PathwayPCA: an R/Bioconductor Package for Pathway-Based Integrative Analysis of Multi-Omics Data

Analysis Scripts and Workflow to Replicate Examples

Gabriel Odom

4/17/2020

Table of Contents

Overview

This document shows the entire analysis workflow for the examples in the main manuscript, following the subsections Section 3:

3.2 A WikiPathways analysis of CPTAC ovarian cancer protein expression data
3.3 An integrative pathway analysis of gene expression and protein expression data for ovarian cancer
3.4 Integrating gene expression data with experimental design information: an analysis of sex-specific pathway gene expression effects on kidney cancer
3.5 A pathway based integrative prediction model for patient prognosis

Names, descriptions, file names, and sources for all data sets can be accessed from `extdata/README` at https://github.com/TransBioInfoLab/pathwayPCA_PROTEOMICS_Supplement.

3.2 A WikiPathways analysis of CPTAC ovarian cancer protein expression data

Data Import and Setup

For this section, we will need the following packages:

```
library(impute)
library(tidyverse)
library(pathwayPCA)
```

We import the raw data:

```
# Proteomics
ovarianProtPNNL_tbldf <- read_delim(

"extdata/Human_TCGA_OV_PNNL_Proteome_Velos_QExact_01_28_2016_PNNL_Gene_CDAP_iTRAQ_Unshare
dLogRatio_r2.cct",
  "\t", escape_double = FALSE, trim_ws = TRUE
)

# Clinical Response
ovarianPheno_tbldf <- read_delim(
  "extdata/Human_TCGA_OV_MS_Clinical_Clinical_01_28_2016_BI_Clinical_Firehose.tsi",
  "\t", escape_double = FALSE, trim_ws = TRUE
)
```

We also import the WikiPathways pathway collection used at the time of original analysis, translated from Entrez Gene IDs to Gene Symbols.

```
human_PC <- read_gmt(
  "extdata/wikipathways_human_symbol.gmt",
  description = TRUE
)
human_PC

## Object with Class(es) 'pathwayCollection', 'list' [package 'pathwayPCA'] with 3
elements:
##  $ pathways   :List of 457
##  $ TERMS      : chr [1:457] "WP23" ...
##  $ description: chr [1:457] "B Cell Receptor Signaling Pathway" ...
```

Join and Clean the Data

Many of the recorded protein values are missing, so we need to remove proteins and subjects with high missing proportions and impute the remaining missing values.

Join Data

We transpose the data from $p \times n$ to $n \times p$ form and then join the clinical to the proteomic data.

```
# Proteomics
ovarianProtT_df <-
  ovarianProtPNNL_tbldf %>%
  TransposeAssay()
rm(ovarianProtPNNL_tbldf)
```

```
# Clinical Response
ovarianSurv_df <-
  ovarianPheno_tbldf %>%
  TransposeAssay() %>%
  mutate(OS_time = as.numeric(overall_survival)) %>%
  mutate(OS_event = as.numeric(status)) %>%
  select(Sample, OS_time, OS_event)
rm(ovarianPheno_tbldf)

# Join
ovarian_df <- inner_join(ovarianSurv_df, ovarianProtT_df)

## Joining, by = "Sample"

rm(ovarianSurv_df, ovarianProtT_df)
```

Check Data Missingness

We have 84 samples, but one is missing survival response:

```
badSampleIDs_char <-
  ovarian_df %>%
  select(Sample, OS_time, OS_event) %>%
  filter(!complete.cases(.)) %>%
  pull(Sample)

badSampleIDs_char

## [1] "TCGA.24.2288"
```

Without this sample, the overall missingness is

```
ovarian_df %>%
  filter(!(Sample %in% badSampleIDs_char)) %>%
  is.na() %>%
  mean

## [1] 0.1226747
```

We remove any proteins that have missingness higher than the 80th percentile of missingness in the data set (about 30%).

```
pctMissing_num <-
  ovarian_df %>%
  filter(!(Sample %in% badSampleIDs_char)) %>%
  map_dbl(
    ~{ mean(is.na(.x)) }
  )

quantile(pctMissing_num, 0.8)

##       80%
## 0.2891566

dropProteins_idx <- which(pctMissing_num > 0.2891566)

ovarianTrim_df <-
  ovarian_df %>%
  filter(!(Sample %in% badSampleIDs_char)) %>%
  select(-dropProteins_idx)
```

With this sample and these proteins removed, the overall missingness drops to a touch under 5% (which we believe will be safe to impute).

```
ovarianTrim_df %>%
  is.na() %>%
  mean

## [1] 0.04457015

rm(ovarian_df, badSampleIDs_char, dropProteins_idx, pctMissing_num)
```

Impute Missing Values

We perform imputation via the impute package on Bioconductor. This method uses k-Nearest-Neighbors. The `impute.knn()` function requires a gene matrix in $p \times n$ form, so we transpose, impute, then transpose back.

```
ovarianProtClean_df <-
  ovarianTrim_df %>%
  # Remove clinical data
  select(-Sample, -OS_time, -OS_event) %>%
  # transpose and coerce to a matrix
  t() %>%
  # impute with kNN
  impute.knn() %>%
  # extract the imputed data
  pluck("data") %>%
  # promote to data frame
  as.data.frame() %>%
  # transpose as a data frame
  TransposeAssay(omeNames = "rowNames") %>%
  # Re-attach clinical data
  bind_cols(
    select(ovarianTrim_df, Sample, OS_time, OS_event),
    .
  )

## Cluster size 5162 broken into 4396 766
## Cluster size 4396 broken into 969 3427
## Done cluster 969
## Cluster size 3427 broken into 1445 1982
## Done cluster 1445
## Cluster size 1982 broken into 432 1550
## Done cluster 432
## Cluster size 1550 broken into 1008 542
## Done cluster 1008
## Done cluster 542
## Done cluster 1550
## Done cluster 1982
## Done cluster 3427
## Done cluster 4396
## Done cluster 766

anyNA(ovarianProtClean_df)
```

```
## [1] FALSE

rm(ovarianTrim_df)
```

AES-PCA Analysis

Now that we have clean data, we can create an `Omics*` data container and perform the AES-PCA analysis.

Data Container

```
ovProt_OmicsSurv <- CreateOmics(
  assayData_df = ovarianProtClean_df[, -(2:3)],
  pathwayCollection_ls = human_PC,
  response = ovarianProtClean_df[, 1:3],
  respType = "survival"
)

##
##   ======  Creating object of class OmicsSurv  =======

## The input pathway database included 5831 unique features.

## The input assay dataset included 5162 features.

## Only pathways with at least 3 or more features included in the assay dataset are
##   tested (specified by minPathSize parameter). There are 363 pathways which meet
##   this criterion.

## Because pathwayPCA is a self-contained test (PMID: 17303618), only features in
##   both assay data and pathway database are considered for analysis. There are 2078
##   such features shared by the input assay and pathway database.

ovProt_OmicsSurv

## Formal class 'OmicsSurv' [package "pathwayPCA"] with 6 slots
##   ..@ eventTime             : num [1:83] 2279 3785 2154 2553 2856 ...
##   ..@ eventObserved         : logi [1:83] TRUE TRUE TRUE TRUE TRUE TRUE ...
##   ..@ assayData_df          :Classes 'tbl_df', 'tbl' and 'data.frame':    83 obs. of
5162 variables:
##   ..@ sampleIDs_char        : chr [1:83] "TCGA.09.1664" "TCGA.13.1484" "TCGA.13.1488"
"TCGA.13.1489" ...
##   ..@ pathwayCollection     :List of 4
##   .. ..- attr(*, "class")= chr [1:2] "pathwayCollection" "list"
##   ..@ trimPathwayCollection:List of 5
##   .. ..- attr(*, "class")= chr [1:2] "pathwayCollection" "list"
```

AES-PCA

We have roughly 350 pathways to test, and this takes about two minutes with parallel computing. Methods for false discovery rate adjustments can be passed to the `adjustment` argument (each method supplied will add a column to the results table).

```
ovProt_aespcOut <- AESPCA_pVals(
  ovProt_OmicsSurv,
  numPCs = 1,
  parallel = TRUE,
  numCores = 16,
  adjustment = "BH"
)

## Part 1: Calculate Pathway AES-PCs

## Initializing Computing Cluster: DONE
## Extracting Pathway PCs in Parallel: DONE
##
## Part 2: Calculate Pathway p-Values
## Initializing Computing Cluster: DONE
## Extracting Pathway p-Values in Parallel: DONE
##
## Part 3: Adjusting p-Values and Sorting Pathway p-Value Data Frame
## DONE
```

Inspect the Results

We will inspect the top most significant pathways, extract data from a specific pathway, and inspect the protein loadings and subject pathway scores from that pathway.

Top Pathways

We use the `getPathpVals()` function to look at the top most significant pathways. Note that the *p*-values from these top pathways are can vary slightly (because the estimated first principal component from each pathway can vary slightly depending on the convergence of the adaptive, elastic-net, sparse PCA algorithm). But the differences in *p*-values are usually small, and differ only in the third or fourth decimals. As a result, the ranking order of the pathways in this workflow is slightly different from those described in main manuscript.

```
# Use kable for pretty tables
getPathpVals(ovProt_aespcOut, numPaths = 10L) %>%
  knitr::kable()
```

terms	description	rawp	FDR_BH
WP2036	TNF related weak inducer of apoptosis (TWEAK) Signaling Pathway	0.002	0.3111429
WP3617	Photodynamic therapy-induced NF-kB survival signaling	0.005	0.3111429
WP1531	Vitamin D Metabolism	0.006	0.3111429
WP3850	Factors and pathways affecting insulin-like growth factor (IGF1)-Akt signaling	0.006	0.3111429
WP363	Wnt Signaling Pathway	0.007	0.3111429
WP384	Apoptosis Modulation by HSP70	0.008	0.3111429
WP3851	TLR4 Signaling and Tolerance	0.008	0.3111429
WP195	IL-1 signaling pathway	0.009	0.3111429
WP2447	Amyotrophic lateral sclerosis (ALS)	0.011	0.3111429
WP3858	Toll-like Receptor Signaling	0.011	0.3111429

Pathway Data Subset

We see that the "IL-1 Signalling Pathway" is abbreviated by the WikiPathways code "WP195". Thus, we can extract the protein data specific to this pathway with the `SubsetPathwayData()` function.

```
il1Signal_df <- SubsetPathwayData(ovProt_OmicsSurv, "WP195")

dim(il1Signal_df)

## [1] 83 33

il1Signal_df[1:5, 1:5]

## # A tibble: 5 x 5
##   sampleID     EventTime EventObs  ATF2  MAPK14
##   <chr>            <dbl> <lgl>    <dbl>   <dbl>
## 1 TCGA.09.1664      2279 TRUE     0.651 -0.995
## 2 TCGA.13.1484      3785 TRUE     1.20   1.71
## 3 TCGA.13.1488      2154 TRUE     0.351 -0.127
## 4 TCGA.13.1489      2553 TRUE     0.510 -0.0588
## 5 TCGA.13.1494      2856 TRUE     0.158  0.610
```

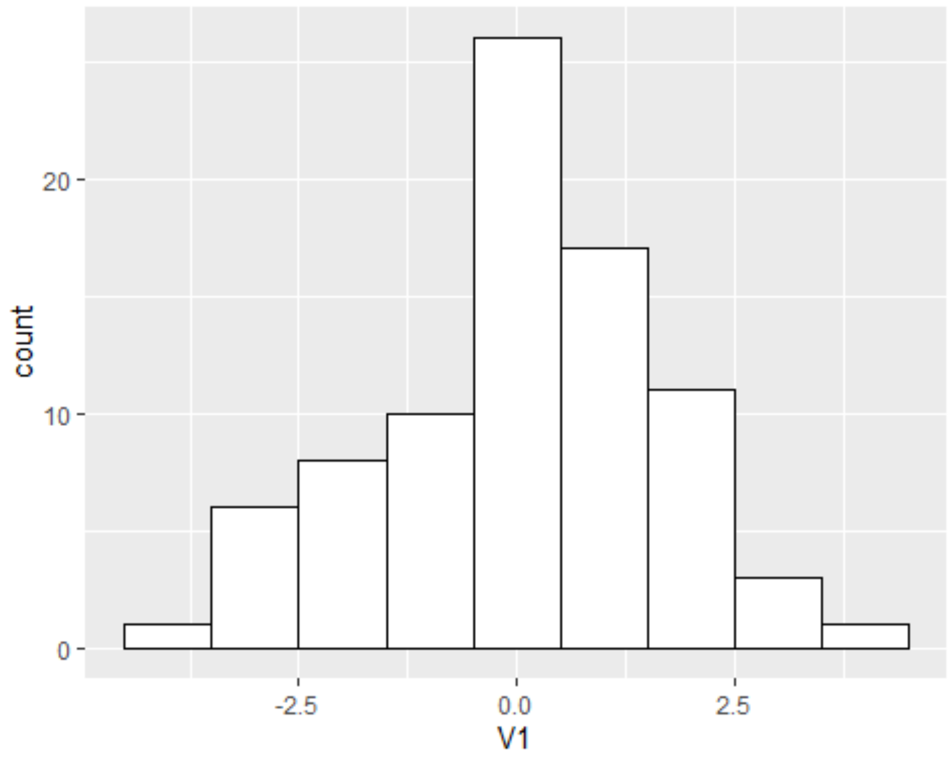
Loadings and Scores

We now extract the principal component loadings and scores for this pathway.

```
wp195PCA_ls <- getPathPCLs(ovProt_aespcOut, "WP195")
```

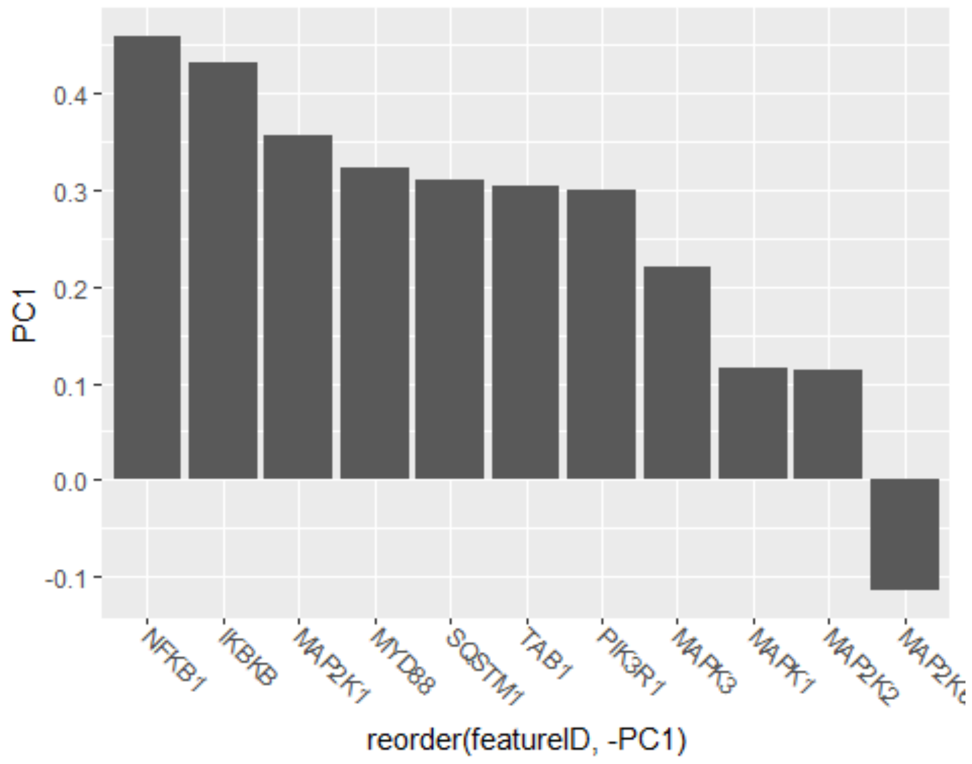
Subject-Specific Pathway Estimates

```
ggplot(wp195PCA_ls$PCs) +
  aes(x = V1) +
  geom_histogram(
    binwidth = 1,
    colour = "black",
    fill = "white"
  )
```



Protein Loadings within a Pathway

```
ggplot(
  wp195PCA_ls$Loadings %>%
    filter(PC1 != 0)
) +
  theme(axis.text.x = element_text(angle = -45, hjust = 0, vjust = 0.5)) +
  aes(x = reorder(featureID, -PC1), y = PC1) +
  geom_col()
```



```
rm(wp195PCA_ls)
```

Changing the Pathway Collection

We can also analyze this same data with pathways in the GSEA Molecular Signatures Database.

Re-Build the Data Container

Because we change the pathway collection, the features retained in the assay will also change.

```
c2cp_PC <- read_gmt(
  "extdata/c2.cp.v6.2.symbols.gmt",
  description = TRUE
)

ovProt_OmicsSurv <- CreateOmics(
  assayData_df = ovarianProtClean_df[, -(2:3)],
  pathwayCollection_ls = c2cp_PC,
  response = ovarianProtClean_df[, 1:3],
  respType = "survival"
)

##
##   ======  Creating object of class OmicsSurv  =======
## The input pathway database included 8904 unique features.

## The input assay dataset included 5162 features.

## Only pathways with at least 3 or more features included in the assay dataset are
##   tested (specified by minPathSize parameter). There are 1236 pathways which meet
##   this criterion.

## Because pathwayPCA is a self-contained test (PMID: 17303618), only features in
##   both assay data and pathway database are considered for analysis. There are 2981
##   such features shared by the input assay and pathway database.

ovProt_OmicsSurv

## Formal class 'OmicsSurv' [package "pathwayPCA"] with 6 slots
##   ..@ eventTime            : num [1:83] 2279 3785 2154 2553 2856 ...
##   ..@ eventObserved        : logi [1:83] TRUE TRUE TRUE TRUE TRUE TRUE ...
##   ..@ assayData_df         :Classes 'tbl_df', 'tbl' and 'data.frame':    83 obs. of
5162 variables:
##   ..@ sampleIDs_char       : chr [1:83] "TCGA.09.1664" "TCGA.13.1484" "TCGA.13.1488"
"TCGA.13.1489" ...
##   ..@ pathwayCollection    :List of 4
##   .. ..- attr(*, "class")= chr [1:2] "pathwayCollection" "list"
##   ..@ trimPathwayCollection:List of 5
##   .. ..- attr(*, "class")= chr [1:2] "pathwayCollection" "list"
```

Updated AES-PCA Analysis

Let's compute the PCs for this pathway collection and inspect the top pathways. (As we noted previously, this pathway collection is nearly four times larger, so this computation takes four times longer.)

```
ovProt_aespcOut <- AESPCA_pVals(
  ovProt_OmicsSurv,
  numPCs = 1,
  parallel = TRUE,
  numCores = 16,
  adjustment = "BH"
)

## Part 1: Calculate Pathway AES-PCs

## Initializing Computing Cluster: DONE
## Extracting Pathway PCs in Parallel: DONE
##
## Part 2: Calculate Pathway p-Values
## Initializing Computing Cluster: DONE
## Extracting Pathway p-Values in Parallel: DONE
##
## Part 3: Adjusting p-Values and Sorting Pathway p-Value Data Frame
## DONE
```

The top pathways of the C2 Canonical Pathways collection are:

```
getPathpVals(ovProt_aespcOut, numPaths = 10L) %>%
  mutate(terms = str_trunc(terms, 45)) %>%
  select(terms, rawp) %>%
  knitr::kable()
```

terms	rawp
BIOCARTA_41BB_PATHWAY	0.000
BIOCARTA_TID_PATHWAY	0.001
BIOCARTA_GCR_PATHWAY	0.001
ST_GAQ_PATHWAY	0.001
REACTOME_TRANSPORT_OF_VITAMINS_NUCLEOSIDES…	0.001
BIOCARTA_AKT_PATHWAY	0.002
ST_GA13_PATHWAY	0.002
BIOCARTA_FMLP_PATHWAY	0.003
BIOCARTA_CARDIACEGF_PATHWAY	0.003
BIOCARTA_IL1R_PATHWAY	0.003

3.3 An integrative pathway analysis of gene expression and protein expression data for ovarian cancer

Given the ovarian cancer proteomics results in Section 3.2 (using the C2 CP collection), we additionally analyze the gene expression data for this integrative analysis.

Data Import, Wrangling, Cleaning, and Joining

We import the raw gene expression RNAseq - IlluminaHiSeq pancan normalized data:

```r
# RNAseq
ovarianRNAseq_tbldf <- read_delim(
  "extdata/HiSeqV2_PANCAN",
  "\t", escape_double = FALSE, trim_ws = TRUE
)

# Clinical Response
ovarianPheno_tbldf <- read_delim(
  "extdata/Human_TCGA_OV_MS_Clinical_Clinical_01_28_2016_BI_Clinical_Firehose.tsi",
  "\t", escape_double = FALSE, trim_ws = TRUE
)
```

As before, we transpose the data and join.

```r
# RNAseq
ovarianRNAseqT_df <-
  ovarianRNAseq_tbldf %>%
  TransposeAssay() %>%
  # detect if the sample is a primary sample
  mutate(primary = str_detect(Sample, pattern = "\\w*01\\b")) %>%
  # remove the sample number
  mutate(Sample = str_sub(Sample, end = -4)) %>%
  # detect if samples are duplicated
  mutate(duplicate = duplicated(Sample)) %>%
  # mark if the sample is a primary sample or is not duplicated
  mutate(keep = primary | !duplicate) %>%
  # remove the rows that don't meet our qualifications
  filter(keep) %>%
  # remove these "working" columns
  select(-primary, -duplicate, -keep) %>%
  # replace the dashes with full stops in the sample IDs
  mutate(
    Sample = str_replace_all(Sample, pattern = "-", replacement = ".")
  )

rm(ovarianRNAseq_tbldf)

# Clinical Response
ovarianSurv_df <-
  ovarianPheno_tbldf %>%
  TransposeAssay() %>%
  mutate(OS_time = as.numeric(overall_survival)) %>%
  mutate(OS_event = as.numeric(status)) %>%
  select(Sample, OS_time, OS_event) %>%
  drop_na()
rm(ovarianPheno_tbldf)

# Join
ovarianRNAseq_df <- inner_join(ovarianSurv_df, ovarianRNAseqT_df)

## Joining, by = "Sample"

rm(ovarianSurv_df, ovarianRNAseqT_df)
```

AES-PCA Analysis

Now that we have joined data, we can create an `Omics*` data container and perform the AES-PCA analysis.

Data Container

```
ovRNAseq_OmicsSurv <- CreateOmics(
  assayData_df = ovarianRNAseq_df[, -(2:3)],
  pathwayCollection_ls = c2cp_PC,
  response = ovarianRNAseq_df[, 1:3],
  respType = "survival"
)

## 347 genes have variance < epsilon and will be removed. These gene(s) are:

##   [1] "RBMY1A3P"     "SNORD115-17" "SNORD114-31" "SNORD114-30" "SNORD115-6"
##   [6] "SNORD115-4"   "SNORD104"    "TBL1Y"       "SNORD115-9"  "SNORD38A"
##  [11] "OR8U1"        "KRTAP19-2"   "KRTAP19-6"   "KRTAP19-7"   "KRTAP19-4"
##  [16] "SNORD114-3"   "SRY"         "?|441362"    "HBII-52-27"  "SNORD11"
##  [21] "SNORD12"      "SNORD21"     "SNORD20"     "SNORD25"     "SNORD127"
##  [26] "DAZ3"         "DAZ2"        "DAZ4"        "SNORD124"    "TTTY17A"
##  [31] "SNORD115-5"   "SNORD80"     "TSPY3"       "TSPY1"       "TSPY2"
##  [36] "SNORD56B"     "SNORD78"     "SNORD113-7"  "SNORD96A"    "SNORD119"
##  [41] "HBII-52-46"   "HBII-52-45"  "SNORD47"     "OR2G3"       "SNORD4B"
##  [46] "SNORD4A"      "SNORD116-24" "SNORD116-25" "SNORD116-23" "SNORD116-29"
##  [51] "RBMY1B"       "RBMY1F"      "CSN1S2A"     "BPY2"        "SNORD115-8"
##  [56] "OR6Q1"        "SNORD66"     "HBII-52-28"  "SNORD69"     "SNORD68"
##  [61] "SNORD38B"     "SNORD113-4"  "SNORD116-26" "SNORD115-1"  "SNORD58A"
##  [66] "SNORD58C"    "PRR20D"      "SNORD96B"    "SNORD63"     "SNORD116-22"
##  [71] "PRY2"        "SNORD81"     "PRAMEF3"     "SNORD114-20" "SNORD115-20"
##  [76] "SNORD115-22" "SNORD115-25" "SNORD114-29" "OR8H3"       "OR5L1"
##  [81] "NCRNA00230B" "DEFB130"     "DEFB131"     "SNORD29"     "S100A7L2"
##  [86] "PPIAL4B"     "SNORD16"     "SNORD36A"    "SNORD91B"    "RBMY1J"
##  [91] "SNORD87"     "SNORD86"     "DEFB116"     "DEFB114"     "DEFB113"
##  [96] "TTTY4C"      "RBMY1E"      "SNORD91A"    "SNORD88B"    "SNORD50B"
## [101] "SNORD65"     "KRTAP22-1"   "SNORD61"     "SNORD115-35" "SNORD114-10"
## [106] "KRTAP13-1"   "C9orf27"     "SNORD114-6"  "SNORD114-5"  "FAM41AY1"
## [111] "SNORD12C"    "SNORD123"    "SNORD126"    "SNORD125"    "DEFB108B"
## [116] "KRTAP23-1"   "SNORD48"     "OR5L2"       "SNORD114-9"  "SNORD46"
## [121] "SNORD115-2"  "SNORD115-3"  "SNORD27"     "SNORD105"    "RBMY2EP"
## [126] "SNORD35A"    "SNORD35B"    "RBMY3AP"     "OR6K2"       "TTTY22"
## [131] "SNORD24"     "SNAR-B2"     "SNORD28"     "XKRY2"       "KRTAP25-1"
## [136] "TTTY6B"      "SNORD49A"    "C11orf40"    "SNORD102"    "RPS4Y2"
## [141] "SNORD115-14" "LCE4A"       "SNORD115-16" "POM121L12"   "SNORD30"
## [146] "SNORD31"     "SNORD115-10" "SNORD18B"    "?|728045"    "OR4A5"
## [151] "LOC730811"   "SNORD26"     "SNORD114-7"  "SNORD45C"    "SNORD111B"
## [156] "SNORD18A"    "TTTY2"       "TTTY6"       "TTTY7"       "TTTY5"
## [161] "TTTY8"       "KRTAP9-9"    "TTTY20"      "SNORD109B"   "SNORD45B"
## [166] "TTTY23"      "SNORD45A"    "KRTAP6-1"    "OR5D16"      "SNORD114-22"
## [171] "SNORD114-23" "SNORD114-26" "SNORD114-27" "SNORD114-24" "SNORD114-25"
## [176] "SNORD114-28" "SNORD115-41" "SNORD115-40" "CDY1B"       "SNORD1A"
## [181] "SNORD1B"     "SNORD19"     "SNORD115-37" "SNORD44"     "SNORD34"
## [186] "SNORD121B"   "SNORD121A"   "SNAR-E"      "SNAR-H"      "SNORD51"
## [191] "SNORD115-44" "SNORD36C"    "SNORD115-48" "SNORD11B"    "SNORD49B"
## [196] "SNORD116-19" "SNORD116-11" "DEFB110"     "SNORD59B"    "SNORD59A"
## [201] "SNORD116-27" "OR5W2"       "SNORD114-21" "SNORD114-11" "SNORD72"
## [206] "SNORD74"     "SNORD75"     "SNORD79"     "SNORD114-4"  "OR2T29"
## [211] "OR2T27"      "CDY1"        "RNU5E"       "SNORD88C"    "RNY5"
## [216] "RNY4"        "SNORD114-16" "SNORD52"     "SNORD53"     "SNORD115-11"
## [221] "SNORD113-9"  "SNORD113-5"  "SNORD113-6"  "SNORD113-1"  "SNORD113-2"
## [226] "SNORD114-14" "SNORD116-18" "KRTAP20-3"   "SNORD98"     "SNORD99"
## [231] "SNORD93"     "KRTAP20-1"   "OR8I2"       "SNORD116-12" "NMS"
## [236] "SNORD73A"    "SPINK14"     "SNORD115-32" "SNORD115-39" "DEFB128"
## [241] "SNORD116-10" "SNORD116-13" "SNORD116-15" "SNORD116-14" "SNORD116-16"
## [246] "OR5M8"       "XKRY"        "SNORD95"     "SNORD90"     "SNORD114-18"
## [251] "SNORD92"     "TSSK2"       "SNORA70C"    "SNORD114-15" "SNORD19B"
## [256] "SNORD70"     "SNORD76"     "?|317712"    "SNORD77"     "HBII-52-24"
## [261] "SNAR-A13"    "TREML2P1"    "SNORD60"     "TTTY3B"      "KRTAP13-3"
## [266] "SNORD105B"   "TTTY17B"     "SNORD6"      "SNORD7"      "SNORD5"
## [271] "SNORD2"      "SNORD33"     "VTRNA1-2"    "OR5A2"       "SNORD54"
## [276] "TTTY14"      "TTTY16"      "TTTY10"      "TTTY18"      "TTTY1B"
## [281] "SNORD114-8"  "SNORD55"     "SNORD56"     "SNORD57"     "TTTY21"
## [286] "RBMY1A1"     "KRTAP27-1"   "DUX4"        "SNORD50A"    "SNORD117"
## [291] "SNORD110"    "SNORD111"    "KRTAP4-5"    "SNAR-A4"     "TTTY12"
## [296] "SNORD32A"    "SNORD43"     "SNORD114-12" "SNORD71"     "PRR20B"
## [301] "SNAR-C3"     "SNAR-C2"     "RBMY2FP"     "SNORD88A"    "SNORD36B"
## [306] "SNAR-I"      "VTRNA1-3"    "VTRNA1-1"    "SNORD41"     "SNORD62A"
## [311] "LCE3B"       "SNORD116-5"  "SNORD103A"   "OR4F15"      "SNORD42B"
## [316] "OR6C74"      "IFNA16"      "SNORD42A"    "SNORD37"     "SNORD114-1"
## [321] "TTTY11"      "TTTY13"      "TTTY19"      "SNORD115-33" "SNORD115-30"
## [326] "TTTY9B"      "SNORD115-31" "SNORD114-2"  "SNORD115-38" "OR5M10"
## [331] "SNORD18C"    "SNORD12B"    "SNORD114-19" "SNORD114-13" "SNORD114-17"
## [336] "CT47A11"     "OR4P4"       "SNORD116-8"  "SNORD116-1"  "SNORD116-3"
## [341] "SNORD116-2"  "SNORD85"     "SNORD84"     "SNORD82"     "SNORD32B"
## [346] "IFNA7"       "AMELY"

## 29 gene name(s) are invalid. Invalid name(s) are:

##  [1] "?|26823"     "?|100133144" "?|90288"     "?|645851"    "?|404770"
##  [6] "?|653553"    "?|10357"     "?|388795"    "?|100134869" "?|729884"
## [11] "HGC6.3"      "CN5H6.4"     "?|57714"     "?|652919"    "?|100130426"
## [16] "?|553137"    "?|390284"    "?|728603"    "?|155060"    "?|136542"
## [21] "?|340602"    "RP1-177G6.2" "?|10431"     "?|442388"    "?|8225"
## [26] "?|391714"    "?|280660"    "?|391343"    "?|728788"

## These genes may be excluded from analysis. Proper gene names
## contain alphanumeric characters only, and start with a letter.

##
##   ======  Creating object of class OmicsSurv  =======

## The input pathway database included 8904 unique features.

## The input assay dataset included 20183 features.

## Only pathways with at least 3 or more features included in the assay dataset are
##   tested (specified by minPathSize parameter). There are 1329 pathways which meet
##   this criterion.

## Because pathwayPCA is a self-contained test (PMID: 17303618), only features in
##   both assay data and pathway database are considered for analysis. There are 8421
##   such features shared by the input assay and pathway database.

ovRNAseq_OmicsSurv

## Formal class 'OmicsSurv' [package "pathwayPCA"] with 6 slots
##   ..@ eventTime            : num [1:296] 420 624 361 1483 1348 ...
##   ..@ eventObserved        : logi [1:296] FALSE TRUE FALSE TRUE TRUE TRUE ...
##   ..@ assayData_df         :Classes 'tbl_df', 'tbl' and 'data.frame':    296 obs. of
20183 variables:
##   ..@ sampleIDs_char       : chr [1:296] "TCGA.3P.A9WA" "TCGA.59.A5PD" "TCGA.5X.AA5U"
"TCGA.04.1348" ...
##   ..@ pathwayCollection    :List of 4
##   .. ..- attr(*, "class")= chr [1:2] "pathwayCollection" "list"
##   ..@ trimPathwayCollection:List of 5
##   .. ..- attr(*, "class")= chr [1:2] "pathwayCollection" "list"
```

AES-PCA

We have roughly 1300 pathways to test, and this takes about sixteen minutes in parallel.

```
ovRNAseq_aespcOut <- AESPCA_pVals(
  ovRNAseq_OmicsSurv,
  numPCs = 1,
  parallel = TRUE,
  numCores = 16,
  adjustment = "BH"
)

## Part 1: Calculate Pathway AES-PCs
## Initializing Computing Cluster: DONE
## Extracting Pathway PCs in Parallel: DONE
##
## Part 2: Calculate Pathway p-Values
## Initializing Computing Cluster: DONE
## Extracting Pathway p-Values in Parallel: DONE
##
## Part 3: Adjusting p-Values and Sorting Pathway p-Value Data Frame
## DONE
```

Integrative Analysis

Intersection Method

There are 116 nominally-significant pathways from the protein analysis, and 93 nominally-significant pathways from the gene expression analysis. Of these, the following pathways are in common:

```
intersect(
  ovProt_aespcOut$pVals_df %>%
    filter(rawp < 0.05) %>%
    pull(terms),
  ovRNAseq_aespcOut$pVals_df %>%
    filter(rawp < 0.05) %>%
    pull(terms)
)
```

```
## [1] "PID_ECADHERIN_KERATINOCYTE_PATHWAY"
## [2] "PID_NCADHERIN_PATHWAY"
## [3] "REACTOME_SYNTHESIS_OF_PIPS_AT_THE_PLASMA_MEMBRANE"
## [4] "PID_TGFBR_PATHWAY"
```

However, none of them have FDR values below 30%.

Global-p Method

Here is the function we use to test the Cox Proportional Hazards model for each pathway:

```
library(survival)

pathSurvSignif <- function(pathway, resp1_df, omicsOut1, resp2_df, omicsOut2){
  # browser()

  ###  Matched Data  ###
  keptSamples <- intersect(resp1_df$sampleID, resp2_df$sampleID)
  innerResp_df <- resp1_df %>%
    filter(sampleID %in% keptSamples)
  colnames(innerResp_df)[2:3] <- c("time", "status")

  path_df <- innerResp_df %>%
    left_join(
      getPathPCLs(omicsOut1, pathway)$PCs, by = "sampleID"
    ) %>%
    rename(ome1 = V1) %>%
    left_join(
      getPathPCLs(omicsOut2, pathway)$PCs, by = "sampleID"
    ) %>%
    rename(ome2 = V1) %>%
    select(-sampleID)


  ###  Model Fits  ###
  # Data Set 1
  ome1_mod <- coxph(Surv(time, status) ~ ome1, data = path_df)

  # Dat Set 2
  ome2_mod <- coxph(Surv(time, status) ~ ome2, data = path_df)

  # Full Model
  full_mod <- coxph(Surv(time, status) ~ ., data = path_df)


  ###  Model p-Values  ###
  ome1p <- summary(ome1_mod)$logtest["pvalue"]
  ome2p <- summary(ome2_mod)$logtest["pvalue"]
  fullpVal <- summary(full_mod)$logtest["pvalue"]


  ###  Return  ###
  tibble(
    terms = pathway,
    ome1 = ome1p,
    ome2 = ome2p,
    global = fullpVal
  )

}
```

We create the two response data frames necessary:

```
protResp_df <- tibble(
  sampleID = getSampleIDs(ovProt_OmicsSurv),
  time     = getEventTime(ovProt_OmicsSurv),
  status   = getEvent(ovProt_OmicsSurv)
)
geneResp_df <- tibble(
  sampleID = getSampleIDs(ovRNAseq_OmicsSurv),
  time     = getEventTime(ovRNAseq_OmicsSurv),
  status   = getEvent(ovRNAseq_OmicsSurv)
)
```

Now we test the function on the *T* Cell Receptor Signalling Pathway (`BIOCARTA_TCR_PATHWAY`):

```
pathSurvSignif(
  pathway = "BIOCARTA_TCR_PATHWAY",
  resp1_df = protResp_df,
  omicsOut1 = ovProt_aespcOut,
  resp2_df = geneResp_df,
  omicsOut2 = ovRNAseq_aespcOut
)

## # A tibble: 1 x 4
##   terms                      ome1    ome2   global
##   <chr>                      <dbl>   <dbl>    <dbl>
## 1 BIOCARTA_TCR_PATHWAY 0.0000324 0.00386 0.000176
```

Once we have confirmed that the function works as expected for one pathway, we can apply it across all the pathways (so long as they are shared by both analysis results—recall that pathways are dependent on the assay, so not all pathways will have been measured for proteomics as were measured for gene expression and vice versa).

```
measuredPaths_char <- intersect(
  ovProt_aespcOut$pVals_df %>%
    pull(terms),
  ovRNAseq_aespcOut$pVals_df %>%
    pull(terms)
)

globalp_df <-
  map_dfr(
    measuredPaths_char,
    pathSurvSignif,
    resp1_df = protResp_df,
    omicsOut1 = ovProt_aespcOut,
    resp2_df = geneResp_df,
    omicsOut2 = ovRNAseq_aespcOut
  ) %>%
  rename(
    Proteomics = ome1,
    RNAseq = ome2
  )
```

We add on the false discovery rate:

```
globalp_df %>%
  arrange(global) %>%
  mutate(FDR = p.adjust(global)) %>%
  filter(FDR < 0.3) %>%
  mutate(terms = str_trunc(terms, 35)) %>%
  knitr::kable(digits = 4)
```

terms	Proteomics	RNAseq	global	FDR
REACTOME_CELL_DEATH_SIGNALLING_V...	0e+00	0.1825	1e-04	0.0968
BIOCARTA_NKCELLS_PATHWAY	2e-04	0.0329	1e-04	0.1334
BIOCARTA_EPONFKB_PATHWAY	0e+00	0.0079	1e-04	0.1758
PID_IL2_PI3K_PATHWAY	0e+00	0.0017	1e-04	0.1759
REACTOME_SIGNALING_BY_RHO_GTPASES	0e+00	0.0033	2e-04	0.2096
BIOCARTA_TCR_PATHWAY	0e+00	0.0039	2e-04	0.2171
REACTOME_INTERFERON_GAMMA_SIGNALING	5e-04	0.0229	2e-04	0.2175
ST_T_CELL_SIGNAL_TRANSDUCTION	1e-04	0.0028	2e-04	0.2351
BIOCARTA_RAC1_PATHWAY	1e-04	0.1152	2e-04	0.2673

The first pathway name is truncated, but it is "Cell death signalling via NRAGE, NRIF and NADE" (`REACTOME_CELL_DEATH_SIGNALLING_VIA_NRAGE_NRIF_AND_NADE`). Before, none of the pathways were significant even at FDR < 0.3 when analyzing each type of omics data separately. Using the Global-*p* statistic, we see the above pathways significant have FDR < 0.3. Once again, recall that the results are dependent on the PCs estimated by the elastic net algorithm (which may not be numerically unique). However, the p-values for these pathways are only slightly different from those in the main manuscript.

Because this is the last example with these data sets, we will clean our working environment.

```
rm(list = ls())
```

3.4 Integrating gene expression data with experimental design information: an analysis of sex-specific pathway gene expression effects on kidney cancer

This example is shown in our website's main vignette: https://gabrielodom.github.io/pathwayPCA/articles/Introduction_to_pathwayPCA.html#case-study-analysis-of-studies-with-complex-designs. However, we will show the data cleaning and processing steps here.

Gene Expression Data Import and Wrangling

We will import the gene expression data and transpose it directly.

```
kirpRNAseq_df <- read_delim(
  "extdata/HiSeqV2", "\t", escape_double = FALSE, trim_ws = TRUE
) %>%
  TransposeAssay()

## Parsed with column specification:
## cols(
##   .default = col_double(),
##   sample = col_character()
## )

## See spec(...) for full column specifications.
```

Clinical Data Import and Cleaning

We now import the clinical data.

```
kirpPheno_tbldf <- read_delim(
  "extdata/KIRP_clinicalMatrix", "\t", escape_double = FALSE, trim_ws = TRUE
)

## Parsed with column specification:
## cols(
##   .default = col_character(),
##   `_EVENT` = col_double(),
##   OS.time = col_double(),
##   OS = col_double(),
##   RFS.time = col_double(),
##   RFS = col_double(),
##   `_TIME_TO_EVENT` = col_double(),
##   age_at_initial_pathologic_diagnosis = col_double(),
##   days_to_additional_surgery_locoregional_procedure = col_double(),
##   days_to_additional_surgery_metastatic_procedure = col_double(),
##   days_to_birth = col_double(),
##   days_to_collection = col_double(),
##   days_to_death = col_double(),
##   days_to_initial_pathologic_diagnosis = col_double(),
##   days_to_last_followup = col_double(),
##   days_to_new_tumor_event_after_initial_treatment = col_double(),
##   eastern_cancer_oncology_group = col_double(),
##   height = col_double(),
##   initial_weight = col_double(),
##   intermediate_dimension = col_double(),
##   karnofsky_performance_score = col_double()
##   # ... with 10 more columns
## )

## See spec(...) for full column specifications.
```

We also need to create a column indicating male subjects and removing missing response information.

```
kirpResp_df <-
  kirpPheno_tbldf %>%
  # Select and rename the columns we need
  select(
    Sample = sampleID,
    time = OS.time,
    status = OS,
    gender
  ) %>%
  # Create a male logical indicator variable
  mutate(male = (gender == "MALE")) %>%
  select(-gender) %>%
  # Remove missing values
  na.omit()

rm(kirpPheno_tbldf)
```

Joining the Data

```
kidney_df <- inner_join(kirpResp_df, kirpRNAseq_df)

## Joining, by = "Sample"

kidney_df[1:5, 1:5]

## # A tibble: 5 x 5
##   Sample           time status male  ARHGEF10L
##   <chr>           <dbl>  <dbl> <lgl>     <dbl>
## 1 TCGA-2K-A9WE-01   214      0 TRUE       10.8
## 2 TCGA-2Z-A9J1-01  2298      0 TRUE       10.8
## 3 TCGA-2Z-A9J2-01  1795      0 FALSE      10.4
## 4 TCGA-2Z-A9J3-01  1771      1 TRUE       10.7
## 5 TCGA-2Z-A9J5-01  3050      0 TRUE       10.6

dim(kidney_df)

## [1]   320 20534
```

At this point, we repeat the same steps shown in Section 4 of our main vignette (link above).

3.5 A pathway based integrative prediction model for patient prognosis

The example requires 100+ random subsamples for training and testing purposes. Therefore, this section will only show the code necessary to complete one random training/test data split. We invite the reader to perform other random splits of the data, but note that this can be computationally expensive.

Data Wrangling

Data Import and Sample Intersection

We import the raw gene expression RNAseq - IlluminaHiSeq pancan normalized data, the log-ratio normalised proteomics data, and the clinical data:

```r
# RNAseq
coadRNAseq_tbldf <- read_delim(

"extdata/Human__TCGA_COADREAD__UNC__RNAseq__GA_RNA__01_28_2016__BI__Gene__Firehose_RSEM_l
og2.cct",
  "\t", escape_double = FALSE, trim_ws = TRUE
)

# Proteomics
coadProt_tbldf <- read_delim(

"extdata/Human__TCGA_COADREAD__VU__Proteome__Velos__01_28_2016__VU__Gene__CDAP_UnsharedPr
ecursorArea_r2.cct",
  "\t", escape_double = FALSE, trim_ws = TRUE
)

# Clinical Response
coadPheno_tbldf <- read_delim(

"extdata/Human__TCGA_COADREAD__MS__Clinical__Clinical__01_28_2016__BI__Clinical__Firehose
.tsi",
  "\t", escape_double = FALSE, trim_ws = TRUE
)
```

These data tables are all in long form, so we transpose them. For the clinical data, we remove any samples with missing survival response.

```r
# RNAseq
coadRNAseq_df <- TransposeAssay(
  coadRNAseq_tbldf, omeNames = "firstCol"
)

# Protein
coadProt_df <- TransposeAssay(
  coadProt_tbldf, omeNames = "firstCol"
)

# Clinical
coadPheno_df <-
  TransposeAssay(
    coadPheno_tbldf, omeNames = "firstCol"
  ) %>%
  mutate(
    # These are the required names for the glmnet survival ROC function I wrote
    time = as.numeric(overall_survival),
    status = as.integer(status)
  ) %>%
  select(Sample, time, status) %>%
  drop_na(time, status)
```

Sample Matching

We then find the set of samples shared across both platforms and clinical data.

```
sharedSamples_char <-
  intersect(
    coadProt_df$Sample,
    intersect(
      coadRNAseq_df$Sample,
      coadPheno_df$Sample
    )
  )
```

We select the rows from each data set which match these samples, and we re-order the rows of the clinical data (this assumption will make our analysis faster later: we won't have to join the data by sample within each simulation replicate).

```
coadRNAseqM_df <-
  coadRNAseq_df %>%
  filter(Sample %in% sharedSamples_char)
dim(coadRNAseqM_df)

## [1]   68 6150

coadRNAseqM_df[1:5, 1:5]

## # A tibble: 5 x 5
##   Sample         ELMO2 CREB3L1 RPS11  PNMA1
##   <chr>          <dbl>   <dbl> <dbl>  <dbl>
## 1 TCGA.A6.3807 -0.0536    1.32 0.937 -1.33
## 2 TCGA.A6.3808 -0.774     1.30 0.802 -0.661
## 3 TCGA.A6.3810  0.0895    1.33 1.13  -1.53
## 4 TCGA.AA.3518 -0.735     2.16 1.68  -0.908
## 5 TCGA.AA.3525 -0.921     2.62 0.689 -0.387

coadProtM_df <-
  coadProt_df %>%
  filter(Sample %in% sharedSamples_char) %>%
  arrange(match(Sample, coadRNAseqM_df$Sample))
dim(coadProtM_df)

## [1]   68 5539
```

```
coadProtM_df[1:5, 1:5]

## # A tibble: 5 x 5
##   Sample        A1BG  A1CF   A2M  AAAS
##   <chr>        <dbl> <dbl> <dbl> <dbl>
## 1 TCGA.A6.3807  25.1     0  31.9  24.0
## 2 TCGA.A6.3808  33.4     0  33.1   0
## 3 TCGA.A6.3810  26.5     0  29.6  22.1
## 4 TCGA.AA.3518  31.9     0  33.1  23.4
## 5 TCGA.AA.3525  25.5     0  30.3  22.5

coadPhenoM_df <-
  coadPheno_df %>%
  filter(Sample %in% sharedSamples_char) %>%
  arrange(match(Sample, coadRNAseqM_df$Sample))
coadPhenoM_df

## # A tibble: 68 x 3
##    Sample        time status
##    <chr>        <dbl>  <int>
##  1 TCGA.A6.3807  1054      0
##  2 TCGA.A6.3808  1014      0
##  3 TCGA.A6.3810  1111      0
##  4 TCGA.AA.3518    31      0
##  5 TCGA.AA.3525   245      0
##  6 TCGA.AA.3526   580      0
##  7 TCGA.AA.3531  1035      0
##  8 TCGA.AA.3534   882      0
##  9 TCGA.AA.3552   396      1
## 10 TCGA.AA.3554   546      0
## # … with 58 more rows
```

Missing Data

With this matched-sample data subset, we check for missing values.

```
anyNA(coadRNAseqM_df)

## [1] TRUE

anyNA(coadProtM_df)

## [1] FALSE
```

Because there are missing values in the RNAseq data, we check which genes have missingness.

```
badGenes_char <-
  coadRNAseqM_df %>%
  map_int(~sum(is.na(.x))) %>%
  `>`(0) %>%
  which %>%
  names

badGenes_char

##  [1] "CCDC60"     "KCNA3"      "KIR3DL2"    "PCDHA9"     "C1orf110"   "LIPF"
##  [7] "DEFB127"    "CPLX4"      "KRT26"      "NLRP10"     "OR10A6"     "ASB11"
## [13] "C18orf26"  "TRIML1"    "S100G"     "RPTN"      "HYMAI"     "TRIM73"
## [19] "UNC5D"     "PRKG2"     "LOC200261"
```

There are not many genes with missing values (only 21), so we remove them rather than impute them.

```
coadRNAseqM_df <-
  coadRNAseqM_df %>%
  select(-one_of(badGenes_char))
```

Cleaned Data

Now we make a list of matched dataframes, and clean up our environment:

```
coadMatched_ls <-
  list(
    clinical = coadPhenoM_df,
    proteomics = coadProtM_df,
    rnaSeq = coadRNAseqM_df
  )

rm(
  coadPheno_tbldf, coadPheno_df, coadPhenoM_df,
  coadProt_tbldf, coadProt_df, coadProtM_df,
  coadRNAseq_tbldf, coadRNAseq_df, coadRNAseqM_df,
  badGenes_char
)
```

Finally, we import the chosen pathway collection: the KEGG pathways in C2 Canonical Pathways collection from MSigDB (with pathways larger than 100 genes removed).

```
keggC2CP_PC <- read_gmt(
  file = "extdata/c2.cp.v6.2.symbols_trimmed.gmt",
  description = TRUE
)
```

Training-Test Data Split

Now that we have sample-matched cross-platform data, we will partition the data into training and test subsets, and create `OmicsSurv` data containters for each.

Split Indicators

Because the end goal is to repeat this process over hundreds of random splits, we create a list with training-data indicators for each of the replicates. Because this process could take more than a day, we set the date of the simulation as the date we started. (*For the purpose of this document, we will set the number of runs to 1, but we normally set it at 100 or 500.*)

```
date_char <- format(Sys.Date(), "%Y%m%d")
nRuns <- 1
labels_char <- rep_along(sharedSamples_char, c(TRUE, FALSE))

samples_ls <-
  rerun(
    .n = nRuns,
    {
      tibble(
        Sample = sharedSamples_char,
        Train  = sample(labels_char, replace = FALSE)
      )
    }
  )
names(samples_ls) <- str_pad(
  as.character(seq_len(nRuns)),
  width = 3, side = "left", pad = "0"
)
```

Data Containers and AESPCA Calls

For each set of training-test splits, we will store the centering and scaling vectors (for the prediction step later) and then create the data containers. Once we have these, we will execute AESPCA on the training data. Because these steps can be time consuming, we recommend saving the intermediate results from each run within the `for()` loop.

```
for (run in seq_len(nRuns)) {

  ###  Setup  ###
  run_char <- names(samples_ls)[run]
  samples_df <- samples_ls[[run]]
  # res_dir <- paste0(
  #   "<YOUR PATH HERE>/tcga_coad_surv_results_run_", run_char
  # )
  # dir.create(res_dir)


  ###  Split Index  ###
  train_logi <- samples_df$Train
  trainResp_df <-
    coadMatched_ls$clinical[train_logi, ]


  ###  Store Centring and Scaling Vectors  ###
  centerAndScale_ls <- list(
    protCenter = colMeans(
      coadMatched_ls$proteomics[train_logi, -1]
    ),
    protScale = apply(
      coadMatched_ls$proteomics[train_logi, -1], MARGIN = 2, FUN = sd
    ),
    rnaSeqCenter = colMeans(
      coadMatched_ls$rnaSeq[train_logi, -1]
    ),
    rnaSeqScale = apply(
      coadMatched_ls$rnaSeq[train_logi, -1], MARGIN = 2, FUN = sd
    )
  )


  ###  Create Omics Data Containers  ###
  protTrain_OmicsSurv <- CreateOmics(
    assayData_df = coadMatched_ls$proteomics[train_logi, ],
    pathwayCollection_ls = keggC2CP_PC,
```

```r
  response = trainResp_df,
  respType = "surv"
)
rnaSeqTrain_OmicsSurv <- CreateOmics(
  assayData_df = coadMatched_ls$rnaSeq[train_logi, ],
  pathwayCollection_ls = keggC2CP_PC,
  response = trainResp_df,
  respType = "surv"
)


###  AES-PCA  ###
# Each step takes a while:
# 4.8 min on FIU PC; 5.9 on UM PC; 12.1 on mac
protTrain_aespcOut <- AESPCA_pVals(
  protTrain_OmicsSurv,
  parallel = TRUE,
  numCores = 16,
  adjustment = "BH"
)
rnaSeqTrain_aespcOut <- AESPCA_pVals(
  rnaSeqTrain_OmicsSurv,
  parallel = TRUE,
  numCores = 16,
  adjustment = "BH"
)


###  Compile Results  ###
trainingData_ls <- list(
  trainingIDs = samples_df,
  response_df = trainResp_df
)
coadProtTrainResults_ls <- list(
  aespcaRes_ls = protTrain_aespcOut,
  cs_ls = centerAndScale_ls[c("protCenter", "protScale")]
)
coadRNAseqTrainResults_ls <- list(
  aespcaRes_ls = rnaSeqTrain_aespcOut,
  cs_ls = centerAndScale_ls[c("rnaSeqCenter", "rnaSeqScale")]
)


###  Save  ###
# saveRDS(
#   trainingData_ls,
#   file = paste0(
#     res_dir, "/pathwayPCA_tcga_coad_trainMeta_res_", date_char, ".RDS"
#   )
# )
# saveRDS(
#   coadProtTrainResults_ls,
#   file = paste0(
#     res_dir, "/pathwayPCA_tcga_coad_protTrain_res_", date_char, ".RDS"
#   )
```

```
  # )
  # saveRDS(
  #   coadRNAseqTrainResults_ls,
  #   file = paste0(
  #     res_dir, "/pathwayPCA_tcga_coad_rnaSeqTrain_res_", date_char, ".RDS"
  #   )
  # )

}

## 17 genes have variance < epsilon and will be removed. These gene(s) are:

##  [1] "C6orf203" "COA5"     "CYP3A5"   "DNM3"     "EOGT"     "GGA2"
##  [7] "HSPB8"    "INSL5"    "LYSMD2"   "PTGDS"    "PTGIR"    "RARRES1"
## [13] "SGCE"     "TMOD2"    "TRIAP1"   "ZNF563"   "ZNF787"

##
##   ======  Creating object of class OmicsSurv  =======

## The input pathway database included 7075 unique features.

## The input assay dataset included 5521 features.

## Only pathways with at least 3 or more features included in the assay dataset are
##   tested (specified by minPathSize parameter). There are 1118 pathways which meet
##   this criterion.

## Because pathwayPCA is a self-contained test (PMID: 17303618), only features in
##   both assay data and pathway database are considered for analysis. There are 2839
##   such features shared by the input assay and pathway database.

## 24 gene name(s) are invalid. Invalid name(s) are:

##  [1] "RP5-1077B9.4"  "tcag7.1260"    "hCG_1990170"   "tcag7.23"
##  [5] "RP6-213H19.1"  "RP11-217H1.1"  "RP4-747L4.3"   "hCG_18290"
##  [9] "RP5-1022P6.2"  "CX40.1"        "7A5"           "CTB-1048E9.5"
## [13] "RP11-413M3.2"  "RP11-679B17.1" "RP13-36C9.6"   "RP11-125A7.3"
## [17] "hCG_31916"     "RP13-122B23.3" "hCG_16001"     "RP11-35N6.1"
## [21] "hCG_2001000"   "NOP5/NOP58"    "tcag7.1017"    "tcag7.1015"

## These genes may be excluded from analysis. Proper gene names
## contain alphanumeric characters only, and start with a letter.

##
##   ======  Creating object of class OmicsSurv  =======

## The input pathway database included 7075 unique features.

## The input assay dataset included 6128 features.

## Only pathways with at least 3 or more features included in the assay dataset are
##   tested (specified by minPathSize parameter). There are 1136 pathways which meet
##   this criterion.

## Because pathwayPCA is a self-contained test (PMID: 17303618), only features in
##   both assay data and pathway database are considered for analysis. There are 2152
##   such features shared by the input assay and pathway database.

## Part 1: Calculate Pathway AES-PCs
## Initializing Computing Cluster: DONE
## Extracting Pathway PCs in Parallel: DONE
##
## Part 2: Calculate Pathway p-Values
## Initializing Computing Cluster: DONE
## Extracting Pathway p-Values in Parallel: DONE
##
## Part 3: Adjusting p-Values and Sorting Pathway p-Value Data Frame
## DONE
## Part 1: Calculate Pathway AES-PCs
## Initializing Computing Cluster: DONE
## Extracting Pathway PCs in Parallel: DONE
##
## Part 2: Calculate Pathway p-Values
## Initializing Computing Cluster: DONE
## Extracting Pathway p-Values in Parallel: DONE
##
## Part 3: Adjusting p-Values and Sorting Pathway p-Value Data Frame
## DONE
```

Because we have created a lot of extra objects for this process, let's clean up the environment.

```
rm(
  labels_char, sharedSamples_char,
  samples_df, trainResp_df,
  protTrain_OmicsSurv, protTrain_aespcOut,
  rnaSeqTrain_OmicsSurv, rnaSeqTrain_aespcOut,
  centerAndScale_ls, train_logi
)
```

Build Training-Data Design Matrices

We now have `pathwayPCA` results for both platforms, so we want to create design matrices to train the Survival GLMNET model. **From this point on, we will assume that you are working within an individual replicate of a `for()` loop, other than to define functions.**

Extract Designs

We first create a helper function to extract a design and response matrix from the `pathwayPCA` output.

```
ExtractDesign <- function(aespcaResults_ls,
                          response_df,
                          platform = c("prot", "gene")){
  # Input:
  #   - aespcaResults_ls: the output of the  AESPCA_pVals() function
  #   - response_df: the data frame of survival information
  #   - platform: tag indicating if the results are from proteomics or RNAseq
  # Output: a data frame with the survival information and principal component
  #   vectors from each pathway.

  design_df <- bind_cols(aespcaResults_ls$PCs_ls)
  colnames(design_df) <- paste(
    names(aespcaResults_ls$PCs_ls), platform, sep = "_"
  )
  bind_cols(
    response_df,
    design_df
  )

}
```

We then create the design matrices that compile the pathway principal components from each platform.

```
protPCs_df <- ExtractDesign(
  aespcaResults_ls = coadProtTrainResults_ls$aespcaRes_ls,
  response_df = trainingData_ls$response_df,
  platform = "prot"
)

rnaSeqPCs_df <- ExtractDesign(
  aespcaResults_ls = coadRNAseqTrainResults_ls$aespcaRes_ls,
  response_df = trainingData_ls$response_df,
  platform = "gene"
)
```

Remove Noisy Pathway Principal Components

We already know that some of these pathways will not be related to the survival rates of colon cancer. Moreover, we have pathway-specific p-values from a Cox Proportional Hazards model. Therefore, we can rank the pathway principal components by their p-values and retain all in the top, say 200 (this number can obviously be adjusted, but remember to adhere to the recommended practices of the Elastic-Net; because there are only 34 training observation, the number of features in the design space should not be too large).

We first create the vectors of pathway-specific p-values.

```
protPVs <- coadProtTrainResults_ls$aespcaRes_ls$pVals_df$rawp
names(protPVs) <- paste0(
  coadProtTrainResults_ls$aespcaRes_ls$pVals_df$pathways, "_prot"
)

genePVs <- coadRNAseqTrainResults_ls$aespcaRes_ls$pVals_df$rawp
names(genePVs) <- paste0(
  coadRNAseqTrainResults_ls$aespcaRes_ls$pVals_df$pathways, "_gene"
)

allPVs <- sort(c(protPVs, genePVs))
```

From the output of the `AESPCA_pVals()` function, these p-values are already sorted in increasing order. We now cut to the top pathways within each platform and across both.

```
keptPaths_idx <- seq_len(200)

keepProtPCs <- names(protPVs[keptPaths_idx])
keepGenePCs <- names(genePVs[keptPaths_idx])
keepAllPCs  <- names(allPVs[keptPaths_idx])
```

Finally, we create the design matrices for each platform and for their combination.

```
protDesign_df <- protPCs_df %>%
  select(Sample, time, status, one_of(keepProtPCs))
rnaSeqDesign_df <- rnaSeqPCs_df %>%
```

```r
  select(Sample, time, status, one_of(keepGenePCs))

combinedDesign_df <- bind_cols(
  protPCs_df %>%
    select(Sample, time, status, one_of(keepAllPCs)),
  rnaSeqPCs_df %>%
    select(one_of(keepAllPCs))
)
```

Each of the three design matrices are the same dimension:

```r
ncol(protDesign_df); ncol(rnaSeqDesign_df); ncol(combinedDesign_df)

## [1] 203

## [1] 203

## [1] 203

# Clean up
rm(
  protPVs, genePVs, allPVs, keptPaths_idx,
  protPCs_df, rnaSeqPCs_df
)
```

Finally, because we are assuming that this process is being replicated hundreds of times, we will also save this set of intermediary results.

```r
# write_csv(
#   protDesign_df,
#   path = paste0(
#     res_dir,
#     "/tcga_coad_protTrain_Design_nKpt", nKept, "_",
#     resDate_char, ".csv"
#   )
# )
#
# write_csv(
#   rnaSeqDesign_df,
#   path = paste0(
#     res_dir,
#     "/tcga_coad_rnaSeqTrain_Design_nKpt", nKept, "_",
#     resDate_char, ".csv"
#   )
# )
#
# write_csv(
#   combinedDesign_df,
#   path = paste0(
#     res_dir,
#     "/tcga_coad_protRNAseqTrain_Design_nKpt", nKept, "_",
#     resDate_char, ".csv"
#   )
# )
```

Project the Test Data

We have a training design matrix for each platform. Now, we want to load the test data onto these PCs. First we set the test data sample indices.

```r
test_logi <- !trainingData_ls$trainingIDs$Train
```

Load Proteomics Test Data onto Training-Data PCs

We use the pathwayPCA function `LoadOntoPCs()` to project the training data, pathway-specific, onto their respective principal components.

```r
protTest_df <-
  # Select the test samples
  coadMatched_ls$proteomics[test_logi, -1] %>%
  # Center and scale them to the training data, resulting in a matrix
  scale(
    center = coadProtTrainResults_ls$cs_ls$protCenter,
    scale  = coadProtTrainResults_ls$cs_ls$protScale
  ) %>%
  # Transform back into a data frame and append the sample IDs
  as_tibble() %>%
  bind_cols(
    coadMatched_ls$proteomics[test_logi, 1],
    .
  ) %>%
  # Project the test data onto each pathway PC
  LoadOntoPCs(
    loadings_ls = coadProtTrainResults_ls$aespcaRes_ls$loadings_ls
  ) %>%
  # Remove the sample IDs (we kept the rows in the appropriate order)
  select(-SampleID)

colnames(protTest_df) <- paste(colnames(protTest_df), "prot", sep = "_")

protTest_df <-
  coadMatched_ls$clinical %>%
  filter(test_logi) %>%
  bind_cols(protTest_df) %>%
  select(Sample, time, status, one_of(keepProtPCs))
```

Load Gene Expression Test Data onto Training-Data PCs

We repeat the above process for gene expression data.

```r
rnaSeqTest_df <-
  coadMatched_ls$rnaSeq[test_logi, -1] %>%
  scale(
    center = coadRNAseqTrainResults_ls$cs_ls$rnaSeqCenter,
    scale  = coadRNAseqTrainResults_ls$cs_ls$rnaSeqScale
  ) %>%
  as_tibble() %>%
  bind_cols(
    coadMatched_ls$rnaSeq[test_logi, 1],
    .
  ) %>%
  LoadOntoPCs(
    loadings_ls = coadRNAseqTrainResults_ls$aespcaRes_ls$loadings_ls
  ) %>%
  select(-SampleID)

colnames(rnaSeqTest_df) <- paste(colnames(rnaSeqTest_df), "gene", sep = "_")

rnaSeqTest_df <-
  coadMatched_ls$clinical %>%
  filter(test_logi) %>%
  bind_cols(rnaSeqTest_df) %>%
  select(Sample, time, status, one_of(keepGenePCs))
```

Combine Test Data across Platforms

We also create the cross-platform test data.

```r
combinedTest_df <- bind_cols(
  protTest_df %>%
    select(Sample, time, status, one_of(keepAllPCs)),
  rnaSeqTest_df %>%
    select(one_of(keepAllPCs))
)
```

At this point, we would save another round of intermediary results.

```r
# write_csv(
#   protTest_df,
#   path = paste0(
#     res_dir,
#     "/tcga_coad_protTest_Design_nKpt", nKept, "_",
#     resDate_char, ".csv"
#   )
# )
# write_csv(
#   rnaSeqTest_df,
#   path = paste0(
#     res_dir,
#     "/tcga_coad_rnaSeqTest_Design_nKpt", nKept, "_",
#     resDate_char, ".csv"
#   )
# )
# write_csv(
#   combinedTest_df,
#   path = paste0(
#     res_dir,
#     "/tcga_coad_protRNAseqTest_Design_nKpt", nKept, "_",
#     resDate_char, ".csv"
#   )
# )
```

Survival GLMNET Prediction

Helper Functions

`ElasticNetReduce()`

We need a function to estimate the elastic net and identify the necessary features “caught” in this net.

```
ElasticNetReduce <- function(fullTrain_df, fullTest_df){
  # Input:
  #   - fullTrain_df: a data frame of the survival time and event status (named
  #       "time" and "status" exactly) and the PCs from the top 200 most
  #       significant pathways and NO OTHER COLUMNS (this function uses column
  #       index positions for subsetting).
  #   - fullTest_df: a data frame of the same format as fullTrain_df
  #
  # Output: a list of three components:
  #   - redTrain: a data frame with survival time, event status, and pathway PCs
  #       selected by the elastic net from the training data
  #   - redTest: a data frame with the same format as redTrain
  #   - modelBetas: the pathway PC coefficients from the glmnet of best fit

  ###  Train CoxPH Elastic Net  ###
  require(glmnet)
  # The results of the cv.glmnet() are highly variable, so we repeat the model
  #   fit and find the best results.
  fits_ls <- map(
    .x = 1:10,
    .f = ~{
      cv.glmnet(
        as.matrix(fullTrain_df[, -(1:2)]),
        Surv(fullTrain_df$time, fullTrain_df$status),
        family = "cox", alpha = 0.5
      )
    }
  )


  ###  Best-Fitting Model  ###
  # Average of all of the mean cross-validated errors at each lambda
  meanCVE <- fits_ls %>%
    map("cvm") %>%
    map(t) %>%
    do.call(rbind, .) %>%
    colMeans()

  # Note that lambda values are fixed beforehand, so the lambda values for one
  #   model will be identical to the lambda values for all the others. This also
  #   means that once we pick a lambda, the features selected as significant
  #   will not change across model fits
  bestLambda <- fits_ls[[1]]$lambda[which.min(meanCVE)]
  betas_df <- summary(coef(fits_ls[[1]], s = bestLambda))[, -2]


  ###  Train and Test Data Subsets  ###
```

```r
  # If the elastic-net returns no predictors, then betas_df will have 0 rows.
  # If we have no predictors in the model, we need to find the model with the
  #   next smallest set of predictors:
  if(nrow(betas_df) == 0){

    idx <- 2
    while(nrow(betas_df) == 0){

      betas_df <- summary(
        coef(fits_ls[[1]], s = fits_ls[[1]]$lambda[idx])
      )[, -2]
      idx <- idx + 1

    }

  }

  # Now that we have a set of betas, we extract the pathway columns matching
  #   the features retained by the elastic net.
  redTrain_df <- fullTrain_df[, c(1, 2, betas_df$i + 2)]
  redTest_df  <- fullTest_df[, c(1, 2, betas_df$i + 2)]


  ###  Return  ###
  list(
    redTrain = redTrain_df,
    redTest = redTest_df,
    modelBetas = betas_df$x
  )

}
```

We test the function (because this function is internal, it assumes that we have already removed the sample IDs):

```r
reducedProtData_ls <- ElasticNetReduce(
  fullTrain_df = protDesign_df[, -1],
  fullTest_df = protTest_df[, -1]
)
```

```
## Loading required package: glmnet

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 3.0-2
```

```r
reducedProtData_ls
```

```
## $redTrain
## # A tibble: 34 x 4
##      time status path1152_prot path184_prot
##     <dbl>  <int>         <dbl>        <dbl>
##   1  1014      0       -0.886        -1.96
##   2   245      0       -1.59         -2.10
##   3   580      0        0.0309        0.947
##   4  1035      0        2.34          1.40
##   5   396      1       -0.549        -0.877
##   6   546      0        2.81          0.866
##   7   638      0        1.22          0.109
##   8   424      0        0.852         1.92
##   9   579      1       -0.163        -3.99
## 10   306      1        0.701         0.406
## # … with 24 more rows
##
## $redTest
## # A tibble: 34 x 4
##      time status path1152_prot path184_prot
##     <dbl>  <int>         <dbl>        <dbl>
##   1  1054      0       -0.841        -0.724
##   2  1111      0       -2.28         -1.16
##   3    31      0        1.41          0.935
##   4   882      0       -0.0307       -0.331
##   5  1643      0        0.626         1.64
##   6    61      1        1.56          1.01
##   7   821      0        1.84          0.919
##   8    30      1        1.15          0.862
##   9  1612      0       -0.493         0.226
## 10   580      0       -0.0758       -0.367
## # … with 24 more rows
##
## $modelBetas
## [1] -0.007203762 -0.011347740
```

It is worth noting that principal components are unique up to a sign, so we may see positive or negative values for the **β** calculated by the GLMNET.

PredictCoxSurvival()

Given reduced training and test data, we predict survival.

```
PredictCoxSurvival <- function(reducedTrain_df, reducedTest_df, modelCoeff_num,
                               predictor = "lp"){
  # Input:
  #   - reducedTrain_df: the "redTrain" object returned by ElasticNetReduce()
  #   - reducedTest_df: the "redTest" object returned by ElasticNetReduce()
  #   - modelCoeff_num: the "modelBetas" object returned by ElasticNetReduce()
  #   - predictor: the type of predictor, as passed to the predict.coxph()
  #       function. Options are "lp", "risk", "expected", "terms", "survival".
  #       Defaults to the linear predictor. See ?predict.coxph, argument "type"
  #       for more details.
  #
  # Output: a list of two data frames (one for training data, one for test) with
  #   observed survival time and status paired with predicted survival time

  ###  Train Reduced Cox PH Model  ###
  # We pass in the intial values for beta and "iter = 0" to train the model
  #   exactly as it was returned by the GLMNET. See
  # https://r.789695.n4.nabble.com/estimating-survival-times-with-glmnet-and-coxph-
td4614225.html
  coxph_train <- coxph(
    Surv(time, status) ~ .,
    data = reducedTrain_df,
    init = modelCoeff_num,
    iter = 0
  )

  ###  Predictions  ###
  # Obtain the linear predictor from the training data
  trainPred <- predict(coxph_train, type = predictor)

  # predict on the test data
  testSurvCols <- which(colnames(reducedTest_df) %in% c("time", "status"))
  testPred <- predict(
    coxph_train,
    newdata = reducedTest_df[, -testSurvCols],
    type = predictor
  )

  ###  Return  ###
  list(
    trainPrediction = data.frame(
      reducedTrain_df[, c("time", "status")],
      predicted = trainPred
    ),
    testPrediction = data.frame(
      reducedTest_df[, c("time", "status")],
      predicted = testPred
    )
  )

}
```

We test this function on the output returned by the `ElasticNetReduce()` function above. See the `predict.coxph()` manual files for more details.

```
predProtSurv_ls <- PredictCoxSurvival(
  reducedTrain_df = reducedProtData_ls$redTrain,
  reducedTest_df = reducedProtData_ls$redTest,
  modelCoeff_num = reducedProtData_ls$modelBetas
)
```

`WrangleTimeROC()`

Now that we have a set of relative risks from the Cox Proportional Hazards predicton, we can predict survival at different times.

```
WrangleTimeROC <- function(testPred_df, timePts = 8, pctile = 0.8){
  # Input:
  #   - testPred_df: the "testPrediction" data frame returned by the
  #       PredictCoxSurvival() function
  #   - timePts: number of time points to predict survival and calculate ROC
  #   - pctile: calculate ROC up to this percent of the subjects. Because
  #       survival times often have long tails, this should be somewhere less
  #       than 90%.
  #
  # Output: a data frame of the time points, the true positives and false
  #   positives at that time point (TP and FP, respectively), and the AUC value
  #   for the prediction at that time point.

  require(timeROC)

  timeSteps <- quantile(
    testPred_df$time,
    probs = seq(0, pctile, length.out = timePts + 1)[-1]
  )

  # The help file states that higher marker values are assumed to be related to
  #   *lower* survival values. We observed that our predicted risk values have
  #   the opposite interpretation. Therefore, we set the marker to -1 times its
  #   true predicted value.
  data_sROC <- timeROC(
    T = testPred_df$time,
    delta = testPred_df$status,
    marker = -testPred_df$predicted,
    times = timeSteps,
    cause = 1,
    weighting = "cox"
  )

  dataSROC_ls <- list(
    tp_df = data_sROC$TP %>%
      as_tibble() %>%
      gather(key = "time", value = "TP"),
    fp_df = data_sROC$FP %>%
      as_tibble() %>%
      gather(key = "time", value = "FP") %>%
      select(FP),
    auc_df = tibble(
      time = names(data_sROC$AUC),
      auc  = data_sROC$AUC
    )
  )

  dataSROC_ls$tp_df %>%
    bind_cols(dataSROC_ls$fp_df) %>%
    left_join(dataSROC_ls$auc_df) %>%
    mutate(time = str_remove(time, "t=")) %>%
    mutate(time = as.numeric(time))

}
```

We test this function.

```
WrangleTimeROC(
  testPred_df = predProtSurv_ls$testPrediction
)
```

```
## Loading required package: timeROC
## Joining, by = "time"
```

```
## # A tibble: 280 x 4
##     time    TP     FP   auc
##    <dbl> <dbl>  <dbl> <dbl>
##  1    40     0 0      0.725
##  2    40     0 0.0348 0.725
##  3    40     0 0.0695 0.725
##  4    40     0 0.104  0.725
##  5    40     0 0.138  0.725
##  6    40     0 0.173  0.725
##  7    40     0 0.207  0.725
##  8    40     0 0.241  0.725
##  9    40     0 0.241  0.725
## 10    40     0 0.275  0.725
## # … with 270 more rows
```

The Wrapper Function

Now that we have these three internal functions, we wrap them.

```r
GlmnetSurvivalROC <- function(train_df, test_df, predictor_char = "lp", ...){
  # Input:
  #   - train_df: a data frame of the survival time and event status (named
  #       "time" and "status" exactly) and the PCs from the top 200 most
  #       significant pathways and NO OTHER COLUMNS (this function uses column
  #       index positions for subsetting).
  #   - test_df: a data frame of the same format as fullTrain_df
  #   - predictor: the type of predictor, as passed to the predict.coxph()
  #       function. Options are "lp", "risk", "expected", "terms", "survival".
  #       Defaults to the linear predictor. See ?predict.coxph, argument "type"
  #       for more details.
  #   - ...: dots for additional arguments to the WrangleTimeROC() function
  #
  # Output: a list of two components:
  #   - predictorCount_df: a data frame counting the number of pathways used
  #       to predict survival outcome by protein (nProteinPaths) or gene
  #       expression (nRNAseqPaths). If you supply a combined design matrix with
  #       both RNAseq and proteomics data, this will count how many pathways
  #       from each platform were "captured" by the elastic net.
  #   - survivalROC_df: a data frame of the time points, the true positives and
  #       false positives at that time point (TP and FP, respectively), and the
  #       AUC value for the prediction at that time point.
  #
  # Details: this is a wrapper function for the ElasticNetReduce(),
  #   PredictCoxSurvival(), and WrangleTimeROC() functions.

  redData_ls <- ElasticNetReduce(
    fullTrain_df = train_df,
    fullTest_df  = test_df
  )
  keptPathTypes_char <- str_split(
    colnames(redData_ls$redTest[, -(1:2)]),
    pattern = "_"
  ) %>%
    map_chr(2)
  predSurv_ls <- PredictCoxSurvival(
    reducedTrain_df = redData_ls$redTrain,
    reducedTest_df  = redData_ls$redTest,
    modelCoeff_num  = redData_ls$modelBetas,
    predictor       = predictor_char
  )

  aucOut_df <- WrangleTimeROC(
    testPred_df = predSurv_ls$testPrediction, ...
  )

  list(
    predictorCount_df = tibble(
      nProteinPaths = sum(keptPathTypes_char == "prot"),
      nRNAseqPaths  = sum(keptPathTypes_char == "gene")
    ),
    survivalROC_df = aucOut_df
  )

}
```

And we finally test the wrapper function.

```
GlmnetSurvivalROC(
  train_df = protDesign_df[, -1],
  test_df = protTest_df[, -1],
  timePts = 6
)

## Joining, by = "time"

## $predictorCount_df
## # A tibble: 1 x 2
##   nProteinPaths nRNAseqPaths
##           <int>        <int>
## 1             2            0
##
## $survivalROC_df
## # A tibble: 210 x 4
##     time    TP     FP   auc
##    <dbl> <dbl>  <dbl> <dbl>
##  1  85.4 0     0      0.807
##  2  85.4 0     0.0360 0.807
##  3  85.4 0     0.0719 0.807
##  4  85.4 0     0.108  0.807
##  5  85.4 0     0.143  0.807
##  6  85.4 0.531 0.143  0.807
##  7  85.4 0.531 0.179  0.807
##  8  85.4 0.531 0.214  0.807
##  9  85.4 0.531 0.214  0.807
## 10  85.4 0.531 0.249  0.807
## # … with 200 more rows
```

Elastic-Net Results

We have all of the functions we need to predict survival times for the three platforms.

```
# Proteomics
protROC_ls <-
  GlmnetSurvivalROC(
    train_df = protDesign_df[, -1],
    test_df  = protTest_df[, -1],
    timePts = 12
  )

# RNAseq
rnaSeqROC_ls <-
  GlmnetSurvivalROC(
    train_df = rnaSeqDesign_df[, -1],
    test_df  = rnaSeqTest_df[, -1],
    timePts = 12
  )

# Combined
comboROC_ls <-
  GlmnetSurvivalROC(
    train_df = combinedDesign_df[, -1],
    test_df  = combinedTest_df[, -1],
    timePts = 12
  )
```

We can inspect how many protein and RNAseq pathways were selected in the GLMNET for this particular random training-test split:

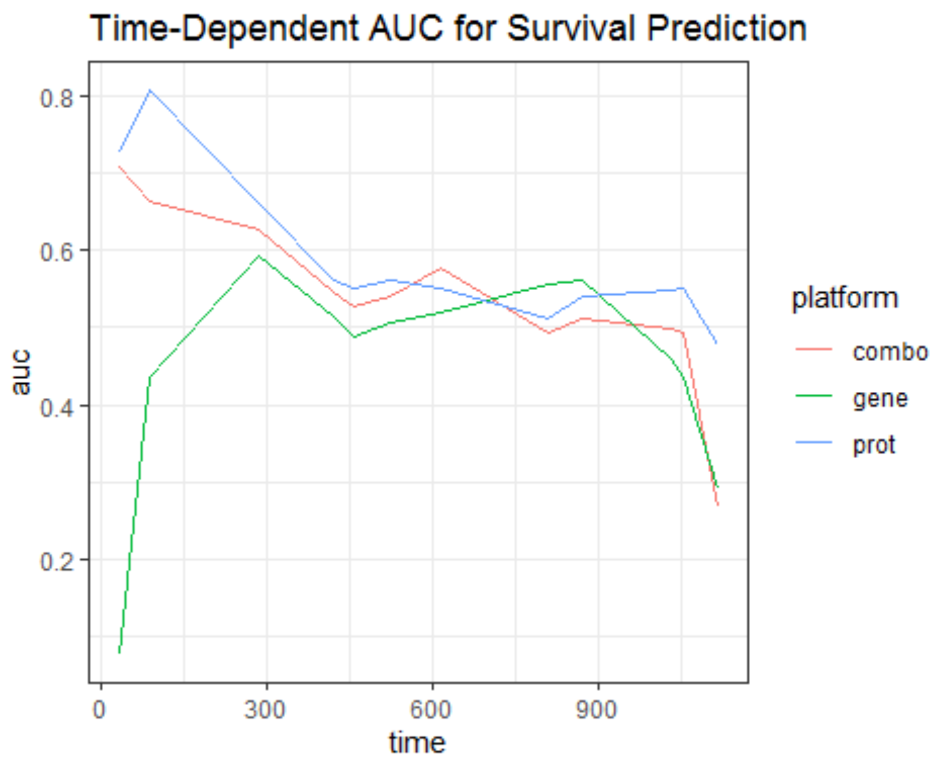
```
comboROC_ls$predictorCount_df

## # A tibble: 1 x 2
##   nProteinPaths nRNAseqPaths
##           <int>        <int>
## 1             3           10
```

We can also look at the prediction accuracy over time. First we create a data frame with all of the AUC values for each platform:

```
timeAUC_df <-
  list(
    prot = protROC_ls,
    gene = rnaSeqROC_ls,
    combo = comboROC_ls
  ) %>%
  map_dfr(2, .id = "platform") %>%
  select(-TP, -FP) %>%
  arrange(time)
```

Now for a graph:

```
ggplot(timeAUC_df) +
  theme_bw() +
  aes(x = time, y = auc, group = platform, colour = platform) +
  labs(title = "Time-Dependent AUC for Survival Prediction") +
  geom_line()
```



At this point, we would repeat this entire process a few hundred times and inspect the results for different random training and test partitions of the data.