PREPARATION OF A GENOMIC DATASET FOR AN INVESTIGATIVE PROJECT: "DISCOVERY OF SPORADIC ALZHEIMER'S DISEASE IMPLICATED GENE VARIANTS THROUGH ANALYSIS OF EPISTASIS WITHIN PATHWAYS"

By

Rex M. Williamson

A CAPSTONE PROJECT

Presented to
the Department of Medical Informatics and Clinical Epidemiology
and the Oregon Health & Science University School of Medicine
in partial fulfillment of
the requirements for the degree of

Master of Biomedical Informatics

March 2013

School of Medicine

Oregon Health & Science University

CERTIFICATE OF APPROVAL

This is to certify that the Master's Capstone Project of

Rex M. Williamson

"Preparation of a Genomic Dataset for an Investigative Project: Discovery of Sporadic Alzheimer's Disease Implicated Gene Variants through Analysis of Epistasis within Pathways"

Has been approved

Beth Wilmot, Ph.D., Capstone Advisor

TABLE of CONTENTS

AKNOWLEDGMENTS:

I would like to thank the faculty and staff of the Department of Medical Informatics and Clinical Epidemiology (DMICE) within the School of Medicine at the Oregon Health & Science University for giving me the opportunity to develop and work on a project such as this.

I especially wish to thank:

Beth Wilmot, Ph.D., my advisor, for her unflagging encouragement and patience;

Kyle Lambert (soon to be Ph.D.) for his tutelage in Python and R;

Jane Rossiter, my wife, with all my love and appreciation because this truly could not have happened without her, and;

Ryan, our son, who showed amazing patience and understanding for a 4-6 year old while Daddy had to work and go to school – this degree is for him.

ABSTRACT:

Today’s genomics data requires a great deal of preprocessing before it can be utilized in analysis of biological questions. This work details the steps and requirements for processing genome-wide association studies (GWAS) in preparation for analysis. The scripting language ‘Python’ is employed to open and read files of genomic datasets including phenotypic data, genotypic data, and demographics data, of a GWAS performed by the Harvard Brain Tissue Resource Center (HBTRC) as well as the Alzheimer’s Disease Neuroimaging Initiative (ADNI). The data files’ subjects remain de-identified. These raw files are then processed by the scripting language ‘Python’ to create hypothesis-dependent edited versions of those files suitable for use in a bioinformatics investigative genomics study. Exploratory data analysis (EDA) is performed using ‘R’ to describe the datasets and explore their suitability for the investigative study, including simple graphs. Reasons for dataset rejection as well as acceptance are discussed. Data cleaning methods are then utilized to remove non-informant subject data (those missing data necessary to the project’s biological question and hypothesis) from the accepted dataset files, with EDA then performed on the cleaned data files. The resultant ADNI data files are suitable for use by ‘R’ for an investigative study project. Data request procedures as well as data use agreements are discussed, as are remote file access, scripting naming conventions, file management, and data security. Scripts described are included as addenda.

I. INTRODUCTION

A. REASON FOR PROJECT

Preparation of a genomic dataset for an investigative project (hereafter referred to simply as “this project”) entails the exploratory data analysis and data cleaning of electronic genomics files prior to their selection for use for research purposes, as well as and data manipulation/formatting to prepare the data for use in the research study.

Prior to selecting any dataset for a study it must be explored to determine its suitability for the study. This exploratory data analysis (EDA) examines the important aspects of the dataset as relates to its use in the proposed study. These aspects might include: sample size (N) before and after any data cleaning; the inclusion or omission of necessary information, variables or covariates related to the proposed study; the methods employed to gather the data. EDA seeks both to get a better general understanding of the dataset and to determine whether or not the dataset is suitable for use in any particular research study. This “study suitability determination” should be done in such a way that non-suitable datasets can be identified and eliminated from consideration or excluded from the research study as quickly as possible.

This project was in support of a genetic variation research study. Genetic variation research strives to enhance our growing understanding of the linkages between genes or gene variants, and diseases. Such research is data-intensive, often employing large, varied datasets.[1] Before any genomics dataset may be used for research, it must be cleaned. That is, it must be checked for duplicated or out-of-range values, for missing and incomplete values, variances in syntax and semantics (especially in the case of non-numeric data), and must often be transformed into values or styles required by its intended application(s).[2]

Data cleaning requires knowledge of the data and usually cannot be performed without the assistance of a domain expert, but must be as automated as possible due to the size of datasets.[2] This project involves data cleaning and preparation for a case-control association study of a genome-wide association study (GWAS).

B. BIOLOGICAL QUESTION

This project focused on preparing a genetic variation dataset in order to answer the biological question, "Can pair-wise analysis of epistasis within genetic pathways using genome-wide common genetic variants lead to the discovery of genes implicated in Late-Onset Alzheimer's Disease (LOAD)?" Stated another way, "Do small-effect alleles interact within biological pathways to cause the phenotype?" The goal of the study was the identification of novel genes and pathways involved in sporadic Alzheimer's Disease (sporadic AD) using often-discounted information in a Genome Wide Association Study (GWAS).[3] The project had two aims: 1) Identify pathways enriched in sporadic AD by case-control logistic regression of single nucleotide polymorphic gene variants (SNPs), and 2) Identify epistatic interactions of genes in pathways enriched in sporadic AD.

This required domain knowledge of LOAD (also known as sporadic AD), its pathology, diagnosis and predictive tests, as well as of the genomics platforms and methods used to obtain the GWAS genetic dataset(s) used for the project.

C. LOGISTIC REGRESSION EQUATION COMPONENTS NECESSARY TO ANSWER THE BIOLOGICAL QUESTION

Alzheimer's Disease (AD) is the most common form of dementia and is expected to afflict 1:85 persons world-wide by 2050.[4] Its neural pathology of extra-cellular mis-folded amyloid-β plaques coupled with inter-cellular tangles of hyperphosphorylated Tau proteins leads to neural cell disequilibrium, dysfunction and cell death, culminating in the afflicted individual's progressive loss of cognition and function.[5, 6, 7] AD is diagnosed as "suspected" by batteries of cognitive and motor tests as well as neural imaging – but it can only be "confirmed" by autopsy.[8, 9] Mean life expectancy after a "suspected" diagnosis is seven years.[10] In 2010, the cost to care for AD patients in the United States was $172 Billion and these costs are estimated to reach $1.1 Trillion in 2050.[11]

LOAD is also referred to as "Sporadic AD" to differentiate it from familial, or early-onset, AD which account for only 0.1% of AD cases and whose genetic causes are well known.[12, 13] LOAD is defined as Alzheimer's Disease having an onset at or after age 65 years.[14]

The first covariate is the average risk of any individual developing sporadic AD. This has been determined to be 13 per cent for individuals age 65 and older.15 This value is used as a baseline - the chance for any random individual in the study to have the phenotype. This constant was to be included in the logistic regression equation as a numeric decimal value of 0.13.

The chances of having the disease increase with age: 5% of the population at age 65 years, between 30%-50% at age 80+ years.[5] For this reason, the individual's age had to be considered as one of the covariates in the logistic regression equation. Age was determined from date of birth (DoB) subtracted from date of death (DoD) or date of first diagnosis (DoFD) and was to be represented as a numeric value. DoFD minus DoB was to be used to yield the afflicted individual's age at onset (AaO) for comparison to non-afflicted individuals of the same age, while DoD minus DoB was to be used to determined age at death (AaD) for all individuals in the study. AaD was proposed as the "age" to be used as the covariate for the study but AaO was to be collected in hopes that it might yield interesting results when used as a covariate.

The greatest genetic link to the sporadic form of the disease has so far been the number of copies of Apolipoprotein E gene, variant ε4 (ApoE-ε4). The disease is thought to be polygenic but between 40%-80% of sporadic AD cases have at least one copy of ApoE-ε4.[13, 14] Therefore, the individual's ApoE-ε4 allele status had to be considered as one of the covariates in the logistic regression equation. This covariate was to be expressed as the stratified integer number of ApoE-ε4 alleles each individual had in their genome: 0 (homozygous negative: no copies of ApoE-ε4), 1 (heterozygous: 1 copy of -ε4, plus 1 copy of some other ApoE variant), or 2 (homozygous positive: both alleles of the individual's ApoE gene were variant -ε4).

Significantly, 2/3 of the persons afflicted with LOAD or sporadic AD are female.[4] Thus, the individual's gender must be considered as one of the logistic regression equation covariates. This is a binary value (either male or female). For this study, male was to be represented as '0' while female was to be represented as '1'.

Lastly, each SNP was to be tested for pair-wise epistatic interactions within its member biological pathway(s) by genomics software outside the scope of and not covered by this study.

These covariates result in the following logistic regression equation model: ***Outcome*** *(AD Status)* ***= average risk of having AD + individual's gender*** *(0, 1)* ***+ individual's ApoE-ε4 allele status*** *(0, 1, 2)* ***+ each SNP tested individually***.

This research study was designed to focus on significance levels of 1x10-4 for association as opposed to the generally-accepted significance level of 1x10-7.[16]

This project did not perform logistic regression analysis. However, the logistic equation covariates determined the EDA focus, data cleaning requirements, data manipulation requirements, and ultimately the selection criteria for any dataset(s) related to the research study. These requirements drove the project.

II. Software Tools Employed

The computational platform for this project was an Apple MacBook Pro running Mac OS-X version 10.6.8 with a 2.7 GHz Intel Core i7 processor and 8 GB 1333 MHz DDR3 memory. Python version 2.6.7 as well as R 64-bit version 2.14.1 software packages were installed (both free) and used for the EDA, data cleaning, and data manipulation of this project. This served as my computational interface with the Oregon Health and Science "fisher" server, a firewall- and password-secured device maintained by OHSU for housing genomic and other data files.

Python is (quoting from the python.org web site):

> "… an interpreted, object-oriented, high-level programming language with dynamic semantics... built in data structures, combined with dynamic typing and dynamic binding… for use as a scripting or glue language to connect existing components together. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed."[17]

R is an open source programming language used for data analysis and for its graphics capabilities.[18] It has numerous "packages" - software scripts created to perform various tasks and functions from the mundane to the very complex - available for free download from its website, R-project.org. One commonly-used graphics package for multivariate data employed for this project was "lattice".[19] The graphics, counts, statistics and tables for this project were produced using the lattice software package, and software code script shown with these are in R using the lattice package unless otherwise specified. "Bioconductor"[20] is a large library of bundled packages for biological and GWAS research and one of its packages, "limma",[21] is especially useful for interpreting and modeling microarray data.

III. Dataset Selection Criteria

The biological question and method of investigation (case-control study) imposes many requirements on the prospective dataset(s) to be studied. Beyond simply including age, sex, and ApoE status, the prospective study sample group cases (individuals with the condition of interest – in this case, AD) must be matched to study sample group controls (individuals without the condition of interest) for age, sex (within 5 years) and ethnicity. That is, for every case within a given stratification of age, sex, and ethnicity there is control of the same age (within 5 years), sex and ethnicity. A statistical association between a given SNP and the trait in question can be due to 1) random chance (especially when that SNP is tested repeatedly – this study proposed Bonferroni-adjusted values) or confounding due to selection bias, 2) the SNP or allelic locus regardless of any polymorphism is itself directly responsible for the trait (this study hoped to identify novel loci and SNPs associated with AD), or 3) linkage disequilibrium with another SNP or allelic locus which causes the trait (something this study was hoping to find). Matching cases and controls negates a great part of 1), that the chance that associated outcomes are the result of the matched variables or selection bias.

In order to match on age, sex and ethnicity a large GWAS study population was required. As this study was geared towards identifying very small p-value information from multiple tests (each SNP tested iteratively in pairs with every other SNP in the GWAS), every attempt was made to ensure that sample size remained as high as possible. In order to achieve this, the age/sex/ethnicity-stratified case and control groups had to be similar in distribution of the matched variables of age, sex, and ethnicity. Each individual case did not have its own individual matched control. Because of this fact, this study was not set up to be a true case-control study, statistically-speaking.

As this study focused on sporadic or late-onset Alzheimer's disease, only individuals aged 65 years or more at time of AD diagnosis were included. Any individual younger than 65 years was excluded from the study. Any individual lacking AD-diagnostic data (either AD-positive or AD-negative) was excluded from the study. There was some discussion as to whether or not to allow other (non-AD) diagnoses of dementia to be included in the control group, and whether or not to allow individuals co-diagnosed with AD plus another form of dementia (such as Huntington's disease) to be included in the case group. This was a very interesting and important question since it raised the possibility of introducing confounding any genetic information from SNPs common to AD and other dementias. It was determined that including dementia diagnoses other than AD was to be allowed only if a GWAS could not be found that had a large enough sample size after stratifying on ethnicity, age and sex, that would have enough samples in the case and control groups to have sufficient power to show associations of SNPs at the stated p-value of $1x10^{-4}$ to AD. For the purposes of this study, the GWAS was first stratified into cases and controls by excluding co-diagnosed individuals and individuals diagnosed with non-AD dementias.

Only individuals reported as "Caucasian" were included in the study so as to negate the confounding effects of population stratification. Any study individual not ethnically reported as Caucasian, including any individual missing ethnic group data, was omitted from the study.

As mentioned previously, this study would not be a true case-control association study where there are 1:1 controls to cases. In order to keep the post-stratification and cleaned dataset large enough to have acceptable power, any prospective GWAS dataset should optimally have a cleaned, post-stratified control:case ratio of between 1:1 and 4:1.[22]

One goal of the investigative research project was that as many resources as possible used for the project be found in the public domain or publicly available. This was so to enable students at any level to replicate the study, to keep costs down, and to avoid wait times for approvals and right to use applications. This goal was not without trade-offs. Older, smaller, less complete and comprehensive GWAS datasets might be available in the public domain but newer, more complete datasets might not. Genomics data analysis software available in the public domain might not be considered powerful enough. But nonetheless, public availability was initially a primary goal of the research project.

IV. Dataset Rejection – The HBTRC GWAS Dataset

The first genome-wide association study dataset considered for this project study was obtained from the Harvard Brain Tissue Resource Center (HBTRC) and made available on the Sage bionetworks repository,[23] which is maintained by McLean Hospital of Belmont, MA, as one of three federally-funded central collection and distribution sites for human brain tissue used for research. The HBTRC GWAS dataset is publicly available by online request and has been publicly available since January 2011. The newness of this dataset also made it attractive for use by this research project.

At the time this dataset was first accessed, Sage was in beta and the HBTRC dataset was just being moved onto the repository. Access was free to the public and granted by an online emailed request. This policy for data access has since changed, and a web-based tutorial regarding their rules for data use has been made available (https://sagebionetworks/jira.com/wiki/display/SWC/Synapse+Human+Data+Privacy+R ules). But at the time of my request it was determined that use of the dataset was to be that of a private (not public) access project, meaning, data access was restricted to designated investigators on the project (myself and my advisor, Dr. Beth Wilmot, only). Now, per Sage's governance, a user must submit a data sharing plan and IRB approval to Sage Bionetworks, who then determines the level of protection the dataset requires for the project: Tier 1, open and unrestricted; Tier 2, restricted per informed consent terms and data-specific terms of use, or; Tier 3, highly-controlled because it contains genotype information from living individuals or rare information or includes information gathered from vulnerable populations. If the data is not Tier 1 the user then agrees to standard use terms, as well as data-specific terms, and again receives IRB approval, after which the dataset(s) are released to the user.

As this dataset consisted of human genotype data obtained from deceased individuals it is considered Tier 1 (open, or unrestricted), and would have been considered Tier 1 at the time of my request though it was released with the understanding and agreement that the dataset would not be released to any 3rd party. At the time I requested the dataset, all I had to do was submit an online email. However, now all users must register with Synapse Commons and agree to their Standard Terms of Use prior to requesting any dataset.

All human data was de-identified before being placed in their repository and coded to ensure subject privacy per the United States' Health Insurance Portability and Accountability Act (HIPAA) guidelines.

The GWAS was conducted on approximately 800 individuals (approximately 400 AD cases, 230 Huntington's Disease cases, and 170 controls). These were matched for age, gender and post-mortem interval. These individuals were also to be well-characterized for phenotyped clinical data, some of which this project would potentially utilize (age at disease onset, and Braak scores as an indication of neuronal damage, for example). Study individuals were genotyped on two different platforms: the Illumina HumanHap650Y array, and a custom Perlegen 300K array focused on the detection of what the study termed "singleton SNPs." Clinical data included in the dataset consisted of age at onset (AaO), age at death (AaD), Braak scores (for AD+ individuals), Vonsattel scores (for Huntington's + individuals), and scores denoting regional brain enlargement or atrophy.

The HBTRC files were downloaded initially onto the laptop's hard drive as I did not have access to the secure server. However, the files were moved onto the OHSU fisher server after access had been granted and my secure directory created, and the files were removed from the laptop.

Python was used to open and read the files. Genotype information was stored in two files, one for the data obtained from an Illumina.

Files for the HBTRC study were all in plain text (.txt) format with the exception of the Sage Bionetworks user agreement, which was in portable document format (.pdf). The data of interest to this research study were found in the following directories and files:

```
hbtrc/
        individuals.text
        readme.txt
        /phenotype
```

```
                    pheno.txt
                    phenotype.txt
                    phenotype_description.txt
                    phenotype_notes.txt
        /genotype
                    illumina/
                                genotype.txt
                                marker.txt
                    perlegen/
                                genotype.txt
                                marker.txt
```

The “readme.txt” file contained the data file structure and file names as well as an overview of the HBTRC study. It was the roadmap for which files to search for information.

The “individuals.txt” file contained column headings with data in tab-delimited format. The fields of interest to this project were ‘individual_rid’ (numbered 1 through 803) and ‘mrl_tid’ (both identifiers for the study individuals), “phenotypes” (meaning, had phenotypes been collected and recorded for that individual) and “genotype” (meaning, had genotyping been completed and recorded for that individual). R uses indexing beginning with 1 (unlike Python, which uses zero indexing) so these were columns 1, 2, 6, and 10.

The “pheno.txt” was a tab-delimited file containing 43 column headings, each with a heading identifier number. In order to ascertain the information in each column it was necessary to refer to the “phenotype_description.txt” file (for example, header identifier “1029” was for patient gender). The “phenotype_description.txt” file was referenced in order to determine which “pheno.txt” columns were needed for this project.

I created an “importData” script. Using R, I first imported (or, ‘read’) the HBTRC “individuals.txt” file as a table I named “ind” – this file contained the “RID”, the unique identifier number, for each individual in the HBTRC study. The RID was used as the ‘key’ to match data to individuals from the different HBTRC files. I next read the HBTRC phenotype/phenol.txt file as a table which I named “pheno” – this file contained the RID as well as each phenotypic marker used by the HBTRC study (by identifier number) and its value for each individual. These were ordered numerically L-R by phenotypic marker identifier number as the row headings, such that each column was the value for one particular phenotypic trait for one individual. In order to make the “pheno” file usable to humans it needed a key in language, other than simply numeric codes and abbreviations. The HBTRC dataset had such a file, the phenotype/phenotype_description.txt file. This was read in and given the variable name pheno.key. I then created a variable called pheno.translate, which took the “trait_name” entries on my pheno.key file and converted them to characters instead of strings. I then created a names file from the pheno.translate file, pasting each trait_id entry as a name. I then created a variable ind.pheno for each individual’s phenotype, which meshed the ind individuals text file I had created with the pheno phenotype file I had created, but now using text descriptors and names for headings.

I then created geno.ill and geno.ill.key variables, which stored (as tables) the Illumina genotype information and the Illumina genotype marker keys, respectively, so that these could be queried at a later time. When the “importData.R” file was run it created and merged these files, “ind.pheno”, “geno.ill”, “geno.ill.key”, saving them as the ‘hbtrc.RData” file I could then examine and query. The “hbtrc.RData” file had each individual’s RID, their phenotypic traits (column headings by name) with human-readable values for each entry, with that individual’s Illumina genotypic information along with the Illumina marker key for each SNP. This was what I considered a rather massive file, but it had most of the information I would need for the investigative project.

When Sage Bionetworks provided an updated phenotype text file (which included gender) I took the opportunity to fix a problem my first “importData” file had, which was, it created in table form a file in which each subsequent file read and meshed was read down the page from top to bottom. In other words, the file was “sideways” to a human reader. It was easier to understand (and to find data) for me if the resulting table had each individual’s information from left to right by row. So I created an “importData2” file using the “hbtrc.RData” file resulting from my “importData” file.

To do this, I first loaded “hbtrc.RData” and created a variable called pheno2 which simply was the table created by reading the HBTRC phenotype/phenotype2.txt file. Then I started “juggling” this file. I first transposed the pheno2 rows to columns, calling this pheno3. Then I converted pheno3 into an R data frame I called pheno4. A data frame is a table, a 2-dimensional array, where each column is for one variable and each row is an individual case. So the phenotypic data was now in the format I wanted. But it was still lacking usable names for rows and column headings, and I wanted each row to begin with the individual’s RID instead of the HBTRC’s phenotype variable number identifiers. And I wanted to use the HBTRC phenotype key names for the phenotype variables, not number identifiers. Again, I was attempting to create a single data frame, readable by a human, for the project.

As in my “importData” file described above, I read the HBTRC phenotype/phenotype_description.txt file as a table and gave it the variable name pheno.key. I then created the variable pheno.translate by converting the pheno.key trait_name column entries, casting them as characters, and the pheno.key trait_id column (from the pheno.translate variable) became the actual name (in English), a translation from a number (HBTRC’s phenotypic identifier) to a name.

Next I had to slice the pheno4 file such that the second (l-r, starting at 0) column in the data frame table was the individual_rid values, and salled that pheno5. I then sliced the first row off the pheno5 data frame, saving those values as rownames. This allowed me to have a row names key, but to not have to worry about “skipping” the 1st row when querying the pheno5 data frame – I would have only data in each row and column, not the names of the column headers as the first row.

Reading a file (importing) as I’ve described results in all data in the file being seen as strings to R. All numeric data get formatted to a string, and no arithmetic operations can be performed on strings. The HBTRC dataset contained lots of numeric data. In order to convert the (originally) numeric data from strings back to numeric values I created a variable called numerical which first examined the first row of the table for fields, which could be represented as a numerical value. Once these were found I looped over each row, changing the appropriate columns from string to the numeric type using the ‘as.numeric’ function. Since I was over-cautious about keeping earlier working versions of script variables, I copied pheno5 as pheno6, and ran a nested “if” loop (`if (numerical[i]) )` for each entry in each column of pheno6 which translated every field in each column that could be represent as numeric values as numeric values in pheno6.

Once this was completed, I read the individual2 data as a table called ind. I then merged the individual and the phenotype data together (ind merged with pheno6) into what I called ind.pheno. I then created a near duplicate of this ind.pheno file but which removed any Huntington’s-positive (H+) individuals from the data frame and called this ind.pheno2. I then saved the following variables: ind.pheno, ind.pheno2, pheno.key, geno.ill, geno.ill.key and saved the file as “hbtrc2.Rdata.” In this way I had an “all subjects” data frame (ind.pheno) for general, full HBTRC study population EDA; a “no Huntington’s patients” data frame (ind.pheno2) for simplifying queries made to the Alzheimer’s and Control population for this project (so I did not have to write script to check for H+ subjects in future queries, as I had a data frame with those individuals removed); the pheno.key as the key to the names HBTRC gave their phenotypic data; the Illumina genotypes as geno.ill; the Illumina SNP identifiers as geno.ill.key. These were the data I needed to conduct the project. There were no mis-matched data frame sizes resulting from any of the merges (meaning, no rows or columns were lost or omitted). It was time for EDA on the full HBTRC dataset (ind.pheno) and the non-Huntington’s patient dataset (ind.pheno2) to obtain some descriptive statistics and ascertain the usability of the HBTRC dataset for my project.

I loaded the lattice library into R. A quick EDA inspection of the “pheno.txt” file first downloaded from the Sage repository revealed that very few of the subjects had sex recorded.

```
> table(ind.pheno1$gender, useNA = "ifany") # count by gender of 1st
HBTRC dataset
   F    M <NA>
  88   83  632
```

That represents 632 out of the total 803 individuals missing gender, or 78.70 percent.

When this was brought to the attention of Sage they quickly provided a “phenotype2.txt” file, which contained sex for the subjects and was identical otherwise to the original “phenotype.txt” file. All of the following is derived from that second HBTRC phenotype2.txt file provided from Sage.

```
> table(ind.pheno2$gender, useNA = "ifany") # count by gender of 2nd
HBTRC dataset

  F   M
394 409
```

A summary command (summary(ind.pheno2)) of the entire ind.pheno2 data set shows there were 803 individuals, and all 803 had phenotype information.

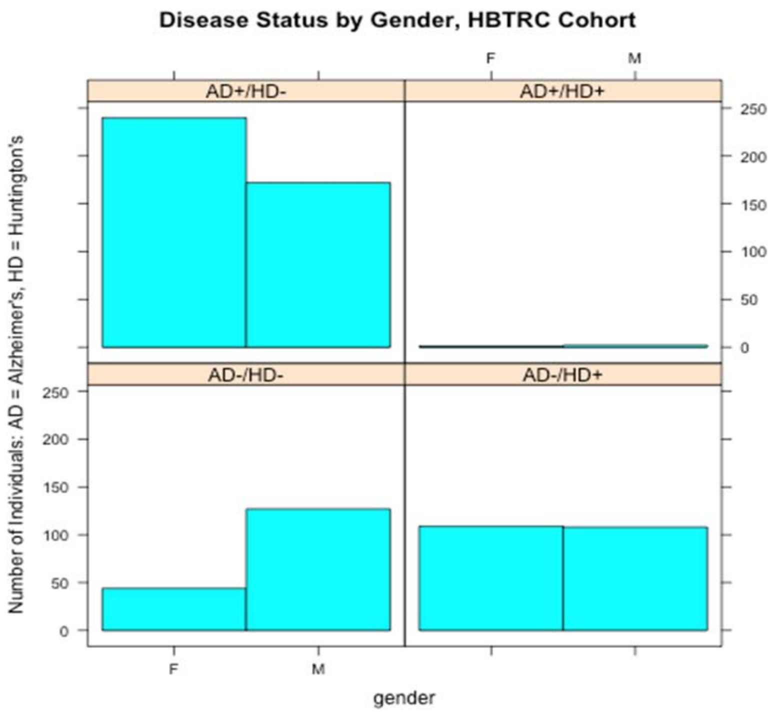
The HBTRC study focused on both AD and Huntington’s Disease (HD), and the study cohort could be broken down by sex into the following groups: AD+/HD- Males, AD+/HD- Females, AD-/HD+ Males, AD-/HD+ Females, AD+/HD+ co-morbid males, AD+/HD+ co-morbid females, and AD-/HD- Males, AD-/HD- Females (these AD-/HD- groups would be considered controls for this project).

The cohort could be broken into groups by their disease status (the presence or absence of Alzheimer's and/or Huntington's) by gender:

```
> by(ind.pheno2$gender, list(ind.pheno2$diseaseStatus), summary,
simplify = FALSE) # summary disease  status (AD+/-, HD+/-) by gender
: AD-/HD-
  F   M
 44 127
------------------------------------------------------------------------
----
: AD-/HD+
  F   M
109 108
------------------------------------------------------------------------
----
: AD+/HD-
  F   M
240 172
------------------------------------------------------------------------
----
: AD+/HD+
F M
1 2
```

>

This data is more easily absorbed when displayed as a histogram:



So of the 803 individuals, a total of 3 (1 F, 2 M) were co-morbid with both Alzheimer’s and Huntington’s disease (AD+/HD+), and 217 (109 F, 108 M) had Huntington’s only (AD-/HD+). These 220 individuals (27.40 percent of the total HBTRC cohort) would not be included in the Alzheimer’s Disease project, leaving an HBTRC subset of 412 AD+/HD- Alzheimer’s patients (240 F, 172 M) against 171 AD-/HD- controls (44F, 127 M).

This would result in a diseased:control ratio of 412:171 or 2.41:1 when gender was not considered. With gender was considered in the diseased:control ratio, the results are 240:44 = 5.45:1 F and 172:127 = 1.35:1 M. The “Disease Status by Gender” histogram (above) points out a potential issue with using the HBTRC dataset for this project: Male controls (127) far outnumber female controls (44), for a ratio of 2.89 male controls for every female control, nearly 3:1. In a random sampling of a population, we would expect a ratio closer to 1:1.

The age at death for the HBTRC cohort was as follows:

```
> summary(ind.pheno2$age_death) # age at death, HBTRC cohort
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  18.00   60.00   73.00   70.65   83.00  106.00
```

The age at onset for the HBTRC cohort, which includes Huntington’s diseased individuals as well as Alzheimer’s, was as follows:

```
> summary(ind.pheno2$age_onset) # HBTRC cohort
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   6.00   33.75   42.00   41.66   50.00   81.00  671.00
```

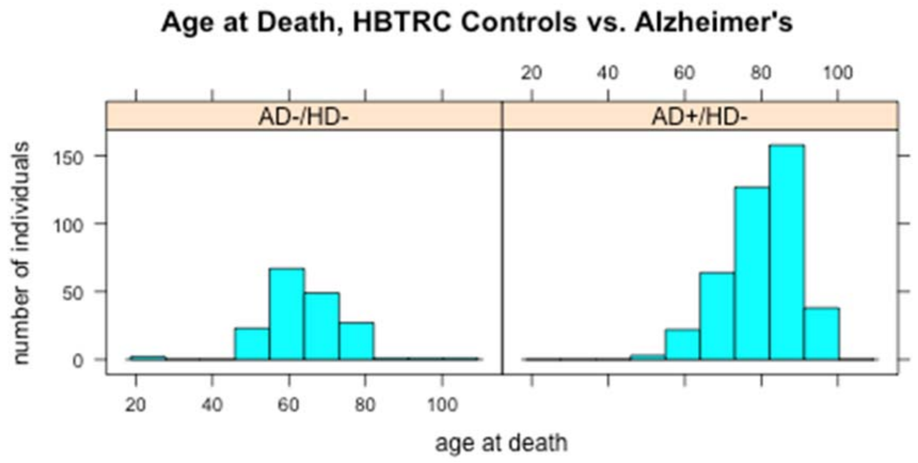
The age at disease onset can be further explored by gender and disease:

```
> by(ind.pheno2$age_onset, list(ind.pheno2$diseaseStatus,
ind.pheno2$gender), summary, simplify = FALSE) # summary age onset by
disease status and gender
```

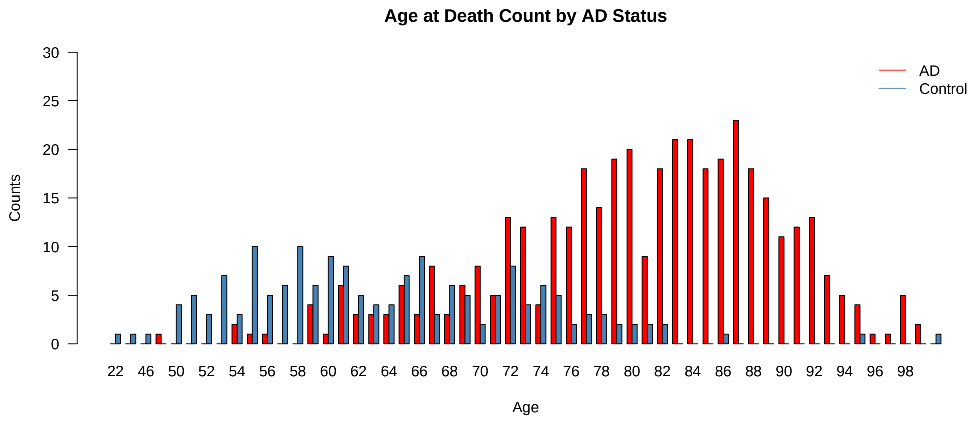
```
: AD-/HD-
: F
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
                                                     44
--------------------------------------------------------------------------
----
: AD-/HD+
: F
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  10.00   29.00   40.00   39.02   48.00   74.00   43.00
--------------------------------------------------------------------------
----
: AD+/HD-
: F
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
                                                    240
--------------------------------------------------------------------------
----
: AD+/HD+
: F
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
     68      68      68      68      68      68
--------------------------------------------------------------------------
----
: AD-/HD-
: M
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
                                                    127
--------------------------------------------------------------------------
----
: AD-/HD+
: M
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   6.00   35.00   45.00   43.87   53.50   81.00   45.00
--------------------------------------------------------------------------
----
: AD+/HD-
: M
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
                                                    172
--------------------------------------------------------------------------
----
: AD+/HD+
: M
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   43.0    44.5    46.0    46.0    47.5    49.0
>
```

This table points out an interesting fact about the HBTRC dataset: There is no age at onset data recorded for any AD+/HD- patient, female or male (refer to lines 3 and 7, highlighted above).

A further investigation of the age at death for the Alzheimer's group compared to the control group resulted in the following histogram:



Viewed another way, with all ages shown, results in the following:



Unfortunately, this EDA exercise revealed two problems with the HBTRC dataset as regards this project: 1) age at onset was lacking in the Alzheimer's (AD+/HD-) cohort, and 2) when the "control" subjects are compared to the AD+/HD- subjects by age at death, it becomes apparent that the groups are <u>not</u> in fact age-matched. After discussing this fact with my advisor it was quickly decided that another dataset should be found for the study and to abandon any further investigation of the HBTRC dataset.

V. Dataset Acceptance – The ADNI GWAS Dataset

Another GWAS dataset was selected for consideration for this project, that of the Alzheimer’s Disease Neuroimaging Initiative (ADNI).[24] Access to their data is by approved application only (application links available from their website).[25] The ADNI Data and Publications Committee (DPC) states that they wish for the ADNI data to be available to the general scientific community and that they do not judge who is or is not “qualified.” They base each decision to share their data by making a judgment of each applicant’s affiliation with a scientific or educational institution as well as the reason for the request. They do not monitor IRB compliance though requestors must provide IRB numbers and expiration dates. The DPC recommends full and open access of all de-identified ADNI imaging and clinical data to any who register with ADNI, agree to the “ADNI Data Use Agreement” and who pass the limited screening of the DPC. The request and data use agreement are web-based. In order to facilitate investigative collaboration, applicants area asked to update their information annually to include titles of manuscripts being developed with their status as well as lead author’s contact information, citations of each published manuscript which used ADNI data, plus an uploaded file of any published manuscript(s) which used ADNI data. Failure to comply with the required annual updates jeopardizes further access to ADNI data. Any requestor of ADNI data agrees that the purpose of their request is scientific investigation, teaching, or planning of clinical research. They agree that: 1) they will not attempt to establish the identity of or contact any ADNI subjects; 2) they will not make or attempt direct contact with ADNI primary investigators or staff concerning specific results of individual subjects; 3) they will require any person they authorize to use, or any they share the ADNI data with, to adhere to the Data Use Agreement; 4) they agree that use of the data beyond the uses outlined in the agreement, and redistribution of the data in any manner, is prohibited; they will accurately provide requested information for the person(s) who use the data and analyses planned using the data; 6) they will promptly and accurately provide annual information updates; 7) they will comply with rules and regulations of their respective institutions and its review board in requesting the data.

My advisor, Dr. Beth Wimot, has been granted approval to use the dataset in controlled environments – meaning, this is not a public domain dataset. This is in compliance with current GWAS data use standards.[26] In order to gain access to the dataset it was required that I be given virtual private network (VPN) access with a password onto OHSU's secured server, with access to OHSU's network using my existing student network ID. I was granted a directory on the secure server on which to store my data and scripts, etc. To do so, I was included in Dr. Wilmot's Data User Agreement as though I had signed the agreement myself. Pursuant to the Data Use Agreement, it was agreed that at no time would any of the actual ADNI dataset genomic information be removed from the OHSU server. I could remove "sand box" datasets – small datasets produced from the original and containing minimal genomic information but lacking some data fields, used to test script performance and file outputs, etc. – and my own output files created from the original ADNI dataset so long as it contained no original genomic information. I was very diligent to adhere to this agreement.

In order to VPN onto OHSU's server and work remotely, taking information from one directory on that server and manipulating it within another directory (my directory on the same server) or on my own computer and transferring data files back and forth, I chose to employ Cyberduck for Mac (available at http://cyberduck.ch). Cyberduck is an open source file transfer protocol (FTP) client and secure shell file transfer protocol (SSH file transfer protocol, or SFTP) which allows drag and drop file transfer functionality. It is freely available to the public for download (GNU general public license). Cyberduck made working between a local computer and different remote server directories as simple and intuitive as using a local computer with an installed thumb drive.

Workspace and file organization is important in any project, both for the worker and for any individual who might have to access the files during or after a project. The naming and filing system can be any, so long as it works for the user(s) and aids in locating files. I created four subdirectories on my secure server directory, for code (any scripts related to the project), data (text, tab-delimited, spreadsheet, etc. files which resulted from my scripts acting on the original dataset files, and also the cleaned data files I used for the project), graphs (self-explanatory), and text (for any text files which did not result from a manipulation of data, such as my own notes).

The ADNI Files:

Opening the shared ADNI file revealed hundreds of comma separated value (.csv) files – one for each subject of the ADNI GWAS. Each subject file was identified by the study subject unique numerical identifier. The vast majority of these files had the terminal extension ".gz", indicating they were GNU zipped archive files. However, a few of the files had the terminal extension ".zip". This was an important observation since opening and extracting .gz files is different from opening and extracting .zip files since uncompressed .gz files result in a single uncompressed file while uncompressed .zip files result in (usually) more than a single file, only one of which is the actual data. In addition to the subject files, the main ADNI directory contained three Illumina-specific .pdf (Adobe portable document format) files. This revealed, without having to research, that the ADNI study was done at least in part using Illumina (www.illumina.com/) micro-array technology. Also within the main shared ADNI directory was an "adcs_clinical" directory. I looked here first.

At the bottom of the list of .csv files in the shared/ADNI/adcs_clinical directory was the README.txt file. This contained a notice regarding use authorization, where to address questions regarding clinical data and about supplementary information about the ADNI project, a brief description of the contents of the ADNI study including its DATADIC (data dictionary) table, a notice regarding a known MicroSoft Excel issue reading the .csv files, and a list of data field and file name definitions for the ADNI files. As the README.txt file was the list of files names and the DATADIC. (adni_datadic_2009-09-01.csv) was the data dictionary for this project and would be referenced repeatedly, these were the first files I copied into my server directory.

The datadic file contained 2,700 "ID" entries. Each ID was for a field name and file name (found in the README file). Referencing column headings within the files would be by ID number, not by text. Where applicable, each listing contained information about the data: its type (Numeric, Text, Date, or TimeStamp), the units used and a description of the units (e.g., the range and definition of the values used, or 1 = yes, 0 = no). Referencing the logistic regression equation for the required data, I found the appropriate field names as well as the ADNI subject files where those field names/IDs could be found by inspecting candidate files found on the README and datadic lists.

I employed three data files from the adcs_clinical directory: “adni_apoeres_2009-09-01.csv”, “adni_pdxconv_2009-09-01.csv”, and “adni_ptdemog_2009-09-01.csv” to supply the data necessitated by the biological question.

The first adcs_clinical file, “adni_apoeres_2009-09-01.csv”, contained the APOE haplotyping information for the ADNI study cohort. Each row represented information from a single study subject, sorted by an ID number. Each row also had the study subject RID, which would be used as a key to link information within the file data fields to the proper individuals. This file had each of the two APOE genotypes for each patient in two fields, titled “APOEGEN1” and “APOEGEN2”. The values stored in those fields were the numeric values 2, 3, or 4, representing the genotyped APOE isoforms: APOE-ε2, -ε3, or –ε4. Other data fields contained within this file were not considered relevant for this project.

The second acds_clinical file, “adni_pdxconv_2009-09-01.csv”, contained the summary indexes for various clinical diagnoses for each of the ADNI study cohort. This file was arranged by row with each row representing a different visit/diagnosis date. Each visit was given an ID, and each visit was associated to an individual by RID. Each patient could therefore have more than a single row entry in this file, as diagnoses or severity of diagnoses could change for over time for any individual study patient. This file had 40 data fields per row, 36 of them being diagnosis information. The majority of the valid recorded information was entered as numeric data (ranging from 0 through 4), some was timestamp, and some was text string. This was the first file I encountered which had a preponderance of “-4” field entries, which denoted “N/A” or missing data. Of the 36 diagnostic data fields, three were of particular interest related to my project. These 3 were titled “DXAD” (Alzheimer’s diagnosis: numeric data where 1 = ‘yes’, -4 = ‘N/A’ or none), a field for the cause of the patient’s mild cognitive impairment (MCI) titled “DXMDUE” (also a numeric data field: MCI diagnosis due to Alzheimer’s = 1, due to other etiology = 2), and a field titled “DXADES” which rated the severity of a patient’s Alzheimer’s disease (numeric data: 1 = mild, 2 = moderate, 3 = severe). These fields are of interest because they relate directly to any diagnosis of Alzheimer’s, it’s severity (and change over time), and the severity and cause of MCI over time. A patient was considered to be AD-positive (AD+) at any occurrence of a “1” in the “DXAD” fields for that given patient. This “DXAD” field was directly used in my study project. MCI was of interest for future studies and so the other two fields, “DXADES” and “DXMDUE”, would be included and utilized in my Python scripts and the resulting data from these fields and some of my output included or used data from those fields.

The third and last adcs_clinical file I employed, “adni_ptdemog_2009-09-01.csv”, contained demographics information on the ADNI study cohort. This file consisted of 23 columns, with column headings, each row pertaining to a specific study individual. Each row was identified by a unique ID in the first column. The patient RID was provided in the second column. Unlike the “adni_pdxconv_2009-09-01.csv” file, which had multiple entries for each RID depending on the number of visits, the “adni_ptdemog_2009-09-01.csv” file had only a single row of data field entries for each patient RID. The data types included numeric, text string, and date (some in “month/day/year” format (1 or 2 digits each), some as “/year” format (4-digit)). As on the “adni_ptdemog_2009-09-01.csv” file, the numeric value “-4” was used to denote “missing” or not applicable” data. Referring to the datadic in order to interpret the column headings, the columns needed were “RID” (numeric data: individual patient identifier number used throughout the ADNI study), “PTGENDER” (numeric data: patient gender where 1 = male, 2 = female), “PTDOB” (date in “/year” format: patient year of birth), and “PTETHCAT” (numeric data: patient’s ethnic category where 1 = Hispanic or Latino, 2= not Hispanic or Latino, 3 = unknown) plus “PTRACCAT” (numeric data: patient’s race). In order to avoid racial confounders it was determined that my project should focus only on non-Hispanic whites. Using “PTETHCAT” and “PTRACCAT” in concert I could determine any non-Hispanic (“PTETHCAT = 2) white (“PTRACCAT = 5).

In order to determine age at death, “PTDOB” would have to be converted to numeric data and subtracted from 2009, the year of the ADNI study. One other field of interest was labeled “PTADBEG” (numeric data: the year the patient’s AD was determined to have begun). This would not be used directly in my project but may have been of interest for further study.

VI. Exploratory Data Analysis

A. Scripts for the ADNI Dataset – Demographics

My first task was to create a demographics matrix for the entire ADNI cohort. The scripting language I selected to create this file was Python.[17] I called this script "*ForADNIDemoMatrix.py*". This script would create a master demographics matrix of ADNI phenotypes and would include each participant's RID, exam date, gender, DOB, ethnicity and race, and APOE allelic types from combining APGEN1 + APGEN2 as described above.

This script included a files counter so that I could ensure that all files were "read" and processed by the script. It then opened the "adni_ptdemog_2009-09-01.csv" file, to which I gave the variable name, "DEMO." It then opened the "adni_apoeres_2009-09-01.csv" file, to which I ascribed the variable name, "APOE." I then opened the "adni_pdxconv_2009-09-01.csv" file, giving it the variable name "AD_MCIstatus."

Python differs from R in how the programs open and read files, such that in Python the opened file must explicitly be "read" and that read file must be given a variable name. I read these opened files, and gave them the variable names "ReadDEMO", "ReadAPOE", and "ReadAD_MCIstatus" rescpectively. In order to remove the extra new line character Python inserts at the terminus of every read file, each "Read…" variable was split (sliced) on the new line character ('\n') beginning at index 0 and ending on the last item in that list. This split each line of text in the files into a single entity, divided by a new line character. This split version of each file was given the variable name "Split...". Python uses zero indexing, meaning the 1st item in a list is indexed as zero. In R, the 1st item is indexed as one. By slicing on every field except the last one, the final new line character is not included in the “Split…” version of each file. The final line of information is maintained but it is not terminated with a new line character, so that an empty line is not created after the final line of information.

In order to have the ADNI field headers for my reference, I next sliced just the first (zero index) line from the SplitAPOE and SplitDEMO files and named these variables APOEheader and DEMOheader. As these files were saved by ADNI as comma separated value (,csv) files, I split each line of the headers by the comma character. For my project files I did not need every field, only a few. So I examined the original ADNI .csv files to determine precisely which headers I needed – which columns of data my project required – and wrote those down. I called each split header an “item”. From the APOE file header I needed only the 4th and 5th “items” and so I created a variable “APOE_HEADER” which included only items 4 and 5 (remembering Python’s zero indexing, these would be coded as item[3] and item[4]). Those represented the APOE allelic pairs for each patient (APGEN1 and APGEN2). I required more header fields from the demographics file including RID, ExamDate, Gender, DOB, Ethnicity, and Race. These equated to (using zero indexing) items [1], [3], [5], [6], [21] and [22]. My FINAL_HEADER was the combination (concatenating or joining) of the DEMO_HEADER plus the APOE_HEADER. I wanted to add my own header fields (MCI Status (DXADES), and the reason for MCI (DXMDUE) fields), so by concatenating using the comma character I added the strings, ‘AD Status’ and ‘MCI(AD/other/none)’ to the end of the FINAL_HEADER. The new header for my demographics matrix now included column headings in this order: RID, ExamDate, Gender, DOB, Ethnicity, Race, APOE-1, APOE-2, AD status, and MCI(AD/other/none).

Saving the new demographics header as a variable unto itself allowed me to delete the header lines from the Splitxxx files while retaining the header I wanted in my final file which I would add after processing and creating the file. This made processing these files (especially looping over them) easier because my script code would not have to account for any non-data lines. As each ADNI file had a header as its first row (index 0), I merely had to delete each Split…[0] to accomplish this. I could now process each Split… file without having to account for its header.

My concept of creating my master demographics and APOE files for this project was by utilizing data dictionaries. I created a file handle “fh” to open and write to my file, “ADNIDemoMatrix”. The key for each data dictionary would be each patient’s RID. The values stored would be PTGENDER (patient gender), PTDOB (patient date of birth), PTADBEG (patient AD beginning date) from the “adni_ptdemog…” file as well as APGEN1 and APGEN2 from the “adni_apores…” file. I looped over each line in the SplitDEMO variable and created an “item” by splitting on each comma character. Each line of SplitDEMO was now a separate item and could be sliced using indexing. I then rebuilt each line the way I wanted it, to match the data dictionary format I described, by populating the DEMOdict by items [1], [3], [5], [6], [21], and [22], each item separated by commas. I repeated this procedure for the APOEdict, using SplitAPOE and populating items [1], [3], and [4]. Item [1] was the patient RID for both of these files – my data dictionaries key.

I next wanted to add the AD status to the end of the existing DEMOdict. This data would come from the SplitAD_MICstatus variable I had created, using items [18] (DXMDUE) and [21] (AD ‘T/F’). I created an “AD_MCIdict” data dictionary to contain this data. Because this data came from (potentially) different diagnoses from multiple patient visits, I had to come up with a looping structure in my script to update status conditions if and when they changed. I determined that once any patient had a diagnosis of MCI due to AD from any of their diagnostic visits their MCI cause status would not revert to non-AD, even in the unlikely event that their diagnosis had changed from AD to non-AD from their visit history. My reasoning was, 1) AD is officially diagnosed post-mortem from brain tissue pathology [8, 9] and so a pre-mortem diagnosis was not as informative as a post-mortem diagnosis, and 2) as this project was a “fishing expedition” for associating SNPS with AD I needed to capture as many potential AD patients as possible for the project and so I cast a “wide net” as regarded pre-mortem diagnoses and so I included any patient whose MCI was ever diagnosed as being caused by AD.

I created a variable “ADstatus” and set it’s starting condition to 0. Next I created a variable “MCIstatus” and set its starting condition to 0. This MCIstatus variable could have 3 possible values: ‘0’ (data missing), ‘1’ (MCI was diagnosed as being caused by AD), and ‘2’ (MCI due to something other than AD). As I described previously, if this MCIstatus ever got set to ‘1’ it would not revert to ‘2’ or ‘0’ on subsequent loops. I then created a variable “previousRID”, which I set to 0 as its starting condition, so that I could track (by RID) when a series of visits concluded for any one RID and the next series of visits began for the next RID. This "previousRID" became the triggering condition for entering, re-entering, and exiting the MCIstatus update loop.

In order to loop over the lines in SplitAD_MCIstatus, I first created an "item" by splitting each line on a comma character. I set a "currentRID" variable to item[1] and checked (using 'if') to see whether the currentRID was exactly equivalent to the previousRID. Nested within this 'if' loop were the status update loops. The first checked 'if' item [21] (the ADstatus field, whether the patient was diagnosed AD+ or not) was equivalent to '1', and populated an ADstatus variable with a '1' if ADNI reported the patient as AD+. The possible values for this field were '1' (AD positive) and '-4' (AD negative, or missing data). I next checked 'if' item [18] (DXMDUE) value was equivalent to '1'. The possible values for this field were '1' (MCI due to AD), '2' (MCI due to other than AD), and '0'. Missing data was recorded by ADNI as '-4' but I had forced it to have a value of '0' as the starting condition for this series of loops. The value was recorded in a MCIstatus variable. As described above, once this MCIstatus variable contained a value of '1' it would not revert to any other value. I then included another 'if' condition so that MCIstatus would not revert or downgrade from '2' (MCI due to non-AD) to '0'.

If an item had bypassed the first loop (meaning, all processes had been completed on all instances of the current RID) and the current RID was not equal to the previous RID, a nested 'else' loop updated the AD_MCIdict dtata dictionary with the RID obtained as the next item[1], an 'if' checked to if the value of the MCIstatus in item [18] was '-4' and, if it was, set that value to '0' and, (in the next 'if' check) if it was not '-4' it set the value to whatever value was entered by ADNI in the MCIstatus field (item [18]). This series of checks was then repeated for the ADstatus fields for item [21]: if entered by ADNI as '-4', change the value to '0' or, if not entered as '-4', to whatever value ADNI entered into the ADstatus field. This completed the creation of the AD_MCIdict data dictionary. This dictionary was then "written" to the computer screen using a standard error write command from the Python "system" (sys) library package in order to see what was being stored, what the data dictionary looked like, to ensure functionality of the script.

The DEMO, APOE, and AD_MCIstatus file handles were closed (good scripting practice as it removes unnecessary files from active memory) and the FINAL_HEADER (plus a new line character, so that more lines could be written) was written to a file handle variable I named simply, "fh". A 'for' loop with nested 'if' and 'else' conditional loops was created, using each key in the sorted list of the set of keys from the demographics data dictionary (DEMOdict.keys). Each key had to also be a key in the set of sorted set of APOE data dictionary keys (APOEdict.keys) as well as the sorted set of AD_MCI data dictionary keys (AD_MCIdict.keys). This was done as a check to ensure that only RIDs found in all three files would be recorded into the demographics master matrix, in case there existed an instance of an RID which only appeared in one or two of the ADNI files but not all that this project required. This was done so that I did not have to keep track of which, if any, RIDs were found only in certain files and not others, and so avoid data mismatches in my final demographics matrix. Every RID entry would have the same number of demographics, AD status and AD_MCI status fields, even if they were blank. I first wrote the current key (RID) to the screen to see that the script was updating through the RIDs in sequence. I created a variable called DEMO_info which stored the demographics data stored as values associated with the current key (RID) from the demographics data dictionary (DEMOdict). The first ‘if’ checked that DEMO_info was an empty set (meaning, no demographics data for that RID) and, if it was, to write three empty comma-separated fields. If DEMO_info was not an empty set, it wrote the DEMO_info (demographics data) and then retrieved the APOE_info (APOE value associated with the same RID key). If that was an empty set, two empty comma-separated fields were written. But if APO_info had values associated with that RID, they were then written (appended) as the next values in the demographics matrix and the AD_MCI_info values associated with the AD_MCIdict data dictionary were retrieved and compared to an empty set. If there were no values in the AD_MCI data dictionary for that particular RID, a single empty comma-separated field was written to the demographics matrix for the AD_MCI portion. But if values existed in the AD_MCI data dictionary, those values were appended to that RID’s matrix entry, along with a new line character. In this way, the file handle “fh” was populated, row by row as ordered RID by RID, with first the demographics data values appended, then the MCI status values appended, then AD status values appended to each RID row before entering a new line character and starting on the next ordinal RID. In this way I had created my master demographics matrix, “ADNIDemoMatrix”, which was a comma-separated value (.csv) file. This file would be queried for this project. At this point, this file contained demographics information for all ADNI participants and these entries could be missing data necessary to the investigative project or have data not required by the project. A snapshot of the final output ADNIDemoMatrix file at this point is shown here:

RID	EXAMDATE	PTGENDER	PTDOB	PTETHCAT	PTRACCAT	APGEN1	APGEN2	ADstatus	MCI(AD/other/none)
1	8/18/05	2	/1944	-4	-4				
10	10/26/05	2	/1931	2	5	3	4	1	0
100	12/12/05	2	/1930	2	5	3	4		
1000	10/11/06	1	/1924	2	5	3	3		
1001	11/6/06	1	/1937	2	4	3	4	1	0
1002	10/13/06	2	/1930	2	5	3	3	0	0
1003	10/13/06	1	/1935	2	5	3	4		
1004	11/10/06	-4	-4	-4	-4	3	4	1	1
1005	10/18/06	1	/1947	2	5	3	4		

Note that this sample has many missing fields as well as examples of ethnic and racial categories, which would be excluded from the final study.

It is noteworthy that, while creating this project write-up, a simple “eyeballing” of the expected output from my master matrix revealed an error in the script. Everything appeared to be operating as planned except the AD_MCIstatus field was not being updated from ‘0’. This bug was soon and easily fixed. But it demonstrates that simple “does this make sense” investigations of the output of a script is a necessary part of creating functional and usable output.

B. Scripts for the ADNI Dataset – Genomics

My next task was to create a master matrix for the genome information of all the ADNI participants’ genomic files. This was done using Python as the scripting language, and I called the script “ForADNIMstrMatrix4.py” as it was the 4th iteration of the script that finally worked properly. As mentioned earlier, the ADNI genome files were stored in one of two formats: as .zip files (very few and easily overlooked in such a large list) or as .gzip files. This was an important observation, as these files must be handled differently for Python to be able to open and process the information contained in the different formats. I downloaded and then imported into Python the existing libraries for such tasks, “gzip”, “zipfile, and “zlib” along with “sys”, “glob” for error reporting and general file handling functionality, plus “os” for path splitting.

I first defined a function, “get_compressed_filename” which would allow me to input the directory and path to the compressed file(s), would recognize that their output type should be .csv when they were uncompressed, and would return the path plus file name, split the ‘.’ in it’s file extension, and give it the .csv file extension plus the extension of whatever compression type it originally was (/home/shared/ADNI/141_S_1004.gz became /home/shared/ADNI/141_S_1004.csv.gz, as an example). I created a variable “list_of_files” which was simply a holding place for the file names without the directory path (for the example given, 141_S_1004.gz.csv). I checked the total number of files in the list_of_files by querying its length – this “totalGeneFiles” should have equaled 818, the number of genomic files in the ADNI study. As part of my error reporting as well as progress log, I used the sys.stderr.write command on the totalGeneFiles variable (as a string data type, so that I could use string concatenation) to report the total number of gene files.

I noticed that all of the ADNI genomic files followed the same naming convention: a series of three numerals followed by an underscore followed by a capital letter followed by another underscore followed by four numerals prior to the period and compression type extension. The series of four numerals was the RID. In order to extract the RID from each file name I created an “unorderedRIDs” variable which used string slicing to split the file name at positions [6] through [10]. Remembering Python’s zero indexing, this would be the 7th through 11th characters when read left to right. Python slicing nomenclature means to begin and include the first index shown, and to stop prior to the last index shown. This means that positions [6] through [10] would actually be recorded – the four digits of the RID. I included an in-line “for” loop to do this for each file (by file_name) in the list_of_files. I used this to populate a data dictionary called “RIDtoFilenameDict” which used the unorderedRIDs as its key and list_of_files as its values. I then put the RIDs in numerical order, which I called “orderedRIDs”, by using Python’s ‘sorted’ command on the unorderedRIDs variable. I now had a way to track RIDs in numerical order.

It was not enough to use RIDs from the file names. I had to check those against ADNI’s patient roster of RIDs. I had to also handle each file differently depending on what type of compression was used to store it (file extension ‘.gxz’ or ‘.zip’). I also needed to ensure that the genome matrix I was creating allowed for files to be missing calls on SNPs (in other words, files of differing lengths).

I started by creating a “files_counter” with a starting condition of ‘0’. I then opened and read each ADNI roster file to ensure that each genotype file name RID also appears as an RID on the ADNI study roster. To do this I opened and read the adni_roster_2009-09-01.csv file and saved its contents to a variable I named “roster”. I then read the lines of the roster as roster_lines, and then split roster_lines by new line characters. I then further split each line of roster_lines by the comma character. I then created a variable “rosterNumber” which stored the length (number of entries) of the roster_lines variable, and a “rosterNames” variable which sliced and stored the RIDs (index [1]) from each line in roster_lines. I closed the roster document, now having a list of the ADNI roster RIDs.

Next, I had to open and read each file and handle them differently depending on their compression method. To do this, I first created a variable called “first_file” which sliced the RIDs from the ordered RIDtoFilenameDict data dictionary – the first entries, the keys in that data dictionary. I then created two different ‘if’ loops. These checked the file extension by using the ‘endswith’ command, one checked for ‘endswith’ .gz, and the other checked for ‘endswith’ .zip. Both loops next used the sys.stderr.write functionality to write to the computer screen “opening” plus the name of the file it was opening so that I could see what progress my script was making in handling the files. For the .gz files, a variable called “FI” used gzip to open and read the first_file. For .zip files, a variable called “FF” used zipfile.ZipFile read functionality, then opened the first file in the list of files opened and called that file “FI”. This was done because, unlike .gz (which opens as only the stored decompressed file), .zip files open into the stored compressed file plus additional files. The only file needed was the first file in the list of files opened, the actual decompressed data. A variable named “Read” actually read the FI data. Outside of the loops, a variable I called “Splitter” split the Read data on the new line character so that I now had a list of decompressed genomics files with each RID having its own line of genomic data. Next, I removed the header from each Read file by slicing and then deleting the zero-eth index item (the first row of each Read file, which would be the header row).

Next, I created an empty list I called "SNPCheckList" in which to store the names of all the SNPs used for the genomics assay. I created a 'for' loop that worked on each line in "Splitter", with an 'item' being each line split on the comma character (these were .csv files). I then populated the SNPCheckList by appending to its end the sliced item [4], which was the SNP name. When this was complete I created a "number_of_SNPs" variable to store the length (quantity) of the SNPCheckList, and I closed the FI variable so that the genomics files were no longer maintained in my computer's memory. The number_of_SNPs value, when completed, was 620,901.

I next started an "RID_ctr" counter, initially set to zero, and started nested 'if' loops within a 'for' loop to operate on each "file_name" in the RIDtoFilenameDict data dictionary by the now ordered RIDs (key). As each RID was read, the RID-ctr counter increased by one. I created the variable "fileRosterCheck" which split the file_name at the final (.csv) extension so that only the file name with its original .gz or .zip extension remained. The file_name then entered one of two 'if' loops, conditional on the file extension. For this I used the "endswith" function. If the file_name ended with a '.gz' file extension, I used the variable "FI" for the gzip.open function on the file_name to read the file, then "Read" as the read FI variable. If the file ended with '.zip' I created an "FF" variable to read the zipfile using zipfile.ZipFile(file_name) to read the file, converted FF to FI by opening and reading FF as the first file in the opened .zip list of files, and "Read" again as the read FI variable. "Read" was now my read genomic files, no matter their original extension. In each case, I used the sys.stderr.write functionality to print to the computer screen the RID of the file being opened and read as a way of tracking progress.

I next slit each Read file on new line characters (calling that variable “Splitter”), removed the header by slicing Splitter index [0], and deleted the Splitter index [0] header information from the Read file.

Next, I began a “lines_ctr” counter initialized to zero, and started a ‘for’ loop for each "line", splitting each line into an “item” at the comma character, and then checked the length of the file to ensure that there were twenty-four lines (each genomic file had been formatted as comma-separated lines, twenty-four lines to a complete file). This was to ensure that no partial or incomplete files were employed in the study. I used the sys.stderr.write error reporting functionality to notify me of such an event (which did not occur). I then checked each “item” at index [4] (the SNP name) against the SNPCheckList to make sure the SNPs on each Read file were in the same order (agin, with error reporting if such an event occurred – which it did not). I then combined the ‘Allele 1 forward’ and ‘Allele 2 forward’ fields from each Read genomic file into a single field (so that ‘A, A’ became ‘AA’) in my newly-created “totalGenomeMatrixFile”. I then increased my “files_counter” by one, flushed the existing Read file out of memory, and closed FI (the Read file).

I then created and opened an output file path, “totalGenome MatrixFile”, gave it a header by writing “SNPname” followed by each ordered RID. In this way, the first field on each row of the matrix had a SNP identifier name, followed by the value for that SNP for each of the ordered RIDs, left to right. A new line character was added in-line at the terminus of the “orderedRIDs” to accomplish this. I then looped over the SNPs, looping over the RIDs by a ‘for’ loop over SNPindex in the range of number_of_SNPs, then by writing SNPCheckList keys in the totalGenomeMatrixFile. I closed the totalGenomeMatrixFile output file and could now query this matrix for every SNP value for every genomic file ADNI provided, ordered (rows) by alpha-numeric ordered SNP names (620,901) and by columns left-to-right by numerically ordered RIDs (818).

Refer to the addenda section of this write-up for the scripts described.

VII. Data Cleaning

Now that I had a master demographics matrix of all the ADNI participants plus a matrix of all the genomic files (ordered by SNP name as well as by RID), my next task was to work creating methods for excluding RIDs and creating a matrix for those excluded (for which I created an exclusion log) as well as a matrix for those not excluded from the ADNI study (an included genome matrix). This comprised the data cleaning portion of my project. I called this script “*ForExcludedMatrix.py*”. Reasons for exclusion from my project were any failure to meet the requirements of the biological question posed by the investigative study: missing age (date of birth), missing gender, failure to meet AD+ or “control” population criteria (such as being co-morbid AD and HD), or having no genomic information file. It was also determined that only non-Hispanic white patient data would be used for the study so as to lessen racial confounding. The scripting language used for this task was Python.

For this script, I used my existing “totalGenomeMatrixFile” script as a template. I had faith it worked, it took only 6 seconds per file to read and write so it didn’t take very long to run over the entire ADNI set of ADNI genomic files, and it only needed extra conditional checks and processes to filter out the files that didn’t meet the inclusion criteria.

First I created “exclusionLog”, which opened and created a path to a FileExclusionLog.txt file. I then opened and read my demoMatrix, reading the lines (“demoMatrix_lines”), splitting them on new line characters, and then further splitting these on the comma characters.

I then checked that each demoMatrix line had items indexed [1], [3], [4], and [5] – the information required by this study. I started an “exclusionCount” varaiable initialized to zero, and established an empty “exclusionDict” dictionary to store excluded file RIDs. “For” each line in the demoMatrix_lines I had a long list of in-line “if” and “and” conditionals. These checked that each of the required item indexes were populated (were not empty strings), and were not populated with ‘-4’ (the missing data format used by ADNI). If the line (which represented demographics information for a single RID) passed the series of tests, its RID was printed out to the screen using the sys.stderr.write function along with the text, “File Demographics Data Complete” with a new line character so that I could visually see what progress was being made in checking these files.

If the line did not pass the checks, they entered an ‘else’ loop. In this loop, I first created an empty list called “exclusionList”. To keep track of the reasons any file could be excluded (each could have multiple, different reasons) I decided to establish an “exclusionBitField” which I initialized to zero. The idea is that each exclusion criteria could be mapped as a value to the bit field (1, 2, 4, 8, 16, and 32) – a binary “tick” for each reason a file failed the inclusion criteria - and that bit “signature” could be read out to define the reason(s) that particular RID was excluded from the study. When a file failed at any of the item checks the exclusionList was appended with the RID and the reason and the exclusionDict was also appended. I’ve included that code here:

```
      else:
            exclusionList = []
            exclusionBitField = 0
            if line[1] == "" or line[1] == '-4':
                  sys.stderr.write(line[0] + " is missing exam date."
+'\n') # 1's position
                  exclusionList.append("missing exam date")
                  exclusionBitField = exclusionBitField + 1
            if line[2] == "" or line[2] == '-4':
                  sys.stderr.write(line[0] + " is missing gender."
+'\n') # 2's position
                  exclusionList.append("missing gender")
                  exclusionBitField = exclusionBitField + 2
            if line[3] == "" or line[3] == '-4':
                  sys.stderr.write(line[0] + " is missing DOB." +'\n')
# 4's position
                  exclusionList.append("missing DOB")
                  exclusionBitField = exclusionBitField + 4
            if line[4] == "" or line[4] == '-4':
                  sys.stderr.write(line[0] + " is missing ethnicity."
+'\n') # 8's position
                  exclusionList.append("missing ethnicity")
                  exclusionBitField = exclusionBitField + 8
            if line[4] != "2" or line[5] != "5":
                  sys.stderr.write(line[0] + " is non-caucasian."
+'\n') # 16's position
                  exclusionList.append("non-caucasian")
                  exclusionBitField = exclusionBitField + 16
            if line[6] == "" or line[6] == '-4' or line[7] == "" or
line[7] == '-4':
                  sys.stderr.write(line[0] + " is missing APOE values."
+'\n') # 32's position
                  exclusionList.append("missing APOE data")
                  exclusionBitField = exclusionBitField + 32

            exclusionDict[line[0].zfill(4)] = [exclusionList,
exclusionBitField]
      continue
```

I next opened and read each ADNI Roster file to check that each genomic file RID appeared on the official ADNI roster. This was done exactly as scripted in the “4ADNIDemoMatrix” script and established my rosterNumber and rosterNames (4-digit RIDs). I then opened the genomic files by ordered RIDs and matched the RID to file_names in my “list_of_files”, as was done in the “4ADNIDemoMatrix” script. Again, there were 818 total RIDs.

I looped over the orderedRIDs, using the RID as the operand, and checked ‘if’ the RID appeared in the exclusionDict exclused files data dictionary as a key (RID). If it was in that excluded list of RIDs, I wrote to the console that the file was being skipped because it did not meet the demographics or APOE inclusion criteria, and wrote that RID into the exclusionLog. ‘If’ the RID was not in rosterNames (in other words, an RID had somehow made it into a file name but did not appear in the official ADNI roster), I wrote to the console that the file (RID) did not appear in the roster, and wrote to the exclusionLog that the file was being excluded because its RID did not appear on the roster. In both cases, files_counter incremented by one to show that another file had been checked. When all the files had been checked I wrote to the console the exclusionCount (number of files excluded out of the total number of files) as well as the total number of files examined (by the length of demoMatrix_lines). I also appended the end of the exclusionLog with the total number in the ADNI roster (rosterNumber), the number of files examined (orderedRIDs), and the total number of files excluded (exclusionCount).

Then I set about creating a variable (“includedRIDs”) to store the RIDs, which met the inclusion criteria and would be used for the study. This was the difference of the set of the sorted orderedRIDs minus the set of the exclusionDict keys (RIDs in the exclusion dictionary). I then appended the exclusionLog with the number of files remaining to be studied (the length of the includedRIDs), and closed the exclusionLog.

I then set about creating an "Included Matrix" which was the same as the "Master Matrix" but which only contained non-excluded RIDs. This followed the same file extension filtering and processing as the "ForADNIMstrMatrix" script, checking each file extension ('.gz' or '.zip'), removing the header, splitting the lines into individual items, combining allelic information into a single field, etc. The only difference in this part of the script was the name of the output file being written to "includedGenomeMatrix") and the fact that it operated only on RIDs found in the "includedRIDs".

The last part of the project I performed was a script for manipulating and creating some basic statistics on the total genome matrix (not just the included, but the population) such as minor allele frequencies and call rates for the SNPs. This constituted EDA for the project. This was done using R as the scripting language. I called this script "*GenomeMatrixManipulation.r*". To do this I first read the "TotalGenomeMatrixFile" as a .csv file and called it "genomeDF". In-line with reading the matrix file, I globally replaced "--", empty, or "-4" strings (missing data) to 'NA'. This was because the R software recognizes 'NA' as a missing value, but does not recognize '-4' so I wanted to get that handled right away. I next sliced off the top row (headings) and saved them in a variable called "rownames", saving the data (only) as "genomeMatrix". I then examined the SNP statistics: used the numberOfFiles (the length of a row, the number of RIDs), summed the number of 'NA' in a row as "missedCalls", found the "callRate" which I defined as ([numberOfFiles – missedCalls] / [numberOfFiles]), and then found the total number of each base (A, T, G, or C) by making a table of the factors of each row, split by individual characters where the levels of the factors were “A”, “T”, “G”, and “C”. The values for each SNP were simply which bases were returned for the two loci (either Allele 1 forward or Allele 2 forward). I simply had to add up the bases for each row (each row represented a single SNP). I found the “minorAllele” by making a sorted list of the bases by frequency (or count) returned for each row (each SNP) and slicing off the 2nd position (the 2nd-most frequent base), with the understanding that purine-pyrimidine mis-matches would be very rare in any population. The minor allele frequency, “minorAlleleFreq”, was defined as the number of times the minor allele base appeared in the row, divided by the sum of all the bases in the row.

I created column names for the SNPstats variable (“callRate”, “minorAllel”, and “minorAlleleFrequency”) using a vector (numerals) such that 1 = “A”, 2 = “T”, 3 = “G”, and 4 = “C”. I then used the cbind function to add the SNPstats and minor allele columns to the end of my master matrix, calling this new output file “AnotatedTotalGenomeMatrix.csv”. I saved the data from these SNPstats manipulations of the genome matrix as “TotalSNPstats.RData” and wrote the file, “TotalSNPstats.csv”.

I then did much the same thing to the IncludedGenomeMatrix.csv file, globally replacing empty fields and ‘-4’ entries with ‘NA’, and stripping off headers so I was working with only data and not header strings. My “IncludedSNPstats” listed the numberOfFiles as the length (by row) of the genomeMatrix file (the number of RIDs – each RID is a single row), the number of “missedCalls” (the sum of the ‘NA’ entries for each row), the “callRate” ([numerOfFiles – missedCalls] / numberOfFiles), “baseTotals” (which split each row into individual characters and summed the bases as levels in a factored table), found the “minorAllele” and “minorAlleleFrequency”as described above including vectorizing the numerical output for the bases such that, again, 1 = “A”, 2 = “T”, 3 = “G”, and 4 = “C”. I bound these columns to the end of the existing master matrix, and wrote an output file, “AnotatedIncludedGenomeMatrix.csv” for further study. I also saved the data as “IncludedSNPstats.RData” and wrote an output file of the results, “IncludedSNPstats.csv”, for further study.

I took this opportunity to globally change all empty string fields and ‘-4’ fields in my “ADNIDemoMatrix.csv” output file to ‘NA’ (again, for software requirements), saved the demoDF demographics data to “Demographics.RData”, and wrote the “ADNIDemoMatrix.csv” file.

Refer to the addenda section of this write-up for the scripts described.

VIII. File Management

Per my agreement, I kept all ADNI genomic files on OHSU’s secure server. These were never moved. I did copy the roster, data dictionary, demographics, and diagnostic files as described onto my computer for ease and speed of reference (I didn’t have to be logged in to the OHSU server to check the proper fields and their indexing for my script, for example). As put forth in section V., above, my script files were stored under “Code”, output data under “Data”, my graphs under “Graphs”, and text (for any text files which did not result from a manipulation of data, such as my own notes as well as the final version of this write-up) were to be stored under “Text”. I named my script files for what they ultimately did – describing their purpose. I updated versions of working scripts and kept older versions. Sometimes this was done as a numeric increment, sometimes as the date I worked on the script. I found later that including the date as a revision within the script file name was the most helpful.

When naming variables used in my script, I again tried to describe them using combinations of camel-casing (“exampleOfCamelCasing”) and underscoring (example_of_underscoring). This greatly aided finding variables, functions, within a script file and when trouble-shooting. This practice was also very helpful when time had passed since I had last looked at the script!

I strived to keep the server directories clean, and store only the “latest and greatest” versions on that server. The server became my “library” – the last best working script file version would be kept there, then down-loaded (copied) to my computer for revision and trial, saved (with revision), and then up-loaded to the server over-writing the old master file. During this project write-up, however, it became much easier simply do work on (and save) graphics on my local computer using copies of the files I had created and stored in my server “Data” sub-directory, and then move the completed graphics onto the server.

One of the things I benefited from is keeping in-line notes within my script files. I use lots of comment lines. I state what I am doing with the script code section that follows, and what each line is supposed to accomplish. This made trouble-shooting much easier. I thought of it this way: comments aren’t so much for me at the time, they are for myself in the future, or for others, to understand what I am attempting to accomplish with my script. I kept track not only of what I was attempting, but the methods used and even definitions of indexes used, etc. For examples of this, refer to any of my scripts in the addenda section of this write-up.

IX. Results

A. Deliverables: Scripts

At the termination of my project, I had created scripts and completed EDA for the HBTRC dataset (leading to its rejection for further study related to this project) plus four ADNI-related scripts to convert their data into query-able matrices for: 1) the demographics information; 2) genomics information; 3) data cleaning both the excluded files' data as well as the included files' data along with a log detailing excluded and included files, and; 4) EDA descriptive statistics of the genomic files. These can be found in the Addenda.

B. Deliverables: Data Files

I had also, through my scripts, created matrices of the ADNI data files: 1) the ADNIDemoMatrix020513, a .csv file which contained all the demographics information for the ADNI study cohort; 2) Demographics.RData, the R version of demographics information; 3) TotalSNPstats.RData, the R version of the statistics related to SNPs for the cohort; and 4) IncludedSNPstats.Rdata, the R version of the statistics related to SNPs for the included study group. These files can be loaded into R and the data contained within them examined statistically using R.

I also created numerous .csv data files (stored under the CSVfiles subdirectory of DATA), some rather large. These included: ADNIDemoMatrix.csv, AnotatedIncludedGenomeMatrix.csv, AnotatedTotalGenomeMatrix.cv, IncludedGenomeMatrix.csv, IncludedSNPstats.csv, MasterADNIMatrix.csv, SNPstats.csv, TotalGenomeMatrixFile.csv, and TotalSNPstats.csv.

A very small sample of my “MasterADNIMatrix” is shown here, as an Excel file. RIDs were stored as 4-digit numbers, but have been shown here truncated. This shows the SNPname (each SNP is a row) with RIDs ordered by columns, each RID then has the allelic information (1 forward and 2 forward) as separate fields, then a field for the combined complimentary nucleotides together as a pair in a single field. This field was used for SNP call rates and minor allele frequency equations.

SNPname	RID	Allele1-forw	Allele2-forw	Nucleotide1	RID	Allele1-forw	Allele2-forw	Nucleotide1
200003	5	A	A	TT	50	G	G	CC
200006	5	C	C	GG	50	T	C	AG

A very small sample of the “ADNIDemoMatrix.csv” file is shown here, showing all empty and ‘-4’ fields converted to ‘NA’. Note that this is for the cohort, not just those included in the study.

	RID	EXAMDATE	PTGENDER	PTDOB	PTETHCAT	PTRACCAT	APGEN1	APGEN2	AD.status	MCI.AD.other.none.
1	1	8/18/05	2	/1944	NA	NA	NA	NA	NA	NA
2	10	10/26/05	2	/1931	2	5	3	4	1	NA
3	100	12/12/05	2	/1930	2	5	3	4	NA	NA
4	1000	10/11/06	1	/1924	2	5	3	3	NA	NA
5	1001	11/6/06	1	/1937	2	4	3	4	1	NA
6	1002	10/13/06	2	/1930	2	5	3	3	NA	NA
7	1003	10/13/06	1	/1935	2	5	3	4	NA	NA
8	1004	11/10/06	NA	NA	NA	NA	3	4	1	NA

Here is an example of the “SNPstats.csv” file, showing the SNP name (rsXXXX), the call rate, minor allele and its frequency. The project was interested in SNPs, minor alleles, only so major alleles were not recorded.

	callRate	minorAllele	minorAlleleFrequency
rs7060463	1	4	0.42787286
rs941009	1	4	0.26405868
rs4446471	0.99877751	2	0.27050184
rs9821655	0.99755501	1	0.01286765
rs7330809	0.93887531	3	0.078125
rs1898321	1	3	0.07334963
rs1898320	0.99633252	3	0.13803681

The alleles are: 1 = “A”, 2 = “T”, 3 = “G”, and 4 = “C”.

Lastly, in running the exclusion conditions on the demographics and genomics data matrices, I came up with two text files. One, “FileExclusionLog.txt” was for the RIDs excluded from the study and the reason(s). The other, “FileFailureLog.txt” merely tracked exclusions by RID such that a single file with more than one exclusion criteria would be counted for each “failure” – it was merely a tracking device but was kept because it would be easier to parse by “reason” than the “FileExclusionLog.txt” file.

The terminus of the “FileExclusionLog.txt” is shown here. Note that it gives the bit count for the failure codes, the RID, what is missing (as a text string), the total number of files excluded up to that point and how many files total are being examined. It then gives the total number in the ADNI roster (a significantly greater number on the roster than actually had demographics and genomics files), how many files were examined, how many were excluded, and how many were to be included in the investigative study project. I have shown the output as code for clarity, but it is a text file.

```
Exclusion 30: 1387 is missing gender, missing DOB, missing ethnicity,
non-caucasian. 119 files excluded out of 818 total files input. 26
files remaining.
Exclusion 16: 1407 is non-caucasian. 120 files excluded out of 818
total files input. 16 files remaining.
Exclusion 30: 1417 is missing gender, missing DOB, missing ethnicity,
non-caucasian. 121 files excluded out of 818 total files input. 10
files remaining.
1431 total Roster number.
818 total Gene Files examined.
121 Gene Files excluded.
```

```
697 files INCLUDED to be studied.
```

C. Deliverables: EDA (statistics and graphics)

These were constructed using R. The script code as well as output is shown below.

The number of unique RIDs (patients studied) in the ADNI cohort was:

```
> length(unique(demoDF$RID))
[1] 1388
```

The gender distribution of the ADNI cohort was:

```
> table(demoDF$PTGENDER, useNA = "ifany") # count by gender
(male/female/na), ADNI dataset

  male female   <NA>
   774    569     45
```

So, of the 1,388 studied patients, 45 did not have their gender recorded. Those 45 would not have met the inclusion criteria for further study.

The Alzheimer's disease (AD) status of the cohort was:

```
> table(demoDF$AD.status, useNA = 'ifany') # count by AD status, ADNI
dataset

 AD-  AD+ <NA>
 478  341  569
```

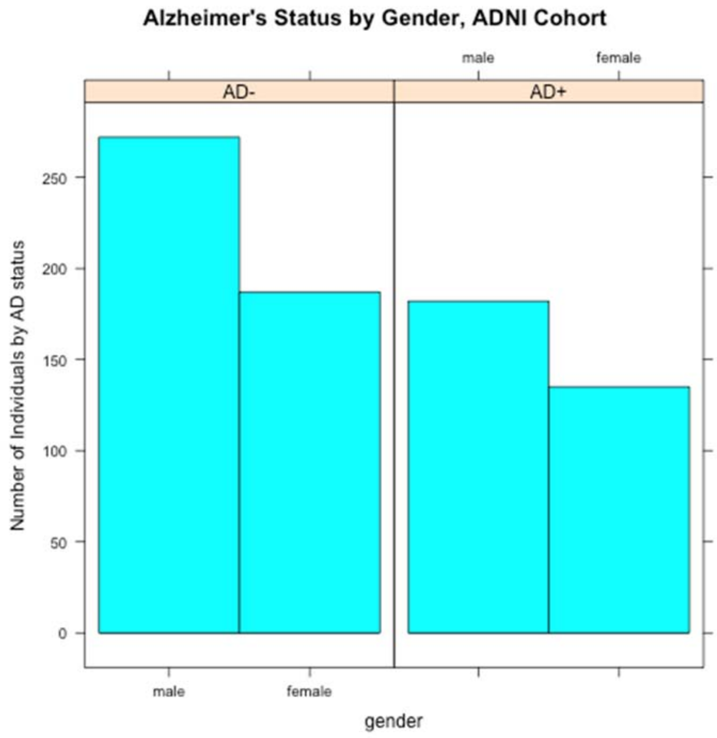
The AD status by gender of the cohort was:

```
> tapply(demoDF$PTGENDER, demoDF$AD.status, summary) # summary of
disease status by gender, ADNI cohort
$`AD-`
  male female   NA's
   272    187     19

$`AD+`
  male female   NA's
   182    135     24
```

Those without an AD status recorded would not have been included in this study.

Graphically, the AD status by gender was much less skewed Female for the AD- group than was the HBTRC cohort:



The prevalence and cause of MCI in these patients was:

```
> table(demoDF$MCI.AD.other.none., useNA = 'ifany') # prevalence and
cause of MCI, ADNI cohort

           No MCI     MCI due to AD MCI due to non-AD              <NA>
              411               400                 8               569
```

This can be further broken down by AD status:

```
> table(demoDF$AD.status, demoDF$MCI.AD.other.none., useNA = 'ifany') #
prevalence and cause of MCI by AD status, ADNI cohort

       No MCI MCI due to AD MCI due to non-AD <NA>
  AD-     220           251                 7    0
  AD+     191           149                 1    0
  <NA>      0             0                 0  569
```

Examples of the Date of Birth (DOB) by gender for the ADNI cohort (this is not the entire cohort, merely an example):

```
> table(demoDF$PTDOB, demoDF$PTGENDER, useNA="ifany") # date of birth
by gender, ADNI cohort

        male female <NA>
           0      0    2
  /1915    0      1    0
  /1916    2      2    0
  /1917    6      2    0
  /1918    7      5    0
  /1919   16      4    0
  /1920   18     13    0
  /1921   21     18    0
```

Those without a date of birth would have been excluded from the study.

The APOE genotypes for both alleles are given below, as is the number of the cohort that has (TRUE) and does not have (FALSE) the APOE genotypes listed:

```
> table(demoDF$APGEN1, useNA="ifany")

   2    3    4 <NA>
 101  945  111  231
> table(demoDF$APGEN1, useNA="ifany") # APOE type (-2/-3/-4), first
allele, ADNI cohort

   2    3    4 <NA>
 101  945  111  231
> table(demoDF$APGEN2, useNA="ifany") # APOE type (-2/-3/-4), second
allele, ADNI cohort

   2    3    4 <NA>
   3  607  547  231
> table(is.na(demoDF$APGEN1), is.na(demoDF$APGEN2), useNA="ifany") #
missing APOE type info, ADNI cohort

        FALSE TRUE
  FALSE  1157    0
  TRUE      0  231
```

This indicates that the 231 individuals missing APOE genotype information were missing all (both) alleles, and those that had APOE genotype information had both allelic types recorded. Those missing APOE types would have been excluded from the study.

X. Discussion and Summary

More EDA can be done on the ADNI dataset, relative to future specific hypotheses. Also, R and/or Python packages or libraries might be able to handle tasks I scripted myself such as finding allele frequencies or denoting allelic heterogeniety as “0” (no minor alleles), “1” (1 minor allele), or “2” (both alleles are the minor base variant). This would have reduced the amount of time but, being new at programming as well as genomic dataset manipulation, the experience of developing my own tools was invaluable. I have produced the proper outcomes for my project: a dataset converted into appropriate data matrices, cleaned, and EDA performed (or data prepared for EDA) to suit the needs of an exploratory investigative study. EDA showed that the HBTRC dataset was not suitable for use in testing this particular project’s hypothesis, but the ADNI GWAS dataset was. This cleaned ADNI dataset can be used for this and future studies.

CITATIONS

1. Louie B, Mork P, Martin-Sanchez F, Halevy A, Tarczy-Hornoch P. Methodological Review: Data integration and genomic medicine. Journal of Biomedical Informatics 2007 Feb;40(1): 5-16.

2. Müller, Heiko, and Johann-Christph Freytag. *Problems, methods, and challenges in comprehensive data cleansing*. Professoren des Inst. Für Informatik, 2005.

3. Johnson RC, Nelson GW, Troyer JL, Lautenberger JA, Kessing BD, Winkler CA, et al. Accounting for multiple comparisons in a genome-wide association study (GWAS). BMC Genomics. 11:724–724.

4. Hebert LE, Scherr PA, Bienias JL, Bennett DA, Evans DA. Alzheimer Disease in the US Population: Prevalence Estimates Using the 2000 Census. Arch Neurol. 2003 Aug 1;60(8):1119–22.

5. Jun G, Naj AC, Beecham GW, Wang L-S, Buros J, Gallins PJ, et al. Meta-analysis confirms CR1, CLU, and PICALM as alzheimer disease risk loci and reveals interactions with APOE genotypes. Arch. Neurol. 2010 Dec;67(12):1473–84.

6. Brookmeyer R, Johnson E, Ziegler-Graham K, Arrighi HM. Forecasting the global burden of Alzheimer's disease. Alzheimers Dement. 2007 Jul;3(3):186–91.

7. Devi L, Prabhu BM, Galati DF, Avadhani NG, Anandatheerthavarada HK. Accumulation of Amyloid Precursor Protein in the Mitochondrial Import Channels of Human Alzheimer's Disease Brain Is Associated with Mitochondrial Dysfunction. J. Neurosci. 2006 Aug 30;26(35):9057–68.

8. Blacker D, Albert MS, Bassett SS, Go RC, Harrell LE, Folstein MF. Reliability and validity of NINCDS-ADRDA criteria for Alzheimer's disease. The National Institute of Mental Health Genetics Initiative. Arch. Neurol. 1994 Dec;51(12):1198–204.

9. McKhann G, Drachman D, Folstein M, Katzman R, Price D, Stadlan EM. Clinical diagnosis of Alzheimer's disease: report of the NINCDS-ADRDA Work Group under the auspices of Department of Health and Human Services Task Force on Alzheimer's Disease. Neurology. 1984 Jul;34(7):939–44.

10. Mölsä PK, Marttila RJ, Rinne UK. Survival and cause of death in Alzheimer's disease and multi-infarct dementia. Acta Neurol. Scand. 1986 Aug;74(2):103–7.

11. Alzheimer's Association. 2011 Alzheimer's disease facts and figures. Alzheimer's and Dementia. 2011 Mar;7(2):208–44.

12. Waring SC, Rosenberg RN. Genome-wide association studies in Alzheimer disease. Arch. Neurol. 2008 Mar;65(3):329–34.

13. Mahley RW, Weisgraber KH, Huang Y. Apolipoprotein E4: a causative factor and therapeutic target in neuropathology, including Alzheimer's disease. Proc. Natl. Acad. Sci. U.S.A. 2006 Apr 11;103(15):5644–51.

14. Bird TD. Genetic aspects of Alzheimer disease. Genet. Med. 2008 Apr;10(4):231–9.

15. Alzheimer's Association. 2012 Alzheimer's disease facts and figures. Alzheimer's and Dementia. 2012 Mar;8(2):131-68.

16. Johnson RC, Nelson GW, Troyer JL, Lautenberger JA, Kessing BD, Winkler CA, et al. Accounting for multiple comparisons in a genome-wide association study (GWAS). BMC Genomics. 11:724–724.

17. Van Rossum G, Drake FL (eds). Python Reference Manual. PythonLabs, Virginia, USA, 2001. Available at http://www.python.org

18. R Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, 2012. Available at http://www.R-project.org

19. Sarkar D. Lattice: Multivariate Data Visualization with R. Springer, New York, 2008. Available at http://lmdvr.r-forge.r-project.org

20. Gentleman RC, Carey VJ, Bates DM, Bolstad B, Dettling M, Dudoit S, Ellis B, Gautier L, Ge Y, Gentry J, Hornik K, Hothorn T, Huber W, Iacus S, Irizarry R, Leisch F, Li C, Maechler M, Rossini AJ, Sawitzki G, Smith C, Smyth G, Tierney L, Yang LYH, Zhang J. Bioconductor: open software development for computation biology and bioinformatics. Genome Biology. 2004, 5:R80.

21. Smyth GK. Limma: linear models for microarray data. Bioinformatics and Computational Solutions using R and Bioconductor. Gentleman R, Carey S, Dudoit S, Irizarry R, Huber W (eds). Springer, New York, 2005. 397-420.

22. Wacholder S, McLaughlin JK, Silverman DT, Mandel JS. Selection of controls in case-control studies. I. Principles. Am. J. Epidemiol. 1992 May 1;135(9):1019–28

23. Sage Bionetworks Repository website, available at: http://www.sagebase.org/commons/repository.php

24. Mueller SG, Weiner MW, Thal LJ, Petersen RC, Jack CR, Jagust W, et al. Ways toward an early diagnosis in Alzheimer's disease: the Alzheimer's Disease Neuroimaging Initiative (ADNI). Alzheimers Dement. 2005 Jul;1(1):55–66.

25. ADNI website, available at: http://adni-info.org/Home.aspx

26. Oregon Health and Science University Data Use Agreement, available at: www.ohsu.edu/xd/about/services/integrity/policies/upload/Data-Use-Agreement.pdf

ADDENDA

Scripts referenced in this write-up:

1. ForADNIDemoMatrix020513.py

```
#!/usr/local/bin/python
# Rex M. Williamson
# Master Demograhics Matrix of ADNI phenotypes (RIDs, exam date,
gender, DOB, ethnicity (2=non-hispanic), race (5=white/caucasian); APOE
Types (APGEN1, APGEN2))
# all original files are .csv

import sys

# Constants...
DIRPATH="/Users/rexwilliamson/Desktop/ADNI/"

fh = open(DIRPATH + "ADNIDemoMatrix020513.csv", 'w')

files_counter = 0
# open files, read them, split on new lines --> patient demographics
and APOE-typing
DEMO = open(DIRPATH+"adni_ptdemog_2009-09-01.csv", 'r')
APOE = open(DIRPATH+"adni_apoeres_2009-09-01.csv", 'r')
AD_MCIstatus = open(DIRPATH+"adni_pdxconv_2009-09-01.csv", 'r')
ReadDEMO = DEMO.read()
ReadAPOE = APOE.read()
ReadAD_MCIstatus = AD_MCIstatus.read()

SplitAPOE = ReadAPOE.split('\n')[0:-1] # [0:-1] gets rid of the final
extra '/n' newline
SplitDEMO = ReadDEMO.split('\n')[0:-1]
SplitAD_MCIstatus = ReadAD_MCIstatus.split('\n')[0:-1]

# get headers from both files and make them into the appropriate
"final" header
APOEheader = SplitAPOE[0]
DEMOheader = SplitDEMO[0]
item = DEMOheader.split(',')
DEMO_HEADER = item[1]+',' + item[3]+',' + item[5]+',' + item[6]+',' +
item[21]+',' + item[22]+','
# [1]=RID, , [3]=ExamDate, [5]=Gender(1m/2f), [6]=DOB,
[21]=ethnicity(2=non-hispanic), [22]=race(5=white/caucasion), AD
status, MCI(AD/other)
item = APOEheader.split(',')
APOE_HEADER = item[3]+',' + item[4]

FINAL_HEADER = DEMO_HEADER + APOE_HEADER + ',' + 'AD status' + ',' +
'MCI(AD/other/none)'

# remove header from files to make looping easier and not have to
remove multiple header lines...
del SplitAPOE[0]
del SplitDEMO[0]
del SplitAD_MCIstatus[0]

# start 2 data dictionaries (1 APOE, 1 DEMO): KEY = RID, VALUES =
(PTGENDER, PTDOB, PTADBEG){adni_ptdemog}+(APGEN1, APGEN2){adni_apoeres}
DEMOdict = {}
APOEdict = {}
AD_MCIdict = {}

# split each line on ',' and add ',' back how I want it, assigning data
dict Keys to Values
for line in SplitDEMO:
      item = line.split(',')
      DEMOdict[item[1]+','] = item[3]+',' + item[5]+',' + item[6]+',' +
item[21]+',' + item[22]+','

for line in SplitAPOE:
      item = line.split(',')
      APOEdict[item[1]+','] = item[3]+',' + item[4]+','

# add the AD status info into the existing DEMO dictionary (at the
end): item[18] = MCI(1=AD, 2=not AD, 0 = none), item[21] = AD (1=T,
0=F)
ADstatus = '0'
MCIstatus = '0' # 0 = missing, 1 = due to AD, 2 = due to non-AD (if
ever goes to "1", it stays 1, can be "2" if was 0, if nothing it stays
0
previousRID = '0'
for line in SplitAD_MCIstatus:
      item = line.split(',')
      currentRID = item[1]
      if currentRID == previousRID:
            # accumulate AD status
            # ADstatus = item[21] # input values can be 1 (AD+) and -4
(AD-)
            # MCIstatus = item[18] # input values can be 1 (AD), 2
(non-AD), -4 (none/missing value)
            if item[21] == '1': # my currrent AD status - once it gets
to "1", it stays there (= AD+)
                  ADstatus = '1' # my output
            if item[18] == '1': # my current MCI status
                  MCIstatus = '1' # my output
            if item[18] == '2' and MCIstatus == "0": # to verify MCI
status doesn't get down-graded
                  MCIstatus = '2'

      else:
            if previousRID != '0':
                  AD_MCIdict[previousRID +','] = ADstatus +',' +
MCIstatus
            previousRID = item[1]
            if item[18] == '-4':
                  MCIstatus = '0'
            else:
                  MCIstatus = item[18]
            if item[21] == '-4':
                  ADstatus = '0'
            else:
                  ADstatus = item[21]

AD_MCIdict[previousRID +','] = ADstatus +',' + MCIstatus # write the
last block of RID values so loop doesn't "drop off" and hang up/not
print.

sys.stderr.write(str(AD_MCIdict))

# close the input files
DEMO.close()
APOE.close()
AD_MCIstatus.close()

# write my master header to my output file...
fh.write(FINAL_HEADER +'\n')

#sys.stderr.write(str(list(set(DEMOdict.keys()) | set(APOEdict.keys())
| set(AD_MCIdict.keys()))))
#sys.stderr.write(str(sorted(list(set(DEMOdict.keys()) |
set(APOEdict.keys()) | set(AD_MCIdict.keys())))))

# now, combine dictionaries into 1 "master" with RID as KEY, sorted by
RID
for key in sorted(list(set(DEMOdict.keys()) | set(APOEdict.keys()) |
set(AD_MCIdict.keys()))):
      fh.write(key)
      DEMO_info = DEMOdict.get(key, [])
      if DEMO_info == []:
            fh.write(',,,')
      else:
            fh.write(DEMO_info)
      APOE_info = APOEdict.get(key, [])
      if APOE_info ==[]:
            fh.write(',,')
      else:
            fh.write(APOE_info)
      AD_MCI_info = AD_MCIdict.get(key, [])
      if AD_MCI_info ==[]:
            fh.write(',')
      else:
            fh.write(AD_MCI_info)
      fh.write('\n')

# close the write file
fh.close()
```

2. ForADNIMstrMatrix4.py

```
#!/usr/local/bin/python
# Rex M. Williamson
# Master Matrix (4) of ALL ADNI genotypes (no files excluded) for
thesis
```

```
import sys
import glob
import gzip
import zipfile
import zlib
import os

# Constants...
DIRPATH="/home/shared/ADNI/"

def get_compressed_filename(path_to_compressed_file,
uncompressed_type='csv'): # so can read 'gz' or 'zip'
        return os.path.split(path_to_compressed_file)[-1].split('.') +
uncompressed_type

list_of_files = glob.glob(os.path.join(DIRPATH, "*.csv.*"))[0:]
#list_of_files = [DIRPATH+"141_S_1004.csv.gz",
DIRPATH+"057_S_1007.csv.gz"]
#list_of_files = []
totalGeneFiles = len(list_of_files) # should yield 818
sys.stderr.write(str(totalGeneFiles) + " Total Gene Files " + '\n')

unorderedRIDs = [os.path.split(file_name)[1][6:10] for file_name in
list_of_files]

RIDtoFilenameDict = dict(zip(unorderedRIDs, list_of_files))

orderedRIDs = sorted(unorderedRIDs)


files_counter = 0

### open and read ADNI Roster file to check that each genotype file
read also appears on Roster
roster = open("/home/shared/ADNI/adcs_clinical/adni_roster_2009-09-
01.csv", 'r')
roster_lines = roster.read()
roster_lines = roster_lines.split('\n')[1:-1]
roster_lines = [line.split(',') for line in roster_lines]
rosterNumber = len(roster_lines)
rosterNames = [lines[1] for lines in roster_lines]
roster.close()

first_file = RIDtoFilenameDict[orderedRIDs[0]] # gives the whole file
name for the first file

# actually opening and reading each genome data file
if first_file.endswith('.gz'):
      sys.stderr.write("opening " + first_file +'\n')
      FI = gzip.open(first_file, 'r')
      Read = FI.read()

if first_file.endswith('.zip'):
      sys.stderr.write("opening " + first_file + '\n')
      FF = zipfile.ZipFile(first_file, 'r')
      FI = FF.open(FF.namelist()[0], 'r')
      Read = FI.read()


# Split on new line characters...
Splitter = Read.split('\n')[0:-1]

# Remove header...
header = Splitter[0]
del Splitter[0]

# SNP check list, to ensure all SNPs are present on each file

SNPCheckList = [] # ordered list of SNPs - should be the same order in
all files (already checked)

for line in Splitter:
      item = line.split(',')
      SNPCheckList.append(item[4]) # [4] = snp name

number_of_SNPs = len(SNPCheckList)

FI.close()


# Let's pre-allocate a matrix for my SNPnames/RIDs and allele1+2 data
for ALL gene files
totalGenomeMatrix = [['**'] * totalGeneFiles for i in
range(number_of_SNPs)]

##### NOW iterate over RIDs by opening and reading the genome files, in
order

RID_ctr = 0
for RID in orderedRIDs:
      file_name = RIDtoFilenameDict[RID]

      RID_ctr = RID_ctr + 1

      fileRosterCheck = file_name.split("/")[-1].split('.') #new line -
only file name w/o extension should remain

      # actually opening and reading each genome data file
      if file_name.endswith('.gz'):
            sys.stderr.write("opening " + file_name +'\n')
            FI = gzip.open(file_name, 'r')
            Read = FI.read()

      if file_name.endswith('.zip'):
            sys.stderr.write("opening " + file_name + '\n')
            FF = zipfile.ZipFile(file_name, 'r')
            FI = FF.open(FF.namelist()[0], 'r')
            Read = FI.read()
      # Split on new line characters...
      Splitter = Read.split('\n')[0:-1]

      # Remove header...
      header = Splitter[0]
      del Splitter[0]


      lines_ctr = 0
      for line in Splitter: # non-header info
            lines_ctr = lines_ctr + 1
            item = line.split(',')
            #sys.stderr.write(str(item))
            if len(item) != 24: # there are 24 csv items in each row of
each file - if not 24, bad file
                  sys.stderr.write("NOT 24 ITEMS: line
{CTR}\n".format(CTR=lines_ctr))
                  sys.stderr.write(str(line))
                  sys.stderr.write("\n")
                  sys.stderr.write(str(item))
                  sys.stderr.write("\n")
                  continue

            if item[4] != SNPCheckList[lines_ctr - 1]: # makes sure
SNPs are in order
                  sys.stderr.write("SNP being read does not match
standard ordered SNP list, line " + str(lines_ctr) + "." + '\n')
                  continue

            # combine the 'Allele1 Forward' ('A')[7] and 'Allele2
Forward' ('A')[8] into a single field ('AA')

            totalGenomeMatrix[lines_ctr - 1][RID_ctr - 1] = item[7] +
item[8]


      files_counter = files_counter +1
      sys.stderr.write("{FILEN}: {FN}\n".format(FILEN=files_counter,
FN=file_name))
      sys.stderr.write('\n')
      sys.stderr.flush()

      FI.close() # closing any generic genome file, be it .gz or .zip

# open our OUTPUT FILE
totalGenomeMatrixFile =
open("/home/willirex/TotalGenomeMatrixFile.csv", 'w')

# output file header
totalGenomeMatrixFile.write("SNPname," + ",".join(orderedRIDs) + '\n')

# Loop over SNPs, loop over RIDs
for SNPindex in range(number_of_SNPs): # the key is the snp name
      # print SNPname, then info for each RID - kept in order
      totalGenomeMatrixFile.write(SNPCheckList[SNPindex] + "," +
",".join(totalGenomeMatrix[SNPindex]) + '\n') # gives the initial SNP
name

totalGenomeMatrixFile.close()
```

3. ForExcludedMatrix.py

```
#!/usr/local/bin/python
# Rex M. Williamson
# Master Matrix (4) of EXLUDED genome files (RIDs)
# uses ADNIDemoMatrix demographics (demo) file, roster, and raw genome
data files
# outputs an Exclusion Log, Included master matrix (only those RIDs
that aren't excluded)


import glob
import gzip
import zipfile
import zlib
import sys
import os

# Constants...
DIRPATH="/home/shared/ADNI/"

### adding an EXCLUSION "failure" file
exclusionLog = open("/home/willirex/FileExclusionLog.txt", 'w')
### open and read my ADNIDemoMatrix to check that all records have
valid, non-empy/non"-4" value for gender, dob, & both APOE fields.
demoMatrix = open("/home/willirex/ADNIDemoMatrix.csv", 'r') # checking
items [1], [3], [4], & [5].
demoMatrix_lines = demoMatrix.read()
demoMatrix_lines = demoMatrix_lines.split('\n')[1:-1]
demoMatrix_lines = [line.split(',') for line in demoMatrix_lines]

### HERE IS WHERE I NEED TO CHECK THAT EACH RID IN DEMOMATRIX HAS
VALUES FOR items [1], [3], [4], & [5]

exclusionCount = 0
exclusionDict = {}
for line in demoMatrix_lines: # let's try bit fields!
      if line[1] != "" and line[1] != '-4' and line[2] != "" and
line[2] != '-4' and line[3] != "" and line[3] != '-4' and line[4] != ""
and line[4] != '-4' and line[4] == "2" and line[5] == "5" and line[6]
!= "" and line[6] != '-4' and line[7] != '' and line[7] != '-4':
            sys.stderr.write(line[0] +" File Demographics Data
Complete" +'\n')

      else:
            exclusionList = []
            exclusionBitField = 0
            if line[1] == "" or line[1] == '-4':
                  sys.stderr.write(line[0] + " is missing exam date."
+'\n') # 1's position
                  exclusionList.append("missing exam date")
                  exclusionBitField = exclusionBitField + 1
            if line[2] == "" or line[2] == '-4':
                  sys.stderr.write(line[0] + " is missing gender."
+'\n') # 2's position
                  exclusionList.append("missing gender")
                  exclusionBitField = exclusionBitField + 2
            if line[3] == "" or line[3] == '-4':
                  sys.stderr.write(line[0] + " is missing DOB." +'\n')
# 4's position
                  exclusionList.append("missing DOB")
                  exclusionBitField = exclusionBitField + 4
            if line[4] == "" or line[4] == '-4':
                  sys.stderr.write(line[0] + " is missing ethnicity."
+'\n') # 8's position
                  exclusionList.append("missing ethnicity")
                  exclusionBitField = exclusionBitField + 8
            if line[4] != "2" or line[5] != "5":
                  sys.stderr.write(line[0] + " is non-caucasian."
+'\n') # 16's position
                  exclusionList.append("non-caucasian")
                  exclusionBitField = exclusionBitField + 16
            if line[6] == "" or line[6] == '-4' or line[7] == "" or
line[7] == '-4':
                  sys.stderr.write(line[0] + " is missing APOE values."
+'\n') # 32's position
                  exclusionList.append("missing APOE data")
                  exclusionBitField = exclusionBitField + 32

            exclusionDict[line[0].zfill(4)] = [exclusionList,
exclusionBitField]
      continue

### open and read ADNI Roster file to check that each genotype file
read also appears on Roster
roster = open("/home/shared/ADNI/adcs_clinical/adni_roster_2009-09-
01.csv", 'r')
roster_lines = roster.read()
roster_lines = roster_lines.split('\n')[1:-1]
roster_lines = [line.split(',') for line in roster_lines]
rosterNumber = len(roster_lines)
rosterNames = [lines[1][6:10] for lines in roster_lines]
roster.close()

### open genome files and put the RIDs in order, matching RID to file
name
list_of_files = glob.glob(os.path.join(DIRPATH, "*.csv.*"))[0:]
totalGeneFiles = len(list_of_files) # should yield 818
sys.stderr.write(str(totalGeneFiles) + " Total Gene Files " + '\n')
unorderedRIDs = [os.path.split(file_name)[1][6:10] for file_name in
list_of_files]
RIDtoFilenameDict = dict(zip(unorderedRIDs, list_of_files))
```

```
orderedRIDs = sorted(unorderedRIDs)

files_counter = 0
# now loop over ordered RIDs
for RID in orderedRIDs:
      if RID in exclusionDict.keys():
            exclusionCount = exclusionCount + 1
            sys.stderr.write(str(RID) + " did not meet demographic/APOE
inclusion criteria; skipping." + "\n")
            exclusionLog.write("Exclusion " +
str(exclusionDict[RID][1]) + ": " + str(RID) + " is " + ",
".join(exclusionDict[RID][0]) + ". " + str(exclusionCount) + " files
excluded out of " + str(totalGeneFiles) + " total files input. " +
str(totalGeneFiles - exclusionCount - files_counter) + " files
remaining." +'\n')
            continue
      if RID not in rosterNames: #new line - roster [1] is file name
w/o extension
            exclusionCount = exclusionCount +1
            sys.stderr.write(file_name + " did not appear in ADNI
Roster." +'\n')
            exclusionLog.write(file_name + " did not appear in ADNI
Roster. "  + str(exclusionCount) + " files excluded out of " +
str(totalGeneFiles) + " total files input. " +  str(totalGeneFiles -
exclusionCount - files_counter) + " files remaining." +'\n')
            continue
      files_counter = files_counter + 1


sys.stderr.write(str(exclusionCount) + " files excluded out of " +
str(totalGeneFiles) + " total files." +'\n')
sys.stderr.write(str(len(demoMatrix_lines)) + " files reviewed."+'\n')

exclusionLog.write(str(rosterNumber) + " total Roster number." + '\n')
exclusionLog.write(str(len(orderedRIDs)) + " total Gene Files
examined." + '\n')
exclusionLog.write(str(exclusionCount) + " Gene Files excluded." +
'\n')


includedRIDs = sorted(set(orderedRIDs) - set(exclusionDict.keys()))

exclusionLog.write(str(len(includedRIDs)) + " files INCLUDED to be
studied." +'\n')

exclusionLog.close()

#################### CREATE "INCLUDED" MATRIX - same as "Master Matrix"
but only for RID's not excluded above ######

first_file = RIDtoFilenameDict[includedRIDs[0]] # gives the whole file
name for the first file

# actually opening and reading each genome data file, depending on its
type (.gz vs .zip)
```

```
if first_file.endswith('.gz'):
      sys.stderr.write("opening " + first_file +'\n')
      FI = gzip.open(first_file, 'r')
      Read = FI.read()

if first_file.endswith('.zip'):
      sys.stderr.write("opening " + first_file + '\n')
      FF = zipfile.ZipFile(first_file, 'r')
      FI = FF.open(FF.namelist()[0], 'r')
      Read = FI.read()


# Split on new line characters...
Splitter = Read.split('\n')[0:-1]

# Remove header...
header = Splitter[0]
del Splitter[0]

# SNP check list, to ensure all SNPs are present on each file
SNPCheckList = [] # ordered list of SNPs - should be the same order in
all files (already checked)

for line in Splitter:
      item = line.split(',')
      SNPCheckList.append(item[4]) # [4] = snp name

number_of_SNPs = len(SNPCheckList) # should be 620,901

FI.close()


# Let's pre-allocate a matrix for my SNPnames/RIDs and allele1+2 data
for ALL gene files
# and over-write it with the RID and Allele1-F+Allele2-F for each SNP
(any blanks are '**')
includedGenomeMatrix = [['**'] * len(includedRIDs) for i in
range(number_of_SNPs)]

##### NOW iterate over RIDs by opening and reading the genome files, in
order
files_counter = 0 # this counter SHOULD equal the RID counter (RID_ctr)
but it's a sanity check
RID_ctr = 0
for RID in includedRIDs:
      file_name = RIDtoFilenameDict[RID]

      RID_ctr = RID_ctr + 1

      fileRosterCheck = file_name.split("/")[-1].split('.') #new line -
only file name w/o extension should remain

      # actually opening and reading each genome data file
      if file_name.endswith('.gz'):
            sys.stderr.write("opening " + file_name +'\n')
            FI = gzip.open(file_name, 'r')
            Read = FI.read()

      if file_name.endswith('.zip'):
            sys.stderr.write("opening " + file_name + '\n')
            FF = zipfile.ZipFile(file_name, 'r')
            FI = FF.open(FF.namelist()[0], 'r')
            Read = FI.read()


      # Split on new line characters...
      Splitter = Read.split('\n')[0:-1]

      # Remove header...
      header = Splitter[0]
      del Splitter[0]


      lines_ctr = 0
      for line in Splitter: # non-header info
            lines_ctr = lines_ctr + 1
            item = line.split(',')
            #sys.stderr.write(str(item))
            if len(item) != 24: # there are 24 csv items in each row of
each file - if not 24, bad file
                  sys.stderr.write("NOT 24 ITEMS: line
{CTR}\n".format(CTR=lines_ctr))
                  sys.stderr.write(str(line))
                  sys.stderr.write("\n")
                  sys.stderr.write(str(item))
                  sys.stderr.write("\n")
                  continue

            if item[4] != SNPCheckList[lines_ctr - 1]: # makes sure
SNPs are in order
                  sys.stderr.write("SNP being read does not match
standard ordered SNP list, line " + str(lines_ctr) + "." + '\n')
                  continue

            # combine the 'Allele1 Forward' ('A')[7] and 'Allele2
Forward' ('A')[8] into a single field ('AA')

            includedGenomeMatrix[lines_ctr - 1][RID_ctr - 1] = item[7]
+ item[8]


      files_counter = files_counter +1
      sys.stderr.write("{FILEN}: {FN}\n".format(FILEN=files_counter,
FN=file_name))
      sys.stderr.write('\n')
      sys.stderr.flush()

      FI.close() # closing any generic genome file, be it .gz or .zip
```

```
# open our OUTPUT FILE
includedGenomeMatrixFile =
open("/home/willirex/IncludedGenomeMatrixFile.csv", 'w')

# output file header
includedGenomeMatrixFile.write("SNPname," + ",".join(includedRIDs) +
'\n')

# Loop over SNPs, loop over RIDs
for SNPindex in range(number_of_SNPs): # the key is the snp name
      # print SNPname, then info for each RID - kept in order
      includedGenomeMatrixFile.write(SNPCheckList[SNPindex] + "," +
",".join(includedGenomeMatrix[SNPindex]) + '\n') # gives the initial
SNP name

includedGenomeMatrixFile.close()
```

4. GenomeMatrixManipulation.r

```
# Rex M. Williamson
# Manipulations (counts, percentages, etc.) of the
TotalGenomeMatrix.csv file
# to find the minor allele frequencies, call percentages, etc. for each
SNP name for the UNEXCLUDED (total) genome files list
# and adds these columns to any genome matrix (MastrADNIMatrix.csv,
IncludedGenomeMatrix.csv) instead of as a stand-alone file.
# Also, changes '-4' and '--' to NA in those files as well as my
demographics master matrix (ForADNIDemoMatrix.py).


### make changes to TotalGenomeMatrix.csv first ###
genomeDF <- read.csv("TotalGenomeMatrixFile.csv", stringsAsFactors =
FALSE, na.strings = "--") # this is now 1 big data frame and
# any '--' it finds is converted to NA. For demographics, '-4' is
converted to NA.
cat("I've read a file!\n")

# change '-4' and '--' to NA, globally
#genomeDF[genomeDF == '-4'] <- NA # for demographics

genomeMatrix <- as.matrix(genomeDF[,-1]) # excluding snp names [row,
col]
rownames(genomeMatrix) <- genomeDF[,1] # everything except the 1st row
names = snp IDs
# byRow <- genomeMatrix[5,,drop = FALSE] # pulls 1 row out (5) for
trouble-shooting
cat("Sub-matrix created!\n")

SNPstats <- t(apply(genomeMatrix, 1, function(byRow) {
      numberOfFiles <- length(byRow)
      missedCalls <- sum(is.na(byRow)) # gives the number of NAs
(missed calls)
      callRate <- ((numberOfFiles - missedCalls) / numberOfFiles)
      baseTotals <- table(factor(unlist(strsplit(byRow, split = "")),
levels = c("A", "T", "G", "C")))
```

```
      minorAllele <- rev(sort(baseTotals))[2] # get the 2nd-most
frequent base (gives name (A/T/G/C) and number)
      minorAlleleFrequency <- minorAllele / sum(baseTotals)
      c(callRate = callRate, minorAllele = which(names(minorAllele) ==
c("A","T", "G", "C")), minorAlleleFreq = minorAlleleFrequency)
}))
colnames(SNPstats) <- c("callRate", "minorAllele",
"minorAlleleFrequency")
# returns a vector where 1 = 'A', 2 = 'T', 3 = 'G', and 4 = 'C'.
cat("SNP Stats column names created!\n")

genomeDF <- cbind(genomeDF, SNPstats) # binds new columns w/stats to
the MstrADNIMatrix = adds the columns to the end of the matrix
genomeDF$minorAllele <- factor(genomeDF$minorAllele, levels = c(1, 2,
3, 4), labels = c("A", "T", "G", "C"))
write.csv(genomeDF, file = "AnotatedTotalGenomeMatrix.csv")

save(SNPstats, file = "TotalSNPstats.RData") # For making graphs,
histograms of call rates, in R
write.csv(SNPstats, file = "TotalSNPstats.csv")

###### Now, do the same for the IncudedGenomeMatrix.csv file...

### make changes to IncludedGenomeMatrix.csv first ###
genomeDF <- read.csv("IncludedGenomeMatrixFile.csv", stringsAsFactors =
FALSE, na.strings = "--") # this is now 1 big data frame and
# any '--' it finds is converted to NA. For demographics, '-4' is
converted to NA.

# change '-4' and '--' to NA, globally
#genomeDF[genomeDF == '-4'] <- NA # for demographics

genomeMatrix <- as.matrix(genomeDF[,-1]) # excluding snp names [row,
col]
rownames(genomeMatrix) <- genomeDF[,1] # everything except the 1st row
names = snp IDs
# byRow <- genomeMatrix[5,,drop = FALSE] # pulls 1 row out (5) for
trouble-shooting

IncludedSNPstats <- t(apply(genomeMatrix, 1, function(byRow) {
numberOfFiles <- length(byRow)
missedCalls <- sum(is.na(byRow)) # gives the number of NAs (missed
calls)
callRate <- ((numberOfFiles - missedCalls) / numberOfFiles)
baseTotals <- table(factor(unlist(strsplit(byRow, split = "")), levels
= c("A", "T", "G", "C")))
minorAllele <- rev(sort(baseTotals))[2] # get the 2nd-most frequent
base (gives name (A/T/G/C) and number)
minorAlleleFrequency <- minorAllele / sum(baseTotals)
c(callRate = callRate, minorAllele = which(names(minorAllele) ==
c("A","T", "G", "C")), minorAlleleFreq = minorAlleleFrequency)
}))
colnames(IncludedSNPstats) <- c("callRate", "minorAllele",
"minorAlleleFrequency")
```

```
# returns a vector where 1 = 'A', 2 = 'T', 3 = 'G', and 4 = 'C'.

genomeDF <- cbind(genomeDF, IncludedSNPstats) # binds new columns
w/stats to the MstrADNIMatrix = adds the columns to the end of the
matrix
genomeDF$minorAllele <- factor(genomeDF$minorAllele, levels = c(1, 2,
3, 4), labels = c("A", "T", "G", "C"))
write.csv(genomeDF, file = "AnotatedIncludedGenomeMatrix.csv")

save(IncludedSNPstats, file = "IncludedSNPstats.RData") # For making
graphs, histograms of call rates, in R
write.csv(IncludedSNPstats, file = "IncludedSNPstats.csv")

##### make changes to ADNIDemoMatrix.csv #####
demoDF <- read.csv("ADNIDemoMatrix.csv", stringsAsFactors = FALSE,
na.strings = "-4") # this is now 1 big data frame and
# any '--' it finds is converted to NA. For demographics, '-4' is
converted to NA.

save(demoDF, file = "Demographics.RData") # For making graphs,
histograms of call rates, in R
write.csv(demoDF, file = "ADNIDemoMatrix.csv")
```