Chapter 2 - Import and Tidy Data

Gabriel Odom

2018-04-18

Table of Contents

[Overview 1](#_Toc511835054)

[Outline 1](#_Toc511835055)

[GMT Files 2](#_Toc511835056)

[GMT Format Description 2](#_Toc511835057)

[Importing with the read\_gmt Function 3](#_Toc511835058)

[Creating Your Own pathwaySet List 4](#_Toc511835059)

[Import and Tidy an Assay Matrix 4](#_Toc511835060)

[Import with readr 5](#_Toc511835061)

[Tidy the Assay Data Frame 6](#_Toc511835062)

[Subsetting a Tidy Data Frame 7](#_Toc511835063)

[Import and Join Response Data 8](#_Toc511835064)

[Example Tidy Assay and Gene Set 10](#_Toc511835065)

[Review 11](#_Toc511835066)

# Overview

This vignette is the second chapter in the “Pathway Significance Testing with pathwayPCA” workflow. This vignette will discuss importing Gene Matrix Transposed (.gmt) [gene set files](https://software.broadinstitute.org/cancer/software/gsea/wiki/index.php/Data_formats#GMT:_Gene_Matrix_Transposed_file_format_.28.2A.gmt.29) to a list object with class pathwaySet with the read\_gmt function. Also, we will discuss importing assay and response data, and how to make your assay data [tidy](https://www.jstatsoft.org/article/view/v059i10). For our pathway analysis to be meaningful, we need gene expression data (from a microarray or something similar), corresponding subject information (such as weight, type of cancer, or survival time and censoring indicator), and a gene set list.

### Outline

Before we move on, we will outline our steps. After reading this vignette, you should be able to

1. Import a .gmt file and save the gene sets therein as a pathwaySet object using the read\_gmt function.
2. Import an assay .csv file with the read\_csv function from the readr package, and transpose this data frame into “tidy” form with the transpose\_assay function.
3. Import a subject information .csv file, and join (merge) it to the assay data frame with the inner\_join function from the dplyr package.

First, load the pathwayPCA package and the [tidyverse package suite](https://www.tidyverse.org/). If you don’t have the tidyverse package suite, install that first.

# install.packages("tidyverse")  
library(tidyverse)  
#> -- Attaching packages ------------------------------------------------------------------------------------------------------------------- tidyverse 1.2.1 --  
#> v ggplot2 2.2.1 v purrr 0.2.4  
#> v tibble 1.4.2 v dplyr 0.7.4  
#> v tidyr 0.8.0 v stringr 1.3.0  
#> v readr 1.1.1 v forcats 0.3.0  
#> -- Conflicts ---------------------------------------------------------------------------------------------------------------------- tidyverse\_conflicts() --  
#> x dplyr::filter() masks stats::filter()  
#> x dplyr::lag() masks stats::lag()  
library(pathwayPCA)

# GMT Files

GMT files are one form of gene set file officially recognized by the Gene Set Enrichment Analysis committee of the Broad Institute. These Molecular Signatures Database (MSigDB) GMT files can be downloaded from the [MSigDB Collections](http://software.broadinstitute.org/gsea/msigdb/collections.jsp) page.

### GMT Format Description

GMT-formatted files follow a very specific set of rules:

1. Each row of the file represents a gene set, and only one gene set is allowed per line.
2. The first entry in each row is the gene set name. Eg. "KEGG\_STEROID\_BIOSYNTHESIS"
3. The second entry in each row is a brief description of the gene set. Eg. "http://www.broadinstitute.org/gsea/msigdb/cards/KEGG\_STEROID\_BIOSYNTHESIS"
4. The third to the last entry on each row are the genes in the gene set. Eg. "SOAT1" "LSS" "SQLE" "EBP" "CYP51A1" "DHCR7" "CYP27B1" "DHCR24" "HSD17B7" "MSMO1" "FDFT1" "SC5DL" "LIPA" "CEL" "TM7SF2" "NSDHL" "SOAT2"
5. Each entry in each line is seperated by a tab.

### Importing with the read\_gmt Function

Based on the clearly-organized .gmt data format, we are able to write a very fast function to read .gmt files into R. The read\_gmt function takes in a path specifying where your .gmt file is stored, and outputs a list of gene set information.

gene\_set\_ls <- read\_gmt("path\_to\_your\_directory/your\_file\_name.gmt")

The form of this information must be discussed carefully. This gene\_set\_ls object has class pathwaySet and contains the following components:

1. pathways: A named list of character vectors. Each character vector should contain a subset of the -Omes measured in your assay data frame. These pathways should not be too short, otherwise we devolve the problem into simply testing individual genes. Therefore, the pathwayPCA package requires each pathway to have a minimum of three genes with recorded expressions in the predictor data frame.

**Important**: some protein set lists have proteins markers recorded as character numerics, so make sure the feature names of your assay have an overlap with the gene or protein names in the pathwaySet list. Ensure that there is a non-empty overlap between the names in the pathways and the feature names of the assay. Not every feature in your assay data frame will be in a pathway gene set, and not every gene in each pathway will have a measurement in the assay data frame. *However, for meaningful results, there should be a significant overlap between the genes measured in the data frame and the gene names sorted into the pathways.* If your gene set list has very few matching genes in your data frame, then your pathway-based analysis results will be significantly degraded. **Make sure your pathway set and data assay are compatible.**

1. TERMS: A character vector the same length as the pathways list with the proper names of the pathways.
2. GSEA\_link: A character vector the same length as the pathways list with hyperlinks to the MSigDB description card for that pathway.
3. setsize: the number of genes originally recorded in each pathway, stored as an integer vector. NOTE: *this information is calculated and added to the pathway set list later, at Omics-class object creation.* This information is useful to measure the ratio of the number of genes from each pathway expressed in your assay to the number of genes defined to be in that pathway. This ratio should be very close to 1 for best pathway analysis results.

The object itself has the following structure:

gene\_set\_ls  
#> Object with Class(es) 'pathwaySet', 'list' [package 'pathwayPCA'] with 3 elements:   
#> $ pathways :List of 1329  
#> $ TERMS : chr [1:1329] "KEGG\_GLYCOLYSIS\_GLUCONEOGENESIS" ...  
#> $ GSEA\_link: chr [1:1329] "http://www.broadinstitute.org/gsea/msigdb/cards/KEGG\_GLYCOLYSIS\_GLUCONEOGENESIS" ...

This object will be the list supplied to the pathwaySet\_ls argument in the create\_OmicsPath, create\_OmicsSurv, create\_OmicsReg, or create\_OmicsCateg functions.

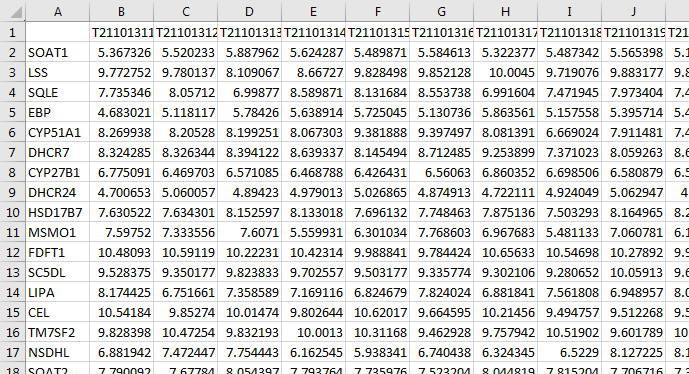
### Creating Your Own pathwaySet List

Additionally, you can create a pathwaySet object from scratch with the create\_pathwaySet function. This may be useful to users who have their pathway information stored in some form other than a .gmt file. You must supply a list of -Omes to the pathways argument, and the proper names of the pathways to the TERMS argument. You could also store any other pertinant pathway information by passing a <name> = <value> pair to this function.

myPathways\_ls <- list(pathway1 = c("Gene1", "Gene2"),  
 pathway2 = c("Gene3", "Gene4", "Gene5"),  
 pathway3 = "Gene6")  
myPathway\_names <- c("KEGG\_IMPORTANT\_PATHWAY\_1",  
 "KEGG\_IMPORTANT\_PATHWAY\_2",  
 "SOME\_OTHER\_PATHWAY")  
create\_pathwaySet(pathways = myPathways\_ls,  
 TERMS = myPathway\_names,  
 website = "URL\_TO\_PATHWAY\_CITATION")  
#> Object with Class(es) 'pathwaySet', 'list' [package 'pathwayPCA'] with 3 elements:   
#> $ pathways:List of 3  
#> $ TERMS : chr [1:3] "KEGG\_IMPORTANT\_PATHWAY\_1" ...  
#> $ website : chr "URL\_TO\_PATHWAY\_CITATION"

# Import and Tidy an Assay Matrix

We assume that the assay data (be it genomic, proteomic, metabolomic, lipidomic, or transcriptomic data) is either in an Excel file or flat text file. For example, your data may look like this:



In this data set, the columns are individual samples, patients, tumors, cell lines, etc. The rows are the -Omic expression measures.

### Import with readr

To import these files in .csv (comma-separated), .fwf (fixed-width), or .txt (tab-delimited), we recommend the [readr package](https://readr.tidyverse.org/). We would read a .csv data file via

assay\_df <- read\_csv("path\_to\_your\_directory/your\_assay.csv")

#> Warning: Missing column names filled in: 'X1' [1]  
#> Parsed with column specification:  
#> cols(  
#> .default = col\_double(),  
#> X1 = col\_character()  
#> )  
#> See spec(...) for full column specifications.

You can read fixed-width files with the read\_fwf function, and general delimited files with read\_delim. Both of these functions are from the readr package. For data in .xls or .xlsx format, we recommend the [readxl](http://readxl.tidyverse.org/) package.

Let’s inspect our assay data frame. Note that the gene names were imported as a character column (see the <chr> tag at the top of the first column). This data import step stored the row names (the gene names) as the first column, and preserved the column names (sample labels) of the data.

assay\_df  
#> # A tibble: 17 x 37  
#> X1 T21101311 T21101312 T21101313 T21101314 T21101315 T21101316  
#> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
#> 1 SOAT1 5.37 5.52 5.89 5.62 5.49 5.58  
#> 2 LSS 9.77 9.78 8.11 8.67 9.83 9.85  
#> 3 SQLE 7.74 8.06 7.00 8.59 8.13 8.55  
#> 4 EBP 4.68 5.12 5.78 5.64 5.73 5.13  
#> 5 CYP51A1 8.27 8.21 8.20 8.07 9.38 9.40  
#> 6 DHCR7 8.32 8.33 8.39 8.64 8.15 8.71  
#> 7 CYP27B1 6.78 6.47 6.57 6.47 6.43 6.56  
#> 8 DHCR24 4.70 5.06 4.89 4.98 5.03 4.87  
#> 9 HSD17B7 7.63 7.63 8.15 8.13 7.70 7.75  
#> 10 MSMO1 7.60 7.33 7.61 5.56 6.30 7.77  
#> 11 FDFT1 10.5 10.6 10.2 10.4 9.99 9.78  
#> 12 SC5DL 9.53 9.35 9.82 9.70 9.50 9.34  
#> 13 LIPA 8.17 6.75 7.36 7.17 6.82 7.82  
#> 14 CEL 10.5 9.85 10.0 9.80 10.6 9.66  
#> 15 TM7SF2 9.83 10.5 9.83 10.0 10.3 9.46  
#> 16 NSDHL 6.88 7.47 7.75 6.16 5.94 6.74  
#> 17 SOAT2 7.79 7.68 8.05 7.79 7.74 7.52  
#> # ... with 30 more variables: T21101317 <dbl>, T21101318 <dbl>,  
#> # T21101319 <dbl>, T21101320 <dbl>, T21101321 <dbl>, T21101322 <dbl>,  
#> # T21101323 <dbl>, T21101324 <dbl>, T21101325 <dbl>, T21101326 <dbl>,  
#> # T21101327 <dbl>, T21101328 <dbl>, T21101329 <dbl>, T21101330 <dbl>,  
#> # T21101331 <dbl>, T21101332 <dbl>, T21101333 <dbl>, T21101334 <dbl>,  
#> # T21101335 <dbl>, T21101336 <dbl>, T21101337 <dbl>, T21101338 <dbl>,  
#> # T21101339 <dbl>, T21101340 <dbl>, T21101341 <dbl>, T21101342 <dbl>,  
#> # T21101343 <dbl>, T21101344 <dbl>, T21101345 <dbl>, T21101346 <dbl>

### Tidy the Assay Data Frame

The assay input to the pathwayPCA package must be in [*tidy data*](https://www.jstatsoft.org/article/view/v059i10) format. The “Tidy Data” format requires that each observation be its own row, and each measurement its own column. This means that we must transpose our assay data frame, while preserving the row and column names.

To do this, we can use the transpose\_assay function. This function takes in a data frame as imported by the readr function based on data in a format similar to that shown above: genes are the rows, gene names are the first column, samples are are the stored in the subsequent columns, and all values in the assay (other than the single names column) are numeric.

(assayT\_df <- transpose\_assay(assay\_df))  
#> # A tibble: 36 x 18  
#> Sample SOAT1 LSS SQLE EBP CYP51A1 DHCR7 CYP27B1 DHCR24 HSD17B7  
#> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
#> 1 T21101311 5.37 9.77 7.74 4.68 8.27 8.32 6.78 4.70 7.63  
#> 2 T21101312 5.52 9.78 8.06 5.12 8.21 8.33 6.47 5.06 7.63  
#> 3 T21101313 5.89 8.11 7.00 5.78 8.20 8.39 6.57 4.89 8.15  
#> 4 T21101314 5.62 8.67 8.59 5.64 8.07 8.64 6.47 4.98 8.13  
#> 5 T21101315 5.49 9.83 8.13 5.73 9.38 8.15 6.43 5.03 7.70  
#> 6 T21101316 5.58 9.85 8.55 5.13 9.40 8.71 6.56 4.87 7.75  
#> 7 T21101317 5.32 10.0 6.99 5.86 8.08 9.25 6.86 4.72 7.88  
#> 8 T21101318 5.49 9.72 7.47 5.16 6.67 7.37 6.70 4.92 7.50  
#> 9 T21101319 5.57 9.88 7.97 5.40 7.91 8.06 6.58 5.06 8.16  
#> 10 T21101320 5.16 9.87 7.42 5.50 7.43 8.68 6.55 4.85 8.20  
#> # ... with 26 more rows, and 8 more variables: MSMO1 <dbl>, FDFT1 <dbl>,  
#> # SC5DL <dbl>, LIPA <dbl>, CEL <dbl>, TM7SF2 <dbl>, NSDHL <dbl>,  
#> # SOAT2 <dbl>

This transposed data frame has the gene names as the column names and the sample names as a column of character (chr) values. Notice that the data itself is 17 genes measured on 36 samples. Before transposition, we had 37 columns because the feature names were stored in the first column. After transposition, we have 36 rows but 18 columns: the first column stores the sample names. This transposed data frame (after filtering to match the response data) will be supplied to the assayData\_df argument in the create\_OmicsPath, create\_OmicsSurv, create\_OmicsReg, or create\_OmicsCateg functions. (*See the* [*Creating Omics Data Objects*](https://gabrielodom.github.io/pathwayPCA/articles/Create_Omics_Objects.html) *vignette for more information on creating Omics\*-class objects.*)

### Subsetting a Tidy Data Frame

If ever we need to extract individual components of a tidy data frame, we can use the assay[row, col] syntax. If we need entire measurements (columns), then we can call the column by name with the assay$ColName syntax. For example,

* If we need the second row of assayT\_df—corresponding to Sample “T21101312”—then we type

assayT\_df[2, ]  
#> # A tibble: 1 x 18  
#> Sample SOAT1 LSS SQLE EBP CYP51A1 DHCR7 CYP27B1 DHCR24 HSD17B7  
#> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
#> 1 T21101312 5.52 9.78 8.06 5.12 8.21 8.33 6.47 5.06 7.63  
#> # ... with 8 more variables: MSMO1 <dbl>, FDFT1 <dbl>, SC5DL <dbl>,  
#> # LIPA <dbl>, CEL <dbl>, TM7SF2 <dbl>, NSDHL <dbl>, SOAT2 <dbl>

* If we need the third column of assayT\_df—corresponding to Gene “LSS”—then we type

assayT\_df[, 3]  
#> # A tibble: 36 x 1  
#> LSS  
#> <dbl>  
#> 1 9.77  
#> 2 9.78  
#> 3 8.11  
#> 4 8.67  
#> 5 9.83  
#> 6 9.85  
#> 7 10.0   
#> 8 9.72  
#> 9 9.88  
#> 10 9.87  
#> # ... with 26 more rows

* If we need the intersection of these two (the expression level of Gene “LSS” in Sample “T21101312”), then we type

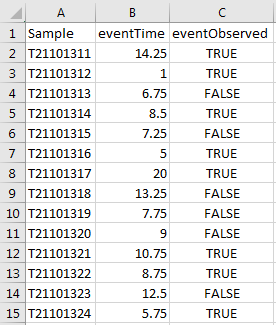
assayT\_df[2, 3]  
#> # A tibble: 1 x 1  
#> LSS  
#> <dbl>  
#> 1 9.78

* If we need the third column of assayT\_df, but we want the result back as a vector instead of a data frame, we have to call the column by name:

assayT\_df$LSS  
#> [1] 9.772752 9.780137 8.109067 8.667270 9.828498 9.852128 10.004496  
#> [8] 9.719076 9.883177 9.871709 7.892335 8.263840 8.215201 9.133894  
#> [15] 10.116890 9.107010 9.820948 9.057552 9.268507 9.382549 7.852329  
#> [22] 8.585521 8.949471 8.931112 9.597493 9.657331 7.800871 9.858547  
#> [29] 9.758955 8.383229 10.562910 9.815668 10.013560 11.094211 9.217539  
#> [36] 10.667547

# Import and Join Response Data

We now have an appropriate pathway set list and a tidy -Omics assay data frame. Let’s imagine that your response data looks something like this:



We now need to import the response information. We can use the read\_csv function once again:

pInfo\_df <- read\_csv("path\_to\_your\_directory/your\_subject\_info.csv")

#> Parsed with column specification:  
#> cols(  
#> Sample = col\_character(),  
#> eventTime = col\_double(),  
#> eventObserved = col\_logical()  
#> )

This data frame has a column for the sample labels (Sample) and the response information. In this case, our response is a survival response with an event time and observation indicator.

pInfo\_df  
#> # A tibble: 36 x 3  
#> Sample eventTime eventObserved  
#> <chr> <dbl> <lgl>   
#> 1 T21101311 14.2 TRUE   
#> 2 T21101312 1.00 TRUE   
#> 3 T21101313 6.75 FALSE   
#> 4 T21101314 8.50 TRUE   
#> 5 T21101315 7.25 FALSE   
#> 6 T21101316 5.00 TRUE   
#> 7 T21101317 20.0 TRUE   
#> 8 T21101318 13.2 FALSE   
#> 9 T21101319 7.75 FALSE   
#> 10 T21101320 9.00 FALSE   
#> # ... with 26 more rows

This pInfo data frame has the sample names as a column of character values, just like the transposed assay data frame. This is crucially important for the “joining” step. We can use the inner\_join function from the dplyr library to only retain the rows of the assayT\_df data frame which have responses in the pInfo data frame and vice versa. This way, every response has matching assay predictors, and every recorded assay sample has a matching response. **This requires you to have a *key* column in both data frames with the same name.** If the key column was called “Sample” in the pInfo\_df data set but “SampleID” in the assay, then the by argument should be changed to c("Sample" = "SampleID"). It’s much nicer to just keep them with the same names, however.

(joinedExperiment\_df <- inner\_join(pInfo\_df, assayT\_df, by = "Sample"))  
#> # A tibble: 36 x 20  
#> Sample eventTime eventObserved SOAT1 LSS SQLE EBP CYP51A1 DHCR7  
#> <chr> <dbl> <lgl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
#> 1 T21101311 14.2 TRUE 5.37 9.77 7.74 4.68 8.27 8.32  
#> 2 T21101312 1.00 TRUE 5.52 9.78 8.06 5.12 8.21 8.33  
#> 3 T21101313 6.75 FALSE 5.89 8.11 7.00 5.78 8.20 8.39  
#> 4 T21101314 8.50 TRUE 5.62 8.67 8.59 5.64 8.07 8.64  
#> 5 T21101315 7.25 FALSE 5.49 9.83 8.13 5.73 9.38 8.15  
#> 6 T21101316 5.00 TRUE 5.58 9.85 8.55 5.13 9.40 8.71  
#> 7 T21101317 20.0 TRUE 5.32 10.0 6.99 5.86 8.08 9.25  
#> 8 T21101318 13.2 FALSE 5.49 9.72 7.47 5.16 6.67 7.37  
#> 9 T21101319 7.75 FALSE 5.57 9.88 7.97 5.40 7.91 8.06  
#> 10 T21101320 9.00 FALSE 5.16 9.87 7.42 5.50 7.43 8.68  
#> # ... with 26 more rows, and 11 more variables: CYP27B1 <dbl>,  
#> # DHCR24 <dbl>, HSD17B7 <dbl>, MSMO1 <dbl>, FDFT1 <dbl>, SC5DL <dbl>,  
#> # LIPA <dbl>, CEL <dbl>, TM7SF2 <dbl>, NSDHL <dbl>, SOAT2 <dbl>

It is vitally important that you check your sample IDs. Obviously the expressed genetic data should pair with the subject information, but **it is your responsibility as the user to confirm that the assay rows match the correct responses.** We can show you the proper methods to check your data, but you are ultimately responsible to use this package properly.

# Example Tidy Assay and Gene Set

Included in this package, we have a small tidy assay and corresponding gene subset list. We will load and inspect this assay. This data set has 656 gene expression measurements on 250 colon cancer patients. Further notice that the assay and survival response information have already been merged.

data("colonSurv\_df")  
str(colonSurv\_df, list.len = 10)  
#> Classes 'tbl\_df', 'tbl' and 'data.frame': 250 obs. of 658 variables:  
#> $ OS\_time : num 64.9 59.8 62.4 54.5 46.3 ...  
#> $ OS\_event: int 0 0 0 0 1 0 0 0 1 0 ...  
#> $ JUN : num 9.29 9.13 9.37 10.6 8.7 ...  
#> $ SOS2 : num 5.48 6.35 5.67 4.94 5.6 ...  
#> $ PAK3 : num 8.21 8.33 7.82 8.79 8.75 ...  
#> $ RAF1 : num 8.03 7.94 7.74 7.64 8.05 ...  
#> $ PRKCB : num 5.49 6.26 6.05 5.37 6.07 ...  
#> $ BTC : num 6.65 7.02 7.52 6.87 6.49 ...  
#> $ SHC1 : num 8.26 8.39 8.69 7.81 8.45 ...  
#> $ PRKCA : num 8.94 9.61 8.4 9.8 8.21 ...  
#> [list output truncated]

We also have a small list of 15 pathways which correspond to our example colon cancer assay. We have curated this list to include seven significant pathways and eight non-significant pathways.

data("colon\_pathwaySet")  
colon\_pathwaySet  
#> Object with Class(es) 'pathwaySet', 'list' [package 'pathwayPCA'] with 2 elements:   
#> $ pathways:List of 15  
#> $ TERMS : chr [1:15] "KEGG\_PENTOSE\_PHOSPHATE\_PATHWAY" ...  
str(colon\_pathwaySet$pathways, list.len = 10)  
#> List of 15  
#> $ pathway3 : chr [1:27] "RPE" "RPIA" "PGM2" "PGLS" ...  
#> $ pathway60 : chr [1:64] "RPE65" "CYP3A5" "UGT2B28" "CYP4A11" ...  
#> $ pathway87 : chr [1:87] "JUN" "SOS2" "PAK3" "RAF1" ...  
#> $ pathway120 : chr [1:89] "HLA-DOA" "HLA-DOB" "KLRC3" "KLRD1" ...  
#> $ pathway176 : chr [1:54] "CASP9" "SOS2" "E2F1" "PRKCB" ...  
#> $ pathway177 : chr [1:30] "HLA-DRB4" "HLA-DRB5" "HLA-DOA" "HLA-DOB" ...  
#> $ pathway187 : chr [1:16] "IKBKG" "CHUK" "EP300" "RELA" ...  
#> $ pathway266 : chr [1:11] "PRF1" "DFFA" "DFFB" "HMGB2" ...  
#> $ pathway390 : chr [1:29] "JUN" "BAG4" "CASP8" "MAPK8" ...  
#> $ pathway413 : chr [1:23] "PLD1" "RAF1" "EPHB2" "VAV1" ...  
#> [list output truncated]

The pathway set list and tidy assay (with matched patient response information) are all the information we need to create an Omics-class data container.

# Review

We now summarize our steps so far. We have

1. Imported a .gmt file and saved the gene sets therein as a pathwaySet object using the read\_gmt function.
2. Imported an assay .csv file with the read\_csv function from the readr package, and transposed this data frame into “tidy” form with the transpose\_assay function.
3. Imported a subject information .csv file, and joined it to the assay data frame with the inner\_join function from the dplyr package.

Now we are prepared to create our first Omics-class object for analysis with either AES-PCA or Supervised PCA. Please read the next vignette: [Creating Omics Data Objects](https://gabrielodom.github.io/pathwayPCA/articles/C3-Create_Omics_Objects.html).