User Manual: TCGA2STAT in Python

Development Environment

- OS:
  - Ubuntu
  - Version: 20.04.1 LTS
- PC: Latitude E6440
- Anaconda:
  - Version: 2020.02
  - Build: py37_0
- Python: 3.7.6

Python Packages Required and Tested

- requests: 2.22.0
- bs4: 4.8.2
- re: 2.2.1
- pandas: 1.0.1
- numpy: 1.18.1

Installation

The following describes how to use TCGA2STAT-Python on a Linux machine.

- Install Python on a Linux machine. It's recommended to install Anaconda that comes with Python and most of the commonly used packages.
- Install the following Python packages
  - requests
  - bs4
  - re
  - pandas
  - numpy
- Download TCGA2STAT-Python and place it in a working directory

Usage

To run the TCGA2STAT-Python in a Python interactive environment such as Python interpreter or Jupter Notebook, issue the following commands.

- Import sys
- Add the directory of TCGA2STAT-Python to Python sys.path
- Import getTCGA

```
>>> import sys
>>> sys.path.append('home/somebody/tcga2stat-python')
>>> import getTCGA
```

Now the command getTCGA can be issued for downloading various types of data. Please refer to the examples in the Examples section for details of the usage.

We can also run Python scripts that run getTCGA commands under the Linux command line. For example, we can run the following example script that comes with TCGA2STAT-Python under the Linux command line.

Python example-TumorNormalMatch.py

Examples

TCGA2STAT enables users to download any type of data via one single, uniform interface: the getTCGA function. The type of data to be downloaded can be specified via three parameters:

- Disease – acronym for the cancer type
- dataType – the type of omics-profiles
- type – the specific type of measurement for the omics-profile

Example-1 Download RPKM values of genes profiled using RNA-Sequencing from ovarian cancer patients

The following command will return the RPKM values of genes profiled using RNA-Sequencing from ovarian cancer patients

```
>>> rnaseq_ov = getTCGA.getTCGA(disease='OV', dataType='RNASeq', type='RPKM')
```

The returned object rnaseq_ov is a Python dictionary containing three items:

- dat
- clinical
- merged_dat

Each of the items is a list which contains the arrays of downloaded or post processed data.

```
>>> print(rnaseq_ov)
{'dat': [        TCGA-04-1348-01A-01R-1565-13-2  ... TCGA-61-2113-01A-01R-1568-13-2
AADACL3                          0.0539  ...                          0.0087
AADACL4                          0.0000  ...                          0.0000
ABCA4                            0.4204  ...                          0.4106
ABCB10                           3.6155  ...                          3.7246
ABCD3                            5.3981  ...                          8.4151
...                                 ...  ...                             ...
UTY                              0.0000  ...                          0.0012
VCY                              0.0000  ...                          0.0000
XGPY2                            0.0000  ...                          0.0000
XKRY2                            0.0000  ...                          0.0000
ZFY                              0.0007  ...                          0.0007

[19990 rows x 299 columns]], 'clinical': [], 'merged_dat': []}
```

The items in Python dictionary are accessed by its corresponding keys.

```
>>> print (rnaseq_ov.keys())
dict_keys(['dat', 'clinical', 'merged_dat'])
```

Here each item is a Python list. The following shows how to find the length of each item.

```
>>> print ('len of dat:', len(rnaseq_ov['dat']))
len of dat: 1
>>> print ('len of dat:', len(rnaseq_ov['merged_dat']))
len of dat: 0
>>> print ('len of dat:', len(rnaseq_ov['clinical']))
len of dat: 0
```

- dat – A Python list that contains matrices in Pandas DataFrame format of omics-profiles data. In this case its length is 1. It contains only one omics-profiles data matrix with genes in rows and patients in columns; in this example, this is a matrix of RPKM values.
- clinical – A Python list that contains matrices of clinical data in Pandas DataFrame format. Length of zero is returned as the clinical data is not specified (default).
- merged_dat – A Python list that contains data matrices in Pandas DataFrame format that combines both omics-profiles and clinical covariates if clinical data is imported. Patients are in the rows with the sample ID in the first column, followed by clinical covariates, and then the omics-profiles. Length of zero is returned here as there is no merged data.

The following shows how to find the shape of the matrix contained in rnaseq_ov['dat'][0].

```
>>> print (rnaseq_ov['dat'][0].shape)
(19990, 299)
```

The data matrix of the omics profiles returned are of dimension gene x patients; the row names are gene symbols and the column names are samples ID (BCR code).

```
>>> print (rnaseq_ov['dat'][0].iloc[:,0:3])
        TCGA-04-1348-01A-01R-1565-13-2 TCGA-04-1357-01A-01R-1565-13-2 TCGA-04-1362-01A-01R-1565-13-2
AADACL3                         0.0539                         0.2341                         0.0573
AADACL4                         0.0000                         0.0000                         0.0000
ABCA4                           0.4204                         0.1647                         0.2120
ABCB10                          3.6155                         7.1614                         3.1549
ABCD3                           5.3981                         3.8884                         7.2949
...                                ...                            ...                            ...
UTY                             0.0000                         0.0000                         0.0000
VCY                             0.0000                         0.0000                         0.0000
XGPY2                           0.0000                         0.0000                         0.0000
XKRY2                           0.0000                         0.0000                         0.0000
ZFY                             0.0007                         0.0000                         0.0000

[19990 rows x 3 columns]
```

To obtain default clinical data together with the omics profiles, specify clinical=True.

```
>>> ranseq_os_ov = getTCGA.getTCGA(disease='OV', dataType='RNASeq', type='RPKM', clinical=True)
```

The following commands look at the information of the merged RPKM-OS data matrix and its data.

```
>>> # Look at the merged RPKM-OS data matrix
>>> print ( len(ranseq_os_ov['merged_dat']) )
1
>>> print ( ranseq_os_ov['merged_dat'][0].shape )
(295, 19993)
>>> print ( ranseq_os_ov['merged_dat'][0].iloc[:,0:5] )
             bcr  status    OS AADACL3 AADACL4
0    TCGA-04-1348       1  1483  0.0539  0.0000
1    TCGA-04-1357       0   nan  0.2341  0.0000
2    TCGA-04-1362       1  1348  0.0573  0.0000
3    TCGA-04-1364       1  1024  0.0000  0.0242
4    TCGA-04-1365       0  2329  0.0177  0.0091
..            ...     ...   ...     ...     ...
290  TCGA-13-0899       1  2012  0.0000  0.0132
291  TCGA-13-0913       0  3325  0.0053  0.0000
292  TCGA-13-0916       0  1785  0.0000  0.0474
293  TCGA-13-0920       1  1484  0.1137  0.0225
294  TCGA-13-0924       0  2614  0.0131  0.0000

[295 rows x 5 columns]
```

To customize the clinical variables, specify cvars to the desired clinical variable.

```
>>> # Get the RPKM gene expression profiles along with all clinical data, and combine the expression with age
>>> ranseq_age_ov = getTCGA.getTCGA(disease='OV', dataType='RNASeq', type='RPKM', clinical=True, cvars ='yearstobirth')
```

```
>>> # Look at the merged RPKM-age data matrix
>>> print (len(ranseq_age_ov['merged_dat']))
1
>>> print (ranseq_age_ov['merged_dat'][0].shape)
(295, 19992)
>>> print (ranseq_age_ov['merged_dat'][0].iloc[0:6,0:5])
            bcr YEARSTOBIRTH AADACL3 AADACL4   ABCA4
0  TCGA-04-1348           44  0.0539  0.0000  0.4204
1  TCGA-04-1357           52  0.2341  0.0000  0.1647
2  TCGA-04-1362           59  0.0573  0.0000  0.2120
3  TCGA-04-1364           61  0.0000  0.0242  0.0560
4  TCGA-04-1365           87  0.0177  0.0091  0.0757
5  TCGA-04-1514           45  0.0019  0.0000  0.1713
```

Example-2 Import mRNA-microarray gene expression data

Gene expression data profiled by microarray are available in different forms depending on the type of microarray employed. Specifically, three forms are available.

1. Agilent 244K Custom Gene Expression G4502A-07 (type='G450')
2. Affymetrix Human Genome U133A 2.0 Array (type='U133')
3. Affymetrix Human Exon 1.0 ST Array (type='Huex')

The following shows how to get Agilent G450 for ovarian cancer patients.

```
>>> exp_ov = getTCGA.getTCGA( disease='OV', dataType='mRNA_Array', type='G450')
```

```
>>> # Look at the data
>>> print ('keys: %s' %exp_ov.keys())
keys: dict_keys(['dat', 'clinical', 'merged_dat'])
>>> print ('# of arrays in \'dat\':', len(exp_ov['dat']))
# of arrays in 'dat': 1
>>> print('dim of array:', exp_ov['dat'][0].shape)
dim of array: (17814, 597)
```

```
>>> print (exp_ov['dat'][0].iloc[:,0:3])
       TCGA-01-0630-11A-01R-0363-07 TCGA-01-0631-11A-01R-0363-07 TCGA-01-0633-11A-01R-0363-07
ELMO2                         0.092          0.00818750000000001                    -0.0953125
CREB3L1                      0.4695           0.147333333333333                        0.0235
RPS11                        0.9656                       0.7915                        1.2695
PNMA1                        0.8154                        0.896                        0.8314
MMP2                       -0.969125                      -0.04525                        -0.423
...                              ...                          ...                           ...
PIK3IP1            1.39516666666667             1.67733333333333             2.07883333333333
SLC39A6           -3.01871428571429            -2.37064285714286            -1.79942857142857
SNRPD2            0.359153846153846            0.162769230769231            0.202615384615385
AQP7                         -0.151            0.107333333333333           -0.264666666666667
CTSC              0.273692307692308            0.116153846153846            0.364384615384615

[17814 rows x 3 columns]
```

The following shows downloading Affymetrix Human Genome U133A expression for ovarian cancer patients.

```
>>> # Get Affymetrix Human U133A expression for ovarian cancer patients
>>> u133a_ov = getTCGA.getTCGA(disease='OV', dataType='mRNA_Array', type='U133')
```

```
>>> # Look at the data
>>> print ('keys: %s' %u133a_ov.keys())
keys: dict_keys(['dat', 'clinical', 'merged_dat'])
>>> print ('# of arrays in \'dat\':', len(u133a_ov['dat']))
# of arrays in 'dat': 1
>>> print('dim of array:', u133a_ov['dat'][0].shape)
dim of array: (12042, 543)
```

```
>>> print (u133a_ov['dat'][0].iloc[:,0:3])
        TCGA-01-0628-11A-01R-0362-01 TCGA-01-0630-11A-01R-0362-01 TCGA-01-0631-11A-01R-0362-01
AACS                 5.77226291802245             6.42842432754612             5.57423652793502
FSTL1                 8.2204141988226             8.49656014920149             8.71024089973616
ELMO2                4.93335027907364             5.43091291713807             4.70552176058355
CREB3L1              3.62271956519996             3.87023827592548              3.5142345164437
RPS11                  9.970795242234             10.3873060978660             10.1243551596926
...                               ...                          ...                          ...
RPS27                12.7560635369073             12.7799185923844             12.8971649451472
SNRPD2               11.1382621117328             10.9455364430979             11.1508678365719
SLC39A6              7.59507908338818             8.37633421519248             8.02450765802168
CTSC                 8.49827524319921             9.16241514800735             8.46334949967826
AQP7                  3.5773050384372             3.58279618679531             3.00545802290256

[12042 rows x 3 columns]
```

Example-3 Import RNA-Seq gene expression data

TCGA also provides gene expression data measured via RNA-Sequencing technology, especially via Illumina Genome Analyzer (GA) or Illumina HiSeq 2000. For either platform, the data is available in two forms: RNA-Seq (preprocessed using the first pipeline), and RNA-SeqV2 (preprocessed using version two analysis).

TCGA2STAT-Python package allows user to download RNA-Seq data from both Illumina GA and Illumina HiSeq, depending on data availability. If gene expression is profiled via both platforms, only the data from HiSeq will be imported. In addition, users have the following options for which type of data to import: raw-counts (type='count') or the normalized counts (type='RPKM').

The following shows downloading raw count, RNA-Seq data for ovarian cancer patients.

```
>>> # Get raw count, RNA-Seq data for ovarian cancer patients
>>> counts_ov = getTCGA.getTCGA(disease='OV', dataType='RNASeq', type='count')
```

```
>>> # Look at the data
>>> print ('# of arrays in dat:', len(counts_ov['dat']) )
# of arrays in dat: 1
>>> print ('dim of 1st array in dat:', counts_ov['dat'][0].shape )
dim of 1st array in dat: (19990, 299)
```

```
>>> print (counts_ov['dat'][0].iloc[:,0:3])
        TCGA-04-1348-01A-01R-1565-13 TCGA-04-1357-01A-01R-1565-13 TCGA-04-1362-01A-01R-1565-13
AADACL3                           36                           64                           30
AADACL4                            0                            0                            0
ABCA4                            575                           92                          226
ABCB10                          2308                         1864                         1572
ABCD3                           4890                         1437                         5160
...                              ...                          ...                          ...
UTY                                0                            0                            0
VCY                                0                            0                            0
XGPY2                              0                            0                            0
XKRY2                              0                            0                            0
ZFY                                1                            0                            0

[19990 rows x 3 columns]
```

```
>>> rnaseq_ov = getTCGA.getTCGA(disease='OV', dataType='RNASeq', type='RPKM')
```

```
>>> # Look at the data
>>> print ('# of arrays in dat:', len(rnaseq_ov['dat']) )
# of arrays in dat: 1
>>> print ('dim of 1st array in dat:', rnaseq_ov['dat'][0].shape )
dim of 1st array in dat: (19990, 299)
>>> print (rnaseq_ov['dat'][0].iloc[:,0:3])
        TCGA-04-1348-01A-01R-1565-13-2 TCGA-04-1357-01A-01R-1565-13-2 TCGA-04-1362-01A-01R-1565-13-2
AADACL3                         0.0539                         0.2341                         0.0573
AADACL4                         0.0000                         0.0000                         0.0000
ABCA4                           0.4204                         0.1647                         0.2120
ABCB10                          3.6155                         7.1614                         3.1549
ABCD3                           5.3981                         3.8884                         7.2949
...                                ...                            ...                            ...
UTY                             0.0000                         0.0000                         0.0000
VCY                             0.0000                         0.0000                         0.0000
XGPY2                           0.0000                         0.0000                         0.0000
XKRY2                           0.0000                         0.0000                         0.0000
ZFY                             0.0007                         0.0000                         0.0000

[19990 rows x 3 columns]
```

For RNA-Sequencing data from the second analysis pipeline (RNASeqV2), TCGA2STAT Python package provides information of data from Illumina HiSeq only as this is the most common profiling method. The values returned are the RSEM values for the genes, which usually contains 20501 genes, with RSEM values ranging from 0 to 10^6.

The following demonstrates downloading RSEM values of RNA-SeqV2 data for ovarian cancer patients.

```
>>> # Get RSEM values, RNA-SeqV2 data for ovarian cancer patients
>>> rsem_ov = getTCGA.getTCGA( disease='OV', dataType='RNASeq2')
```

```
>>> # Look at the data
>>> print ('# of arrays in dat:', len(rsem_ov['dat']) )
# of arrays in dat: 1
>>> print ('dim of 1st array in dat:', rsem_ov['dat'][0].shape )
dim of 1st array in dat: (20501, 307)
>>>
>>> print (rsem_ov['dat'][0].iloc[:,0:3])
        TCGA-04-1348-01A-01R-1565-13 TCGA-04-1357-01A-01R-1565-13 TCGA-04-1362-01A-01R-1565-13
A1BG                          66.4695                      65.5664                      41.6412
A1CF                           0.0000                       0.0000                       0.3310
A2BP1                          0.2689                       0.6510                       4.3025
A2LD1                        221.5219                     141.2826                     265.8161
A2ML1                          7.5289                      54.6875                       5.6263
...                               ...                          ...                          ...
ZYX                        15871.2019                    5378.9062                    6137.2982
ZZEF1                        505.7811                     805.3385                     901.5315
ZZZ3                         475.9344                     415.3646                     803.8987
psiTPTE22                      4.5711                      13.0208                     882.3359
tAKR                           0.0000                       0.0000                       0.0000

[20501 rows x 3 columns]
```

Example-4 Import Mutation Data

The level-3 files of called mutations from the Broad Firehose are saved in Mutation Annotation Format (MAF).Each MAF file lists mutations found for the particular patient. Since each patient may have different mutations, the number of genes in each MAF file vary from patient to patient. Hence, the MAF files are not in a format ready for statistical analysis. To solve this, our package aggregates the obtained MAF files into a matrix of dimension, *gene x patient*, with a value of one in cell(i,j) if a mutation is found in gene-i from patient-j, and a zero otherwise.

The TCGA2STAT-Python package permits two options for the mutation data:

- Only somatic non-silent mutations called (type="somatic") which is the default. This will only return those mutations with “somatic”" as the mutation-status and not with “silent” as the variant-classification in the original MAF files.
- All mutations called, (type="all").

The following is the example of downloading somatic-non-silent mutations for ovarian cancer patients.

```
>>> # Get somatic non-silent mutations for ovarian cancer patients
>>> mut_ov = getTCGA.getTCGA( disease='OV', dataType='Mutation', type='somatic')
```

```
>>> print ('keys: %s' %mut_ov.keys() )
keys: dict_keys(['dat', 'clinical', 'merged_dat'])
>>> print (mut_ov)
{'dat': [         TCGA-24-1471  TCGA-13-0899  TCGA-23-1021  ...  TCGA-04-1349  TCGA-24-1423  TCGA-13-1
498
PPP2R5A              1             0             0  ...             0             0             0
TBC1D4               1             0             0  ...             0             0             0
ICAM1                1             0             0  ...             0             0             0
CPS1                 1             0             0  ...             0             0             0
EP300                1             0             0  ...             0             0             0
...                ...           ...           ...  ...           ...           ...           ...
PLD1                 0             0             0  ...             0             0             1
NR5A2                0             0             0  ...             0             0             1
FBXL5                0             0             0  ...             0             0             1
ZNF527               0             0             0  ...             0             0             1
NAA35                0             0             0  ...             0             0             1

[5821 rows x 232 columns]], 'clinical': [], 'merged_dat': []}
```

The following is the example of downloading data for all mutations called for ovarian cancer patients.

```
>>> # Get all mutations called for ovarian cancer patients
>>> allmut_ov = getTCGA.getTCGA(disease='OV', dataType='Mutation', type='all')
```

```
>>> print ('keys: %s' %allmut_ov.keys() )
keys: dict_keys(['dat', 'clinical', 'merged_dat'])
>>> print (allmut_ov)
{'dat': [          TCGA-24-1471  TCGA-13-0899  TCGA-23-1021  ...  TCGA-04-1349  TCGA-24-1423  TCGA-13-
1498
HRNR                  1             0             0  ...             0             0             0
LHX9                  1             0             0  ...             0             0             0
PPP2R5A               1             0             0  ...             0             0             0
ZP4                   1             0             0  ...             0             0             0
NKX2-3                1             0             0  ...             0             0             0
...                 ...           ...           ...  ...           ...           ...           ...
KIRREL                0             0             0  ...             0             0             1
C1orf226              0             0             0  ...             0             0             1
NR5A2                 0             0             0  ...             0             0             1
FBXL5                 0             0             0  ...             0             0             1
NAA35                 0             0             0  ...             0             0             1

[10060 rows x 316 columns]], 'clinical': [], 'merged_dat': []}
```

Example-5 Import Methylation Expression

DNA methylation profiles were obtained either via Illumina Infinium HumanMethulation27 BeadChip or the HumanMethylation450 BeadChip. The first platform probes for about 27,000 CpG sites (commonly known as 27K) and the second platform probes for about 450,000 CpG sites. Our package allows users to import either methylation data when available, via type="27K' or type="450K" as shown below.

```
>>> # Get 27K methylation profiles for ovarian cancer patients
>>> methyl_ov = getTCGA.getTCGA(disease='OV', dataType='Methylation', type='27K')
```

```
>>> # Get 450K methylation profiles for ovarian cancer patients
>>> methyl45_ov = getTCGA.getTCGA(disease='OV', dataType='Methylation', type='450K')
```

Here we look at the downloaded methylation data.

```
>>> # Look at the data
>>> print ('keys: %s' %methyl_ov.keys() )
keys: dict_keys(['cpgs', 'dat', 'clinical', 'merged_dat'])
>>> print ('# of arrays in cpgs:', len(methyl_ov['cpgs']) )
# of arrays in cpgs: 1
>>> print ('# of arrays in dat:', len(methyl_ov['dat']) )
# of arrays in dat: 1
>>> print ('# of arrays in clinical:', len(methyl_ov['clinical']) )
# of arrays in clinical: 0
>>> print ('# of arrays in merged_dat:', len(methyl_ov['merged_dat']) )
# of arrays in merged_dat: 0
```

```
>>> print ('dim of the array in dat:', methyl_ov['dat'][0].shape )
dim of the array in dat: (27578, 612)
>>> print ('dim of the array in cpgs:', methyl_ov['cpgs'][0].shape )
dim of the array in cpgs: (27578, 3)
```

Note that the matrix of the methylation profiles returned is NOT aggregated at the gene level. Each row in the data matrix (dat) is a probe from the methylation assay, which represents a CpG site. Since genes often contain more than one CpG site and each CpG site can differ significantly in the methylation level, gene level aggregation is less desirable. Hence, our package returns the probe-level data.

Now we take a look at the data in first three rows and the first three columns of the matrices in 'dat' and 'cpgs'.

```
>>> print (methyl_ov['dat'][0].iloc[0:5,0:3])
           TCGA-01-0628-11A-01D-0383-05 TCGA-01-0630-11A-01D-0383-05 TCGA-01-0631-11A-01D-0383-05
cg00000292             0.799408581806102             0.620394166707065             0.911488407991807
cg00002426             0.339004437825915             0.180304596166719             0.293850485176926
cg00003994            0.0281193010321092            0.0360729815167471            0.0615912255800526
cg00005847              0.60116496510252             0.649557771329293             0.473949078041184
cg00006414                           NaN                           NaN                           NaN
>>>
>>> print (methyl_ov['cpgs'][0].iloc[0:5,0:3])
             Gene_Symbol Chromosome Genomic_Coordinate
cg00000292         ATP2A1         16           28890100
cg00002426          SLMAP          3           57743543
cg00003994          MEOX2          7           15725862
cg00005847          HOXD3          2          177029073
cg00006414  ZNF425;ZNF398          7          148822837
```

As shown above, the returned list for methylation data has one additional element, cpgs, which contains the probe's genomic information. Each row of cpgs is a CpG site and contains gene symbols and genomic coordinates in the columns; rows in cpgs are of the same number of order as the methylation profiles data, dat.

Values in the methylation data are the beta values. They range from zero to one, with values greater than 0.5 representing hypermethylation, values less than 0.5 representing hypomethylation.

Example-6 Combine multiple OMICs profiles

TCGA2STAT-Python package includes a function names OMICSBind that allows users to combine two matrices of omics-profiles for the same set of patients into a single object. In order to combine more than two types of omics-profiles together, users can call the OMICSBind function multiple times. The example below shows how to combine the RNA-Seq data from the second pipeline analysis, methylation profiles from 27K, and mutation data for ovarian cancer patients.

The following shows how to download RNA-Seq, methylation, and mutation profiles for OV cancer patients.

```
>>> # Get the RNA-seq, methylation, and mutation profiles for OV cancer patients
>>> seq = getTCGA.getTCGA( disease='OV', dataType='RNASeq2')
```

```
>>> meth = getTCGA.getTCGA( disease='OV', dataType='Methylation', type='27K')
```

```
>>> mut = getTCGA.getTCGA(disease='OV', dataType='Mutation', type='all')
```

The following shows merging the three OMICs-data into one Pandas DataFrame object.

```
>>> # Step1: merge RNA-Seq and mutation data
>>> m1 = getTCGA.OMICSBind( dat1=seq['dat'][0], dat2=mut['dat'][0] )
>>> # Step2: further concatenate the methylation data to the merged data-object
>>> m2 = getTCGA.OMICSBind( dat1=m1['merged_data'], dat2=meth['dat'][0] )
```

Now we look at the dat.

```
>>> # Look at the data
>>> print (seq['dat'][0].shape)
(20501, 307)
>>> print (meth['dat'][0].shape)
(27578, 612)
>>> print (mut['dat'][0].shape)
(10060, 316)
>>> print (m1['merged_data'].shape)
(30561, 185)
>>> print (m2['merged_data'].shape)
(58139, 185)
```

As shown above, the final merged data matrix m2[‘merged_data’] has 185 samples. These are the tumor samples of patients common in RNA-Seq, mutation, and methylation data. On the other hand, m2[‘merged_data’] has 58139 rows, which is the combination of 20501 genes from RNA-Seq, 10060 genes from mutation, and 30561 CpG sites from methylation data. In order to distinguish gene symbols from different omics-profiles, we affix the gene names with d1 and d2 in the merged data. Specifically, d1 is affixed to gene names of the first omics-data input to OMICSBind function, d2 is affixed to the names of the second input object.

Now we look at the more details of data in m1.

```
>>> print (m1.keys())
dict_keys(['merged_data', 'X', 'Y'])
>>> print (m1['merged_data'].shape)
(30561, 185)
>>> print (m1['X'].shape)
(20501, 185)
>>> print (m1['Y'].shape)
(10060, 185)
```

```
>>> print (list(m1['merged_data'].columns)[:3])
['TCGA-25-2042', 'TCGA-61-2000', 'TCGA-10-0928']
>>> print (list(m1['merged_data'].index)[:3])
['d1.A1BG', 'd1.A1CF', 'd1.A2BP1']
```

Now we look at the details of data in m2.

```
>>> print (m2.keys())
dict_keys(['merged_data', 'X', 'Y'])
>>> print (m2['merged_data'].shape)
(58139, 185)
>>> print (m2['merged_data'].shape)
(58139, 185)
>>> print (m2['X'].shape)
(30561, 185)
>>> print (m2['Y'].shape)
(27578, 185)
```

```
>>> print (list(m2['merged_data'].columns)[:3])
['TCGA-25-2042', 'TCGA-61-2000', 'TCGA-10-0928']
>>> print (list(m2['merged_data'].index)[:3])
['d1.d1.A1BG', 'd1.d1.A1CF', 'd1.d1.A2BP1']
```

Example-7 Separating OMICs profiles by sample types

The TCGA OMICs profiles downloaded for a cancer type often include samples of different types, such as from tumor tissue, normal tissue, or other controls. The type of samples is encoded in the sample's BCR code. Users who are interested in studying only tumor samples, for example, would need to parse the codes to extract only these samples. To spare users the difficulties of decoding the sample ID (BCR code) to extract particular types of samples, the TCGA2STAT-Python package includes a function, SampleSplit to accomplish this task according to the sample types. The object returned is a dictionary of three matrices corresponding to the omics profiles of primary solid tumor, recurrent solid tumor, and normal tissues/blood samples.

The following shows downloading the data and split the data.

```
>>> lusc_ranseq2 = getTCGA.getTCGA( disease='LUSC', dataType='RNASeq2' )
```

```
>>> # Split the OMICs data by sample types
>>> lusc_ranseq2_bytype = getTCGA.SampleSplit( lusc_ranseq2['dat'][0] )
```

The following shows getting the details of the data from the results of SampleSplit.

```
>>> print ( lusc_ranseq2_bytype['primary_tumor'].shape )
(20501, 501)
>>> print ( lusc_ranseq2_bytype['recurrent_tumor'].shape )
(20501, 0)
>>> print ( lusc_ranseq2_bytype['normal'].shape )
(20501, 51)
```

```
>>> print ( lusc_ranseq2_bytype['primary_tumor'].iloc[0:5, 0:3] )
      TCGA-18-3406-01A-01R-0980-07 TCGA-18-3407-01A-01R-0980-07 TCGA-18-3408-01A-01R-0980-07
A1BG                     741.6929                      46.7127                       1.1864
A1CF                       0.0000                       0.4757                       0.0000
A2BP1                      0.0000                       0.0000                       3.5592
A2LD1                    170.2362                     118.4063                     148.3316
A2ML1                    128.3465                    1413.4158                     167.8778
>>>
>>> print ( lusc_ranseq2_bytype['normal'].iloc[0:5, 0:3] )
      TCGA-22-4593-11A-01R-1820-07 TCGA-22-4609-11A-01R-2125-07 TCGA-22-5471-11A-01R-1635-07
A1BG                      49.4892                      68.5126                      42.2731
A1CF                       0.0000                       0.0000                       0.0000
A2BP1                      1.0373                       0.0000                       0.0000
A2LD1                    105.2180                      66.7109                      55.1627
A2ML1                      0.0000                       3.8408                       0.0000
```

Example-8 Getting matched tumor-normal OMICs profiles

TCGA2STAT-Python package also includes a utility function, TumorNormalMatch to prepare the downloaded omics-profiles for pairwise analysis between tumor tissues and normal tissues/blood samples in a user-friendly manner. The example below shows how to first get the RNA-Seq data of analysis pipeline 2 from LUSC patients and then split the data into tumor tissues and normal tissues/blood samples for the same set of patients.

```
>>> # Get the RNA-SeqV2 data for LUSC cancer patients
>>> lusc_ranseq2 = getTCGA.getTCGA( disease='LUSC', dataType='RNASeq2' )
```

```
>>> # tumor-normal matched profiles
>>> lusc_ranseq2_tum_norm = getTCGA.TumorNormalMatch( lusc_ranseq2['dat'][0] )
```

The TumorNormalMatch command will return a Python dictionary of two omics-profile data matrices with dimension of gene by patient. Both matrices have the same dimension and order for both columns (patient) and rows (gene). Thus, enabling users to make direct pairwise comparisons.

```
>>> # Look at the data
>>> print ('keys: %s' %lusc_ranseq2_tum_norm.keys() )
keys: dict_keys(['primary_tumor', 'normal'])
>>>
>>> print (lusc_ranseq2_tum_norm['primary_tumor'].shape)
(20501, 51)
>>> print (lusc_ranseq2_tum_norm['normal'].shape)
(20501, 51)
```

```
>>> print ( lusc_ranseq2_tum_norm['primary_tumor'].iloc[0:5,0:3] )
      TCGA-22-4609 TCGA-34-7107 TCGA-22-5471
A1BG       55.3917      33.1617      75.8906
A1CF        0.0000       0.3491       0.0000
A2BP1       0.3391       1.7454       0.0000
A2LD1      78.1757      82.7297     185.0630
A2ML1    7797.5585     127.7598   10927.3120
>>>
>>> print (lusc_ranseq2_tum_norm['normal'].iloc[0:5,0:3])
      TCGA-22-4609 TCGA-34-7107 TCGA-22-5471
A1BG       68.5126     122.3186      42.2731
A1CF        0.0000       0.0000       0.0000
A2BP1       0.0000       0.4564       0.0000
A2LD1      66.7109      67.3026      55.1627
A2ML1       3.8408       2.2821       0.0000
```

Functions

getTCGA

gdats = getTCGA(dataType) # dataType!=Methylation

- gdats is a Python dictionary containing 3 items: “dat”, “clinical”, “merged_dat”
- gdats[“dat”] is a Python List contains data in Pandas DataFrame.
  gdats[“dat”][0] gives the first array of dat
- gdats[“clinical”] is a Python List contains data in Pandas DataFrame format.
  gdats[“clinical”][0] gives the first array of clinical data
- gdats[“merged_data”] is a Python List contains merged data arrays in Pandas DataFrame format
  gdats[“merged_dat”] gives the first array of the merged data.

gdats = getTCGA(dataType) # dataType==Methylation

- gdats is a Python dictionary containing 4 items: “dat”, “cpgs”, “clinical”, and “merged_dat”
- Each of the item is a list of data arrays in Pandas DataFrame format
- Accessing the data in each item is the same as the case that dataType is not “Methylation”

SampleSplit

tumors = SampleSplit(dat)

- tumors is a Python dictionary containing 3 items: “primary_tumor”, “recurrent_tumor”, and “normal”. Each item is an data array in Pandas DataFrame format
- Accessing the 3 items with the following:
  - tumors[“primary_tumor”]
  - tumors[“recurrent_tumor”]
  - tumors[“normal”]

TumorNormalMatch

tumors = TumorNormalMatch(dat)

- tumors is a Python dictionary containing 2 items: “primary_tumor” and “normal”. Each item is an data array in Pandas DataFrame format
- Accessing the 2 items with the following:
  - tumors[“primary_tumor”]
  - tumors[“normal”]

OMICSBind

mdat = OMICSBind(dat1, dat2)

- mdat is a Python dictionary containing one item: “merged_dat”. The item is an data array in Pandas DataFrame format

- Accessing the item with the following:
    - mdat[“merged_dat”]