RCPA: An Open-Source R Package for Data Processing, Differential Analysis, Consensus Pathway Analysis, and Visualization

Hung Nguyen,[1,6] Ha Nguyen,[1,6] Zeynab Maghsoudi,[2] Bang Tran,[3] Sorin Draghici,[4,5] and Tin Nguyen[1,7]

[1]Department of Computer Science and Software Engineering, Auburn University, Auburn, Alabama
[2]Department of Computer Science and Engineering, University of Nevada, Reno, Nevada
[3]College of Engineering and Computer Science, California State University, Sacramento, California
[4]Department of Computer Science, Wayne State University, Detroit, Michigan
[5]AdvaitaBio, Ann Arbor, Michigan
[6]These authors contributed equally to this work
[7]Corresponding author: *tinn@auburn.edu*

Published in the Bioinformatics section

Identifying impacted pathways is important because it provides insights into the biology underlying conditions beyond the detection of differentially expressed genes. Because of the importance of such analysis, more than 100 pathway analysis methods have been developed thus far. Despite the availability of many methods, it is challenging for biomedical researchers to learn and properly perform pathway analysis. First, the sheer number of methods makes it challenging to learn and choose the correct method for a given experiment. Second, computational methods require users to be savvy with coding syntax, and comfortable with command-line environments, areas that are unfamiliar to most life scientists. Third, as learning tools and computational methods are typically implemented only for a few species (i.e., human and some model organisms), it is difficult to perform pathway analysis on other species that are not included in many of the current pathway analysis tools. Finally, existing pathway tools do not allow researchers to combine, compare, and contrast the results of different methods and experiments for both hypothesis testing and analysis purposes. To address these challenges, we developed an open-source R package for Consensus Pathway Analysis (RCPA) that allows researchers to conveniently: (1) download and process data from NCBI GEO; (2) perform differential analysis using established techniques developed for both microarray and sequencing data; (3) perform both gene set enrichment, as well as topology-based pathway analysis using different methods that seek to answer different research hypotheses; (4) combine methods and datasets to find consensus results; and (5) visualize analysis results and explore significantly impacted pathways across multiple analyses. This protocol provides many example code snippets with detailed explanations and supports the analysis of more than 1000 species, two pathway databases, three differential analysis techniques, eight pathway analysis tools, six meta-analysis methods, and two consensus analysis techniques. The package is freely available on the CRAN repository. © 2024 The Authors. Current Protocols published by Wiley Periodicals LLC.

Basic Protocol 1: Processing Affymetrix microarrays
Basic Protocol 2: Processing Agilent microarrays

Current Protocols e1036, Volume 4
Published in Wiley Online Library (wileyonlinelibrary.com).
doi: 10.1002/cpz1.1036

Support Protocol: Processing RNA sequencing (RNA-Seq) data
Basic Protocol 3: Differential analysis of microarray data (Affymetrix and Agilent)
Basic Protocol 4: Differential analysis of RNA-Seq data
Basic Protocol 5: Gene set enrichment analysis
Basic Protocol 6: Topology-based (TB) pathway analysis
Basic Protocol 7: Data integration and visualization

Keywords: differential analysis • integration and visualization • microarray • pathway analysis • RNA sequencing

How to cite this article:
Nguyen, H., Nguyen, H., Maghsoudi, Z., Tran, B., Draghici, S., & Nguyen, T. (2024). RCPA: An open-source R package for data processing, differential analysis, consensus pathway analysis, and visualization. *Current Protocols*, *4,* e1036. doi: 10.1002/cpz1.1036

INTRODUCTION

Together with the ability to generate a large amount of data, high-throughput technologies have also brought the challenge of translating such data into biological knowledge. Regardless of the laboratory technology being used, a comparative study (e.g., disease vs healthy) typically yields a set of genes or proteins that are differentially expressed (DE) between the two phenotypes. Though important, these lists of DE genes do not explain the mechanisms involved in the underlying conditions by themselves. To translate DE genes into biological knowledge, researchers have developed knowledge bases that capture the knowledge about the function, location and other properties of the genes and gene products. One of the first such knowledge bases was the Gene Ontology (GO) (The Gene Ontology Consortium, 2021). GO consists of a controlled vocabulary of terms that describe biological processes, cellular locations, and biochemical functions, as well as the relationships between them. These together form an ontology. Furthermore, GO also provides associations between genes and these terms, thus capturing the knowledge about the gene functions and localization within the cell.

As soon as such annotations started to become available, analysis methods were developed to take advantage of them. The first analysis approach was the over-representation analysis (ORA) that identifies the gene sets, e.g., GO terms, which are enriched in differentially expressed (DE) genes (Beissbarth & Speed, 2004; Hosack et al., 2003; Khatri et al., 2002). The drawbacks of ORA approaches include that they: (1) only consider the number of DE genes and ignore the actual expression changes, and (2) assume that genes are independent, which is not true. Functional Class Scoring (FCS) approaches have been developed to address these drawbacks. These include the GSEA family of methods (Efron & Tibshirani, 2007; Mootha et al., 2003; Subramanian et al., 2005). The main improvement is that these approaches can identify situations in which small but coordinated changes in the expression of functionally related genes are important.

While GO captures the associations between genes and various biological processes, cellular locations, and biochemical functions, it does not provide any direct information about the interactions between the genes and/or gene products. Basically, each GO term can be seen as an unordered, unstructured set of genes that are associated with it. The next step was taken by trying to describe the complex phenomena that take place in living organisms by describing the various signals and interactions between genes, gene products, and/or metabolites. These are captured in directed graphs that are commonly referred to as pathways. Examples include the Kyoto Encyclopedia of Genes and Genomes (KEGG) (Kanehisa et al., 2017) and Reactome (Croft et al., 2014). Pathways can be further divided into gene signaling pathways and metabolic pathways. In gene signaling pathways, the nodes represent genes, and the edges represent signals or interactions between genes and/or gene products. In a metabolic pathway, nodes represent biochemical molecules and edges represent reactions that take place between such biomolecules. The reactions are carried out by enzymes that are coded by genes so that in a metabolic pathway, genes are associated with the edges rather than the nodes.

Once such sophisticated pathway models have become available, the challenge was to identify those pathways that are important in a given phenotype. The first analysis approaches for pathways were to simply consider the pathways as simple sets of genes and use the methods previously developed for gene set analysis: ORA and FCS. However, ORA and FCS are limited because they do not account for the hierarchical structure of pathways or interactions between genes. Topology-based (TB) approaches were developed to further incorporate knowledge about gene topology and network in their hypothesis testing (Draghici et al., 2007; Tarca et al., 2009). Topology-based approaches are able to consider all important elements ignored by ORA and FCS methods, i.e., the positions and roles of all the genes in every pathway, the direction and type of signals between them, etc. Because of their advantages, many more topology-based approaches have since been proposed (Glaab et al., 2010; Gu et al., 2012; Gu & Wang, 2013; Mitrea et al., 2013; Nguyen, Diaz, et al., 2016; Nguyen et al., 2020).

Despite the availability of many pathway analysis methods, it is tremendously challenging for biomedical researchers to learn and to properly perform pathway analysis. First, the sheer number of methods makes it demanding for scientists to learn and choose the correct method for their experiments. As reported in our previous benchmarking article (Nguyen et al., 2019), there is no single method that is always superior to others. The suitability of a method depends on the research hypothesis users seek to answer. Second, testing available methods requires users to be savvy with coding syntax and comfortable with command-line environments, areas that are unfamiliar to most life scientists. Third, many available tools are typically implemented only for human and a few model organisms, thus making it difficult to perform pathway analysis on hundreds of other species that are not included in current analysis tools. Finally, meta-analysis and consensus analysis are missing from many existing pathway analysis tools. Meta-analysis techniques focus on combining independent but related studies to increase statistical power (Normand, 1999) whereas consensus analysis combines analysis results obtained from methods with different underlying hypotheses for a better understanding of biological mechanisms (Nguyen et al., 2021).

To address the challenges above, we introduce the R package for Consensus Pathway Analysis (RCPA) that implements a complete analysis pipeline, including: (1) downloading and processing data from NCBI Gene Expression Omnibus, (2) performing differential analysis using techniques developed for both microarray and sequencing data, (3) performing systems-level analysis using different methods for enrichment analysis and topology-based (TB) analysis, (4) performing meta-analysis and consensus analysis, and (5) visualizing analysis results and exploring significantly impacted pathways across multiple analyses. The package supports the analysis of $>$1000 species, two pathway databases, three differential analysis techniques, eight pathway analysis tools, six meta-analysis methods, and two consensus analysis techniques. The package is freely available on the CRAN repository (see Internet Resources).

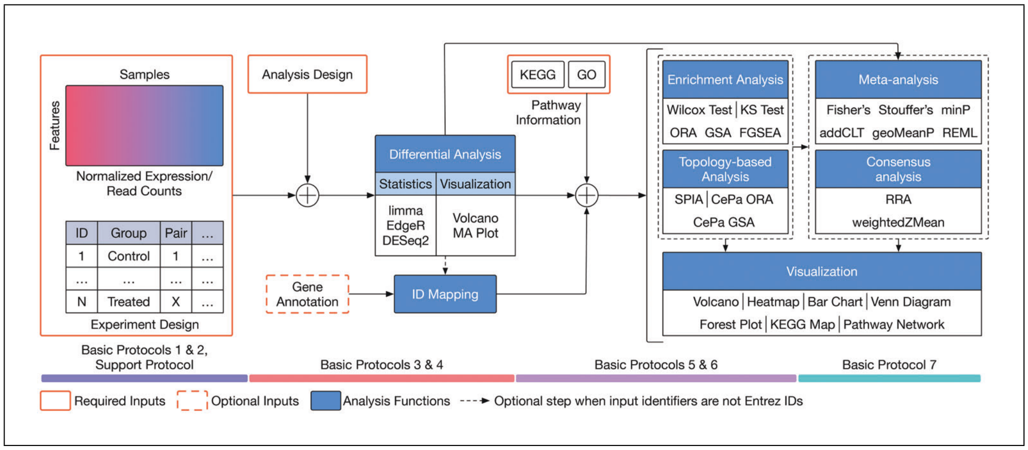


Figure 1 Overview of the analysis pipeline implemented in the RCPA package. The pipeline consists of eight protocols that can be divided into four main modules: (1) data processing, (2) differential analysis, (3) systems-level analysis (gene set enrichment and topology-based pathway analysis), and (4) integrative analysis (meta-analysis and consensus analysis) and visualization. The colored bars at the bottom of the figure represent the four modules, each labeled with the relevant protocols. Each protocol introduces step-by-step instructions for analysis using established methods and visualization techniques.

STRATEGIC PLANNING

Flowchart

Figure 1 shows the analysis workflow implemented in the RCPA package. The full pipeline consists of eight protocols that can be divided into four main modules: (1) data processing (Basic Protocols 1 and 2, and Support Protocol), (2) differential analysis (Basic Protocols 3 and 4), (3) systems-level analysis (Basic Protocols 5 and 6), and (4) integrative analysis (Basic Protocol 7). Each protocol includes established analysis methods and visualization techniques. The pipeline is designed to work with transcriptome data, which can be either the expression data obtained from microarrays or raw read counts obtained from RNA sequencing (RNA-Seq).

The first module is data processing. We introduce Basic Protocols 1 and 2, and Support Protocol that can process Affymetrix, Agilent, and RNA-Seq, respectively. The output of this module is a `SummarizedExperiment` object that stores both the expression data and metadata (i.e., sample information, condition, tissue, etc.).

The second module focuses on differential analysis. We introduce Basic Protocols 3 and 4 that provide instructions for performing differential analysis of microarray and RNA-Seq data, respectively. The module includes differential analysis techniques available in limma (Ritchie et al., 2015), DESeq2 (Love et al., 2014), and edgeR (Robinson et al., 2010). The input of the module is the `SummarizedExperiment` object obtained from the first module. The output is a `SummarizedExperiment` object that includes both the input (expression and metadata) and the differential analysis results.

The third module focuses on systems-level analysis. We introduce Basic Protocols 5 and 6 for gene set enrichment analysis and topology-based (TB) pathway analysis, respectively. The main difference between gene set enrichment and TB pathway analysis is that the former treats each pathway as a set of genes, whereas the latter considers gene interactions and pathway topology. Basic Protocol 5 covers five gene set enrichment analysis techniques: the Wilcoxon test (Wilcoxon, 1992), the Kolmogorov-Smirnov (KS) test (Massey Jr, 1951), over-representation analysis (ORA) (Huang et al., 2009; Khatri et al., 2002), fast gene set enrichment analysis (FGSEA) (Korotkevich et al., 2021; Sergushichev, 2016), and gene set analysis (GSA) (Efron & Tibshirani, 2007). Basic

Protocol 6 covers three TB pathway analysis methods: signaling pathway impact analysis (SPIA) (Draghici et al., 2007; Tarca et al., 2009), centrality-based pathway enrichment for ORA extension (CePa ORA), and for GSA extension (CePa GSA) (Gu et al., 2012; Gu & Wang, 2013). The input of the third module is the `SummarizedExperiment` object obtained from the second module. The output is a data frame that includes the systems-level analysis results.

The fourth module focuses on integrative analysis. We introduce Basic Protocol 7 for meta-analysis and consensus analysis. Meta-analysis includes a range of techniques to integrate independent but related datasets to increase statistical power and accuracy. Meta-analysis can be performed at both gene and pathway levels to find robust sets of differentially expressed genes and impacted pathways. In contrast, the main objective of consensus pathway analysis is to allow users to see the differences, as well as the consensus results across many methods that rely on distinctively different hypotheses. Consensus analysis can also be extended to compare multiple experiments and computational methods at the same time, so that users can compare different hypotheses, experimental designs, and technologies. The package includes six meta-analysis methods to combine *p*-values and statistics across independent datasets: Fisher's method (Fisher, 1925), Stouffer's method (Stouffer et al., 1949), addCLT (Nguyen et al., 2017; Nguyen, Tagett, et al., 2016), minimum *p*-value (Tippett, 1931), geometric mean (Vovk & Wang, 2020), and restricted maximum likelihood (REML) (Viechtbauer, 2005). The package also implements two methods for consensus analysis: robust rank aggregation (RRA) (Kolde et al., 2012) for combining pathway ranks, and weighted mean for combining z-values obtained from multiple analyses.

QuickStart

Throughout the eight protocols in this article, we consistently use three datasets of three data platforms (GSE5281 of Affymetrix, GSE61196 of Agilent, and GSE153873 of RNA-Seq) in our examples. The data and analysis results of all three datasets are pre-saved by the package and this allows users to skip any protocol. For example, users can go directly to Basic Protocol 7 and perform data visualization without executing any code from the preceding protocols. They only need to call the function `RCPA::loadData()` to load the results from the preceding protocols before running the code in visualization. We also include the code for `RCPA::loadData()` to load necessary data at the beginning of each protocol.

For the convenience of readers, we also created Table 1 to list all data objects obtained from each protocol. The first column of the table shows the names of the objects, while the second column explains the content of each object. Users can load any of the above objects using the function `RCPA::loadData()`. For example, users can simply use `affyDEExperiment <- RCPA::loadData("affyDEExperiment")` to load the differential analysis results of the Affymetrix dataset GSE5281 stored in the object `affyDEExperiment`. The availability of pre-saved data objects would allow users to test any of the eight protocols without the need of going through the preceding protocols. Note that the objects of Basic Protocol 7 (Data Integration and Visualization) are not listed here because it is the last protocol described in this article, and its data objects are not used in any other protocols. We also organized all code snippets in this article into a single Jupyter notebook, and it is freely available on GitHub (see Internet Resources).

PROCESSING AFFYMETRIX MICROARRAYS

BASIC PROTOCOL 1

Basic Protocol 1 guides users through the process of converting raw CEL files into a `SummarizedExperiment` object, which is a data structure designed for efficient downstream analysis. There are three main steps in this protocol: (1) preparing the CEL

Table 1 Saved Data Objects Obtained from Protocols For Data Processing, Differential Analysis, and Pathway Analysis[a]

Object name	Description
Basic Protocol 1: Processing Affymetrix data	
affyDataset	Affymetrix dataset GSE5281 after pre-processing and normalization
Basic Protocol 2: Processing Agilent data	
agilDataset	Agilent dataset GSE61196 after pre-processing and normalization
Support Protocol: Processing RNA-Seq data	
RNASeqDataset	RNA-Seq dataset GSE153873 after pre-processing and normalization
Basic Protocol 3: Differential analysis of microarray data	
affyDEExperiment	Differential analysis results of the Affymetrix dataset GSE5281
agilDEExperiment	Differential analysis results of the Agilent dataset GSE61196
Basic Protocol 4: Differential analysis of RNA-Seq data	
RNASeqDEExperiment	Differential analysis results of the RNA-Seq dataset GSE153873
Basic Protocol 5: Gene set enrichment analysis	
KEGGGenesets	Gene sets downloaded from KEGG
GOTerms	GO terms downloaded from Gene Ontology (GO)
affyKSResult	Enrichment results of the Affymetrix dataset GSE5281 using KS test
affyWilcoxResult	Enrichment results of the Affymetrix dataset GSE5281 using Wilcoxon test
affyORAResult	Enrichment results of the Affymetrix dataset GSE5281 using ORA
affyFGSEAResult	Enrichment results of the Affymetrix dataset GSE5281 using FGSEA
affyGSAResult	Enrichment results of the Affymetrix dataset GSE5281 using GSA
agilKSResult	Enrichment results of the Agilent dataset GSE61196 using KS test
agilWilcoxResult	Enrichment results of the Agilent dataset GSE61196 using Wilcoxon test
agilORAResult	Enrichment results of the Agilent dataset GSE61196 using ORA
agilFGSEAResult	Enrichment results of the Agilent dataset GSE61196 using FGSEA
agilGSAResult	Enrichment results of the Agilent dataset GSE61196 using GSA
RNASeqKSResult	Enrichment results of the RNA-Seq dataset GSE153873 using KS test
RNASeqWilcoxResult	Enrichment results of the RNA-Seq dataset GSE153873 using Wilcoxon test
RNASeqORAResult	Enrichment results of the RNA-Seq dataset GSE153873 using ORA
RNASeqFGSEAResult	Enrichment results of the RNA-Seq dataset GSE153873 using FGSEA
RNASeqGSAResult	Enrichment results of the RNA-Seq dataset GSE153873 using GSA
Basic Protocol 6: Topology-based (TB) pathway analysis	
SPIANetwork	KEGG pathway graphs for TP analysis using SPIA
CePaNetwork	KEGG pathway graphs for TP analysis using CePa ORA and CePa GSA
affySPIAResult	TB pathway analysis results of the Affymetrix dataset GSE5281 using SPIA
affyCePaORAResult	TB pathway analysis results of the Affymetrix dataset GSE5281 using CePa ORA
affyCePaGSAResult	TB pathway analysis results of the Affymetrix dataset GSE5281 using CePa GSA
agilSPIAResult	TB pathway analysis results of the Agilent dataset GSE61196 using SPIA
agilCePaORAResult	TB pathway analysis results of the Agilent dataset GSE61196 using CePa ORA
agilCePaGSAResult	TB pathway analysis results of the Agilent dataset GSE61196 using CePa GSA

(*Continued*)

Table 1 Saved Data Objects Obtained from Protocols For Data Processing, Differential Analysis, and Pathway Analysis[a], *continued*

Object name	Description
RNASeqSPIAResult	TB pathway analysis results of the RNA-Seq dataset GSE153873 using SPIA
RNASeqCePaORAResult	TB pathway analysis results of the RNA-Seq dataset GSE153873 using CePa ORA
RNASeqCePaGSAResult	TB pathway analysis results of the RNA-Seq dataset GSE153873 using CePa GSA

[a] Users can load any of the above objects using the function RCPA::loadData(). For example, users can use affyDEExperiment <- RCPA::loadData("affyDEExperiment") to load the differential analysis results of the Affymetrix dataset GSE5281 stored in the object affyDEExperiment. Note that the objects of Basic Protocol 7 (Data Integration and Visualization) are not listed here because it is the last protocol, and its data objects are not used in any other protocols.

files and sample information file, (2) processing the CEL files to obtain the expression data, and (3) creating the `SummarizedExperiment` object.

Necessary Resources

Hardware

An internet-connected computer or laptop with at least 8 GB of RAM and 20 GB of free hard drive space

Software

R runtime environment (version 4.0.0 or later)

RCPA package from the CRAN repository (see Internet Resources)

This can be done by simply executing the following command in the R console:

```
install.packages("RCPA")
```

Files

A list of CEL.gz files

CEL files are the raw data files generated by Affymetrix microarray scanners. The .gz extension indicates that the files are compressed using the gzip program. The raw content of the CEL files can be read by any text editor. Figure 2 shows an example CEL file. In this protocol, the example CEL files used for analysis will be downloaded from the GEO dataset with the accession number GSE5281.

A spreadsheet containing sample information in CSV or TSV format

In this spreadsheet, each row represents a sample, and each column represents its attribute, e.g., sample ID, disease status, tissue, etc. In this protocol, we will create the example spreadsheet by extracting the sample information from the GEO dataset GSE5281.

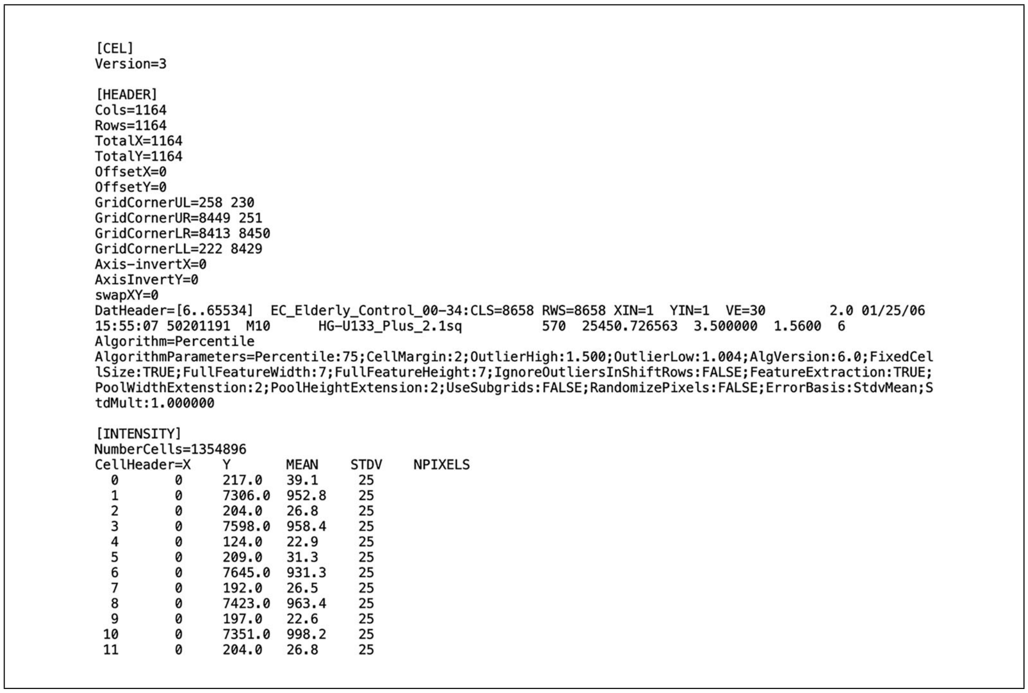
Preparing the CEL files and sample information file

To proceed with the data processing, users need to organize all CEL files in a single folder. In the following example, we will download the CEL files from the GEO dataset GSE5281 using the `downloadGEO()` function implemented in the RCPA package. If users already have the CEL files, they can skip this step and go directly to the next step.

1. Create a local directory to save the data:

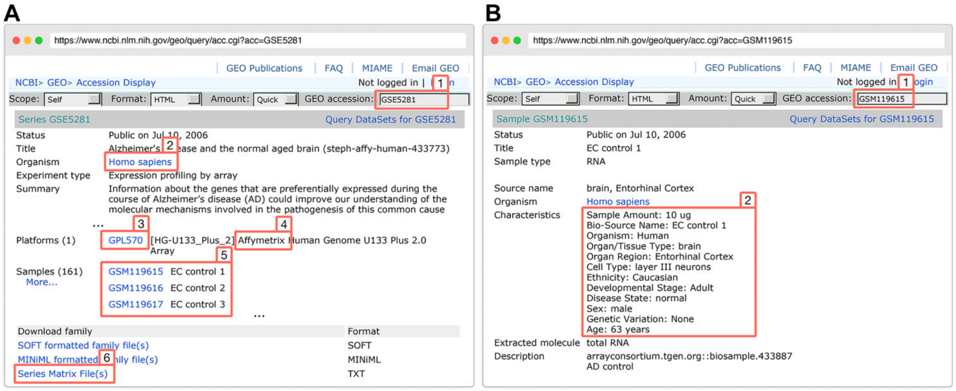
```
userPath <- tempdir() # or user-defined directory path
downloadPath <- file.path(userPath, "GSE5281")
if(!dir.exists(downloadPath)) dir.create(downloadPath)
```

This step creates a local directory where we can download the data. This practice enables users to conveniently reuse the data without repetitive downloads, a particularly advantageous approach when dealing with large datasets that entail significant download times. The first line of code assigns the storage path for the downloaded data to a variable named



```
[CEL]
Version=3

[HEADER]
Cols=1164
Rows=1164
TotalX=1164
TotalY=1164
OffsetX=0
OffsetY=0
GridCornerUL=258 230
GridCornerUR=8449 251
GridCornerLR=8413 8450
GridCornerLL=222 8429
Axis-invertX=0
AxisInvertY=0
swapXY=0
DatHeader=[6..65534]  EC_Elderly_Control_00-34:CLS=8658 RWS=8658 XIN=1  YIN=1  VE=30        2.0 01/25/06
15:55:07 50201191  M10      HG-U133_Plus_2.1sq          570  25450.726563  3.500000  1.5600  6
Algorithm=Percentile
AlgorithmParameters=Percentile:75;CellMargin:2;OutlierHigh:1.500;OutlierLow:1.004;AlgVersion:6.0;FixedCel
lSize:TRUE;FullFeatureWidth:7;FullFeatureHeight:7;IgnoreOutliersInShiftRows:FALSE;FeatureExtraction:TRUE;
PoolWidthExtenstion:2;PoolHeightExtension:2;UseSubgrids:FALSE;RandomizePixels:FALSE;ErrorBasis:StdvMean;S
tdMult:1.000000

[INTENSITY]
NumberCells=1354896
CellHeader=X    Y       MEAN    STDV    NPIXELS
  0       0     217.0   39.1     25
  1       0     7306.0  952.8    25
  2       0     204.0   26.8     25
  3       0     7598.0  958.4    25
  4       0     124.0   22.9     25
  5       0     209.0   31.3     25
  6       0     7645.0  931.3    25
  7       0     192.0   26.5     25
  8       0     7423.0  963.4    25
  9       0     197.0   22.6     25
 10       0     7351.0  998.2    25
 11       0     204.0   26.8     25
```

Figure 2 An example of the content of a CEL file. The CEL file is a text file that contains the raw data generated by Affymetrix microarray scanners.



A
B
1
2
3
4
5
6
1
2
GEO Publications
FAQ
MIAME
Email GEO
NCBI> GEO> Accession Display
Not logged in | Login
Query DataSets for GSE5281
Query DataSets for GSM119615
More...

https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE5281

Scope: Self Format: HTML Amount: Quick GEO accession: GSE5281

Series GSE5281

Status Public on Jul 10, 2006
Title Alzheimer's disease and the normal aged brain (steph-affy-human-433773)
Organism Homo sapiens
Experiment type Expression profiling by array
Summary Information about the genes that are preferentially expressed during the course of Alzheimer's disease (AD) could improve our understanding of the molecular mechanisms involved in the pathogenesis of this common cause
...
Platforms (1) GPL570 [HG-U133_Plus_2] Affymetrix Human Genome U133 Plus 2.0 Array
Samples (161)
GSM119615 EC control 1
GSM119616 EC control 2
GSM119617 EC control 3
...

Download family	Format
SOFT formatted family file(s)	SOFT
MINiML formatted family file(s)	MINiML
Series Matrix File(s)	TXT

https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM119615

Scope: Self Format: HTML Amount: Quick GEO accession: GSM119615

Sample GSM119615

Status Public on Jul 10, 2006
Title EC control 1
Sample type RNA

Source name brain, Entorhinal Cortex
Organism Homo sapiens
Characteristics Sample Amount: 10 ug
Bio-Source Name: EC control 1
Organism: Human
Organ/Tissue Type: brain
Organ Region: Entorhinal Cortex
Cell Type: layer III neurons
Ethnicity: Caucasian
Developmental Stage: Adult
Disease State: normal
Sex: male
Genetic Variation: None
Age: 63 years
Extracted molecule total RNA
Description arrayconsortium.tgen.org::biosample.433887
AD control

Figure 3 Querying results of the Affymetrix dataset GSE5281 from NCBI GEO. (**A**) Metadata of the dataset GSE5281. The following information can be found in this window: (**A1**) GEO accession ID, (**A2**) Organism of study, (**A3**) Platform, (**A4**) Technology, (**A5**) List of samples, and (**A6**) Preprocessed data. (**B**) Query results of the sample GSM119615 in the dataset. Users can find the following information in this window: (**B1**) Sample ID, and (**B2**) Sample characteristics.

`downloadPath`. To exemplify this concept, we use the `tempdir()` function, which returns a character string representing the path to a directory specific to the current session. In other words, each time a new R session is initiated, a temporary directory is automatically created, and it remains accessible only for that R session. In this example, the value of `downloadPath` is set to `"userPath/GSE5281"`. Nevertheless, it is advisable for users to specify a local directory of their choice for data storage and to remove it when it is no longer used. In the second line, we check whether the folder exists. If we do not have that folder, we can create it by using the function `dir.create()`.

2. Browse and search for the underlying dataset on the GEO website.

Before executing the function `downloadGEO()` to download the data, we need to collect the metadata that will be used as parameters of this function. As shown in Figure 3, searching for a dataset in GEO is relatively straightforward. We only need to *provide the accession number of the dataset (e.g., GSE5281) in the search box at https://www.ncbi.nlm.nih.gov/geo/. When the search is done, the webpage displays many details of the dataset, including published date, title, organism, experiment type, dataset summary, and other information. In this example, the platform of the dataset is "GPL570", and the protocol is "affymetrix", as highlighted in boxes 3 and 4. We will use these values as input in the function `downloadGEO()`.*

3. Download the Affymetrix dataset from GEO using the function `downloadGEO()`:

```
# download the data
downloadedFiles <- RCPA::downloadGEO(GEOID = "GSE5281", platform = "GPL570", protocol =
"affymetrix", destDir = downloadPath)
# display the list of downloaded files
print(head(downloadedFiles))

# console output
[1] "metadata.csv" "GSM119615.CEL.gz" "GSM119616.CEL.gz" "GSM119617.CEL.gz"
[5] "GSM119618.CEL.gz" "GSM119619.CEL.gz"
```

The function `downloadGEO()` requires the following parameters to download an Affymetrix dataset from GEO: (1) `GEOID`, a character parameter specifying the GEO accession ID of the desired dataset; (2) `platform`, a character parameter specifying the platform used to generate the GEO dataset in the first parameter; (3) `protocol`, a character parameter indicating the protocol of the GEO dataset; and (4) `destDir`, a character parameter indicating a user-defined path to save downloaded data. Given these parameters, the function downloads the raw CEL files and metadata for the samples and stores them in the directory specified by users. This function returns a list of downloaded files, which is assigned to the `downloadedFiles` variable. The first element of the list is the metadata file, and the remaining elements are the CEL files.

Reading sample information and processing CEL files

4. Read the sample information from the metadata file:

```
# read the metadata file
affySampleInfo <- read.csv(file.path(downloadPath, "metadata.csv"))

# Display the metadata
print(head(affySampleInfo[, c("geo_accession", "characteristics_ch1.4",
"characteristics_ch1.8")]))

# console output
geo_accession   characteristics_ch1.4        characteristics_ch1.8
GSM119615       Organ Region: Entorhinal     Cortex Disease State: normal
GSM119616       Organ Region: Entorhinal     Cortex Disease State: normal
GSM119617       Organ Region: Entorhinal     Cortex Disease State: normal
GSM119618       Organ Region: Entorhinal     Cortex Disease State: normal
GSM119619       Organ Region: Entorhinal     Cortex Disease State: normal
GSM119620       Organ Region: Entorhinal     Cortex Disease State: normal
```

Here, we use the function `read.csv()` to read the file `metadata.csv` generated in the previous step in the directory `downloadPath`. The output of this function is a data frame, which is assigned to the `affySampleInfo` variable. Next, we print some rows and some columns of the data frame to the console. Depending on the dataset, the metadata file might contain different columns. In the above example, we chose to print the columns `geo_accession`, `characteristics_ch1.4`, and `characteristics_ch1.8` that represent sample ID, organ region, and sample condition.

5. Process the CEL files and obtain the expression matrix:

```
# read the CEL files
affyExprs <- RCPA::processAffymetrix(dir = downloadPath, samples =
affySampleInfo$geo_accession)

# display the expression matrix
```

```
print(head(affyExprs, c(5, 6)))

# console output

                   GSM119615    GSM119616    GSM119617    GSM119618    GSM119619    GSM119620
1007_s_at          3.043234     3.055157     3.144277     3.150378     3.084336     2.989966
1053_at            1.698974     1.645050     1.618537     1.589216     1.676278     1.581733
117_at             1.795751     1.770719     1.805597     1.995794     1.688068     1.961556
121_at             2.553174     2.668456     2.801450     2.784216     2.588638     2.692849
1255_g_at          1.626691     1.940055     1.663162     1.483875     2.354096     1.671216
```

Here, we use the function `processAffymetrix()` to process the CEL files and obtain the expression matrix. This function requires two parameters: (1) `dir`, a character parameter specifying the path to the directory containing the CEL files; and (2) `samples`, a character vector specifying the list of sample IDs to be processed. The sample IDs are also the names of the CEL files without the ".CEL.gz" extension. For example, the sample ID of the CEL file `GSM119615.CEL.gz` is `GSM119615`. In this code snippet, the `dir` parameter is set to the `downloadPath` variable, which is the path to the directory containing the CEL files downloaded from the GEO dataset GSE5281 in the previous step. The samples parameter is set to the `geo_accession` column of the `affySampleInfo` data frame, which contains the sample IDs that match the names of the CEL files.

In this function, all CEL files are read using the oligo package (Carvalho & Irizarry, 2010), followed by a three-step normalization process. These steps include background correction using the Robust Multi-array Average algorithm (RMA) (Harbron et al., 2007), quantile normalization, and median-polish summarization. The background correction function subtracts a measure of the background noise from the raw signal intensity values. The quantile normalization function normalizes the intensity values across arrays so that the distribution of intensities is similar for all arrays. Lastly, the probe-level model-fitting function fits a probe-level model to the normalized intensities, which allows for the estimation of expression values for each probe set. We have selected these methods based on the highly impacted research works (Bolstad et al., 2003; Irizarry et al., 2003).

Creating the SummarizedExperiment object

We will store the processed data in a `SummarizedExperiment` object, which is an S4 data object defined by the SummarizedExperiment package. The `SummarizedExperiment` object allows users to perform unified operations, e.g., add or remove samples, for both the metadata and assay. It thereby ensures that the metadata and observational data remain in sync, mitigating the risk of data mishandling that might occur when manually processing expression data and metadata.

6. Check that the required package is installed:

```
# load the SummarizedExperiment package
library(SummarizedExperiment)
```

We can ensure the required package is installed by loading it as shown in the above code snippet. We will need to use the functions in the SummarizedExperiment package to create and access the data stored in the `SummarizedExperiment` object.

7. Create the `SummarizedExperiment` object:

```
# create the SummarizedExperiment object
affyDataset <- SummarizedExperiment::SummarizedExperiment(assays = affyExprs, colData =
affySampleInfo)

# display the SummarizedExperiment object
print(affyDataset)

# console output
class: SummarizedExperiment
dim: 54675 161
metadata(0):
assays(1): ''
```

```
rownames(54675): 1007_s_at 1053_at … AFFX-r2-P1-cre-3_at
AFFX-r2-P1-cre-5_at
rowData names(0):
colnames(161): GSM119615 GSM119616 … GSM238955 GSM238963
colData names(71): X title … sample.amount.ch1 sex.ch1
```

Here, we use the `SummarizedExperiment()` function to create a SummarizedExperiment object. The function requires two parameters: (1) `assays`, one or more matrices containing the assay data; and (2) `colData`, a data frame containing the sample information. These data can be accessed using the two functions `assay()` and `colData()` from the SummarizedExperiment package, as shown in the next step.

8. Access the expression data and sample information stored in `SummarizedExperiment` object:

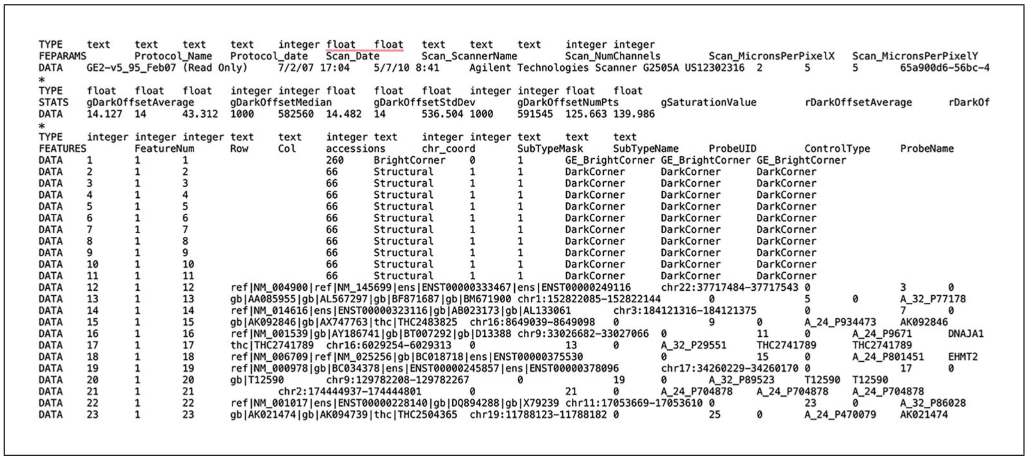
```
# access expression data
affyExprs <- SummarizedExperiment::assay(affyDataset)
# display affyExprs
print(head(affyExprs, c(5, 6)))
# console output

             GSM119615  GSM119616  GSM119617  GSM119618  GSM119619  GSM119620
1007_s_at    3.043234   3.055157   3.144277   3.150378   3.084336   2.989966
1053_at      1.698974   1.645050   1.618537   1.589216   1.676278   1.581733
117_at       1.795751   1.770719   1.805597   1.995794   1.688068   1.961556
121_at       2.553174   2.668456   2.801450   2.784216   2.588638   2.692849
1255_g_at    1.626691   1.940055   1.663162   1.483875   2.354096   1.671216


# Access to sample information
affySampleInfo <- SummarizedExperiment::colData(affyDataset)
# display affySampleInfo
head(affySampleInfo[, c("title", "characteristics_ch1.4", "characteristics_ch1.8")])
# console output

title                        characteristics_ch1.4        characteristics_ch1.8
GSM119615 EC control 1       Organ Region: Entorh..       Disease State: normal
GSM119616 EC control 2       Organ Region: Entorh..       Disease State: normal
GSM119617 EC control 3       Organ Region: Entorh..       Disease State: normal
GSM119618 EC control 4       Organ Region: Entorh..       Disease State: normal
GSM119619 EC control 5       Organ Region: Entorh..       Disease State: normal
GSM119620 EC control 6       Organ Region: Entorh..       Disease State: normal
```

In the above snippet, one can use the functions `assay()` and `colData()` to extract the expression data matrix and sample information contained within the `affyDataset` object. We then assign these extracted data sets to variables `affyExprs` and `affySampleInfo`, respectively. Afterward, we use the R built-in function `head()` to display selected rows and columns from these data matrices in the R console. By following this example, users can observe the example of the processed data as it appears in the console output.

PROCESSING AGILENT MICROARRAYS

BASIC PROTOCOL 2

This Basic Protocol guides users through the process of converting raw Agilent TXT files into a `SummarizedExperiment` object. Similar to Basic Protocol 1, there are three main steps in this protocol: (1) preparing the TXT files and sample information file, (2) processing the TXT files to obtain the expression data, and (3) creating the `SummarizedExperiment` object.



```
TYPE    text    text    text    text    integer float   float   text    text    text    integer integer
FEPARAMS        Protocol_Name   Protocol_date   Scan_Date       Scan_ScannerName        Scan_NumChannels        Scan_MicronsPerPixelX   Scan_MicronsPerPixelY
DATA    GE2-v5_95_Feb07 (Read Only)     7/2/07 17:04    5/7/10 8:41     Agilent Technologies Scanner G2505A US12302316  2       5       5       65a900d6-56bc-4
*
TYPE    float   float   float   integer integer float   float   float   integer integer float   float
STATS   gDarkOffsetAverage      gDarkOffsetMedian       gDarkOffsetStdDev       gDarkOffsetNumPts       gSaturationValue        rDarkOffsetAverage      rDarkOf
DATA    14.127  14      43.312  1000    582560  14.482  14      536.504 1000    591545  125.663 139.986
*
TYPE    integer integer integer text    text    integer text    integer integer text    text    text
FEATURES        FeatureNum      Row     Col     accessions      chr_coord       SubTypeMask     SubTypeName     ProbeUID        ControlType     ProbeName
DATA    1       1       1                       260     BrightCorner    0       1       GE_BrightCorner GE_BrightCorner GE_BrightCorner
DATA    2       1       2                       66      Structural      1       1       DarkCorner      DarkCorner      DarkCorner
DATA    3       1       3                       66      Structural      1       1       DarkCorner      DarkCorner      DarkCorner
DATA    4       1       4                       66      Structural      1       1       DarkCorner      DarkCorner      DarkCorner
DATA    5       1       5                       66      Structural      1       1       DarkCorner      DarkCorner      DarkCorner
DATA    6       1       6                       66      Structural      1       1       DarkCorner      DarkCorner      DarkCorner
DATA    7       1       7                       66      Structural      1       1       DarkCorner      DarkCorner      DarkCorner
DATA    8       1       8                       66      Structural      1       1       DarkCorner      DarkCorner      DarkCorner
DATA    9       1       9                       66      Structural      1       1       DarkCorner      DarkCorner      DarkCorner
DATA    10      1       10                      66      Structural      1       1       DarkCorner      DarkCorner      DarkCorner
DATA    11      1       11                      66      Structural      1       1       DarkCorner      DarkCorner      DarkCorner
DATA    12      1       12      ref|NM_004900|ref|NM_145699|ens|ENST00000333467|ens|ENST00000249116     chr22:37717484-37717543 0               3       0
DATA    13      1       13      gb|AA085955|gb|AL567297|gb|BF871687|gb|BM671900 chr1:152822085-152822144        0               5       0       A_32_P77178
DATA    14      1       14      ref|NM_014616|ens|ENST00000323116|gb|AB023173|gb|AL133061       chr3:184121316-184121375        0               7       0
DATA    15      1       15      gb|AK092846|gb|AX747763|thc|THC2483825  chr16:8649039-8649098   0               9       0       A_24_P934473    AK092846
DATA    16      1       16      ref|NM_001539|gb|AY186741|gb|BT007292|gb|D13388 chr9:33026682-33027066  0               11      0       A_24_P9671      DNAJA1
DATA    17      1       17      thc|THC2741789  chr16:6029254-6029313   0               13      0       A_32_P29551     THC2741789      THC2741789
DATA    18      1       18      ref|NM_006709|ref|NM_025256|gb|BC018718|ens|ENST00000375530             0               15      0       A_24_P801451    EHMT2
DATA    19      1       19      ref|NM_000978|gb|BC034378|ens|ENST00000245857|ens|ENST00000378096       chr17:34260229-34260170 0               17      0
DATA    20      1       20      gb|T12590       chr9:129782208-129782267        0               19      0       A_32_P89523     T12590  T12590
DATA    21      1       21              chr2:174444937-174444801        0               21      0       A_24_P704878    A_24_P704878    A_24_P704878
DATA    22      1       22      ref|NM_001017|ens|ENST00000228140|gb|DQ894288|gb|X79239 chr11:17053669-17053610 0               23      0       A_32_P86028
DATA    23      1       23      gb|AK021474|gb|AK094739|thc|THC2504365  chr19:11788123-11788182 0               25      0       A_24_P470079    AK021474
```

Figure 4 An example of the content of a TXT file from Agilent platform.

Necessary Resources

Hardware

An internet-connected computer or laptop with at least 8 GB of RAM and 20 GB of free hard drive

Software

R runtime environment (version 4.0.0 or later)
RCPA package from the CRAN repository

Files

A list of TXT.gz files

The TXT files here are the raw data files generated by Agilent microarray scanners as Agilent does not use any file extension for its raw data files. These TXT files can be read by any text editor. Figure 4 shows an example TXT file for Agilent. In this protocol, the example TXT files used for analysis will be downloaded from the GEO dataset with the accession number GSE61196.

A spreadsheet containing sample information in CSV or TSV format

In this spreadsheet, each row represents a sample, and each column represents its attribute, e.g., sample ID, disease status, tissue, etc. In this protocol, we will create the example spreadsheet by extracting the sample information from the GEO dataset GSE61196.

Preparing the TXT files and sample information file

Similar to Basic Protocol 1, users need to organize all TXT files in a single folder. In the following example, we will download the TXT files from the GEO dataset GSE61196 using the `downloadGEO()` function implemented in the RCPA package. If users already have the TXT files, they can skip this step and go directly to the next step.

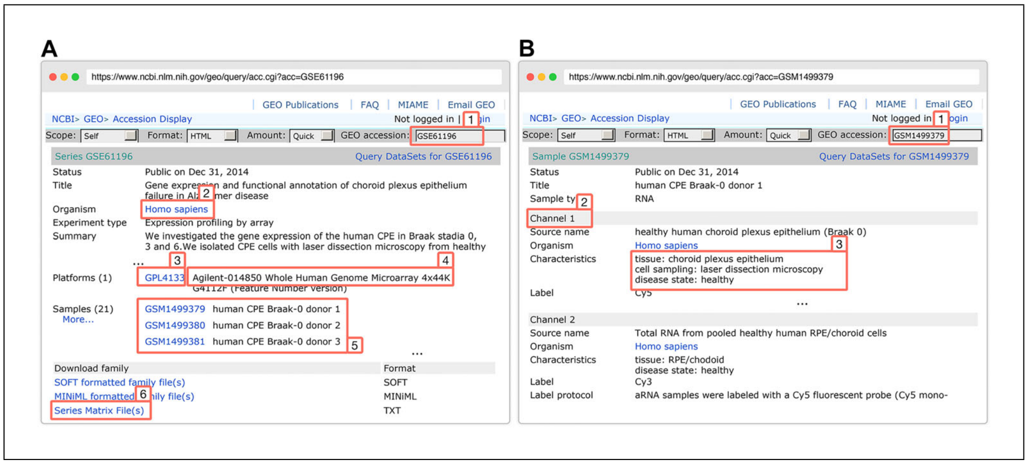
1. Create a local directory to save the downloaded data:

```
userPath <- tempdir() # or user-defined directory path
downloadPath <- file.path(userPath, "GSE61196")
if(!dir.exists(downloadPath)) dir.create(downloadPath)
```

The above code snippet is similar to that of Basic Protocol 1. Again, it is recommended that users specify their own directory, instead of using the temporary folder created by `tempdir()`.

2. Browse and search for the dataset on NCBI GEO.

To obtain the parameters necessary for the function `downloadGEO()`, we need to search for the dataset on https://www.ncbi.nlm.nih.gov/geo/, as shown in Figure 5. In addition to



https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE61196

| GEO Publications | FAQ | MIAME | Email GEO |

NCBI> GEO> Accession Display   Not logged in | gin

Scope: Self Format: HTML Amount: Quick GEO accession: GSE61196

Series GSE61196   Query DataSets for GSE61196

Status	Public on Dec 31, 2014
Title	Gene expres n and functional annotation of choroid plexus epithelium failure in Al mer disease
Organism	Homo sapiens
Experiment type	Expression profiling by array
Summary	We investigated the gene expression of the human CPE in Braak stadia 0, 3 and 6.We isolated CPE cells with laser dissection microscopy from healthy ...
Platforms (1)	GPL4133 Agilent-014850 Whole Human Genome Microarray 4x44K G4112F (Feature Number version)
Samples (21) More...	GSM1499379 human CPE Braak-0 donor 1 GSM1499380 human CPE Braak-0 donor 2 GSM1499381 human CPE Braak-0 donor 3 ...

Download family	Format
SOFT formatted family file(s)	SOFT
MINiML formatted ily file(s)	MINiML
Series Matrix File(s)	TXT

https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM1499379

| GEO Publications | FAQ | MIAME | Email GEO |

NCBI> GEO> Accession Display   Not logged in ogin

Scope: Self Format: HTML Amount: Quick GEO accession: GSM1499379

Sample GSM1499379   Query DataSets for GSM1499379

Status	Public on Dec 31, 2014
Title	human CPE Braak-0 donor 1
Sample ty	RNA
Channel 1	
Source name	healthy human choroid plexus epithelium (Braak 0)
Organism	Homo sapiens
Characteristics	tissue: choroid plexus epithelium cell sampling: laser dissection microscopy disease state: healthy
Label	Cy5 ...
Channel 2	
Source name	Total RNA from pooled healthy human RPE/choroid cells
Organism	Homo sapiens
Characteristics	tissue: RPE/chodoid disease state: healthy
Label	Cy3
Label protocol	aRNA samples were labeled with a Cy5 fluorescent probe (Cy5 mono-

Figure 5 Querying results of the Agilent dataset GSE61196. (**A**) Metadata of the dataset GSE61196. The following information can be found in this window: (**A1**) Accession ID, (**A2**) Organism of study, (**A3**) Platform, (**A4**) Technology, (**A5**) List of samples, and (**A6**) Preprocessed data. (**B**) Query results of the sample GSM1499379 in the dataset. Users can find the following information in this window: (**B1**) Sample ID, (**B2**) Channel information (one or two colors), and (**B3**) Sample characteristics.

platform and protocol, users also need to look for the channel information, as it is required as one of the inputs of `downloadGEO()` *for Agilent datasets. This information can be found in the query results of a sample in the dataset, which can be obtained by clicking on one of the sample IDs. The information is highlighted as box 2 in Figure 5b.*

3. Download the Agilent data using the `downloadGEO()` function:

```
# download the data
downloadedFiles <- RCPA::downloadGEO(GEOID = "GSE61196", platform = "GPL4133", protocol
 = "agilent", destDir = downloadPath)

# display the list of downloaded files
head(downloadedFiles)

# console output
[1] "metadata.csv" "GSM1499379.TXT.gz" "GSM1499380.TXT.gz"
[4] "GSM1499381.TXT.gz" "GSM1499382.TXT.gz" "GSM1499383.TXT.gz"
```

Similar to Basic Protocol 1, users can also apply the same function `downloadGEO()` *to download Agilent datasets from NCBI GEO to a local directory specified by the* `destDir` *parameter.*

Reading sample information and processing Agilent TXT files

4. Read the sample information from the metadata file:

```
# read the metadata file
agilSampleInfo <- read.csv(file.path(downloadPath, "metadata.csv"))

# Display the metadata
print(agilSampleInfo[9:14, c("geo_accession", "characteristics_ch2.1", "tissue.ch2")])

# console output
 geo_accession    characteristics_ch2.1      tissue.ch2
 GSM1499387       disease state: healthy     RPE/chodoid
 GSM1499388       disease state: healthy     RPE/chodoid
 GSM1499389       disease state: healthy     RPE/chodoid
 GSM1499390       disease state: healthy     RPE/chodoid
 GSM1499391       disease state: healthy     RPE/chodoid
 GSM1499392       disease state: healthy     RPE/chodoid
```

We use the `read.csv()` *function to read the* `metadata.csv` *file generated in the previous step in the* `downloadPath` *directory. In this code snippet, instead of using the* *`head()` function to display the data, users have the option to specify specific rows and columns of interest by employing the [..., ...] syntax. Specifically, within this code snippet, we print the expression data for rows 9 to 14 and three columns with predefined names.*

5. Process the TXT files and obtain the expression matrix:

```
# read the TXT files
agilExprs <- RCPA::processAgilent(dir = downloadPath, samples =
agilSampleInfo$geo_accession, greenOnly = FALSE)

# display the expression matrix
print(agilExprs[9:14, 1:6])

# console output
                GSM1499379   GSM1499380   GSM1499381   GSM1499382   GSM1499383   GSM1499384

DarkCorner      3.765860     4.191198     4.543230     4.663244     5.094978     3.860652

DarkCorner      3.962802     4.467307     4.200014     4.417445     4.574157     4.362014

DarkCorner      4.258405     4.216067     3.825804     3.993204     4.787838     4.295524

A_24_P66027     7.853252     8.038021     7.858425     7.573315     7.814866     7.882954

A_32_P77178     4.835717     4.678353     5.217154     5.307724     5.299094     4.591740

A_23_P212522    11.136671    11.407050    11.531872    11.202280    11.476940    10.980672
```

Here, we use the `processAgilent()` function to process the TXT files and obtain the expression matrix. This function requires three parameters. (1) `dir`, a character parameter specifying the path to the directory containing the TXT files. (2) `samples`, a character vector specifying the list of sample IDs to be processed. The sample IDs are also the names of the TXT files without the ".TXT.gz" extension. For example, the sample ID of the TXT file `GSM1499379.TXT.gz` is `GSM1499379`. (3) `greenOnly`, a logical parameter (TRUE/FALSE) which indicates that should the green (Cy3) channel only be read or are both red and green required. In this code snippet, the sample parameter is set to the `geo_accession` column of the `agilSampleInfo` data frame, which contains the sample IDs that match the names of the TXT files. Next, we print the expression data for rows 9 to 14 and columns 1 to 6 to the console.

Creating the `SummarizedExperiment` object

6. Create the `SummarizedExperiment` object:

```
# create the SummarizedExperiment object
agilDataset <- SummarizedExperiment::SummarizedExperiment(assays = agilExprs, colData =
agilSampleInfo)

# display the SummarizedExperiment object
print(agilDataset)

# console output
class: SummarizedExperiment
dim: 45015 21
metadata(0):
assays(1): ''
rownames(45015): GE_BrightCorner DarkCorner … GE_BrightCorner
GE_BrightCorner
rowData names(0):
colnames(21): GSM1499379 GSM1499380 … GSM1499398 GSM1499399
colData names(49): X title … tissue.ch1 tissue.ch2
```

In the above code snippet, we use the function `SummarizedExperiment()` function to create the `SummarizedExperiment` object.

7. Access the expression data and sample information stored in `SummarizedExperiment` object:

```
# Access expression data
agilExprs <- SummarizedExperiment::assay(agilDataset)
```

```
# Display agilExprs
print(agilExprs[9:14, 1:6])

# Console output
                 GSM1499379   GSM1499380   GSM1499381   GSM1499382   GSM1499383   GSM1499384

DarkCorner       3.765860     4.191198     4.543230     4.663244     5.094978     3.860652

DarkCorner       3.962802     4.467307     4.200014     4.417445     4.574157     4.362014

DarkCorner       4.258405     4.216067     3.825804     3.993204     4.787838     4.295524

A_24_P66027      7.853252     8.038021     7.858425     7.573315     7.814866     7.882954

A_32_P77178      4.835717     4.678353     5.217154     5.307724     5.299094     4.591740

A_23_P212522     11.136671    11.407050    11.531872    11.202280    11.476940    10.980672

# Access sample information
agilSampleInfo <- SummarizedExperiment::colData(agilDataset)

# Display agilSampleInfo
print(agilSampleInfo[9:14, c("geo_accession", "characteristics_ch2.1", "tissue.ch2")])

# Console output
            geo_accession    characteristics_ch2.1     tissue.ch2

  GSM1499387GSM1499387       disease state: healthy    RPE/chodoid

  GSM1499388GSM1499388       disease state: healthy    RPE/chodoid

  GSM1499389GSM1499389       disease state: healthy    RPE/chodoid

  GSM1499390GSM1499390       disease state: healthy    RPE/chodoid

  GSM1499391GSM1499391       disease state: healthy    RPE/chodoid

  GSM1499392GSM1499392       disease state: healthy    RPE/chodoid
```

In this code snippet, we employ the functions `assay()` and `colData()` to access the expression data matrix and sample information data. Within this code snippet, we print the expression data for rows 9 to 14 and the three columns named "geo_accession", "characteristics_ch2.1", and "tissue.ch2". These columns store sample accession, disease state, and tissue, as shown in the console output.

SUPPORT PROTOCOL

PROCESSING RNA SEQUENCING (RNA-Seq) DATA

This Support Protocol guides users through the process of converting a raw count matrix file into a `SummarizedExperiment` object. There are two main steps in this protocol: (1) preparing the count matrix file and sample information file, and (2) creating the `SummarizedExperiment` object.

Necessary Resources

Hardware

An internet-connected computer or laptop with at least 8 GB of RAM and 20 GB of free hard drive space

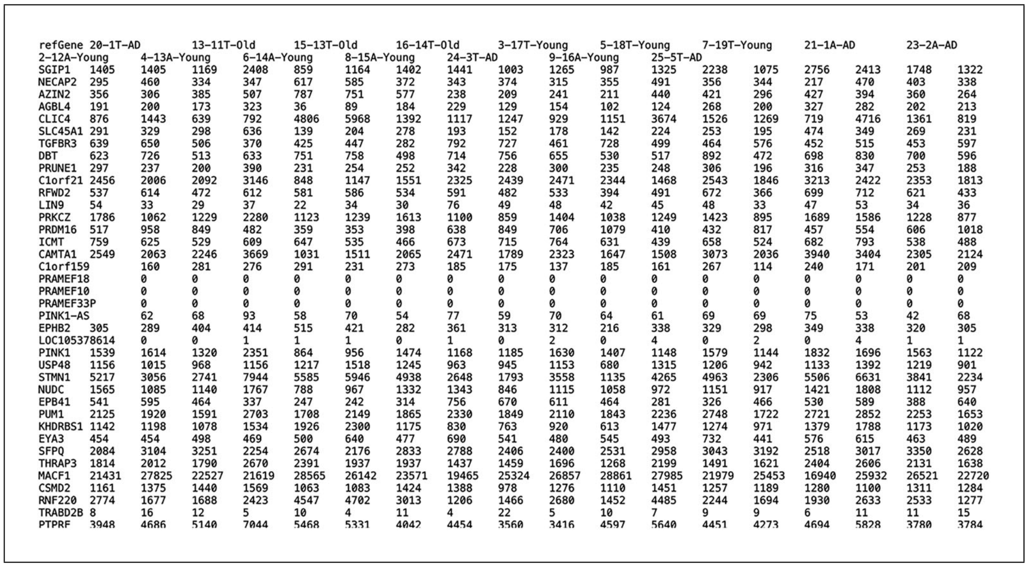
Software

R runtime environment (version 4.0.0 or later)
RCPA package from the CRAN repository

Files

A count matrix file

The count matrix file is a table that contains the raw read counts for each gene in each sample. The count matrix file can be saved in any format, e.g., TXT, CSV, TSV, etc., depending on what data processing pipeline users use to generate the count matrix file. We recommend that users follow the best practices for data processing and normalization specific to each RNA-Seq technology (Conesa et al., 2016; Robertson et al., 2010). Figure 6 shows an example count matrix. In this protocol, we will download the count matrix file from the GEO dataset with the accession number GSE153873.



```
refGene 20-1T-AD        13-11T-Old      15-13T-Old      16-14T-Old      3-17T-Young     5-18T-Young     7-19T-Young     21-1A-AD        23-2A-AD
2-12A-Young     4-13A-Young     6-14A-Young     8-15A-Young     24-3T-AD        9-16A-Young     25-5T-AD
SGIP1   1405    1405    1169    2408    859     1164    1402    1441    1003    1265    987     1325    2238    1075    2756    2413    1748    1322
NECAP2  295     460     334     347     617     585     372     343     374     315     355     491     356     344     217     470     403     338
AZIN2   356     306     385     507     787     751     577     238     209     241     211     440     421     296     427     394     360     264
AGBL4   191     200     173     323     36      89      184     229     129     154     102     124     268     200     327     282     202     213
CLIC4   876     1443    639     792     4806    5968    1392    1117    1247    929     1151    3674    1526    1269    719     4716    1361    819
SLC45A1 291     329     298     636     139     204     278     193     152     178     142     224     253     195     474     349     269     231
TGFBR3  639     650     506     370     425     447     282     792     727     461     728     499     464     576     452     515     453     597
DBT     623     726     513     633     751     758     498     714     756     655     530     517     892     472     698     830     700     596
PRUNE1  297     237     200     390     231     254     252     342     228     300     235     248     306     196     316     347     253     188
C1orf21 2456    2006    2092    3146    848     1147    1551    2325    2439    2471    2344    1468    2543    1846    3213    2422    2353    1813
RFWD2   537     614     472     612     581     586     534     591     482     533     394     491     672     366     699     712     621     433
LIN9    54      33      29      37      22      34      30      76      49      48      42      45      48      33      47      53      34      36
PRKCZ   1786    1062    1229    2280    1123    1239    1613    1100    859     1404    1038    1249    1423    895     1689    1586    1228    877
PRDM16  517     958     849     482     359     353     398     638     849     706     1079    410     432     817     457     554     606     1018
ICMT    759     625     529     609     647     535     466     673     715     764     631     439     658     524     682     793     538     488
CAMTA1  2549    2063    2246    3669    1031    1511    2065    2471    1789    2323    1647    1508    3073    2036    3940    3404    2305    2124
C1orf159        160     281     276     291     231     273     185     175     137     185     161     267     114     240     171     201     209
PRAMEF18        0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
PRAMEF10        0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
PRAMEF33P       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
PINK1-AS        62      68      93      58      70      54      77      59      70      64      61      69      69      75      53      42      68
EPHB2   305     289     404     414     515     421     282     361     313     312     216     338     329     298     349     338     320     305
LOC105378614    0       0       1       1       1       0       1       0       2       0       4       0       2       0       4       1       1
PINK1   1539    1614    1320    2351    864     956     1474    1168    1185    1630    1407    1148    1579    1144    1832    1696    1563    1122
USP48   1156    1015    968     1156    1217    1518    1245    963     945     1153    680     1315    1206    942     1133    1392    1219    901
STMN1   5217    3056    2741    7944    5585    5946    4938    2648    1793    3558    1135    4265    4963    2306    5506    6631    3841    2234
NUDC    1565    1085    1140    1767    788     967     1332    1343    846     1115    1058    972     1151    917     1421    1808    1112    957
EPB41   541     595     464     337     247     242     314     756     670     611     464     281     326     466     530     589     388     640
PUM1    2125    1920    1591    2703    1708    2149    1865    2330    1849    2110    1843    2236    2748    1722    2721    2852    2253    1653
KHDRBS1 1142    1198    1078    1534    1926    2300    1175    830     763     920     613     1477    1274    971     1379    1788    1173    1020
EYA3    454     454     498     469     500     640     477     690     541     480     545     493     732     441     576     615     463     489
SFPQ    2084    3104    3251    2254    2674    2176    2833    2788    2406    2400    2531    2958    3043    3192    2518    3017    3350    2628
THRAP3  1814    2012    1790    2670    2391    1937    1937    1437    1459    1696    1268    2199    1491    1621    2404    2606    2131    1638
MACF1   21431   27825   22527   21619   28565   26142   23571   19465   25324   26857   28861   27985   21979   25453   16940   25932   26521   22720
CSMD2   1161    1375    1440    1569    1063    1083    1424    1388    978     1276    1110    1451    1257    1189    1280    1100    1311    1284
RNF220  2774    1677    1688    2423    4547    4702    3013    1206    1466    2680    1452    4485    2244    1694    1930    2633    2533    1277
TRABD2B 8       16      12      5       10      4       11      4       22      5       10      7       9       9       6       11      11      15
PTPRF   3948    4686    5140    7044    5468    5331    4042    4454    3560    3416    4597    5640    4451    4273    4694    5828    3780    3784
```

Figure 6 An example of the content of a count matrix file. In this example, the count matrix file is saved in TSV format. The first column contains the gene IDs in form of gene symbols, and the first row contains the sample IDs. The remaining cells contain the raw read counts for each gene in each sample.

A spreadsheet containing sample information, CSV or TSV format

In this spreadsheet, each row represents a sample, and each column represents its attribute, e.g., sample ID, disease status, tissue, etc. In this protocol, we will create the example spreadsheet by extracting the sample information from the GEO dataset GSE153873.

Preparing the count matrix file and sample information file

Users can skip this step if they already have the count matrix file and sample information file. In this protocol, we will create the sample information file by extracting the sample information from the GEO dataset GSE153873 using the `getGEO()` function implemented in the GEOquery package. We will also download the count matrix file from the GEO dataset GSE153873 using the `getGEOSuppFiles()` function.

1. Browse and search for the RNA-Seq dataset on GEO:

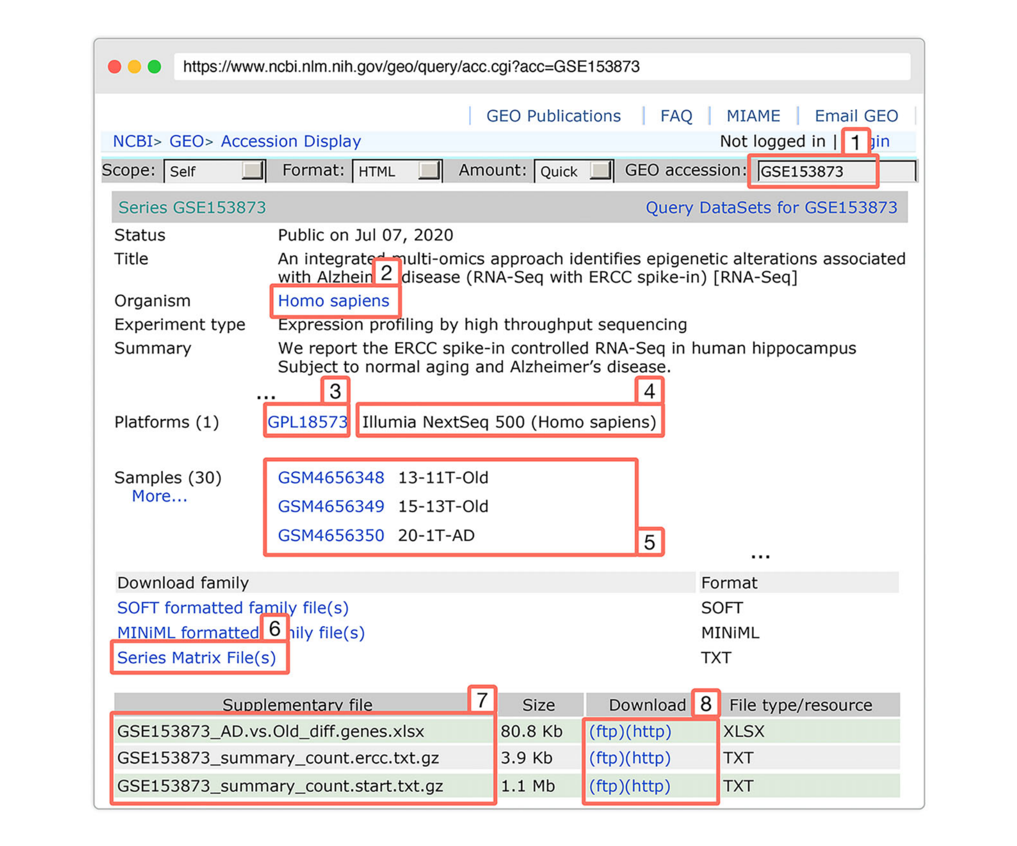
   We can search for the dataset on https://www.ncbi.nlm.nih.gov/geo/ by providing the accession number (GSE153873) in the search box, as shown in Figure 7. When the searching is done, the webpage displays the details of the dataset, including the published date, title, organism, experiment type, dataset summary, and supplementary files that contain the raw read counts. In this protocol, we will guide users on how to download these data using functions from the GEOquery package in R, as shown in the following steps.

2. Create a local directory to save the downloaded data:

```
userPath <- tempdir() # or user-defined directory path
downloadPath <- file.path(userPath, "GSE153873")
if(!dir.exists(downloadPath)) dir.create(downloadPath)
```

3. Get the sample information from the GEO dataset:

```
# Download the GEO object to get metadata
GEOObject <- GEOquery::getGEO(GEO = "GSE153873", GSEMatrix = T, getGPL = F, destdir =
downloadPath)

# Check the length of GEOObject
message("length: ", length(GEOObject))

# console output
length: 1
```



https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE153873

| GEO Publications | FAQ | MIAME | Email GEO |

NCBI> GEO> Accession Display — Not logged in | Login

Scope: Self Format: HTML Amount: Quick GEO accession: GSE153873

Series GSE153873 — Query DataSets for GSE153873

Status	Public on Jul 07, 2020
Title	An integrated multi-omics approach identifies epigenetic alterations associated with Alzheimer's disease (RNA-Seq with ERCC spike-in) [RNA-Seq]
Organism	Homo sapiens
Experiment type	Expression profiling by high throughput sequencing
Summary	We report the ERCC spike-in controlled RNA-Seq in human hippocampus Subject to normal aging and Alzheimer's disease.
...	
Platforms (1)	GPL18573 Illumia NextSeq 500 (Homo sapiens)
Samples (30) More...	GSM4656348 13-11T-Old GSM4656349 15-13T-Old GSM4656350 20-1T-AD ...

Download family	Format
SOFT formatted family file(s)	SOFT
MINiML formatted family file(s)	MINiML
Series Matrix File(s)	TXT

Supplementary file	Size	Download	File type/resource
GSE153873_AD.vs.Old_diff.genes.xlsx	80.8 Kb	(ftp)(http)	XLSX
GSE153873_summary_count.ercc.txt.gz	3.9 Kb	(ftp)(http)	TXT
GSE153873_summary_count.start.txt.gz	1.1 Mb	(ftp)(http)	TXT

Figure 7 Querying RNA-Seq data from NCBI GEO. The numbered red boxes show the information that might be required when downloading data using GEOquery package. The following information can be found from this query: (1) dataset GEO accession ID, (2) organism of study, (3) platform, (4) technology, (5) list of samples, (6) preprocessed assays, (7) supplementary files, and (8) buttons for manually downloading these supplementary files.

```
# Extract the dataset from the GEOObject
samplesData <- GEOObject[[1]]

# Export sample data
metadata <- Biobase::pData(samplesData)

# save the metadata for later use
write.csv(metadata, file.path(downloadPath, "metadata.csv"))

# preview the metadata
print(head(metadata[, c("title", "status", "characteristics_ch1.1")]))

# console output

              title          status                    characteristics_ch1.1
GSM4656348    13-11T-Old     Public on Jul 07 2020     disease state: old
GSM4656349    15-13T-Old     Public on Jul 07 2020     disease state: old
GSM4656350    20-1T-AD       Public on Jul 07 2020     disease state: Alzheimer's disease
GSM4656351    16-14T-Old     Public on Jul 07 2020     disease state: old
GSM4656352    3-17T-Young    Public on Jul 07 2020     disease state: young
GSM4656353    5-18T-Young    Public on Jul 07 2020     disease state: young
```

In this code snippet, we use the function `getGEO()` *to download the GEO object of the dataset GSE153873. We set the parameter* `GSEMatrix` *to* `TRUE` *to download the GSE Series Matrix files and the parameter* `getGPL` *to* `FALSE` *to avoid downloading the platform information. The* `destdir` *parameter specifies the directory to save the downloaded data. The function returns a list of* `ExpressionSet` *objects, which are assigned to the* `GEOObject` *variable. This list can contain one or more* `ExpressionSet` *objects,* *depending on how many series matrix files are available for the dataset. These series matrix files can come from different platforms or different processing methods. In this example, the list contains only one `ExpressionSet` object. Next, we extract the `ExpressionSet` object from the list and obtain the sample information from the `pData()` function. Finally, we save the sample information to a CSV file named `metadata.csv` in the `downloadPath` directory.*

4. Download the raw count data from the GEO dataset:

```
# Download the supplementary files
GEOquery::getGEOSuppFiles(GEO = "GSE153873", fetch_files = TRUE, baseDir = userPath)

# Check the downloaded data files
list.files(file.path(userPath, "GSE153873"))

# Console output
[1] "GSE153873_AD.vs.Old_diff.genes.xlsx" "GSE153873_summary_count.ercc.txt.gz"
[3] "GSE153873_summary_count.star.txt.gz"
```

In this code snippet, we use the function `getGEOSuppFiles()` to download the supplementary files of the dataset GSE153873. The function has the following parameters: (1) `GEO`, a character parameter that specifies the GEO accession number; (2) `baseDir`, a character parameter that specifies the directory for downloaded data; and (3) `fetch_files`, a logical parameter, with `TRUE` means telling the function to actually download the files and `FALSE` telling the function to just return the filenames that would have been downloaded. The function returns a data frame in which row names represent the full path to the downloaded files.

Note that the function `getGEOSuppFiles()` automatically creates a folder named "GSE153873" (`GEO`) in the folder defined by the `userPath` variable and saves all the downloaded data there. We can check the downloaded file in this folder using the function `list.files()`. As we can see in the console output, the raw count data is saved under the name "GSE153873_summary_count.star.txt.gz" (as in Figure 7).

Creating the `SummarizedExperiment` object

5. Examine the format of the count matrix file:

```
# read the first 10 lines of the count matrix file
countsFile <- file.path(userPath, "GSE153873", "GSE153873_summary_count.star.txt.gz")
lines <- readLines(countsFile, n = 10)

# display the first 50 characters of each line
print(substr(lines, start = 1, stop = 50))

# console output

[1]       "refGene\t20-1T-AD\t13-11T-Old\t15-13T-Old\t16-14T-Old\t"
[2]       "SGIP1\t1405\t1405\t1169\t2408\t859\t1164\t1402\t1441\t1003\t"
[3]       "NECAP2\t295\t460\t334\t347\t617\t585\t372\t343\t374\t315\t355"
[4]       "AZIN2\t356\t306\t385\t507\t787\t751\t577\t238\t209\t241\t211\t"
[5]       "AGBL4\t191\t200\t173\t323\t36\t89\t184\t229\t129\t154\t102\t12"
[6]       "CLIC4\t876\t1443\t639\t792\t4806\t5968\t1392\t1117\t1247\t92"
[7]       "SLC45A1\t291\t329\t298\t636\t139\t204\t278\t193\t152\t178\t14"
[8]       "TGFBR3\t639\t650\t506\t370\t425\t447\t282\t792\t727\t461\t728"
[9]       "DBT\t623\t726\t513\t633\t751\t758\t498\t714\t756\t655\t530\t51"
[10]      "PRUNE1\t297\t237\t200\t390\t231\t254\t252\t342\t228\t300\t235"
```

Before we read the count matrix file into R, we need to examine the format of the file to determine which function to use to read the file. Our count matrix file is a gzip compressed file with the extension `.gz`. We can use the `readLines()` function to read the first 10 lines of the count matrix file without extracting the entire file. This function can read the file line by line, and the n parameter specifies the number of lines to read. Next, we extract the first 50 characters of each line using the `substr()` function and print them to the *console. As we can see in the console output, this count matrix file is a tab-delimited text file, in which the first column contains the gene IDs in the form of gene symbols, and the first row contains the sample IDs. With this format, we can use the* `read.table()` *function to read the count matrix file.*

6. Read the sample information and count matrix file:

```
# read the previously saved metadata
RNASeqSampleInfo <- read.csv(file.path(downloadPath, "metadata.csv"))

# read the count matrix file
countsData <- read.table(countsFile, header = TRUE, sep = "\t", fill = 0, row.names = 1,
check.names = FALSE)

# preview the metadata and count matrix
print(head(RNASeqSampleInfo[, c("title", "status", "characteristics_ch1.1")], 3))
print(head(countsData, c(3,6)))

# console output
# metadata
              title        status                 characteristics_ch1.1
GSM4656348    13-11T-Old   Public on Jul 07 2020  disease state: old
GSM4656349    15-13T-Old   Public on Jul 07 2020  disease state: old
GSM4656350    20-1T-AD     Public on Jul 07 2020  disease state:
                                                  Alzheimer's disease

# count matrix
            20-1T-AD    13-11T-Old   15-13T-Old   16-14T-Old   3-17T-Young   5-18T-Young
 SGIP1      1405        1405         1169         2408         859           1164
 NECAP2     295         460          334          347          617           585
 AZIN2      356         306          385          507          787           751
```

Here, we use the function `read.csv()` *to read the file* `metadata.csv` *that was generated in the previous step in the directory specified by the variable* `downloadPath`*. We also use the function* `read.table()` *to read the file that was downloaded in the previous step:* `GSE153873_summary_count.star.txt`*. The function* `read.table()` *is able to read the compressed "* `.txt.gz` *" file directly without decompressing the file first. We set the* `header` *parameter to* `TRUE` *to keep the sample IDs as the column labels. We set the* `sep` *parameter to "* `\t` *" to indicate that the data are separated by tab. We set the* `fill` *parameter to* `0` *to fill the missing values with 0. We set the* `row.names` *parameter to* `1` *to use the first column as the row names. We set the* `check.names` *parameter to* `FALSE` *to avoid checking the validity of the column names. The output of this function is a data frame, which is assigned to the* `countsData` *variable.*

Next, we print the first 3 rows and 6 columns of the sample information and count matrix to the console. As shown in the console output, the row names of `RNASeqSampleInfo` *are in the format of* `GSM*`*, which are defined by the GEO database. However, the column names of the count matrix* `countsData` *are in the format of* `XXX-XXX-XXX`*, which is an arbitrary format that was generated by the authors of the dataset. This sample ID can be found in the metadata file that was assigned to* `RNASeqSampleInfo` *under the column* `title`*. To keep the sample IDs consistent between the metadata and count matrix, we need to rename the column names of the count matrix to match the row names of the metadata.*

7. Rename the column names of the count matrix:

```
# Get the sample titles
sampleTitles <- RNASeqSampleInfo$title

# Rearrange the column of the count matrix
countsData <- countsData[, sampleTitles]

# Rename the column names
colnames(countsData) <- rownames(RNASeqSampleInfo)
```

```
# Preview the count matrix
print(head(countsData, c(5,6)))

# console output
            GSM4656348    GSM4656349    GSM4656350    GSM4656351    GSM4656352    GSM4656353
SGIP1       1405          1169          1405          2408          859           1164
NECAP2      460           334           295           347           617           585
AZIN2       306           385           356           507           787           751
AGBL4       200           173           191           323           36            89
CLIC4       1443          639           876 7         92            4806          5968
```

We first rearrange the columns of the count matrix `countsData` to match the order of the sample in the metadata `RNASeqSampleInfo`. Next, we rename the column names of the count matrix `countsData` to match the row names of the metadata `RNASeqSampleInfo`. As we can see in the console output, the column names of the count matrix `countsData` are now in the format of `GSM*`, which are the same as the row names of the metadata `RNASeqSampleInfo`.

8. Create the `SummarizedExperiment` object:

```
RNASeqDataset <- SummarizedExperiment::SummarizedExperiment(assays =
 as.matrix(countsData), colData = metadata)

# Display the SummarizedExperiment object
print(RNASeqDataset)

# Console output
class: SummarizedExperiment
dim: 27135 30
metadata(0):
assays(1): ”
rownames(27135): SGIP1 NECAP2 … KIR2DS1 KIR2DL5B
rowData names(0):
colnames(30): GSM4656348 GSM4656349 … GSM4656376 GSM4656377
colData names(38): title geo_accession … disease state:ch1 tissue:ch1
```

In the above code, we create the `SummarizedExperiment` to store the expression matrix and the sample information in the `assays` and the `colData` attributes, respectively.

9. Access the expression data and sample information stored in `SummarizedExperiment` object:

```
# Access the expression data:
RNASeqExprs <- SummarizedExperiment::assay(RNASeqDataset)

# Display RNASeqExprs
head(RNASeqExprs, c(5, 6))

# Console output:
             GSM4656348    GSM4656349    GSM4656350    GSM4656351    GSM4656352    GSM4656353
  SGIP1      1405          1169          1405          2408          859           1164
  NECAP2     460           334           295           347           617           585
  AZIN2      306           385           356           507           787           751
  AGBL4      200           173           191           323           36            89
  CLIC4      1443          639           876           792           4806          5968

# Access to the sample information data
RNASeqSampleInfo <- SummarizedExperiment::colData(RNASeqDataset)

# Display RNASeqSampleInfo
print(RNASeqSampleInfo[1:5, c("characteristics_ch1", "characteristics_ch1.1", "disease
state:ch1")])

# Console output:
```

```
DataFrame with 5 rows and 3 columns

             characteristics_ch1     characteristics_ch1.1     disease state:ch1

GSM4656348   tissue: Lateral temp..  disease state: old        old

GSM4656349   tissue: Lateral temp..  disease state: old        old

GSM4656350   tissue: Lateral temp..  disease state: Alzhe..    Alzheimer's disease

GSM4656351   tissue: Lateral temp..  disease state: old        old

GSM4656352   tissue: Lateral temp..  disease state: young      young
```

Once the `SummarizedExperiment` object is created, users can follow the same procedure as discussed in Basic Protocols 1 and 2 to access the assay and sample data stored in the created object, as shown in the above snippet.

DIFFERENTIAL ANALYSIS OF MICROARRAY DATA (AFFYMETRIX AND AGILENT)

BASIC PROTOCOL 3

Differential expression analysis is a fundamental analysis that aims to identify genes that are differentially expressed between distinct conditions or phenotypes. The analysis of Affymetrix and Agilent only differs in data processing and normalization. After the processing step, their downstream analysis procedures are the same. There are many popular excellent techniques that can be used for differential analysis (Kerr et al., 2000; Love et al., 2014; Student, 1908). In this article, we choose to describe three basic methods that have been widely used in the field, namely limma (Ritchie et al., 2015), DESeq2 (Love et al., 2014), and edgeR (Robinson et al., 2010). Note that limma is often applied for continuous data (microarray expression, or normalized RNA-Seq expression, e.g., RPKM, TPM, etc.) whereas DESeq2 and edgeR are often used for read counts (positive integers).

In this protocol, we introduce the function `runDEAnalysis()` that applies limma (Ritchie et al., 2015) for differential expression analysis, specifically designed for microarray data. The procedure involves several key steps, including the definition and representation of experimental design using design and contrast matrices. This is followed by the conversion of probe IDs into Entrez gene IDs, which will be used for subsequent analysis. Additionally, we offer two visualization functions, `plotMA()` and `plotVolcanoDE()`, that are designed to enhance the visualization of the differential analysis results. Visualization is a common practice to examine the results of differential analysis before users proceed with pathway analysis and data integration.

Here, we provide step-and-step guidelines on differential analysis of the two microarray datasets (GSE5281 and GSE61196) that have been used thus far. Note that after data processing, as described in Basic Protocols 1 and 2, we have the `SummarizedExperiment` objects that include expression data and sample metadata.

Necessary Resources

Hardware

An internet-connected computer or laptop with at least 8 GB of RAM and 20 GB of free hard drive space

Software

R runtime environment (version 4.0.0 or later)

RCPA package from the CRAN repository (see Internet Resources)

Besides the functions in RCPA, we will need to use the functions in the SummarizedExperiment to access the data stored in the SummarizedExperiment object. Similarly, some functions from the ggplot2 package will be used to add the title or modify the figures generated by the plot functions in RCPA. We can ensure the required packages are installed by loading them as shown in the following code snippet:

```
library(RCPA)

library(SummarizedExperiment)

library(ggplot2)
```

Files

A SummarizedExperiment object

A SummarizedExperiment object has at least two attributes: (1) assays, storing the expression data matrix, in which rows are genes/transcripts and columns are samples; and (2) colData, storing a data frame containing the sample information. Users can refer to Basic Protocols 1 and 2 for more information about the SummarizedExperiment object.

A gene mapping data frame

A gene mapping data frame has two columns: (2) FROM, containing the genes/transcript IDs currently used in the assay data; and (2) TO, containing the corresponding Entrez IDs. Users need to manually prepare this data frame and can save it as .rda, .rds, .txt, or .csv files, or other formats. Consequently, they will use the appropriate R functions to load the file. We will guide users on how to prepare this data frame in our practical example below.

Sample files

Users can use function RCPA::loadData() to load the pre-saved Microarray datasets obtained from Basic Protocols 1 and 2 from our GitHub repository (https://github.com/tinnlab/RCPA/tree/main/.data). The first step in this protocol will guide the user in loading the pre-saved data using the function.

Affymetrix: GSE5281

1. If users skipped Basic Protocol 1, they could use `RCPA::loadData()` to load the data:

```
# Load the Affymetrix data processed in Basic Protocol 1

affyDataset <- RCPA::loadData("affyDataset")
```

2. Create a design matrix for differential analysis:

To perform differential analysis, users need to provide the experimental design of the data. The experimental design can be represented in the form of a “design matrix” in which rows represent samples and columns encompass a range of experimental variables. Among many variables, we are particularly interested in knowing sample phenotype/condition, such as disease and control. Here is the code snippet for generating a design matrix tailored for the Affymetrix dataset GSE5281.

```
# Read metadata from the SummarizedExperiment object named affyDataset

affySampleInfo <- SummarizedExperiment::colData(affyDataset)

# Add a column specifying the condition of each sample (normal or Alzheimer’s)

affySampleInfo$condition <- ifelse(grepl("normal",

affySampleInfo$characteristics_ch1.8), "normal", "alzheimer")

# Factorize the new column

affySampleInfo$condition <- factor(affySampleInfo$condition)

# Add a new column to specify the region of the sample tissue,

# use make.names() to remove special characters and

# use tolower() to make all characters lowercase

affySampleInfo$region <- make.names(affySampleInfo$characteristics_ch1.4)

affySampleInfo$region <- tolower(affySampleInfo$region)

# Factorize the newly added column

affySampleInfo$region <- factor(affySampleInfo$region)

# Update the affyDataset object

SummarizedExperiment::colData(affyDataset) <- affySampleInfo

# Create a design matrix
```

```
affyDesign <- model.matrix(~0 + condition + region + condition:region, data =
affySampleInfo)

# Remove special characters in column names
colnames(affyDesign) <- make.names(colnames(affyDesign))
```

The samples of the GSE5281 dataset fall into two conditions, "normal" and "Alzheimer's," which are specified in the `characteristics_ch1.8` *column. Each sample is also associated with a specific brain region, such as the entorhinal cortex, hippocampus, primary visual cortex, and so on, denoted in the* `characteristics_ch1.4` *column. Consequently, both attributes serve as experimental variables within the design matrix.*

The initial step in the code snippet involves the addition of two new columns that represent the sample's condition and the associated brain region to the sample information that is stored in the returned `SummarizedExperiment` *object described in Basic Protocol 1. These new columns are essentially cleaner versions of the original* `characteristics_ch1.8` *and* `characteristics_ch1.4` *columns. The original columns are often manually curated and may contain special characters or duplicated data, which could potentially lead to errors in the analysis. Therefore, it is crucial to perform data cleaning before proceeding with any further steps. Users can verify the updated sample information by executing the following command line:*

```
# Check update
affyDataset

# Console output:
class: SummarizedExperiment
dim: 54675 161
metadata(0):
assays(1): ”
rownames(54675): 1007_s_at 1053_at … AFFX-TrpnX-5_at AFFX-TrpnX-M_at
rowData names(0):
colnames(161): GSM119615 GSM119616 … GSM238955 GSM238963
colData names(72): title geo_accession … condition region
```

The above code shows that the new columns `condition` *and* `region` *were added to the* `affyDataset` *object, as indicated in the final row of the console (i.e., under "colData names"). To generate the design matrix, we introduce the function* `model.matrix()`*, sourced from the built-in R package called stats. Utilizing this function entails providing two essential pieces of information: a formula resembling a regression model, such as* `~var1 + var2`*, and the dataset containing the variables referenced in this formula. Here, in the above snippet, the formula "*`~0 + condition + region + condition:region`*" defines the design matrix to include the main effects of condition and region, as well as the interaction between condition and region. The term* `~0` *indicates that the intercept should not be included in the design matrix since the intercept is not of interest in the analysis. By following the snippet, users can observe the example of the design matrix as it appears in the following console output:*

```
# print the design matrix
print(affyDesign[1:5, 1:3])

# Console output
                    conditionalzheimer  conditionnormal   regionorgan.region..hippocampus.
  GSM119615         0                   1                 0
  GSM119616         0                   1                 0
  GSM119617         0                   1                 0
  GSM119618         0                   1                 0
  GSM119619         0                   1                 0
```

As evident from the matrix, the design matrix takes the form of a binary matrix. In each column, a value of 1 denotes that the patient corresponds to the condition represented by that column.

3. Create a contrast matrix for differential analysis:

In addition to the design matrix, users also need to provide the contrast matrix when performing differential analysis. A "contrast matrix" has rows associated with columns in the corresponding design matrix and columns associated with contrasts. Contrasts play a crucial role in determining which conditions are being compared to one another, especially in situations where there are more than two conditions in the experimental design. In R programming, we code for contrast matrix using the function `makeContrasts()` *from the limma package. The usage of this function will be discussed in the example below.*

```
# Create a contrast matrix
affyContrast <- limma::makeContrasts(conditionalzheimer-conditionnormal,
levels = affyDesign)

# print contrast
head(affyContrast)

# Console output


                                             Contrasts
Levels                                           conditionalzheimer - conditionnormal
                                                                                    1
conditionalzheimer
                                                                                   -1
conditionnormal
                                                                                    0
regionorgan.region..hippocampus.
                                                                                    0
regionorgan.region..medial.temporal.gyrus.
                                                                                    0
regionorgan.region..posterior.cingulate.
                                                                                    0
regionorgan.region..posterior.singulate.
```

The function `makeContrasts()` *requires the following parameters: (1) a mathematical expression, such as* `x-y`*, specifying the contrast between a set of variables; and (2)* `levels`*, which refers to a design matrix with column names corresponding to the variables mentioned in the first parameter. In our example, we are comparing gene expression between two patient groups: "normal" and "Alzheimer's". In the design matrix* `affyDesign`*, this grouping information is captured by two columns:* `conditionnormal` *and* `conditionalzheimer`*. Consequently, our first parameter for the function* `makeContrasts()` *is* `conditionalzheimer-conditionnormal`*, while the second parameter is* `affyDesign`*.*

The contrast matrix is displayed in the console output in the above example. The contrast of (−1, 1) signifies subtracting the gene expression of normal samples from that of Alzheimer's samples. A value of 0 indicates that we are not considering the other columns in the design matrix for this contrast. For more information on how to define the design matrix and contrast matrix for different scenarios and interests, users can refer to the manual of the limma package (Law et al., 2020).

4. Perform differential analysis using the function `runDEAnalysis()`:

```
# Run differential expression analysis
affyDEExperiment <- RCPA::runDEAnalysis(affyDataset, method = "limma", design =
affyDesign, contrast = affyContrast, annotation = "GPL570")
```

```
# Display affyDEExperiment
affyDEExperiment

# Console output
class: SummarizedExperiment
dim: 21367 161
metadata(4): DEAnalysis.method DEAnalysis.design DEAnalysis.contrast DEAnalysis.mapping
assays(1): counts
rownames(21367): 55101 92840 … 9695 83887
rowData names(9): PROBEID ID … sampleSize pFDR
colnames(161): GSM119615 GSM119616 … GSM238955 GSM238963
colData names(72): title geo_accession … condition region
```

The function `runDEAnalysis()` requires the following parameters: (1) a `SummarizedExperiment` object storing the assay data and sample metadata; (2) `method`, a character parameter specifying the differential analysis method ("limma" in this example); (3) `design`, a design matrix; (4) `contrast`, a contrast matrix; and (5) `annotation`, a character parameter used to define the assay platform, or a data frame that maps probe IDs to Entrez IDs. This data frame should consist of at least two columns: `FROM` (representing the probe ID in the original dataset), and `TO` (representing the corresponding Entrez ID). This fifth parameter is crucial for converting probe IDs into Entrez IDs, which will serve as the consistent gene identifiers for subsequent analyses. In RCPA, we support gene ID mapping for all platforms that have corresponding annotation packages on Bioconductor. Users can verify the compatibility of their data platform by running the function `getSupportedPlatforms()`. If their platform is listed, users can conveniently input the platform ID into the `annotation` parameter. The platform of the dataset GSE5281 is GPL570, which has an annotation package on Bioconductor named "hgu133plus2.db". Therefore, we simply specify the annotation parameter as "GPL570".

The function `runDEAnalysis()` outputs a `SummarizedExperiment` object, and this new object extends the input `SummarizedExperiment` object to include the results of the differential analysis. In essence, the resulting object not only retains the expression data matrix and sample information but also incorporates the differential expression analysis results, which are stored under `rowData` attribute. As we can see in the R console, the new attribute `rowData` is added into the `SummarizedExperiment` object. Similar to the data stored under other attributes, this data can also be accessed using the function `rowData()` from the SummarizedExperiment package, as shown in the next step.

5. Access to the differential expression analysis result stored in the `SummarizedExperiment` object:

```
# Extract the differential analysis result
affyDEResults <- SummarizedExperiment::rowData(affyDEExperiment)

# Print in R console
head(affyDEResults, c(3,5))

# console output
DataFrame with 3 rows and 5 columns
           PROBEID      ID        p.value          statistic        logFC

 55101     45828_at     55101     1.53869e-23      -11.8894         -0.465108

 92840     226597_at    92840     3.56762e-23      -11.7546         -0.661995

 727957    227778_at    727957    3.55473e-22      -11.3858         -0.525213
```

In the above snippet, we use the function `rowData()` to extract the differential analysis results in the object named `affyDEExperiment`. This will yield a data frame presenting the differential analysis outcomes, comprising the following columns: (1) `PROBEID`, the original probe ID extracted from the input assay; (2) `ID`, the corresponding Gene Entrez ID; (3) `p.value`, the p-value of the statistical test (limma t-statistic in this case); (4) `logFC`, the log (base 2) fold change; (5) `avgExpr`, the average expression; (6) `logFCSE`, the standard error of logFC; (7) `sampleSize`, the number of samples;

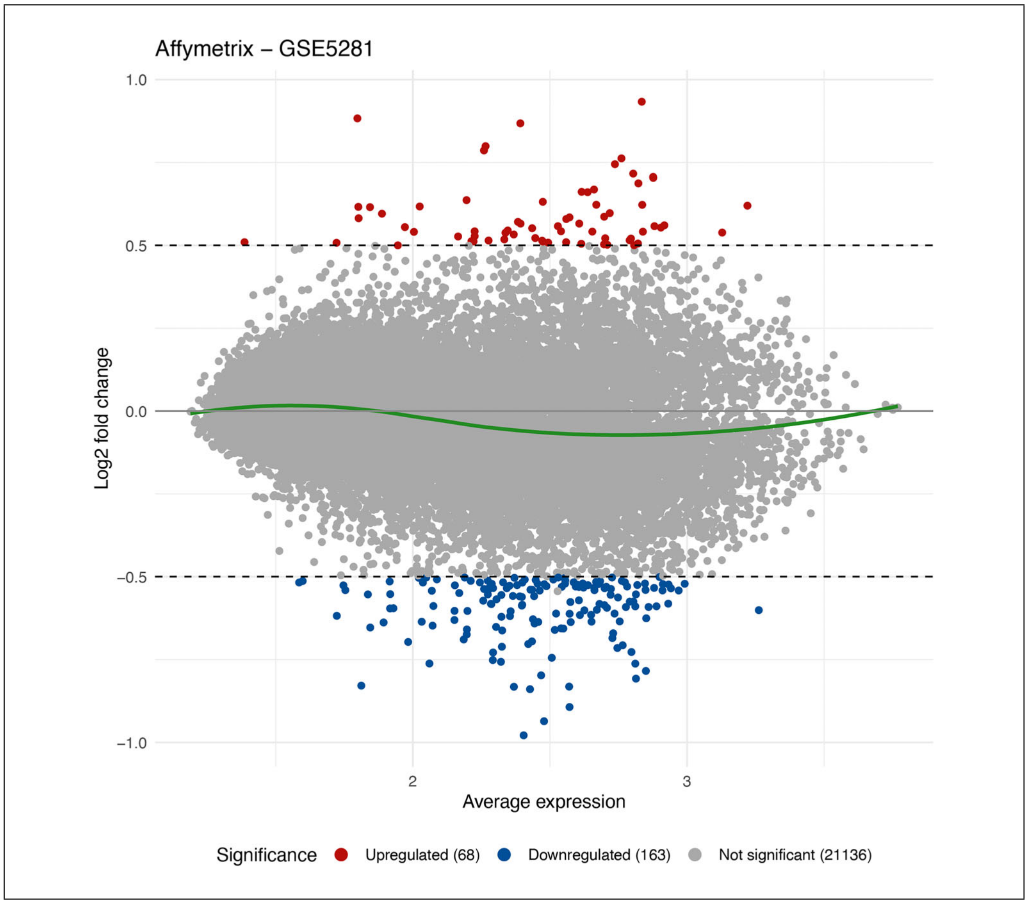


Figure 8 MA plot obtained from the differential analysis of the dataset GSE5281. The x-axis shows the average expression while and the y-axis shows the log2 fold-change. The colored points are the differentially expressed (DE) genes with FDR-adjusted *p*-value (pFDR) $<$.05 and absolute log2 fold-change $>$0.5.

and (8) `pFDR`*, the adjusted p-value for multiple comparisons using Benjamini-Hochberg (Benjamini & Hochberg, 1995). By following this example, users can observe the results of the differential expression analysis as it appears in the console output.*

6. Visualize the differential analysis results using MA plot:

```
# Visualize the differential analysis results using MA plot
RCPA::plotMA(affyDEResults, logFCThreshold = 0.5) +
ggplot2::ggtitle("Affymetrix - GSE5281")
```

Figure 8 shows the MA plot generated from the above code snippet. An MA plot is a scatter plot that compares the average expression level (x-axis) against the log2 fold-change (y-axis). A typical MA plot has most of the points centered around the line of zero logs fold-change. If all points are shifted up or down, it indicates a systematic bias in the data that needs to be corrected.

The function `plotMA()` *takes as input the following parameters: (1)* `DEResult`*, a data frame with DE analysis results as described in the previous step; (2)* `pThreshold`*, a numerical parameter specifying the p-value threshold so that data points corresponding to values exceeding this threshold are depicted in distinct colors (0.05 by default); (3)* `useFDR`*, a logical parameter specifying whether the corrected p-value is used instead of the nominal p-value (*`TRUE` *by default); and (4)* `logFCThreshold`*, a numerical parameter specifying the absolute log2 fold-change so that data points corresponding to values surpassing this threshold are assigned distinct colors (1 by default). In our example, we specify* `logFCThreshold` *as 0.5. The function returns a* `ggplot` *object that can be further customized using the ggplot2 package. We use* `ggtitle()` *function from ggplot2 package to specify the title for the figure.*

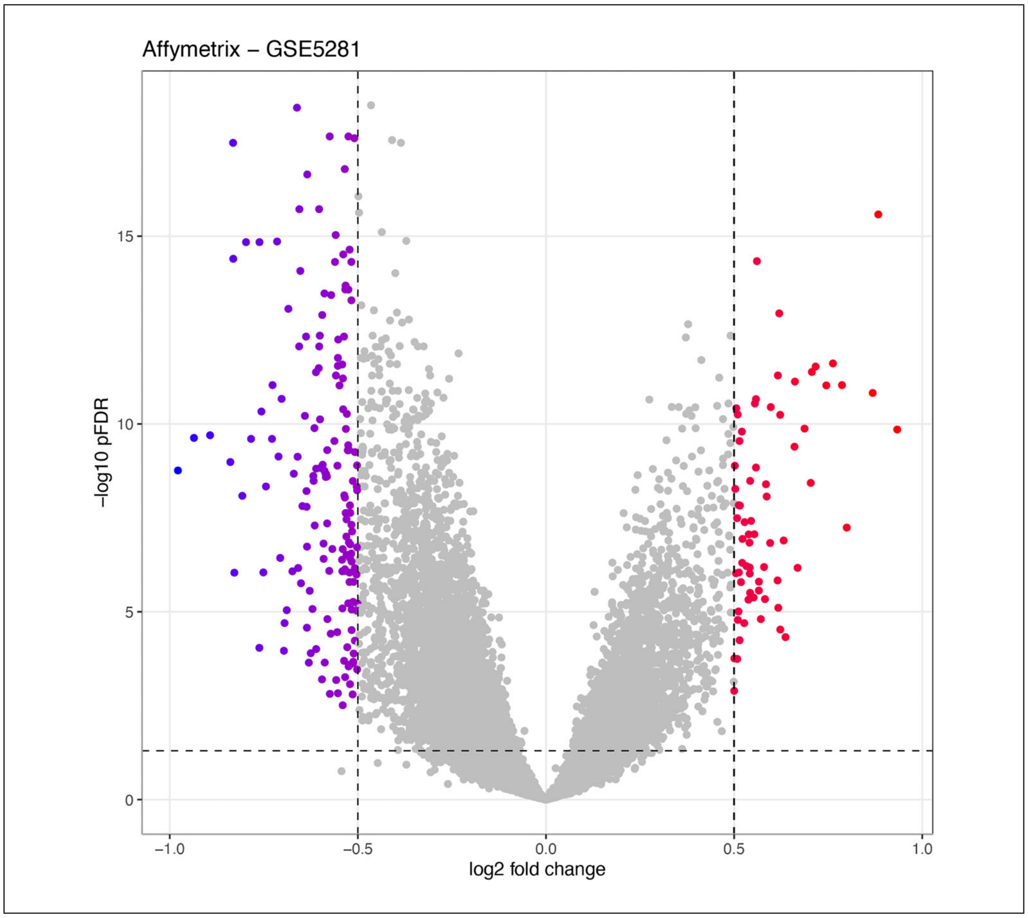


Figure 9 Volcano plot obtained from the differential analysis of the dataset GSE5281. The x-axis shows the log2 fold-change while the y-axis shows the minus log10 of pFDR. The colored points are the differentially expressed (DE) genes with pFDR <.05 and absolute log2 fold-change >0.5.

7. Visualize the differential analysis results using volcano plot:

```
# Visualize the differential analysis results using volcano plot
RCPA::plotVolcanoDE(affyDEResults, logFCThreshold = 0.5) +
ggplot2::ggtitle("Affymetrix - GSE5281")
```

The function `plotVolcanoDE()` requires as input the following parameters: (1) `DEResult`, a data frame containing the differential expression analysis results; (2) `pThreshold`, a numerical parameter specifying the p-value threshold to use for the horizontal line (0.05 by default); (3) `useFDR`, a logical parameter specifying whether the corrected p-value is used instead of the nominal p-value (`TRUE` by default); and (4) `logFCThreshold`, a numerical parameter specifying the logFC threshold to use for the vertical line (1 by default). In our example, we only specify the `logFCThreshold` as 0.5 and use `ggtitle()` to put the title on the graph.

Figure 9 shows the volcano plot generated from the above code snippet. Volcano plot is a scatter plot that compares log2 fold-change against minus log10 p-value. A typical volcano plot has points that are relatively symmetrical around the y-axis of zero log fold-change.

Agilent: GSE61196

The analysis of Agilent datasets is similar to that of Affymetrix datasets. Users can use the following snippets to perform differential expression analysis for Agilent datasets.

8. If users skipped Basic Protocol 2, they can use `RCPA::loadData()` to load the data:

```
# Load the Agilent data processed in Basic Protocol 2
agilDataset <- RCPA::loadData("agilDataset")
```

9. Create a design matrix and a contrast matrix:

```
# Access to the sample information data
agilSampleInfo <- SummarizedExperiment::colData(agilDataset)

# Add a column specifying the condition of the sample,
# which can be either normal or alzheimer
agilSampleConditions <- ifelse(grepl("healthy", agilSampleInfo$source_name_ch1),
"normal", "alzheimer")

# Factorize the newly added column
agilSampleInfo$condition <- factor(agilSampleConditions)

# Update the colData attribute with new column
SummarizedExperiment::colData(agilDataset) <- agilSampleInfo

# Create a design matrix
agilDesign <- model.matrix(~0 + condition, data = SummarizedExperiment::colData(agilDataset))

# Create a contrast matrix
agilContrast <- limma::makeContrasts("conditionalzheimer-conditionnormal", levels =
agilDesign)
```

Similar to working with Affymetrix data, we start by processing the sample condition information that will be used for differential analysis. For the GSE61196 dataset, the samples have two conditions (normal and Alzheimer's) as defined in the `source_name_ch1` *column. We define the design matrix to include the conditions, and similarly, the contrast matrix to compare the Alzheimer's condition vs the normal condition. Here, the formula "*`~0 + condition` *" defines the design matrix to include the effect of the condition. The term* `~0` *indicates that the intercept should not be included in the design matrix since the intercept is not of interest in the analysis. Next, we use the function* `limma::makeContrasts()` *to define the contrast matrix. The formula* `conditionalzheimer-conditionnormal` *defines the contrast matrix to compare the two conditions.*

10. Retrieve the platform information and its gene annotation from GEO database:

The platform of the dataset GSE61196 is GPL4133, whose annotation is not included in our package. Therefore, we need to manually create a mapping data frame that will be used as annotation input of the function `runDEAnalysis()`*. The following are the steps that users can use to obtain this information:*

```
# Download the information for GPL4133 platform:
GPL4133Pl <- GEOquery::getGEO(GEO = "GPL4133")

# Display GPL4133Pl
str(GPL4133Pl)

# Console output
Formal class 'GPL' [package "GEOquery"] with 2 slots
..@ dataTable:Formal class 'GEODataTable' [package "GEOquery"] with 2 slots
.. .. ..@ columns:'data.frame': 22 obs. of 2 variables:
.. .. .. ..$ Column       : chr [1:22] "ID" "COL" "ROW" "NAME" ...
.. .. .. ..$ Description: chr [1:22] "Agilent feature number" "Column" "Row" "NAME" ...
.. .. ..@ table :'data.frame': 45220 obs. of 22 variables:
.. .. .. ..$ ID              : int [1:45220] 1 2 3 4 5 6 7 8 9 10 ...
.. .. .. ..$ COL              : int [1:45220] 266 266 266 266 266 266 266 266 266 266 ...
.. .. .. ..$ ROW             : int [1:45220] 170 168 166 164 162 160 158 156 154 152 ...
.. .. .. ..$ NAME             : chr [1:45220] "GE_BrightCorner" "DarkCorner" "DarkCorner"
   "DarkCorner" ...
.. .. .. ..$ SPOT_ID        : chr [1:45220] "GE_BrightCorner" "DarkCorner" "DarkCorner"
   "DarkCorner"
```

```
  .. .. .. .. .. .. .. ..
..@ header :List of 27
.. ..$ catalog_number          : chr "G4112F"
.. ..$ contact_city             : chr "Palo Alto"
.. ..$ contact_country          : chr "USA"
.. ..$ contact_email             : chr "cag_sales-na@agilent.com"
.. ..$ contact_institute          : chr "Agilent Technologies"
.. ..[output truncated] .. ..
```

In the above code, we use `getGEO()` from the GEO query package to obtain the platform from GEO. When using this function, users need to provide the platform ID to the `GEO` parameter. In this example, we set `GEO` as "GPL4133". The function returns an object of "GPL" class, and we assign the result to `GPL4133Pl` variable. Next, users can use the built-in function `str()` to examine the data structure of this object, as shown in the example code. From the console output, we can see that the `GPL4133Pl` variable has 2 slots. The first slot is `dataTable` that is used to store the gene annotation data, and the second one is `header`, which is a list containing platform metadata.

```
  # Access to the dataTable slot in GPL4133Pl:
        GPL4133Anno <- GEOquery::dataTable(GPL4133Pl)

  # Display GPLL4133Anno:
        str(GPL4133Anno)

  # Console output
        Formal class 'GEODataTable' [package "GEOquery"] with 2 slots
        ..@ columns:'data.frame': 22 obs. of 2 variables:
        .. ..$ Column       : chr [1:22] "ID" "COL" "ROW" "NAME" ...
        .. ..$ Description   : chr [1:22] "Agilent feature number" "Column" "Row" "NAME" ...
        ..@ table :'data.frame': 45220 obs. of 22 variables:
        .. ..$ ID              : int [1:45220] 1 2 3 4 5 6 7 8 9 10 ...
        .. ..$ COL              : int [1:45220] 266 266 266 266 266 266 266 266 266 266 ...
        .. ..$ ROW             : int [1:45220] 170 168 166 164 162 160 158 156 154 152 ...
        .. ..$ NAME            : chr [1:45220] "GE_BrightCorner" "DarkCorner" "DarkCorner"
     "DarkCorner" ...
        .. ..$ SPOT_ID           : chr [1:45220] "GE_BrightCorner" "DarkCorner" "DarkCorner"
     "DarkCorner" ...
        .. ..$ CONTROL_TYPE       : chr [1:45220] "pos" "pos" "pos" "pos" ...
        .. ..$ REFSEQ         : chr [1:45220] "" "" "" "" ...
        .. ..$ GB_ACC         : chr [1:45220] "" "" "" "" ...
        .. ..$ GENE         : int [1:45220] NA NA NA NA NA NA NA NA NA NA ...
        .. ..$ GENE_SYMBOL       : chr [1:45220] "" "" "" "" ...
        .. ..$ GENE_NAME        : chr [1:45220] "" "" "" "" ...
        .. ..$ UNIGENE_ID        : chr [1:45220] "" "" "" "" ...
        .. ..$ ENSEMBL_ID        : chr [1:45220] "" "" "" "" ...
        .. ..$ TIGR_ID           : chr [1:45220] "" "" "" "" ...
        .. ..$ ACCESSION_STRING     : chr [1:45220] "" "" "" "" ...
        .. ..$ CHROMOSOMAL_LOCATION    : chr [1:45220] "" "" "" "" ...
        .. ..$ CYTOBAND          : chr [1:45220] "" "" "" "" ...
        .. ..$ DESCRIPTION        : chr [1:45220] "" "" "" "" ...
        .. ..$ GO_ID           : chr [1:45220] "" "" "" "" ...
        .. ..$ SEQUENCE          : chr [1:45220] "" "" "" "" ...
        .. ..$ SPOT_ID          : logi [1:45220] NA NA NA NA NA NA NA ...
        .. ..$ ORDER            : int [1:45220] 1 2 3 4 5 6 7 8 9 10 ...
```

Because the `dataTable` slot is an object of `GEODataTable` class from the GEOquery package. Therefore, users can use the function `GEOquery::dataTable()` to access this slot as shown in the above code. We assign the object returned by this function to `GPL4133Anno` variable. Users can also use the command `str(GPL4133Anno)` to further examine the returned object. As in the console output, the `GPL4133Anno` *variable stores two data frames. One of the data frames is stored in the `table` attribute, in which the columns are spot IDs and their matched gene annotation. The other data frame is stored in `columns`, which contain the full descriptions of columns in the former data frame.*

```
# Access to annotation data
GPL4133AnnoTbl <- GEOquery::Table(GPL4133Anno)

# display annotation
print(GPL4133AnnoTbl[9:14, c("SPOT_ID", "GENE", "GENE_SYMBOL")])

# Console output
 SPOT_ID            GENE          GENE_SYMBOL

 DarkCorner         NA

 DarkCorner         NA

 DarkCorner         NA

 A_24_P66027        9582          APOBEC3B

 A_32_P77178        NA

 A_23_P212522       23200         ATP11B
```

In a similar manner, one can use the function `GEOquery::Table()` applying on the `GPL4133Anno` object to access the gene annotation data frame. This data frame is assigned to `GPL4133AnnoTbl` variable as in the code snippet. Users can use the function `print()` to print out selected rows and columns.

11. Create a mapping data frame that will be used for differential analysis:

```
# Create the mapping data frame
GPL4133GeneMapping <- data.frame(FROM = GPL4133AnnoTbl$SPOT_ID, TO =
as.character(GPL4133AnnoTbl$GENE), stringsAsFactors = F)

#Display GPL4133GeneMapping:
print(GPL4133GeneMapping[15:20,])

# Console output:
     FROM             TO

     A_24_P934473     100132006

     A_24_P9671       3301

     A_32_P29551      <NA>

     A_24_P801451     10919

     A_32_P30710      9349

     A_32_P89523      <NA>
```

The table `GPL4133AnnoTbl` contains the gene Entrez ID corresponding to the probe IDs under the `GENE` column. Using the two columns `SPOT_ID` and `GENE` from `GPL4133AnnoTbl`, we can create a mapping data frame, which contains two columns showing the probe IDs and their matched Entrez IDs, as required for the function `runDEAnalysis()`. This data frame is assigned to the variable named `GPL4133GeneMapping`.

12. Perform differential analysis using the function `runDEAnalysis()`:

```
# Run differential expression analysis
agilDEExperiment <- RCPA::runDEAnalysis(agilDataset, method = “limma”, design =
agilDesign, contrast = agilContrast, annotation = GPL4133GeneMapping)

# Extract the outcome of differential expression analysis
agilDEResults <- SummarizedExperiment::rowData(agilDEExperiment)

# Print in R console
head(agilDEResults, c(3,5))

# Console output
DataFrame with 3 rows and 5 columns
```

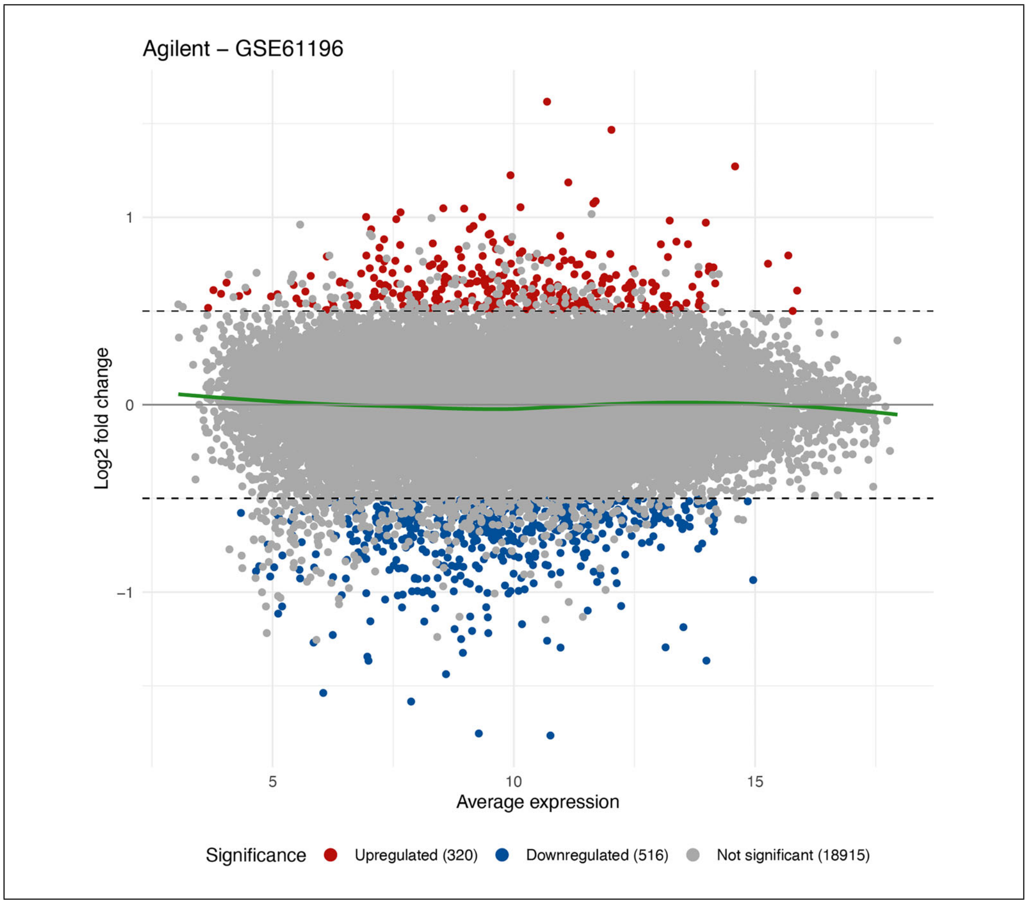


Figure 10 MA plot obtained from the differential analysis of the dataset GSE61196. The x-axis shows the average expression, and the y-axis shows the log2 fold-change. The colored points are the differentially expressed (DE) genes with pFDR <.05 and absolute log2 fold-change >0.5.

```
                PROBEID          ID           p.value        statistic     logFC
150166          A_32_P103815     150166       1.53359e-08    8.42703       0.817593
50717           A_24_P76725      50717        5.69899e-08    -7.81746      -0.858175
79818           A_23_P38830      79818        7.09742e-08    -7.71774      -0.871066
```

In this example, we still use limma for differential analysis, and pass the mapping data frame we just created in the parameter named `annotation`*. We also use the function* `rowData()` *to access the analysis results. By following the above snippet, users expect to see the example of differential analysis result as appears in the console output.*

13. Visualize the results using MA and volcano plots:

```
# MA plot
RCPA::plotMA(agilDEResults, logFCThreshold = 0.5) +
    ggplot2::ggtitle("Agilent - GSE61196")

# Volcano plot
RCPA::plotVolcanoDE(agilDEResults, logFCThreshold = 0.5) +
    ggplot2::ggtitle("Agilent - GSE61196")
```

Figures 10 and 11 show the MA plot and volcano plot generated from the above code snippet. We use the same code to generate the MA plot and volcano plot as in the Affymetrix example.

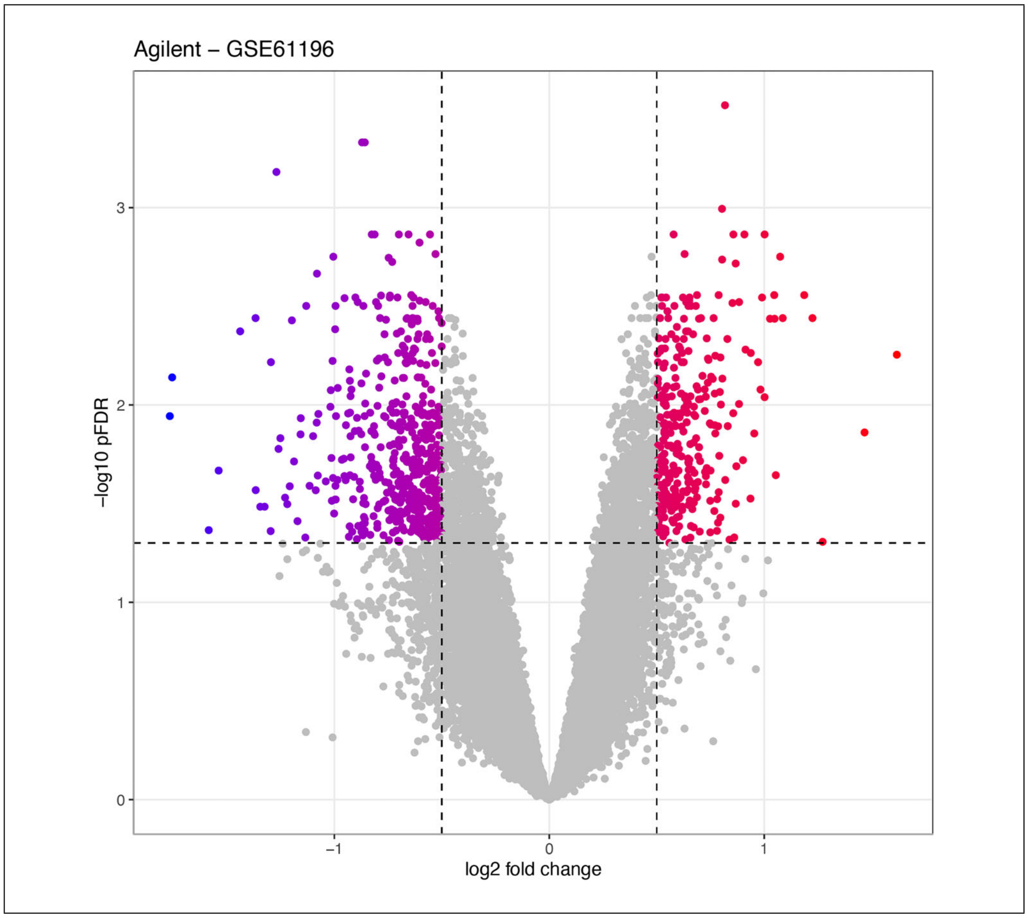


Figure 11 Volcano plot obtained from the differential analysis of the dataset GSE61196. The x-axis shows the log fold-change while the y-axis shows the minus log10 of pFDR. The colored points are the differentially expressed (DE) genes with pFDR <.05 and absolute log2 fold-change >0.5.

BASIC PROTOCOL 4

DIFFERENTIAL ANALYSIS OF RNA-Seq DATA

Users can utilize the same function `runDEAnalysis()` to perform differential analysis for RNA-Seq data. Users are required to provide the `SummarizedExperiment` object containing the expression data and sample information, design matrix, contrast matrix, gene IDs mapping, and the differential analysis method. For RNA-Seq data, users can choose from three different methods implemented in RCPA package: “limma”, “DESeq2”, or “EdgeR”. The choice of the differential analysis method depends on how users pre-process and normalize their data. If users want to work with raw count data, then “DESeq2” or “EdgeR” can be employed as the analysis method. In contrast, if users want to work with normalized expression data, such as TPM, FPKM, or RPKM, then they should choose “limma” as the differential analysis method.

In this protocol, we perform differential analysis for the RNA-Seq dataset GSE153873 processed in Support Protocol. We assume that users have already executed the provided code to obtain the `SummarizedExperiment` object, that contains the raw count data and sample metadata. Here we illustrate the application of `runDEAnalysis()` for conducting differential expression analysis using RNA-Seq data.

Necessary Resources

Hardware

An internet-connected computer or laptop with at least 8 GB of RAM and 20 GB of free hard drive space

Software

R runtime environment (version 4.0.0 or later)

RCPA package from the CRAN repository (see Internet Resources)

Beside the functions in RCPA, we will need to use the functions in the SummarizedExperiment to access the data stored in the SummarizedExperiment object. Similarly, some functions from the ggplot2 package will be used to add the title or modify the figures generated by the plot functions in RCPA. We can ensure the required packages are installed by loading them as shown in the following code snippet:

```
library(RCPA)

library(SummarizedExperiment)

library(ggplot2)
```

Files

A SummarizedExperiment object

A SummarizedExperiment object that has at least two attributes: (1) assays, storing the expression data matrix, in which rows are genes/transcripts and columns are samples; and (2) colData, storing a data frame containing the sample information. Users can refer to Support Protocol for more information about the object SummarizedExperiment.

A gene mapping data frame

A gene mapping data frame has two columns: (1) FROM, containing the genes/transcript IDs currently used in the assay data, and (2) TO, containing the corresponding Entrez ID. Users need to manually prepare this data frame and can save it as .rda, .rds, .txt, or .csv files, or other formats. Consequently, they will use the appropriate R functions to load the file. We will guide users on how to prepare this data frame in our practical example below.

Sample files

Users can use function RCPA::loadData() to load the pre-saved RNA-Seq dataset in Support Protocol from our GitHub repository (https://github.com/tinnlab/RCPA/tree/main/.data). The first step in this protocol will guide the user in loading the pre-saved data using the function.

1. If users skipped Support Protocol, they can use `RCPA::loadData()` to load the data:

```
# Load the RNA-Seq dataset
RNASeqDataset <- RCPA::loadData("RNASeqDataset")
```

2. Create the design and contrast matrices:

```
# Access to the sample information data
RNASeqSampleInfo <- SummarizedExperiment::colData(RNASeqDataset)

# Add a column specifying the condition of the sample,
# which can be either normal or alzheimer
RNASeqSampleConditions <- ifelse(grepl("Alzheimer",
RNASeqSampleInfo$characteristics_ch1.1), "alzheimer", "normal")

# Factorize the newly added column
RNASeqSampleInfo$condition <- factor(RNASeqSampleConditions)

# Update the colData attribute with new column
SummarizedExperiment::colData(RNASeqDataset) <- RNASeqSampleInfo

# Create a design matrix
RNASeqDesign <- model.matrix(~0 + condition, data =
SummarizedExperiment::colData(RNASeqDataset))

# Create a contrast matrix
```

```
RNASeqContrast <- limma::makeContrasts("conditionalzheimer-conditionnormal", levels =
RNASeqDesign)
```

We start our differential analysis by defining the two experimental matrices: design and contrast matrices. For GSE153873, the `characteristics_ch1.1` *column defines the two conditions of the samples: normal and Alzheimer's. We also define the design matrix to include the conditions and the contrast matrix to compare the Alzheimer's condition vs the normal condition. Same as in the example of the Agilent dataset, the formula "*`~0 + condition` *" defines the design matrix to include the effect of condition. The term* `~0` *indicates that the intercept should not be included in the design matrix since the intercept is not of interest in the analysis. Next, we use the* `limma::makeContrasts` *function to define the contrast matrix. The formula* `conditionalzheimer-conditionnormal` *defines the contrast matrix to compare the two conditions.*

3. Define the mapping between the ID of the assay in `SummarizedExperiment` and Entrez ID:

The assay platform of GSE153873 is GPL18573 (Illumina NextSeq 500), whose annotation is not included in our package. Additionally, RNA-Seq datasets available on the GEO database are typically presented as raw counts without any prior preprocessing. This means that annotation information is not provided. Therefore, we cannot employ the `getGEO()` *function to download platform annotations. Instead, we introduce the utilization of the* `select()` *function from the AnnotationDbi package to query various genome-wide annotation databases and to convert gene IDs to the desired IDs.*

```
# Install the genome-wide annotation database for human
if (!require("BiocManager", quietly = TRUE))
install.packages("BiocManager")
BiocManager::install("org.Hs.eg.db")

# Load the annotation database
library(org.Hs.eg.db)

# Get the current gene IDs used in RNA-Seq dataset
RNASeqGeneIDs <- rownames(RNASeqDataset)

# Create a mapping dataframe
GeneSymbolMapping <- AnnotationDbi::select(x = org.Hs.eg.db, keys = RNASeqGeneIDs,
keytype = "SYMBOL",
columns = c("SYMBOL", "ENTREZID"))
   colnames(GeneSymbolMapping) <- c("FROM", "TO")

# Print the first 6 rows into R console
head(GeneSymbolMapping)

# Console output
      FROM      TO
1     SGIP1     84251
2     NECAP2    55707
3     AZIN2     113451
4     AGBL4     84871
5     CLIC4     25932
6     SLC45A1   50651
```

For human genome-wide annotation, we need to load the package org.Hs.eg.db package. Users can install this package from Bioconductor following the provided instructions in our code snippet. After installation, it is necessary to load this database and input it as a parameter for the `select()` *function, which requires the following parameters:* `x`*, an AnnotationDb object such as org.Hs.eg.db;* `keys`*, a vector containing the current IDs;* `keytype`*, a character parameter indicating the type of current gene IDs (Ensembl ID, gene symbol, etc.); and* `columns`*, a vector specifying which types of data (i.e., ID types) can be returned as output.*

In our case, the gene IDs are used as row labels in the expression data matrix. This information is stored under the `rownames` *attribute within the* `SummarizedExperiment` *object. Users can access this attribute using the R built-in function* `rownames()`. *In the above snippet, we assign the output of this function to the variable* `RNASeqGeneIDs` *and pass it as the* `keys` *parameter for the function* `select()`.

Furthermore, based on the R console output shown in Support Protocol, gene symbols are used in this dataset. Therefore, we specify the parameter `keytype` *as* `"SYMBOL"`. *As for the requirement of the annotation parameter when using the* `runDEAnalysis()` *function, we only need the output to have two columns: one for the current gene IDs (*`FROM`*) and the other for Entrez gene IDs (*`TO`*). Thus, we specify the* `columns` *parameter as* `c("SYMBOL", "ENTREZID")`.

The output of the `select()` *function is a data frame that contains two columns as specified in the* `columns` *parameter. In this snippet, we assign the output to the variable* `GeneSymbolMapping`. *Additionally, we modify the column labels of this data frame to match the* `FROM` *and* `TO` *labels, as required for input to the* `runDEAnalysis()` *function. By following this code, users expect to see the example of the mapping data frame as shown in the console output.*

4. Perform differential analysis using the function `runDEAnalysis()`:

```
# Perform differential analysis
RNASeqDEExperiment <- RCPA::runDEAnalysis(RNASeqDataset, method = "DESeq2", design =
RNASeqDesign, contrast = RNASeqContrast, annotation = GeneSymbolMapping)

# Extract the differential analysis results
RNASeqDEResults <- SummarizedExperiment::rowData(RNASeqDEExperiment)
```

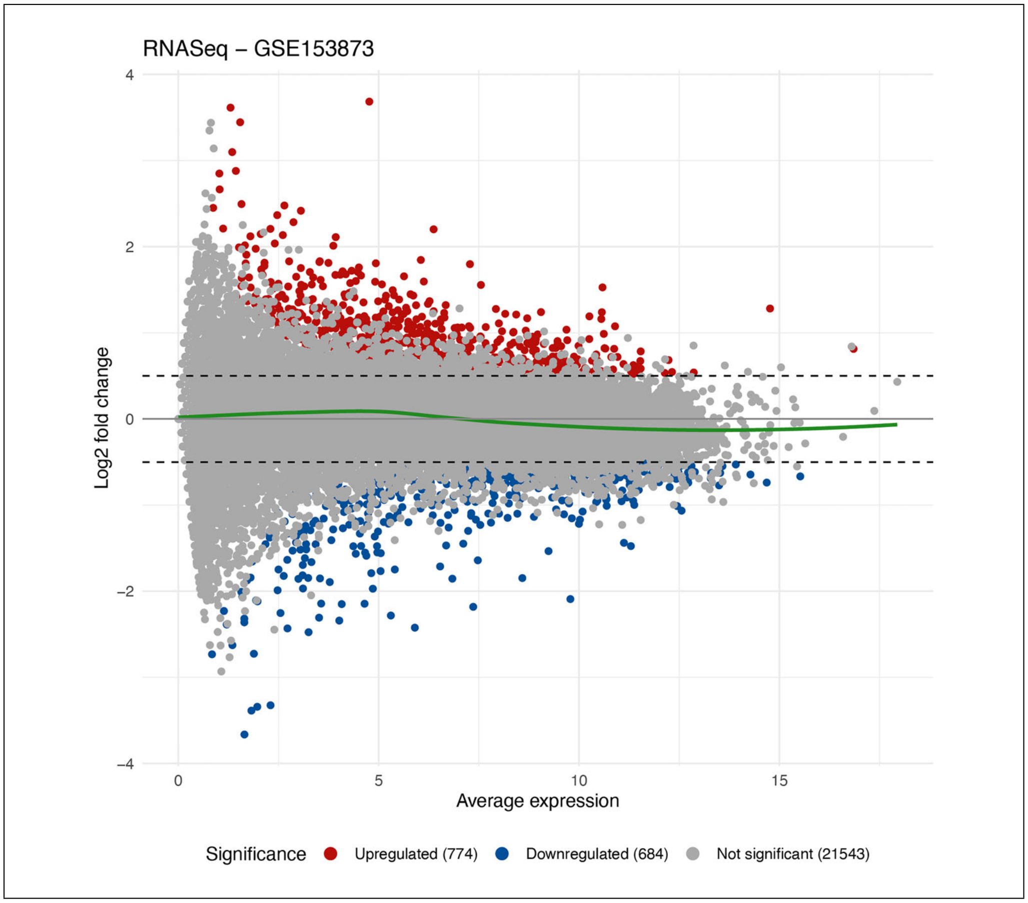


Figure 12 MA plot obtained from the differential analysis of the dataset GSE153873. The x-axis shows the average expression, and the y-axis shows the log2 fold-change. The colored points are the differentially expressed (DE) genes with pFDR <.05 and absolute log fold-change >0.5.

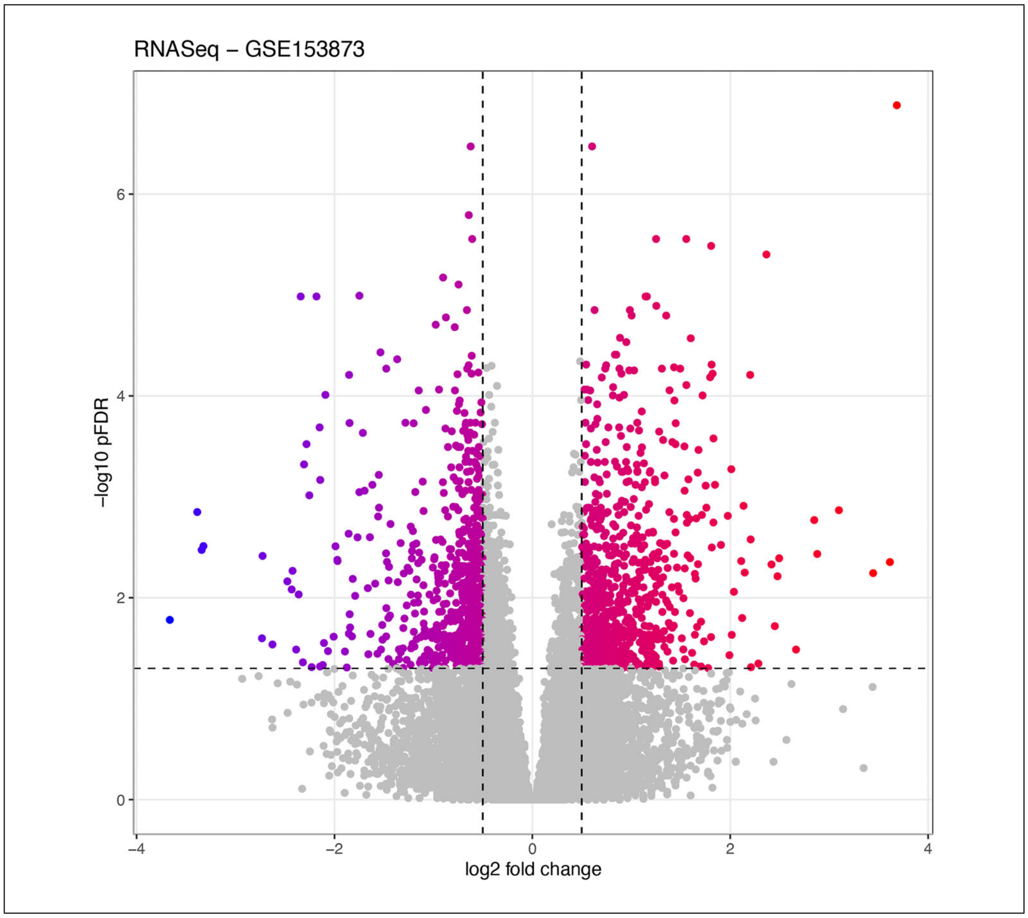


Figure 13 Volcano plot obtained from the differential analysis of the dataset GSE153873. The x-axis shows the log fold-change while the y-axis shows the negative log10 of pFDR. The colored points are the differentially expressed (DE) genes with pFDR <.05 and absolute log fold-change >0.5.

```
# Print out the obtained DE analysis results
head(RNASeqDEResults, c(3,5))

# Console output
DataFrame with 3 rows and 5 columns
             PROBEID        ID          p.value        statistic    logFC
29881        NPC1L1         29881       5.34761e-12    6.89603      3.683626
58485        TRAPPC1        58485       3.22943e-11    -6.63571     -0.623056
643749       TRAF3IP2-AS1   643749      4.10562e-11    6.60022      0.603634
```

Users can simply pass the experimental design and mapping information into the `runDEAnalysis()` function along with the `RNASeqDataset` object. For this example, we use DESeq2 for differential analysis, but users can also apply edgeR for this dataset by setting the `method` parameter to "`edgeR`".

5. Visualize the differential analysis results using MA and volcano plots:

```
# MA plot
RCPA::plotMA(RNASeqDEResults, logFCThreshold = 0.5) +
   ggplot2::ggtitle("RNASeq - GSE153873")

# Volcano plot
RCPA::plotVolcanoDE(RNASeqDEResults, logFCThreshold = 0.5) +
   ggplot2::ggtitle("RNASeq - GSE153873")
```

Users can generate the MA and volcano plots to visualize the differential analysis results by running the above snippet. Figures 12 and 13 show the two plots generated by the above code.

GENE SET ENRICHMENT ANALYSIS

BASIC PROTOCOL 5

Pathway analysis is a systems-level approach that translates differential expression evidence into meaningful biological insights. Within our RCPA package, we implement eight methods for pathway analysis, which can be classified into two groups: gene set enrichment analysis and topology-based (TB) analysis methods. Gene set enrichment analysis generally does not take pathway topology and gene interactions into account whereas topology-based (TB) pathway analysis methods consider pathway topology and gene interactions when performing pathway analysis (Nguyen et al., 2018).

This protocol provides step-by-step instructions for performing gene set enrichment analysis using the implemented function `RCPA::runGeneSetAnalysis()`. The methods implemented for enrichment analysis include the Kolmogorov-Smirnov (KS) test (Massey Jr, 1951), the Wilcoxon test (Wilcoxon, 1992), over-representation analysis (ORA) (Huang et al., 2009; Khatri et al., 2002), fast gene set enrichment analysis (FGSEA) (Korotkevich et al., 2021; Sergushichev, 2016), and gene set analysis (GSA) (Efron & Tibshirani, 2007).

Necessary Resources

Hardware

An internet-connected computer or laptop with at least 8 GB of RAM and 20 GB of free hard drive space

Software

R runtime environment (version 4.0.0 or later)

RCPA package from the CRAN repository (see Internet Resources)

Besides the functions in RCPA, we will need to use the functions in the SummarizedExperiment package to access the data stored in the SummarizedExperiment object. Similarly, some functions from the ggplot2 package will be used to add the title or modify the figures generated by the plot functions in RCPA. We can ensure the required packages are installed by loading them as shown in the following code snippet:

```
library(RCPA)

library(SummarizedExperiment)

library(ggplot2)
```

Files

`SummarizedExperiment` object obtained from differential analysis outlined in Basic Protocols 3 and 4

Sample files

Users can use the function RCPA::loadData() to load the pre-saved differential analysis results of the three datasets from our GitHub repository https://github.com/tinnlab/RCPA/tree/main/.data. The first step in this protocol will guide the user in loading the pre-saved data using the function.

1. If users skipped Basic Protocols 3 and 4, they could use the function `RCPA::loadData()` to load the results:

```
    # loading differential results for Affymetrix data
    affyDEExperiment <- RCPA::loadData("affyDEExperiment")

    # loading the results for Agilent data
    agilDEExperiment <- RCPA::loadData("agilDEExperiment")

    # loading the results for RNA-Seq data
    RNASeqDEExperiment <- RCPA::loadData("RNASeqDEExperiment")
```

If users have not executed all the steps in the previous protocols, they can use the snippet to load the differential analysis results, which is required to run the functions in this protocol. In the following example, we will perform enrichment analysis using the differential analysis results of the RNA-Seq dataset GSE153873, but users can use the same code to perform enrichment analysis using the differential analysis results of Affymetrix and Agilent data.

2. Download gene sets from KEGG or GO:

```
# Download gene sets from KEGG for human
KEGGGenesets <- RCPA::getGeneSets(database = "KEGG", org = "hsa")

# Display gene sets in R console
str(KEGGGenesets)

# Console output
   List of 3
   $ database: chr "KEGG"
   $ genesets:List of 355
   ..$ path:hsa00010: chr [1:67] "10327" "124" "125" "126" ...
   ..$ path:hsa00020: chr [1:30] "1431" "1737" "1738" "1743" ...
   ..$ path:hsa00030: chr [1:31] "132158" "2203" "221823" "226" ...
   ..$ path:hsa00040: chr [1:36] "10327" "10720" "10941" "231" ...
   .. [list output truncated] ...
   $ names : Named chr [1:340] "Glycolysis / Gluconeogenesis" "Citrate cycle (TCA cycle)"
   "Pentose phosphate pathway" "Pentose and glucuronate interconversions" ...
   ..- attr(*, "names")= chr [1:340] "path:hsa00010" "path:hsa00020" "path:hsa00030"
   "path:hsa00040" ...

# Download the gene sets from GO database
   GOTerms <- RCPA::getGeneSets(database = "GO", taxid = 9606, namespace =
   "biological_process")

# Display GOTerms in R console
   str(GOTerms)

# Console output
   List of 3
   $ database: chr "GO"
   $ genesets:List of 12386
   ..$ GO:0000002: chr [1:11] "291" "1890" "4205" "4358" ...
   ..$ GO:0000038: chr [1:19] "30" "51" "215" "225" ...
   ..$ GO:0000079: chr [1:49] "60" "595" "641" "890" ...
   ..$ GO:0000082: chr [1:66] "91" "586" "595" "596" ...
   .. [list output truncated]
   $ names : Named chr [1:12386] "mitochondrial genome maintenance" "very long-chain fatty acid
   metabolic process" "regulation of cyclin-dependent protein serine/threonine kinase activity"
   "G1/S transition of mitotic cell cycle" ...
   ..- attr(*, "names")= chr [1:12386] "GO:0000002" "GO:0000038" "GO:0000079" "GO:0000082" ...
```

A gene set refers to a collection of genes associated with a specific pathway or functional module. This information can be sourced from public databases like GO and KEGG. Within the RCPA package, we have implemented the function `getGeneSets()` *to retrieve gene sets for specific organisms from these databases.*

To download KEGG gene sets, users need to provide the following parameters to the `getGeneSets()` *function: (1)* `database`*, a character parameter specifying the name of pathway database, which is "KEGG"; and (2)* `org`*, a character parameter specifying the organism abbreviation, such as "hsa" for human or "mmu" for mouse. The full list of organism abbreviations can be found at https://www.genome.jp/kegg/catalog/org_list.html.*

To download gene sets from GO (which are called GO terms), users need to provide the following parameters: (1) `database`*, a character parameter specifying the name of pathway database, which is "GO"; (2)* `taxid`*, a character parameter indicating the NCBI* *taxonomy ID of the organism; and (3) `namespace`, the namespace of the GO terms, which includes the following options: "biological_process", "molecular_function", or "cellular_component".*

The function `getGeneSets()` returns a named list with three attributes: (1) `database`, a character parameter specifying the database; (2) `genesets`, a list of gene sets, in which each gene set is a vector genes (Entrez IDs) belonging to the gene set; and (3) `names`, a vector containing the complete names or descriptions of the gene sets. The output of the function `getGeneSets()` can be used as part of the input in any of the methods described in steps 3 to 5 below.

3. Perform enrichment analysis using the Kolmogorov-Smirnov (KS) or Wilcoxon test:

```
# Set seed to create reproducible results
set.seed(1)

# Enrichment analysis using KS test and KEGG pathways
RNASeqKSResult <- RCPA::runGeneSetAnalysis(summarizedExperiment = RNASeqDEExperiment,
genesets = KEGGGenesets, method = "ks")

# Display the result for KS test
print(RNASeqKSResult[1:5, c("ID", "p.value", "pFDR", "score", "name")])

# Console output
 ID              p.value      pFDR       score      name
 path:hsa00190    0            0          1.58       Oxidative
                                                     phosphorylation
 path:hsa05010    0            0          1.05       Alzheimer disease
 path:hsa05012    0            0          1.28       Parkinson disease
 path:hsa05014    0            0          1.14       Amyotrophic lateral
                                                     sclerosis
 path:hsa05016    0            0          1.24       Huntington disease
# Enrichment analysis using Wilcoxon test and KEGG pathways
RNASeqWilcoxResult <- RCPA::runGeneSetAnalysis(summarizedExperiment =
RNASeqDEExperiment, genesets = KEGGGenesets, method = "wilcox")

# Display the result for Wilcoxon test
print(RNASeqWilcoxResult[1:5, c("ID", "p.value", "pFDR", "score", "name")])
# Console output
ID             p.value     pFDR       score      name
path:hsa05016  6.81e-38    1.86e-35   1.242      Huntington disease
path:hsa05014  1.05e-37    1.86e-35   1.138      Amyotrophic lateral sclerosis
path:hsa05012  5.72e-34    6.77e-32   1.281      Parkinson disease
path:hsa05022  1.73e-32    1.53e-30   0.967      Pathways of neurodegeneration - multiple
                                                 diseases
path:hsa00190  2.08e-28    1.48e-26   1.584      Oxidative phosphorylation
```

To perform gene set enrichment analysis, users need to invoke the function `runGeneSetAnalysis()` that implements five different methods: the KS test, the Wilcox test, ORA, FGSEA, and GSA. The parameters for running KS and Wilcox tests in the function `runGeneSetAnalysis()` are as follows: (1) `summarizedExperiment`, a `SummarizedExperiment` object generated from the results of the differential expression analysis, as described in both Basic Protocols 3 and 4; (2) `genesets`, a list of gene set definitions, which can be obtained through the `getGeneSets()` function; and (3) `method`, a character parameter specifying the chosen pathway analysis method, which can be either "ks" or "wilcox". In this example, we perform enrichment analysis using KEGG pathways. If users wish to perform enrichment analysis on GO terms, they can set the parameter `genesets` to `GOTerms` instead.

After the analysis, the output of the function `getGeneSets()` is a table containing the results of the enrichment analysis. The table comprises of the following columns: (1) `ID`, *the ID of the gene set; (2) `p.value`, the p-value of the gene set; (3) `pFDR`, the adjusted p-value of the gene set using the Benjamini-Hochberg method; (4) `score`, the enrichment score of the gene set; (5) `normalizedScore`, the normalized enrichment score of the gene set; (6) `sampleSize`, the total number of samples in the study; (7) `name`, the name of the gene set, and (8) `pathwaySize`, the size of the gene set. Users can examine the table by printing out the content using the `print()` function in the R console, as illustrated above.*

The pathway enrichment score is one of the important statistics returned as one of the columns in the pathway enrichment analysis results. Each method employs a different statistic to represent this score. Understanding how this score is calculated and interpreted is important so that users can choose the most appropriate method for a specific analysis purpose. For Wilcoxon and KS tests, the enrichment score, and the normalized enrichment score (both scores have the same value), represent the log ratio of the number of the observed DE genes in the pathway to the expected DE genes in the pathway. A positive enrichment score indicates that the pathway has more DE genes than expected and vice versa.

4. Perform enrichment analysis using over-representation analysis (ORA):

```
# Specify the threshold to identify DE genes, which are required for ORA
oraArgsList <- list(pThreshold = 0.05)

# Set seed to create reproducible results
set.seed(1)

# Enrichment analysis using ORA and KEGG pathways
RNASeqORAResult <- RCPA::runGeneSetAnalysis(summarizedExperiment = RNASeqDEExperiment,
genesets = KEGGGenesets, method = "ora", ORAArgs = oraArgsList)

# Display the result for ORA
print(RNASeqORAResult[1:5, c("ID", "p.value", "pFDR", "score", "name")])

# Console output

ID               p.value       pFDR        score       name
path:hsa00190    0             0           1.58        Oxidative phosphorylation
path:hsa05010    0             0           1.05        Alzheimer disease
path:hsa05012    0             0           1.28        Parkinson disease
path:hsa05014    0             0           1.14        Amyotrophic lateral
                                                       sclerosis
path:hsa05016    0             0           1.24        Huntington disease
```

To perform gene set enrichment analysis using ORA, the function `runGeneSetAnalysis()` requires the following parameters: (1) `summarizedExperiment`, a `SummarizedExperiment` object generated from the results of the differential expression analysis; (2) `genesets`, a list of gene sets; (3) `method`, a character indicating the enrichment analysis method, which is "ora"; and (4) `ORAArgs`, a list of arguments specific to the ORA method. For the last parameter, the only critical argument to set is the p-value cutoff, which determines which genes are considered differentially expressed. To configure this, users can create a list that includes `pThreshold = 0.05`, as demonstrated in the snippet above. If necessary, users can modify this threshold according to their specific analysis requirements. The same `summarizedExperiment` and `genesets` parameters set in the previous steps should still be used.

Once the `runGeneSetAnalysis()` function has completed its execution, users can anticipate viewing an example of the output in the R console, similar to the previous step. The pathway enrichment score returned by ORA has the same meaning as the ones returned by the KS test and Wilcox test.

5. Perform enrichment analysis using fast gene set enrichment analysis (FGSEA):

```
# Specify a list of arguments tailored for FGSEA:
FGSEAArgsList <- list(minSize = 10)
```

```
# Set seed to create reproducible results
set.seed(1)

# Enrichment analysis using FSGEA and KEGG pathway
RNASeqFGSEAResult <- RCPA::runGeneSetAnalysis(summarizedExperiment = RNASeqDEExperiment,
genesets = KEGGGenesets, method = "fgsea", FgseaArgs = FGSEAArgsList)

# Display the result for FGSEA
print(RNASeqFGSEAResult[c(1:5), c("ID", "p.value", "pFDR", "score", "name")])

# Console output
ID              p.value       pFDR          score         name
path:hsa05014   1.69e-35      5.73e-33      -0.584        Amyotrophic lateral
                                                          sclerosis
path:hsa05016   9.76e-33      1.65e-30      -0.597        Huntington disease
path:hsa05012   2.84e-29      3.21e-27      -0.607        Parkinson disease
path:hsa05020   9.29e-29      7.87e-27      -0.598        Prion disease
path:hsa00190   8.11e-28      5.50e-26      -0.743        Oxidative phosphorylation
```

To gene set perform enrichment using FSGEA, the function runGeneSetAnalysis() requires the following parameters: (1) `summarizedExperiment`, a `SummarizedExperiment` object generated from the results of the differential expression analysis; (2) `genesets`, a list of gene sets; (3) `method`, a character parameter specifying the pathway analysis method, which is "fgsea"; and (4) `FgseaArgs`, a list of arguments customized for FGSEA.

Users have the flexibility to define various arguments for the last parameter `FgseaArgs`, including: (1) `nPermSimple`, the number of permutations for estimation of p-value (1000 by default); (2) `minSize`, the minimum gene set sizes to be tested, with all pathways below this threshold excluded (1 by default); (3) `maxSize`, the maximum gene set sizes to be tested, with all pathways above the threshold excluded (Infinity by default); and (4) `scoreType`, the GSEA score type, which is "std" by default, where the enrichment score is computed as in the original GSEA method (Subramanian et al., 2005) "pos" for a positive one-tailed enrichment test, or "neg" for a negative one-tailed enrichment test. The complete list of arguments for this algorithm can be found in the manual of the FGSEA package. Users can provide the complete list of arguments using `list()`. Here we choose to run FGSEA with `minSize = 10` to exclude gene sets with fewer than 10 genes. By following this code snippet, users can anticipate viewing the output in the R console, similar to the one provided above. If no customization is applied, the `runGeneSetAnalysis()` function will run the method with its default settings.

For FGSEA, the enrichment score (ES) represents the overall rank of genes in the pathway in the ranked list of genes based on the statistics from the differential analysis. A positive enrichment score indicates that the genes in the pathway are ranked higher than expected and vice versa. Note that this score cannot be used to compare pathways since the size of the pathways affects the distribution of the enrichment scores. Instead, to compare the enrichment scores between pathways, FGSEA provides a normalized enrichment score (NES) by dividing the ES by the absolute means of the distribution of the enrichment scores that have the same sign as ES from random permutations.

6. Perform enrichment analysis using gene set analysis (GSA):

```
# Specify the list of arguments customized for GSA
GSAArgsList <- list(method = "maxmean", minsize = 15, maxsize = 500, nperms = 1000)

# Set seed to create reproducible results
set.seed(1)

# Enrichment analysis using GSA and KEGG pathways
RNASeqGSAResult <- RCPA::runGeneSetAnalysis(summarizedExperiment = RNASeqDEExperiment,
genesets = KEGGGenesets, method = "gsa", GSAArgs = GSAArgsList)

# Display the result for GSA
print(RNASeqGSAResult[c(1:5), c("ID", "p.value", "pFDR", "score", "name")])
```

```
# Console output
    ID                  p.value       pFDR          score         name
    path:hsa00020       0.000         0.000         1.417         Citrate cycle (TCA cycle)
    path:hsa01230       0.000         0.000         0.484         Biosynthesis of amino acids
    path:hsa05014       0.000         0.000         0.668         Amyotrophic lateral sclerosis
    path:hsa05016       0.000         0.000         0.704         Huntington disease
    path:hsa01200       0.002         0.111         0.747         Carbon metabolism
```

The function `runGeneSetAnalysis()` takes the following parameters to execute GSA: (1) `summarizedExperiment`, a `SummarizedExperiment` object obtained from differential analysis; (2) `genesets`, a list of gene sets; (3) `method`, a character parameter specifying the pathway analysis method, which is "gsa" in this case; and (4) `GSAArgs`, a list of arguments customized for GSA, for which users can specify the following: (1) `method`, the statistic used to summarize a gene set among "maxmean", "mean", or "absmean" ("maxmean" by default); (2) `minSize`, the minimum threshold for the gene set sizes (10 by default); (3) `maxSize`, the maximum threshold for the gene set sizes (Infinity by default); and (4) `nperms`, the number of permutations. We recommend users consult the manual of the GSA package for a complete list of arguments.

The interpretation of GSA score depends on the values of the method argument in the `GSAArgs` parameter passed to `runGeneSetAnalysis()`. If it is "maxmean", the enrichment score indicates the average of the positive or negative t-statistics of the genes in the pathway, whichever has a larger absolute value. When this argument is "mean" or "absmean", the enrichment score indicates the average t-statistic or the absolute average t-statistic of the genes in the pathway, respectively. In the case of "maxmean" or "mean", a positive enrichment score indicates that the genes in the pathway are more likely to be upregulated and vice versa. In the case of "absmean", a higher enrichment score indicates that the genes in the pathway are more likely to be differentially expressed. For GSA, the normalized enrichment score has the same value as the enrichment score.

7. Visualize enrichment analysis results using volcano plot:

```
RCPA::plotVolcanoPathway(PAResult = RNASeqFGSEAResult, topToLabel = 10) +
   ggplot2::ggtitle("RNASeq - GSE153873 - FGSEA")
```

Using the function `plotVolcanoPathway()`, users can visualize the enrichment results. Figure 14 shows the results using the FGSEA method and KEGG pathways. The volcano plot shows the pathway enrichment scores on the x-axis and minus log10 p-values on the y-axis. The `plotVolcanoPathway()` function returns a `ggplot` object which can be customized using functions from ggplot2 package. The figure also displays the names of the 10 most significant pathways. Using the volcano plot, users can have an overview of the enrichment analysis, as well as the general trend of the pathway regulation. In this analysis, many of the significant pathways have negative enrichment scores, i.e., downregulated.

8. Visualize enrichment analysis results using forest plot:

```
# Create a list containing top 20 pathways from the result
RNASeqFGSEAToPlot <- list("RNASeq - GSE153873 - FGSEA" = RNASeqFGSEAResult[1:20,])

# Generate forest plot:
RCPA::plotForest(resultsList = RNASeqFGSEAToPlot, yAxis = "name", statLims = c(-3.5, 1))
```

Figure 15 shows the plot generated by the above code using the function `plotForest()`. The forest plot shows the normalized enrichment score and their confidence interval as individual points along a horizontal line. The `plotForest()` function requires the following parameters: (1) `resultsList`, a named list of data frames from pathway analysis; (2) `yAxis`, a character parameter specifying which column of the result data frame from pathway analysis is used to label to y-axis; and (3) `statLims`, a numeric vector of length 2 specifying the limits for the score to use in the x-axis. In this code snippet, we first sort the results according to the p-values and select the top 20 pathways as RNASeqFGSEAToPlot. After that, we set `resultsList = RNASeqFGSEAToPlot`, `yAxis = "name"`, and `statLims = c(-3.5, 1)`.

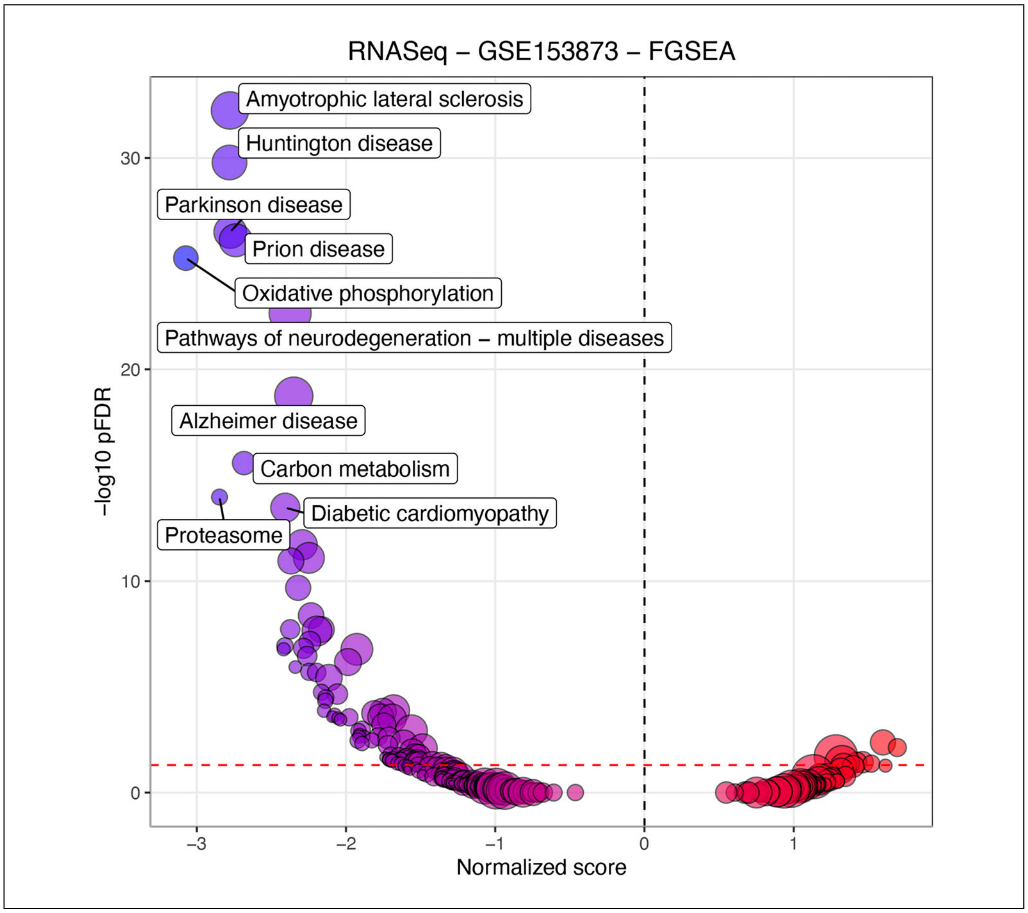


Figure 14 Volcano plot of enrichment analysis obtained from FGSEA for the dataset GSE153873. The x-axis shows the normalized enrichment score while the y-axis shows the minus log10 pFDR. Each point on the figure represents a pathway or gene set. The size of a point is proportional to the number of genes in the corresponding gene set. The color of each point is determined by the normalized enrichment score. The top 10 pathways with the smallest pFDR are labeled with the pathway names.

9. Visualize enrichment analysis results using a network graph of pathways:

```
# Select the top 20 pathways from the results
RNASeqFGSEAToPlot <- list("RNASeq -- GSE153873 -- FGSEA" = RNASeqFGSEAResult[1:20,])

# Get IDs for top 20 pathways
selectedPathways <- RNASeqFGSEAResult$ID[1:20]

# Generate network graph of selected pathways
pltHtml <- RCPA::plotPathwayNetwork(
PAResults = RNASeqFGSEAToPlot,
genesets = KEGGGenesets,
selectedPathways = selectedPathways,
statistic = "normalizedScore",
mode = "continuous",
edgeThreshold = 0.75,
file = tempfile(fileext = ".html") # Or use a user-specified file path)
```

Figure 16 shows the resulting pathway network. Users can flexibly change the layout into the following styles: breadthfirst, circle, cola, concentric, cose, cose-bilkent, dagre, grid, and random. The function `plotPathwayNetwork()` *visualizes the pathway network, in which nodes are pathways. If two pathways have common genes among them, then there is an edge connecting the pathways. This plot is useful for understanding the relationships among the pathways and identifying the modules of related pathways that are*

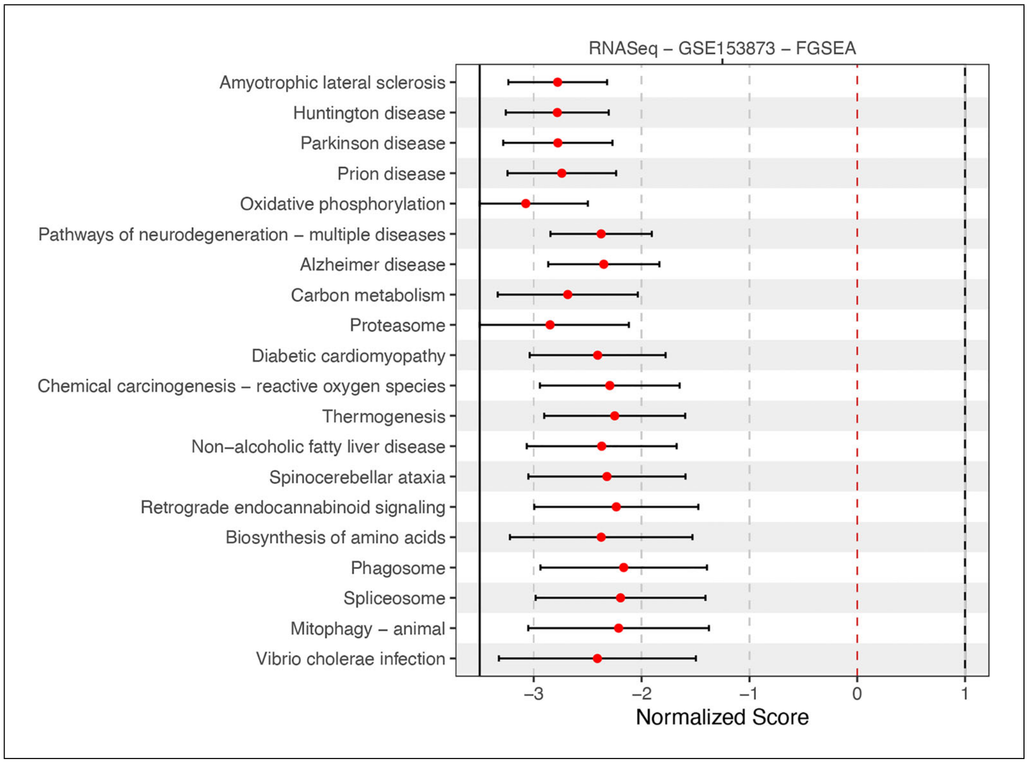


Figure 15 Forest plot of enrichment scores of the most significant pathways obtained from FGSEA on GSE6153873 dataset. The x-axis shows the enrichment scores while the y-axis shows the pathway names. The red dots are the normalized scores for each pathway in each dataset. The horizontal segments around the red dots represent the confidence interval for the normalized scores.

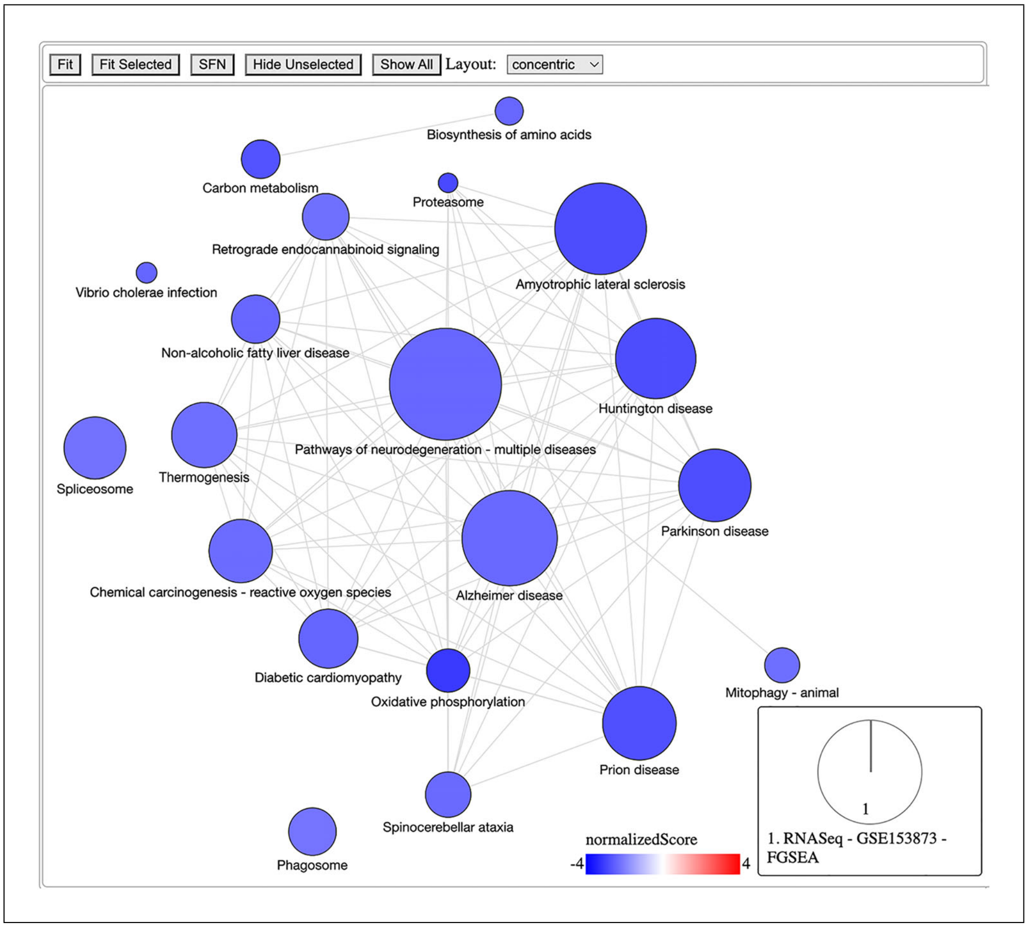
impacted together. The function has the following parameters: (1) `PAResults`*, a named list of pathway analysis results; (2)* `genesets`*, a list of gene sets, which can be obtained through the getGeneSets() function; (3)* `selectedPathways`*, a vector of pathway IDs to be displayed (NULL by default, in which all pathways would be shown); (4)* `statistic`*, a character parameter specifying which statistic of the pathway analysis results to be displayed; (5)* `mode`*, a character parameter of the mode to use to color the nodes, which can be "discrete" (the color of the nodes is determined by p-value significance) or "continuous" (the color of the nodes is determined by the magnitude of the statistic); and (6)* `edgeThreshold`*, a numeric value from 0 to 1 indicating the threshold to draw edges (0.5 by default).*

In the above snippet, we generate the pathway network for the pathways in `selectedPathways` *variables using the results from FGSEA on RNA-Seq dataset. Thus, we set* `PAResults` *as* `RNASeqFGSEAToPlot`*. We extract the list of gene sets to plot and their names from* `KEGGGenesets` *and pass them to the* `genesets` *and* `labels` *parameters. We also set* `statistic` *as the* `normalizedScore` *column and* `mode` *as continuous. Once the execution is complete, the function* `plotPathwayNetwork()` *will display the pathway network in a web browser. At the same time, it will write the HTML code of the graph into a file specified by the* `file` *parameter. If the file parameter is not specified, the function will write the HTML code into a temporary file and print the path to the file in the R console. The function finally returns the HTML code as a string.*

BASIC PROTOCOL 6

TOPOLOGY-BASED (TB) PATHWAY ANALYSIS

The previous protocol discusses gene set enrichment analysis using five different methods (Wilcoxon test, KS test, ORA, FGSEA, and GSA). Though powerful, enrichment


Fit
Fit Selected
SFN
Hide Unselected
Show All
Biosynthesis of amino acids
Carbon metabolism
Proteasome
Retrograde endocannabinoid signaling
Vibrio cholerae infection
Amyotrophic lateral sclerosis
Non-alcoholic fatty liver disease
Huntington disease
Pathways of neurodegeneration - multiple diseases
Spliceosome
Thermogenesis
Parkinson disease
Chemical carcinogenesis - reactive oxygen species
Alzheimer disease
Diabetic cardiomyopathy
Oxidative phosphorylation
Mitophagy - animal
Prion disease
Spinocerebellar ataxia
Phagosome
normalizedScore
-4
4
1
1. RNASeq - GSE153873 - FGSEA

Layout: concentric

Figure 16 Pathway network obtained from FGSEA results for the dataset GSE153873. In this graph, each node represents a pathway. Two pathways are connected by an edge if they share at least 75% of the genes of the smaller pathway. The node color shows the direction and magnitude of the normalized score from the enrichment analysis result. The width of the edges is proportional to the number of genes shared by the two pathways. Users can change the layout of the graph into the following styles: breadthfirst, circle, cola, concentric, cose, cose-bilkent, dagre, grid, and random.

methods do not take into consideration pathway topology and the interactions among the genes in the pathways. There exist also topology-based (TB) methods that consider pathway topology and gene interactions when performing pathway analysis. The RCPA package includes three TB methods: signaling pathway impact analysis (SPIA) (Draghici et al., 2007; Tarca et al., 2009), centrality-based pathway enrichment for ORA extension (CePa ORA), and for GSA extension (CePa GSA) (Gu et al., 2012; Gu & Wang, 2013). These methods are all accessible through the `runPathwayAnalysis()` function.

In addition to a `SummarizedExperiment` object that contains the differential analysis results, users need to provide pathways with gene interactions. In our context, each pathway is represented by a graph in which genes are nodes and edges are interactions among genes. Note that SPIA and CePa packages require completely different formats for their graph objects. To ease this process, we also include functions in RCPA that can automatically generate the required network data from KEGG for each method. These include `getSPIAKEGGNetwork()` for SPIA, and `getCePaPathwayCatalogue()` for CePa ORA and CePa GSA. If users want to understand more about the graph objects for these methods, they can consult the documentation for SPIA and CePa. Subsequently, users have the option to create their own network object, adhering to the same data structure used in SPIA and CePa, and supply it to the function `runPathwayAnalysis()`.

Necessary Resources

Hardware

An internet-connected computer or laptop with at least 8 GB of RAM and 20 GB of free hard drive space

Software

R runtime environment (version 4.0.0 or later)
RCPA package from the CRAN repository (see Internet Resources)

Besides the functions in RCPA, we will need to use the functions in the SummarizedExperiment to access the data stored in the SummarizedExperiment object. Similarly, some functions from the ggplot2 package will be used to add the title or modify the figures generated by the plot functions in RCPA. We can ensure the required packages are installed by loading them as shown in the following code snippet:

```
library(RCPA)

library(SummarizedExperiment)

library(ggplot2)
```

Files

`SummarizedExperiment` object obtained from differential analysis outlined in Basic Protocols 3 and 4
Sample files

Users can use the function RCPA::loadData() to load the pre-saved differential analysis results of the three datasets from our GitHub repository (https://github.com/tinnlab/RCPA/tree/main/.data). The first step in this protocol will guide the user in loading the pre-saved data using the function.

1. If users skipped Basic Protocols 3 and 4, they can use `RCPA::loadData()` to load the results:

```
# loading differential results for Affymetrix data

affyDEExperiment <- RCPA::loadData("affyDEExperiment")

# loading the results for Agilent data

agilDEExperiment <- RCPA::loadData("agilDEExperiment")

# loading the results for RNA-Seq data

RNASeqDEExperiment <- RCPA::loadData("RNASeqDEExperiment")

# loading KEGG gene sets

KEGGGenesets <- RCPA::loadData("KEGGGenesets")
```

If users have not executed all the steps in the previous protocols, they can use the snippet to load the differential analysis results, which is required to run the functions in this protocol. In the following example, we will perform enrichment analysis using the differential analysis results of the RNA-Seq dataset GSE153873, but users can use the same code to perform enrichment analysis using the differential analysis results of microarray datasets.

2. Retrieve pathway topology from KEGG for SPIA:

```
# Retrieve gene networks from KEGG for SPIA

SPIANetwork <- RCPA::getSPIAKEGGNetwork(org = "hsa", updateCache = FALSE)

# Display SPIANetwork

str(SPIANetwork)

# Console output

List of 3

   $ network:List of 312

   .. $ path:hsa00010:Formal class ’graphNEL’ [package "graph"] with 6 slots

   .. ..@ nodes : chr [1:67] "226" "229" "230" "217" …
.. ..@ edgeL :List of 67
.. .. ..$ 226 :List of 1
.. .. .. ..$ edges: int [1:8] 45 46 42 43 44 41 39 40
.. .. ..$ 229 :List of 1
.. .. .. ..$ edges: int [1:8] 45 46 42 43 44 41 39 40
.. .. ..$ 230 :List of 1
.. .. .. ..$ edges: int [1:8] 45 46 42 43 44 41 39 40
.. .. ..$ 217 :List of 1
.. .. .. .. .. [list output truncated]
.. .. .. ..@ defaults:List of 1
.. .. .. .. ..$ subtype: logi NA
.. ..@ nodeData :Formal class ’attrData’ [package "graph"] with 2 slots
.. .. .. ..@ data : Named list()
.. .. .. ..@ defaults: list()
.. ..@ renderInfo:Formal class ’renderInfo’ [package "graph"] with 4 slots
.. .. .. ..@ nodes: list()
.. .. .. ..@ edges: list()
.. .. .. ..@ graph: list()
.. .. .. ..@ pars : list()
.. ..@ graphData :List of 1
.. .. ..$ edgemode: chr "directed"
.. [list output truncated] …
$ names: Named chr [1:312] "Glycolysis / Gluconeogenesis" "Citrate cycle (TCA cycle)"
"Pentose phosphate pathway" "Pentose and glucuronate interconversions" …
..- attr(*, "names")= chr [1:312] "path:hsa00010" "path:hsa00020" "path:hsa00030"
"path:hsa00040" …
$ sizes: Named int [1:312] 67 30 31 36 34 32 30 36 49 47 …
..- attr(*, "names")= chr [1:312] "path:hsa00010" "path:hsa00020" "path:hsa00030"
"path:hsa00040" …
```

The function `getSPIAKEGGNetwork()` retrieves pathway information from KEGG and puts the information in a format that is compatible with SPIA. This function requires the following parameters: (1) `org`, a character specifying the organism abbreviation; and (2) `updateCache`, a boolean parameter to enable and disable cache update (`FALSE` at default). In our specific example, we have curated the KEGG network information for humans by specifying the org parameter as “hsa”. The function returns a network object, which is represented as a list with the following attributes: (1) `network`, a named list of pathway network definitions; (2) `names`, a named character list containing the names of pathways; and (3) `sizes`, a named integer list indicating the sizes of the pathways.

The network definition (`network` attribute) encompasses a list of pathways, with each pathway represented by a “graphNEL” graph object from the graph package (Gentleman et al., 2023). Users can refer to the document of this package to learn more about the functions designed for parsing this object. The output of the above code will be used as one of the input parameters when running SPIA using `runPathwayAnalysis()` function.

3. Perform topology-based (TB) pathway analysis using SPIA:

```
# Specify the list of arguments specific for SPIA
SPIAArgsList <- list(nB = 1000, pThreshold = 0.05)

# Set seed to create reproducible results
set.seed(1)

# Run SPIA on RNA-Seq dataset
RNASeqSPIAResult <- RCPA::runPathwayAnalysis(summarizedExperiment = RNASeqDEExperiment,
network = SPIANetwork, method = "spia", SPIAArgs = SPIAArgsList)

# Display the result for dataset
print(RNASeqSPIAResult[1:5, c("ID", "p.value", "pFDR", "score", "name")])

# Console output
```

```
ID                p.value      pFDR         score        name
path:hsa05016     1.84e-28     5.22e-26     33.5         Huntington disease
path:hsa05014     8.02e-27     1.14e-24     68.4         Amyotrophic lateral sclerosis
path:hsa05020     1.77e-24     1.44e-22     43.6         Prion disease
path:hsa00190     2.02e-24     1.44e-22     46.9         Oxidative phosphorylation
path:hsa05022     3.59e-23     2.04e-21     213.5        Pathways of neurodegeneration -
                                                         multiple diseases
```

For SPIA, runPathwayAnalysis() requires the following parameters: (1) `summarizedExperiment`, a SummarizedExpriment object that has differential expression results; (2) `network`, an object of pathway networks, which can be obtained from the function `getSPIAKEGGNetwork()`; (3) `method`, a character parameter indicating the pathway analysis method, which is "SPIA" in this case; and (4) `SPIAArgs`, a list of arguments tailored for the SPIA method. Because SPIA is based on over-representation and signaling perturbations accumulation, therefore, users can specify the list of arguments for this method containing: (1) `nB`, the number of bootstrap iterations for perturbation (2000 by default); and (2) `pThreshold`, the p-value cutoff, which determines DE genes (0.05 by default). If the list of arguments is not provided, `runPathwayAnalysis()` function will use default settings. If users want to change the default arguments of the SPIA method, we recommend that they consult the manual of the ROntoTools package.

The output of the function `runPathwayAnalysis()` is a table that has the following columns: (1) `ID`, the ID of the pathway; (2) `p.value`, the p-value of the pathway; (3) `pFDR`, the adjusted p-value of the pathway using the Benjamini-Hochberg method; (4) `score`, the score of the pathway; (5) `normalizedScore`, the normalized score of the pathway; (6) `sampleSize`, the total number of samples in the study; (7) `name`, the name of the pathway; and (8) `pathwaySize`, the size of the pathway.

The pathway score returned by SPIA represents the sum of the absolute difference between the perturbation factors of the differentially expressed genes with its observed log fold change in the pathway. The perturbation factor in SPIA not only considers the statistic of the gene but also the topology of the pathway. A higher score indicates that the pathway is more likely to be perturbed. This score is also not suitable for comparing pathways of different sizes. Instead, SPIA provides a normalized score that is the z-score of the pathway score with respect to the scores obtained from random permutations of the sample labels.

4. Retrieve pathway topology from KEGG for CePa ORA and CePa GSA:

```
# Retrieve pathway information from KEGG for CePa ORA and CePa GSA:
CePaNetwork <- RCPA::getCePaPathwayCatalogue(org = "hsa", updateCache = FALSE)

# Display CePaNetwork
str(CePaNetwork)

# Console output
List of 3
   $ network:List of 3
   ..$ pathList :List of 312
   .. ..$ path:hsa00010: chr [1:268] "1" "2" "3" "4" …
   .. ..$ path:hsa00020: chr [1:101] "269" "270" "271" "272" …
   .. ..$ path:hsa00030: chr [1:158] "1" "2" "3" "4" …
   .. ..$ path:hsa00040: chr [1:75] "508" "509" "510" "511" …
   .. .. [list output truncated]
   ..$ interactionList:'data.frame': 62995 obs. of 3 variables:
   .. ..$ interaction.id: chr [1:62995] "1" "2" "3" "4" …
   .. ..$ input : chr [1:62995] "226" "226" "226" "226" …
   .. ..$ output : chr [1:62995] "2203" "8789" "5211" "5213" …
   ..$ Mapping :'data.frame': 7758 obs. of 2 variables:
   .. ..$ node.id: chr [1:7758] "226" "229" "230" "217" …
   .. ..$ symbol : chr [1:7758] "226" "229" "230" "217" …
   ..- attr(*, "class")= chr "pathway.catalogue"
```

```
$ names: Named chr [1:312] "Glycolysis / Gluconeogenesis" "Citrate cycle (TCA cycle)"
"Pentose phosphate pathway" "Pentose and glucuronate interconversions" …
..- attr(*, "names")= chr [1:312] "path:hsa00010" "path:hsa00020" "path:hsa00030"
"path:hsa00040" …
$ sizes: Named int [1:312] 268 101 158 75 138 74 101 192 154 223 …
..- attr(*, "names")= chr [1:312] "path:hsa00010" "path:hsa00020" "path:hsa00030"
"path:hsa00040" …
```

The function `getCePaPathwayCatalogue()` retrieves the network objects required for CePa ORA and CePa GSA. The function takes as input the following parameters: (1) `org`, a character specifying the organism abbreviation; and (2) `updateCache`, a boolean parameter to enable and disable cache update (`FALSE` at default). The function `getCePaPathwayCatalogue()` returns a network object, which is represented as a list with the following attributes: (1) `network`, a named list of pathway network definitions; (2) `names`, a named character list containing the names of pathways; and (3) `sizes`, a named integer list indicating the sizes of the pathways.

Users can apply the function `str()` to inspect the resulting network object, as shown in the snippet above. The console output reveals that the network definitions returned by the functions `getSPIAKEGGNetwork()` and `getCePaPathwayCatalogue()` exhibit distinct data structures. The `CePaNetwork` defines the network such that each pathway is represented as a vector of unique interaction IDs. To identify the genes associated with each interaction ID, users can refer to the `interactionList`, which is presented as a three-column matrix. The first column represents the interaction ID, the second column signifies the input node ID, and the third column designates the output node ID. Both input and output nodes correspond to genes identified by their Entrez IDs.

5. Perform pathway analysis using CePa ORA:

```
# Specify the list of argument tailored for CePa ORA
CePaORAArgsList <- list(cen = "equal.weight", pThreshold = 0.05)

# Set seed to create reproducible results
set.seed(1)

# Run CePa ORA
RNASeqCePaORAResult <- RCPA::runPathwayAnalysis(summarizedExperiment = RNASeqDEExperiment,
network = CePaNetwork, method = "cepaORA", CePaORAArgs = CePaORAArgsList)

# Display the result
print(RNASeqCePaORAResult[1:5, c("ID", "p.value", "pFDR", "score", "name")])

# Console output

ID               p.value       pFDR         score        name
path:hsa00010    0.000999      0.0156       30.3         Glycolysis / Gluconeogenesis
path:hsa00020    0.000999      0.0156       19.2         Citrate cycle (TCA cycle)
path:hsa00640    0.000999      0.0156       16.2         Propanoate metabolism
path:hsa00650    0.000999      0.0156       14.1         Butanoate metabolism
path:hsa00270    0.000999      0.0156       24.2         Cysteine and methionine
                                                         metabolism
```

The function `runPathwayAnalysis()` for the method CePa ORA requires the following set of parameters: (1) `summarizedExperiment`, a `SummarizedExperiment` object obtained from differential analysis; (2) `network`, the pathways network object, which can be obtained from the function `getCePaPathwayCatalogue()`; (3) `method`, a character parameter specifying the pathway analysis method, which is “cepaORA” in this case; and (4) `CePaORAArgs`, a list of arguments customized for CePa ORA. For the list of CePa ORA arguments, users can specify the following: (1) `cen`, the centrality (i.e., gene weights) measurement among “equal.weight”, “in.degree”, “out.degree”, “betweenness”, “in.reach”, and “out.reach” (“equal.weight”’ by default); and (2) `pThreshold`, p-value cutoff determining the DE genes (0.05 by default). Users can consult the manual of the CePa package for a complete list of arguments.

For each pathway, CePa ORA returns the score calculated by summing up the weights of the DE genes in the pathway, where the weights represent the importance of the genes in the pathway based on the centrality of the genes. A higher score indicates that the DE genes appear in more important positions in the pathway. This score is, however, not suitable for comparing pathways of different sizes. In RCPA, we calculate the normalized score by dividing the score by the sum of the weights of all genes in the pathway.

6. Perform pathway analysis using CePa GSA:

```
# Specify the list of argument tailored for CePa GSA
CePaGSAArgsList <- list(cen = "equal.weight", nlevel = "tvalue_abs", plevel = "mean")

# Set seed to reproducible results
set.seed(1)

# Run CePa GSA on RNA-Seq dataset
RNASeqCePaGSAResult <- RCPA::runPathwayAnalysis(summarizedExperiment = RNASeqDEExperiment,
network = CePaNetwork, method = "cepaGSA", CePaGSAArgs = CePaGSAArgsList)

# Display the result
print(RNASeqCePaGSAResult[1:5, c("ID", "p.value", "pFDR", "score", "name")])

# Console output
  ID              p.value     pFDR        score       name

  path:hsa00740   0.257       0.999       0.964       Riboflavin metabolism

  path:hsa04978   0.303       0.999       0.949       Mineral absorption

  path:hsa00020   0.331       0.999       0.885       Citrate cycle (TCA cycle)

  path:hsa05320   0.351       0.999       0.791       Autoimmune thyroid disease

  path:hsa04740   0.359       0.999       0.600       Olfactory transduction
```

For CePa GSA, the function `runPathwayAnalysis()` takes the following parameters: (1) `summarizedExperiment`, a `SummarizedExperiment` object obtained from differential analysis; (2) `network`, the pathways network object, which can be obtained from the functions `getCePaPathwayCatalogue()`; (3) `method`, a character parameter specifying the pathway analysis method, which is "cepaGSA" in this case; and (4) `CePaGSAArgs`, a list of arguments customized for CePa GSA. For the list of CePa GSA arguments, users can specify the following: (1) `cen`, the centrality (i.e., gene weights) measurement among "equal.weight", "in.degree", "out.degree", "betweenness", "in.reach", and "out.reach" ("equal.weight"' by default); (2) `nlevel`, the node-level statistics, which can be "tvalue" (t-statistics), "tvalue_abs" (absolute t-statistics, default) and "tvalue_sq" (squared t-statistics); and (3) `plevel`, the statistic used to summarize a pathway among "max", "min", "median", "sum", "mean" and "rank" ("mean" by default). Users can consult the manual of the CePa package for a complete list of arguments.

Compared to CePa ORA, the interpretation of the pathway score returned by CePa GSA depends on the combination of the nlevel and plevel arguments specified in the CePaGSA parameter. For example, when `nlevel = "tvalue"` and `plevel = "mean"`, the score is the average t-statistic (after adjusted using topology centrality) of the genes in the pathway. In this case, a positive score indicates that the genes in the pathway are more likely to be upregulated and vice versa. The normalized score of CePa GSA is calculated by RCPA depending on the `plevel` argument. When `plevel` is "max", "min", "median", "mean", or "rank", the normalized score is the same as the score. When `plevel` is "sum", the normalized score is calculated by dividing the score by the sum of the weights of all genes in the pathway.

7. Visualize pathway analysis results using volcano, forest plot, and network of pathways:

```
# Generate volcano plot for SPIA results
RCPA::plotVolcanoPathway(PAResult = RNASeqSPIAResult, topToLabel = 15) +
ggplot2::ggtitle("RNASeq - GSE153873 - SPIA")
```

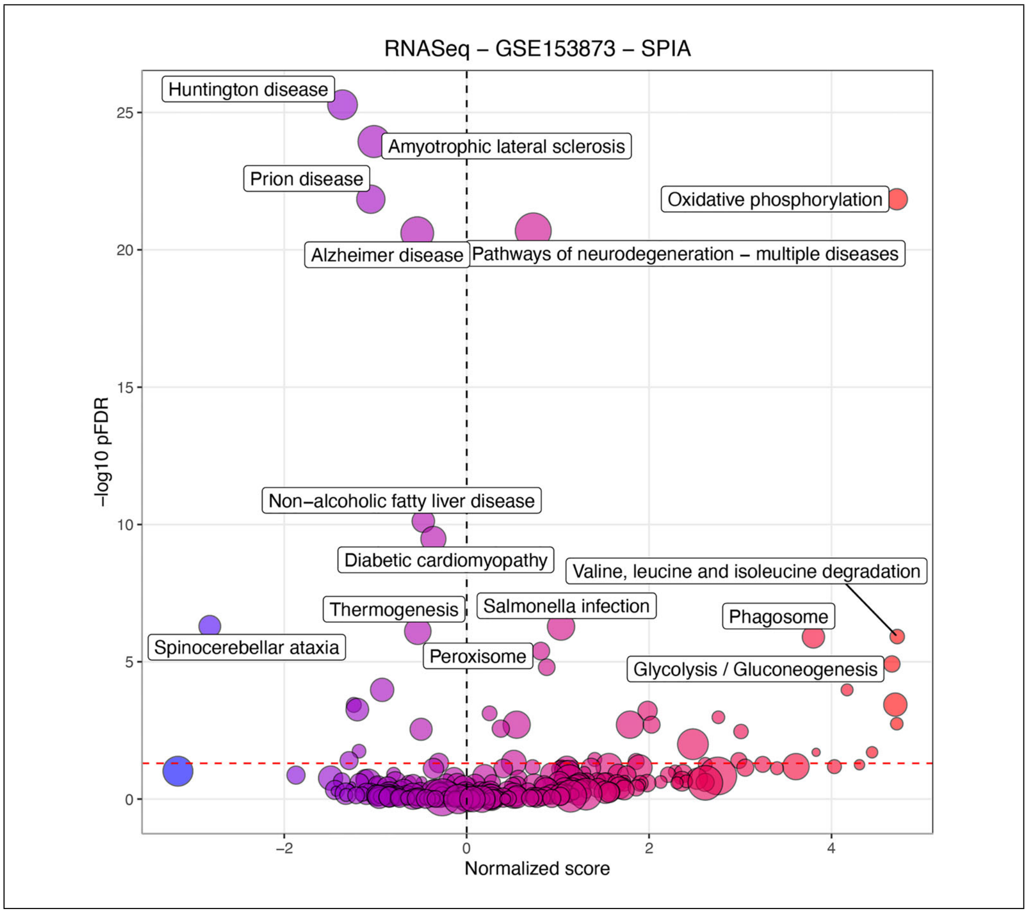


Figure 17 Volcano plot of pathway analysis obtained from SPIA for dataset GSE153873. The x-axis shows the normalized pathway score while the y-axis shows the minus log10 pFDR of the pathways. The size of the points is proportional to the number of genes in the corresponding pathway. The color of the points is determined by the normalized score. The top 15 pathways with the smallest pFDR are labeled with the pathway names.

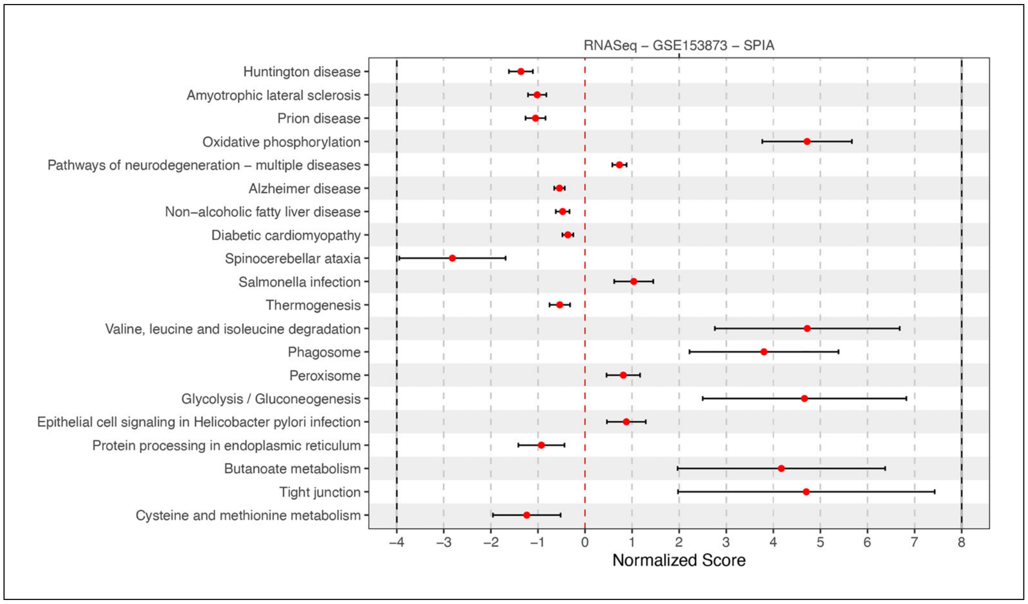


Figure 18 Forest plot of pathway analysis obtained from SPIA for dataset GSE153873. The x-axis shows the normalized enrichment scores while the y-axis shows the pathway names. The red dots are the normalized scores for each pathway in each dataset. The horizontal segments around the red dots represent the confidence interval for the normalized scores.

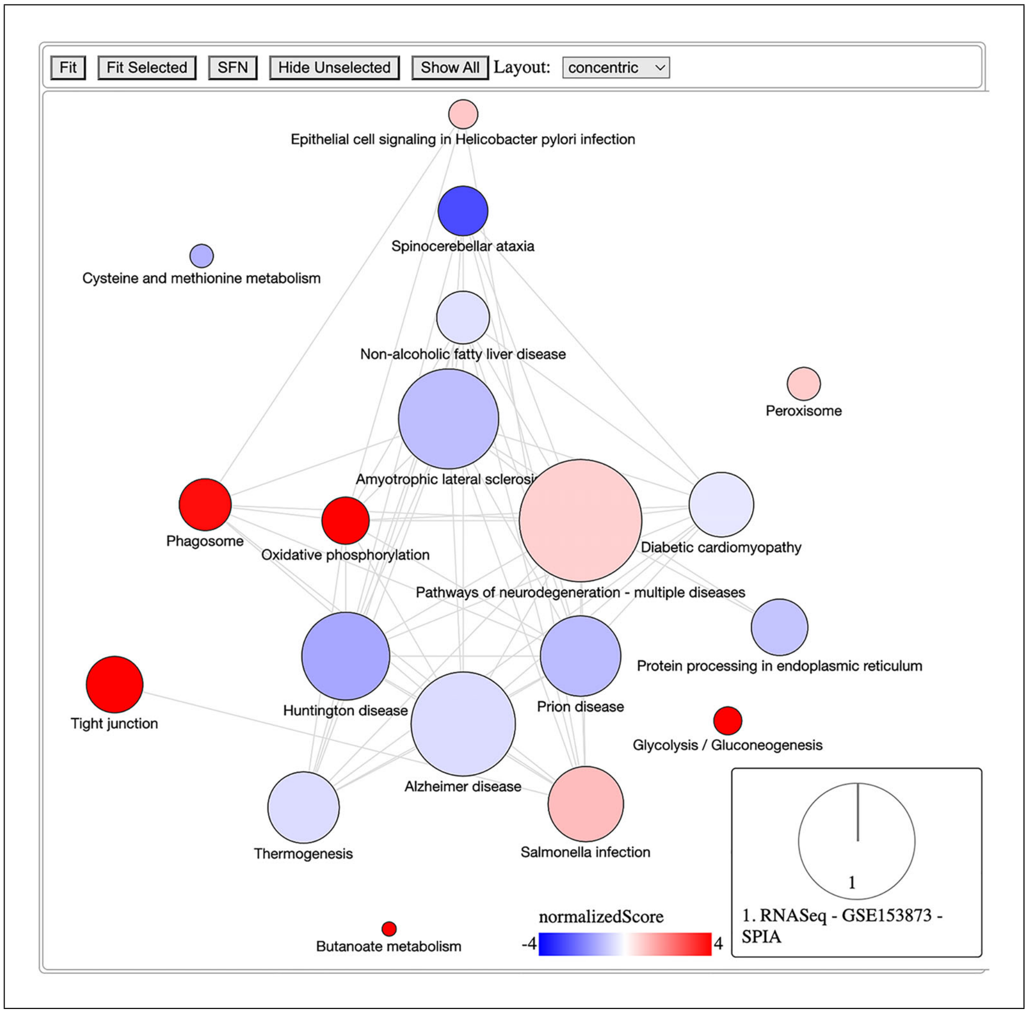

Fit
Fit Selected
SFN
Hide Unselected
Show All
Layout:
concentric
Epithelial cell signaling in Helicobacter pylori infection
Spinocerebellar ataxia
Cysteine and methionine metabolism
Non-alcoholic fatty liver disease
Peroxisome
Amyotrophic lateral sclerosis
Phagosome
Oxidative phosphorylation
Diabetic cardiomyopathy
Pathways of neurodegeneration - multiple diseases
Protein processing in endoplasmic reticulum
Tight junction
Huntington disease
Prion disease
Glycolysis / Gluconeogenesis
Alzheimer disease
Thermogenesis
Salmonella infection
1
1. RNASeq - GSE153873 - SPIA
normalizedScore
-4
4
Butanoate metabolism

Figure 19 Pathway network obtained from SPIA results of the dataset GSE153873. In this graph, each node represents a pathway. Two pathways are connected by an edge if they share at least 75% of the genes of the smaller pathway. The node color shows the direction and magnitude of the normalized score from the pathway analysis result. Users can change the layout of the graph into the following styles: breadthfirst, circle, cola, concentric, cose, cose-bilkent, dagre, grid, and random.

Figure 17 shows the volcano plot for the results obtained from SPIA. Similar to the visualization for gene set enrichment analysis, users can easily visualize the volcano plot for TB pathway analysis using the function `plotVolcanoPathway()`.

```
# Select the top 20 pathways from the results
RNASeqSPIAToPlot <- list("RNASeq - GSE153873 - SPIA" = RNASeqSPIAResult[1:20,])
selectedPathways <- RNASeqSPIAResult$ID[1:20]

# Generate forest plot:
RCPA::plotForest(resultsList = RNASeqSPIAToPlot, yAxis = "name", statLims = c(-4, 8))
```

Using the code snippet above, users can visualize the forest plot, as shown in Figure 18. The figure shows the pathway score and their confidence interval for the 20 most significant pathways obtained from SPIA analysis.

```
# Generate network graph of selected pathways
pltHtml <- RCPA::plotPathwayNetwork(
PAResults = RNASeqSPIAToPlot,
genesets = KEGGGenesets,
selectedPathways = selectedPathways,
statistic = "normalizedScore",
mode = "continuous",
edgeThreshold = 0.75)
```

Figure 19 shows the plot generated from the above code snippet. We use the same code to generate the pathway network plot as in the example of enrichment analysis to visualize the result from pathway analysis using SPIA. For CePa ORA and CePa GSA, users can easily substitute the `RNASeqSPIAResult` *variable with* `RNASeqCePaORAResult` *and* `RNASeqCePaGSAResult`*.*

DATA INTEGRATION AND VISUALIZATION

BASIC PROTOCOL 7

This protocol describes the RCPA functions that are of integrating and visualizing analysis results obtained from multiple datasets and analysis methods. We introduce two distinct kinds of integrative analysis strategies: meta-analysis and consensus analysis. Meta-analysis is a set of statistical approaches used to combine multiple independent but related studies (Normand, 1999). Meta-analysis can potentially increase the statistical power of analysis methods and improve the accuracy of the findings. In contrast, consensus analysis involves merging the outcomes derived from different analyses, resulting in a unified and consolidated conclusion, as shown in our previous publication (Nguyen et al., 2021). This strategy allows users to compare the results of multiple methods. Using consensus pathway analysis, users can observe the effects of the underlying hypotheses that form the basis of the analysis methods of interest. Consensus pathway analysis can be extended to include results obtained from multiple datasets and methods at the same time to understand the effects of method hypothesis, experiment design, and other factors.

Using the RCPA package, meta-analysis can be performed at both gene- and pathway-level, while consensus analysis is only performed at the pathway-level. In the following example, we still use the same three GEO datasets, GSE5281 (Affymetrix), GSE61196 (Agilent), and GSE153873 (RNA-Seq). Note that all three datasets study the same condition: Alzheimer's disease.

Necessary Resources

Hardware

An internet-connected computer or laptop with at least 8 GB of RAM and 20 GB of free hard drive space

Software

R runtime environment (version 4.0.0 or later)
RCPA package from the CRAN repository (see Internet Resources)

Besides the functions in RCPA, we will need to use the functions in the SummarizedExperiment to access the data stored in the SummarizedExperiment object. Similarly, some functions from the ggplot2 package will be used to add the title or modify the figures generated by the plot functions in RCPA. We can ensure the required packages are installed by loading them as shown in the following code snippet:

```
library(RCPA)

library(SummarizedExperiment)

library(ggplot2)
```

Files

`SummarizedExperiment` objects obtained from differential analysis outlined in Basic Protocols 3 and 4
Data frame containing gene set enrichment and/or pathway analysis results described in Basic Protocols 5 and 6, to perform pathway-level meta-analysis and consensus analysis
Sample files

Users can use the function RCPA::loadData() to load the pre-saved data and analysis results of the three datasets in the previous protocols from our GitHub repository *(https://github.com/tinnlab/RCPA/tree/main/.data). There are detailed steps in this protocol that guide the users in loading the pre-saved data using the function.*

Gene-level meta-analysis

Here we illustrate how users can combine the differential analysis results obtained from multiple datasets. We assume that users have performed differential analysis and pathway analysis using our previous protocols, and that these results are stored in `SummarizedExperiment` objects, namely `affyDEExperiment`, `agilDEExperiment`, and `RNASeqDEExperiment`, as described in Basic Protocols 3 and 4.

1. If users skipped Basic Protocols 3 and 4, they can use the function `RCPA::loadData()` to load the results:

```
# DE analysis result from Affymetrix dataset:
affyDEExperiment <- RCPA::loadData("affyDEExperiment")

# DE analysis result from Agilent dataset:
agilDEExperiment <- RCPA::loadData("agilDEExperiment")

# DE analysis result from RNA-Seq dataset:
RNASeqDEExperiment <- RCPA::loadData("RNASeqDEExperiment")
```

   Users can execute the above code snippet to load the results of the differential analysis on three datasets: GSE5281, GSE61196, and GSE153873. These results are prerequisites for running the functions in this protocol.

2. Compile differential expression results of multiple datasets into a list:

```
# Extract the differential analysis result obtained from previous protocols
affyDEResults <- SummarizedExperiment::rowData(affyDEExperiment)
agilDEResults <- SummarizedExperiment::rowData(agilDEExperiment)
RNASeqDEResults <- SummarizedExperiment::rowData(RNASeqDEExperiment)

# Prepare the input list of DE results
DEResults <- list(
"Affymetrix - GSE5281" = affyDEResults,
"Agilent - GSE61196" = agilDEResults,
"RNASeq - GSE153873" = RNASeqDEResults)
```

   Users can apply the function `rowData()` to retrieve the differential analysis results for each dataset and then compile all results from all three datasets into a single list. This list will be used as input for the following steps.

3. Generate a Venn diagram and query common genes:

```
  # Generate a venn diagram plot
  RCPA::plotVennDE(DEResults = DEResults, topToList = 10)

  # Retrieve a list of common DE genes among multiple datasets
  commonDEGenes <- RCPA::getCommonDEGenes(DEResults = DEResults)

  # Display the results
  print(commonDEGenes[1:5,])

  # Console output:
ID        Symbol    Description
84964     ALKBH6    alkB homolog 6
2539      G6PD      glucose-6-phosphate dehydrogenase
8694      DGAT1     diacylglycerol O-acyltransferase 1
23433     RHOQ      ras homolog family member Q
10106     CTDSP2    CTD small phosphatase 2
```

   The function `plotVennDE()` takes as input the following parameters: (1) `DEResults`, a list containing DE analysis results, which is the list created in the previous

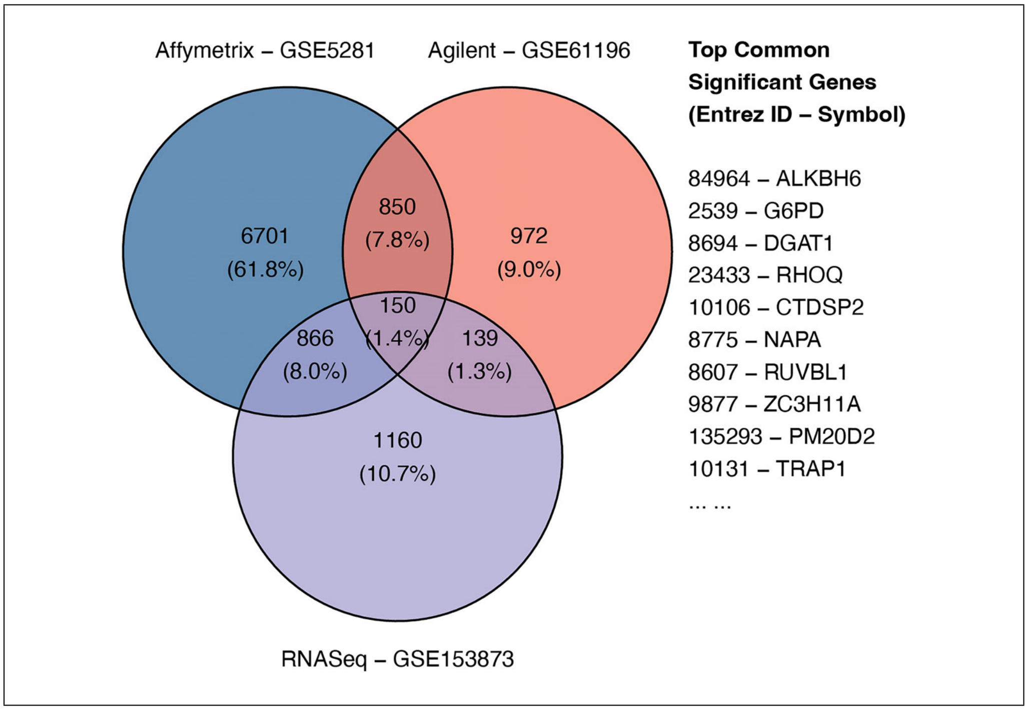


Figure 20 Venn diagram of differentially expressed (DE) genes among three datasets: GSE5281, GSE61196, and GSE153873. The numbers in each section represent the number of DE genes. There are 150 DE genes that are shared among all three datasets. The right side of the figure shows the Entrez IDs and gene symbols of the top 10 genes that are significant in all datasets.

step; (2) `pThreshold`, the p-value threshold to determine if a gene is differentially expressed (0.05 by default); (3) `useFDR`, a boolean parameter indicating that the adjusted p-value is used instead of the nominal p-value (`TRUE` by default); (4) `stat`, a character parameter, which indicates the additional statistics column in the differential analysis result to use for filtering the DE genes ("logFC" by default); and (5) `statThreshold`, a numeric parameter specifying the threshold for the stat parameter (0 by default). This function returns a Venn diagram as a `ggplot` object, and we can use ggplot2 functions to further customize the plot. In this code snippet, we can use the `plotVennDE()` function with its default settings to generate the Venn diagram of DE genes among datasets.

Figure 20 shows the Venn diagram obtained from the above code. In general, it is important to check the intersection of DE genes between different datasets. This helps to identify potential issues in the analysis design. For example, if the intersection of DE genes between these datasets is small, it indicates that these datasets might not be compatible for meta-analysis. Users can plot a Venn diagram to examine the number of common DE genes between analysis results from different datasets. The right side of the figure shows the Entrez IDs and gene symbols of the top 10 genes that are differentially expressed in all datasets. Users can list all common genes using the function `getCommonDEGenes()`, which has the same parameters as of the function `plotVennDE()`. This function returns a data frame containing the following columns: (1) `ID`, Entrez IDs of common DE genes; (2) `Symbol`, gene symbols; and (3) `Description`, gene descriptions.

4. Perform meta-analysis using one of the six methods:

```
# Perform meta-analysis using Stouffer's method
metaDEResult <- RCPA::runDEMetaAnalysis(DEResults = DEResults, method = "stouffer")

# Display the result:
head(metaDEResult)
```

```
# Console output
ID          p.value      pFDR         logFC        logFCSE

84964       1.54e-19     2.59e-15     -0.469       0.0472

10106       6.84e-16     4.22e-12     0.327        0.0392

7108        8.99e-16     4.22e-12     -0.368       0.0480

4713        1.01e-15     4.22e-12     -0.531       0.0604

10382       1.77e-15     5.92e-12     -0.665       0.0733

396         2.20e-15     6.16e-12     -0.343       0.0393
```

The function `runDEMetaAnalysis()` takes as input the following parameters: (1) `DEResults`, a list containing DE analysis results, which is the list created in step 1; and (2) `method`, a character parameter indicates the method used for meta-analysis. The function performs meta-analysis using one of the following methods: Fisher's method (Fisher, 1925), Stouffer's method (Stouffer et al., 1949), addCLT (Nguyen, Tagett, et al., 2016), minimum p-value (Tippett, 1931), geometric mean (Vovk & Wang, 2020), and restricted maximum likelihood (REML) (Viechtbauer, 2005). The first five methods are used for p-value combination, while the last method combines both p-values and log2 fold-change. The last method is implemented using the function `metagen()` from the meta package (Balduzzi et al., 2019), with the following settings: standardized mean differences approach (SMD) as summary measurement, the restricted maximum likelihood (REML) as an estimator of between-study variance distribution and applying Knapp-Hartung adjustment (Knapp & Hartung, 2003).

Each of these methods has its own advantages and limitations, and researchers may choose to use one or more depending on the specific research question and available data. In the above code snippet, we choose Stouffer's method for combining p-value by specifying `method` as "stouffer". To run other methods, users can simply replace "stouffer" by "fisher", "addCLT", "minP", "geoMean", or "REML".

The function returns a data frame that has the following columns: (1) `ID`, the Entrez ID of gene; (2) `p.value`, the meta p-value of gene; (3) `pFDR`, the adjusted meta p-value of genes using Benjamini-Hochberg method; (4) `logFC`, the meta log fold change of gene; and (5) `logFCSE`, the standard error of logFC of gene. It is also important to note that `metagen()` function is executed whenever the `runDEMetaAnalysis()` is called. Therefore, we always obtain the meta log2 fold-change even when the "REML" is not chosen.

5. Visualize meta-analysis results using gene heatmap:

```
# Select the top 40 most significant genes:
genesToPlot <- metaDEResult$ID[1:40]

# Get the full description of the genesToPlot:
genesAnnotation <- RCPA::getEntrezAnnotation(genesToPlot)
labels <- genesAnnotation[genesToPlot, "Description"]

# Create a list containing the results from individual analysis and meta analysis
affyDEResults <- SummarizedExperiment::rowData(affyDEExperiment)
agilDEResults <- SummarizedExperiment::rowData(agilDEExperiment)
RNASeqDEResults <- SummarizedExperiment::rowData(RNASeqDEExperiment)
resultsToPlot <- list(
"Affymetrix - GSE5281" = affyDEResults,
"Agilent - GSE61196" = agilDEResults,
"RNASeq - GSE153873" = RNASeqDEResults,
"Meta-analysis" = metaDEResult)

# Generate gene heatmap
RCPA::plotDEGeneHeatmap(DEResults = resultsToPlot, genes = genesToPlot, labels = labels,
negLog10pValueLims = c(0, 5), logFCLims = c(-1, 1))
```

Figure 21 shows the plot generated by the above code. The figure shows the gene heatmap for the top 40 most significant genes, as chosen based on their nominal p-values in meta-analysis. The gene heatmap shows log2 fold-changes of the genes along with their

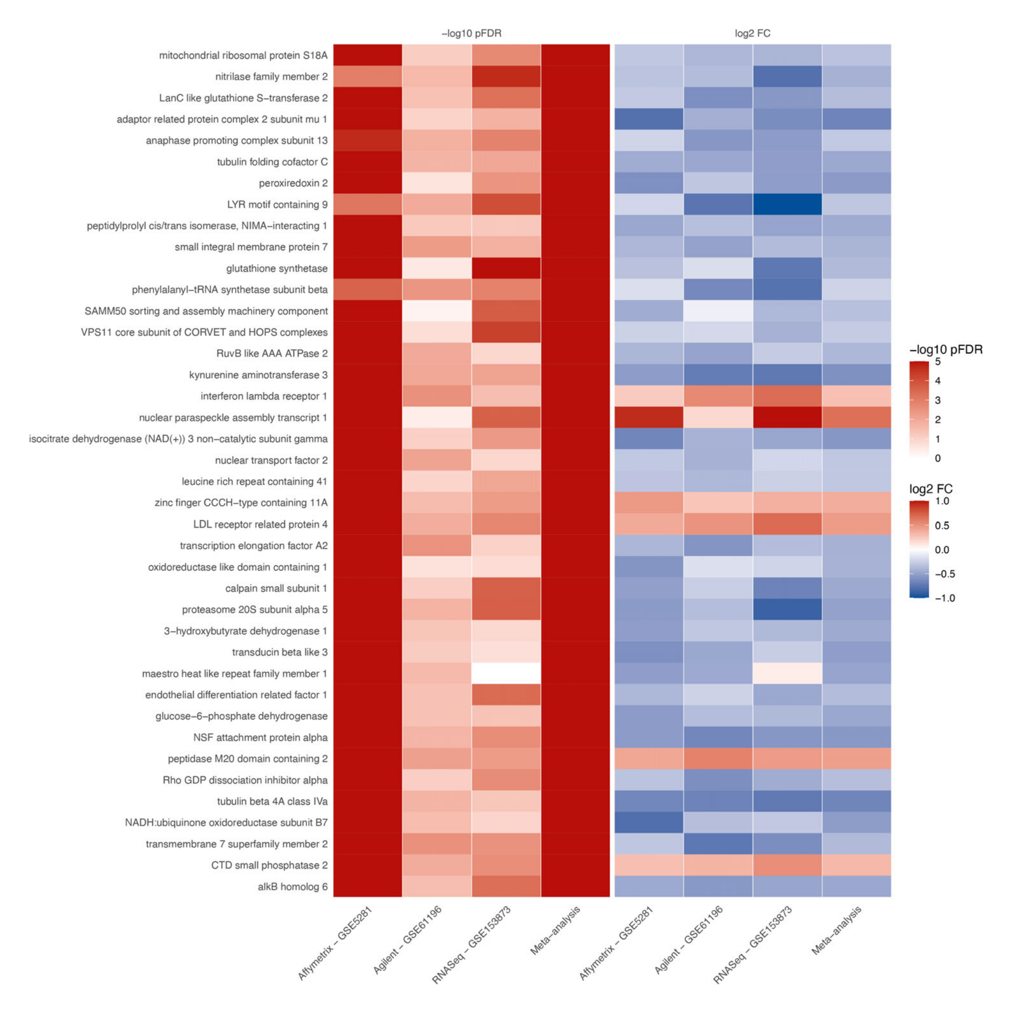


Figure 21 Gene heatmap obtained from the three datasets (GSE5281, GSE61196, and GSE153873) and their meta-analysis. The top x-axis shows the statistics (minus log10 pFDR and log2 fold-change) while the bottom x-axis shows the study names. The y-axis shows the gene description, but users can specify other labels, such as Entrez ID or gene symbol. Only the top 40 most significant genes in meta-analysis are shown. The color of each cell indicates the magnitude of log2 fold-change and minus log10 pFDR value of the gene.

p-values across different datasets. The gene heatmap allows users to: (1) observe the pattern of the gene expression changes, (2) identify the co-expressed genes, and (3) identify the biomarkers (genes that are consistently upregulated or downregulated across multiple studies).

The function `plotDEGeneHeatmap()` *takes as input the following parameters: (1)* `DEResults`*, a list of data frames of differential analysis results; (2)* `genes`*, a vector of Entrez IDs to plot, which must be in the ID column of the data frame in* `DEResults`*; (3)* `labels`*, a vector of labels for the genes, and if not provided, the gene IDs will be used as labels; (4)* `useFDR`*, a boolean value when it is* `TRUE` *indicating using FDR adjusted p- values instead of nominal p-values; (5)* `logFCLims`*, a vector of length 2 specifying the minimum and maximum log fold change to plot; and (6)* `negLog10pValueLims`*, a vector of length 2 specifying the minimum and maximum –log10(p-value) to plot. For the labels parameter, users can use the function* `genesAnnotation()` *to obtain gene annotation, including gene symbol, chromosome vector, and full description. Users can provide the vector of Entrez IDs (for* `genes` *parameter) and get the description as shown in the code snippet.*

As can be seen in the heatmap, the regulation direction of the genes is generally consistent across the three datasets and the meta-analysis. Users can try all meta-analysis methods implemented in the package and choose the one that is most suitable for their analysis purposes.

6. Visualize a KEGG pathway with gene statistics:

```
# Select columns in the results of differential analysis:
selectedColumns <- colnames(metaDEResult)
print(selectedColumns)
# Console output
[1] "ID" "p.value" "pFDR" "logFC" "logFCSE"

# Prepare the list for plotting
DEResultsToPlot <- list(
"Affymetrix - GSE5281" = affyDEResults[, selectedColumns],
"Agilent - GSE61196" = agilDEResults[, selectedColumns],
"RNASeq - GSE153873" = RNASeqDEResults[, selectedColumns],
"Meta-analysis" = metaDEResult)

# Plot for KEGG Alzheimer's Disease pathway
pltObj <- RCPA::plotKEGGMap(DEResults = DEResultsToPlot, KEGGPathwayID = "hsa05010", stat =
"logFC", pThreshold = 1, statLimit = 1)

# Display the plot
pltObj$plot
```

Figure 22 shows the plot generated by the above code snippet. The function `plotKEGGMap()` displays the underlying KEGG pathway and shows the expression changes of the genes. This plot is useful for understanding the biological mechanisms and identifying the key genes that drive the pathway dysregulation.

The function takes as input the following parameters: (1) `DEResults`, a list of data frames containing the results from differential expression analysis; (2) `KEGGPathwayID`, a character specifying the KEGG pathway ID; (3) `statistic`, a character specifying the column name of the data frame used to plot the differentially expressed genes (DE genes); (4) `useFDR`, a boolean parameter indicating that DE genes are selected based on adjusted p-value, otherwise, nominal p-value; and (5) `statLimit`, the limit of the absolute value of the statistic. The function returns a list with the following elements: (1) `plot`, a ggplot object of the KEGG map; (2) `width`, the width of the KEGG map; and (3) `height`, the height of the KEGG map.

The above code displays the KEGG pathway map for Alzheimer's disease (KEGG ID hsa05010). It is important to note that all data frames included in the `DEResults` parameter must contain the same columns. Here the parameter contains the data frames that have the same columns in the data frame returned by `runDEMetaAnalysis()` function. In the figure, a color-coded box under each node in the pathway map is divided into multiple parts corresponding to the number of analysis results from different datasets. The color of each part in the nodes is determined by the log2 fold-change of the genes in the pathway. The parameter `pThreshold` is set to 1 to show the regulation direction of all genes in the pathway.

Pathway-level meta-analysis

Here we illustrate how users can combine pathway analysis results obtained from multiple datasets. We assume that users have performance gene set enrichment analysis or pathway analysis, as described in Basic Protocols 5 and 6. In the following, we will combine the results obtained from FGSEA across the three datasets GSE5281, GSE61196, and GSE153873.

7. If users skipped Basic Protocols 5 and 6, they could use the function `RCPA::loadData()` to load the enrichment results obtained from the method FGSEA for all three datasets:

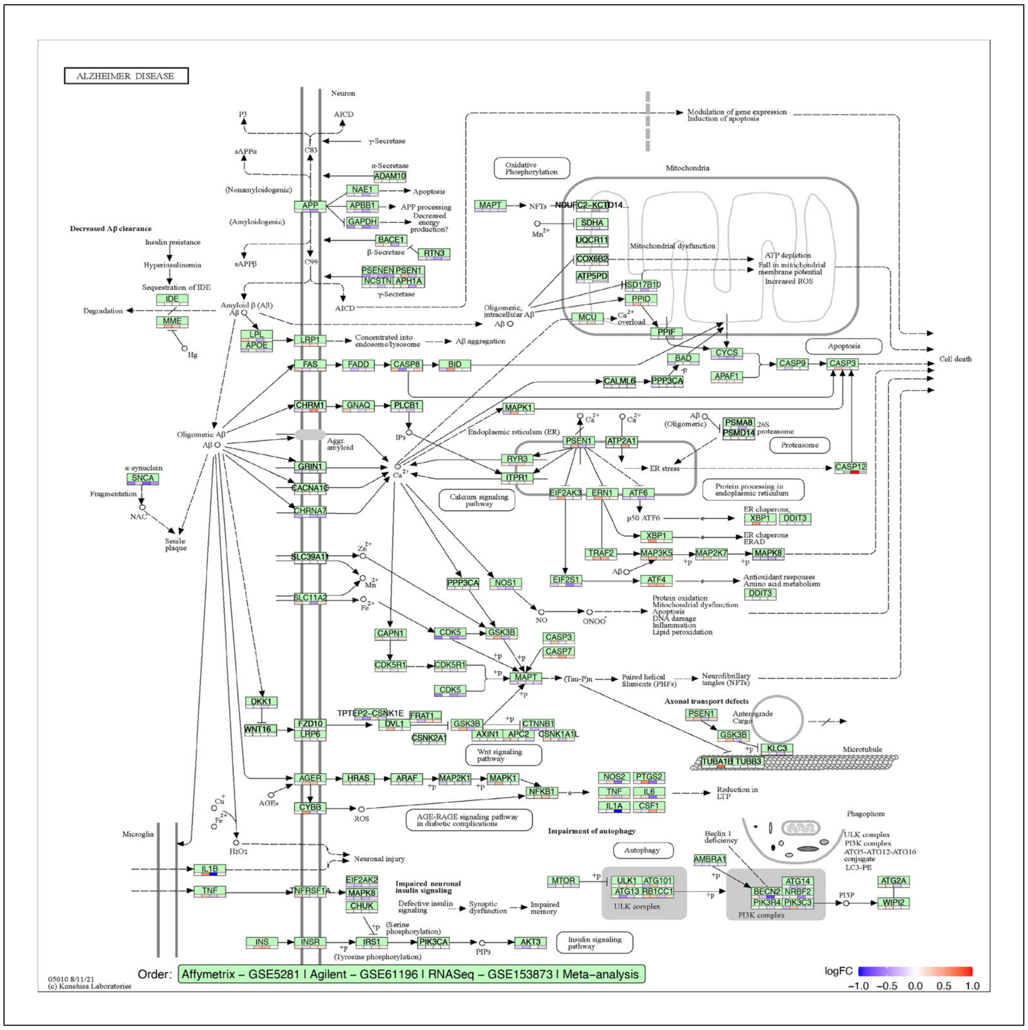


Figure 22 Displaying gene-level meta-analysis on the pathway *Alzheimer's disease* (KEGG ID *hsa05010*). The figure shows the differential analysis results from three datasets (GSE5281, GSE61196, and GSE153873), and their meta-analysis result. The green nodes within the pathway represent the genes involved in this specific pathway. Each node is accompanied by a color-coded box underneath, which is divided into multiple segments corresponding to the number of datasets. The order of the datasets/segments is shown in the green box on the bottom left. The color of each segment is determined by the log fold-change of the genes in the corresponding dataset.

```
# FGSEA analysis result from Affymetrix dataset:
affyFGSEAResult <- RCPA::loadData("affyFGSEAResult")

# FGSEA analysis result from Agilent dataset:
agilFGSEAResult <- RCPA::loadData("agilFGSEAResult")

# FGSEA analysis result from RNA-Seq dataset:
RNASeqFGSEAResult <- RCPA::loadData("RNASeqFGSEAResult")

# Load the KEGG gene sets
KEGGGenesets <- RCPA::loadData("KEGGGenesets")
```

The code snippet is used to load the results of the enrichment analysis results using FGSEA on three datasets: GSE5281, GSE61196, and GSE153873, and the KEGG gene sets obtained from Basic Protocol 5. These results are for running the functions in pathway-level meta-analysis.

8. Prepare a list of pathway analysis results from multiple datasets:

```
# Compile a list of pathway analysis results
PAResults <- list(
```

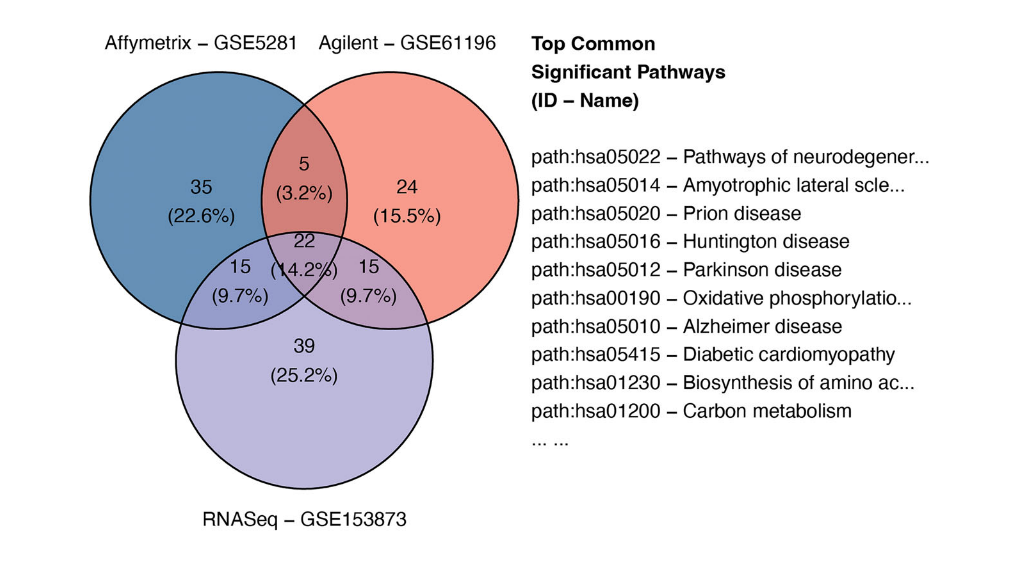

Affymetrix – GSE5281
Agilent – GSE61196
RNASeq – GSE153873
35
(22.6%)
5
(3.2%)
24
(15.5%)
22
(14.2%)
15
(9.7%)
15
(9.7%)
39
(25.2%)
Top Common
Significant Pathways
(ID – Name)
path:hsa05022 – Pathways of neurodegener...
path:hsa05014 – Amyotrophic lateral scle...
path:hsa05020 – Prion disease
path:hsa05016 – Huntington disease
path:hsa05012 – Parkinson disease
path:hsa00190 – Oxidative phosphorylatio...
path:hsa05010 – Alzheimer disease
path:hsa05415 – Diabetic cardiomyopathy
path:hsa01230 – Biosynthesis of amino ac...
path:hsa01200 – Carbon metabolism
... ...

Figure 23 Venn diagram of FGSEA results obtained from three datasets GSE5281, GSE61196, and GSE153873. The number in each section represents the number of significant pathways or gene sets. In this example, there are 22 pathways that are significant in all datasets. The right side of the figure shows the pathway IDs and names of the top 10 pathways that are significant in all three datasets.

```
"Affymetrix - GSE5281" = affyFGSEAResult,
"Agilent - GSE61196" = agilFGSEAResult,
"RNASeq - GSE153873" = RNASeqFGSEAResult)
```

Similar to gene-level meta-analysis, we first use the function `list()` *to compile the pathway analysis results. Here, we consider the enrichment analysis using FGSEA with the same settings on the three GEO datasets, as described in Basic Protocol 5. Alternatively, users can modify this list by including the results of other methods from Basic Protocols 5 and 6.*

9. Generate a Venn diagram and query common pathways:

```
# Plot venn diagram
RCPA::plotVennPathway(PAResults = PAResults, pThreshold = 0.05)

# Query a list of common pathways
commonPathways <- RCPA::getCommonPathways(PAResults = PAResults)

# Display the results:
print(commonPathways[1:5,])

# Console output:
  ID              Name
  path:hsa05022   Pathways of neurodegeneration - multiple diseases
  path:hsa05014   Amyotrophic lateral sclerosis
  path:hsa05020   Prion disease
  path:hsa05016   Huntington disease
  path:hsa05012   Parkinson disease
```

Figure 23 shows the Venn diagram obtained from the above code snippet. The function `plotVennPathway()` *requires the following parameters: (1)* `PAResults`*, a list containing pathway analysis results; (2)* `pThreshold`*, the p-value threshold to determine if a pathway is significant (0.05 by default); and (3)* `useFDR`*, a boolean parameter indicating that the adjusted p-value is used instead of the nominal p-value (*`TRUE` *by default). The Venn diagram shows the overlap of significant pathways among the three analyses. Using the Venn diagram, we can have an overview of the agreement of the list of significant pathways among different datasets.*

As we can see from the plot, using a pFDR cutoff of 0.05, there are 22 pathways that are significant in all three datasets. The right side of the figure shows the top 10 pathways that are significant in all three datasets. Users can list all common pathways using the function `getCommonPathways()`, which has the same parameters as of the function `plotVennPathway()`. This function returns a data frame containing the following columns: (1) `ID`, pathway IDs; and (2) `Name`, pathway names.

10. Perform pathway-level meta-analysis using one of the six methods:

```
# Meta-analysis using Stouffer's method
metaPAResult <- RCPA::runPathwayMetaAnalysis(PAResults = PAResults, method = "stouffer")

# Display the results:
print(metaPAResult[1:5, 1:5])

# Console output

  ID              name                   p.value     pFDR        score

  path:hsa05012   Parkinson disease      7.12e-46    2.39e-43    -2.47

  path:hsa05016   Huntington disease     3.76e-44    6.32e-42    -2.43

  path:hsa05014   Amyotrophic lateral    9.55e-44    1.07e-41    -2.39
                  sclerosis

  path:hsa00190   Oxidative              6.47e-40    5.44e-38    -2.69
                  phosphorylation

  path:hsa05020   Prion disease          1.29e-37    8.68e-36    -2.40
```

To perform meta-analysis at pathway-level, users can call the function `runPathwayMetaAnalysis()`, which requires the following parameters: (1) `PAResults`, a list of at least size two of data frames obtained from pathway analysis in Basic Protocols 5 and 6; and (2) `method`, the method to perform meta-analysis, which can be "fisher", "stouffer", "addCLT", "minP", "geoMean", or "REML".

The output of `runPathwayMetaAnalysis()` is a data frame containing the following columns: (1) `ID`, the ID of pathway; (2) `name`, the name of pathway; (3) `p.value`, the meta p-value of pathway; (4) `pFDR`, the adjusted meta p-value of pathways using Benjamini-Hochberg method; (5) `score`, the combined score of pathway; (6) `normalizedScore`, the combined normalized score of pathway; and (7) `pathwaySize`, the size of pathway. In the code snippet, we still choose Stouffer's method for p-value combination, but users are free to use other methods.

11. Generate a bar chart for pathway-level meta-analysis:

```
# Select the top 30 significant from meta-analysis
selectedPathways <- metaPAResult$ID[1:30]

# Create a list of pathway analysis results of these 30 pathways
PAResultsToPlot <- list(
"Affymetrix - GSE5281" = affyFGSEAResult,
"Agilent - GSE61196" = agilFGSEAResult,
"RNASeq - GSE153873" = RNASeqFGSEAResult,
"Meta-analysis" = metaPAResult)

# Plot bar chart
RCPA::plotBarChart(results = PAResultsToPlot, selectedPathways = selectedPathways) +
ggplot2::ggtitle("FGSEA Analysis Results")
```

Figure 24 shows the bar chart generated from the above code snippet. The figure displays the magnitude and direction of the enrichment scores. Users can compare the impact of the condition (Alzheimer's disease) on pathways across all three datasets, as well as meta-analysis results. The function `plotBarChart()` takes as input the following parameters: (1) `results`, a named list of results obtained from multiple datasets using one of the eight pathway analysis methods described in Basic Protocols 5 and 6; (2) `limit`, the maximum number of pathways to plot (Infinity by default); (3) `label`, the column to use for the labels; (4) `by`, the column to use for the bar heights; (5)

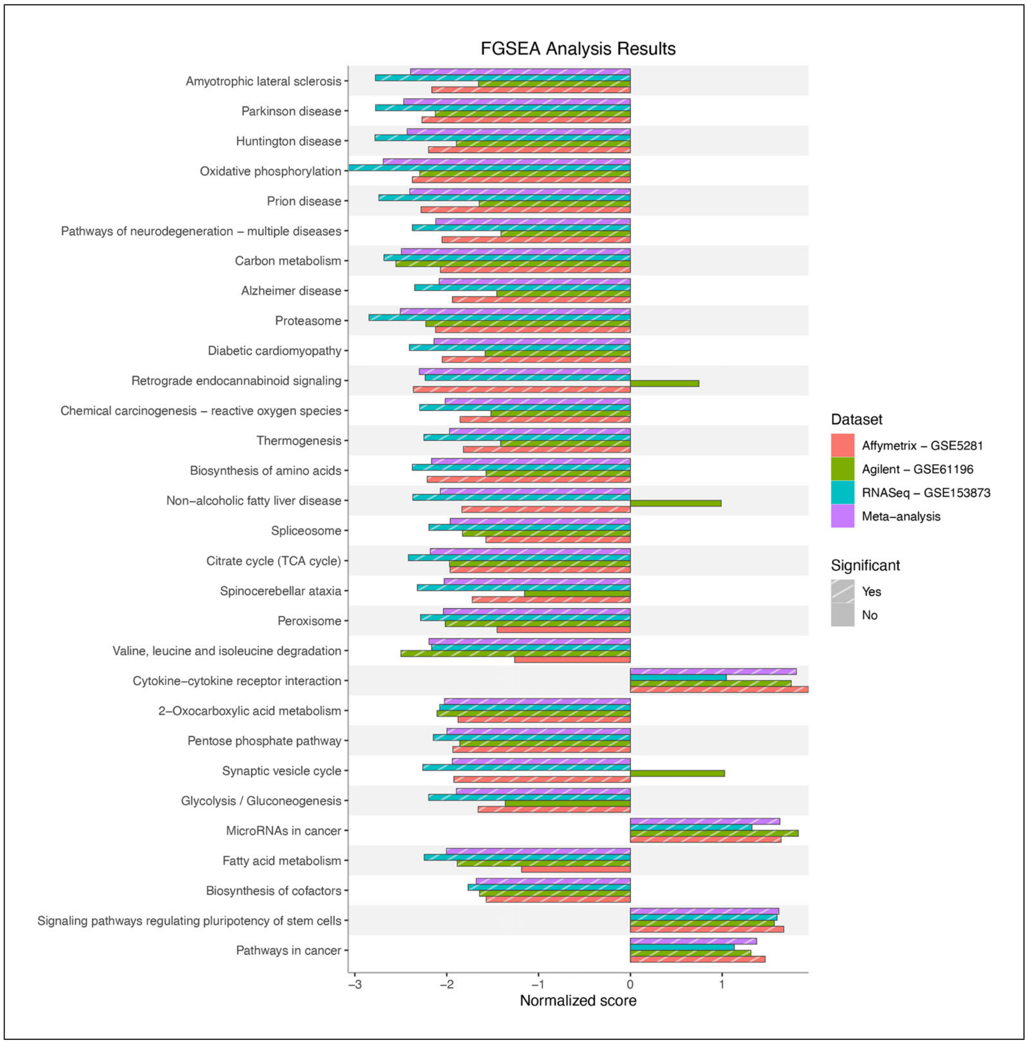


Figure 24 Pathway bar chart obtained from FGSEA results for three datasets (GSE5281, GSE61196, and GSE153873) and their meta- analysis. The x-axis shows the normalized scores while the y-axis shows the pathway names. Only the top 30 KEGG pathways from meta-analysis are shown. The striped bars represent significant pathways. We can see that the top significant pathways are mostly downregulated in all three datasets and the results from individual datasets universally agree with each other.

`maxNegLog10PValue`, the maximum value of minus log10 p-value to plot (5 by default); (6) `pThreshold`, the p-value threshold to determine if a pathway is significant (0.05 by default); (7) `useFDR`, a boolean parameter indicating that the adjusted p-value is used instead of the nominal p-value (`TRUE` by default); and (8) `selectedPathways`, the IDs of pathways to be shown in the plot (`NULL` by default).

In the above code, the first two command lines are for limiting the number of pathways to plot, in which we only plot the top 30 KEGG pathways on the output returned by `runPathwayMetaAnalysis()` function. In the plot, the stripes in the bars indicate if the pathway is significant in the dataset. We can see that the top significant pathways are mostly downregulated in all three datasets and the results from individual datasets mostly agree with each other. On the other hand, there are three pathways found to be significant in Affymetrix and RNA-Seq datasets but not in the Agilent dataset.

12. Generate a pathway heatmap for meta-analysis:

```
RCPA::plotPathwayHeatmap(resultsList = PAResultsToPlot, yAxis = "name", selectedPathways
 = selectedPathways)
```

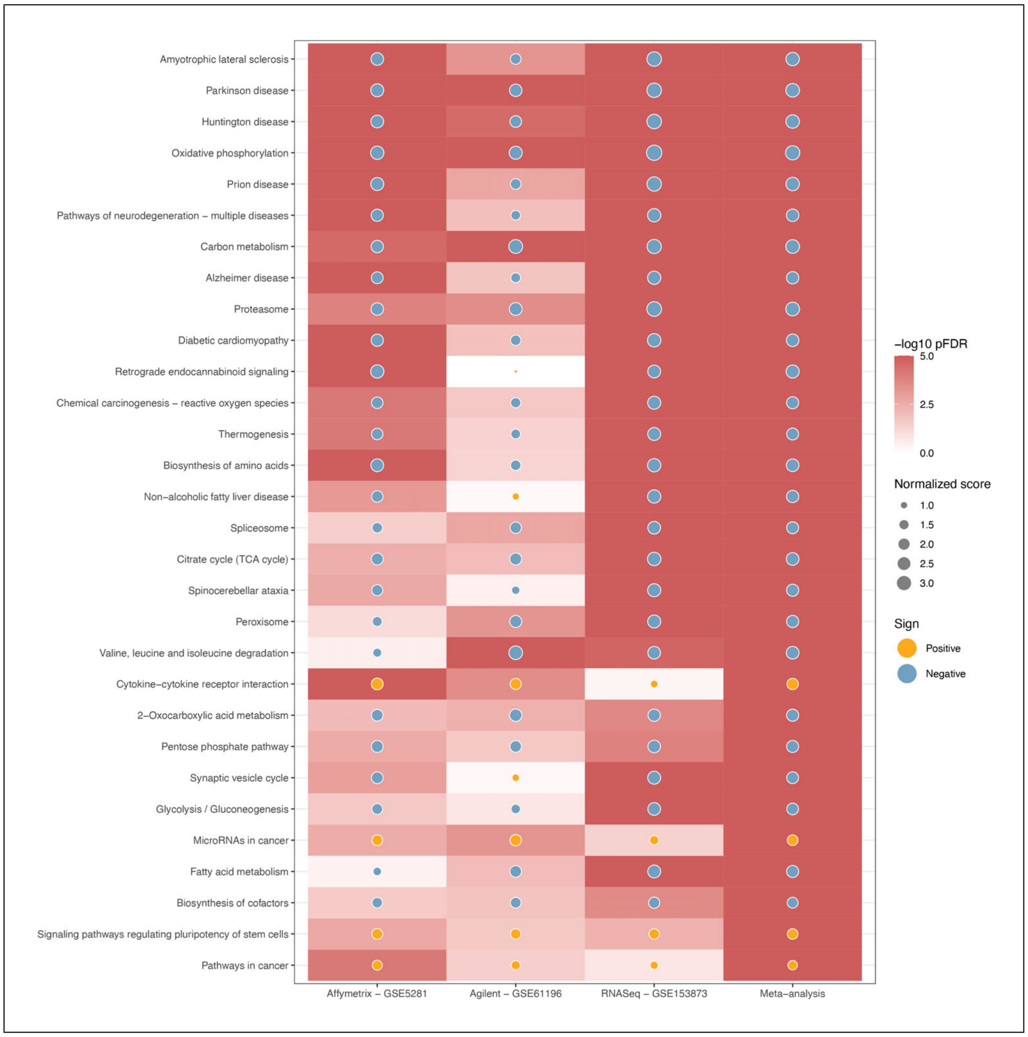


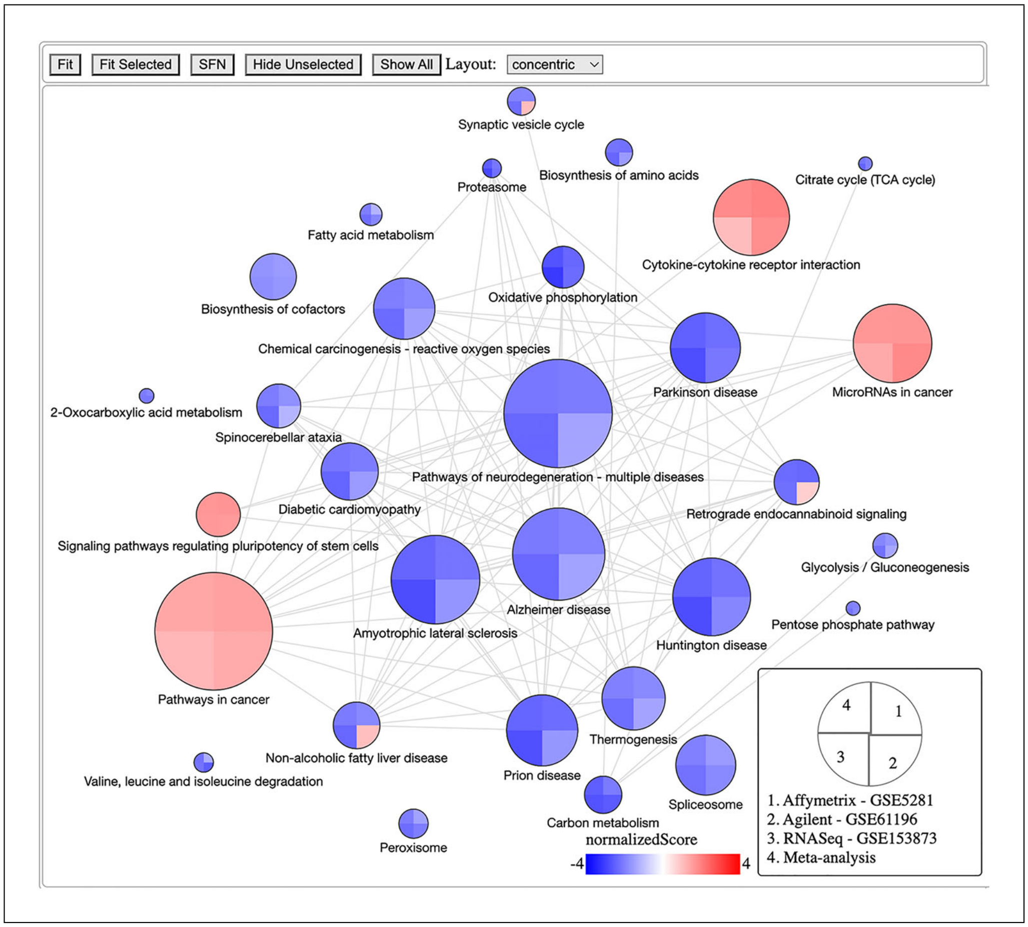
Figure 25 Pathway heatmap obtained from FGSEA results for three datasets (GSE5281, GSE61196, and GSE153873) and their meta-analysis. The x-axis shows the study names while the y-axis shows the pathway names. Only the top 30 KEGG pathways from meta-analysis are shown. In each cell, the background color reflects the magnitude of –log10 pFDR while the size of the circle represents the magnitude of the absolute normalized score. The color of each circle shows the direction of pathway regulation, which can be either up or down.

Figure 25 shows the pathway heatmap generated by the function `plotPathwayHeatmap()`. This type of plot usually shows three pieces of information for each pathway within the analysis: (i) the magnitude of the enrichment score, (ii) the regulation direction, and (iii) the significance of the enrichment score. The function requires the following inputs: (1) `resultsList`, a named list of data frames from pathway analysis; (2) `yAxis`, a character parameter specifying which column of the result data frame from pathway analysis is used to label to y-axis; and (3) `selectedPathways`, the IDs of pathways to be shown in the plot (`NULL` by default). The `plotPathwayHeatmap()` also returns a `ggplot` object as other plot functions that have been introduced thus far.

In this example, we still use the variable `resultsList` containing the pathway analysis results for top 30 KEGG pathways from meta-analysis as described in the previous step and pass it to the function `plotPathwayHeatmap()` as `resultsList` parameter and specify the `yAxis = "name"` as we use the `name` column from the pathway analysis result to label the y-axis.

13. Generate a pathway network for meta-analysis:

```
# Select the top 30 significant from meta-analysis
selectedPathways <- metaPAResult$ID[1:30]
```



Fit
Fit Selected
SFN
Hide Unselected
Show All
Layout:
concentric
Synaptic vesicle cycle
Proteasome
Biosynthesis of amino acids
Citrate cycle (TCA cycle)
Fatty acid metabolism
Cytokine-cytokine receptor interaction
Biosynthesis of cofactors
Oxidative phosphorylation
Chemical carcinogenesis - reactive oxygen species
Parkinson disease
MicroRNAs in cancer
2-Oxocarboxylic acid metabolism
Spinocerebellar ataxia
Pathways of neurodegeneration - multiple diseases
Retrograde endocannabinoid signaling
Diabetic cardiomyopathy
Signaling pathways regulating pluripotency of stem cells
Glycolysis / Gluconeogenesis
Alzheimer disease
Pentose phosphate pathway
Amyotrophic lateral sclerosis
Huntington disease
Pathways in cancer
Thermogenesis
Non-alcoholic fatty liver disease
Prion disease
Valine, leucine and isoleucine degradation
Spliceosome
Carbon metabolism
Peroxisome
normalizedScore
-4
4
4
1
3
2
1. Affymetrix - GSE5281
2. Agilent - GSE61196
3. RNASeq - GSE153873
4. Meta-analysis

Figure 26 Pathway network obtained from FGSEA results for three datasets (GSE5281, GSE61196, and GSE153873) and their meta-analysis. Each node represents a pathway and is further divided into four parts corresponding to the four analyses (GSE5281, GSE61196, GSE153873, and meta-analysis). The color of each part shows the direction and magnitude of the normalized score from the corresponding analysis result. Two pathways are connected if they share at least 75% genes of the smaller pathway. The figure shows the top 30 most significant pathways from the meta-analysis. The Alzheimer's disease pathway is consistently impacted in all three datasets.

```
# Create a list of pathway analysis results of these 30 pathways
allPAResultsToPlot <- list(
"Affymetrix - GSE5281" = affyFGSEAResult,
"Agilent - GSE61196" = agilFGSEAResult,
"RNASeq - GSE153873" = RNASeqFGSEAResult,
"Meta-analysis" = metaPAResult)

# Plot pathway network
pltHtml <- RCPA::plotPathwayNetwork(
PAResults = allPAResultsToPlot,
genesets = KEGGgenesets,
selectedPathways = selectedPathways,
statistic = "normalizedScore",
mode = "continuous",
edgeThreshold = 0.75)
```

Figure 26 shows the pathway network generated for the top 30 significant pathways from the meta-analysis using the function `plotPathwayNetwork()`*. The size of the nodes is proportional to the number of genes in the pathway. Two pathways are connected if they share at least 75% genes of the smaller pathway. Each node is divided into multiple parts corresponding to the number of datasets. The color of each part is determined by the normalized enrichment score. Users can also choose to use the p-value to determine the* *color of the nodes instead of the normalized enrichment score, by replacing* `statistic = "p.value"` *in the above code.*

Pathway-level consensus analysis

The main objective of consensus pathway analysis is to allow users to see the differences, as well as the consensus results across many methods that rely on distinctively different hypotheses. As a result, it helps life scientists who are trying to understand the underlying biological mechanisms of a condition or disease of interest. Consensus analysis can also be extended to compare multiple experiments and computational methods so that users can compare different hypotheses, experimental designs, and technologies. Below we illustrate the application of consensus pathway analysis to combine the pathway analysis results obtained from three methods and three datasets.

14. If users skipped the previous protocols for pathway analysis, they could use `RCPA::loadData()` to load the data:

```
# Wilcoxon test results for RNA-Seq dataset:
RNASeqWilcoxResult <- RCPA::loadData("RNASeqWilcoxResult")

# FGSEA results for RNA-Seq dataset:
RNASeqFGSEAResult <- RCPA::loadData("RNASeqFGSEAResult")

# SPIA results for RNA-Seq dataset:
RNASeqSPIAResult <- RCPA::loadData("RNASeqSPIAResult")

# Load the KEGG gene sets
KEGGGenesets <- RCPA::loadData("KEGGGenesets")
```

The code snippet is used for loading the pre-saved results of the pathway analysis results using the Wilcox test, FGSEA and SPIA on the RNA-Seq dataset GSE153873 obtained from Basic Protocols 5 and 6, and the KEGG gene sets obtained from Basic Protocol 5. These results are for running the functions in pathway-level consensus analysis.

15. Prepare the input list of analysis results for consensus analysis:

```
# Prepare a list of results obtained from the Wilcox test, FGSEA, and SPIA
selectedRNASeqPAResults <- list(
"Wilcox" = RNASeqWilcoxResult,
"FGSEA" = RNASeqFGSEAResult,
"SPIA" = RNASeqSPIAResult)
```

Similar to meta-analysis, we first need to prepare the list containing the pathway analysis results from different methods. In this example, we used the results from pathway analysis for RNA-Seq dataset GSE153873 using Wilcoxon test, FGSEA, and SPIA, as described in Basic Protocols 5 and 6.

16. Generate a Venn diagram for the three analyses:

```
# Plot Venn diagram
RCPA::plotVennPathway(PAResults = selectedRNASeqPAResults, pThreshold = 0.05)

# Query a list of common pathways from Wilcox Test, FGSEA, SPIA:
commonPathways <- RCPA::getCommonPathways(PAResults = selectedRNASeqPAResults)

# Display the result
print(commonPathways[1:5,])

# Console output
ID                 Name

path:hsa05016      Huntington disease

path:hsa05014      Amyotrophic lateral sclerosis

path:hsa05022      Pathways of neurodegeneration - multiple diseases

path:hsa00190      Oxidative phosphorylation

path:hsa05020      Prion disease
```

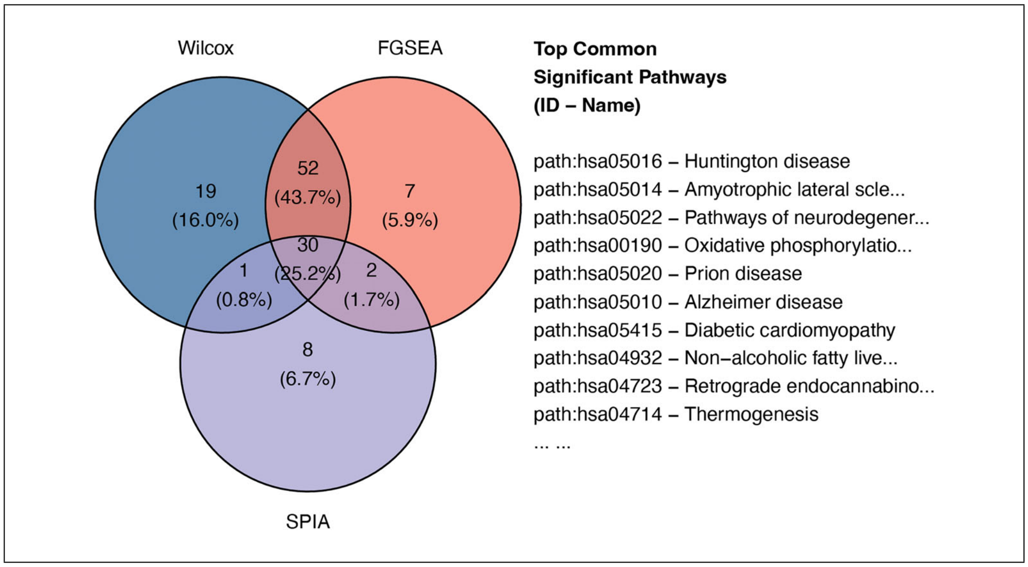


Figure 27 Venn diagram obtained from pathway analysis of the RNA-Seq dataset GSE153873 using three methods: Wilcoxon test, FGSEA, and SPIA. The number in each region represents the number of significant pathways. The right side of the figure shows the top 10 pathways that are significant using any of the three analysis methods.

Figure 27 shows the Venn diagram generated from the above code snippet. The figure shows the common significant pathways obtained from three different methods Wilcoxon test, FGSEA, and SPIA for the same dataset GSE153873. Users can list all common pathways using the function `getCommonPathways()`, which has the same parameters as of the function `plotVennPathway()`. This function returns a data frame containing the following columns: (1) `ID`, pathway IDs; and (2) `Name`, pathway names.

17. Perform consensus analysis using the weightedZMean method:

```
# Set seed to create reproducible result:
set.seed(1)

# Run consensus analysis
consensusWZRNASeqPAResult <- RCPA::runConsensusAnalysis(PAResults =
selectedRNASeqPAResults, method = "weightedZMean")

# Display the result
print(consensusWZRNASeqPAResult[1:6, c("ID", "p.value", "pFDR", "name")])

# Console output
ID              p.value     pFDR        name
path:hsa05014   7.66e-31    7.66e-31    Amyotrophic lateral sclerosis
path:hsa05016   1.57e-30    1.57e-30    Huntington disease
path:hsa00190   5.28e-25    5.28e-25    Oxidative phosphorylation
path:hsa05022   5.86e-25    5.86e-25    Pathways of neurodegeneration - multiple diseases
path:hsa05020   9.00e-25    9.00e-25    Prion disease
path:hsa05010   9.67e-22    9.67e-22    Alzheimer disease
```

The above snippet demonstrates the use of `runConsensusAnalysis()` for consensus analysis using weightedZMean method. This method calculates the weighted average of z-values for each pathway, allowing users to assign different weights to each method according to their research needs and assumptions. In this case, the function requires the following parameters: (1) `PAResults`, a list of at least size two of data frames obtained from pathway analysis in Basic Protocols 5 and 6; (2) `method`, a character specifying the method to perform consensus analysis, which is "weightedZMean" in this case; (3) `useFDR`, a logical parameter indicating if adjusted p-values should be used (`TRUE` by default); (4) `weightsList`, a vector of integer values specifying weights for each individual pathway analysis results (`NULL` by default, meaning that all weights are equal);

and (4) `backgroundSpace`*, a list of lists with the same length as* `PAResults`*. Each list contains underlying space (set of pathways) for each input data frame in* `PAResults`*. The last parameter is optional. If it is set to be* `NULL`*, it means all input data frames share a common space, which is the union of pathways of the input data frames.*

The function returns a data frame with the following columns: (1) `ID`*, the ID of pathway; (2)* `p.value`*, the consensus p-value of pathway; (3)* `pFDR`*, the consensus adjusted p-value of pathways using Benjamini-Hochberg method; (4)* `name`*, the name of pathway; and (5)* `pathwaySize`*, the size of pathway.*

18. Perform consensus analysis using robust rank aggregation:

```
# Set seed to create reproducible result:
set.seed(1)

# Run consensus analysis
consensusRRARNASeqPAResult <- RCPA::runConsensusAnalysis(PAResults =
selectedRNASeqPAResults, method = "RRA", rank.by = "both")

# Display the result
print(consensusRRARNASeqPAResult[1:5, c("ID", "p.value", "pFDR", "name")])

# Console output
ID             p.value     pFDR        name
path:hsa00190  1.45e-05    0.00514     Oxidative phosphorylation
path:hsa00650  2.26e-04    0.04017     Butanoate metabolism
path:hsa00280  1.32e-03    0.15618     Valine, leucine and isoleucine degradation
path:hsa03050  1.77e-03    0.15696     Proteasome
path:hsa00020  4.50e-03    0.31544     Citrate cycle (TCA cycle)
```

In addition to weightedZMean method, we also include the implementation of robust rank aggregation (RRA) method. The RRA method relies on rank aggregation principles, for which we utilize the function provided by the RobustRankAggreg package (Kolde et al., 2012). To execute the RRA method, `runConsensusAnalysis()` *requires the following parameters: (1)* `PAResults`*, a list of at least size two of data frames obtained from pathway analysis in Basic Protocols 5 and 6; (2)* `method`*, a character specifying the method to perform consensus-analysis, which is “RRA”; and (3)* `rank.by`*, a character parameter that specifies how the input results should be ranked, which can be by either “normalizedScore” (default), “pFDR”, or “both” (in this example). By following the code, users will obtain a data frame shown in the console output.*

19. Generate pathway network for consensus analysis:

```
# Create a list of results to plot:
selectedCCRNASeqPAResults <- list(
"Wilcox" = RNASeqWilcoxResult,
"FGSEA" = RNASeqFGSEAResult,
"SPIA" = RNASeqSPIAResult,
"Consensus - weightedZMean" = consensusWZRNASeqPAResult
)

# Select to plot top 30 most significant pathways
# from consensus analysis using weightedZMean:
selectedPathways <- consensusWZRNASeqPAResult$ID[1:30]

# Plot pathway network
pltHtml <- RCPA::plotPathwayNetwork(
PAResults = selectedCCRNASeqPAResults,
genesets = KEGGGenesets,
selectedPathways = selectedPathways,
statistic = "p.value",
mode = "discrete",
pThreshold = 0.05,
edgeThreshold = 0.75)
```

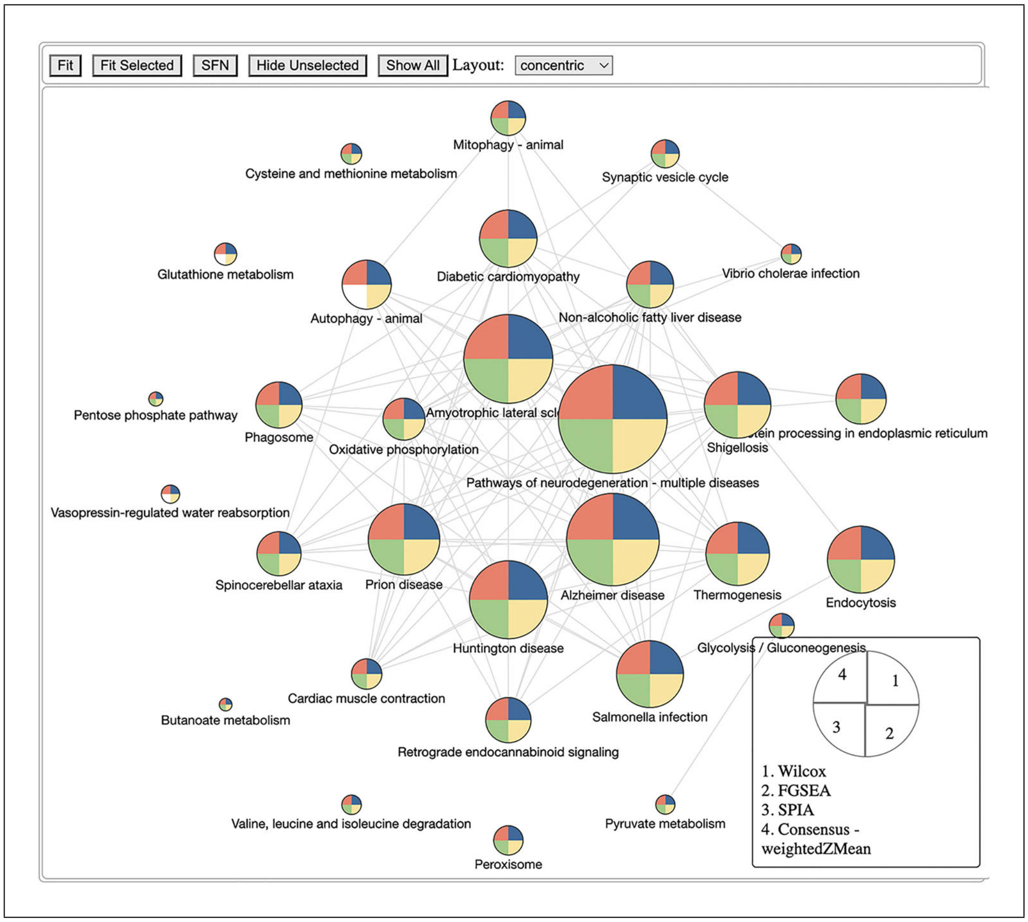

Fit
Fit Selected
SFN
Hide Unselected
Show All
Layout:
concentric
Mitophagy - animal
Cysteine and methionine metabolism
Synaptic vesicle cycle
Glutathione metabolism
Diabetic cardiomyopathy
Vibrio cholerae infection
Autophagy - animal
Non-alcoholic fatty liver disease
Pentose phosphate pathway
Phagosome
Amyotrophic lateral scl
Oxidative phosphorylation
tein processing in endoplasmic reticulum
Shigellosis
Pathways of neurodegeneration - multiple diseases
Vasopressin-regulated water reabsorption
Spinocerebellar ataxia
Prion disease
Alzheimer disease
Thermogenesis
Endocytosis
Huntington disease
Glycolysis / Gluconeogenesis
Cardiac muscle contraction
Butanoate metabolism
Salmonella infection
Retrograde endocannabinoid signaling
Valine, leucine and isoleucine degradation
Pyruvate metabolism
Peroxisome
4
1
3
2
1. Wilcox
2. FGSEA
3. SPIA
4. Consensus - weightedZMean

Figure 28 Pathway network obtained from the analysis of the RNA-Seq dataset GSE153873 using three methods (Wilcox, FGSEA, and SPIA) and consensus analysis (weightedZMean method). The figure shows the top 30 most significant pathways obtained from the consensus analysis using weightedZMean. In the graph, each node represents a pathway. Two pathways are connected if they share at least 75% genes of the smaller pathway. In this case, the node is divided into four parts that correspond to the four analyses. A part is only colored when the pathway is significant in the respective analysis. For example, the pathway *Autography - animal* is not colored in the third part because it is not significant in SPIA analysis.

Figure 28 shows the pathway network obtained from the above code. The pathway network is generated from the analysis results using Wilcox, FGSEA, SPIA, and the consensus analysis. In this example, only the top 30 most significant pathways from the consensus analysis of the three results are displayed, utilizing the weightedZMean method (as outlined in step 3). Note that each analysis method has a different definition of enrichment score. Therefore, it is not meaningful to display their enrichment score in this example. Instead, we color the nodes (pathways) based on the significance of the pathway in the respective analysis.

20. Perform consensus analysis of multiple methods and datasets:

```
# Load the necessary data if users skipped the previous protocols
# Enrichment results using Wilcoxon test:
affyWilcoxResult <- RCPA::loadData("affyWilcoxResult")
RNASeqWilcoxResult <- RCPA::loadData("RNASeqWilcoxResult")
agilWilcoxResult <- RCPA::loadData("agilWilcoxResult")

# Enrichment results FGSEA:
affyFGSEAResult <- RCPA::loadData("affyFGSEAResult")
agilFGSEAResult <- RCPA::loadData("agilFGSEAResult")
RNASeqFGSEAResult <- RCPA::loadData("RNASeqFGSEAResult")
```

```
# TB analysis using SPIA:
affySPIAResult <- RCPA::loadData("affySPIAResult")
agilSPIAResult <- RCPA::loadData("agilSPIAResult")
RNASeqSPIAResult <- RCPA::loadData("RNASeqSPIAResult")

# Load the KEGG gene sets
KEGGGenesets <- RCPA::loadData("KEGGGenesets")

# Create a list of results from multiple pathway analysis methods and datasets:
selectedPAResults <- list(
"Affymetrix - Wilcox" = affyWilcoxResult,
"Affymetrix - FGSEA" = affyFGSEAResult,
"Affymetrix - SPIA" = affySPIAResult,
"Agilent - Wilcox" = agilWilcoxResult,
"Agilent - FGSEA" = agilFGSEAResult,
"Agilent - SPIA" = agilSPIAResult,
"RNASeq - Wilcox" = RNASeqWilcoxResult,
"RNASeq - FGSEA" = RNASeqFGSEAResult,
"RNASeq - SPIA" = RNASeqSPIAResult)

# Run consensus analysis using weightedZMean on selectedPAResults:
consensusPAResult <- RCPA::runConsensusAnalysis(PAResults = selectedPAResults, method =
   "weightedZMean")

# display the results
print(consensusPAResult[1:6, c("ID", "p.value", "pFDR", "name")])

# Console output:
```

```
ID                p.value        pFDR           name
path:hsa05016     7.84e-12       7.84e-12       Huntington disease
path:hsa05014     1.99e-11       1.99e-11       Amyotrophic lateral sclerosis
path:hsa05022     2.17e-10       2.17e-10       Pathways of neurodegeneration -
                                                multiple diseases
path:hsa00190     1.30e-09       1.30e-09       Oxidative phosphorylation
path:hsa05020     2.47e-09       2.47e-09       Prion disease
path:hsa05010     3.32e-08       3.32e-08       Alzheimer disease
```

The above snippet created a list of nine analyses obtained from three pathway analysis methods and three datasets. Similar to the previous steps, users can apply the function `runConsensusAnalysis()` to perform consensus analysis using any of the two integration methods (weightedZMean and robust rank aggregation). After the integration, users can visualize the results as shown in the following steps.

```
# Select the top 20 significant from consensus analysis
selectedPathways <- consensusPAResult$ID[1:20]

# Create a list of results to plot:
selectedPAResultsToPlot <- list(
"Affymetrix - Wilcox" = affyWilcoxResult,
"Affymetrix - FGSEA" = affyFGSEAResult,
"Affymetrix - SPIA" = affySPIAResult,
"Agilent - Wilcox" = agilWilcoxResult,
"Agilent - FGSEA" = agilFGSEAResult,
"Agilent - SPIA" = agilSPIAResult,
"RNASeq - Wilcox" = RNASeqWilcoxResult,
"RNASeq - FGSEA" = RNASeqFGSEAResult,
"RNASeq - SPIA" = RNASeqSPIAResult,
"Consensus Analaysis" = consensusPAResult)

# Plot pathway network
pltHtml <- RCPA::plotPathwayNetwork(
PAResults = selectedPAResultsToPlot,
```

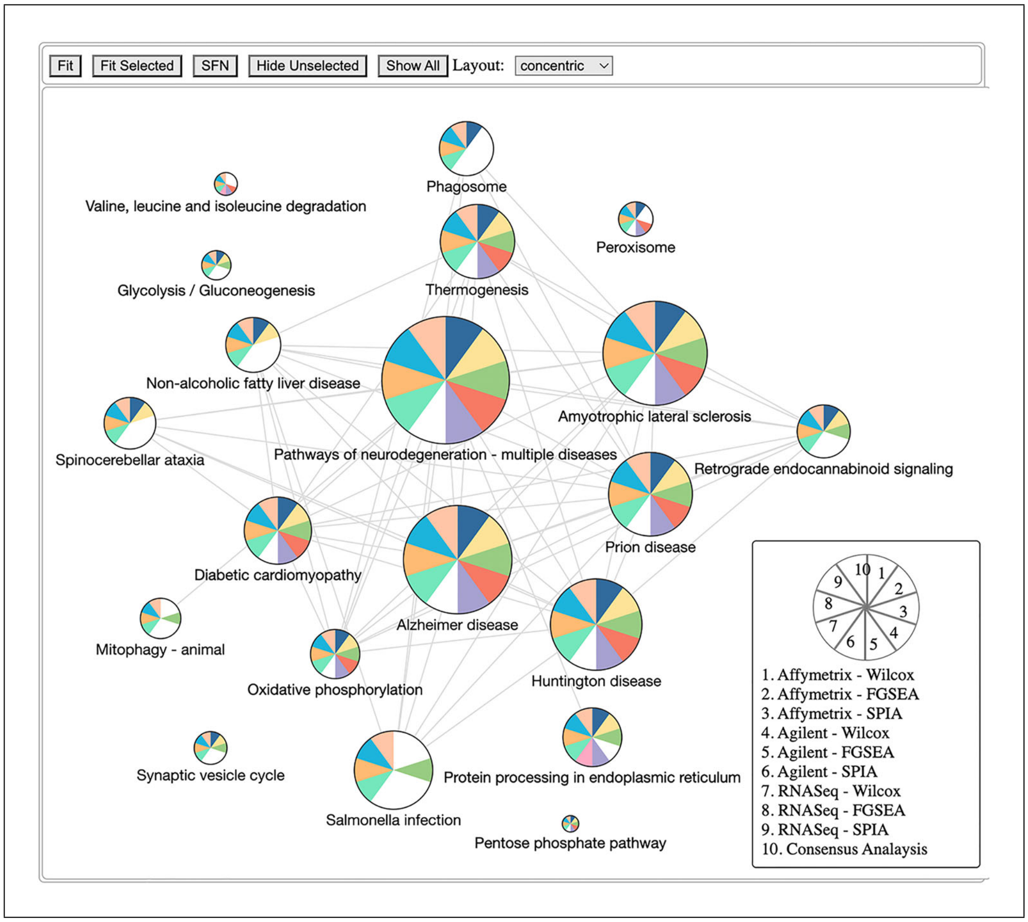


Figure 29 Pathway network obtained from nine different analyses, using three pathway methods (Wilcox test, FGSEA, and SPIA) on three datasets (GSE5281, GSE61196 and GSE153873). In this plot, only the top 20 most significantly impacted pathways from the consensus analysis on these 9 results are displayed. Each node represents a pathway and is further divided into 10 parts corresponding to the 9 analyses and the consensus analysis. The color of each part shows the direction and magnitude of the normalized score from the corresponding analysis result. The width of the edges is proportional to the number of genes shared by the two pathways. Users can change the layout of the graph into the following styles: breadthfirst, circle, cola, concentric, cose, cose-bilkent, dagre, grid, and random.

```
genesets = KEGGGenesets,
selectedPathways = selectedPathways,
statistic = "p.value",
mode = "discrete",
edgeThreshold = 0.75)
```

Figure 29 shows the pathway network graph obtained from the above code. Users can perform consensus analysis on many different pathway analysis results, which can be obtained from different methods and/or different datasets, by following the same procedure as shown in the above code. Here, we performed gene set and pathway analysis using three methods: Wilcox test, FGSEA and SPIA on three GEO datasets. In return, we obtained 9 different pathway analysis results in total. We then perform consensus analysis on these results using weightedZMean. Next, we also use the `plotPathwayNetwork()` *to generate the pathway network for consensus pathway analysis results. The figure shows the top 20 most significant pathways from consensus analysis.*

```
RCPA::plotPathwayHeatmap(resultsList = selectedPAResultsToPlot, yAxis = "name",
selectedPathways = selectedPathways)
```

Figure 30 shows the pathway heatmap obtained from the function `plotPathwayHeatmap()` *to plot the pathway heatmap for the 9 pathway analysis results and their consensus analysis. The function requires the following inputs: (1)* `resultsList`*, a*

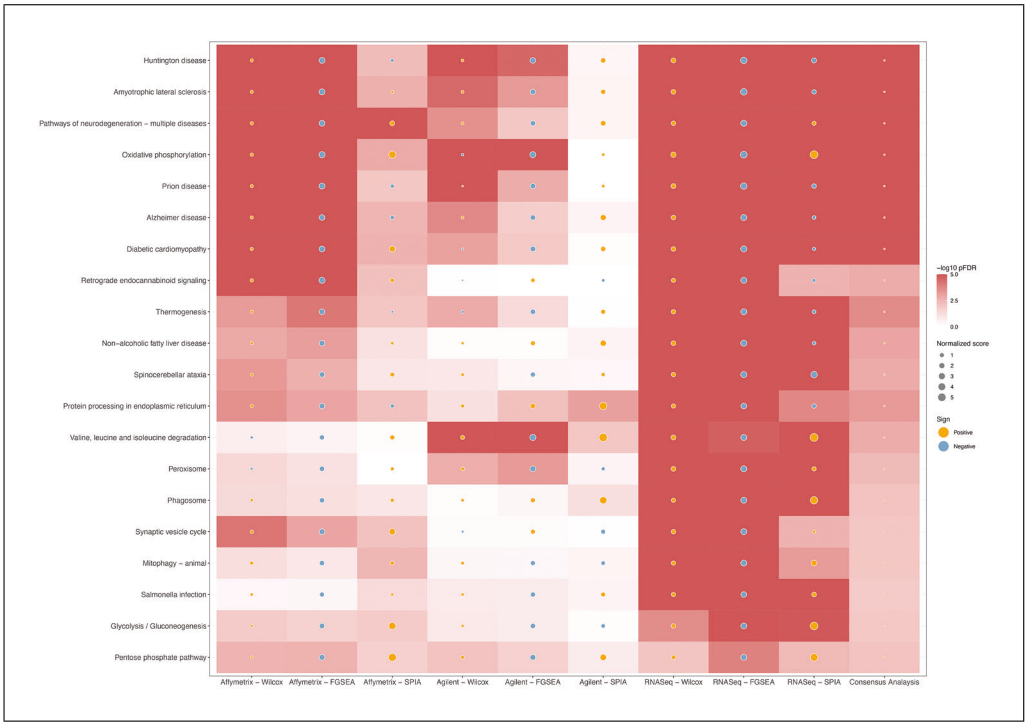


Figure 30 Pathway heatmap obtained from nine different analyses, using three pathway methods (Wilcox test, FGSEA, and SPIA) on three datasets (GSE5281, GSE61196 and GSE153873) and their consensus analysis. The x-axis shows the study names while the y-axis shows the pathway names. Only the top 20 KEGG pathways from consensus analysis are shown. In each cell, the background color reflects the magnitude of –log10 pFDR while the size of the circle represents the magnitude of the absolute normalized score. The color of each circle shows the direction of pathway regulation, which can be either up or down.

named list of data frames from pathway analysis; (2) `yAxis`*, a character parameter specifying which column of the result data frame from pathway analysis is used to label to y-axis; and (3)* `selectedPathways`*, the IDs of pathways to be shown in the plot (*`NULL` *by default). The* `plotPathwayHeatmap()` *also returns a* `ggplot` *object as other plot functions that have been introduced thus far. Note that since each analysis method has its own distinct definition of an enrichment score, we do not combine the scores across the methods in consensus analysis.*

COMMENTARY

Background Information

In this article, we present a comprehensive and user-friendly pipeline implemented in the RCPA package for the complete analysis of gene expression data using various statistical and computational methods. The package includes many functions that allow users to perform: (1) data processing for microarray and RNA-Seq data NCBI GEO; (2) differential analysis using limma, edgeR, and DESeq2; (3) gene set enrichment analysis using the KS test, Wilcox test, ORA, FGSEA, and GSA; (4) topology-based pathway analysis using SPIA, CePa ORA, and CePa GSA; (5) meta-analysis using Fisher's method, Stouffer's method, geometric mean, minP, addCLT, and REML; (6) consensus analysis using weightedZMean and RRA methods; and (7) visualization of analysis results.

We illustrate the applications of those functions on several public datasets, showing its ability to identify differentially expressed genes, significant gene sets, impacted pathways, and consensus results obtained from meta-analysis and consensus analysis. The implemented functions also offer the flexibility to adjust the analysis parameters, making them useful for researchers with diverse experimental designs and objectives. While our package provides a comprehensive solution for gene expression analysis, we acknowledge that there is always room for improvement and extension. Future directions for our work could include incorporating additional

Table 2 Troubleshooting Guide for RCPA

Problem	Possible cause	Solution
Error when installing RCPA	Missing dependencies	RCPA includes some other R packages as dependencies; it is likely that system libraries are missing causing some R packages to fail to install; users need to carefully check the error shown in their console and try to install the missing libraries
`downloadGEO()` function returns empty data frame	No supplementary files are available	Some GEO datasets do not have supplementary files uploaded; in this case, users need to check the dataset on NCBI GEO to see if the supplementary files are available; if not, users can manually download the preprocessed data and create the `SummarizedExperiment` object.
`makeContrasts()` function returns name error	Special characters in column names	The `makeContrasts()` function in the limma package requires the column names to be valid R variable names; in this case, users can remove special characters from the column names and try again
Large fold-change and statistic in the output of limma	The input data is not log-transformed	The limma package requires log-transformed data as input
GSA returns large scores	The input data is not log-transformed	GSA requires log-transformed data as input
CePa cannot find samples for analysis	The contrast matrix has >2 conditions	CePa only supports comparative analysis between two conditions; the contrast matrix should only contain two columns, e.g., “normal” and “disease”
The text in the plots is too small	The default font size is too small for the plot size	Plots are generated using the ggplot2 package; the plot object can be freely modified using functions from the ggplot2 package
The saved plots are different from the plots shown in the R	The graphics device is not set properly	Users can apply the function `ggplot2::ggsave()` to save the plots instead of using the save button from their IDE

methods and pathway databases for pathway analysis to enhance the pipeline’s accuracy and efficiency.

In conclusion, we believe that our package will be a valuable resource for life scientists and practitioners. We hope that the package will facilitate the discovery of new insights into the molecular mechanisms underlying biological processes and diseases. We plan to continue improving and updating our package in response to emerging challenges and opportunities in molecular data analysis.

Critical Parameters

Users need to ensure that the experimental design is appropriate and accounts for any confounding variables, such as age, gender, or treatment duration, etc. The design matrix used in the analysis should accurately reflect the study design and avoid any linear dependencies. Furthermore, they should choose appropriate statistical thresholds to identify differentially expressed genes and/or enriched/impacted pathways/gene sets, such as adjusted *p*-values or false discovery rate (FDR) cutoffs. These thresholds should balance the tradeoff between sensitivity and specificity.

Troubleshooting

Table 2 provides a list of common problems that users may encounter when using our pipeline, along with possible solutions. For other potential problems, users can contact the maintainer of the package at *https://cran.r-project.org/package=RCPA* (see Internet Resources).

Acknowledgments

This work was partially supported by the NSF (grant no. 2343019 and 2203236), NASA (grant no. 80NSSC22M0255, subaward 23-42), NIH NIGMS (grant no.

1R44GM152152-01), and NIH NCI (grant no. 1U01CA274573-01A1). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any of the funding agencies.

Author Contributions

Hung Nguyen: Formal analysis; methodology; software; validation; writing original draft; writing review and editing. **Ha Nguyen:** Methodology; software; writing review and editing. **Zeynab Maghsoudi:** Methodology; software; writing original draft. **Bang Tran:** Data curation; methodology. **Sorin Draghici:** Conceptualization; writing review and editing. **Tin Nguyen:** Conceptualization; methodology; project administration; supervision; writing original draft; writing review and editing

Conflict of Interest

The authors declare no conflict of interest.

Data Availability Statement

The data used in this pipeline can be found at NCBI GEO with accession IDs: GSE5281, GSE61196, and GSE153873.

Literature Cited

Balduzzi, S., Rücker, G., & Schwarzer, G. (2019). How to perform a meta-analysis with R: A practical tutorial. *BMJ Mental Health*, *22*(4), 153–160. https://doi.org/10.1136/ebmental-2019-300117

Beissbarth, T., & Speed, T. P. (2004). GOstat: Find statistically overrepresented Gene Ontologies within a group of genes. *Bioinformatics*, *20*(9), 1464–1465. https://doi.org/10.1093/bioinformatics/bth088

Benjamini, Y., & Hochberg, Y. (1995). Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B (Methodological)*, *57*(1), 289–300.

Bolstad, B. M., Irizarry, R. A., Åstrand, M., & Speed, T. P. (2003). A comparison of normalization methods for high density oligonucleotide array data based on variance and bias. *Bioinformatics*, *19*(2), 185–193. https://doi.org/10.1093/bioinformatics/19.2.185

Carvalho, B. S., & Irizarry, R. A. (2010). A framework for oligonucleotide microarray preprocessing. *Bioinformatics*, *26*(19), 2363–2367. https://doi.org/10.1093/bioinformatics/btq431

Conesa, A., Madrigal, P., Tarazona, S., Gomez-Cabrero, D., Cervera, A., McPherson, A., Szcześniak, M. W., Gaffney, D. J., Elo, L. L., & Zhang, X. (2016). A survey of best practices for RNA-seq data analysis. *Genome biology*, *17*, 13. https://doi.org/10.1186/s13059-016-0881-8

Croft, D., Mundo, A. F., Haw, R., Milacic, M., Weiser, J., Wu, G., Caudy, M., Garapati, P., Gillespie, M., Kamdar, M. R., Jassal, B., Jupe, S., Matthews, L., May, B., Palatnik, S., Rothfels, K., Shamovsky, V., Song, H., Williams, M., … D'Eustachio, P. (2014). The Reactome pathway knowledgebase. *Nucleic Acids Research*, *42*(D1), D472–D477. https://doi.org/10.1093/nar/gkt1102

Draghici, S., Khatri, P., Tarca, A. L., Amin, K., Done, A., Voichita, C., Georgescu, C., & Romero, R. (2007). A systems biology approach for pathway level analysis. *Genome Research*, *17*(10), 1537–1545. https://doi.org/10.1101/gr.6202607

Efron, B., & Tibshirani, R. (2007). On testing the significance of sets of genes. *The Annals of Applied Statistics*, *1*(1), 107–129. https://doi.org/10.1214/07-AOAS101

Fisher, R. A. (1925). *Statistical methods for research workers*. Oliver & Boyd.

Gentleman, R., Whalen, E., Huber, W., & Falcon, S. (2023). *Graph: A package to handle graph data structures*. In *Bioconductor, R package version 1.80.0*.

Glaab, E., Baudot, A., Krasnogor, N., & Valencia, A. (2010). TopoGSA: Network topological gene set analysis. *Bioinformatics*, *26*(9), 1271–1272. https://doi.org/10.1093/bioinformatics/btq131

Gu, Z., Liu, J., Cao, K., Zhang, J., & Wang, J. (2012). Centrality-based pathway enrichment: A systematic approach for finding significant pathways dominated by key genes. *BMC Systems Biology*, *6*, 56. https://doi.org/10.1186/1752-0509-6-56

Gu, Z., & Wang, J. (2013). CePa: An R package for finding significant pathways weighted by multiple network centralities. *Bioinformatics*, *29*(5), 658–660. https://doi.org/10.1093/bioinformatics/btt008

Harbron, C., Chang, K.-M., & South, M. C. (2007). RefPlus: An R package extending the RMA Algorithm. *Bioinformatics*, *23*(18), 2493–2494. https://doi.org/10.1093/bioinformatics/btm357

Hosack, D. A., Dennis, G., Sherman, B. T., Lane, H. C., & Lempicki, R. A. (2003). Identifying biological themes within lists of genes with EASE. *Genome Biology*, *4*(10), 1–8. https://doi.org/10.1186/gb-2003-4-10-r70

Huang, D. W., Sherman, B. T., & Lempicki, R. A. (2009). Systematic and integrative analysis of large gene lists using DAVID bioinformatics resources. *Nature Protocols*, *4*(1), 44–57. https://doi.org/10.1038/nprot.2008.211

Irizarry, R. A., Hobbs, B., Collin, F., Beazer-Barclay, Y. D., Antonellis, K. J., Scherf, U., & Speed, T. P. (2003). Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics*, *4*(2), 249–264. https://doi.org/10.1093/biostatistics/4.2.249

Kanehisa, M., Furumichi, M., Tanabe, M., Sato, Y., & Morishima, K. (2017). KEGG: New perspectives on genomes, pathways, diseases and drugs.

Nucleic Acids Research, *45*(D1), D353–D361. https://doi.org/10.1093/nar/gkw1092

Kerr, M. K., Martin, M., & Churchill, G. A. (2000). Analysis of variance for gene expression microarray data. *Journal of Computational Biology*, *7*(6), 819–837. https://doi.org/10.1089/10665270050514954

Khatri, P., Draghici, S., Ostermeier, G. C., & Krawetz, S. A. (2002). Profiling gene expression using onto-express. *Genomics*, *79*(2), 266–270. https://doi.org/10.1006/geno.2002.6698

Knapp, G., & Hartung, J. (2003). Improved tests for a random effects meta-regression with a single covariate. *Statistics in Medicine*, *22*(17), 2693–2710. https://doi.org/10.1002/sim.1482

Kolde, R., Laur, S., Adler, P., & Vilo, J. (2012). Robust rank aggregation for gene list integration and meta-analysis. *Bioinformatics*, *28*(4), 573–580. https://doi.org/10.1093/bioinformatics/btr709

Korotkevich, G., Sukhov, V., Budin, N., Shpak, B., Artyomov, M. N., & Sergushichev, A. (2021). Fast gene set enrichment analysis. *BioRxiv*, 060012. https://doi.org/10.1101/060012

Law, C. W., Zeglinski, K., Dong, X., Alhamdoosh, M., Smyth, G. K., & Ritchie, M. E. (2020). A guide to creating design matrices for gene expression experiments. *F1000Research*, *9*(1), 1444. https://doi.org/10.12688/f1000research.27893.1

Love, M. I., Huber, W., & Anders, S. (2014). Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology*, *15*, 1–21. https://doi.org/10.1186/s13059-014-0550-8

Massey Jr, F. J. (1951). The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association*, *46*(253), 68–78.

Mitrea, C., Taghavi, Z., Bokanizad, B., Hanoudi, S., Tagett, R., Donato, M., Voichiţa, C., & Drăghici, S. (2013). Methods and approaches in the topology-based analysis of biological pathways. *Frontiers in Physiology*, *4*, 278. https://doi.org/10.3389/fphys.2013.00278

Mootha, V. K., Lindgren, C. M., Eriksson, K.-F., Subramanian, A., Sihag, S., Lehar, J., Puigserver, P., Carlsson, E., Ridderstråle, M., & Laurila, E. (2003). PGC-1α-responsive genes involved in oxidative phosphorylation are coordinately downregulated in human diabetes. *Nature genetics*, *34*(3), 267–273. https://doi.org/10.1038/ng1180

Nguyen, H., Tran, D., Galazka, J. M., Costes, S. V., Beheshti, A., Draghici, S., & Nguyen, T. (2021). CPA: A web-based platform for Consensus Pathway Analysis and interactive visualization. *Nucleic Acids Research*, *49*(W1), W114–W124. https://doi.org/10.1093/nar/gkab421

Nguyen, T., Diaz, D., Tagett, R., & Draghici, S. (2016). Overcoming the matched-sample bottleneck: An orthogonal approach to integrate omic data. *Scientific Reports*, *6*, 29251. https://doi.org/10.1038/srep29251

Nguyen, T., Mitrea, C., & Draghici, S. (2018). Network-based approaches for pathway level analysis. *Current Protocols in Bioinformatics*, *61*, 8.25.21–28.25.24. https://doi.org/10.1002/cpbi.42

Nguyen, T., Mitrea, C., Tagett, R., & Draghici, S. (2017). DANUBE: Data-driven meta-ANalysis using UnBiased Empirical distributions - applied to biological pathway analysis. *Proceedings of the IEEE*, *105*(3), 496–515. https://doi.org/10.1109/JPROC.2015.2507119

Nguyen, T., Shafi, A., Nguyen, T.-M., Schissler, A. G., & Draghici, S. (2020). NBIA: A network-based integrative analysis framework–applied to pathway analysis. *Scientific Reports*, *10*, 4188. https://doi.org/10.1038/s41598-020-60981-9

Nguyen, T., Tagett, R., Donato, M., Mitrea, C., & Draghici, S. (2016). A novel bi-level meta-analysis approach-applied to biological pathway analysis. *Bioinformatics*, *32*(3), 409–416. https://doi.org/10.1093/bioinformatics/btv588

Nguyen, T.-M., Shafi, A., Nguyen, T., & Draghici, S. (2019). Identifying significantly impacted pathways: A comprehensive review and assessment. *Genome Biology*, *20*, 203. https://doi.org/10.1186/s13059-019-1790-4

Normand, S. L. T. (1999). Meta-analysis: Formulating, evaluating, combining, and reporting. *Statistics in medicine*, *18*(3), 321–359. https://doi.org/10.1002/(sici)1097-0258(19990215)18:3⟨321::aid-sim28⟩3.0.co;2-p

Ritchie, M. E., Phipson, B., Wu, D., Hu, Y., Law, C. W., Shi, W., & Smyth, G. K. (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research*, *43*(7), e47–e47. https://doi.org/10.1093/nar/gkv007

Robertson, G., Schein, J., Chiu, R., Corbett, R., Field, M., Jackman, S. D., Mungall, K., Lee, S., Okada, H. M., & Qian, J. Q. (2010). De novo assembly and analysis of RNA-seq data. *Nature methods*, *7*(11), 909–912. https://doi.org/10.1038/nmeth.1517

Robinson, M. D., McCarthy, D. J., & Smyth, G. K. (2010). edgeR: A Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, *26*(1), 139–140. https://doi.org/10.1093/bioinformatics/btp616

Sergushichev, A. (2016). An algorithm for fast preranked gene set enrichment analysis using cumulative statistic calculation. *BioRxiv*, *60012*, 1–9.

Stouffer, S. A., Suchman, E. A., DeVinney, L. C., Star, S. A., & williams Jr, R. M. (1949). *The American Soldier: Adjustment during army life* (Vol. *1*). Princeton University Press.

Student. (1908). The probable error of a mean. *Biometrika*, *6*(1), 1–25. https://doi.org/10.2307/2331554

Subramanian, A., Tamayo, P., Mootha, V. K., Mukherjee, S., Ebert, B. L., Gillette, M. A., Paulovich, A., Pomeroy, S. L., Golub, T. R., Lander, E. S., & Mesirov, J. P. (2005).

Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. *Proceeding of The National Academy of Sciences*, *102*(43), 15545–15550. https://doi.org/10.1073/pnas.0506580102

Tarca, A. L., Draghici, S., Khatri, P., Hassan, S. S., Mittal, P., Kim, J.-S., Kim, C. J., Kusanovic, J. P., & Romero, R. (2009). A novel signaling pathway impact analysis. *Bioinformatics*, *25*(1), 75–82. https://doi.org/10.1093/bioinformatics/btn577

The Gene Ontology Consortium. (2021). The gene ontology resource: Enriching a GOld mine. *Nucleic Acids Research*, *49*(D1), D325–D334. https://doi.org/10.1093/nar/gkaa1113

Tippett, L. H. C. (1931). *The Methods of Statistics. An Introduction mainly for Workers in the Biological Sciences*. Williams \& Norgate.

Viechtbauer, W. (2005). Bias and efficiency of meta-analytic variance estimators in the random-effects model. *Journal of Educational and Behavioral Statistics*, *30*(3), 261–293. https://doi.org/10.3102/10769986030003261

Vovk, V., & Wang, R. (2020). Combining p-values via averaging. *Biometrika*, *107*(4), 791–808. https://doi.org/10.1093/biomet/asaa027

Wilcoxon, F. (1992). Individual comparisons by ranking methods. In *Breakthroughs in Statistics* (pp. 196–202). Springer.

Internet Resources

https://CRAN.R-project.org/package=RCPA

The freely available RCPA package.

https://github.com/tinnlab/RCPA/blob/main/examples/RCPA-Protocol.ipynb

All code snippets in this article organized into a single Jupyter notebook.