recipes

GW-Hive

Oct 15, 2021

CONTENTS:

CHAPTER

ONE

SUMMARY

Recipe files are used to describe the actions that the script “make-datasets.py” will perform on a “.csv” dataset file, to process the dataset for integration into a OncoMX or Glygen.

Recipe files are in “.json” format. For more information on the “.json” format, go to www.json.org

The recipe consists of two major sections: “outputfiles” and “tablelist”.

Put simply, in “tablelist” you will create tables from the list of “actions” that can be performed.

You will then use “outputfiles” to denote what tables you wish to export and the content to include from those tables.

The “outputfile” is typically the reformatted dataset “.csv” that will then move forward in the pipeline to eventually be included on the frontend of OncoMX or Glygen.

CHAPTER

TWO

EXAMPLE RECIPE

Here is the example.json recipe file with comments:

```
{
    "outputfiles": [
        {
            "fields": [

                # for fields, the name must be a name that has been given in the␣
↪tablelist section below
                # these are often field names from the input datasets, but also can be␣
↪fieldnames generated from actions
                # newname is the fieldname that will be generated in the output file and␣
↪replace the old field name
                {
                    "newname": "gene_symbol",
                    "name": "Gene Symbol"
                },
        {
                    "newname": "uniprot_ac",
                    "name": "Uniprot AC"
                },
        {
                    "newname": "chr_num",
                    "name": "Chr Num"
                },
        {
                    "newname": "chr_pos",
                    "name": "Chr Pos"
                },
        {
                    "newname": "ref_nt",
                    "name": "Ref NT"
                },
        {
                    "newname": "alt_nt",
                    "name": "Alt NT"
                }
            ],

        # the tableid denotes what table from the tablelist will be used for this output␣
↪file
```

(continues on next page)

(continued from previous page)

```
            "tableid": 3,

        # the masterlistfilter field denotes the fieldname that is used as the primary␣
↪field name for filtering
            "masterlistfilterfield":"gene_symbol",

        # the filepath gives the name and path for the outputfile
            "filepath": "unreviewed/example_dataset.csv"
        }
    ],
    "tablelist": [
        {
        # every table in tablelist has a tableid, which can be referenced by other␣
↪acgtions or the outputfile
            "id": 1,

        # input denotes we are taking a file in from the filepath listed
        "type": "input",

        # the separator denotes the structure of the input file
        # here the separator is 'doublequote comma doublequote'
        # so each line of the input file looks like this "a","b","c"
        "separator": "\",\"",

        # list of the fields in the inout file
            "fields": [
                "Uniprot AC",
                "Chr Num",
                "Chr Pos",
                "Ref NT",
                "Alt NT",
        "EXAC Alt Frequency",
        "1000 Genomes Alt Frequency"
            ],
            "filepath": "downloads/example_mutations.csv"
        },
    {
        # same as above for table '"id":1', but for a different input file
        "id": 2,
        "type": "input",
            "separator": "\",\"",
            "fields": [
                "uniprot accession",
                "Gene Symbol"
            ],
            "filepath": "downloads/example_mapping.csv"
        },
    {
        # this table performs an action
        "id": 3,

        # the type is output so it will be generating a new table based on the action␣
↪used
```

(continues on next page)

(continued from previous page)

```
        "type": "output",

        # the tables that are input to generate this new table.
        # in this case our input files from tables 1 and 2
        "inputtables":[1,2],

        # what action will be taken to make the new table
        "action":{
            # the name gives the specific action to take
        "name":"jointables",

        # different actions have their own parameters.
        # in this case the jointables action has only the parameter "anchorfields".
        # specific details on actions are below in the README
        "anchorfields":["Uniprot AC", "uniprot accession"]
            },
        }
    ]
}
```

CHAPTER

THREE

RECIPE ACTIONS

3.1 union

3.1.1 Summary

Merge multiple tables and remove duplicate lines.

3.1.2 Format

Output Format:

```
{
    "id":#,
    "type":"output",
    "inputtables":[#,#,#],
    "action":{
        "name":"union"
    }
}
```

3.1.3 Notes:

This action can take multiple inputtable tables.

See use-case recipe example_part1.json for usage example.

3.2 addconstantfield

3.2.1 Summary

Add a new field to the table that has the same constant entry for each line.

3.2.2 Format

Output Format:

```
{
    "id":#,
    "type":"output",
    "inputtables":[#,#,#],
    "action":{
        "name":"addconstantfield",
        "newfields":{
            "example_field_1":"example_entry_1",
            "example_field_2":"example_entry_2"
        }
    }
}
```

3.2.3 Notes:

This action can take multiple inputtable tables.

See use-case recipe example_part2.json for usage example.

3.3 makecombofield

3.3.1 Summary

Combine two fields into one field.

3.3.2 Format

Output Format:

```
{
    "id":#,
    "type":"output",
    "inputtables":[#],
    "action":{
        "name":"makecombofield",
        "fieldlist":[
            "example_field_1",
            "example_field_2"
        ],
        "merge_char":"|",
        "combofield":"example_combo_field"
    }
}
```

3.3.3 Notes:

This action can take a single inputtable table.

The above example makes the combined field example_combo_field from example_field_1 and example_field_2.

The entries in each of the fields will merge into the combined field and be separated by the merge character, in the example the charcater is |.

See use-case recipe example_part1.json for usage example.

3.4 filterin

3.4.1 Summary

Filter only certain lines from one table into another.

Needs review from Robel.

3.4.2 Format

Output Format:

```
{
    "id":#,
    "type":"output",
    "inputtables":[#],
    "action":{
        "name":"filterin",
        "operation":"AND",
        "conditionlist":[
            "field":"example_field_1",
            "value":[
                "ABC",
                "-",
                ""
            ],
            "operation":"in"
    }
}
```

3.4.3 Notes:

This action can take a single inputtable table.

In the above example, only the lines with the values ABC, -, or an empty field will be included in to a new table.

See use-case recipe example_part2.json for usage example.

Question for Robel: What is the purpose of “operation”?

3.5 filterout

3.5.1 Summary

Filter out only certain lines from a table to create a table without those entries.

Needs review from Robel.

3.5.2 Format

Output Format:

```
{
    "id":#,
    "type":"output",
    "inputtables":[#],
    "action":{
        "name":"filterout",
        "operation":"AND",
        "conditionlist":[
            "field":"example_field_1",
            "value":[
                "DEF"
                "123"
                "456"
            ],
            "operation":"in"
    }
}
```

3.5.3 Notes:

This action can take a single inputtable table.

In the above example, lines with the values DEF, 123, or 456 in the field example_field_1 will be excluded from a new table, and all other lines with other values for that field will be included.

See use-case recipe example_part1.json for usage example.

Question for Robel: What is the purpose of "operation"?

3.6 expandrows

3.6.1 Summary

For a single line that has a field with multiple data points in one field, create multiple lines where each line has a single data point from that field.

3.6.2 Format

Output Format:

```
{
    "id":#,
    "type":"output",
    "inputtables":[#],
    "action":{
        "name":"expandrows",
        "expansionfield":"example_field_1",
        "expansiondelim":";"
        }
    }
}
```

3.6.3 Notes:

This action can take a single inputtable table.

In the above example, the field exmple_field_1 will be used to expand rows. For every line that has the expansion delim ;, the data in that row will be separated and a new line created for each data point.

An example:

Table input: “example_field_1” “data_point_1;data_point_2”,”information_abc”

Table output from expandrows above: “example_field_1”,”example_field_2” “data_point_1”,”information_abc” “data_point_2”,”information_abc”

See use-case recipe example_part2.json for usage example.

3.7 transposecols

3.7.1 Summary

The rows switch with columns and vice versa(?)

Needs review from Robel.

3.7.2 Format

Output Format:

```
{
    "id":#,
    "type":"output",
    "inputtables":[#],
    "action":{
        "name":"transposecols",
        "startcolidx":1,
        "newfieldone":"new_example_field_1"
```

(continues on next page)

(continued from previous page)

```
        "newfieldtwo":"new_example_field_2"
        }
    }
}
```

3.7.3 Notes:

This action can take a single inputtable table.

See use-case recipe example_part2.json for usage example.

Question for Robel: How is this action used?

3.8 addnormalizedcol

3.8.1 Summary

Create a new column that takes the numerical data from an existing column and normalizes the numbers according to (?)

Needs review from Robel.

3.8.2 Format

Output Format:

```
{
        "id":#,
        "type":"output",
        "inputtables":[#],
        "action":{
            "name":"addnormalizedcol",
            "startcolidx":1,
            "newfield":"normalized_example_field_1"
            "anchorfields":["example_field_2"]
            }
        }
    }
```

3.8.3 Notes:

This action can take a single inputtable table.

See use-case recipe example_part2.json for usage example.

Question for Robel: How is normalization calculated?

3.9 addaveragecol

3.9.1 Summary

Create a new column that takes the numerical data from an existing column and averages the numbers according to (?)

Needs review from Robel.

3.9.2 Format

Output Format:

```
{
        "id":#,
        "type":"output",
        "inputtables":[#],
        "action":{
            "name":"addaveragecol",
            "field":"example_field_1",
            "newfield":"average_example_field_1"
            "anchorfields":["example_field_2"]
            }
        }
    }
```

3.9.3 Notes:

This action can take a single inputtable table.

See use-case recipe example_part2.json for usage example.

Question for Robel: How is the average being calculated?

3.10 splitcol

3.10.1 Summary

Split a single column into two separate columns after specifying a deliminating character that separates the values.

3.10.2 Format

Output Format:

```
{
    "id":#,
    "type":"output",
    "inputtables":[#],
    "action":{
        "name":"splitcol",
```

(continues on next page)

(continued from previous page)

```
        "field":"example_field_1",
        "newfields":["example_field_1a","example_field_1b"],
        "delim":"|"
        }
    }
}
```

3.10.3 Notes:

This action can take a single inputtable table.

See use-case recipe example_part1.json for usage example.

3.11 jointables

3.11.1 Summary

Create a new table that uses an anchor field to combine columns from multiple tables. Data with the same anchor field value will be added to the same row in the new field.

3.11.2 Format

Output Format:

```
{
    "id":#,
    "type":"output",
    "inputtables":[#,#,#],
    "action":{
        "name":"jointables",
        "anchorfields":["example_field_1","example_field_2"]
        }
    }
}
```

3.11.3 Notes:

This action can take multiple inputtable table.

See use-case recipe example_part1.json for usage example.

CHAPTER

FOUR

USE-CASE EXAMPLES

There are three example data files as well as an example recipe to demonstrate how actions are used on real data.

Example Recipes and the actions they contain:

example_part1.json

- union
- jointables
- makecombofield
- splitcol

example_part2.json

- all actions from part 1
- filterin
- filterout
- expandrows
- addnormalizedcol
- addaveragecol

Actions not yet added:

- transposecol
- addconstantfield

The example recipe takes all of the example data files as input.

In the recipe each of the actions are used. To see the outcome of any one action, change the tableid option listed in the outputfiles section.

Be sure to change the name of the outputfile in the filepath option in the outputfiles section, otherwise each time you run the recipe it will overwrite the previous output file.

Example Data: example_mutations_1.csv example_mutations_2.csv example_mapping.csv

The contents of the data files are the following:

example_mutations_1.csv and example_mutations_2.csv contain SNP data gatehred from UCSC genome browser. - There are both unique and overlapping values between these two mutation files

example_mapping.csv contains uniprot terms and their corresponsing gene symbols, which allows us to show how you can map terms from one table to terms from one table to another

CHAPTER

FIVE

RUNNING THE SCRIPT

To execute a recipe, run the following command:

python make-datasets.py -d your_data_name

For your_data_name provide the name of the recipe before the .json, so in this case the recipe name is your_data_name.json

5.1 Notes:

To run the make-datasets.py script, you must be in the same folder as the script

Your recipe must be in the recipes folder

CHAPTER

SIX

TROUBLESHOOTING

The majority of errors that prevent the script from running result from inproper formatting of the JSON file.

In the error message you will typically receive a line number on which the error is encountered.

To find this exact line number, while using vi to look at your recipe, type :set number and hit enter to see line numbers

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search