

# Package ‘PathoStat’

March 30, 2021

**Type** Package

**Title** PathoStat Statistical Microbiome Analysis Package

**Version** 1.16.0

**Date** 2020-03-27

**Author** Solaiappan Manimaran <manimaran\_1975@hotmail.com>, Matthew Bendall <bendall@gwmail.gwu.edu>, Sandro Valenzuela Diaz <sandrolvalenzuelad@gmail.com>, Eduardo Castro <castronallar@gmail.com>, Tyler Faits <tfaits@gmail.com>, Yue Zhao <jasonzhao0307@gmail.com>, Anthony Nicholas Federico <anfed@bu.edu>, W. Evan Johnson <wej@bu.edu>

**Maintainer** Solaiappan Manimaran <manimaran\_1975@hotmail.com>, Yue Zhao <jasonzhao0307@gmail.com>

**Description** The purpose of this package is to perform Statistical Microbiome Analysis on metagenomics results from sequencing data samples. In particular, it supports analyses on the PathoScope generated report files. PathoStat provides various functionalities including Relative Abundance charts, Diversity estimates and plots, tests of Differential Abundance, Time Series visualization, and Core OTU analysis.

**URL** <https://github.com/mani2012/PathoStat>

**BugReports** <https://github.com/mani2012/PathoStat/issues>

**License** GPL (>= 2)

**Depends** R (>= 3.5)

**Imports** limma, corpcor, matrixStats, reshape2, scales, ggplot2, rentrez, DT, tidyr, plyr, dplyr, phyloseq, shiny, stats, methods, XML, graphics, utils, BiocStyle, edgeR, DESeq2, ComplexHeatmap, plotly, webshot, vegan, shinyjs, glmnet, gmodels, ROCR, RColorBrewer, knitr, devtools, ape

**Collate** 'pathoStat.R' 'utils.R' 'taxonomy.R' 'biomarker.R' 'allClasses.R' 'visualization.R' 'differentialAnalysis.R'

**biocViews** Microbiome, Metagenomics, GraphAndNetwork, Microarray, PatternLogic, PrincipalComponent, Sequencing, Software, Visualization, RNASeq, ImmunoOncology

**RoxygenNote** 6.1.1  
**Encoding** UTF-8  
**Suggests** rmarkdown, testthat  
**VignetteBuilder** knitr  
**git\_url** <https://git.bioconductor.org/packages/PathoStat>  
**git\_branch** RELEASE\_3\_12  
**git\_last\_commit** d3ef0ce  
**git\_last\_commit\_date** 2020-10-27  
**Date/Publication** 2021-03-29

## R topics documented:

Bootstrap_LOOCV_LR_AUC . . . . .	3
Chisq_Test_Pam . . . . .	3
findRAfromCount . . . . .	4
findTaxonMat . . . . .	4
findTaxonomy . . . . .	5
findTaxonomy300 . . . . .	6
Fisher_Test_Pam . . . . .	6
formatTaxTable . . . . .	7
getShinyInput . . . . .	7
getShinyInputCombat . . . . .	8
getShinyInputOrig . . . . .	8
getSignatureFromMultipleGlmnet . . . . .	9
GET_PAM . . . . .	9
grepTid . . . . .	10
loadPathoscopeReports . . . . .	10
loadPstat . . . . .	11
log2CPM . . . . .	12
LOOAUC_simple_multiple_noplot_one_df . . . . .	12
LOOAUC_simple_multiple_one_df . . . . .	13
PathoStat-class . . . . .	13
percent . . . . .	14
phyloseq_to_edgeR . . . . .	15
plotPCAPlotly . . . . .	15
plotPCoAPlotly . . . . .	16
pstat_data . . . . .	17
readPathoscopeData . . . . .	18
runPathoStat . . . . .	18
savePstat . . . . .	19
setShinyInput . . . . .	20
setShinyInputCombat . . . . .	20
setShinyInputOrig . . . . .	21
summarizeTable . . . . .	21
TranslateIdToTaxLevel . . . . .	22
Wilcox_Test_df . . . . .	22
<b>Index</b>	<b>24</b>

---

 Bootstrap\_LOOCV\_LR\_AUC

*Do bootstrap and LOOCV*


---

**Description**

Do bootstrap and LOOCV

**Usage**

```
Bootstrap_LOOCV_LR_AUC(df, targetVec, nboot = 50)
```

**Arguments**

df	Row is sample, column is feature. Required
targetVec	y vector. Required
nboot	number of BOOTSTRAP

**Value**

bootstrap loocv result dataframe

**Examples**

```
data('iris')
Bootstrap_LOOCV_LR_AUC(iris[,1:4],
c(rep(1,100), rep(0,50)), nboot = 3)
```

---

 Chisq\_Test\_Pam

*Given PAM and disease/control annotation, do Chi-square test for each row of PAM*


---

**Description**

Given PAM and disease/control annotation, do Chi-square test for each row of PAM

**Usage**

```
Chisq_Test_Pam(pam, label.vec.num, pvalue.cutoff = 0.05)
```

**Arguments**

pam	Input data object that contains the data to be tested. Required
label.vec.num	The target binary condition. Required
pvalue.cutoff	choose p-value cut-off

**Value**

df.output object

**Examples**

```
tmp <- matrix(rbinom(12,1,0.5), nrow = 3)
rownames(tmp) <- c("a", "b", "c")
Chisq_Test_Pam(tmp, c(1,1,0,0))
```

---

findRAfromCount	<i>Return the Relative Abundance (RA) data for the given count OTU table</i>
-----------------	--

---

**Description**

Return the Relative Abundance (RA) data for the given count OTU table

**Usage**

```
findRAfromCount(count_otu)
```

**Arguments**

count\_otu      Count OTU table

**Value**

ra\_otu Relative Abundance (RA) OTU table

**Examples**

```
data_dir <- system.file("data", package = "PathoStat")
infileName <- "pstat_data.rda"
pstat_test <- loadPstat(data_dir, infileName)
ra_otu <- findRAfromCount(phyloseq::otu_table(pstat_test))
```

---

findTaxonMat	<i>Find the Taxonomy Information Matrix</i>
--------------	---

---

**Description**

Find the Taxonomy Information Matrix

**Usage**

```
findTaxonMat(names, taxonLevels)
```

**Arguments**

names            Row names of the taxonomy matrix  
taxonLevels      Taxon Levels of all tids

**Value**

taxmat Taxonomy Information Matrix

**Examples**

```
example_data_dir <- system.file("example/data", package = "PathoStat")
pathoreport_file_suffix <- "-sam-report.tsv"
datlist <- readPathoscopeData(example_data_dir, pathoreport_file_suffix,
input.files.name.vec = as.character(1:6))
dat <- datlist$data
ids <- rownames(dat)
tids <- unlist(lapply(ids, FUN = grepTid))
# taxonLevels <- findTaxonomy(tids[1:5])
# taxmat <- findTaxonMat(ids[1:5], taxonLevels)
```

---

findTaxonomy	<i>Find the taxonomy for unlimited tids</i>
--------------	---

---

**Description**

Find the taxonomy for unlimited tids

**Usage**

```
findTaxonomy(tids)
```

**Arguments**

tids            Given taxonomy ids

**Value**

taxondata Data with the taxonomy information

**Examples**

```
example_data_dir <- system.file("example/data", package = "PathoStat")
pathoreport_file_suffix <- "-sam-report.tsv"
datlist <- readPathoscopeData(example_data_dir, pathoreport_file_suffix,
input.files.name.vec = as.character(1:6))
dat <- datlist$data
ids <- rownames(dat)
tids <- unlist(lapply(ids, FUN = grepTid))
# taxonLevels <- findTaxonomy(tids[1:5])
```

---

findTaxonomy300	<i>Find the taxonomy for maximum 300 tids</i>
-----------------	---

---

**Description**

Find the taxonomy for maximum 300 tids

**Usage**

```
findTaxonomy300(tids)
```

**Arguments**

tids	Given taxonomy ids
------	--------------------

**Value**

taxondata Data with the taxonomy information

**Examples**

```
example_data_dir <- system.file("example/data", package = "PathoStat")
pathoreport_file_suffix <- "-sam-report.tsv"
datlist <- readPathoscopeData(example_data_dir,
  pathoreport_file_suffix, input.files.name.vec = as.character(1:6))
dat <- datlist$data
ids <- rownames(dat)
tids <- unlist(lapply(ids, FUN = grepTid))
# taxonLevels <- findTaxonomy300(tids[1:5])
```

---

Fisher_Test_Pam	<i>Given PAM and disease/control annotation, do Chi-square test for each row of PAM</i>
-----------------	---

---

**Description**

Given PAM and disease/control annotation, do Chi-square test for each row of PAM

**Usage**

```
Fisher_Test_Pam(pam, label.vec.num, pvalue.cutoff = 0.05)
```

**Arguments**

pam	Input data object that contains the data to be tested. Required
label.vec.num	The target binary condition. Required
pvalue.cutoff	choose p-value cut-off

**Value**

df.output object

**Examples**

```
tmp <- matrix(rbinom(12,1,0.5), nrow = 3)
rownames(tmp) <- c("a", "b", "c")
Fisher_Test_Pam(tmp, c(1,1,0,0))
```

---

formatTaxTable	<i>Format taxonomy table for rendering</i>
----------------	--

---

**Description**

Format taxonomy table for rendering

**Usage**

```
formatTaxTable(ttable)
```

**Arguments**

ttable            Taxonomy table

**Value**

Formatted table suitable for rendering with. DT::renderDataTable

---

getShinyInput	<i>Getter function to get the shinyInput option</i>
---------------	---

---

**Description**

Getter function to get the shinyInput option

**Usage**

```
getShinyInput()
```

**Value**

shinyInput option

**Examples**

```
getShinyInput()
```

---

getShinyInputCombat     *Getter function to get the shinyInputCombat option*

---

**Description**

Getter function to get the shinyInputCombat option

**Usage**

```
getShinyInputCombat()
```

**Value**

shinyInputCombat option

**Examples**

```
getShinyInputCombat()
```

---

getShinyInputOrig     *Getter function to get the shinyInputOrig option*

---

**Description**

Getter function to get the shinyInputOrig option

**Usage**

```
getShinyInputOrig()
```

**Value**

shinyInputOrig option

**Examples**

```
getShinyInputOrig()
```



---

`getSignatureFromMultipleGlmnet`*Use Lasso to do feature selection*

---

**Description**

Use Lasso to do feature selection

**Usage**

```
getSignatureFromMultipleGlmnet(df.input, target.vec, nfolds = 10,  
  logisticRegression = TRUE, nRun = 100, alpha = 1)
```

**Arguments**

<code>df.input</code>	Row is sample, column is feature. Required
<code>target.vec</code>	y vector. Required
<code>nfolds</code>	glmnet CV nfolds
<code>logisticRegression</code>	doing logistic regression or linear regression.
<code>nRun</code>	number of glmnet runs
<code>alpha</code>	same as in glmnet

**Value**

signature

**Examples**

```
data('iris')  
getSignatureFromMultipleGlmnet(iris[,1:4],  
  c(rep(1,100), rep(0,50)), nfolds = 3, nRun = 10)
```

---

`GET_PAM`*transform cpm counts to presence-absence matrix*

---

**Description**

transform cpm counts to presence-absence matrix

**Usage**`GET_PAM(df)`**Arguments**

<code>df</code>	Input data object that contains the data to be tested. Required
-----------------	---

**Value**

df.output object

**Examples**

```
GET_PAM(data.frame(a = c(1,3,0), b = c(0,0.1,2)))
```

---

grepTid	<i>Greps the tid from the given identifier string</i>
---------	---

---

**Description**

Greps the tid from the given identifier string

**Usage**

```
grepTid(id)
```

**Arguments**

id                      Given identifier string

**Value**

tid string

**Examples**

```
grepTid("ti|700015|org|Coriobacterium_glomerans_PW2")
```

---

loadPathoscopeReports	<i>Loads all data from a set of PathoID reports. For each column in the PathoID report, construct a matrix where the rows are genomes and the columns are samples. Returns a list where each element is named according to the PathoID column. For example, ret[["Final.Best.Hit.Read.Numbers"]] on the result of this function will get you the final count matrix. Also includes elements "total_reads" and "total_genomes" from the first line of the PathoID report.</i>
-----------------------	--

---

**Description**

Loads all data from a set of PathoID reports. For each column in the PathoID report, construct a matrix where the rows are genomes and the columns are samples. Returns a list where each element is named according to the PathoID column. For example, ret[["Final.Best.Hit.Read.Numbers"]] on the result of this function will get you the final count matrix. Also includes elements "total\_reads" and "total\_genomes" from the first line of the PathoID report.

**Usage**

```
loadPathoscopeReports(reportfiles, nrows = NULL)
```

**Arguments**

```
reportfiles    Paths to report files
nrows          Option to read first N rows of PathoScope reports
```

**Value**

Returns a list where each element is named according to the PathoID column. For example, `ret[["Final.Best.Hit.Read.Numbers"]]` on the result of this function will get you the final count matrix. Also includes elements "total\_reads" and "total\_genomes" from the first line of the PathoID report.

**Examples**

```
input_dir <- system.file("example/data", package = "PathoStat")
reportfiles <- list.files(input_dir, pattern = "*-sam-report.tsv",
full.names = TRUE)
```

---

loadPstat

*Load the R data(.rda) file with pathostat object*

---

**Description**

Load the R data(.rda) file with pathostat object

**Usage**

```
loadPstat(indir = ".", infileName = "pstat_data.rda")
```

**Arguments**

```
indir          Input Directory of the .rda file
infileName     File name of the .rda file
```

**Value**

pstat pathostat object (NULL if it does not exist)

**Examples**

```
data_dir <- system.file("data", package = "PathoStat")
infileName <- "pstat_data.rda"
pstat <- loadPstat(data_dir, infileName)
```

---

 log2CPM

*Compute log2(counts per mil reads) and library size for each sample*


---

**Description**

Compute log2(counts per mil reads) and library size for each sample

**Usage**

```
log2CPM(qcounts, lib.size = NULL)
```

**Arguments**

qcounts	quantile normalized counts
lib.size	default is colsums(qcounts)

**Value**

list containing log2(quantile counts per mil reads) and library sizes

**Examples**

```
log2CPM(matrix(1:12, nrow = 3))
```

---

LOOAUC\_simple\_multiple\_noplot\_one\_df

*LOOCV*


---

**Description**

LOOCV

**Usage**

```
LOOAUC_simple_multiple_noplot_one_df(df, targetVec)
```

**Arguments**

df	Row is sample, column is feature. Required
targetVec	y vector. Required

**Value**

mean auc

**Examples**

```
data('iris')
LOOAUC_simple_multiple_noplot_one_df(iris[,1:4],
c(rep(1,100), rep(0,50)))
```

---

LOOAUC\_simple\_multiple\_one\_df  
*LOOCV with ROC curve*

---

**Description**

LOOCV with ROC curve

**Usage**

```
LOOAUC_simple_multiple_one_df(df, targetVec)
```

**Arguments**

df	Row is sample, column is feature. Required
targetVec	y vector. Required

**Value**

the ROC

**Examples**

```
data('iris')
LOOAUC_simple_multiple_one_df(iris[,1:4],
c(rep(1,100), rep(0,50)))
```

---

PathoStat-class      *PathoStat class to store PathoStat input data including phyloseq object*

---

**Description**

Contains all currently-supported BatchQC output data classes:

**Details**

slots:

**average\_count** a single object of class otu\_tableOrNULL

**besthit\_count** a single object of class otu\_tableOrNULL

**highconf\_count** a single object of class otu\_tableOrNULL

**lowconf\_count** a single object of class otu\_tableOrNULL

**Examples**

```

otumat = matrix(sample(1:100, 100, replace = TRUE), nrow = 10, ncol = 10)
rownames(otumat) <- paste0("OTU", 1:nrow(otumat))
colnames(otumat) <- paste0("Sample", 1:ncol(otumat))
taxmat = matrix(sample(letters, 70, replace = TRUE),
nrow = nrow(otumat), ncol = 7)
rownames(taxmat) <- rownames(otumat)
colnames(taxmat) <- c("Domain", "Phylum", "Class",
"Order", "Family", "Genus", "Species")
OTU = phyloseq::otu_table(otumat, taxa_are_rows = TRUE)
TAX = phyloseq::tax_table(taxmat)
physeq = phyloseq::phyloseq(OTU, TAX)
pathostat1(physeq)

```

---

percent

*Compute percentage*


---

**Description**

Compute percentage

**Usage**

```
percent(x, digits = 2, format = "f")
```

**Arguments**

x	a number or a vector
digits	how many digit of percentage
format	numeric format, "f" for float

**Value**

the percentage

**Examples**

```
percent.vec <- percent(c(0.9, 0.98))
```

---

phyloseq_to_edgeR	<i>Convert phyloseq OTU count data into DGEList for edgeR package</i>
-------------------	---

---

**Description**

Further details.

**Usage**

```
phyloseq_to_edgeR(physeq, group, method = "RLE", ...)
```

**Arguments**

physeq	(Required).
group	(Required). A character vector or factor giving the experimental group/condition for each sample/library.
method	(Optional).
...	Additional arguments passed on to

**Value**

dispersion

**Examples**

```
data_dir_test <- system.file("data", package = "PathoStat")
pstat_test <- loadPstat(indir=data_dir_test,
  inFile="pstat_data_2_L1.rda")
phyloseq_to_edgeR(pstat_test, group="Sex")
```

---

plotPCAPlotly	<i>Plot PCA</i>
---------------	-----------------

---

**Description**

Plot PCA

**Usage**

```
plotPCAPlotly(df.input, condition.color.vec,
  condition.color.name = "condition", condition.shape.vec = NULL,
  condition.shape.name = "condition", columnTitle = "Title",
  pc.a = "PC1", pc.b = "PC2")
```

**Arguments**

`df.input`            Input data object that contains the data to be plotted. Required  
`condition.color.vec`  
                       color vector. Required  
`condition.color.name`  
                       color variable name. Required  
`condition.shape.vec`  
                       shape vector. Required  
`condition.shape.name`  
                       shape variable name. Required  
`columnTitle`        Title to be displayed at top of heatmap.  
`pc.a`                `pc.1`  
`pc.b`                `pc.2`

**Value**

the plot

**Examples**

```

data('iris')
plotPCAPlotly(t(iris[,1:4]),
condition.color.vec = c(rep(1,100), rep(0,50)),
condition.shape.vec = c(rep(0,100), rep(1,50)))

```

---

`plotPCoAPlotly`            *Plot PCoA*

---

**Description**

Plot PCoA

**Usage**

```

plotPCoAPlotly(physeq.input, condition.color.vec,
condition.color.name = "condition", condition.shape.vec = NULL,
condition.shape.name = "condition", method = "bray",
columnTitle = "Title", pc.a = "Axis.1", pc.b = "Axis.2")

```

**Arguments**

`physeq.input`        Input data object that contains the data to be plotted. Required  
`condition.color.vec`  
                       color vector. Required  
`condition.color.name`  
                       color variable name. Required  
`condition.shape.vec`  
                       shape vector. Required



condition.shape.name	shape variable name. Required
method	which distance metric
columnTitle	Title to be displayed at top of heatmap.
pc.a	pc.1
pc.b	pc.2

**Value**

the plot

**Examples**

```
data_dir_test <- system.file("data", package = "PathoStat")
pstat_test <- loadPstat(indir=data_dir_test,
  inFileName="pstat_data_2_L1.rda")
plotPCoAPlotly(pstat_test, condition.color.vec = rbinom(33,1,0.5),
  condition.shape.vec = rbinom(33,1,0.5))
```

---

pstat\_data

*pathostat object generated from example pathoscope report files*

---

**Description**

This example data consists of 33 samples from a diet study with 11 subjects taking 3 different diets in random order

**Usage**

pstat

**Format**

pathostat object extension of phyloseq-class experiment-level object:

**otu\_table** OTU table with 41 taxa and 33 samples

**sample\_data** Sample Data with 33 samples by 18 sample variables

**tax\_table** Taxonomy Table with 41 taxa by 9 taxonomic ranks

**sample\_data** Phylogenetic Tree with 41 tips and 40 internal nodes

**Value**

pathostat object

---

readPathoscopeData	<i>Reads the data from PathoScope reports and returns a list of final guess relative abundance and count data</i>
--------------------	---

---

### Description

Reads the data from PathoScope reports and returns a list of final guess relative abundance and count data

### Usage

```
readPathoscopeData(input_dir = ".",
  pathoreport_file_suffix = "-sam-report.tsv", use.input.files = FALSE,
  input.files.path.vec = NULL, input.files.name.vec = NULL)
```

### Arguments

input_dir	Directory where the tsv files from PathoScope are located
pathoreport_file_suffix	PathoScope report files suffix
use.input.files	whether input dir to pathoscope files or directly pathoscope files
input.files.path.vec	vector of pathoscope file paths
input.files.name.vec	vector of pathoscope file names

### Value

List of final guess relative abundance and count data

### Examples

```
example_data_dir <- system.file("example/data", package = "PathoStat")
pathoreport_file_suffix <- "-sam-report.tsv"
datlist <- readPathoscopeData(example_data_dir, pathoreport_file_suffix,
  input.files.name.vec = as.character(1:6))
```

---

runPathoStat	<i>Statistical Microbiome Analysis on the pathostat input and generates a html report and produces interactive shiny app plots</i>
--------------	--

---

### Description

Statistical Microbiome Analysis on the pathostat input and generates a html report and produces interactive shiny app plots

**Usage**

```
runPathoStat(pstat = NULL, report_dir = ".",
             report_option_binary = "111111111", interactive = TRUE)
```

**Arguments**

pstat	phyloseq extension pathostat object
report_dir	Output report directory path
report_option_binary	9 bits Binary String representing the plots to display and hide in the report
interactive	when TRUE, opens the interactive shinyApp

**Value**

outfile The output file with all the statistical plots

**Examples**

```
runPathoStat(interactive = FALSE)
```

---

savePstat	<i>Save the pathostat object to R data(.rda) file</i>
-----------	---

---

**Description**

Save the pathostat object to R data(.rda) file

**Usage**

```
savePstat(pstat, outdir = ".", outfileName = "pstat_data.rda")
```

**Arguments**

pstat	pathostat object
outdir	Output Directory of the .rda file
outfileName	File name of the .rda file

**Value**

outfile .rda file

**Examples**

```
data_dir_test <- system.file("data", package = "PathoStat")
pstat_test <- loadPstat(indir=data_dir_test,
                       infileName="pstat_data_2_L1.rda")
outfile <- savePstat(pstat_test)
```

---

setShinyInput	<i>Setter function to set the shinyInput option</i>
---------------	---

---

**Description**

Setter function to set the shinyInput option

**Usage**

```
setShinyInput(x)
```

**Arguments**

x                    shinyInput option

**Value**

shinyInput option

**Examples**

```
setShinyInput(NULL)
```

---

setShinyInputCombat	<i>Setter function to set the shinyInputCombat option</i>
---------------------	---

---

**Description**

Setter function to set the shinyInputCombat option

**Usage**

```
setShinyInputCombat(x)
```

**Arguments**

x                    shinyInputCombat option

**Value**

shinyInputCombat option

**Examples**

```
setShinyInputCombat(NULL)
```

---

setShinyInputOrig	<i>Setter function to set the shinyInputOrig option</i>
-------------------	---

---

**Description**

Setter function to set the shinyInputOrig option

**Usage**

```
setShinyInputOrig(x)
```

**Arguments**

x                    shinyInputOrig option

**Value**

shinyInputOrig option

**Examples**

```
setShinyInputOrig(NULL)
```

---

summarizeTable	<i>Summarize sample</i>
----------------	-------------------------

---

**Description**

Creates a table of summary metrics

**Usage**

```
summarizeTable(pstat)
```

**Arguments**

pstat                Input pstat

**Value**

A data.frame object of summary metrics.

**Examples**

```
data_dir_test <- system.file("data", package = "PathoStat")
pstat_test <- loadPstat(indir=data_dir_test,
  inFileName="pstat_data_2_L1.rda")
st.tmp <- summarizeTable(pstat_test)
```

---

TranslateIdToTaxLevel *Find the taxonomy for the given taxon id name*

---

### Description

Find the taxonomy for the given taxon id name

### Usage

```
TranslateIdToTaxLevel(pstat, input.id.vec, tax.level)
```

### Arguments

pstat	pathostat object
input.id.vec	names containing id
tax.level	target taxon level

### Value

target taxon level names

### Examples

```
data_dir_test <- system.file("data", package = "PathoStat")
pstat_test <- loadPstat(indir=data_dir_test,
  infileName="pstat_data_2_L1.rda")
names.new <- TranslateIdToTaxLevel(pstat_test,
  c("ti|862962|org|Bacteroides_fragilis_638R",
    "ti|697329|org|Ruminococcus_albus_7" ),
  "genus")
```

---

Wilcox\_Test\_df *Mann-whitney test for a dataframe*

---

### Description

Mann-whitney test for a dataframe

### Usage

```
Wilcox_Test_df(df, label.vec.num, pvalue.cutoff = 0.05)
```

### Arguments

df	Input data object that contains the data to be tested. Required
label.vec.num	The target binary condition. Required
pvalue.cutoff	choose p-value cut-off

**Value**

df.output object

**Examples**

```
data('iris')
Wilcox_Test_df(t(iris[,1:4]),
c(rep(1,100), rep(0,50)))
```

# Index

- \* **datasets**
  - pstat\_data, [17](#)
- Bootstrap\_LOOCV\_LR\_AUC, [3](#)
- Chisq\_Test\_Pam, [3](#)
- findRAfromCount, [4](#)
- findTaxonMat, [4](#)
- findTaxonomy, [5](#)
- findTaxonomy300, [6](#)
- Fisher\_Test\_Pam, [6](#)
- formatTaxTable, [7](#)
- GET\_PAM, [9](#)
- getShinyInput, [7](#)
- getShinyInputCombat, [8](#)
- getShinyInputOrig, [8](#)
- getSignatureFromMultipleGlmnet, [9](#)
- grepTid, [10](#)
- loadPathoscopeReports, [10](#)
- loadPstat, [11](#)
- log2CPM, [12](#)
- LOOAUC\_simple\_multiple\_noplot\_one\_df, [12](#)
- LOOAUC\_simple\_multiple\_one\_df, [13](#)
- PathoStat-class, [13](#)
- pathostat1 (PathoStat-class), [13](#)
- percent, [14](#)
- phyloseq\_to\_edgeR, [15](#)
- plotPCAPlotly, [15](#)
- plotPCoAPlotly, [16](#)
- pstat (pstat\_data), [17](#)
- pstat\_data, [17](#)
- readPathoscopeData, [18](#)
- runPathoStat, [18](#)
- savePstat, [19](#)
- setShinyInput, [20](#)
- setShinyInputCombat, [20](#)
- setShinyInputOrig, [21](#)
- summarizeTable, [21](#)
- TranslateIdToTaxLevel, [22](#)
- Wilcox\_Test\_df, [22](#)