# OmicNavigator API

John Blischak

September 27, 2024 OmicNavigator 1.14.2

## Contents

# 1 Introduction

The OmicNavigator Web App is deployed via OpenCPU. Below are example calls to the available endpoints for retrieving data or plots from installed study packages. For more details on each function, please see their individual man pages, which you can access by running `?nameOfFunction` in the R console. The example below displays the endpoints directly called in the R console and exported as JSON. However, note that the OpenCPU API is very flexible. You can query the API endpoints using the JavaScript client library (which is what the OmicNavigator Web App uses) or any HTTP client you prefer. And you aren't limited to JSON output, but can choose one of the many available output formats. For example, the `curl` command below downloads a results table to a CSV file from a hypothetical study package from OpenCPU running on the local machine.

```
curl http://localhost:5656/ocpu/library/OmicNavigator/R/getResultsTable/csv \
  -d 'study="studyABC"&modelID="model_01"&testID="test_01"' \
  > results.csv
```

If OmicNavigator is deployed on an external server, replace `http://localhost:5656` with the server's URL. For more information on the benefits and features of sharing and retrieving data via the OpenCPU API, please see the blog post Publishing dynamic data on ocpu.io.

```
library(jsonlite)
library(OmicNavigator)
```

# 2 List studies

List all the available studies along with their models, tests, annotations, and plots.

**Update:** The `plotType` may be a single string or a nested list of multiple plot types (e.g. "singleFeature" and "multiTest")

```
studies <- listStudies()
```

```
toJSON(studies, auto_unbox = TRUE, pretty = TRUE)
```

```
[
  {
    "name": "ABC",
    "package": {
```

```
  "Package": "ONstudyABC",
  "Title": "OmicNavigator study ABC",
  "Version": "0.0.0.9000",
  "Maintainer": "Unknown <unknown@unknown>",
  "Description": "The OmicNavigator data package for the study\n       \"ABC\"",
  "OmicNavigatorVersion": "1.14.2",
  "department": "immunology",
  "organism": "Mus musculus",
  "Imports": "data.table, ggplot2, graphics, plotly, rlang, stats",
  "Built": "R 4.4.1; ; 2024-09-27 05:30:23 UTC; unix",
  "description": "The OmicNavigator data package for the study\n       \"ABC\""
},
"results": [
  {
    "modelID": "model_01",
    "modelDisplay": "Model 1",
    "tests": [
      {
        "testID": "test_01",
        "testDisplay": "test 1"
      },
      {
        "testID": "test_02",
        "testDisplay": "test 2"
      }
    ]
  },
  {
    "modelID": "model_02",
    "modelDisplay": "Model 2",
    "tests": [
      {
        "testID": "test_01",
        "testDisplay": "test 1"
      },
      {
        "testID": "test_02",
        "testDisplay": "test 2"
      }
    ]
  },
  {
    "modelID": "model_03",
    "modelDisplay": "Model 3",
    "tests": [
      {
```

```
        "testID": "test_01",
        "testDisplay": "test 1"
      },
      {
        "testID": "test_02",
        "testDisplay": "test 2"
      }
    ]
  }
],
"enrichments": [
  {
    "modelID": "model_01",
    "modelDisplay": "Model 1",
    "annotations": [
      {
        "annotationID": "annotation_01",
        "annotationDisplay": "Terms from annotation_01"
      },
      {
        "annotationID": "annotation_02",
        "annotationDisplay": "Terms from annotation_02"
      },
      {
        "annotationID": "annotation_03",
        "annotationDisplay": "Terms from annotation_03"
      }
    ]
  },
  {
    "modelID": "model_02",
    "modelDisplay": "Model 2",
    "annotations": [
      {
        "annotationID": "annotation_01",
        "annotationDisplay": "Terms from annotation_01"
      },
      {
        "annotationID": "annotation_02",
        "annotationDisplay": "Terms from annotation_02"
      },
      {
        "annotationID": "annotation_03",
        "annotationDisplay": "Terms from annotation_03"
      }
    ]
```

```
  },
  {
    "modelID": "model_03",
    "modelDisplay": "Model 3",
    "annotations": [
      {
        "annotationID": "annotation_01",
        "annotationDisplay": "Terms from annotation_01"
      },
      {
        "annotationID": "annotation_02",
        "annotationDisplay": "Terms from annotation_02"
      },
      {
        "annotationID": "annotation_03",
        "annotationDisplay": "Terms from annotation_03"
      }
    ]
  }
],
"plots": [
  {
    "modelID": "model_01",
    "modelDisplay": "Model 1",
    "plots": [
      {
        "plotID": "plotBase",
        "plotDisplay": "Custom plot",
        "plotType": "singleFeature"
      },
      {
        "plotID": "plotMultiFeature",
        "plotDisplay": "PCA",
        "plotType": "multiFeature"
      },
      {
        "plotID": "plotMultiTestSf",
        "plotDisplay": "scatterplot_singlefeat",
        "plotType": "multiTest"
      },
      {
        "plotID": "plotMultiTestMf",
        "plotDisplay": "scatterplot_multifeat",
        "plotType": [
          "multiFeature",
          "multiTest"
```

```
      ]
    },
    {
      "plotID": "multiModel_scatterplot",
      "plotDisplay": "mmplot",
      "plotType": [
        "multiFeature",
        "multiModel"
      ]
    },
    {
      "plotID": "multiModel_barplot_sf",
      "plotDisplay": "mmplot_sf",
      "plotType": [
        "singleFeature",
        "multiModel"
      ]
    }
  ]
},
{
  "modelID": "model_02",
  "modelDisplay": "Model 2",
  "plots": [
    {
      "plotID": "plotBase",
      "plotDisplay": "Custom plot",
      "plotType": "singleFeature"
    },
    {
      "plotID": "plotMultiFeature",
      "plotDisplay": "PCA",
      "plotType": "multiFeature"
    },
    {
      "plotID": "plotMultiTestSf",
      "plotDisplay": "scatterplot_singlefeat",
      "plotType": "multiTest"
    },
    {
      "plotID": "plotMultiTestMf",
      "plotDisplay": "scatterplot_multifeat",
      "plotType": [
        "multiFeature",
        "multiTest"
      ]
```

```
        },
        {
          "plotID": "multiModel_scatterplot",
          "plotDisplay": "mmplot",
          "plotType": [
            "multiFeature",
            "multiModel"
          ]
        },
        {
          "plotID": "multiModel_barplot_sf",
          "plotDisplay": "mmplot_sf",
          "plotType": [
            "singleFeature",
            "multiModel"
          ]
        }
      ]
    },
    {
      "modelID": "model_03",
      "modelDisplay": "Model 3",
      "plots": [
        {
          "plotID": "plotGg",
          "plotDisplay": "Custom ggplot2 plot",
          "plotType": "singleFeature"
        },
        {
          "plotID": "plotPlotly",
          "plotDisplay": "Custom plotly plot",
          "plotType": [
            "singleFeature",
            "plotly"
          ]
        }
      ]
    }
  ]
}
]
```


## 3 Results table

For a given study, model, and test, return a table that contains the feature metadata and the inference results.

The column names are chosen by the user. The first column is the unique featureID used in the study. It should be passed to `plotStudy()` (Section 8) to create any custom plots. All the feature metadata columns are returned as character strings, even if they appear numeric.

```
resultsTable <- getResultsTable(
  study = "ABC",
  modelID = "model_01",
  testID = "test_01"
)
nrow(resultsTable)
```

```
[1] 100
```

```
toJSON(resultsTable[1:2, ], pretty = TRUE)
```


```
[
  {
    "customID": "feature_0060",
    "secondaryID": "feature_2_0041",
    "featureVarNumeric": "74",
    "featureVar01": "m",
    "featureVar02": "d",
    "featureVar03": "i",
    "beta": -3.5204,
    "beta_1": -0.9759,
    "p_val": 0.0001
  },
  {
    "customID": "feature_0084",
    "secondaryID": "feature_2_0017",
    "featureVarNumeric": "49",
    "featureVar01": "c",
    "featureVar02": "k",
    "featureVar03": "t",
    "beta": 3.7157,
    "beta_1": 1.6956,
    "p_val": 0.0002
  }
]
```


To filter the results table to only include features that belong to a specific annotation term, specify the optional arguments `annotationID` and `termID`.

```
resultsTableTerm <- getResultsTable(
  study = "ABC",
  modelID = "model_01",
```

```
  testID = "test_01",
  annotationID = "annotation_01",
  termID =  "term_01"
)
nrow(resultsTableTerm)
```

```
[1] 18
```

```
toJSON(resultsTableTerm[1:2, ], pretty = TRUE)
```


```
[
  {
    "customID": "feature_0032",
    "secondaryID": "feature_2_0069",
    "featureVarNumeric": "13",
    "featureVar01": "n",
    "featureVar02": "z",
    "featureVar03": "j",
    "beta": 3.2953,
    "beta_1": 1.3255,
    "p_val": 0.0003
  },
  {
    "customID": "feature_0086",
    "secondaryID": "feature_2_0015",
    "featureVarNumeric": "74",
    "featureVar01": "o",
    "featureVar02": "o",
    "featureVar03": "a",
    "beta": 2.8006,
    "beta_1": 0.1146,
    "p_val": 0.0065
  }
]
```


# 4 Enrichments table

For a given study, model, and annotation, return a table that contains the enrichment results. The default is to return the nominal statistical values.

```
enrichmentsTable <- getEnrichmentsTable(
  study = "ABC",
  modelID = "model_01",
  annotationID = "annotation_01"
)
toJSON(enrichmentsTable[1:2, ], pretty = TRUE)
```

```
[
  {
    "termID": "term_01",
    "description": "Description of term_01",
    "test_01": 0.03,
    "test_02": 0.03
  },
  {
    "termID": "term_02",
    "description": "Description of term_02",
    "test_01": 0.1,
    "test_02": 0.07
  }
]
```


Set `type = "adjusted"` to obtain statistical values adjusted for multiple testing.

```
enrichmentsTable <- getEnrichmentsTable(
  study = "ABC",
  modelID = "model_01",
  annotationID = "annotation_01",
  type = "adjusted"
)
toJSON(enrichmentsTable[1:2, ], pretty = TRUE)
```


```
[
  {
    "termID": "term_01",
    "description": "Description of term_01",
    "test_01": 0.05,
    "test_02": 0.05
  },
  {
    "termID": "term_02",
    "description": "Description of term_02",
    "test_01": 0.12,
    "test_02": 0.09
  }
]
```


The first two columns are always `termID` and `description`. The remaining columns are the names of the tests defined by the user. The column `termID` is used to create the barcode plot (Section 15).

# 5 Enrichments network

For a given study, model, and annotation, return the nodes and links of the network graph.

```
enrichmentsNetwork <- getEnrichmentsNetwork(
  study = "ABC",
  modelID = "model_01",
  annotationID = "annotation_01"
)
enrichmentsNetworkMinimal <- list(
  tests = enrichmentsNetwork[["tests"]],
  nodes = enrichmentsNetwork[["nodes"]][1:3, ],
  links = enrichmentsNetwork[["links"]][1:3, ]
)
toJSON(enrichmentsNetworkMinimal, auto_unbox = TRUE, pretty = TRUE)
```


```
{
  "tests": ["test_01", "test_02"],
  "nodes": [
    {
      "id": 1,
      "termID": "term_01",
      "description": "Description of term_01",
      "geneSetSize": 18,
      "nominal": [0.03, 0.03],
      "adjusted": [0.05, 0.05]
    },
    {
      "id": 2,
      "termID": "term_02",
      "description": "Description of term_02",
      "geneSetSize": 10,
      "nominal": [0.1, 0.07],
      "adjusted": [0.12, 0.09]
    },
    {
      "id": 3,
      "termID": "term_03",
      "description": "Description of term_03",
      "geneSetSize": 24,
      "nominal": [0.08, 0.1],
      "adjusted": [0.1, 0.12]
    }
  ],
  "links": [
    {
      "id": 1,
      "source": 1,
      "target": 3,
      "overlapSize": 6,
```

```
      "overlap": 0.3333,
      "jaccard": 0.1667
    },
    {
      "id": 2,
      "source": 1,
      "target": 4,
      "overlapSize": 3,
      "overlap": 0.2,
      "jaccard": 0.1
    },
    {
      "id": 3,
      "source": 1,
      "target": 6,
      "overlapSize": 3,
      "overlap": 0.2727,
      "jaccard": 0.1154
    }
  ]
}
```


# 6 Features in a network node

For a given study, annotation, and term, return the features in that term.

```
nodeFeatures <- getNodeFeatures(
  study = "ABC",
  annotationID = "annotation_01",
  termID = "term_01"
)
toJSON(nodeFeatures[1:4], pretty = TRUE)
```


```
["feature_0002", "feature_0010", "feature_0024", "feature_0032"]
```


# 7 Features in a network link

For a given study, annotation, and two terms, return the features shared by the terms.

```
linkFeatures <- getLinkFeatures(
  study = "ABC",
  annotationID = "annotation_01",
  termID1 = "term_01",
  termID2 = "term_03"
)
toJSON(linkFeatures[1:4], pretty = TRUE)
```

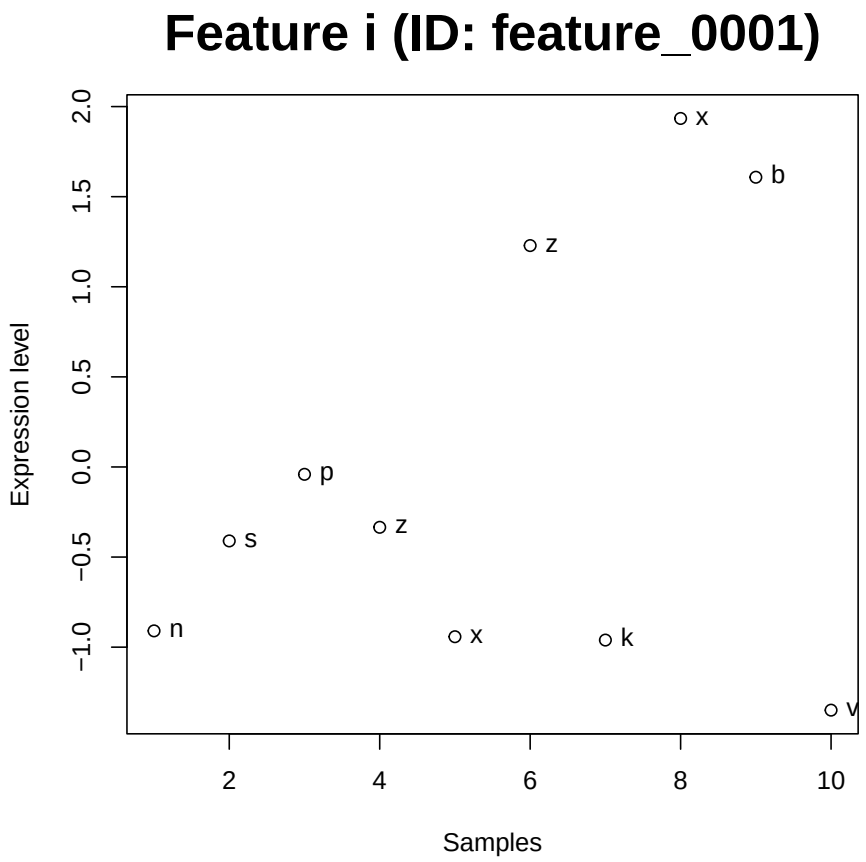
```
["feature_0032", "feature_0038", "feature_0058", "feature_0075"]
```

# 8 Custom plots

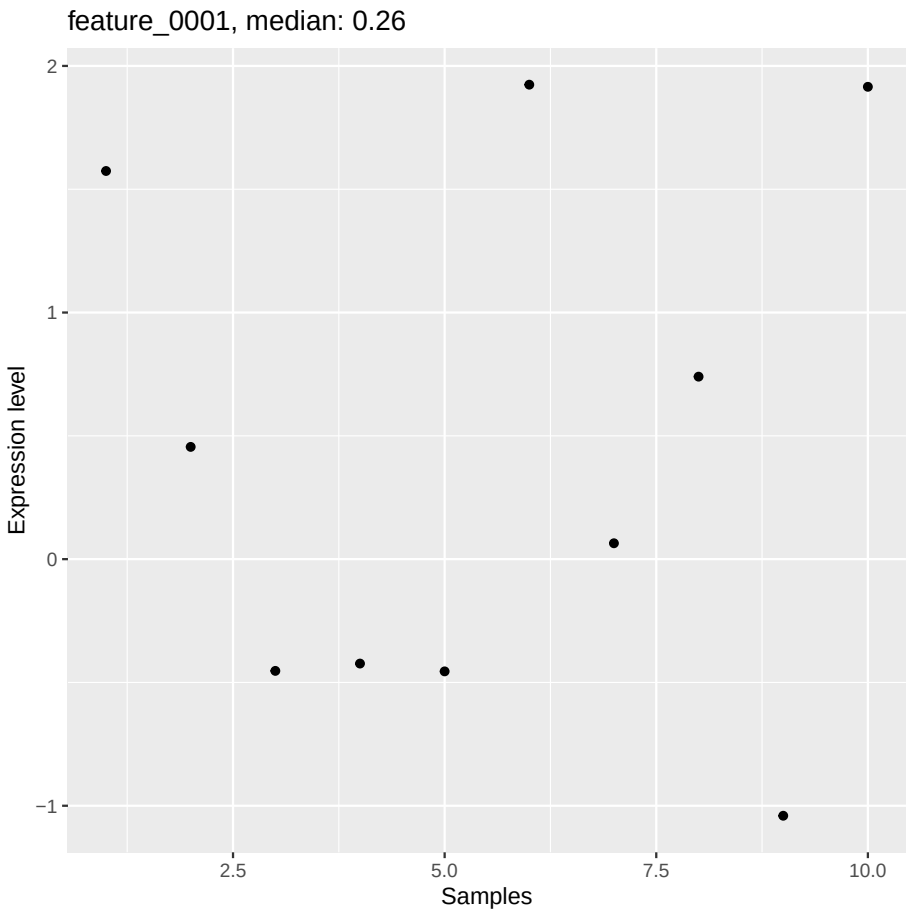
Display the custom plots provided by the user with `plotStudy()`. Provided a study, model, feature, test, and plot, `plotStudy()` generates the custom plot.

The featureID is obtained from the first column returned by `getResultsTable()` (Section 3). The remaining arguments are obtained from the output from `listStudies()` (Section 2).

```
plotStudy(
  study = "ABC",
  modelID = "model_01",
  featureID = "feature_0001",
  plotID = "plotBase",
  testID = "test_01"
)
```



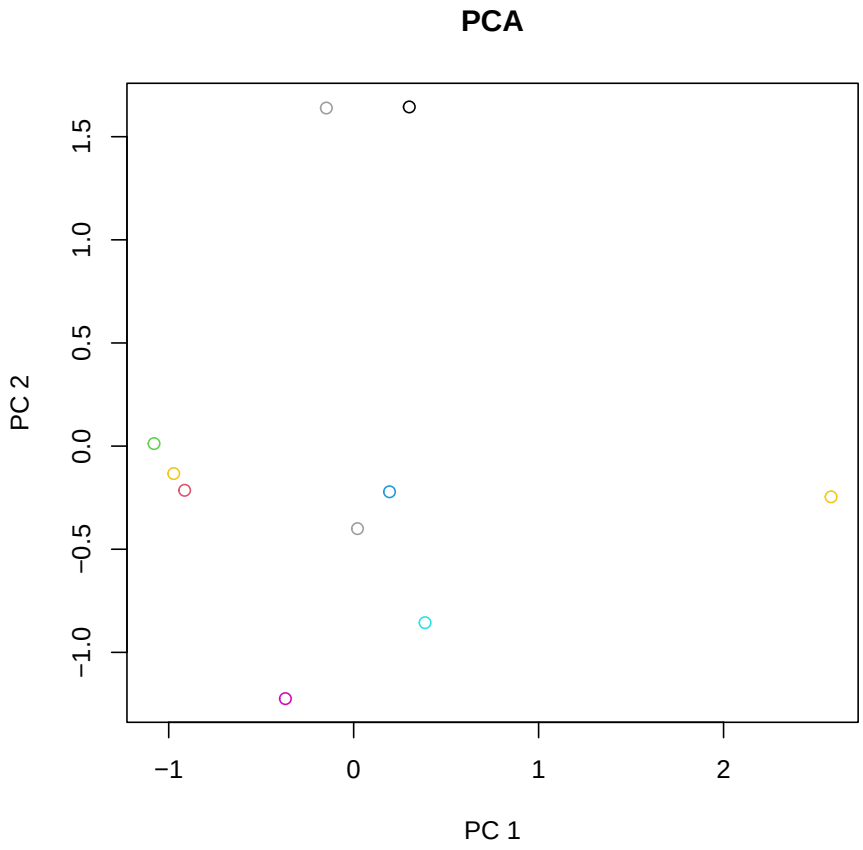
```
plotStudy(
  study = "ABC",
  modelID = "model_03",
  featureID = "feature_0001",
  plotID = "plotGg",
  testID = "test_01"
)
```




The study can also provide "multiFeature" plots, which accept more than one featureID. These can be distinguished from "singleFeature" plots via the field `plotType` in the output from `listStudies()` (Section 2).

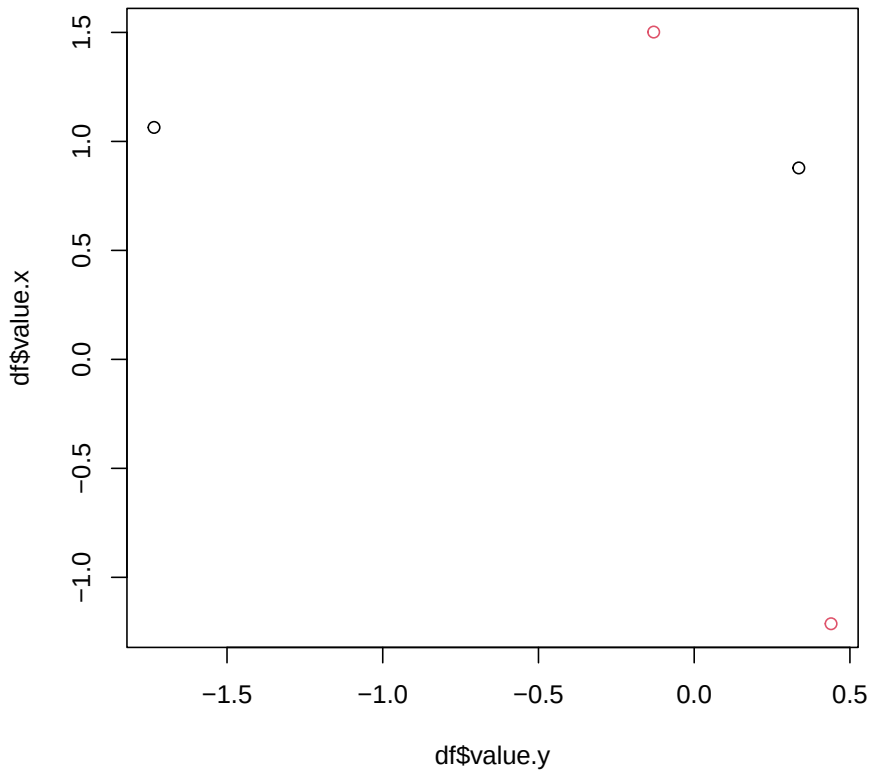
```
plotStudy(
  study = "ABC",
  modelID = "model_01",
  featureID = c("feature_0001", "feature_0002"),
  plotID = "plotMultiFeature",
```

```
  testID = "test_01"
)
```



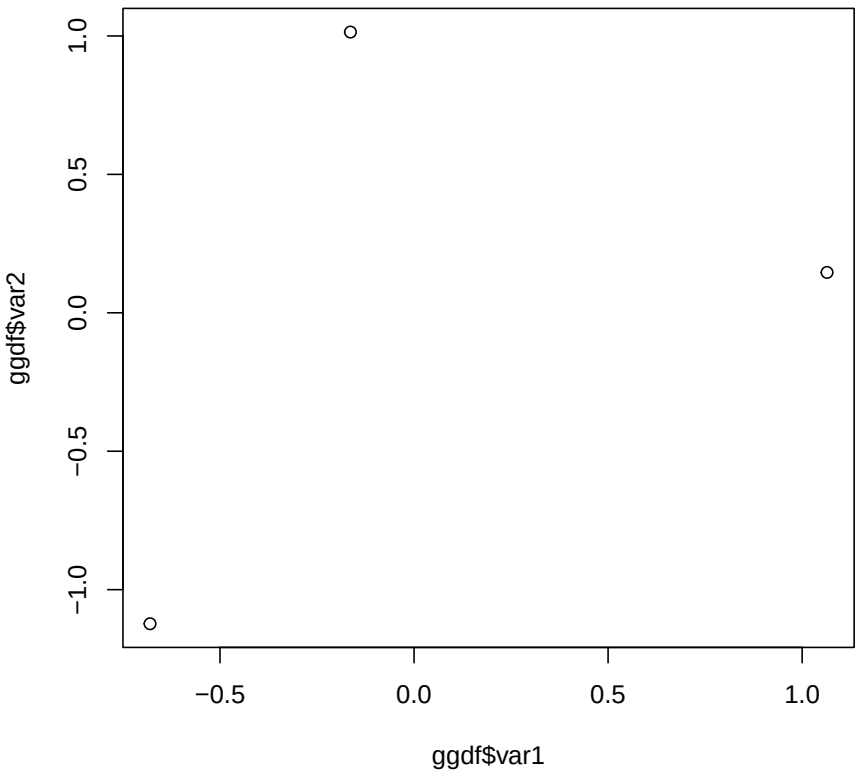

In addition, the study can provide "multiTest" plots, which accept more than one testID. These can be provided via the field `plotType` in the output from `listStudies()` (Section 2). Note that plotType for "multiTest" may be provided in a vector, e.g. plotType = c("multiFeature", "multiTest"). A call for plotType = "multiTest" is set to default to plotType = c("singleFeature", "multiTest")

```
plotStudy(
  study = "ABC",
  modelID = "model_01",
  featureID = c("feature_0001", "feature_0002"),
  plotID = "plotMultiTestMf",
  testID = c("test_01", "test_02")
)
```




Finally, the study can provide "multiModel" plots, which accept more than one modelID and one or more testIDs per modelID. These can be provided via the field `plotType` in the output from `listStudies()` (Section 2). Note that for plotType = "multiModel", testID and modelID should be vectors of same length, where the index position indicate which test in testID relate to which model in modelID.

```
modelID <- c("model_01", "model_02")
testID <- c("test_01", "test_02")
plotStudy(
  study = "ABC",
  modelID = modelID,
  featureID = c("feature_0002", "feature_0003", "feature_0004"),
  plotID = "multiModel_scatterplot",
  testID = testID
)
```




# 9 Results intersection

For a given study and model, filter the inference results table by the values of specific columns in any test of that model. Use `getUpsetCols()` (Section 13) to obtain the common columns across all tests of the model.

```
resultsIntersection <- getResultsIntersection(
  study = "ABC",
  modelID = "model_01",
  anchor = "test_01",
  mustTests = c("test_01", "test_02"),
  notTests = c(),
  sigValue = .5,
  operator = "<",
  column = "p_val"
```

```
)
toJSON(resultsIntersection[1:2, ], pretty = TRUE)
```


```
[
  {
    "customID": "feature_0060",
    "secondaryID": "feature_2_0041",
    "featureVarNumeric": "74",
    "featureVar01": "m",
    "featureVar02": "d",
    "featureVar03": "i",
    "Set_Membership": "test_01 , test_02",
    "beta": -3.5204,
    "beta_1": -0.9759,
    "p_val": 0.0001
  },
  {
    "customID": "feature_0084",
    "secondaryID": "feature_2_0017",
    "featureVarNumeric": "49",
    "featureVar01": "c",
    "featureVar02": "k",
    "featureVar03": "t",
    "Set_Membership": "test_01 , test_02",
    "beta": 3.7157,
    "beta_1": 1.6956,
    "p_val": 0.0002
  }
]
```


## 10 Enrichments intersection

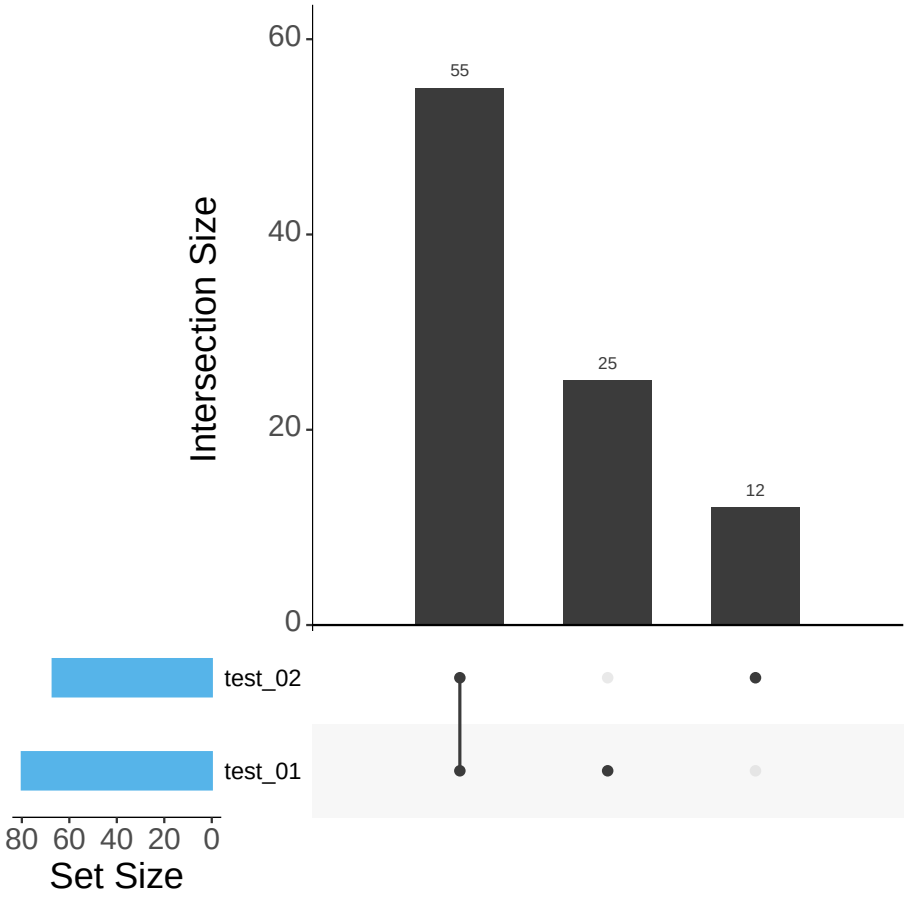
For a given study and model, filter the enrichments table (or network) by the results of the enrichment tests.

```
enrichmentsIntersection <- getEnrichmentsIntersection(
  study = "ABC",
  modelID = "model_01",
  annotationID = "annotation_01",
  mustTests = c("test_01", "test_02"),
  notTests = c(),
  sigValue = .5,
  operator = "<",
  type = "nominal"
)
toJSON(enrichmentsIntersection[1:2, ], pretty = TRUE)
```

```
[
  {
    "termID": "term_01",
    "description": "Description of term_01",
    "test_01": 0.03,
    "test_02": 0.03
  },
  {
    "termID": "term_02",
    "description": "Description of term_02",
    "test_01": 0.1,
    "test_02": 0.07
  }
]
```

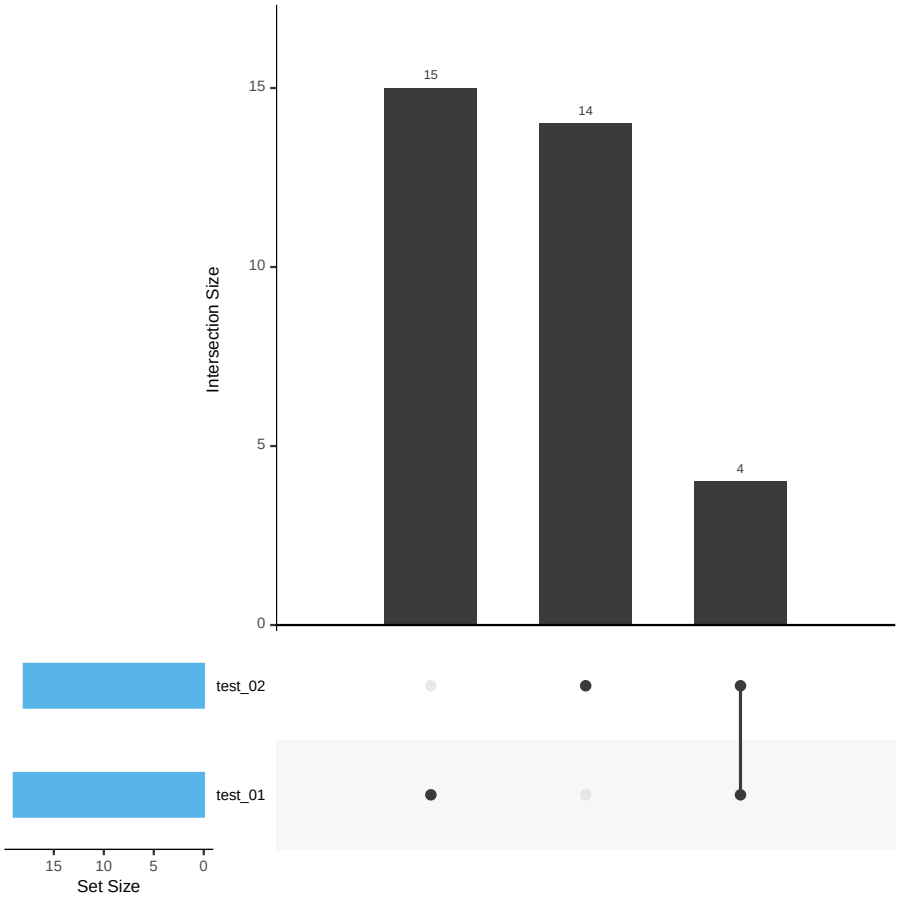

# 11 Results UpSet plot

```
resultsUpset <- getResultsUpset(
  study = "ABC",
  modelID = "model_01",
  sigValue = .5,
  operator = "<",
  column = "p_val"
)
```




## 12 Enrichments UpSet plot

```
enrichmentsUpset <- getEnrichmentsUpset(
  study = "ABC",
  modelID = "model_01",
  annotationID = "annotation_02",
  sigValue = .05,
  operator = "<",
  type = "nominal"
)
```




## 13 UpSet columns

Given a study and model, `getUpsetCols()` returns the columns common across all the available tests, and thus are available for filtering with `getResultsIntersection()` (Section 9).

```
upsetCols <- getUpsetCols(
  study = "ABC",
  modelID = "model_01"
)
toJSON(upsetCols, auto_unbox = TRUE, pretty = TRUE)
```

```
["beta", "p_val"]
```

## 14 metaFeatures table

For a given study, model, and featureID, return a table that contains the metaFeatures associated with that featureID.

```
metaFeaturesTable <- getMetaFeaturesTable(
  study = "ABC",
  modelID = "model_01",
  featureID = "feature_0001"
)
toJSON(metaFeaturesTable[1:2, ], pretty = TRUE)
```


```
[
  {
    "metaFeatureID": "metaFeature_0001",
    "metaFeatureVarNumeric": "296",
    "metaFeatureVar01": "n",
    "metaFeatureVar02": "m",
    "metaFeatureVar03": "r"
  },
  {
    "metaFeatureID": "metaFeature_0101",
    "metaFeatureVarNumeric": "296",
    "metaFeatureVar01": "x",
    "metaFeatureVar02": "i",
    "metaFeatureVar03": "c"
  }
]
```


## 15 Barcode

Given a study, model, test, annotation, and term, `getBarcodeData()` returns the data required to create the barcode and violin plots.

The `termID` is obtained from `getEnrichmentsTable()` (Section 4). The remaining arguments are obtained from the output from `listStudies()` (Section 2).

The elements of the `data` array are sorted by the value of `statistic` (highest to lowest).

```
barcodeData <- getBarcodeData(
  study = "ABC",
  modelID = "model_01",
  testID = "test_01",
  annotationID = "annotation_02",
  termID = "term_05"
)
toJSON(barcodeData, auto_unbox = TRUE, pretty = TRUE)
```

```
{
  "data": [
    {
      "featureID": "feature_0060",
      "featureEnrichment": "feature_0060",
      "featureDisplay": "feature_0060",
      "statistic": 3.5204,
      "logFoldChange": 0
    },
    {
      "featureID": "feature_0062",
      "featureEnrichment": "feature_0062",
      "featureDisplay": "feature_0062",
      "statistic": 2.0138,
      "logFoldChange": 0
    },
    {
      "featureID": "feature_0070",
      "featureEnrichment": "feature_0070",
      "featureDisplay": "feature_0070",
      "statistic": 1.9435,
      "logFoldChange": 0
    },
    {
      "featureID": "feature_0082",
      "featureEnrichment": "feature_0082",
      "featureDisplay": "feature_0082",
      "statistic": 1.5647,
      "logFoldChange": 0
    },
    {
      "featureID": "feature_0085",
      "featureEnrichment": "feature_0085",
      "featureDisplay": "feature_0085",
      "statistic": 1.4568,
      "logFoldChange": 0
    },
    {
      "featureID": "feature_0092",
      "featureEnrichment": "feature_0092",
      "featureDisplay": "feature_0092",
      "statistic": 1.2826,
      "logFoldChange": 0
    },
    {
      "featureID": "feature_0057",
```

```
    "featureEnrichment": "feature_0057",
    "featureDisplay": "feature_0057",
    "statistic": 1.0368,
    "logFoldChange": 0
  },
  {
    "featureID": "feature_0028",
    "featureEnrichment": "feature_0028",
    "featureDisplay": "feature_0028",
    "statistic": 0.9306,
    "logFoldChange": 0
  },
  {
    "featureID": "feature_0005",
    "featureEnrichment": "feature_0005",
    "featureDisplay": "feature_0005",
    "statistic": 0.9088,
    "logFoldChange": 0
  },
  {
    "featureID": "feature_0014",
    "featureEnrichment": "feature_0014",
    "featureDisplay": "feature_0014",
    "statistic": 0.7803,
    "logFoldChange": 0
  },
  {
    "featureID": "feature_0020",
    "featureEnrichment": "feature_0020",
    "featureDisplay": "feature_0020",
    "statistic": 0.4481,
    "logFoldChange": 0
  },
  {
    "featureID": "feature_0003",
    "featureEnrichment": "feature_0003",
    "featureDisplay": "feature_0003",
    "statistic": 0.164,
    "logFoldChange": 0
  }
],
"highest": 4,
"lowest": 0,
"labelStat": "Beta coefficient",
"labelLow": "Small effect size",
"labelHigh": "Large effect size"
```

```
}
```

# 16 Reports

Given a study and model, `getReportLink()` returns a link to a report file. This can either be a URL or a path to a file installed in a study package.

```
reportLink <- getReportLink(
  study = "ABC",
  modelID = "model_01"
)
toJSON(reportLink, auto_unbox = TRUE, pretty = TRUE)
```

```
"https://www.domain.com/default.html"
```

```
reportLink <- getReportLink(
  study = "ABC",
  modelID = "model_02"
)
toJSON(reportLink, auto_unbox = TRUE, pretty = TRUE)
```

```
"ONstudyABC/OmicNavigatorReports/model_02/report.html"
```

# 17 Linkouts in results table

Given a study and model, `getResultsLinkouts()` returns one or more URL patterns that provide linkouts to external resources. The fields refer to columns in the results table returned by `getResultsTable()` (Section 3). The value of the column in each row should be concatenated to the end of the URL pattern to form each linkout.

```
resultsLinkouts <- getResultsLinkouts(
  study = "ABC",
  modelID = "model_01"
)
toJSON(resultsLinkouts, auto_unbox = TRUE, pretty = 2)
```

```
{
  "customID": [
    "https://ensembl.org/Homo_sapiens/Gene/Summary?g=",
    "https://www.targetvalidation.org/target/"
  ],
  "featureVar01": "https://www.ncbi.nlm.nih.gov/gene/"
}
```

Note that if you automatically unbox single strings in the JSON response, then the results for each column may be a string or an array of strings depending on whether there are one or more linkouts.

## 18 Linkouts in enrichments table

Given a study and annotationID, `getEnrichmentsLinkouts()` returns one or more URL patterns that provide linkouts to external resources. The value of the column `termID` in each row should be concatenated to the end of the URL pattern to form each linkout.

```
enrichmentsLinkouts <- getEnrichmentsLinkouts(
  study = "ABC",
  annotationID = "annotation_01"
)
toJSON(enrichmentsLinkouts, auto_unbox = TRUE, pretty = 2)
```

```
[
  "https://amigo.geneontology.org/amigo/term/",
  "https://www.ebi.ac.uk/QuickGO/term/"
]
```

Note that if you automatically unbox single strings in the JSON response, then the results for each annotationID may be a string or an array of strings depending on whether there are one or more linkouts.

```
enrichmentsLinkouts <- getEnrichmentsLinkouts(
  study = "ABC",
  annotationID = "annotation_03"
)
toJSON(enrichmentsLinkouts, auto_unbox = TRUE, pretty = TRUE)
```

```
"https://reactome.org/content/detail/"
```

## 19 Linkouts in metaFeatures table

Given a study and model, `getMetaFeaturesLinkouts()` returns one or more URL patterns that provide linkouts to external resources. The fields refer to columns in the metaFeatures table returned by `getMetaFeaturesTable()` (Section 14). The value of the column in each row should be concatenated to the end of the URL pattern to form each linkout.

```
metaFeaturesLinkouts <- getMetaFeaturesLinkouts(
  study = "ABC",
  modelID = "model_01"
)
toJSON(metaFeaturesLinkouts, auto_unbox = TRUE, pretty = 2)
```

```
{
  "metaFeatureVar01": [
    "https://ensembl.org/Homo_sapiens/Gene/Summary?g=",
```

```
    "https://www.targetvalidation.org/target/"
  ],
  "metaFeatureVar02": "https://www.ncbi.nlm.nih.gov/gene/"
}
```

Note that if you automatically unbox single strings in the JSON response, then the results for each column may be a string or an array of strings depending on whether there are one or more linkouts.

## 20 Favicons for table linkouts

To enhance the display of the table linkouts, `getFavicons()` returns the URL to the favicon for each linkout. You can pass the result from `getResultsLinkouts()`, `getEnrichmentsLinkouts()`, or `getMetaFeaturesLinkouts()`.

```
resultsFavicons <- getFavicons(linkouts = resultsLinkouts)
toJSON(resultsFavicons, auto_unbox = TRUE, pretty = 2)
```

```
{
  "customID": [
    "https://ensembl.org/i/ensembl-favicon.png",
    "https://www.targetvalidation.org/favicon.png"
  ],
  "featureVar01": "https://www.ncbi.nlm.nih.gov/favicon.ico"
}
```

```
enrichmentsFavicons <- getFavicons(linkouts = enrichmentsLinkouts)
toJSON(enrichmentsFavicons, auto_unbox = TRUE, pretty = 2)
```

```
"https://reactome.org//templates/favourite/favicon.ico"
```

**Troubleshooting:** If you receive an error message from R about "unused arguments", this is likely because you passed the object directly to the OpenCPU call. Instead the entire object needs to be wrapped inside of an outer object and named `linkouts`.

## 21 Unavailable data

If a requested data resource is unavailable, an empty array is returned.

```
toJSON(getResultsTable(study = "ABC", modelID = "?", testID = "?"))
```

```
[]
```

```
 toJSON(getEnrichmentsTable(study = "ABC", modelID = "?", annotationID = "?"))
```

```
[]
```

```
 toJSON(getEnrichmentsNetwork(study = "ABC", modelID = "?", annotationID = "?"))
```

```
[]
```

```
 toJSON(getNodeFeatures(study = "ABC", annotationID = "?", termID = "?"))
```

```
[]
```

```
 toJSON(getLinkFeatures(study = "ABC", annotationID = "?", termID1 = "?",
                        termID2 = "?"))
```

```
[]
```

```
 toJSON(getUpsetCols(study = "ABC", modelID = "?"))
```

```
[]
```

```
 toJSON(getMetaFeaturesTable(study = "ABC", modelID = "?", featureID = "?"))
```

```
[]
```

```
 toJSON(getBarcodeData(study = "ABC", modelID = "?", testID = "?",
                       annotationID = "?", termID = "?"))
```

```
[]
```

```
 # Elements that have a value defined for the special modelID "default" will
 # return this value if the modelID cannot be found. Otherwise an empty array
 # would have been returned.
 toJSON(getReportLink(study = "ABC", modelID = "?"),
        auto_unbox = TRUE, pretty = TRUE)
```

```
"https://www.domain.com/default.html"
```

```
 toJSON(getResultsLinkouts(study = "ABC", modelID = "?"),
        auto_unbox = TRUE, pretty = 2)
```

```
{
  "customID": [
    "https://ensembl.org/Homo_sapiens/Gene/Summary?g=",
    "https://www.targetvalidation.org/target/"
  ],
  "featureVar01": "https://www.ncbi.nlm.nih.gov/gene/"
}
```

```
 toJSON(getEnrichmentsLinkouts(study = "ABC", annotationID = "?"))
```

```
[]
```

## 22 Package version

To obtain the current installed version of the OmicNavigator R package, call the function `getPackageVersion()` with no arguments.

```
toJSON(getPackageVersion(), auto_unbox = TRUE)
```

```
"1.14.2"
```