake TA, Friedman RA, Furlotte N, Hui ST, Jentsch JD, Kostem E, Kang HM, Kang EY, Joo JW, Korshunov VA, Laughlin RE, Martin LJ, Ohmen JD, Parks BW, Pellegrini M, Reue K, Smith DJ, Tetradis S, Wang J, Wang Y, Weiss JN, Kirchgessner T, Gargalovic PS, Eskin E, Lusis AJ, LeBoeuf Rc(2012) Hybrid mouse diversity panel: a panel of inbred mouse strains suitable for analysis of complex genetic traits. Mamm Genome 23(9–10):680–692. doi: 10.1007/s00335-012-9411-5 [PubMed: 22892838]

3. Threadgill DW, Miller DR, Churchill GA, de Villena FP (2011) The collaborative cross: a recombinant inbred mouse population for the systems genetic era. ILAR J 52(1):24–31 [PubMed: 21411855]

4. Chang W, Cheng J, Allaire J, Xie Y, McPherson J (2015) Shiny: web application framework for R. http://cran.r-project.org/package=shiny

5. Bennett BJ, Farber CR, Orozco L, Kang HM, Ghazalpour A, Siemers N, Neubauer M, Neuhaus I, Yordanova R, Guan B, Truong A, Yang WP, He A, Kayne P, Gargalovic P, Kirchgessner T, Pan C, Castellani LW, Kostem E, Furlotte N, Drake TA, Eskin E, Lusis AJ (2010) A high-resolution association mapping panel for the dissection of complex traits in mice. Genome Res 20(2):281–290. doi:10.1101/gr.099234.109 [PubMed: 20054062]

6. Calabrese G, Bennett BJ, Orozco L, Kang HM, Eskin E, Dombret C, De Backer O, Lusis AJ, Farber CR (2012) Systems genetic analysis of osteoblast-lineage cells. PLoS Genet 8(12): e1003150. doi: 10.1371/journal.pgen.1003150 [PubMed: 23300464]

7. Farber CR, Bennett BJ, Orozco L, Zou W, Lira A, Kostem E, Kang HM, Furlotte N, Berberyan A, Ghazalpour A, Suwanwela J, Drake TA, Eskin E, Wang QT, Teitelbaum SL, Lusis AJ (2011) Mouse genome-wide association and systems genetics identify Asxl2 as a regulator of bone mineral density and osteoclastogenesis. PLoS Genet 7(4):e1002038. doi:10.1371/journal.pgen.1002038 [PubMed: 21490954]

8. Park CC, Gale GD, de Jong S, Ghazalpour A, Bennett BJ, Farber CR, Langfelder P, Lin A, Khan AH, Eskin E, Horvath S, Lusis AJ, Ophoff RA, Smith DJ (2011) Gene networks associated with conditional fear in mice identified using a systems genetics approach. BMC Syst Biol 5:43. doi: 10.1186/1752-0509-5-43 [PubMed: 21410935]

9. Davis RC, van Nas A, Bennett B, Orozco L, Pan C, Rau CD, Eskin E, Lusis AJ (2013) Genome-wide association mapping of blood cell traits in mice. Mamm Genome 24(3–4):105–118. doi: 10.1007/s00335-013-9448-0 [PubMed: 23417284]

10. Ghazalpour A, Bennett B, Petyuk VA, Orozco L, Hagopian R, Mungrue IN, Farber CR, Sinsheimer J, Kang HM, Furlotte N, Park CC, Wen PZ, Brewer H, Weitz K, Camp DG II, Pan C, Yordanova R, Neuhaus I, Tilford C, Siemers N, Gargalovic P, Eskin E, Kirchgessner T, Smith dJ, Smith RD, Lusis AJ (2011) Comparative analysis of proteome and transcriptome variation in mouse. PLoS Genet 7(6):e1001393. doi:10.1371/journal.pgen.1001393 [PubMed: 21695224]

11. Orozco LD, Bennett BJ, Farber CR, Ghazalpour A, Pan C, Che N, Wen P, Qi HX, Mutukulu A, Siemers N, Neuhaus I, Yordanova R, Gargalovic P, Pellegrini M, Kirchgessner T, Lusis AJ (2012) Unraveling inflammatory responses using systems genetics and gene-environment interactions in macrophages. Cell 151(3):658–670. doi:10.1016/j.cell.2012.08.043 [PubMed: 23101632]

12. Ghazalpour A, Bennett BJ, Shih D, Che N, Orozco L, Pan C, Hagopian R, He A, Kayne P, Yang WP, Kirchgessner T, Lusis AJ (2014) Genetic regulation of mouse liver metabolite levels. Mol Syst Biol 10:730. doi:10.15252/msb.20135004 [PubMed: 24860088]

13. Orozco LD, Morselli M, Rubbi L, Guo W, Go J, Shi H, Lopez D, Furlotte NA, Bennett BJ, Farber CR, Ghazalpour A, Zhang MQ, Bahous R, Rozen R, Lusis AJ, Pellegrini M (2015) Epigenome-wide association of liver methylation patterns and complex metabolic traits in mice. Cell Metab 21(6):905–917. doi:10.1016/j.cmet.2015.04.025 [PubMed: 26039453]

14. Parks BW, Nam E, Org E, Kostem E, Norheim F, Hui ST, Pan C, Civelek M, Rau CD, Bennett BJ, Mehrabian M, Ursell LK, He A, Castellani LW, Zinker B, Kirby M, Drake TA, Drevon CA, Knight R, Gargalovic P, Kirchgessner T, Eskin E, Lusis AJ (2013) Genetic control of obesity and gut microbiota composition in response to high-fat, high-sucrose diet in mice. Cell Metab 17(1):141–152. doi:10.1016/j.cmet.2012.12.007 [PubMed: 23312289]

15. Parks BW, Sallam T, Mehrabian M, Psychogios N, Hui ST, Norheim F, Castellani LW, Rau CD, Pan C, Phun J, Zhou Z, Yang WP, Neuhaus I, Gargalovic PS, Kirchgessner TG, Graham M, Lee R, Tontonoz P, Gerszten RE, Hevener AL, Lusis AJ (2015) Genetic architecture of insulin resistance in the mouse. Cell Metab 21(2):334–346. doi:10.1016/j.cmet.2015.01.002 [PubMed: 25651185]

16. Org E, Parks BW, Joo JW, Emert B, Schwartzman W, Kang EY, Mehrabian M, Pan C, Knight R, Gunsalus R, Drake TA, Eskin E, Lusis AJ (2015) Genetic and environmental control of host-gut microbiota interactions. Genome Res 25(10):1558–1569. doi:10.1101/gr.194118.115 [PubMed: 26260972]

17. Hui ST, Parks BW, Org E, Norheim F, Che N, Pan C, Castellani LW, Charugundla S, Dirks DL, Psychogios N, Neuhaus I, Gerszten RE, Kirchgessner T, Gargalovic PS, Lusis AJ (2015) The genetic architecture of NAFLD among inbred strains of mice. Elife 4:e05607. doi:10.7554/eLife.05607 [PubMed: 26067236]

18. Rau CD, Wang J, Avetisyan R, Romay MC, Martin L, Ren S, Wang Y, Lusis AJ (2015) Mapping genetic contributions to cardiac pathology induced by Beta-adrenergic stimulation in mice. Circ Cardiovasc Genet 8(1):40–49. doi:10.1161/CIRCGENETICS.113.000732 [PubMed: 25480693]

19. Bennett BJ, Davis RC, Civelek M, Orozco L, Wu J, Qi Hx, Pan C, Packard RR, Eskin E, Yan M, Kirchgessner T, Wang Z, Li X, Gregory JC, Hazen SL, Gargalovic P, Lusis AJ (2015) Genetic architecture of atherosclerosis in mice: a systems genetics analysis of common inbred strains. PLoS Genet 11:e1005711 [PubMed: 26694027]

20. Crow AL, Ohmen J, Wang J, Lavinsky J, Hartiala J, Li Q, Li X, Salehide P, Eskin E, Pan C, Lusis Aj, Allayee H, Friedman RA (2015) The genetic architecture of hearing impairment in mice: evidence for frequency specific genetic determinants. G3 (Bethesda) 5:2329–2339. doi:10.1534/g3.115.021592 [PubMed: 26342000]

21. Ohmen J, Kang EY, Li X, Joo JW, Hormozdiari F, Zheng QY, Davis rC, Lusis AJ, Eskin E, Friedman RA (2014) Genome-wide association study for age-related hearing loss (AHL) in the mouse: a meta-analysis. J Assoc Res Otolaryngol 15(3):335–352. doi:10.1007/s10162-014-0443-2 [PubMed: 24570207]

22. Turner S (2014) qqman: Q-Q and Manhattan plots for GWAS data. http://cran.r-project.org/package=qqman

23. Eklund A (2015) Beeswarm: the Bee Swarm plot, an alternative to Stripchart. http://cran.r-project.org/package=beeswarm

24. Yang H, Wang JR, Didion JP, Buus RJ, Bell TA, Welsh CE, Bonhomme F, Yu AH, Nachman MW, Pialek J, Tucker P, Boursot P, McMillan L, Churchill GA, de Villena FP (2011) Subspecific origin and haplotype diversity in the laboratory mouse. Nat Genet 43(7):648–655. doi:10.1038/ng.847 [PubMed: 21623374]

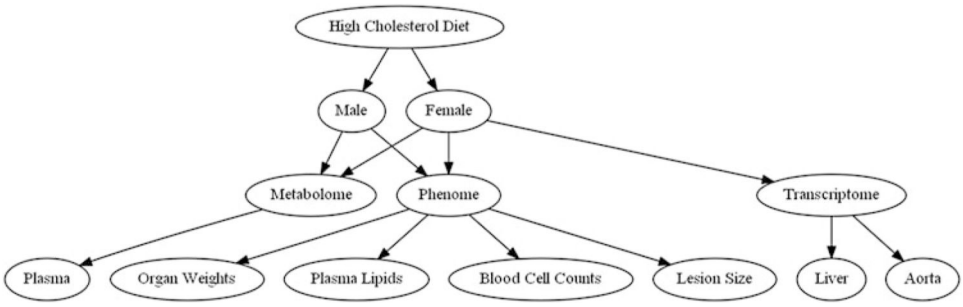

Fig. 1.

A subset of the data available in the HMDP Database. Depicted here is a subset of the total HMDP database, organized with study name (in this case, a high-cholesterol diet) at the *top level,* followed by gender, then -omics level, and finally by individual tissues or phenotypes. A full depiction of the contents of the HMDP Database can be found in Fig. S1

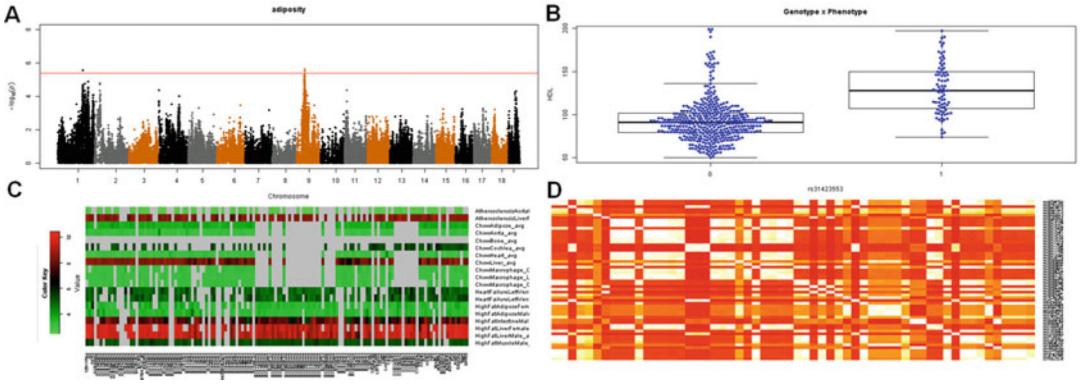

Fig. 2.

Visualization tool outputs in the HMDP Database GUI. (**a**) A Manhattan plot for adiposity reveals a significant association on chromosome 9. (**b**) A beeswarm plot demonstrates the effect of SNP rs31423553 on HDL levels in plasma. (**c**) The expression of the gene *Abcc6* is plotted across the different studies of the HMDP, revealing high expression in liver compared to other tissues. (**d**) A LD plot of chromosome 3 between 10 and 11 Mb shows evidence of a small LD block from 10.1 to 10.5 Mb

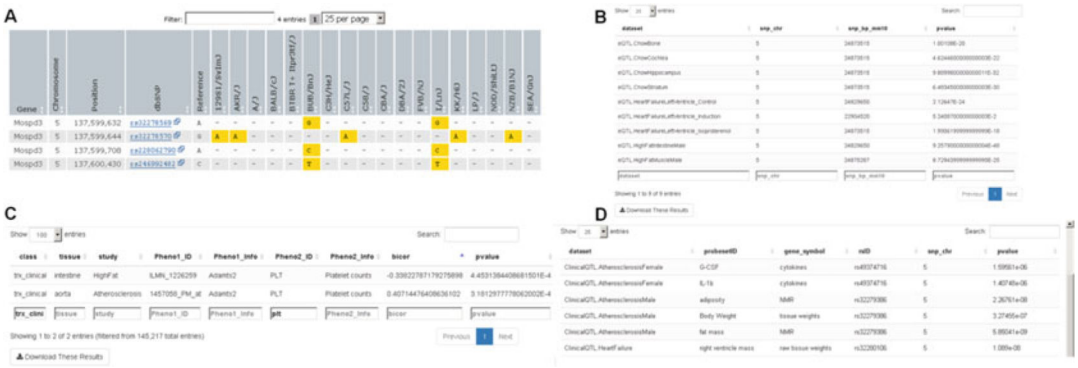

Fig. 3.

Discovery Outputs of the HMDP Database GUI. (**a**) Querying the Wellcome Trust Mouse Genomes resource reveals a number of nonsynonymous mutations in the gene *Mospd3*. (**b**) *cis*-eQTLs located near *Prkag2* show strong local regulation in multiple tissues (**c**) Correlation of *Adamts2* with other phenotypes (*see* Fig. S8 for additional correlations) reveals a previously unappreciated correlation with platelet counts in multiple studies. (**d**) Examination of the *Mospd3* locus for RV weight reveals additional significant loci across the HMDP studies

Table 1

Experimental models in the HMDP database

Environmental condition/stressor	Primary traits [references]
1. Low-fat chow diet	Plasma lipids, adiposity [5]
	Bone density [6, 7]
	Behavior [8]
	Blood cell levels [9]
	Proteomics [10]
	Macrophage inflammation [11]
	Metabolomics (hepatic) [12]
	DNA methylation [13]
2. High-fat, high sucrose diet	Dietary responsiveness [14]
	Diabetes/insulin resistance [15]
	Gut microbiota [14, 16]
	Bone marrow stem cells (ALLAYEE, LUSIS) Fatty liver [17]
3. Isoproterenol treatment	Heart failure [18]
4. High-fat, high-cholesterol diet and ApoE-Leiden, CETP transgenes	Atherosclerosis [19]
5. Low-fat chow diet, auditory stressors	Hearing [20, 21]