iScience, Volume *25*

Supplemental information

Processes in DNA damage response from a whole-cell multi-omics perspective

James C. Pino, Alexander L.R. Lubbock, Leonard A. Harris, Danielle B. Gutierrez, Melissa A. Farrow, Nicole Muszynski, Tina Tsui, Stacy D. Sherrod, Jeremy L. Norris, John A. McLean, Richard M. Caprioli, John P. Wikswo, and Carlos F. Lopez

Processes in DNA-damage response from a whole-cell multi-omics perspective

1 Supplementary Information

Jupyter Notebooks

: _notebook_1_data_exploration.pdf

: _notebook_2_network_creation_and_exploration.pdf

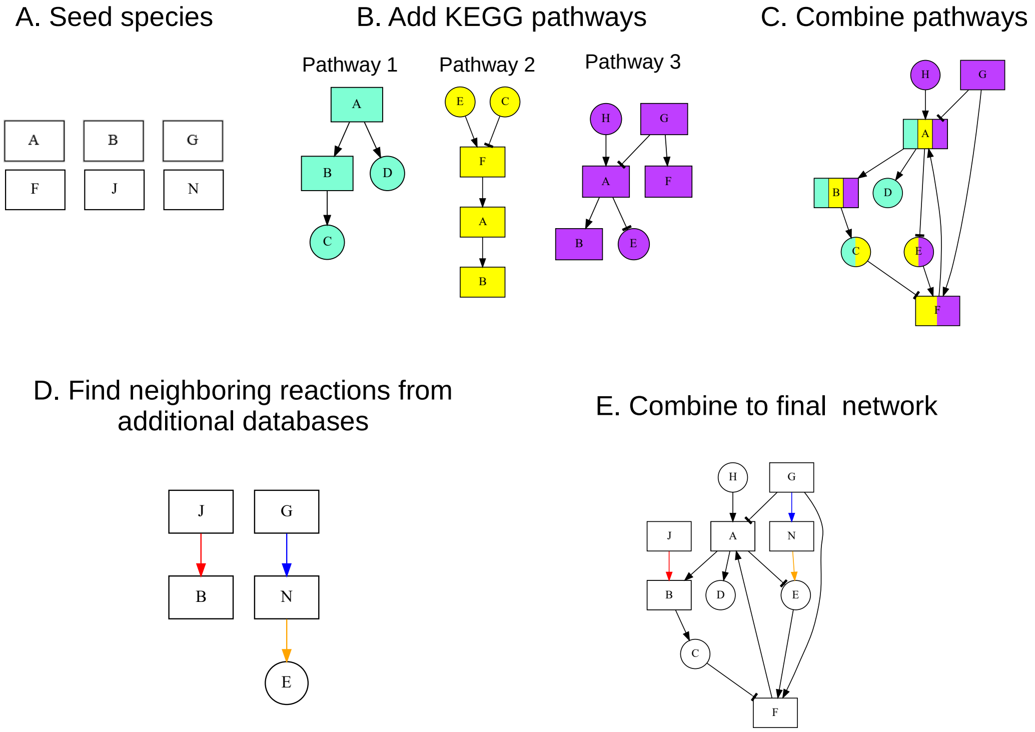
: _notebook_3_enrichment_analysis.pdf

: _notebook_4_agn.pdf
contains the example workflow to generate AGN.

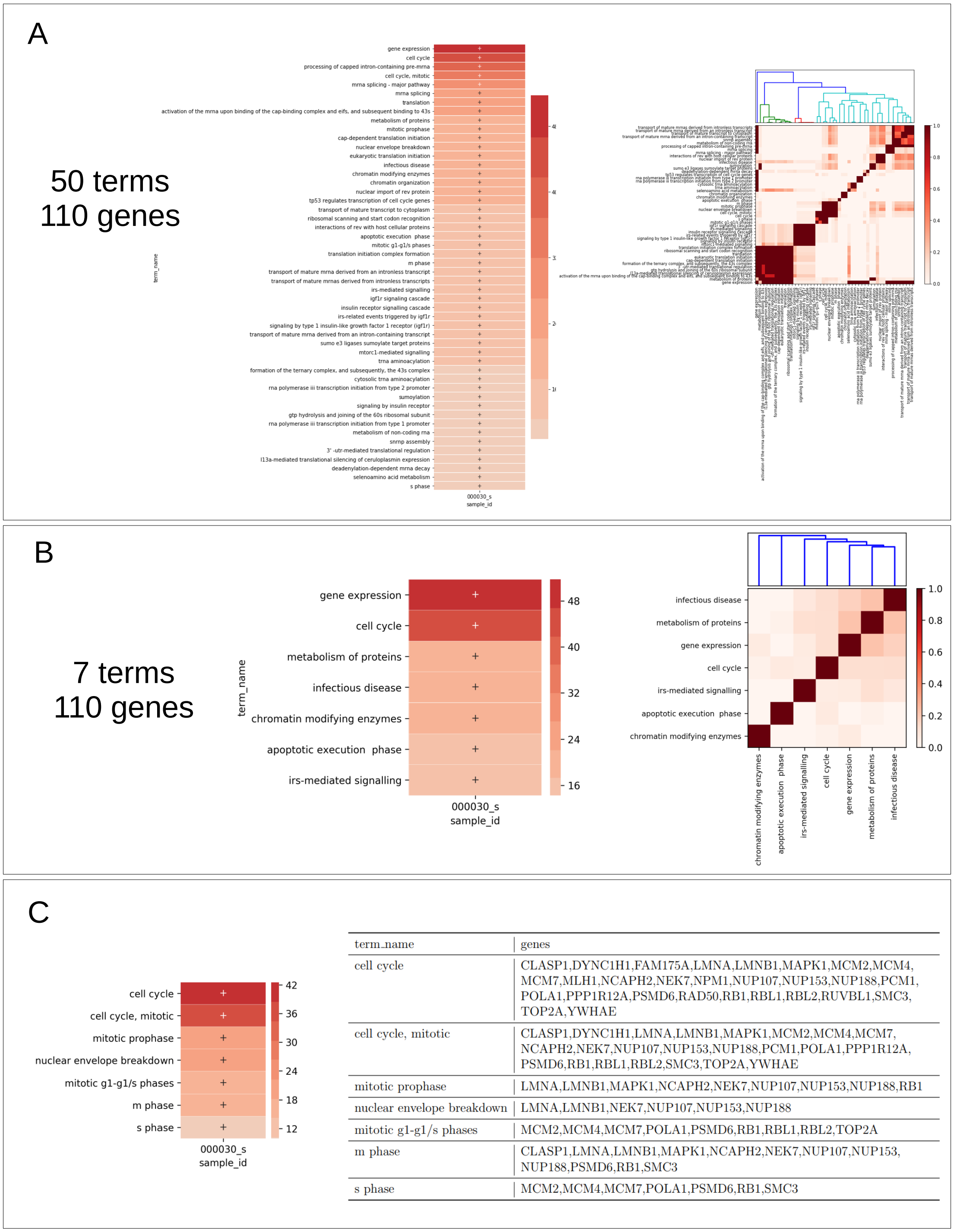
1.2 Animations and Videos

- **Supplementary File S5**: `supplement_agn_trajectory.gif`
  Network demonstrating aggregation of ontology terms performed at each time point (time-series animation). The area of each node is proportional to the enrichment of that ontology term at the indicated time point. The thickness of edges is related to the number of edges connecting the molecular nodes underlying the two term nodes (not time-dependent).

2 Supplementary Figures

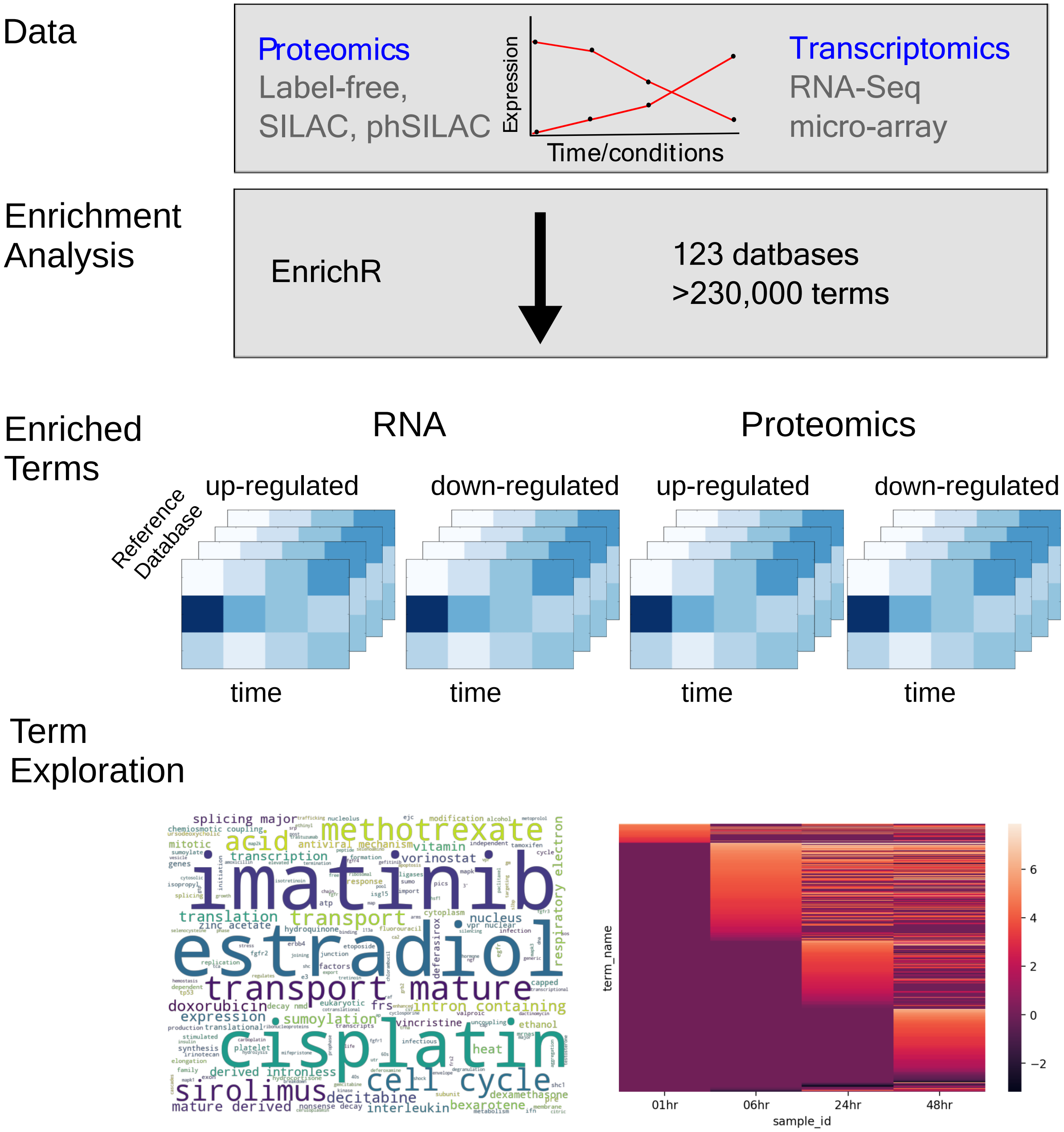


Supplementary Fig. S1: **Construction of data-seeded network.** Related to Figure 2. Example set of seed species to build network: (a) User provides seed species list; (b) KEGG pathways containing seed species are downloaded and merged; (c) Edges between seed species and nodes added by the KEGG pathways are identified; (d) All nodes and edges are combined into a single network, (e).

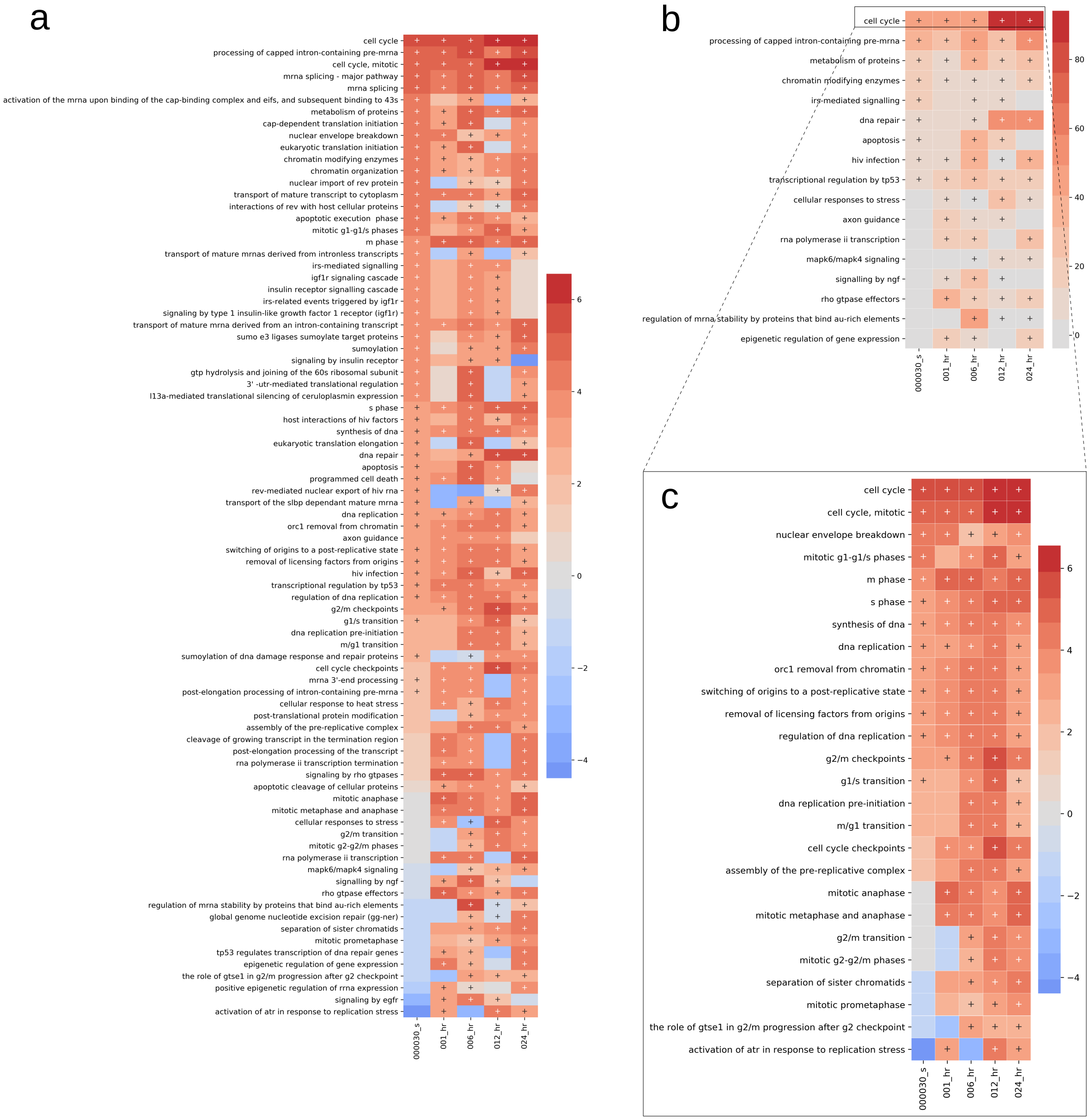


term_name	genes
cell cycle	CLASP1,DYNC1H1,FAM175A,LMNA,LMNB1,MAPK1,MCM2,MCM4,MCM7,MLH1,NCAPH2,NEK7,NPM1,NUP107,NUP153,NUP188,PCM1,POLA1,PPP1R12A,PSMD6,RAD50,RB1,RBL1,RBL2,RUVBL1,SMC3,TOP2A,YWHAE
cell cycle, mitotic	CLASP1,DYNC1H1,LMNA,LMNB1,MAPK1,MCM2,MCM4,MCM7,NCAPH2,NEK7,NUP107,NUP153,NUP188,PCM1,POLA1,PPP1R12A,PSMD6,RB1,RBL1,RBL2,SMC3,TOP2A,YWHAE
mitotic prophase	LMNA,LMNB1,MAPK1,NCAPH2,NEK7,NUP107,NUP153,NUP188,RB1
nuclear envelope breakdown	LMNA,LMNB1,NEK7,NUP107,NUP153,NUP188
mitotic g1-g1/s phases	MCM2,MCM4,MCM7,POLA1,PSMD6,RB1,RBL1,RBL2,TOP2A
m phase	CLASP1,LMNA,LMNB1,MAPK1,NCAPH2,NEK7,NUP107,NUP153,NUP188,PSMD6,RB1,SMC3
s phase	MCM2,MCM4,MCM7,POLA1,PSMD6,RB1,SMC3

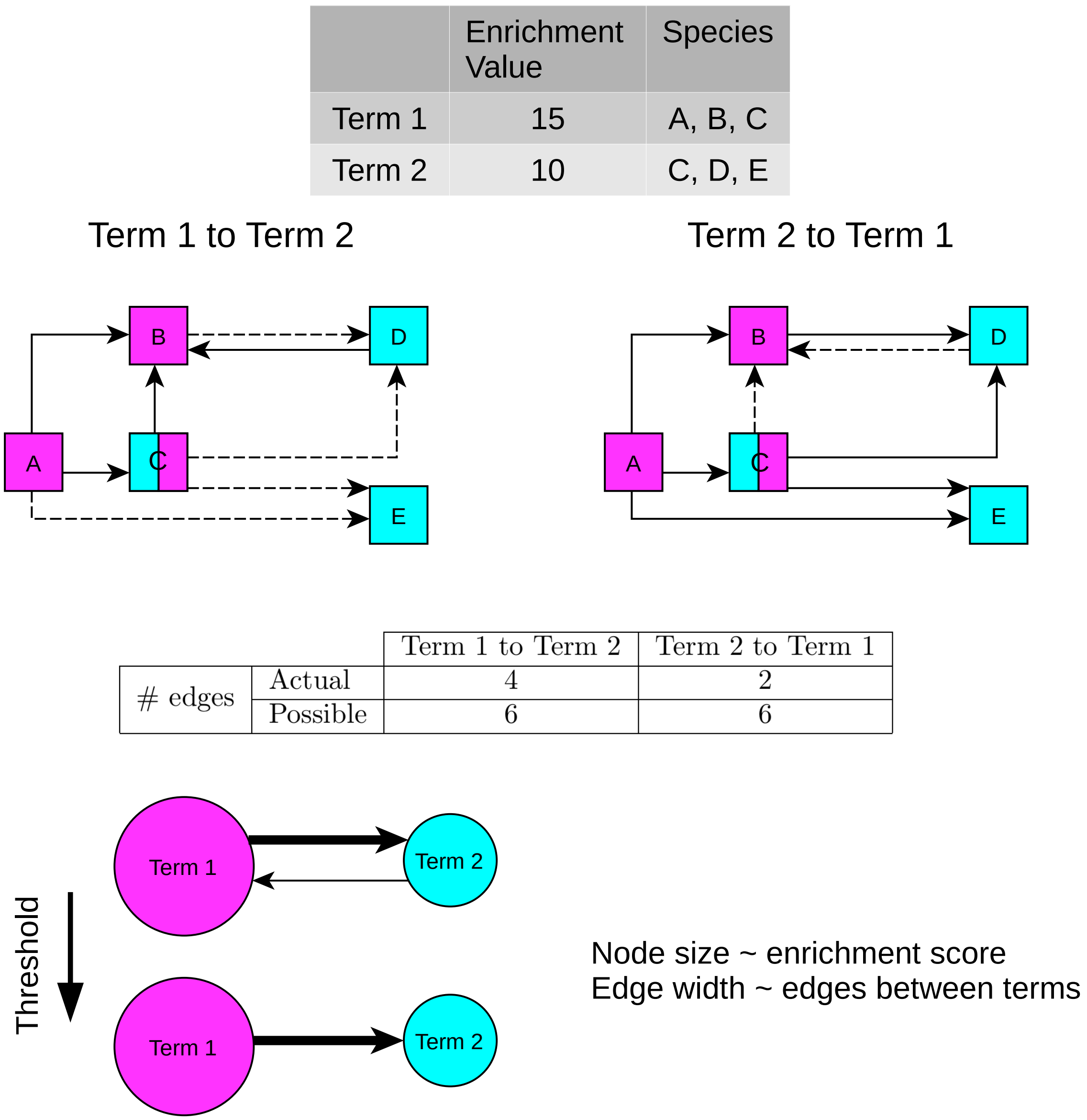
Supplementary Fig. S2: **Demonstration of enrichment compression** Related to Figure 4. A. Top 50 terms from Reactome for the 30 second phSILAC timepoint. The left figure shows the terms and their enriched values while the right shows the Jaccard index between all pairs. B. After filtering based on Jaccard index similarity. The resulting left figure has 7 terms, retaining all 110 genes that were contained in the terms from A. The right figure demonstrates that we have minimal overlap between corresponding terms. C. Demonstration of the collapsing of terms to a parent term. The left shows all terms that were collapsed from the term "cell cycle". The table on the right provides the genes that make up each term. Note that if a term was too broad ("cell cycle"), the user could remove it and repeat the process to maintain the level of detail which they require.



Supplementary Fig. S3: **Time series enrichment analysis pipeline**. Related to Figure 4. Abstract representation of the automation of running enrichment analysis on multiple time points and experimental platforms. We utilize EnrichR to access various databases. We then organize the data. This allows us to explore the output in various ways. We can use word clouds to compress various databases to see common terms across them all. We can also plot enriched terms over times to see trends of biological processes.



Supplementary Fig. S4: **MAGINE provides enrichment compression** Related to Figure 2. A. Enrichment results for the Reactome gene set. The left figure shows the terms and their enriched values, while the right shows the Jaccard index between all pairs. B. Resulting array after filtering based on Jaccard index similarity. The resulting left figure has 17 terms. C. Terms that have high overlap with *cell cycle*. MAGINE not only provides a method to compress the information of the enrichment array, it also provides methods to extract more detailed, but less enriched terms.

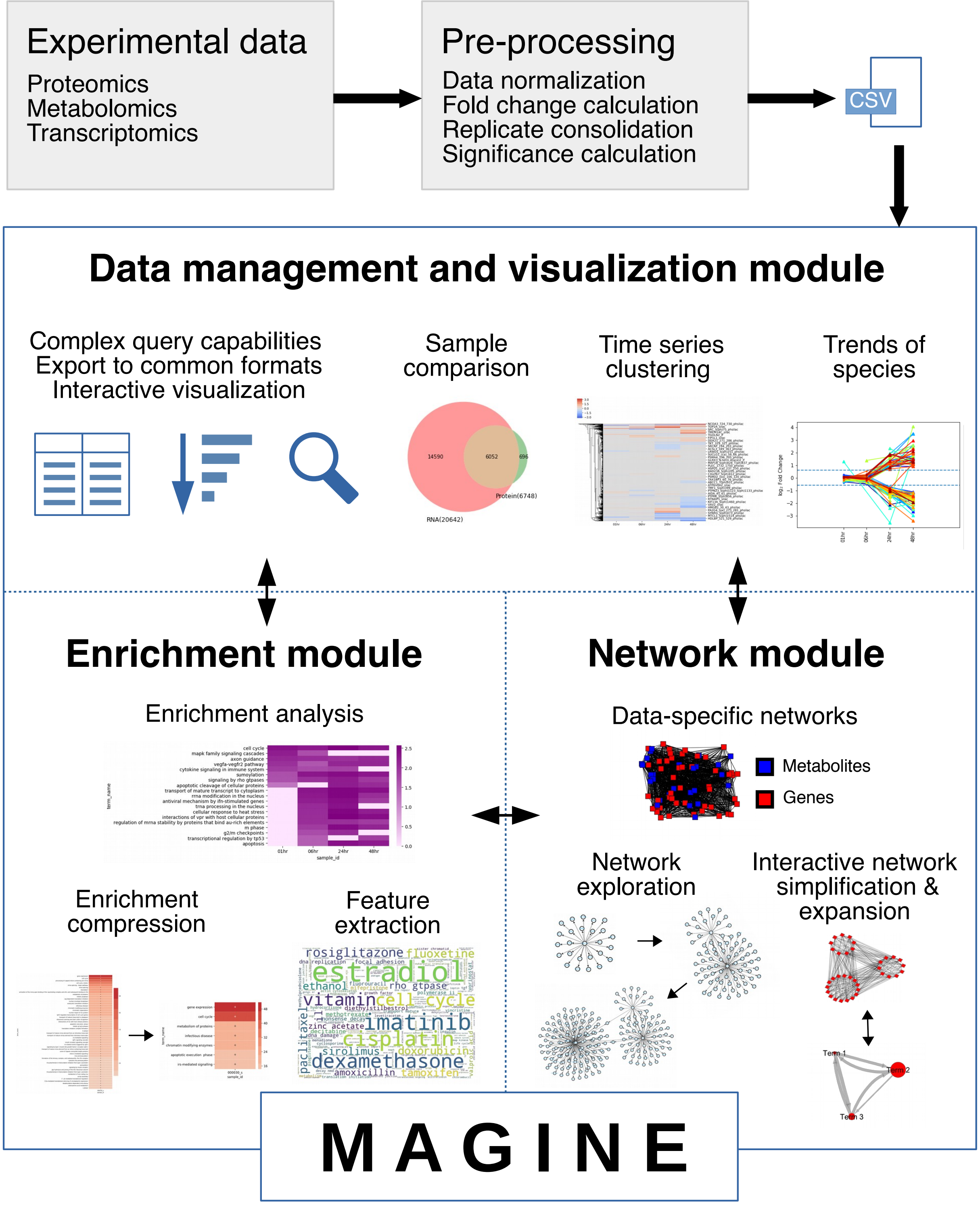


	Enrichment Value	Species
Term 1	15	A, B, C
Term 2	10	C, D, E

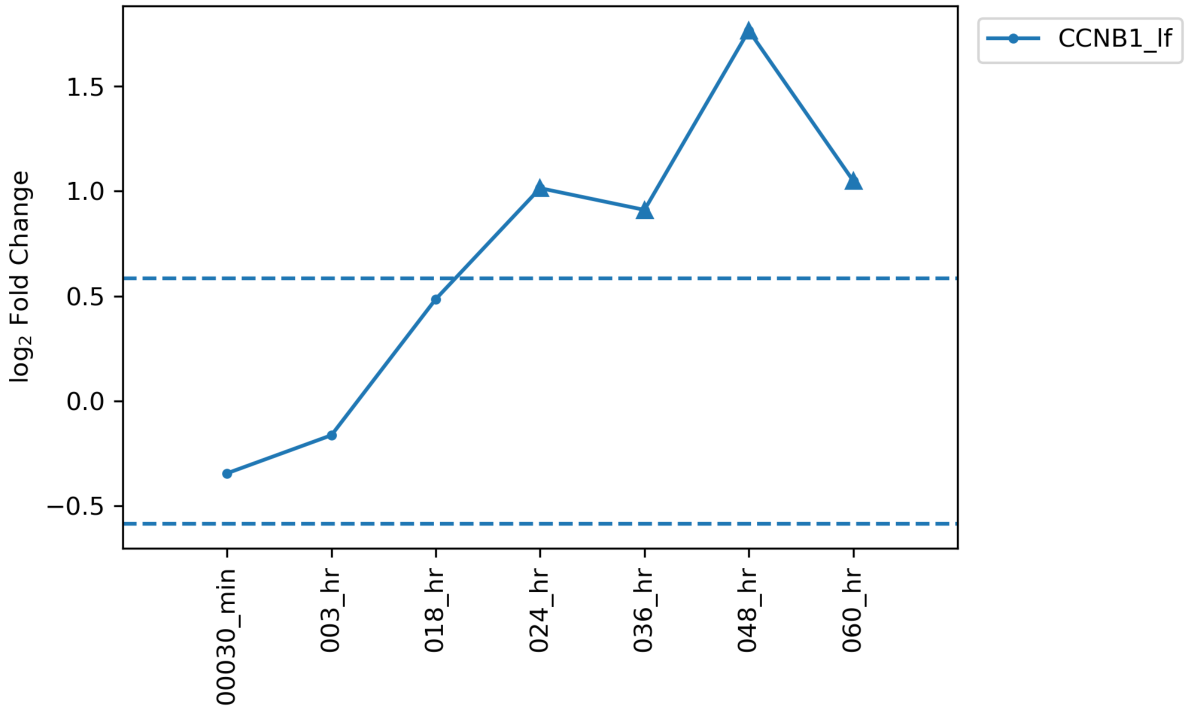
		Term 1 to Term 2	Term 2 to Term 1
# edges	Actual	4	2
	Possible	6	6

Node size ~ enrichment score
Edge width ~ edges between terms

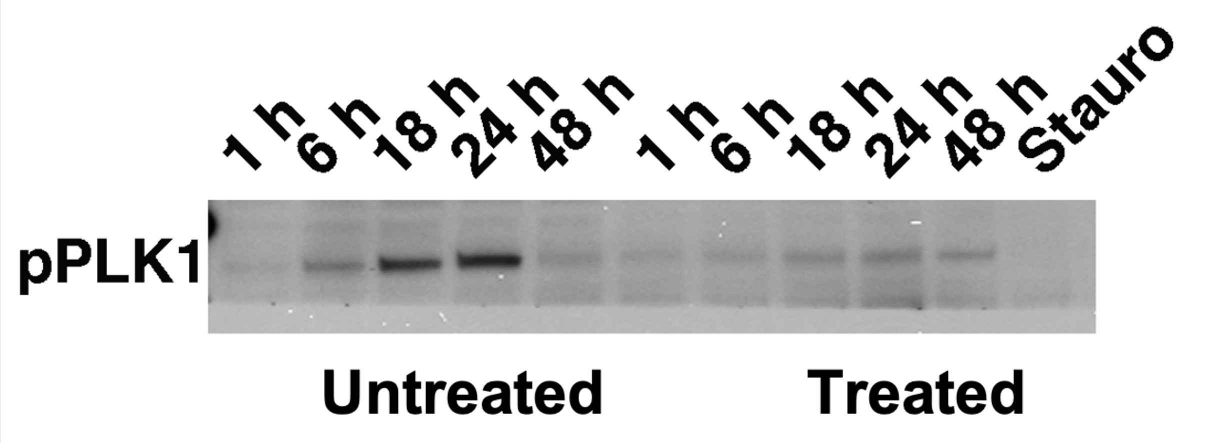
Supplementary Fig. S5: **Annotated gene set network construction** Related to Figure 5. Nodes belonging to Term 1 are labeled as pink, Term 2 as blue, and nodes in both are colored both. We calculate the number of edges between nodes in Term 1 to Term 2 and Term 2 and Term 1 (dashed lines), shown in middle table. We then create nodes for each Term, with the size of the node corresponding to the enrichment value. Edges are created between the terms and width set according to the number of edges between the terms. Finally, we applied a minimum of three edges threshold between terms to arrive at our final network.



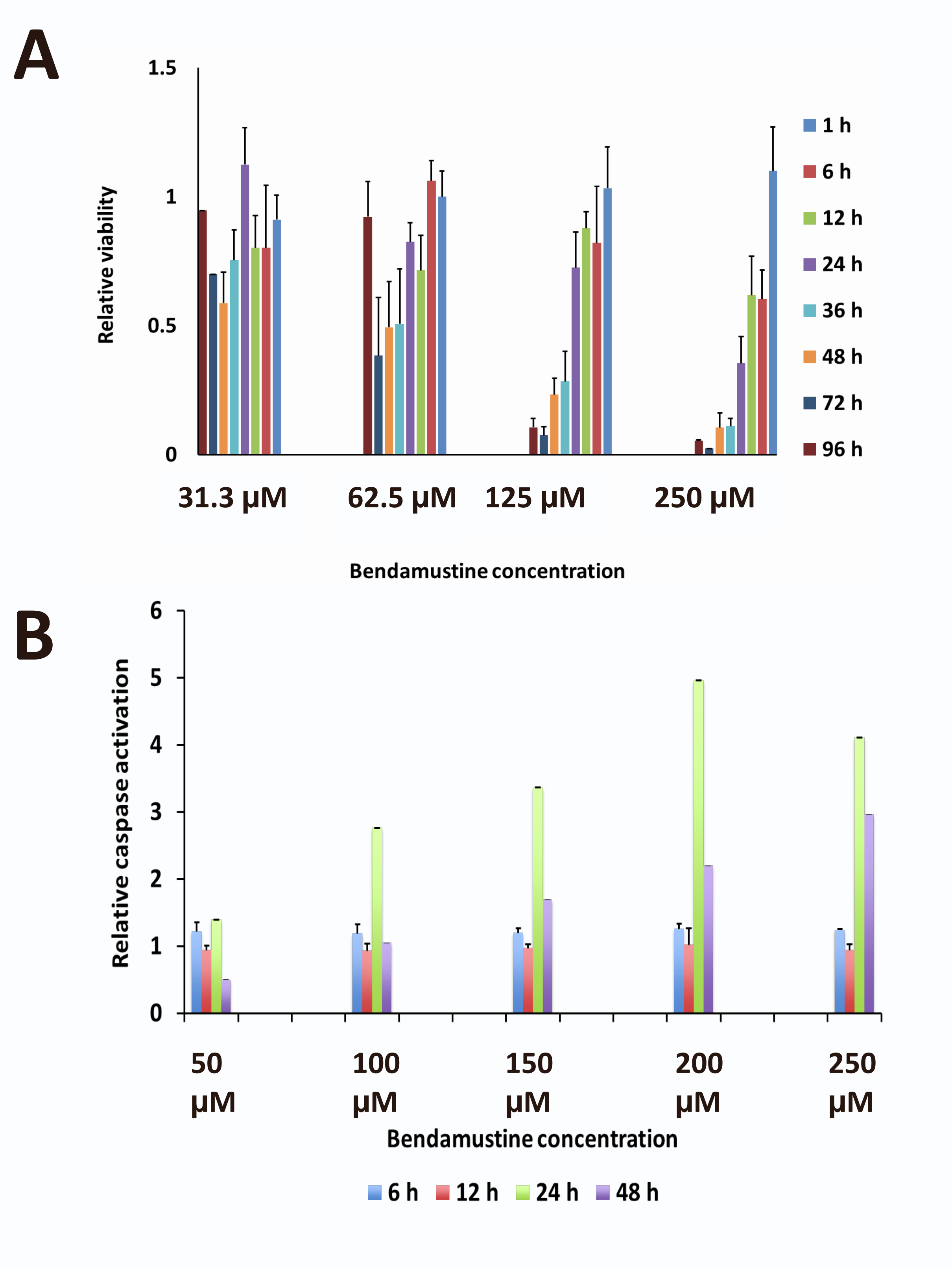
Supplementary Fig. S6: **The MAGINE platform.** Related to Figure 2. MAGINE is designed for quantitative time-series multi-omic data. It is built around three concepts: data management, enrichment analysis, and network exploration. The modular design allows flow of information among the data, enrichment, and network modules, allowing an iterative cycle with varying levels of resolution. Example outputs of each module are provided in each panel.



Supplementary Fig. S7: **Label-free CCNB1 expression.** Cyclin b (CCNB1) fold change versus control over time measured by label-free proteomics. Triangle markers depicts significantly changed versus control. Horizontal dashed lines mark +/- 1.5 fold change



Supplementary Fig. S8: **Western blot of pPLK1.** pPLK1 activity was measured at 1, 6, 18, 24, 48 hour time points in both bendamustine-treated and untreated HL-60 cells.



Supplementary Fig. S9: **Viability and caspase activity of bendamustine-treated HL-60 cells.** A. Relative viability of bendamustine-treated HL-60 cells versus untreated cells. B. Relative caspase activation of bendamustine-treated HL-60 cells versus untreated cells.

3 Supplementary Tables

Feature	**MAGINE**	**Metascape**	**Cytoscape**	**Enrichr**
Target audience	MAGINE serves a broad audience of quantitative biologists - bench and computational.	Metascape has been designed for experimentalists (bench biologists).	Biologists and bioinformaticians	Biologists and bioinformaticians
Workflow types	Human-in-the-loop mechanism exploration and optimization, iterative, dynamic, customizable, extendable through Python ecosystem.	Point-and-click input to output. Lacks ability to customize.	Point-and-click; extensible through Java plugins	Web-based, not extensible (not open source).
Multi-time point / multi-experiment capability	Native program-based interactivity. Can segment and filter time points and experimental platforms as needed for each analysis.	Can upload multiple files, but each is analyzed separately. No ability to combine and filter on-the-fly.	No native support (additional plugins required).	No native support (must upload separately).
Visualization.	Can be included inline within Jupyter Notebooks, as part of the analysis pipeline, or exported to external tools.	No connection to expression/experimental data values. Requires external tools.	Point-and-click network analysis (OS native).	Heat maps of enrichment terms.
Extendable and customizable of tools/workflow	Python ecosystem allows pre-processing of data, analysis, and further downstream analysis to be handled within a single environment.	Predefined options limited by developers (requires additional steps/tools for pre-processing and further analysis)	Extensible through plugins, but no workflow framework provided.	Web-based interface, could extend through programmatic API, but not alter due to lack of open source code.
Reproducibility, transferability	Supports git version control. Jupyter Notebooks and database downloads allow analysis code and inputs to be fixed and reproducible.	Monthly server updates - not guaranteed to be reproducible if the user reruns the same analysis at a later date.	Reproducible if never upgraded, otherwise user would have to track Cytoscape and plugin versions manually.	Regular server updates, but database versions can be specified. Access is reliant on web server being maintained in the future.
Transparency	Open-source - fully auditable, customizable	No public source code provided. Methods are therefore not fully auditable or customizable.	Open-source - fully auditable, customizable	No public source code provided. Methods are therefore not fully auditable or customizable.

Supplementary Table S1: Comparison of MAGINE's features and attributes to Metascape[**?**], Cytoscape[**?**], and Enrichr[**?**].

identifier	label	type	significant	fold_change	p_value	source	sample_id
BAX	BAX_S(ph)292	protein	True	4	0.01	SILAC	01hr
HMDB00012	Deoxyuridine	metabolite	True	-2	0.01	HILIC	02hr

Supplementary Table S2: Example input data for MAGINE. Data are stored in a comma-separated value (CSV) file.

	Name	Description
Function	load_data	Load MAGINE formatted csv
	create_summary_table	Create summary of counts per experimental method and sample_id
	subset	Filter data based on list of species
	require_n_sig	Filter data to include species that were measured at least N times
Plots	heatmap	Create heatmap or clustermap of species
	volcano_plot	Create volcano plot
	volcano_by_sample	Create a series of volcano plots for each sample
	plot_species	Plot scatter plots of selected species
	plot_histogram	Create histogram of fold change values
Properties	sig	Filter data to include only significant flagged entries
	up	Filter data to include fold change >0
	down	Filter data to include fold change <0
	id_list	Compile a list of unique species from sample
	by_sample	Generate list of unique species for each sample

Supplementary Table S3: Categorized list of key functions in MAGINE's `data` module.

	Name	Description
Functions	load_enrichment	Load MAGINE created enrichment analysis csv
	run	Run enrichment analysis for list of genes across selected gene sets
	run_samples	Run enrichment analysis for multiple lists of genes across selected gene sets
	run_enrichment_for_project	Run enrichment for entire ExperimentalData instance
	require_n_sig	Filter data to include species that were measured at least N times
	filter_multi	Filter by multiple criteria (p_value, combined_score, database, etc.)
	find_similar_terms	Rank order terms by similar gene sets
	filter_based_on_words	Filter by key words
	all_genes_from_df	Generate list of all genes in current EnrichmentResult instance
	term_to_genes	Generate list of genes for given term
	remove_redundant	Remove terms that are less enriched but highly similar
Plots	heatmap	Find all paths between list
	dist_matrix	Plot distance matrix of all terms

Supplementary Table S4: Categorized list of key functions in MAGINE's `enrichment` module

Name	Description
build_network	Generate network centered around provided species
create_subnetwork	Create annotated set network from enriched terms and background network
paths_between_pair	Find shortest path(s) between pair
paths_between_list	Find all paths between list
paths_between_two_lists	Find paths between two lists
neighbors	Find neighbors (upstream and/or downstream) of node
expand_neighbors	Add neighbors of node to network
draw	Draw using igraph, matplotlib, graphviz, cytoscape.js
RenderModel	Create Cytoscape instance of network through py2cytoscape

Supplementary Table S5: Categorized list of key functions in MAGINE's `network` module

Data S1. Experiment data exploration. Related to STAR METHODS.

1

1.1 ExperimentalData class demonstration

This purpose of this notebook is to introduce and demonstrate the ExperimentalData.

First we import python packages that we need. Please refer to tutorials on Scipy, NumPy, Matplotlib, and Seaborn if unfamilar with these powerful tools.

```
In [1]: from IPython.display import display
        %matplotlib inline
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import numpy as np
        from scipy.stats import pearsonr, spearmanr
```

```
In [2]: import magine.data.tools as dt
        from magine.plotting.wordcloud_tools import create_wordcloud
        from magine.plotting.venn_diagram_maker import create_venn2, create_venn3
```

1.2 ExperimentalData class structure

Since MAGINE is built for multi-sample, multi-omics data, it is no surprise that the data is the most important aspect. Here we should how to use the :py:class:`ExperimentalData` class.

```
In [3]: # load the experimental data
        from magine.data.experimental_data import load_data
```

```
In [4]: help(load_data)
```

```
Help on function load_data in module magine.data.experimental_data:

load_data(file_name, **kwargs)
    Load data into EnrichmentResult data class

    Parameters
    ----------
    file_name : str
    kwargs :
        Flags to pass to pandas.

    Returns
    -------
    df : EnrichmentResult
```

```
In [5]: exp_data = load_data(
            'Data/bendamustine.csv.gz', # filename and location
            # Following args are passed to pandas.read_csv
            low_memory=False,
            index_col=0
        )
```

1.2.1 Getting counts from data

First, lets quickly view the stats about the data.

```
In [6]: display(exp_data.create_summary_table())
        display(exp_data.create_summary_table(index='label'))
```

```
sample_id  000030_s 00030_min  001_hr 003_hr  006_hr  012_hr 018_hr  024_hr  \
source
C18            5735      5114    5721   5834    6313    6574   6201    4531
HILIC         11891      9412   11880   7882   14215   14702  12666   10451
label_free     3215      3451    3113   4098    2907    3150   4273    4374
ph_silac       3240         -    3495      -    3327    3756      -    3212
rna_seq           -         -   16550      -   15887   16017      -   16418
silac          1629         -    1883      -    1761    1650      -    1664

sample_id  036_hr 048_hr 060_hr 072_hr  Total Unique Across
source
C18          7825   6267   4751   4773                19570
HILIC       11902  13013  10804   6350                27959
label_free   4100   4448   4188   2628                 5611
ph_silac        -      -      -      -                 4877
rna_seq         -      -      -      -                17679
silac           -      -      -      -                 2323


sample_id  000030_s 00030_min  001_hr 003_hr  006_hr  012_hr 018_hr  024_hr  \
source
C18            5629      5014    5626   5729    6188    6454   6074    4458
HILIC         11754      9314   11758   7781   14038   14530  12493   10322
label_free     3730      4113    3553   4717    3329    3645   4953    5058
ph_silac      12224         -   14512      -   12709   15472      -   12252
rna_seq           -         -   16550      -   15887   16017      -   16418
```

```
silac           1629          -    1883       -    1761    1650       -    1664

sample_id  036_hr 048_hr 060_hr 072_hr  Total Unique Across
source
C18          7707   6155   4670   4687                19212
HILIC       11746  12869  10684   6271                27572
label_free   4651   5263   4767   2911                 7428
ph_silac        -      -      -      -                25613
rna_seq         -      -      -      -                17679
silac           -      -      -      -                 2323
```

From here, we can see that we have 12 time points and 6 experimental platforms for the data. This is of all the data. We can filter by significantly measured or by looking at `label` column (default is `identifier`

```
In [7]: display(exp_data.create_summary_table(sig=True))
        display(exp_data.create_summary_table(sig=True, index='label'))
```

```
sample_id  000030_s 00030_min  001_hr 003_hr  006_hr  012_hr 018_hr  024_hr  \
source
C18             870       121     454    555     444     322    293     341
HILIC           729       452     226    891     354    1053    732     410
label_free       14        18      22     37     113      18     85     161
ph_silac        609         -     883      -    1091     722      -     944
rna_seq           -         -      51      -      51      69      -     611
silac            20         -      30      -      19      20      -      58

sample_id  036_hr 048_hr 060_hr 072_hr  Total Unique Across
source
C18          1032    684    787   1224                 5414
HILIC          83   2118    116    137                 6244
label_free     39    162    853    542                 1483
ph_silac        -      -      -      -                 2437
rna_seq         -      -      -      -                  736
silac           -      -      -      -                  133


sample_id  000030_s 00030_min  001_hr 003_hr  006_hr  012_hr 018_hr  024_hr  \
source
C18             840       119     443    542     435     314    289     335
HILIC           699       437     215    866     353    1044    713     394
label_free       14        18      22     37     114      18     86     168
ph_silac        755         -    1189      -    1525     975      -    1570
rna_seq           -         -      51      -      51      69      -     611
silac            20         -      30      -      19      20      -      58

sample_id  036_hr 048_hr 060_hr 072_hr  Total Unique Across
source
```

```
C18           1014    670    776   1198                  5296
HILIC           83   2094    115    134                  6138
label_free      39    170    925    591                  1653
ph_silac         -      -      -      -                  5115
rna_seq          -      -      -      -                   736
silac            -      -      -      -                   133
```

The .species index aggregates all data. Since we utilize a pandas.DataFrame, we can use the .head method to glance at the data.

```
In [8]: exp_data.species.head(5)

Out[8]:   identifier           label  fold_change  significant  p_value species_type  \
        0       UBA6      UBA6_silac    -1.049913        False      1.0      protein
        1       MTDH      MTDH_silac    -1.038867        False      1.0      protein
        2   SLC25A24  SLC25A24_silac     1.014615        False      1.0      protein
        3    ANKRD22   ANKRD22_silac    -1.058937        False      1.0      protein
        4        AGK       AGK_silac     1.001600        False      1.0      protein

          sample_id source
        0    001_hr  silac
        1    001_hr  silac
        2    001_hr  silac
        3    001_hr  silac
        4    001_hr  silac
```

We can filter the data by source using the .name, where name is anything in the source column. We can get a list of these by printing exp_data.exp_methods

```
In [9]: exp_data.exp_methods

Out[9]: ['silac', 'ph_silac', 'HILIC', 'C18', 'label_free', 'rna_seq']

In [10]: # filters to only the 'label_free'
         exp_data.label_free.shape

Out[10]: (50736, 8)

In [11]: exp_data.label_free.head(5)

Out[11]:        identifier                  label  fold_change  significant  p_value  \
         515804    RHOXF2B             RHOXF2B_lf         1.51         True   0.0003
         515805      KIF11               KIF11_lf         1.70         True   0.0008
         515806   MYBBP1A  MYBBP1A_S(ph)1163_lf         1.40        False   0.0009
         515807      CDC20               CDC20_lf         2.78         True   0.0012
         515808       TMPO        TMPO_T(ph)160_lf        -1.47        False   0.0016

               species_type sample_id      source
```

```
515804      protein    036_hr  label_free
515805      protein    036_hr  label_free
515806      protein    036_hr  label_free
515807      protein    036_hr  label_free
515808      protein    036_hr  label_free
```

```
In [12]: exp_data.rna_seq.head(5)
```

```
Out[12]:        identifier              label  fold_change  significant   p_value  \
         566540      CDC20       CDC20_rnaseq    -1.528232         True  0.016321
         566541       ASPM        ASPM_rnaseq    -1.346689        False  0.038802
         566542 FO538757.2  FO538757.2_rnaseq     2.133888         True  0.038802
         566543       GNL3        GNL3_rnaseq    -2.313025         True  0.016321
         566544    SNORD19     SNORD19_rnaseq    -2.313025         True  0.016321

                species_type sample_id   source
         566540      rna_seq    012_hr  rna_seq
         566541      rna_seq    012_hr  rna_seq
         566542      rna_seq    012_hr  rna_seq
         566543      rna_seq    012_hr  rna_seq
         566544      rna_seq    012_hr  rna_seq
```

1.2.2 Significant filter

We can use the `significant` column to filter that data to only contain those species.

```
In [13]: exp_data.species.shape
```

```
Out[13]: (558025, 8)
```

```
In [14]: exp_data.species.sig.shape
```

```
Out[14]: (24620, 8)
```

1.2.3 Filter data to up or down regulated species.

For enrichment analysis, we will want to access up-regulated and down-regulated species using `.up` and `.down`.

```
In [15]: exp_data.rna_seq.up.head(5)
```

```
Out[15]:           identifier                 label  fold_change  significant  \
         566542     FO538757.2     FO538757.2_rnaseq     2.133888         True
         566546 RP4-669L17.10  RP4-669L17.10_rnaseq     3.788399         True
         566547  RP4-669L17.4   RP4-669L17.4_rnaseq     3.788399         True
         566548    RNU6-513P      RNU6-513P_rnaseq     2.462533         True
         566549         RRP7B          RRP7B_rnaseq     2.462533         True

                  p_value species_type sample_id   source
         566542  0.038802      rna_seq    012_hr  rna_seq
         566546  0.029804      rna_seq    012_hr  rna_seq
         566547  0.029804      rna_seq    012_hr  rna_seq
         566548  0.016321      rna_seq    012_hr  rna_seq
         566549  0.016321      rna_seq    012_hr  rna_seq
```

```
In [16]: exp_data.rna_seq.down.head(5)
```

```
Out[16]:        identifier            label  fold_change  significant   p_value  \
         566540      CDC20     CDC20_rnaseq    -1.528232         True  0.016321
         566543       GNL3      GNL3_rnaseq    -2.313025         True  0.016321
         566544    SNORD19   SNORD19_rnaseq    -2.313025         True  0.016321
         566545   SNORD19B  SNORD19B_rnaseq    -2.313025         True  0.016321
         566550     FAM73A    FAM73A_rnaseq    -9.644115         True  0.016321

                species_type sample_id   source
         566540      rna_seq    012_hr  rna_seq
         566543      rna_seq    012_hr  rna_seq
         566544      rna_seq    012_hr  rna_seq
         566545      rna_seq    012_hr  rna_seq
         566550      rna_seq    012_hr  rna_seq
```

1.2.4 Extracting by sample (time point)

We can filter by `sample_id`.

```
In [17]: exp_data.sample_ids
```

```
Out[17]: ['000030_s',
          '00030_min',
          '001_hr',
          '003_hr',
          '006_hr',
          '012_hr',
          '018_hr',
          '024_hr',
          '036_hr',
          '048_hr',
          '060_hr',
          '072_hr']
```

```
In [18]: exp_data['000030_s'].head(5)
```

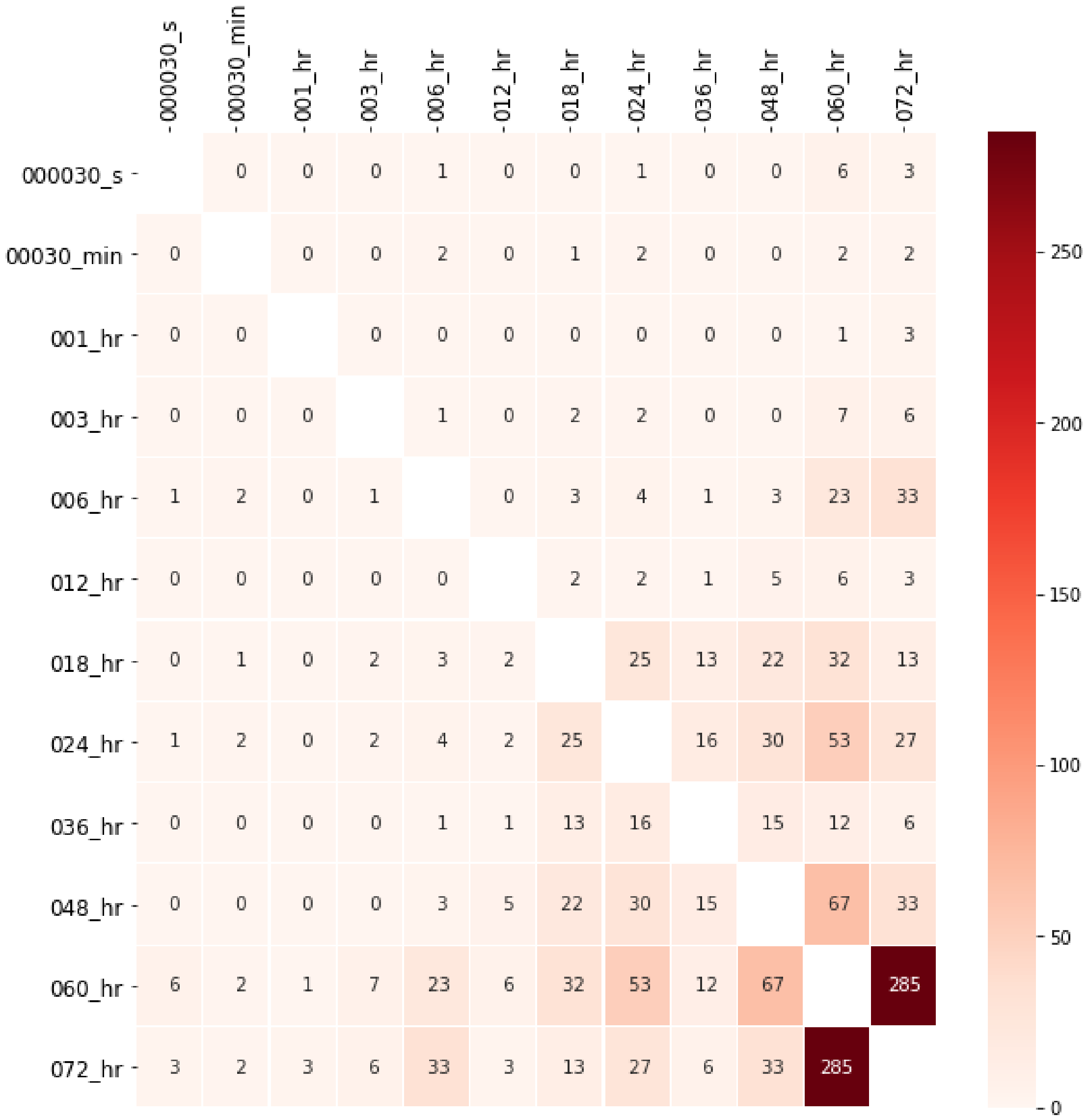
```
Out[18]:      identifier         label  fold_change  significant  p_value species_type  \
         5200       UBA6    UBA6_silac    -1.088427        False      1.0      protein
         5201     AKR1A1  AKR1A1_silac     1.065195        False      1.0      protein
         5202     MTHFD2  MTHFD2_silac    -1.308449        False      1.0      protein
         5203       DLAT    DLAT_silac     1.029963        False      1.0      protein
         5204        CNP     CNP_silac    -1.244300        False      1.0      protein
```

```
     sample_id source
5200  000030_s  silac
5201  000030_s  silac
5202  000030_s  silac
5203  000030_s  silac
5204  000030_s  silac
```

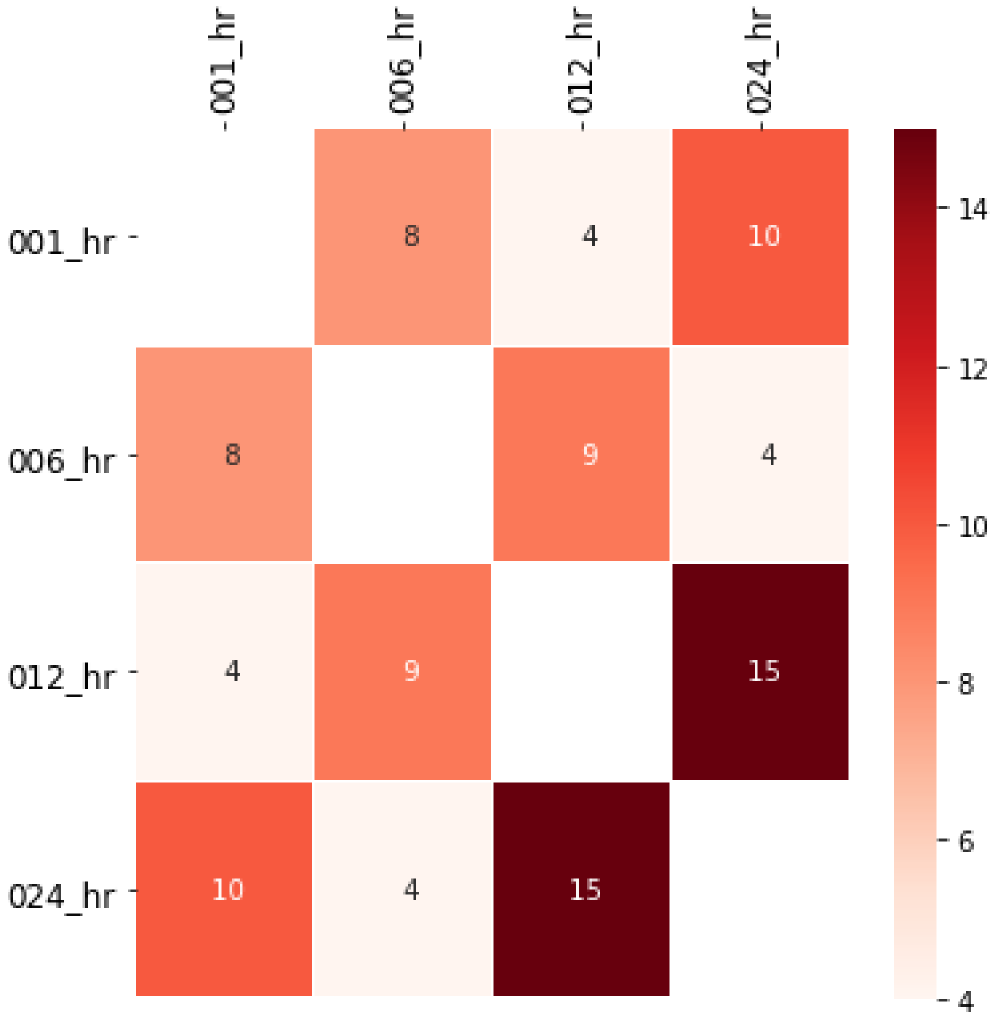
1.2.5 Calculating overlaps between time points

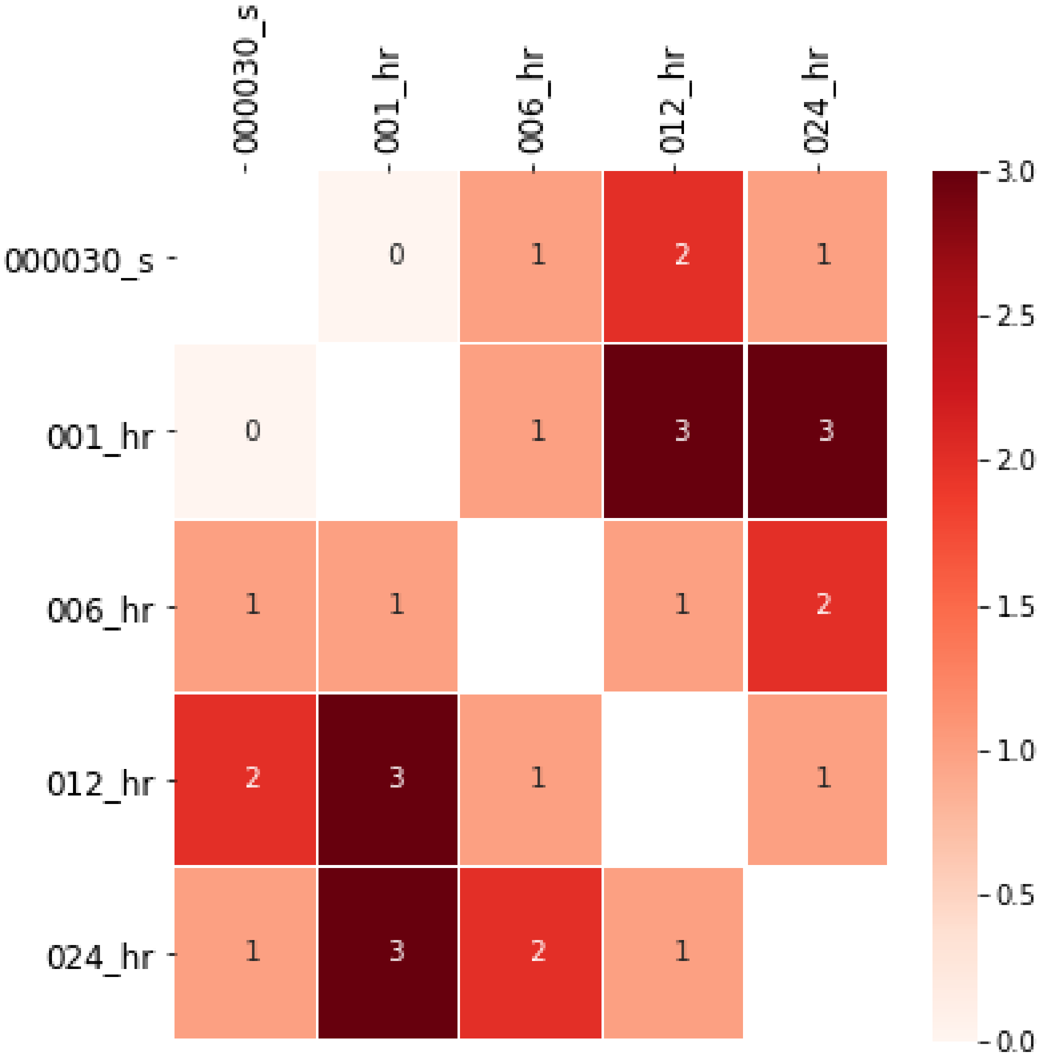
```
In [19]: from itertools import combinations

         def overlap(vals):
             return len(vals[0].intersection(vals[1]))

         def calc_dist(names, gene_sets, figsize=(12, 12)):

             n_dim = len(names)
             scores = list(map(overlap, combinations(gene_sets, 2)))

             dist_mat = np.zeros((n_dim, n_dim), dtype=float)
             ind = 0
             for i in range(n_dim):
                 for j in range(i, n_dim):
                     if i == j:
                         dist_mat[i, i] = np.nan
                         continue
                     elif i>= j:
                         continue
                     dist_mat[i, j] = scores[ind]
                     dist_mat[j, i] = scores[ind]
                     ind += 1

             fig = plt.figure(figsize=figsize)
             ax = fig.add_subplot(111)
             cmap=plt.cm.Reds
             cmap.set_under(".5")
             fig = sns.heatmap(dist_mat, cmap=cmap, fmt='3g', annot=True, linewidths=0.1,
                         xticklabels=names, yticklabels=names, ax=ax, square=False)
             ax.xaxis.tick_top()
             ax.set_xticklabels(names, minor=False, rotation=90, fontsize=12)
             ax.set_yticklabels(names, minor=False, rotation=0, fontsize=12)


         def overlap_by_source(source_name, figsize=(10, 10)):
             sample_ids = np.array(exp_data[source_name].sample_ids)
             sample_sets = exp_data[source_name].sig.by_sample
             calc_dist(sample_ids, sample_sets, figsize)
```

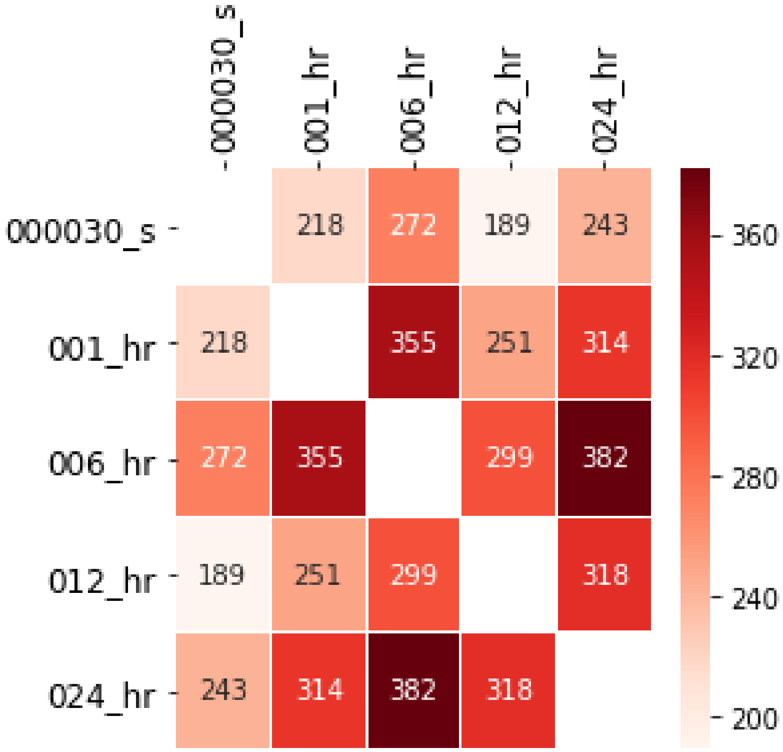
```
overlap_by_source('label_free')
overlap_by_source('rna_seq', figsize=(6, 6))
overlap_by_source('silac', figsize=(6, 6))
overlap_by_source('ph_silac', figsize=(4, 4))
plt.savefig("ph_silac_overlap_by_time.png", dpi=300, bbox_inches='tight')
```



	000030_s	00030_min	001_hr	003_hr	006_hr	012_hr	018_hr	024_hr	036_hr	048_hr	060_hr	072_hr
000030_s		0	0	0	1	0	0	1	0	0	6	3
00030_min	0		0	0	2	0	1	2	0	0	2	2
001_hr	0	0		0	0	0	0	0	0	0	1	3
003_hr	0	0	0		1	0	2	2	0	0	7	6
006_hr	1	2	0	1		0	3	4	1	3	23	33
012_hr	0	0	0	0	0		2	2	1	5	6	3
018_hr	0	1	0	2	3	2		25	13	22	32	13
024_hr	1	2	0	2	4	2	25		16	30	53	27
036_hr	0	0	0	0	1	1	13	16		15	12	6
048_hr	0	0	0	0	3	5	22	30	15		67	33
060_hr	6	2	1	7	23	6	32	53	12	67		285
072_hr	3	2	3	6	33	3	13	27	6	33	285	

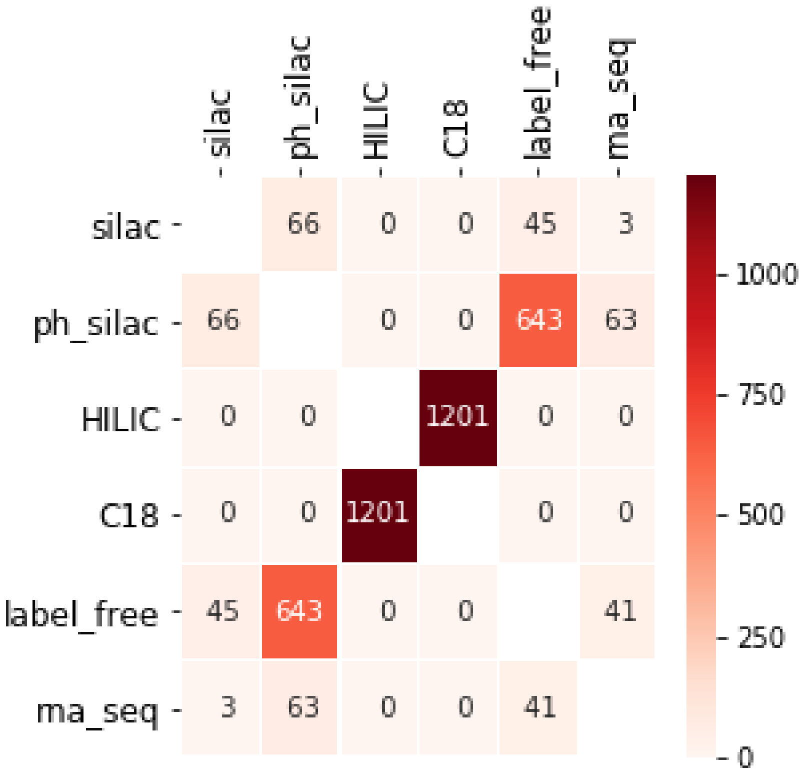

001_hr
006_hr
012_hr
024_hr
001_hr
006_hr
012_hr
024_hr
8
4
10
8
9
4
4
9
15
10
4
15
14
12
10
8
6
4


000030_s
001_hr
006_hr
012_hr
024_hr
000030_s
0
1
2
1
001_hr
0
1
3
3
006_hr
1
1
1
2
012_hr
2
3
1
1
024_hr
1
3
2
1
3.0
2.5
2.0
1.5
1.0
0.5
0.0



We can apply the same function to compare across experimental platforms.

```
In [20]: names, sets = [], []
         for i in exp_data.exp_methods:
             sets.append(exp_data[i].sig.id_list)
             names.append(i)
         calc_dist(names, sets, figsize=(4, 4))
         plt.savefig("overlap_experiments.png", dpi=300, bbox_inches='tight')
```



1.2.6 Pivot table to get table across time

```
In [21]: exp_data.rna_seq.pivoter(
             convert_to_log=False,
             index='identifier',
             columns='sample_id',
             values=['fold_change', 'p_value']
         ).head(5)
```

```
Out[21]:            fold_change                                 p_value           \
         sample_id       001_hr    006_hr    012_hr    024_hr    001_hr   006_hr
         identifier
         7SK           1.139800 -1.148785 -1.014452  1.037751  0.999631  0.99995
         A1BG         -2.402129  1.288824  1.610149  1.168061  0.999631  0.99995
         A1BG-AS1     -1.081850  1.257409  1.056497 -1.099355  0.999631  0.99995
         A2ML1        -1.372659  1.307877 -1.343714  1.793100  0.999631  0.99995
         AAAS          1.203116 -1.058658 -1.012985  1.007011  0.999631  0.99995


         sample_id     012_hr    024_hr
         identifier
         7SK         0.999660  0.864095
```

```
A1BG        0.803284  0.673174
A1BG-AS1    0.999660  0.752914
A2ML1       0.999660  0.274612
AAAS        0.999660  0.970896
```

```
In [22]: exp_data.ph_silac.pivoter(
             convert_to_log=False,
             index='label',
             columns='sample_id',
             values=['fold_change', 'p_value']
         ).head(5)
```

```
Out[22]:                       fold_change                                           \
         sample_id                000030_s    001_hr    006_hr     012_hr    024_hr
         label
         A2M_1004_1014_phsilac         NaN -1.060800       NaN  43.747479       NaN
         A2M_339_345_phsilac           NaN       NaN       NaN  -1.157100       NaN
         AAAS_S(ph)495_phsilac    1.121971 -1.230439 -1.159737  -1.288961 -1.486965
         AAGAB_134_161_phsilac    1.134301       NaN       NaN        NaN       NaN
         AAGAB_274_284_phsilac         NaN       NaN       NaN  -1.123760       NaN

                                p_value
         sample_id             000030_s 001_hr 006_hr 012_hr 024_hr
         label
         A2M_1004_1014_phsilac      NaN    1.0    NaN  0.049    NaN
         A2M_339_345_phsilac        NaN    NaN    NaN  1.000    NaN
         AAAS_S(ph)495_phsilac      1.0    1.0    1.0  1.000    1.0
         AAGAB_134_161_phsilac      1.0    NaN    NaN    NaN    NaN
         AAGAB_274_284_phsilac      NaN    NaN    NaN  1.000    NaN
```

Note that in the previous example, we find that there are NaN values. This is because there might be measurements missing in our experimental data. We can easy check what species are not found in all samples.

```
In [23]: measured_in_all = exp_data.ph_silac.present_in_all_columns(
             index='label',
             columns='sample_id',
         )
```

```
Number in index went from 25613 to 5595
```

```
In [24]: measured_in_all.pivoter(
             convert_to_log=False,
             index='label',
             columns='sample_id',
             values=['fold_change', 'p_value']
         ).head(10)
```

```
Out[24]:                                         fold_change                      \
         sample_id                                  000030_s    001_hr    006_hr
         label
         AAAS_S(ph)495_phsilac                      1.121971 -1.230439 -1.159737
         AAGAB_S(ph)310_S(ph)311_phsilac           -0.031090  1.711100 -1.557812
         AAK1_T(ph)606_phsilac                     -1.105709 -1.013440  1.033784
         AAK1_T(ph)620_S(ph)623_phsilac             1.003735 -1.002832 -0.015575
         AARS_(ca)_173_194_phsilac                  1.039944  1.243000  1.254237
         AARS_225_235_phsilac                       1.094279  1.107042  1.051156
         AASDHPPT_253_267_phsilac                   1.007512  1.013656  1.002298
         AATF_S(ph)316_S(ph)320_S(ph)321_phsilac   -0.020251 -1.538099  1.119900
         ABCE1_(ox)_542_557_phsilac                -1.254600 -1.226600  1.005787
         ABCE1_213_224_phsilac                      1.072050 -1.006802  1.097534

                                                                      p_value         \
         sample_id                                  012_hr    024_hr 000030_s 001_hr
         label
         AAAS_S(ph)495_phsilac                   -1.288961 -1.486965      1.0  1.000
         AAGAB_S(ph)310_S(ph)311_phsilac         -0.000927 -1.657294      1.0  0.049
         AAK1_T(ph)606_phsilac                   -2.708685 -2.866159      1.0  1.000
         AAK1_T(ph)620_S(ph)623_phsilac          -1.082963 -1.051561      1.0  1.000
         AARS_(ca)_173_194_phsilac               -1.347523 -1.034186      1.0  1.000
         AARS_225_235_phsilac                    -1.199240  1.101794      1.0  1.000
         AASDHPPT_253_267_phsilac                -1.222638  1.148329      1.0  1.000
         AATF_S(ph)316_S(ph)320_S(ph)321_phsilac -1.107462 -1.393000      1.0  0.049
         ABCE1_(ox)_542_557_phsilac              -1.732200  1.001636      1.0  1.000
         ABCE1_213_224_phsilac                   -1.155031 -1.085661      1.0  1.000


         sample_id                               006_hr 012_hr 024_hr
         label
         AAAS_S(ph)495_phsilac                    1.000  1.000  1.000
         AAGAB_S(ph)310_S(ph)311_phsilac          0.049  1.000  1.000
         AAK1_T(ph)606_phsilac                    1.000  0.049  0.049
         AAK1_T(ph)620_S(ph)623_phsilac           1.000  1.000  1.000
         AARS_(ca)_173_194_phsilac                1.000  1.000  1.000
         AARS_225_235_phsilac                     1.000  1.000  1.000
         AASDHPPT_253_267_phsilac                 1.000  1.000  1.000
         AATF_S(ph)316_S(ph)320_S(ph)321_phsilac  1.000  1.000  1.000
         ABCE1_(ox)_542_557_phsilac               1.000  0.049  1.000
         ABCE1_213_224_phsilac                    1.000  1.000  1.000
```

This shows that out of the 25613 unique species measured in ph_silac proteomics, only 5595 were measured in all time points. What one can do with this information is dependent on the analysis. For now, we will keep using the full dataset.

We can use this same principle and filter that data by requiring a species to be significant in `n` `sample_id`.

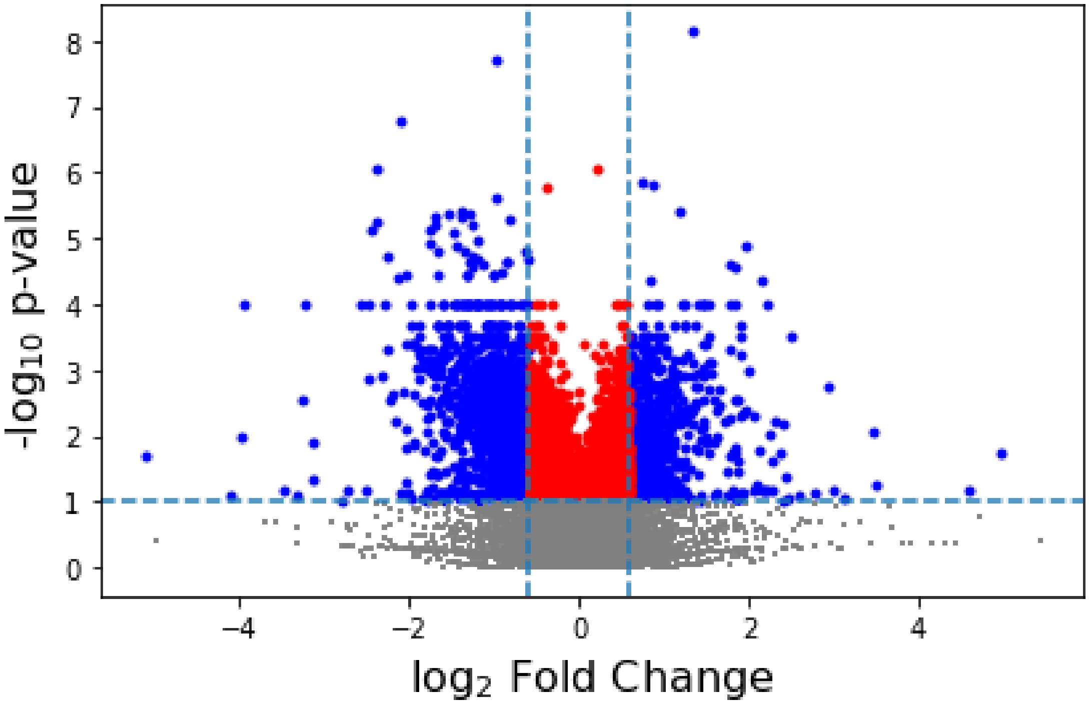
```
In [25]: lf_4_tp = exp_data.label_free.require_n_sig(
            index='label',
            columns='sample_id',
            n_sig=4
        )
        display(lf_4_tp.head(10))
```

```
       identifier                       label  fold_change  significant  \
515805      KIF11                    KIF11_lf         1.70         True
515807      CDC20                    CDC20_lf         2.78         True
515810      CKAP2                    CKAP2_lf         1.63         True
515813       TPX2                     TPX2_lf         1.93         True
515814      KIFC1                    KIFC1_lf         1.73         True
515822       RRM2                     RRM2_lf         1.92         True
515825     NUSAP1                   NUSAP1_lf         1.92         True
515827      CDCA5                    CDCA5_lf         1.76         True
515831      UBE2S                    UBE2S_lf         1.52         True
515835   HIST1H1B  HIST1H1B_N-term S(ace)2_lf        -1.51         True

        p_value species_type sample_id      source
515805   0.0008      protein    036_hr  label_free
515807   0.0012      protein    036_hr  label_free
515810   0.0024      protein    036_hr  label_free
515813   0.0038      protein    036_hr  label_free
515814   0.0040      protein    036_hr  label_free
515822   0.0075      protein    036_hr  label_free
515825   0.0087      protein    036_hr  label_free
515827   0.0108      protein    036_hr  label_free
515831   0.0120      protein    036_hr  label_free
515835   0.0131      protein    036_hr  label_free
```

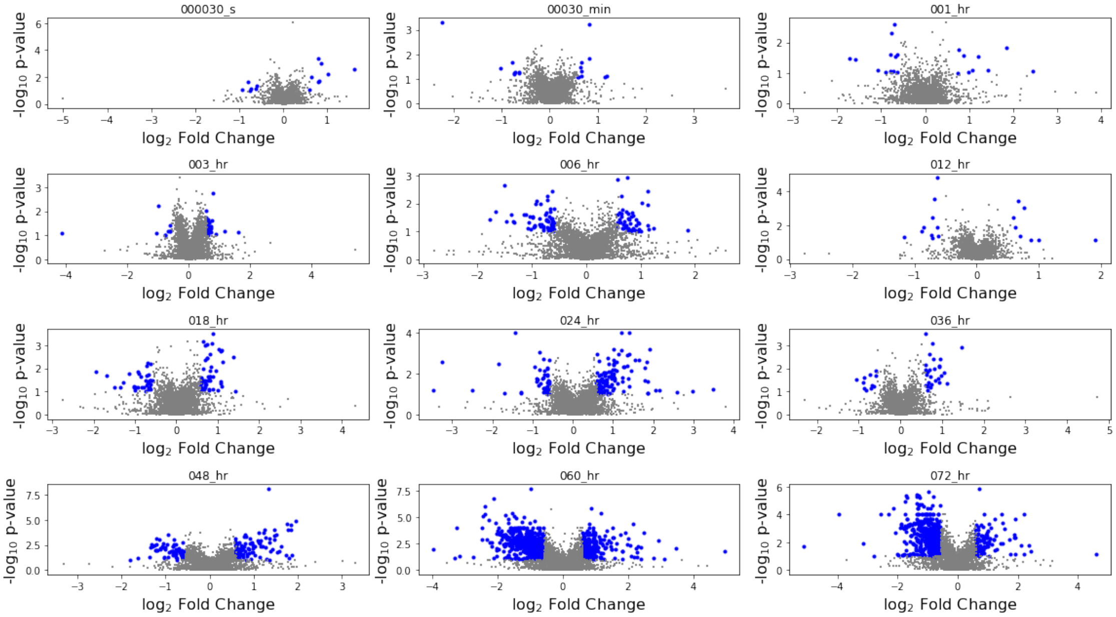
1.2.7 Visualization

Volcano plots

```
In [26]: exp_data.label_free.volcano_plot();
```

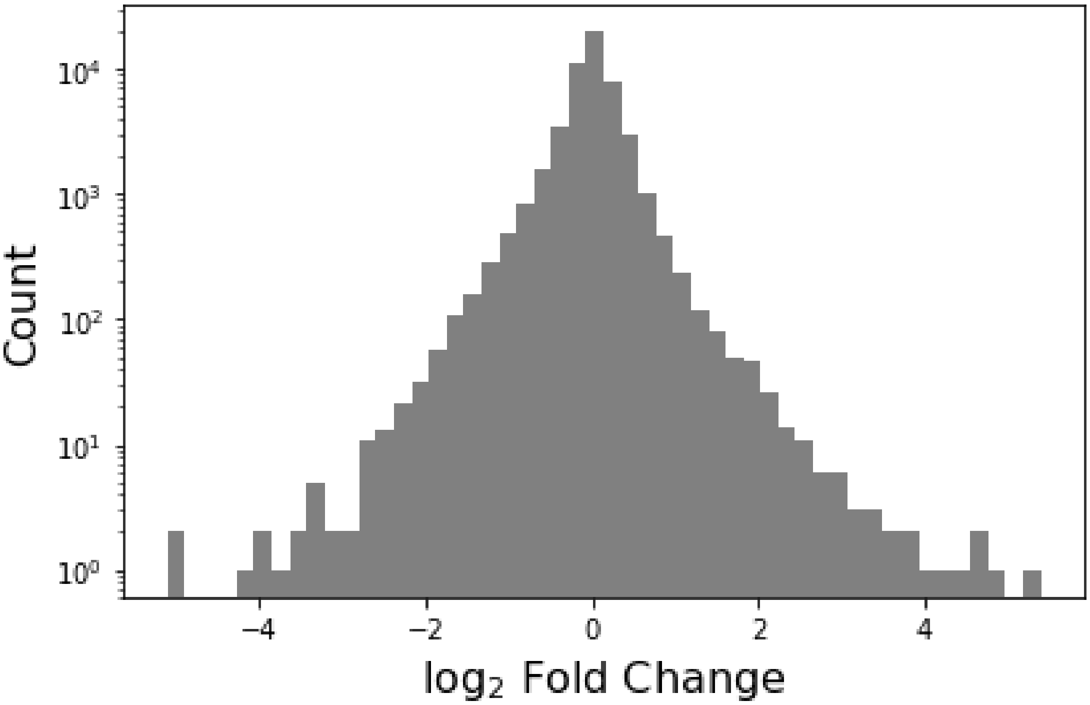


```
In [27]: exp_data.label_free.volcano_by_sample(sig_column=True);
```



Histogram

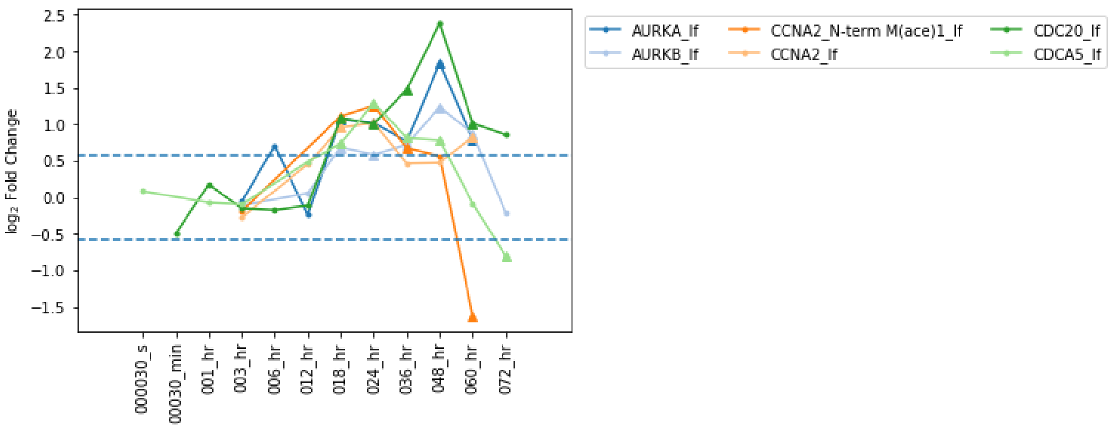
```
In [28]: exp_data.label_free.plot_histogram();
```



Plotting subset of species We provide the a few plotting interfaces to explore that subsets of the data. Basically, you create a list of species and provide it to the function. It filters based on these and then returns the results.

Time series using ploty and matplotlib

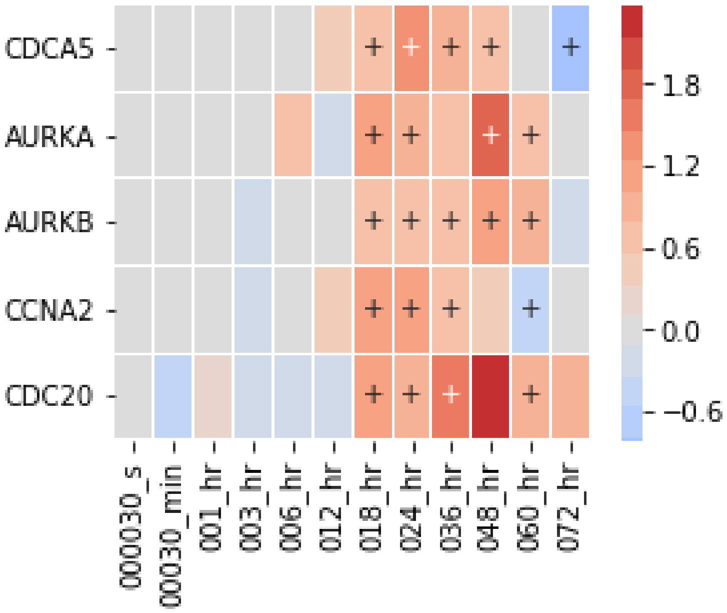
```
In [29]: # sample list for demo purposes
         interesting_list = ['CCNA2', 'CDCA5', 'CDC20', 'AURKA', 'AURKB']

         exp_data.label_free.plot_species(interesting_list, plot_type='matplotlib');
```



```
In [30]: exp_data.label_free.plot_species(interesting_list, plot_type='plotly')
```

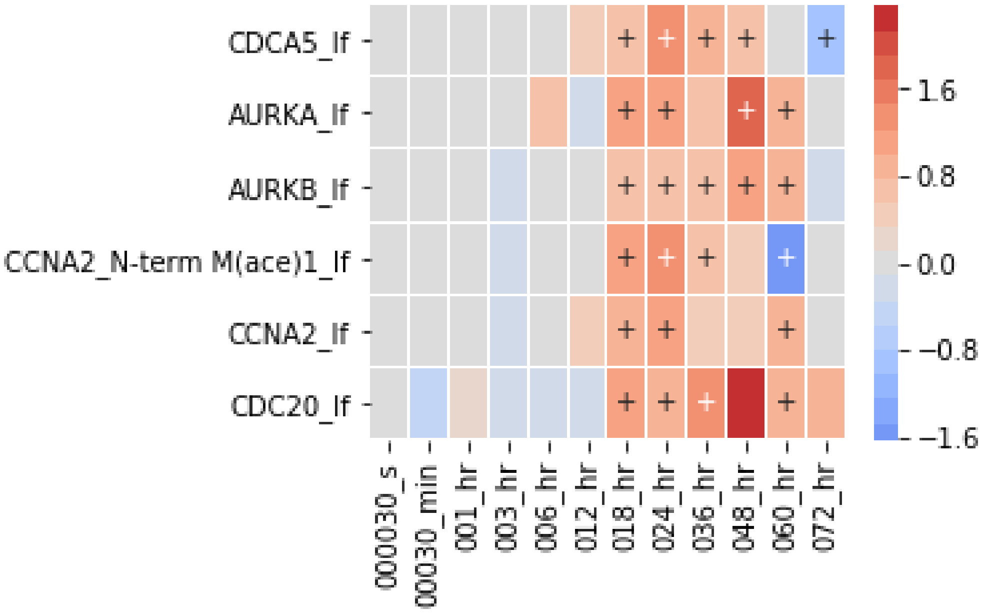
Heatplots

```
In [31]: exp_data.label_free.heatmap(interesting_list, linewidths=0.01, figsize=(4,3));
```



Notice that the above plot doesn't show any of the modifiers of `CCNA2_N-term M(ace)1_lf`. This is because the default index to pivot plots is the `identifier` column. You can set the `label` column for plotting by passing index=`label` to the function. Note, if you want to filter the data using the more generic `identifier` column, you just specify that with subset_index=`identifier`

```
In [32]: exp_data.label_free.heatmap(
             interesting_list,
             index='label',
             subset_index='identifier',
             linewidths=0.01,
             figsize=(4,3)
         );
```



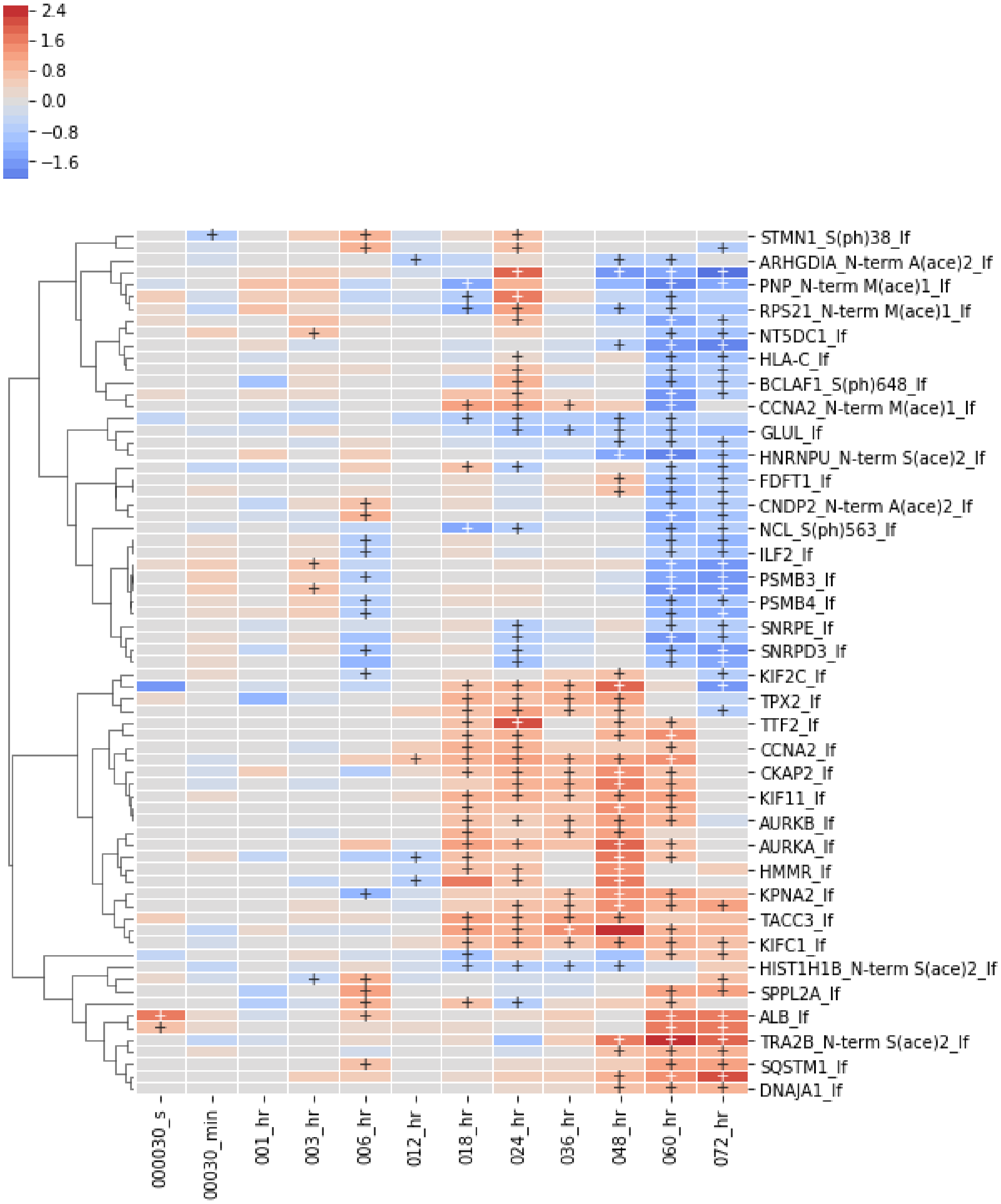
1.2.8 Examples

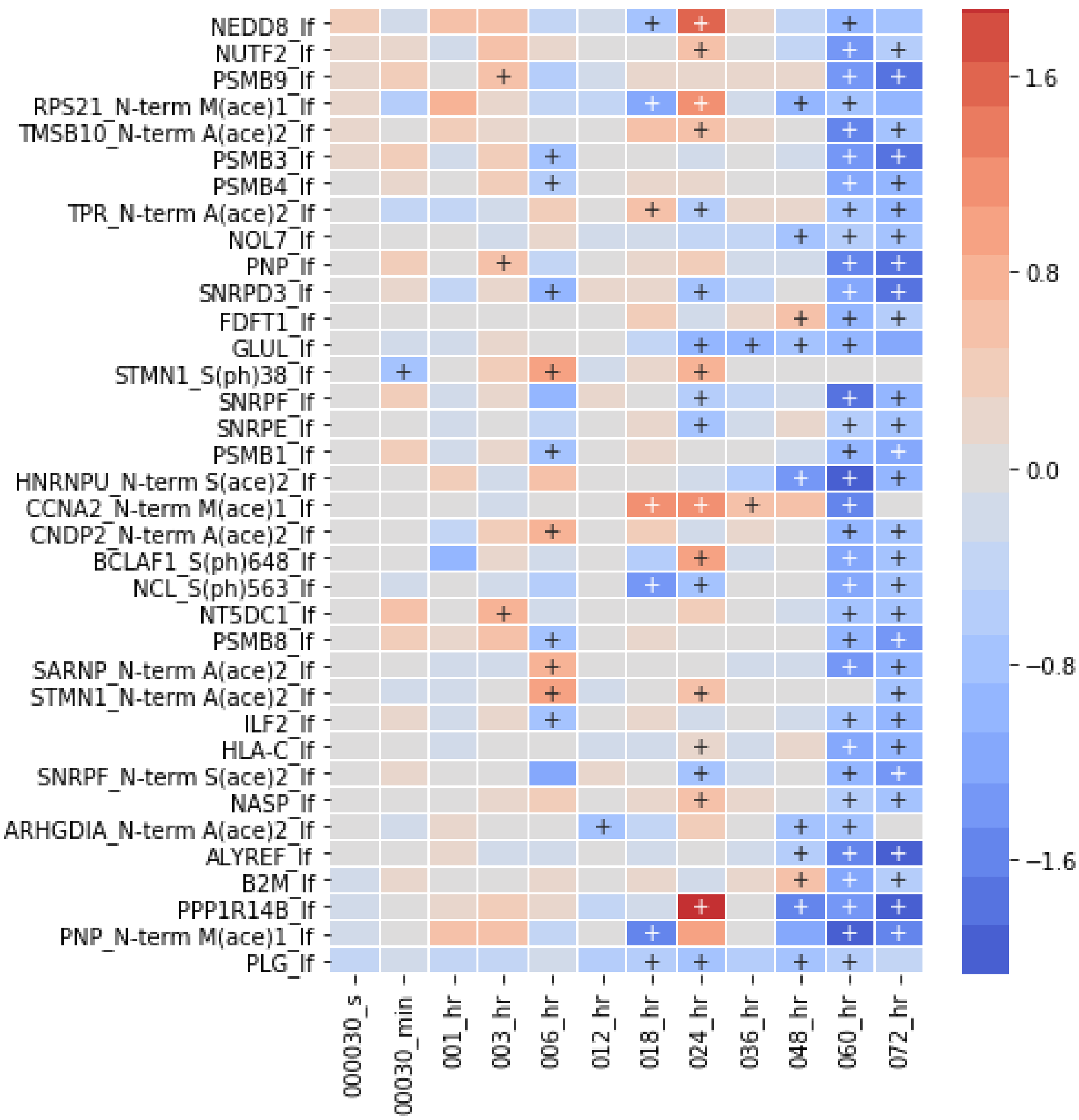
Here are a few examples how all the above commands can be chained together to create plots with varying degrees of critera.

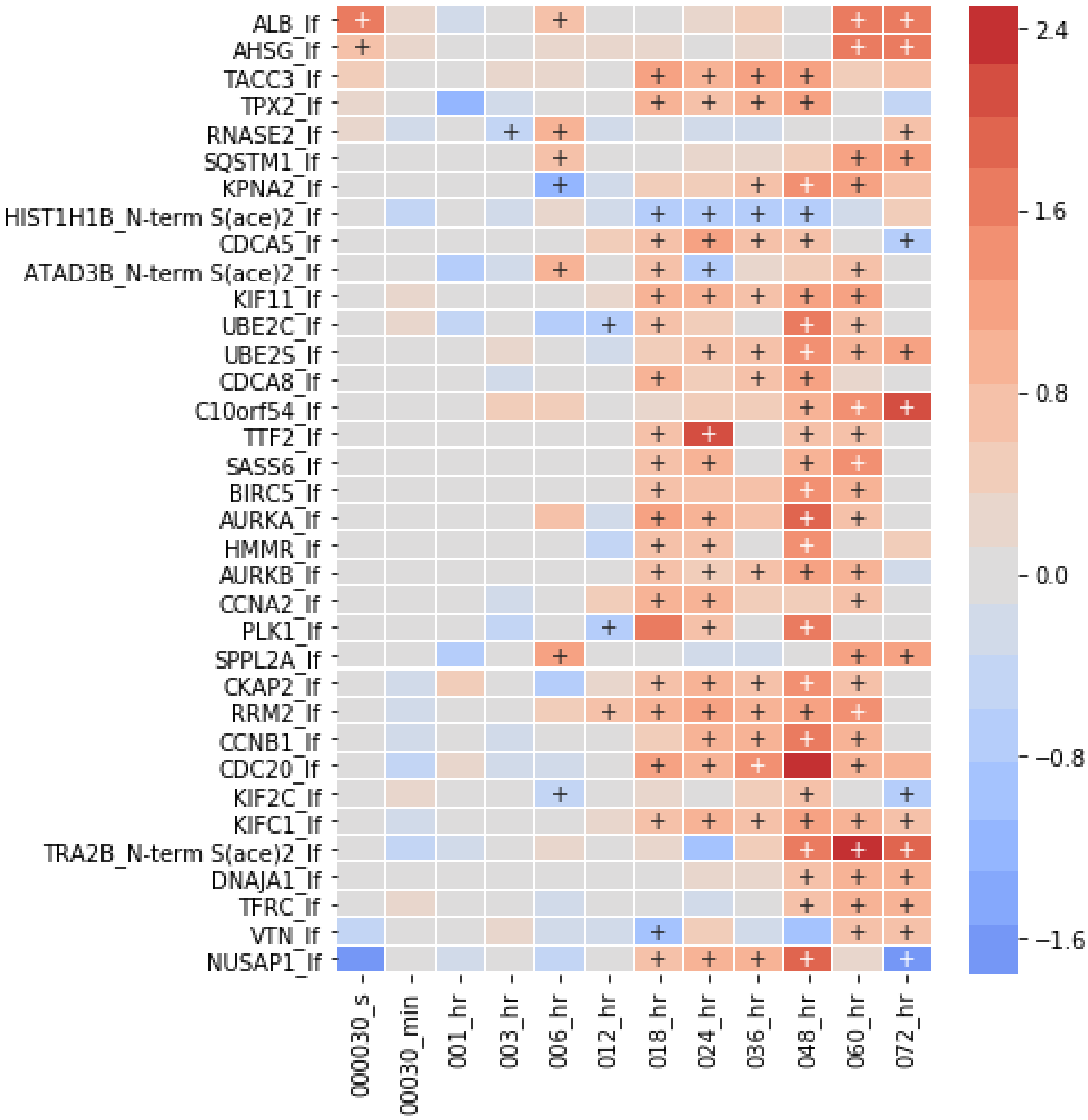
Query 1:

```
Heatmap of label-free proteomics that are signficantly change in at least 3 time points.
Extract clusters and visualize separately.
```

```
In [33]: lf_sig = exp_data.label_free.require_n_sig(
             index='label',
             columns='sample_id',
             n_sig=3
         )
         fig = lf_sig.heatmap(
             convert_to_log=True,
             cluster_row=True,
             index='label',
             values='fold_change',
             columns='sample_id',
             annotate_sig=True,
             figsize=(8, 12),
             div_colors=True,
             num_colors=21,
             linewidths=0.01
         );
         for i, j in fig.row_clusters.items():
             lf_sig.heatmap(j, index='label', linewidths=0.01, figsize=(6, 8))
             plt.show()
```


2.4
1.6
0.8
0.0
−0.8
−1.6
STMN1_S(ph)38_lf
ARHGDIA_N-term A(ace)2_lf
PNP_N-term M(ace)1_lf
RPS21_N-term M(ace)1_lf
NT5DC1_lf
HLA-C_lf
BCLAF1_S(ph)648_lf
CCNA2_N-term M(ace)1_lf
GLUL_lf
HNRNPU_N-term S(ace)2_lf
FDFT1_lf
CNDP2_N-term A(ace)2_lf
NCL_S(ph)563_lf
ILF2_lf
PSMB3_lf
PSMB4_lf
SNRPE_lf
SNRPD3_lf
KIF2C_lf
TPX2_lf
TTF2_lf
CCNA2_lf
CKAP2_lf
KIF11_lf
AURKB_lf
AURKA_lf
HMMR_lf
KPNA2_lf
TACC3_lf
KIFC1_lf
HIST1H1B_N-term S(ace)2_lf
SPPL2A_lf
ALB_lf
TRA2B_N-term S(ace)2_lf
SQSTM1_lf
DNAJA1_lf
000030_s
00030_min
001_hr
003_hr
006_hr
012_hr
018_hr
024_hr
036_hr
048_hr
060_hr
072_hr

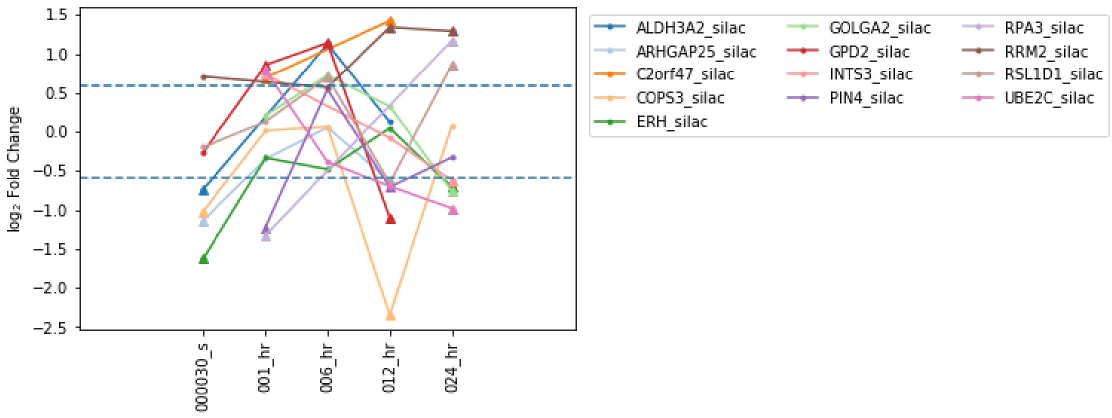

NEDD8_lf
NUTF2_lf
PSMB9_lf
RPS21_N-term M(ace)1_lf
TMSB10_N-term A(ace)2_lf
PSMB3_lf
PSMB4_lf
TPR_N-term A(ace)2_lf
NOL7_lf
PNP_lf
SNRPD3_lf
FDFT1_lf
GLUL_lf
STMN1_S(ph)38_lf
SNRPF_lf
SNRPE_lf
PSMB1_lf
HNRNPU_N-term S(ace)2_lf
CCNA2_N-term M(ace)1_lf
CNDP2_N-term A(ace)2_lf
BCLAF1_S(ph)648_lf
NCL_S(ph)563_lf
NT5DC1_lf
PSMB8_lf
SARNP_N-term A(ace)2_lf
STMN1_N-term A(ace)2_lf
ILF2_lf
HLA-C_lf
SNRPF_N-term S(ace)2_lf
NASP_lf
ARHGDIA_N-term A(ace)2_lf
ALYREF_lf
B2M_lf
PPP1R14B_lf
PNP_N-term M(ace)1_lf
PLG_lf
000030_s
00030_min
001_hr
003_hr
006_hr
012_hr
018_hr
024_hr
036_hr
048_hr
060_hr
072_hr
1.6
0.8
0.0
−0.8
−1.6



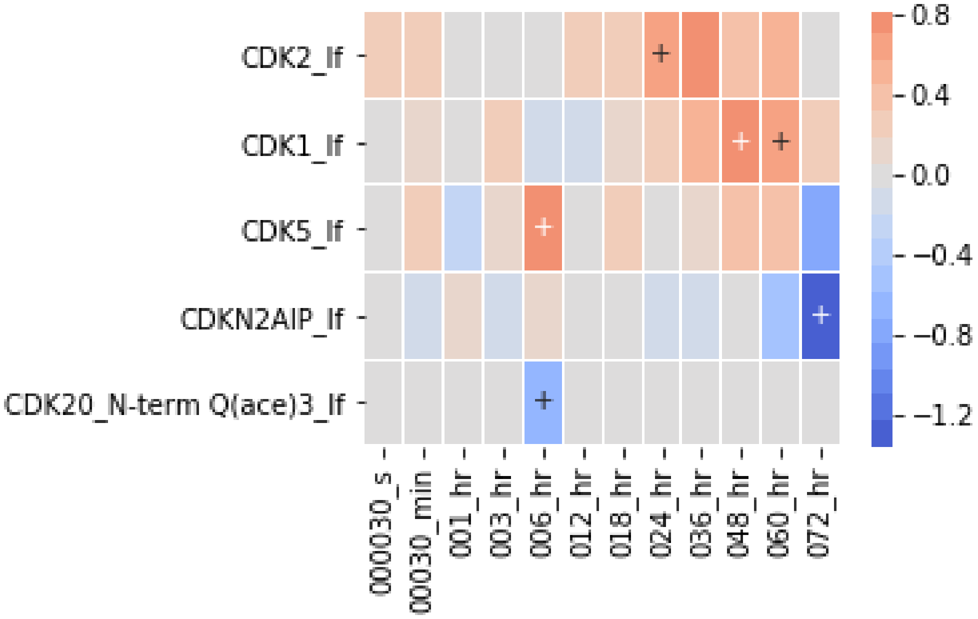
Query 2:

Changes that happen at all 2 timepoints for silac.

```
In [34]: exp_data.silac.require_n_sig(
             n_sig=2, index='label'
         ).plot_species(plot_type='matplotlib');
```



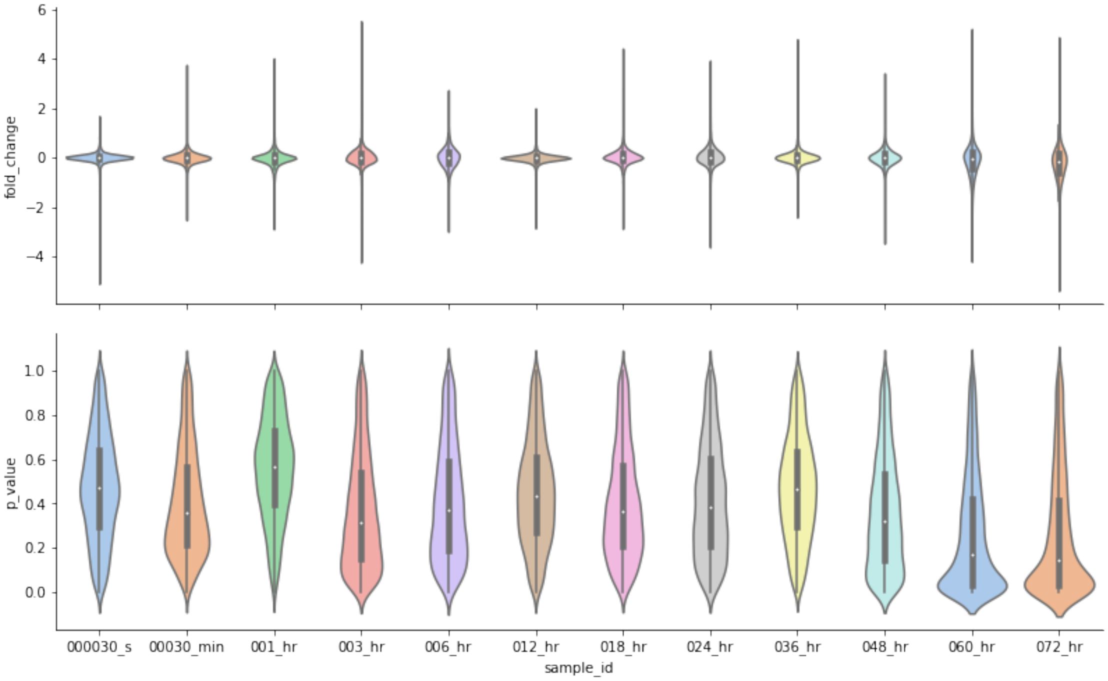
```
In [35]: int_species = 'CDK'
         exp_data.label_free.heatmap(
             int_species,
             subset_index='identifier',
             index='label',
             min_sig=1,
             linewidths=0.01,
             figsize=(4, 3)
         );
```



1.2.9 Extending to other plots

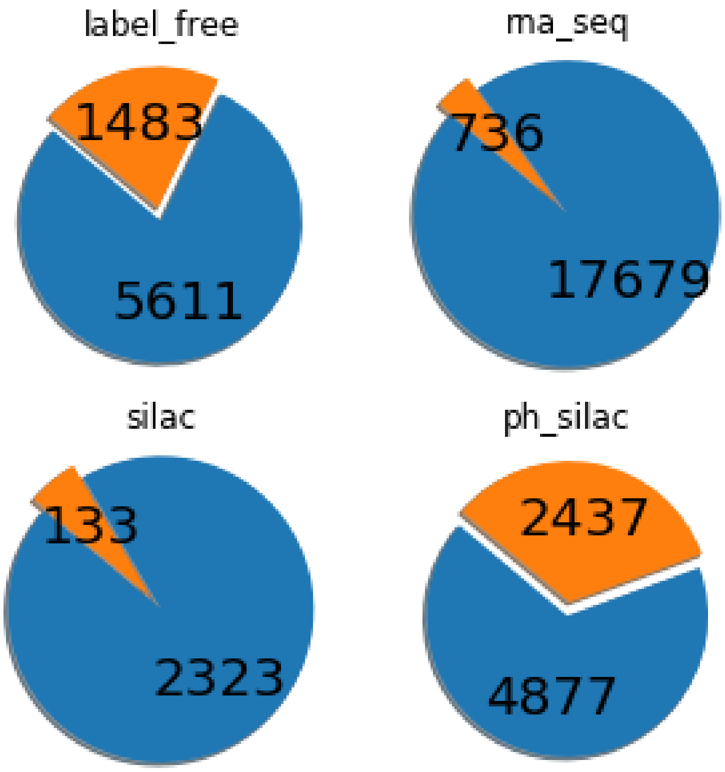
Since our exp_data is built off a pandas.DataFrame, we can use other packages that take that data format. Seaborn is one such tool that provides some very nice plots.

```
In [36]: label_free = exp_data.label_free.copy()
         label_free.log2_normalize_df(column='fold_change', inplace=True)

         g = sns.PairGrid(label_free,
                          x_vars=('sample_id'),
                          y_vars=('fold_change', 'p_value'),
                          hue='source',
                          aspect=3.25, height=3.5)
         g.map(
             sns.violinplot,
             palette="pastel",
             split=True,
             order=label_free.sample_ids
         );
```



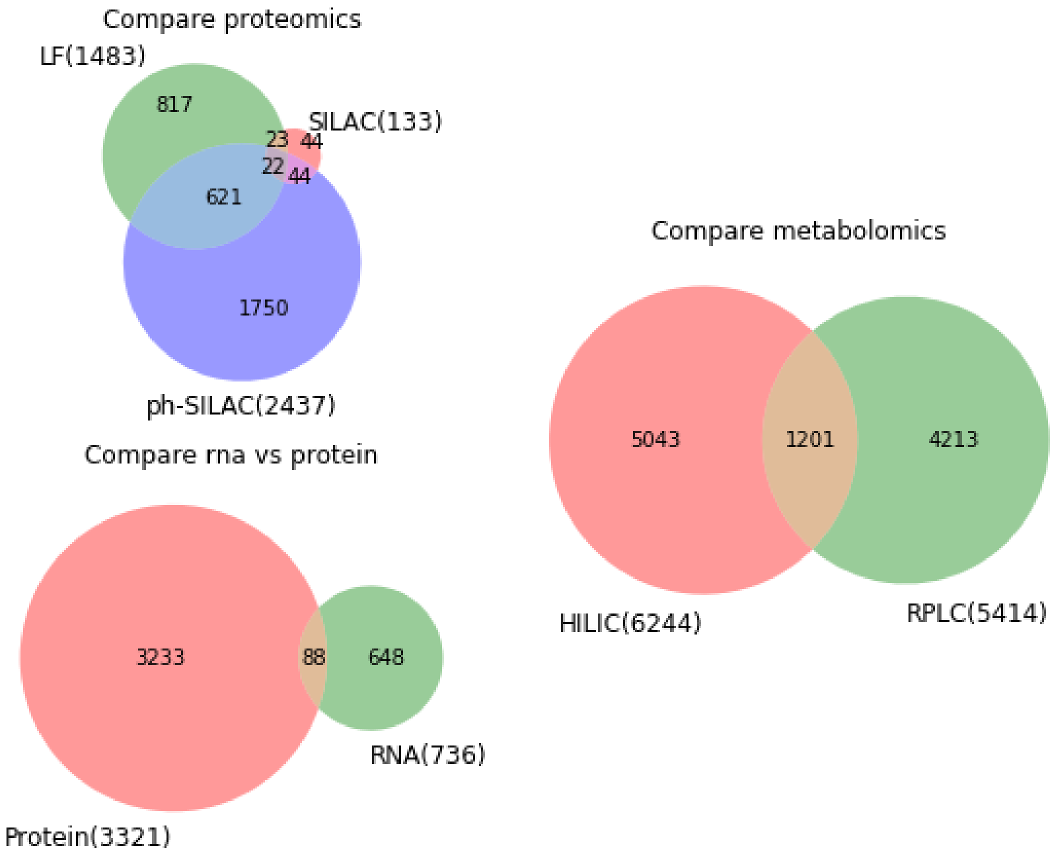
Visualizing signficant fraction of measured species

```
In [37]: fig = plt.figure(figsize=(5,5))
         exp_data.label_free.plot_pie_sig_ratio('pie_label_free', fig.add_subplot(221))
         plt.title("label_free");
         exp_data.rna.plot_pie_sig_ratio('pie_rna_seq', fig.add_subplot(222))
         plt.title("rna_seq");
         exp_data.silac.plot_pie_sig_ratio('pie_silac', fig.add_subplot(223))
         plt.title("silac");
         exp_data.ph_silac.plot_pie_sig_ratio('pie_ph_silac', fig.add_subplot(224))
         plt.title("ph_silac");
         plt.savefig("pie_sig_omics.png", dpi=300, bbox_inches='tight')
```



Venn diagram comparisons between measurements

```
In [38]: from magine.plotting.venn_diagram_maker import create_venn2, create_venn3

         lf = exp_data.label_free.sig.id_list
         silac = exp_data.silac.sig.id_list
         phsilac = exp_data.ph_silac.sig.id_list
         rna_names = exp_data.rna_seq.sig.id_list
         hilic = exp_data.HILIC.sig.id_list
         rplc = exp_data.C18.sig.id_list
         fig = plt.figure(figsize=(8,6))

         create_venn3(lf, silac, phsilac, 'LF', 'SILAC', 'ph-SILAC', ax=fig.add_subplot(221));
         plt.title("Compare proteomics");
         create_venn2(set.union(*[lf, silac, phsilac]), rna_names,
                      'Protein', 'RNA', ax=fig.add_subplot(223));
         plt.title("Compare rna vs protein");
         create_venn2(hilic, rplc, 'HILIC', 'RPLC', ax=fig.add_subplot(122));
         plt.title("Compare metabolomics");
         plt.tight_layout()
         plt.savefig("venn_diagrams.png", dpi=300, bbox_inches='tight')
```

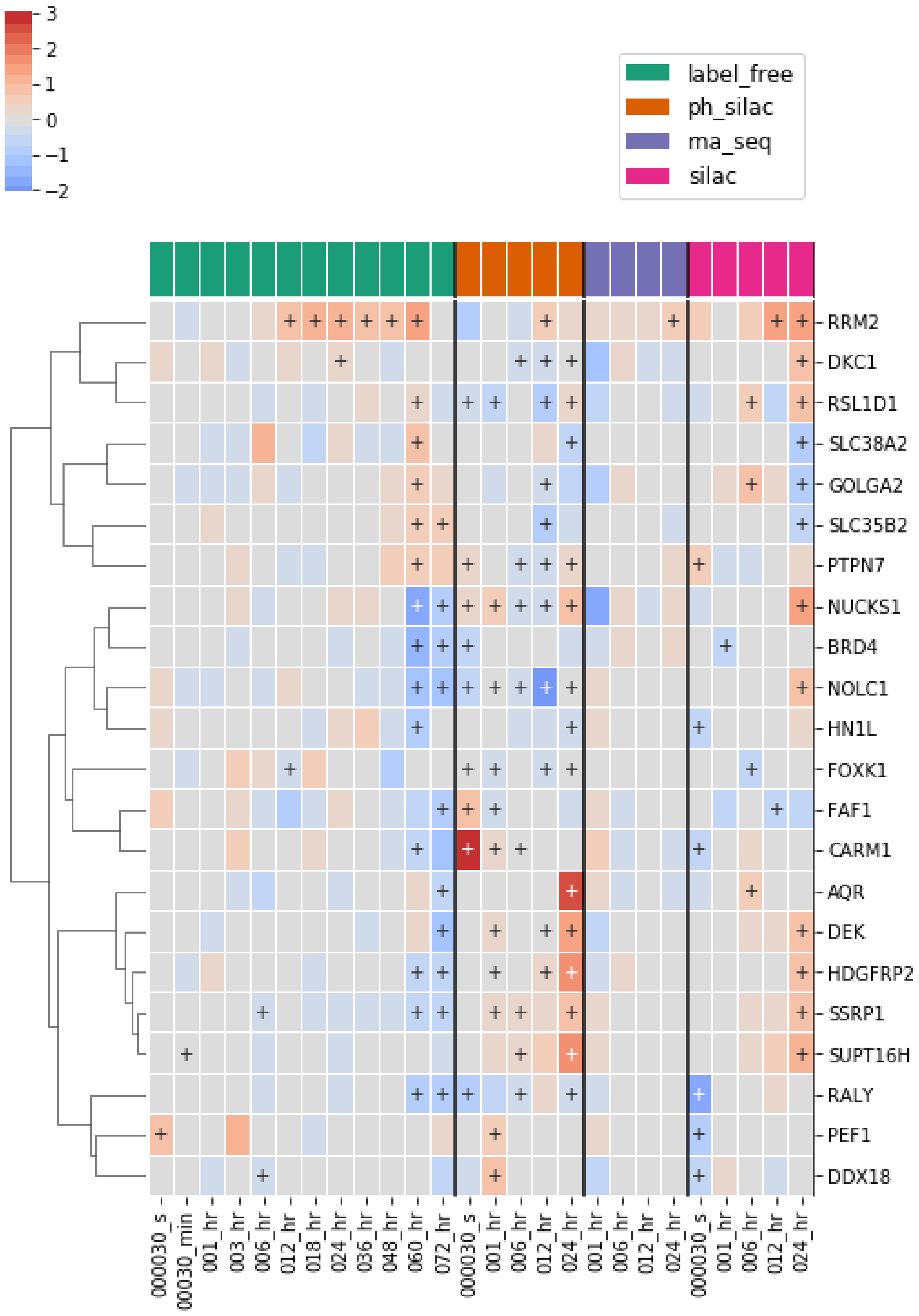


Query 3:

Extract out species that are significantly changed in all three omics (label-free, silac, ph-s

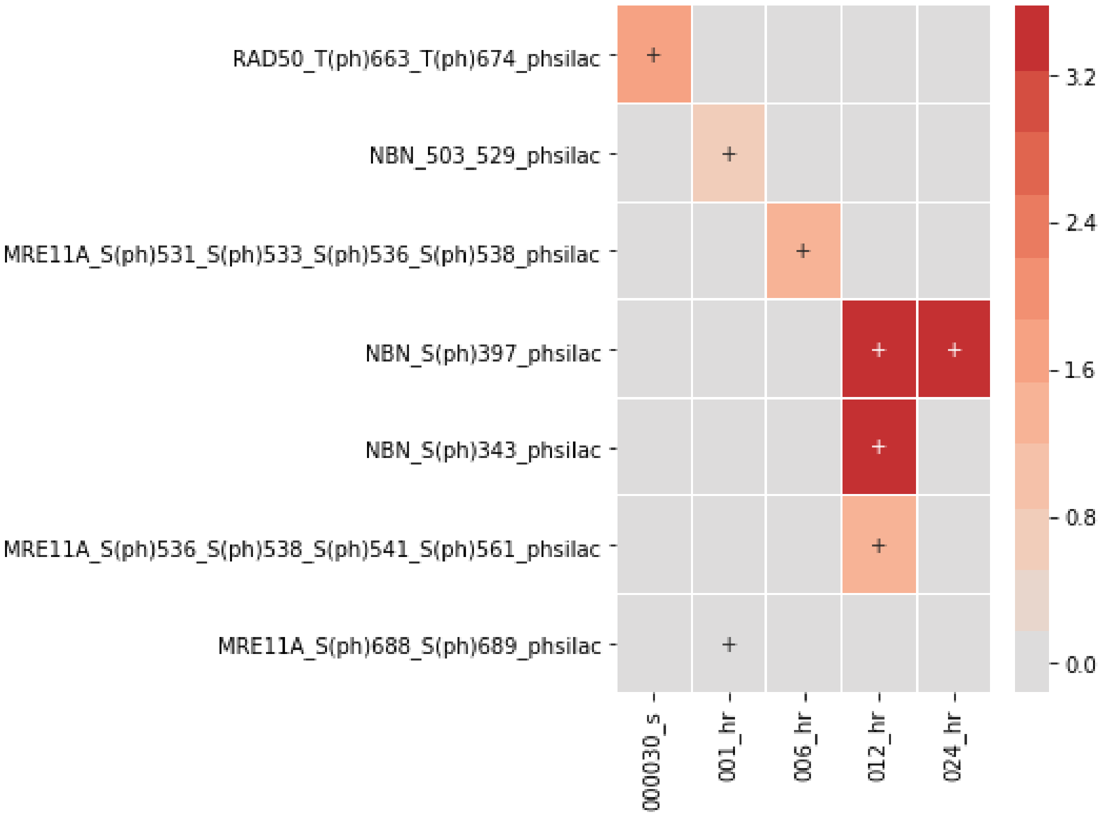
```
In [39]: all_three_omics = lf.intersection(silac).intersection(phsilac)
```

```
In [40]: exp_data.species.subset(all_three_omics).heatmap(
             all_three_omics,
             subset_index='identifier',
             columns=['source', 'sample_id',],
             min_sig=0,
             figsize=(8, 12), cluster_row=True,
             y_tick_labels=True, linewidths=0.01
         );
```

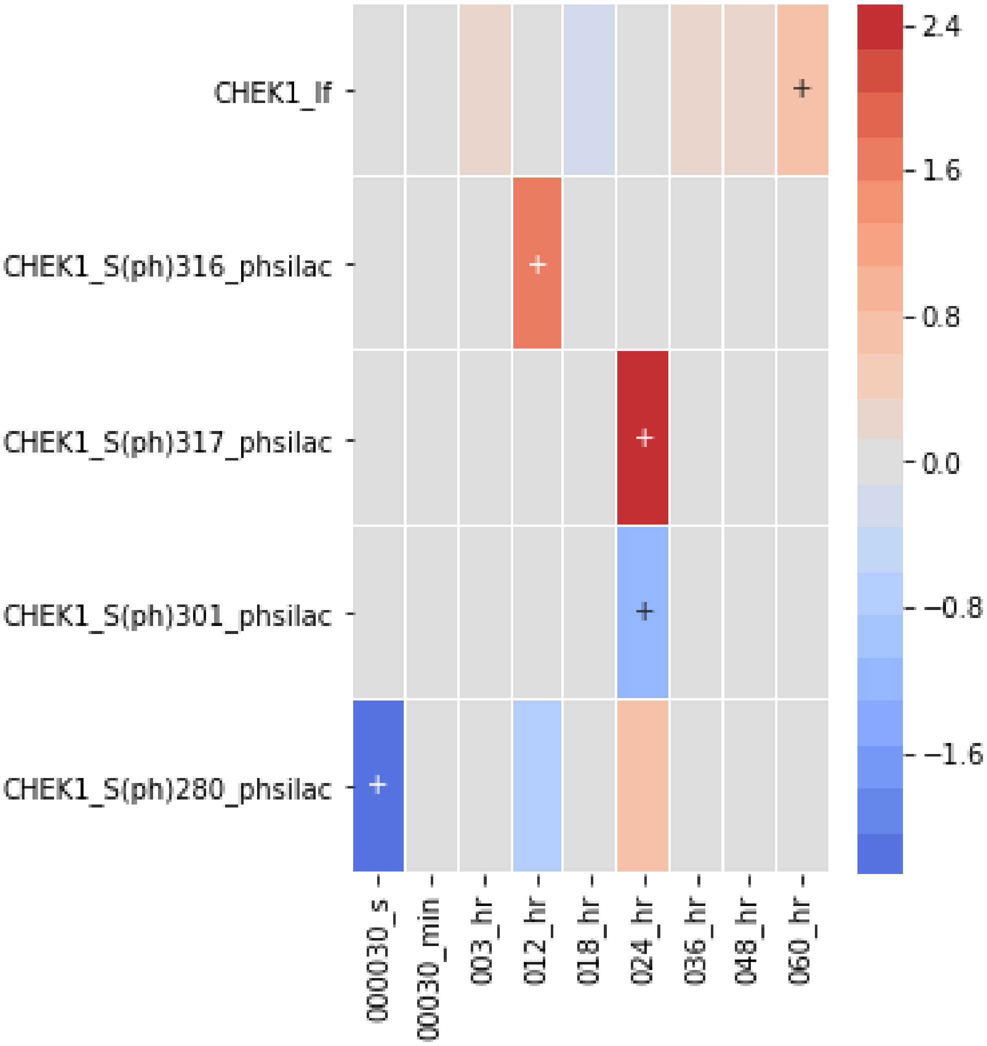

3
2
1
0
−1
−2
label_free
ph_silac
ma_seq
silac
RRM2
DKC1
RSL1D1
SLC38A2
GOLGA2
SLC35B2
PTPN7
NUCKS1
BRD4
NOLC1
HN1L
FOXK1
FAF1
CARM1
AQR
DEK
HDGFRP2
SSRP1
SUPT16H
RALY
PEF1
DDX18
000030_s
00030_min
001_hr
003_hr
006_hr
012_hr
018_hr
024_hr
036_hr
048_hr
060_hr
072_hr
000030_s
001_hr
006_hr
012_hr
024_hr
001_hr
006_hr
012_hr
024_hr
000030_s
001_hr
006_hr
012_hr
024_hr

1.2.10 Plot lists of interest

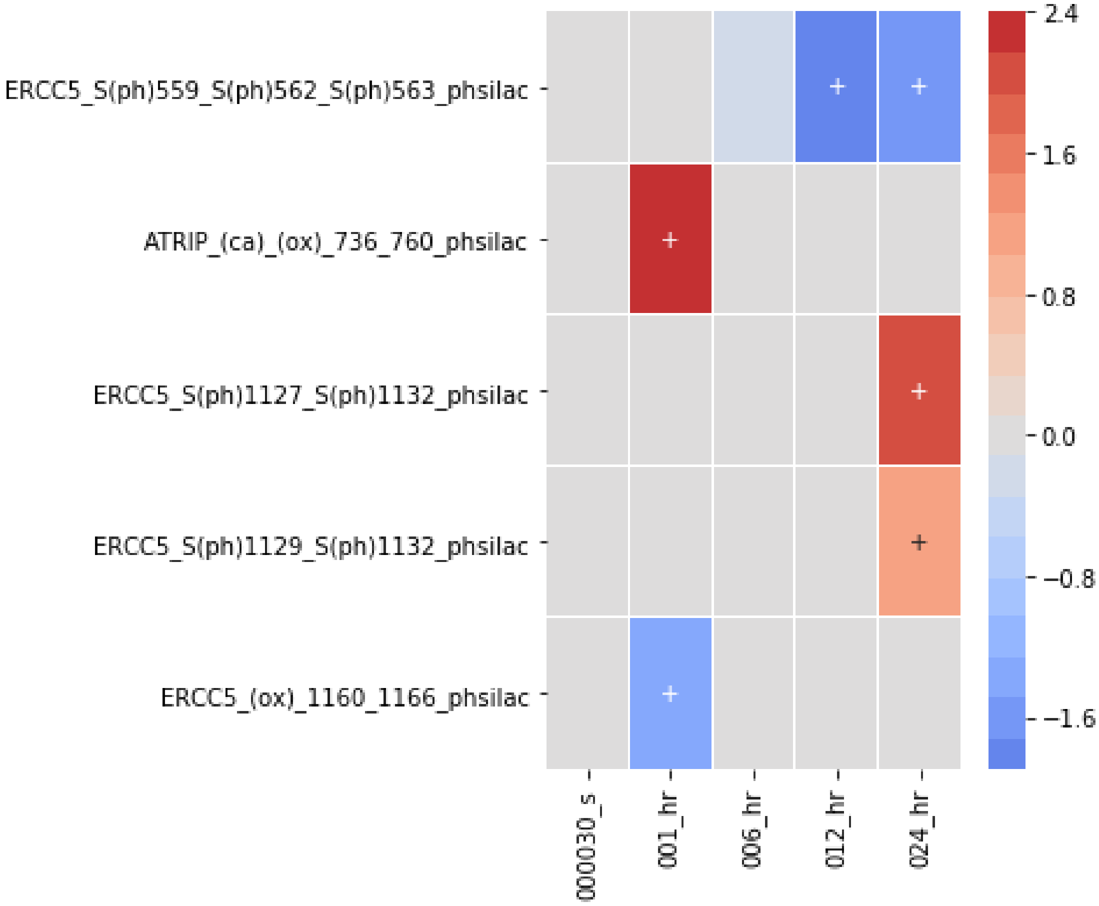
```
In [41]: # sample list for demo purposes
         mrn_complex = ['NBN', 'RAD50', 'MRE11A']

         exp_data.species.heatmap(
             mrn_complex,
             index='label',
             subset_index='identifier',
             min_sig=1,
             linewidths=0.01,
             figsize=(4,6)
         );
```



```
In [42]: # sample list for demo purposes
         interesting_list = ['CHEK1', 'CHEK2']

         exp_data.species.heatmap(
             interesting_list,
             index='label',
             subset_index='identifier',
             min_sig=1,
             linewidths=0.01,
             figsize=(4,6)
         );
```



```
In [43]: # sample list for demo purposes
         interesting_list = ['ATRIP', 'ERCC5']

         exp_data.species.heatmap(
             interesting_list,
             index='label',
             subset_index='identifier',
             min_sig=1,
             linewidths=0.01,
             figsize=(4,6)
         );
```


ERCC5_S(ph)559_S(ph)562_S(ph)563_phsilac
ATRIP_(ca)_(ox)_736_760_phsilac
ERCC5_S(ph)1127_S(ph)1132_phsilac
ERCC5_S(ph)1129_S(ph)1132_phsilac
ERCC5_(ox)_1160_1166_phsilac
000030_s
001_hr
006_hr
012_hr
024_hr
+
2.4
1.6
0.8
0.0
−0.8
−1.6

Data S2. Network construction and exploration demonstration. Related to STAR METHODS and Figure 2.

1 Creating a data driven network

This example shows how we create and add annotations to a data driven network.

[2]:
```
%matplotlib inline
import sys
import os
import networkx as nx
from IPython.display import display, Image
```

[3]:
```
# NBVAL_IGNORE_OUTPUT
from exp_data import exp_data
import magine.networks.utils as utils
from magine.networks.network_generator import build_network
from magine.networks.subgraphs import Subgraph
import magine.networks.exporters as exporters
import magine.networks.visualization as viz
```

```
2019-07-01 11:14:25.012 - magine - INFO - Logging started on MAGINE
2019-07-01 11:14:25.015 - magine - INFO - Log entry time offset from UTC: -7.00
hours
```

Creating list of seed species and background species for network

[4]:
```
measured = exp_data.species.id_list
sig_measured = exp_data.species.sig.id_list
print(len(measured))
print(len(sig_measured))
```

```
54750
14136
```

Now we will create the network. We pass the seed and background list to the network as well as flags turning on all of the network databases. We also trim source/sink nodes (optional). This basically cleans up dangling nodes that are not in our seed or background lists.

[5]:
```
save_name = 'bendamustine_network_w_attributes'

execute = False
```

```
if execute:
    network = build_network(
        seed_species=sig_measured,  # seed species
        all_measured_list=measured,  # all data measured
        use_biogrid=True,  # expand with biogrid
        use_hmdb=True,  # expand with hmdb
        use_reactome=True,  # expand with reactome
        use_signor=True,  # expand with signor
        trim_source_sink=True,  # remove all source and sink nodes not measured
        save_name='Networks/bendamustine_network'
    )
    # add data to networks
    network = utils.add_data_to_graph(network, exp_data)

    # write to GML for cytoscape or other program
    nx.write_gml(network, os.path.join('Networks', save_name+'.gml'))

    # write to gpickle for fast loading in python
    nx.write_gpickle(network, os.path.join('Networks', save_name+'.p'))
else:
    network = nx.read_gpickle(os.path.join('Networks', save_name+'.p'))
```

[7]:

```
print(len(network.nodes))
print(len(network.edges))
```

```
21130
515061
```

21130 nodes and 515061 edges are too much to manually explore. Thus, we are going to use the Subgraph Class to being to query the network.

[8]:

```
# initialize it
net_sub = Subgraph(network)
```

[10]:

```
help(net_sub)
```

```
Help on Subgraph in module magine.networks.subgraphs object:

class Subgraph(builtins.object)
 |  Subgraph(network, exp_data=None, pool=None)
 |
 |  Methods defined here:
 |
 |  __init__(self, network, exp_data=None, pool=None)
 |      Generates network subgraphs
 |
 |      Parameters
 |      ----------
 |      network : networkx.DiGraph
 |      exp_data : magine.data.datatypes.ExperimentalData
 |
 |  downstream_of_node(self, species_1, include_list=None, save_name=None,
draw=False)
 |      Generate network of all downstream species of provides species
 |
 |
 |      Parameters
 |      ----------
 |      species_1 : str
 |          species name
 |      save_name : str
 |          name to save gml file
 |      draw : bool
 |          create figure of graph
 |      include_list : list_like
 |          list of species that must be in path in order to consider a path
 |      Returns
 |      -------
 |      nx.DiGraph
 |
 |
 |      Examples
 |      --------
 |      >>> from networkx import DiGraph
 |      >>> from magine.networks.subgraphs import Subgraph
 |      >>> g = DiGraph()
 |      >>> g.add_edges_from([('a','b'),('b','c'), ('c', 'd'), ('a', 'd'),
('e', 'd')])
 |      >>> net_sub = Subgraph(g)
 |      >>> downstream_d = net_sub.downstream_of_node('d')
 |      >>> sorted(downstream_d.edges)
 |      []
 |      >>> downstream_c = net_sub.downstream_of_node('c')
 |      >>> sorted(downstream_c.edges)
 |      [('c', 'd')]
 |
 |  expand_neighbors(self, network=None, nodes=None, upstream=False,
downstream=False, max_dist=1, include_only=None,
add_interconnecting_edges=False)
 |      Create/expand a network based on neighbors from a list of species
 |
 |      Parameters
 |      ----------
 |      network : nx.DiGraph or None
 |          Starting network to expand nodes. If not provided, will use
 |          default network
 |      nodes : list_like
 |          List of nodes to expand
 |      upstream : bool
 |          Expand upstream nodes
 |      downstream : bool
 |          Expand downstream nodes
 |      max_dist :
 |          Max distance to explore
 |      include_only : list_like
 |          Limit network to only contain these species
 |      add_interconnecting_edges : bool
 |          Add edges connecting all nodes. Default if False, so only direct
 |          edges to neighbors will be added.
 |
 |      Returns
 |      -------
 |      nx.DiGraph
 |
 |  measured_networks_over_time(self, graph, colors, prefix)
 |      Adds color to a network over time
 |
 |      Parameters
 |      ----------
 |      graph : nx.DiGraph
 |      colors : list
 |          List of colors for time points
 |      prefix : str
 |          Prefix for image files
 |
 |
 |      Returns
 |      -------
 |
 |  measured_networks_over_time_up_down(self, graph, prefix, color_up='tomato',
color_down='lightblue')
 |      Parameters
 |      ----------
 |      graph : nx.DiGraph
 |
 |      prefix : str
 |          Prefix for image files
 |      color_up : str
 |
 |      color_down : str
 |
 |
 |      Returns
 |      -------
 |
 |  neighbors(self, node, upstream=True, downstream=True, max_dist=1,
include_only=None, start_network=None)
 |      Create network containing provided node and its neighbors.
 |
 |      Parameters
 |      ----------
 |      node : str
 |      upstream : bool
 |      downstream : bool
 |      max_dist : int
 |      include_only : list
 |      start_network : nx.DiGraph
 |
 |
 |      Returns
 |      -------
 |      nx.DiGraph
 |
 |  paths_between_list(self, species_list, single_path=False, max_length=None,
add_interconnecting_edges=False, include_only=None, pool=None, save_name=None,
draw=False, image_format='png')
 |      Returns graph containing all shortest paths between list.
 |
 |      Parameters
 |      ----------
 |
 |      species_list : list_like
 |          list of species
 |      save_name : str
 |          name to save
 |      single_path : bool
 |          use single shortest path if True, else use all shortest paths
 |      draw : bool
 |          create a dot generated figure
 |      image_format : str
 |          dot acceptable output formats, (pdf, png, etc)
 |      pool : multiprocessing.Pool
 |          If it it provided, it uses its map function to run this function.
 |      max_length : int
 |          Max length for path between any 2 species
 |      include_only : list_like
 |          List of species that must be present
 |      Returns
 |      -------
 |      graph : networkx.DiGraph
 |          graph containing paths between species list provided
 |
 |
 |      Examples
 |      --------
 |      >>> from networkx import DiGraph
 |      >>> from magine.networks.subgraphs import Subgraph
 |      >>> g = DiGraph()
 |      >>> g.add_edges_from([('a','b'),('b','c'), ('c', 'd'), ('a', 'd'), ('e',
'd')])
 |      >>> g.add_path(['g', 'h', 'c', 'i', 'j', 'k'])
 |      >>> net_sub = Subgraph(g)
 |      >>> path_a_d = net_sub.paths_between_list(['a','c','d'])
 |      >>> sorted(path_a_d.edges)
 |      [('a', 'b'), ('a', 'd'), ('b', 'c'), ('c', 'd')]
 |      >>> path_a_f = net_sub.paths_between_list(['g', 'h', 'j'], max_length=4)
 |      >>> sorted(path_a_f.edges)
 |      [('c', 'i'), ('g', 'h'), ('h', 'c'), ('i', 'j')]
 |
 |  paths_between_pair(self, node_1, node_2, bidirectional=False,
single_path=False, draw=False, image_format='png')
 |      Generates a graph based on all shortest paths between two species.
 |
 |
 |      Parameters
 |      ----------
 |      node_1 : str
 |          name of first species
 |      node_2 : str
 |          name of second species
 |      bidirectional : bool
 |          If you want to search bidirectionally
 |      single_path : bool
 |          If you only want a single shortest path
 |      draw : bool
 |          create an image of returned network
 |      image_format : str, optional
 |          If draw=True you can pass an image format. (pdf, png, svg).
 |          default=png
 |
 |      Returns
 |      -------
 |      graph : networkx.DiGraph
 |
 |
 |      Examples
 |      --------
 |      >>> from networkx import DiGraph
 |      >>> from magine.networks.subgraphs import Subgraph
 |      >>> g = DiGraph()
 |      >>> g.add_edges_from([('a','b'),('b','c'), ('c', 'd'),  ('e', 'd'),
('d', 'a')])
 |      >>> net_sub = Subgraph(g)
 |      >>> path_a_d = net_sub.paths_between_pair('a','d', False)
 |      >>> sorted(path_a_d.edges)
 |      [('a', 'b'), ('b', 'c'), ('c', 'd')]
 |      >>> path_a_d = net_sub.paths_between_pair('a','d', True)
 |      >>> sorted(path_a_d.edges)
 |      [('a', 'b'), ('b', 'c'), ('c', 'd'), ('d', 'a')]
 |
 |  paths_between_two_lists(self, list_1, list_2, single_path=False,
max_length=None, include_only=None, reverse=False,
add_interconnecting_edges=False, draw=False, save_name=None, image_format='png')
 |      Generates a graph based on all shortest paths between two species.
 |
 |
 |      Parameters
 |      ----------
 |      list_1 : list
 |          Node names
 |      list_2 : list
 |          Node names
 |      single_path : bool
 |          If you only want a single shortest path.
 |      max_length : int
 |          Maximum distance between any two species.
 |      include_only : list
 |          Species required to be in paths/
 |      reverse : bool
 |          Flag to check list_2 to list_1. Default will only look for list_1
 |          to list_2.
 |      add_interconnecting_edges : bool
 |          Add edges between species even if not between list_1 and list_2
 |          nodes.
 |      save_name : str
 |          Save of figure/network
 |      draw : bool
 |          create an image of returned network
 |      image_format : str, optional
 |          If draw=True you can pass an image format. (pdf, png, svg).
 |          default=png
 |
 |      Returns
 |      -------
 |      graph : networkx.DiGraph
 |
 |      Examples
 |      --------
 |      >>> from networkx import DiGraph
 |      >>> from magine.networks.subgraphs import Subgraph
 |      >>> g = DiGraph()
 |      >>> g.add_path(['a', 'b', 'c', 'd'])
 |      >>> g.add_path(['g', 'h', 'c', 'i'])
 |      >>> net_sub = Subgraph(g)
 |      >>> path_a_d = net_sub.paths_between_two_lists(['a','g'], ['c', 'i'],
max_length=3)
 |      >>> sorted(path_a_d.edges)
 |      [('a', 'b'), ('b', 'c'), ('g', 'h'), ('h', 'c')]
 |
 |  upstream_of_node(self, species_1, include_list=None, save_name=None,
draw=False)
 |      Generate network of all upstream species of provides species
 |
 |      Parameters
 |      ----------
 |      species_1 : str
 |          species name
 |      save_name : str
 |          name to save gml file
 |      draw : bool
 |          create figure of graph
 |      include_list : list_like
 |          Species that must be in path in order to consider a path
 |
 |      Returns
 |      -------
 |      nx.DiGraph
 |
 |
 |      Examples
 |      --------
 |      >>> from networkx import DiGraph
 |      >>> from magine.networks.subgraphs import Subgraph
 |      >>> g = DiGraph()
 |      >>> g.add_edges_from([('a','b'),('b','c'), ('c', 'd'), ('a', 'd'),
('e', 'd')])
 |      >>> net_sub = Subgraph(g)
 |      >>> upstream_d = net_sub.upstream_of_node('d')
 |      >>> sorted(upstream_d.edges())
 |      [('a', 'd'), ('b', 'c'), ('c', 'd'), ('e', 'd')]
 |      >>> upstream_c = net_sub.upstream_of_node('c')
 |      >>> sorted(upstream_c.edges())
 |      [('a', 'b'), ('b', 'c')]
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
```

```
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

1.1 Exploring neighbors of nodes of interest

For demonstration purposes, we are starting with the protein CASP3. CASP3 is an effector caspase that is required for apoptosis. It cleaves other proteins and starts the degradation CASP3. We start our exploration at CASP3 as it is a marker for intrinsic apoptosis, which is what is generally regarded as bendamustines pathway for cell death.

First, we are going to look at all the neighbors of CASP3 that were signficantly changed in our experimental data.

[23]:

```
casp3_neighbors = net_sub.neighbors(
    'CASP3', # node of interest
    upstream=True, # include upstream nodes
    downstream=False,  # include downstream nodes
    include_only=exp_data.species.sig.id_list # limit nodes to only significant␣
 ↪changed species
)
```

[24]:

```
# one of many ways to draw the network.
# draw_cyjs is ideal for Jupyter notebooks since it allows us to move nodes,␣
 ↪apply various layouts
# Notice that if you click on an edge, it provides you with the interaction␣
 ↪type.
# If you click on a node, it provides a link to genecards.
viz.draw_cyjs(casp3_neighbors)
```

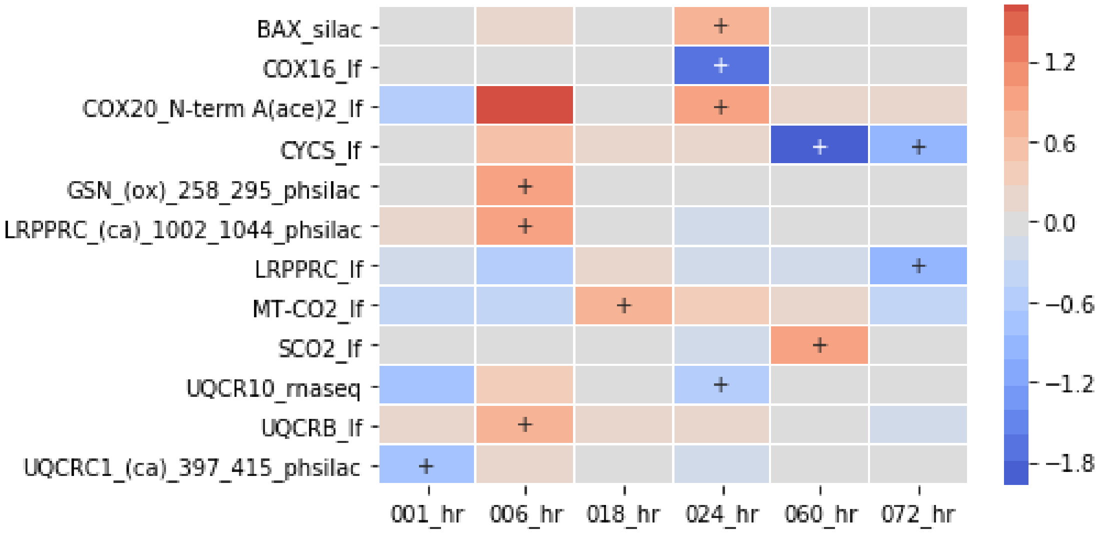
```
<IPython.core.display.HTML object>
```

Next we can continue to expand this network to explore nodes of interest.

This next function expands a single node and creates plots of the species that are connected to thht node.

[61]:

```
def show_neighbors(node, df, upstream=True, downstream=False, max_dist=1,
                   include_only=None, figsize=None, show_network=False):

    neighbors = net_sub.expand_neighbors(
        network=None,
        nodes=node,
        upstream=upstream,
        downstream=downstream,
        max_dist=max_dist,
        include_only=include_only
    )
    df_copy = df.subset(neighbors.nodes).copy()

    # remove nodes not connected to casp3
    neighbors = utils.delete_disconnected_network(neighbors)

    # moves a time point if no signficant changes
    df_copy.require_n_sig(n_sig=1, inplace=True, index='sample_id',␣
↪columns='label',)

    # removes measured species if no signficant changes
    df_copy.require_n_sig(n_sig=1, index='label', inplace=True)
    if show_network:
        # export image
        s_name =  'node_{}.png'.format(node)
        exporters.export_to_dot(neighbors, s_name, image_format='png',␣
↪engine='circo')

        # display image
        display(Image(s_name, width=400))

    # create heatmap of the neighnor nodes
    fig = df_copy.heatmap(
        rank_index=True,
        index='label',
        linewidths=0.01,
        figsize=figsize
    );


show_neighbors('CYCS',
               exp_data.species,
               upstream=True,
               downstream=False,
               max_dist=1,
               include_only=exp_data.species.sig.id_list
)
```



[62]:
```
expand = net_sub.expand_neighbors(casp3_neighbors,
                                  nodes='CYCS',
                                  upstream=True,
                                  include_only=exp_data.species.sig.id_list)
```

[63]:
```
viz.draw_cyjs(expand)
```

```
<IPython.core.display.HTML object>
```

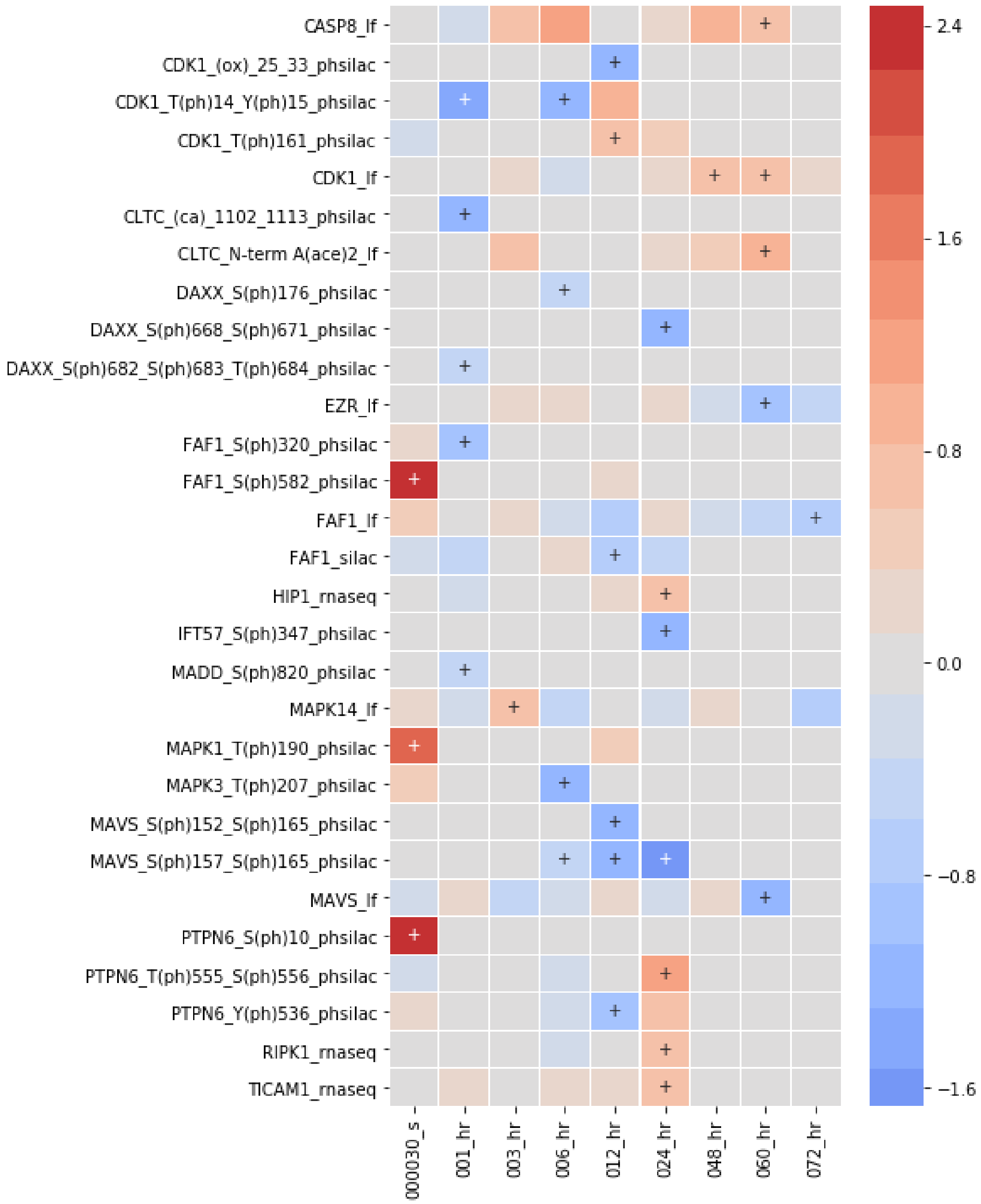
[64]:
```
expand = net_sub.expand_neighbors(expand, nodes='BAX', upstream=True,␣
 ↪include_only=exp_data.species.sig.id_list)
viz.draw_cyjs(expand)
```

```
<IPython.core.display.HTML object>
```

[65]:
```
expand = net_sub.expand_neighbors(expand, nodes='BID', upstream=True,␣
 ↪include_only=exp_data.species.sig.id_list)
viz.draw_cyjs(expand)
```

```
<IPython.core.display.HTML object>
```

[67]:
```
show_neighbors('CASP8',
               exp_data.species,
               upstream=True,
               downstream=False,
               max_dist=1,
               include_only=exp_data.species.sig.id_list,
               figsize=(6, 12)
)
```



[68]:
```
expand = net_sub.expand_neighbors(expand, nodes='CASP8', upstream=True, include_only=exp_data.species.sig.id_list)
```

```
viz.draw_cyjs(expand)
```

```
<IPython.core.display.HTML object>
```

[ ]:

```
show_neighbors(['FAF1', 'MADD', 'DAXX'],
               exp_data.species,
               upstream=True,
               downstream=False,
               max_dist=1,
               include_only=exp_data.species.require_n_sig(n_sig=1).id_list,
               figsize=(8, 6)
)
```

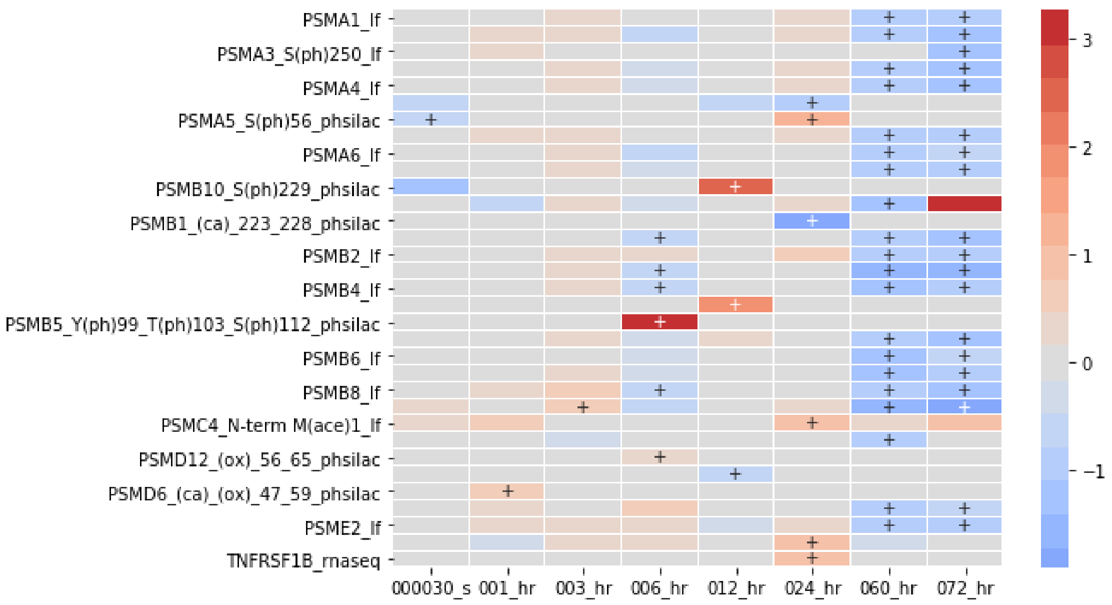
[79]:

```
expand = net_sub.expand_neighbors(expand,
                                  nodes=['FAF1', 'MADD', 'DAXX'],
                                  upstream=True,
                                  include_only=exp_data.species.sig.id_list)
# NBVAL_IGNORE_OUTPUT
viz.draw_cyjs(expand)
```

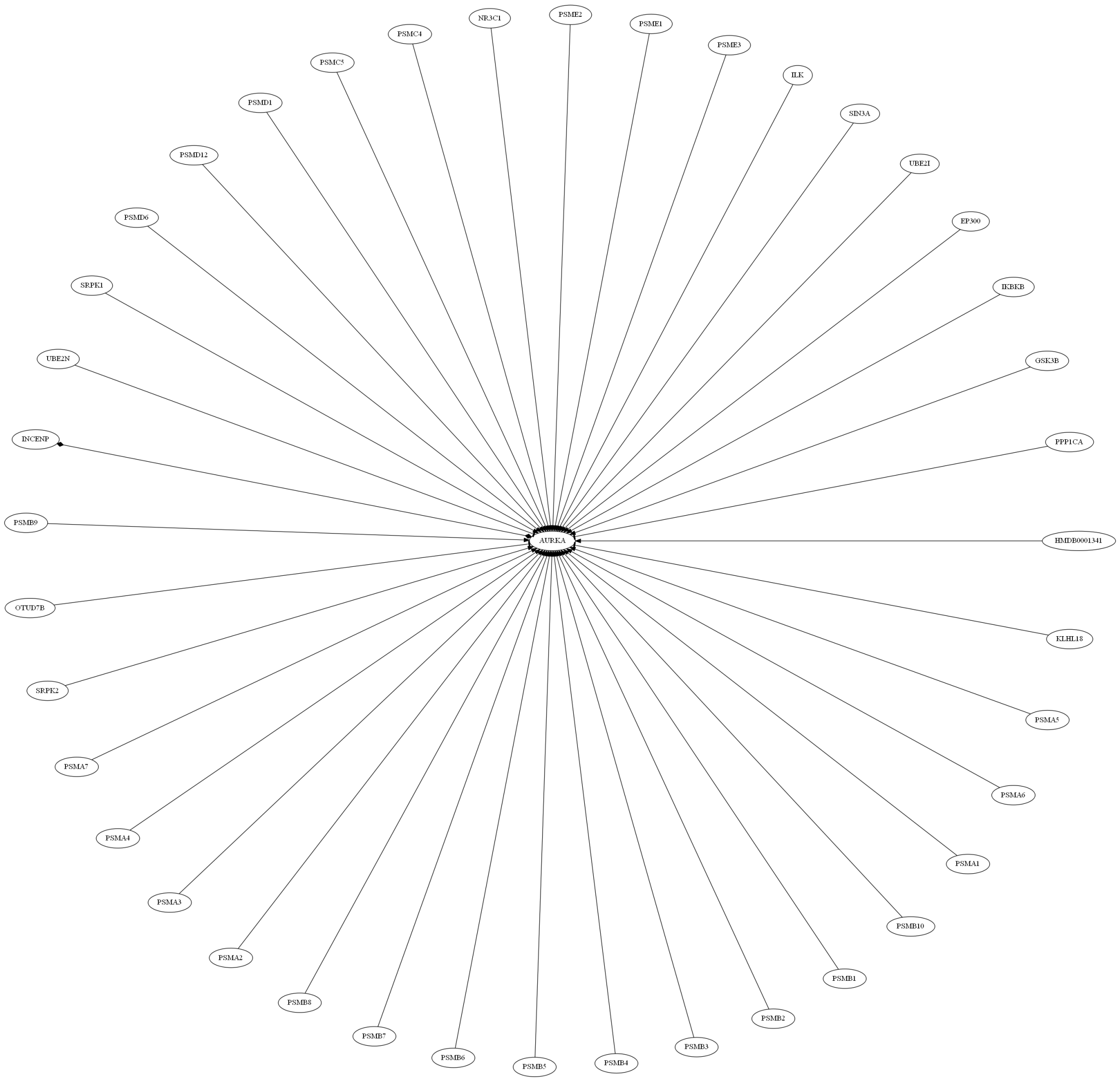
```
<IPython.core.display.HTML object>
```

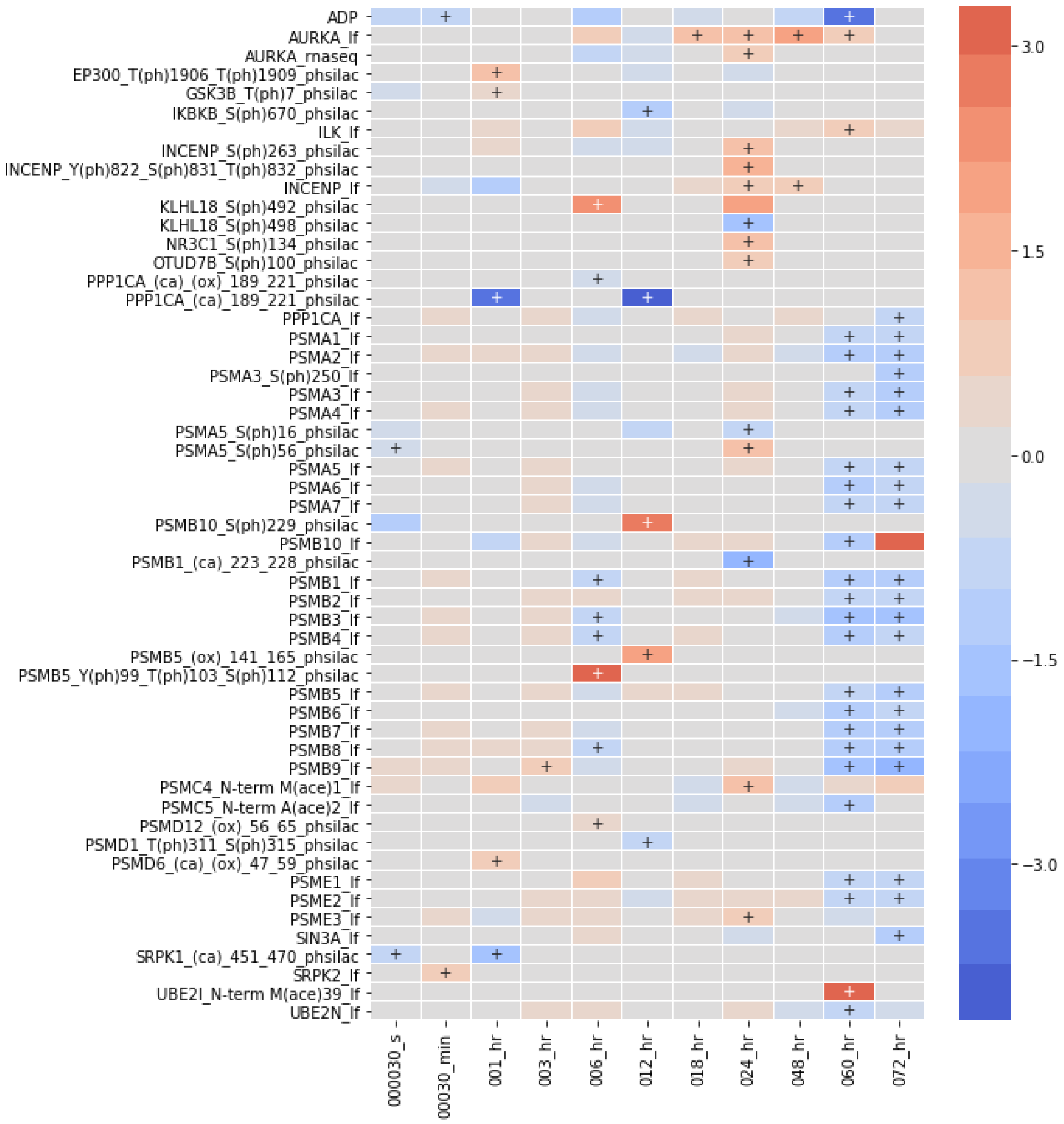
[70]:

```
show_neighbors('TNFRSF1B',
               exp_data.species,
               upstream=True,
               downstream=False,
               max_dist=1,
               include_only=exp_data.species.require_n_sig(n_sig=1).id_list,
               figsize=(8, 6)
)
```



[71]:
```
show_neighbors('AURKA',
               exp_data.species,
               upstream=True,
               downstream=False,
               max_dist=1,
               include_only=exp_data.species.require_n_sig(n_sig=1).id_list,
               figsize=(8, 12),
               show_network=True
)
```


NR3C1
PSME2
PSMC4
PSME1
PSMC5
PSME3
ILK
PSMD1
SIN3A
PSMD12
UBE2I
PSMD6
EP300
SRPK1
IKBKB
UBE2N
GSK3B
INCENP
PPP1CA
PSMB9
AURKA
HMDB0001341
OTUD7B
KLHL18
SRPK2
PSMA5
PSMA7
PSMA6
PSMA4
PSMA1
PSMA3
PSMB10
PSMA2
PSMB1
PSMB8
PSMB2
PSMB7
PSMB3
PSMB6
PSMB4
PSMB5



[83]:
```
expand = net_sub.expand_neighbors(expand, nodes=['AURKA'], upstream=True,␣
    ↪include_only=exp_data.species.require_n_sig(n_sig=2).id_list)
# NBVAL_IGNORE_OUTPUT
viz.draw_cyjs(expand)
```

```
<IPython.core.display.HTML object>
```

This last network has gotten a little hard to explore. We can simplify it by just looking for a single path, in this case AURKA to CYCS

[87]:
```
sub_g_2 = Subgraph(expand)
aurka_to_cycs = sub_g_2.paths_between_pair('AURKA', 'CYCS')
```

[88]:

```
viz.draw_cyjs(aurka_to_cycs)
```

```
```

Data S3. Enrichment analysis demonstration. Related to Figure 4 and STAR methods.

December 6, 2019

[1]:
```
from IPython.display import display
%matplotlib inline

import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
# load the experimental data
from exp_data import exp_data
```

0.0.1 Running enrichment analysis via EnrichR

MAGINE allows users to upload lists of genes for analysis and retrieves the results in an `EnrichmentResult` Class.

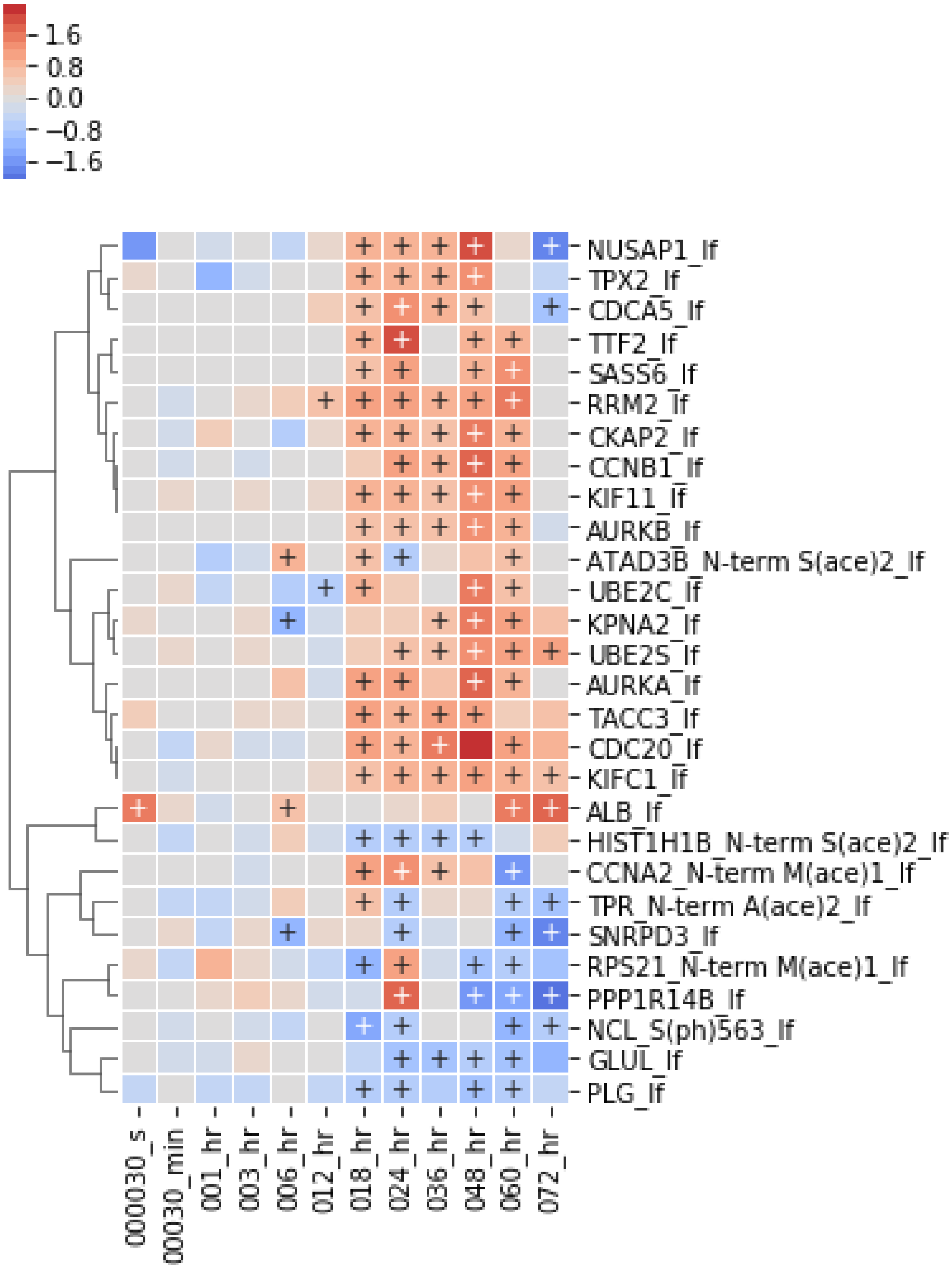
[ ]:
```
from magine.enrichment.enrichr import Enrichr
e = Enrichr()
```

[8]:
```
help(e)
```

```
Help on Enrichr in module magine.enrichment.enrichr object:

class Enrichr(builtins.object)
 |  Enrichr(verbose=False)
 |
 |  Methods defined here:
 |
 |  __init__(self, verbose=False)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  print_valid_libs(self)
 |      Print a list of all available libraries EnrichR has to offer.
 |
 |  run(self, list_of_genes, gene_set_lib='GO_Biological_Process_2017')
 |      Parameters
 |      ----------
 |      list_of_genes : list_like
 |          List of genes using HGNC gene names
 |      gene_set_lib : str or list
 |          Name of gene set library
 |          To print options use Enrichr.print_valid_libs
 |
 |
 |      Examples
 |      --------
 |      >>> import pandas as pd
 |      >>> pd.set_option('display.max_colwidth', 40)
 |      >>> pd.set_option('precision', 3)
 |      >>> e = Enrichr()
 |      >>> df = e.run(['BAX', 'BCL2', 'CASP3', 'CASP8'],
gene_set_lib='Reactome_2016')
 |      >>> print(df[['term_name','combined_score']].head(5))#doctest:
+NORMALIZE_WHITESPACE
 |                                           term_name  combined_score
 |          0  intrinsic pathway for apoptosis_hsa_...      48.157
 |          1  programmed cell death_hsa_r-hsa-5357801      41.516
 |          2                apoptosis_hsa_r-hsa-109581      41.403
 |          3  caspase-mediated cleavage of cytoske...      27.349
 |          4  caspase activation via extrinsic apo...      22.438
 |
 |      Returns
 |      -------
 |      df : EnrichmentResult
 |          Results from enrichR
 |
 |  run_samples(self, sample_lists, sample_ids,
gene_set_lib='GO_Biological_Process_2017', save_name=None, create_html=False,
out_dir=None, run_parallel=False, exp_data=None, pivot=False)
 |      Run enrichment analysis on a list of samples.
 |
 |      Parameters
 |      ----------
 |      sample_lists : list_like
 |          List of lists of genes for enrichment analysis
 |      sample_ids : list
 |          list of ids for the provided sample list
 |      gene_set_lib : str, list
 |          Type of gene set, refer to Enrichr.print_valid_libs
 |      save_name : str, optional
 |          if provided it will save a file as a pivoted table with
 |          the term_ids vs sample_ids
 |      create_html : bool
 |          Creates html of output with plots of species across sample
 |      out_dir : str
 |          If create_html, it will place all html plots into this directory
 |      run_parallel : bool
 |          If create_html, it will create plots using multiprocessing
 |      exp_data : magine.data.ExperimentalData
 |          Must be provided if create_html=True
 |      pivot : bool
 |
 |      Examples
 |      --------
 |      .. plot::
 |          :context: close-figs
 |
 |          >>> import pandas as pd
 |          >>> import matplotlib.pyplot as plt
 |          >>> from magine.enrichment.enrichr import Enrichr
 |          >>> pd.set_option('display.max_colwidth', 40)
 |          >>> pd.set_option('precision', 3)
 |          >>> samples = [['BAX', 'BCL2', 'CASP3', 'CASP8'], ['ATR', 'ATM',
'TP53', 'CHEK1']]
 |          >>> sample_ids = ['apoptosis', 'dna_repair']
 |          >>> e = Enrichr()
 |          >>> df = e.run_samples(samples, sample_ids,
gene_set_lib='Reactome_2016')
 |          >>> print(df[['term_name','combined_score']].head(5))#doctest:
+NORMALIZE_WHITESPACE
 |                                                 term_name  combined_score
 |              0  intrinsic pathway for apoptosis_hsa_...      48.157
 |              1  programmed cell death_hsa_r-hsa-5357801      41.516
 |              2               apoptosis_hsa_r-hsa-109581      41.403
 |              3  caspase-mediated cleavage of cytoske...      27.349
 |              4  caspase activation via extrinsic apo...      22.438
 |          >>> df.filter_multi(rank=10, inplace=True)
 |          >>> df['term_name'] = df['term_name'].str.split('_').str.get(0)
 |          >>> fig = df.sig.heatmap(figsize=(6, 6), linewidths=.05)
 |
 |      Returns
 |      -------
 |      EnrichmentResult
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

[5]:

```
# from supplement_notebook_1
exp_data.label_free.heatmap(
    index='label',
    linewidths=0.01,
    cluster_row=True,
    min_sig=4,
    figsize=(4, 8)
);
```



[10]:

```python
df = e.run_samples(
    [exp_data.label_free.up.require_n_sig(n_sig=4).id_list,
     exp_data.label_free.down.require_n_sig(n_sig=4).id_list,],
    ['label_free_up', 'label_free_down'],
    gene_set_lib='Reactome_2016')
df.term_name = df.term_name.str.split('_').str.get(0)
```

[ ]:

```python
df.heatmap(
    min_sig=1,
    linewidths=0.01,
    convert_to_log=False,
    figsize=(3, 12)
);
print(up_only.shape)
```

[24]:

```python
df.head(10)
```

[24]:

```
                                           term_name  rank       p_value  \
0                                cell cycle, mitotic     1  1.545920e-09
1                                         cell cycle     2  7.603575e-09
2  apc/c-mediated degradation of cell cycle proteins     3  7.316318e-09
3                   regulation of mitotic cell cycle     4  7.316318e-09
4            resolution of sister chromatid cohesion     5  1.279403e-06
5                               mitotic prometaphase     6  1.746644e-06
6  regulation of tp53 activity through phosphoryl...     7  8.341746e-07
7                 transcriptional regulation by tp53     8  8.074179e-06
8                                    g2/m transition     9  1.178583e-05
9                             mitotic g2-g2/m phases    10  1.233244e-05

    z_score  combined_score   adj_p_value  \
0 -2.477071       50.253950  2.056073e-07
1 -2.434067       45.504030  2.528189e-07
2 -2.284682       42.799318  2.528189e-07
3 -2.274341       42.605591  2.528189e-07
4 -2.047109       27.777462  2.836010e-05
5 -1.999010       26.502501  3.318624e-05
6 -1.893455       26.502358  2.218904e-05
7 -2.238861       26.254767  1.145954e-04
8 -2.098984       23.820552  1.261703e-04
9 -2.099087       23.726560  1.261703e-04

                                           genes  n_genes             db  \
0  AURKA,AURKB,CCNA2,CCNB1,CDC20,CDCA5,RRM2,TPX2        8  Reactome_2016
1  AURKA,AURKB,CCNA2,CCNB1,CDC20,CDCA5,RRM2,TPX2        8  Reactome_2016
2                  AURKA,AURKB,CCNA2,CCNB1,CDC20        5  Reactome_2016
3                  AURKA,AURKB,CCNA2,CCNB1,CDC20        5  Reactome_2016
```

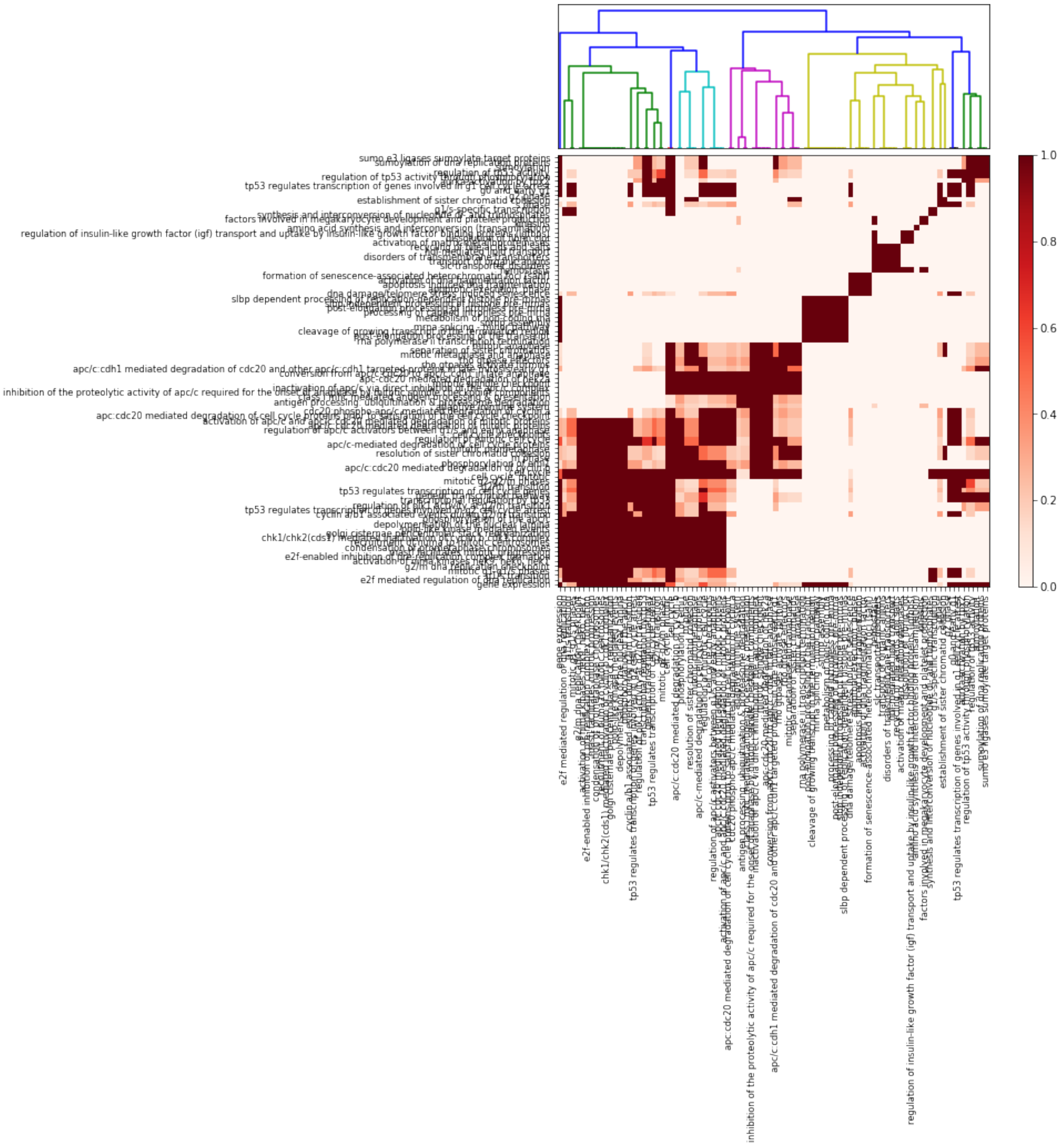
```
4                        AURKB,CCNB1,CDC20,CDCA5        4  Reactome_2016
5                        AURKB,CCNB1,CDC20,CDCA5        4  Reactome_2016
6                         AURKA,AURKB,CCNA2,TPX2        4  Reactome_2016
7                   AURKA,AURKB,CCNA2,CCNB1,TPX2        5  Reactome_2016
8                         AURKA,CCNA2,CCNB1,TPX2        4  Reactome_2016
9                         AURKA,CCNA2,CCNB1,TPX2        4  Reactome_2016

   significant      sample_id
0         True  label_free_up
1         True  label_free_up
2         True  label_free_up
3         True  label_free_up
4         True  label_free_up
5         True  label_free_up
6         True  label_free_up
7         True  label_free_up
8         True  label_free_up
9         True  label_free_up
```

If we look at the top ranked terms, we see that some fo them have similar descriptions `cell cycle`, `cell cycle, mitotic`, `regulation of mitotic cell cycle`. If we look at the gene list, we can also see that some of the genes are similar. To see if there are redundant terms that are enriched, we can calculate their similarity with the Jaccard Index (intersection over union).

[22]:
```
d = df.find_similar_terms('cell cycle', remove_subset=False )
display(d.head(20))
```

```
                                            term_name  similarity_score
0                                 cell cycle, mitotic             1.000
2                    regulation of mitotic cell cycle             0.625
23                      generic transcription pathway             0.625
52                                    gene expression             0.625
6                  transcriptional regulation by tp53             0.625
1   apc/c-mediated degradation of cell cycle proteins             0.625
8                              mitotic g2-g2/m phases             0.500
11                                            m phase             0.500
9                         regulation of tp53 activity             0.500
7                                     g2/m transition             0.500
5   regulation of tp53 activity through phosphoryl...             0.500
4                                mitotic prometaphase             0.500
3             resolution of sister chromatid cohesion             0.500
18                    separation of sister chromatids             0.375
22                             cell cycle checkpoints             0.375
21                     mitotic metaphase and anaphase             0.375
20                             mitotic g1-g1/s phases             0.375
19                                   mitotic anaphase             0.375
15  regulation of apc/c activators between g1/s an...             0.375
16                                    g1/s transition             0.375
```

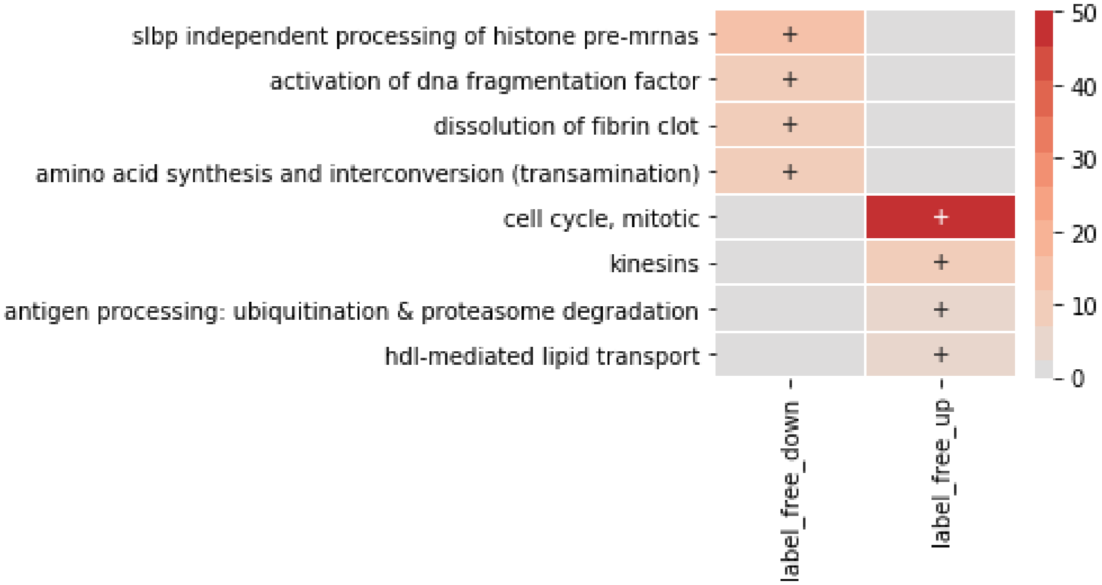
[25]:
```
# We can visualize this with the dist_matrix function
df.dist_matrix(figsize=(9,9));
```



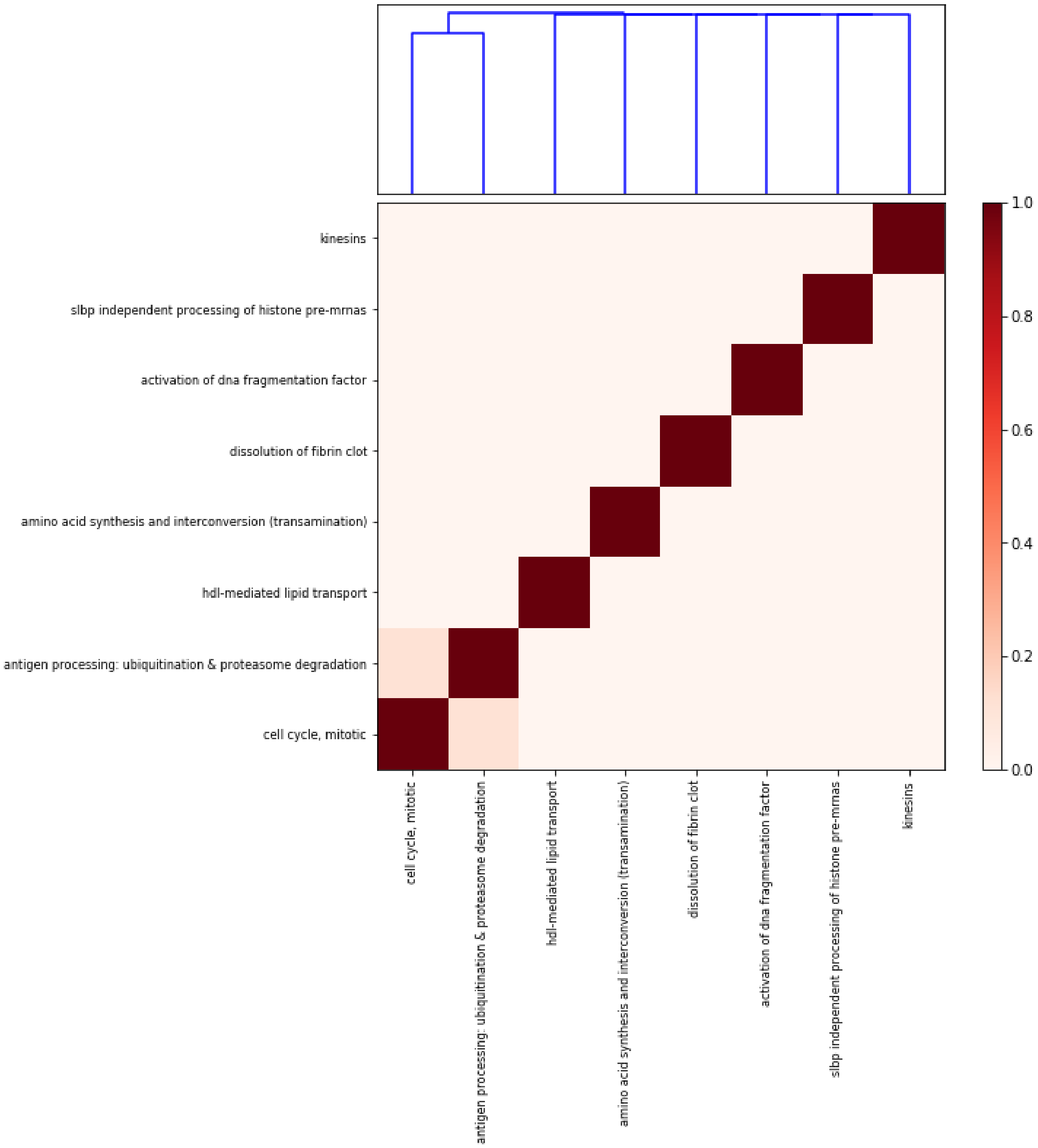
[26]:
```
# we can apply it to remove the lesser enriched terms with the remove_redundant␣
↪function
df_slim = df.remove_redundant(level='dataframe')
```

```
Number of rows went from 92 to 8
```

[27]:
```
df_slim.heatmap(
    min_sig=1,
    linewidths=0.01,
    convert_to_log=False,
    figsize=(3, 3)
);
```



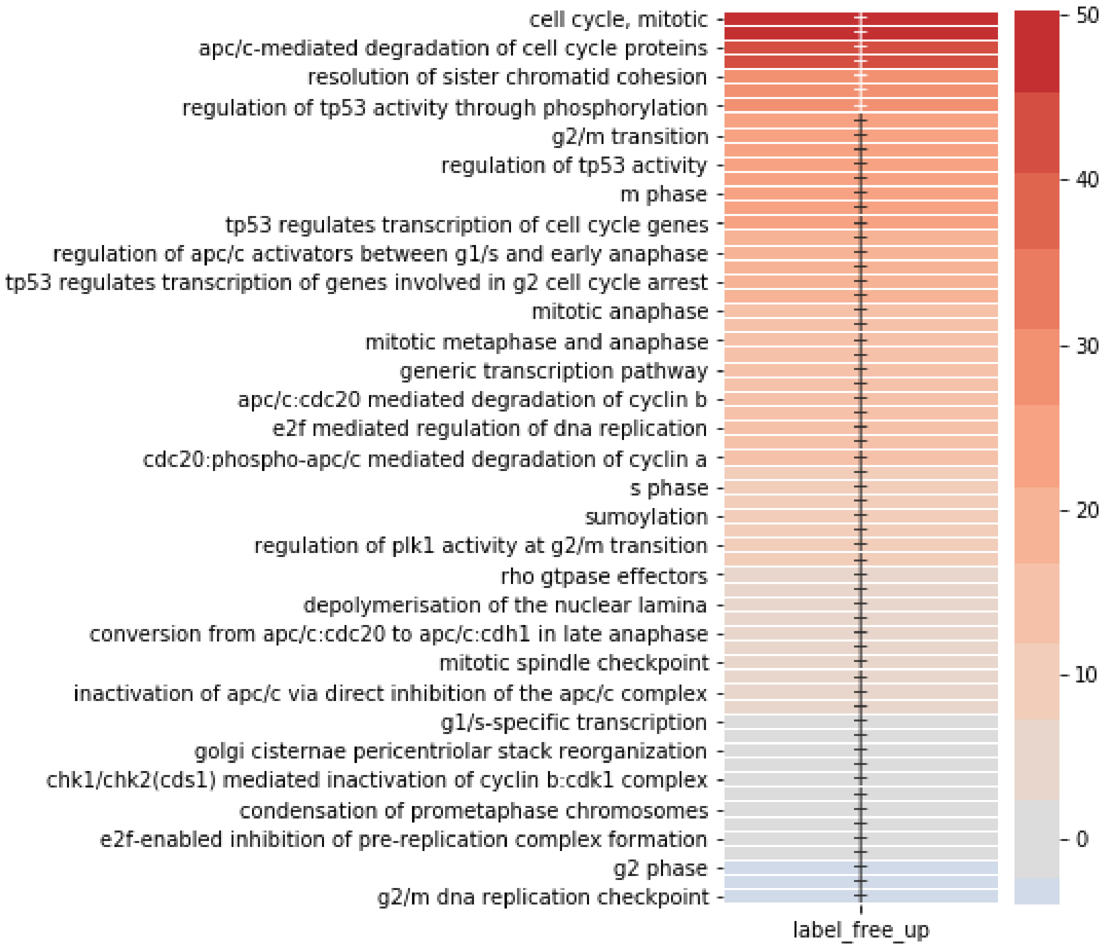
[28]:
```
df_slim.dist_matrix(figsize=(9,9));
```



We can still recover the terms removed based on the highest level term kept.

[29]:
```
df.show_terms_below('cell cycle, mitotic').heatmap(
    linewidths=0.01,
    convert_to_log=False,
    figsize=(3, 8));
```

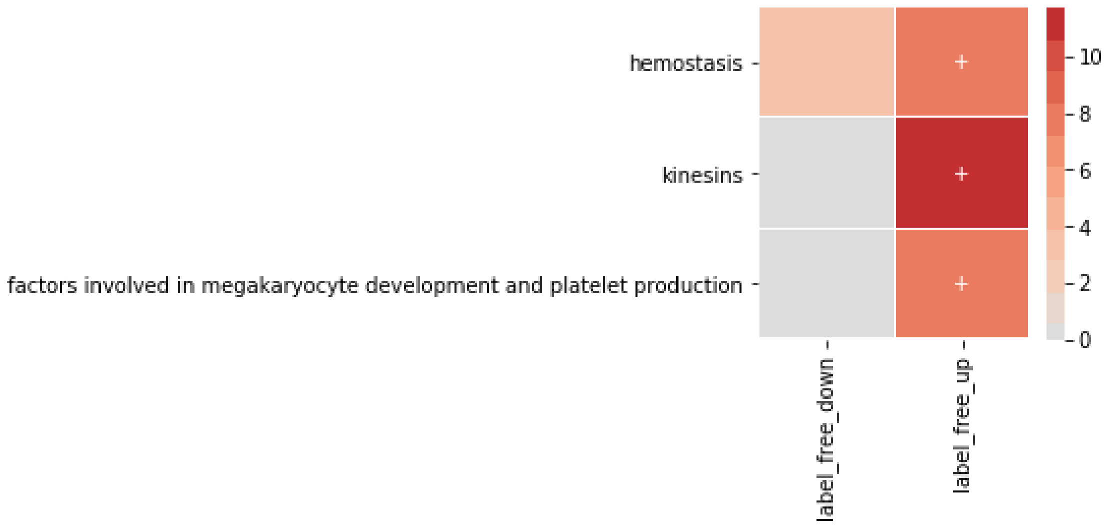
```
Number of rows went from 92 to 8
```



[30]:

```
df.show_terms_below('kinesins').heatmap(
    linewidths=0.01,
    convert_to_log=False,
    figsize=(3, 3));
```

```
Number of rows went from 92 to 8
```



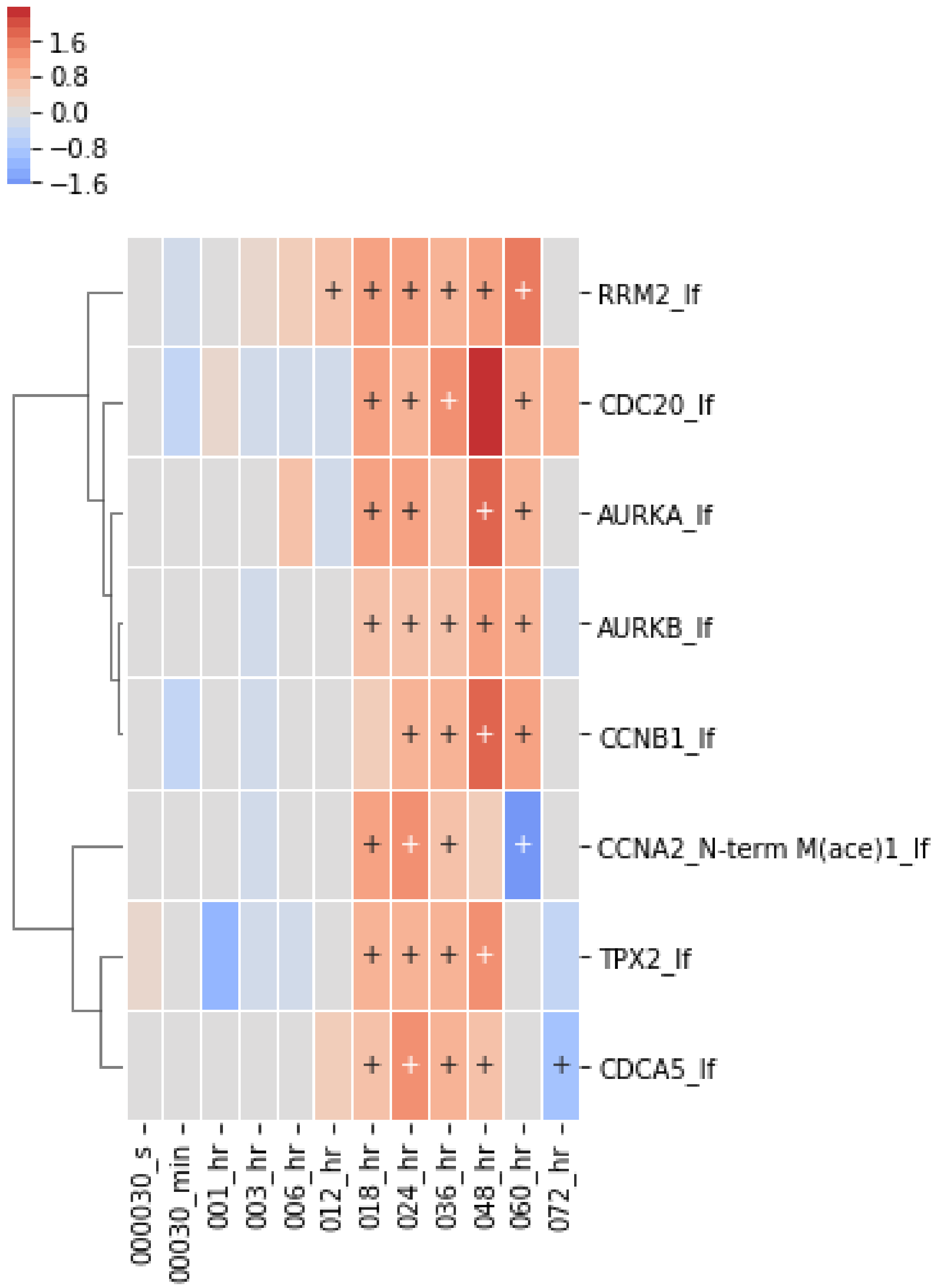
We can use these enriched terms and create a heatmap of those terms.

[ ]:

```
exp_data.label_free.heatmap(
    df_slim.term_to_genes('cell cycle, mitotic'),
    subset_index='identifier',
    index='label',
    linewidths=0.01,
    cluster_row=True,
    min_sig=4,
    figsize=(4, 8)
);
```

[32]:

```
from magine.utils
genes_in_labels = utils.create_dict_from_node_attributes(mol_net, 'termName')
heatmap_by_terms(
    exp_data.rna,
    convert_to_log=True,
    index='label',
    term_labels=list(genes_in_labels.keys()),
    term_sets=list(genes_in_labels.values()),
    div_colors=True,
    linewidths=0.01,
    min_sig=1,
    annotate_sig=True,
    cluster_col=False,
    cluster_row=False,
    y_tick_labels=True,
    figsize=(8, 8)
);
```



0.1 Enrichment analysis

We can use the ExperimentalData class to filter the data to create lists of genes for further analysis. We take these lists and run enrichment analysis using Enrichr. Since this part is time consuming, it is best to do it outside of a notebook. The code to do so can be found in "run_enrichment.py". The results will be a csv file that we will load next.

```
[ ]: enrichment_array = load_enrichment_csv('Data/bendamustine_enrichment.csv.gz',␣
      ↪index_col=0)
```

```
[ ]: drug_dbs = [
         'DrugMatrix',
         'Drug_Perturbations_from_GEO_2014',
     #     'LINCS_L1000_Chem_Pert_down',
     #     'LINCS_L1000_Chem_Pert_up'
     ]
     drug = enrichment_array.filter_multi(
         p_value=0.05,
     #     combined_score=0.0,
         db=drug_dbs,
     #     rank=100,
         category=['ph_silac_down', 'ph_silac_up',],
     )
     drug.sort_values('rank', inplace=True, ascending=True)
     # drug.term_name = drug.term_name.str.split('_').str.get(0)
     drug.head(10)
     drug = clean_drug_dbs(drug)
```

```
[ ]: word_cloud = create_wordcloud(drug)
     word_cloud.plot();
```

Data S4. Annotated gene set network construction and exploration. Related to STAR METHODS

December 6, 2019

1 Exploring enrichment analysis of HL60 response to bendamustine.

This is the second notebook for the Pino Et. Al. In this notebook we demonstrate how MAGINE can be used to explore the enrichment analysis.

[1]:
```python
from IPython.display import display, Image
%matplotlib inline
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd
pd.set_option('display.precision', 2)
pd.set_option('display.max_colwidth', 50)
%load_ext autoreload
%autoreload 2
```

[2]:
```python
# load magine specific tools
from magine.plotting.wordcloud_tools import create_wordcloud
from magine.plotting.heatmaps import heatmap_by_terms
from magine.plotting.venn_diagram_maker import create_venn3, create_venn2

from magine.enrichment import load_enrichment_csv
from magine.networks import visualization as vis
from magine.networks import utils, exporters

from magine.networks.annotated_set import create_subnetwork
from magine.networks.subgraphs import Subgraph
```

2 Exploring enrichment output

2.1 Loading data and networks

[3]:
```python
from exp_data import exp_data
```

Load enrichment array. This `bendamustine_enrichment.csv.gz` was created by `run_enrichment.py` script. If it doesn't exist, run that file to generate the results. Due to the number of samples, we run this outside a Jupyter notebook as it can take quite a bit of time.

[4]:

```
enrichment_array = load_enrichment_csv('Data/bendamustine_enrichment.csv.gz',␣
 ↪index_col=0)

enrichment_array['significant'] = False
enrichment_array.loc[(enrichment_array['adj_p_value'] <= 0.05) &
                     (enrichment_array['combined_score'] > 0.0),
                     'significant'] = True
# Remove terms that are not signficant in at least one time point/sample/
 ↪category
enrichment_array.require_n_sig(
    columns='sample_id',
    index='term_name',
    n_sig=1,
    inplace=True
)
```

[5]:

```
enrichment_array[['term_name', 'db', 'category']].nunique()
```

[5]:

```
term_name    20758
db              52
category        15
dtype: int64
```

[6]:

```
display(enrichment_array.head(5))
```

```
            term_name  rank  combined_score  adj_p_value  \
0      vinblastine-up     1           34.53     2.83e-05
1         mitotane-up     2           31.90     3.64e-05
2  dideoxycytidine-dn     3           30.15     1.38e-04
3          busulfan-dn     4           29.18     1.01e-04
4         mitotane-up     5           28.76     1.39e-04

                                               genes  n_genes  sample_id  \
0  ADSS,APOA1,APOE,BSG,BTF3,CANX,COPA,DNAJC9,EI24...       24   000030_s
1  ABI1,ACLY,ALDH3A2,BTF3,CD97,COL4A3BP,COPA,GIT2...       23   000030_s
2  ACADVL,ACLY,ALDH3A2,APOA1,APOA2,APOE,BSG,CANX,...       23   000030_s
3  ABI1,BTF3,CANX,COPA,DNAJC9,FBXW5,FLNA,HMHA1,HP...       22   000030_s
4  ACLY,ADSS,AKAP8L,BTF3,CANX,CD97,COL4A3BP,COPA,...       21   000030_s

         category          db  significant
0  proteomics_both  DrugMatrix         True
1  proteomics_both  DrugMatrix         True
2  proteomics_both  DrugMatrix         True
3  proteomics_both  DrugMatrix         True
4  proteomics_both  DrugMatrix         True
```

[7]:

```
# clean up printing by selecting fewer columns
cols = ['term_name', 'rank', 'combined_score', 'adj_p_value',
       'n_genes', 'sample_id', 'category']
```

[8]:
```
display(enrichment_array[cols].head(5))
```

```
           term_name  rank  combined_score  adj_p_value  n_genes sample_id  \
0      vinblastine-up     1           34.53     2.83e-05       24  000030_s
1         mitotane-up     2           31.90     3.64e-05       23  000030_s
2  dideoxycytidine-dn     3           30.15     1.38e-04       23  000030_s
3         busulfan-dn     4           29.18     1.01e-04       22  000030_s
4         mitotane-up     5           28.76     1.39e-04       21  000030_s

          category
0  proteomics_both
1  proteomics_both
2  proteomics_both
3  proteomics_both
4  proteomics_both
```

Load network Load in the network and initialize Subgraph. We will use this later to construct networks from queries.

[9]:
```
network = nx.read_gpickle('Networks/bendamustine_network_w_attributes.p')
net_sub = Subgraph(network)
```

3 Single database exploration

Here we will focus on the Reactome enrichment.

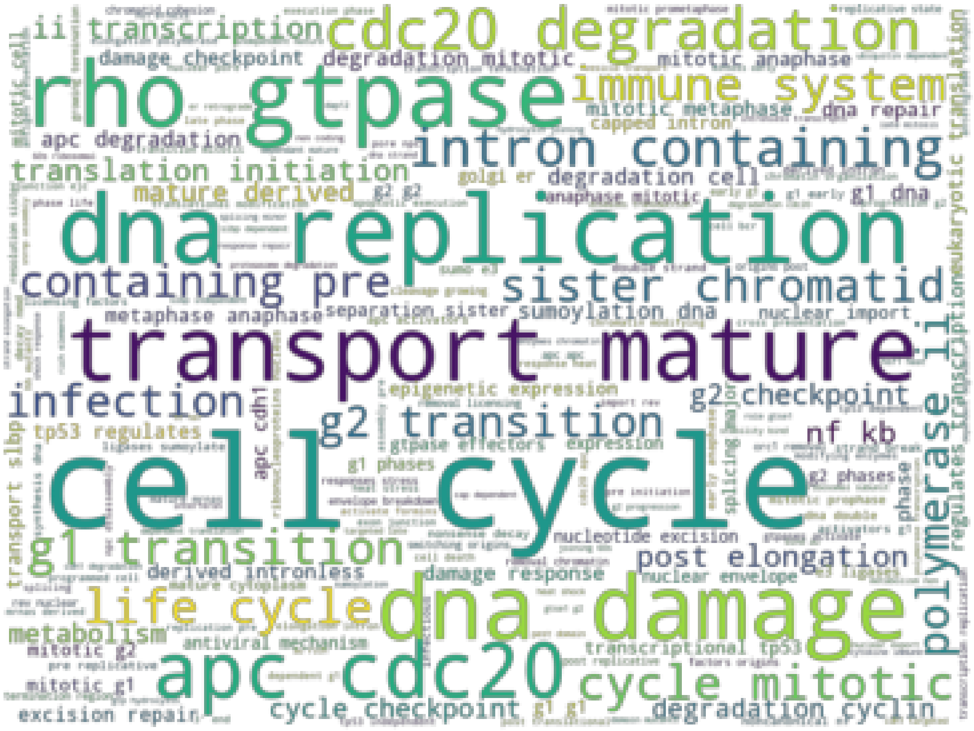
[10]:
```
reactome_only = enrichment_array.filter_multi(
    db='Reactome_2016', # Only reactome db
)
# This just cleans up the term name
display(reactome_only['term_name'].head(5))
reactome_only['term_name'] = reactome_only['term_name'].str.split('_').str.
 ↪get(0)
display(reactome_only['term_name'].head(5))
```

```
80551    processing of capped intron-containing pre-mrn...
80552                         gene expression_hsa_r-hsa-74160
80553                           mrna splicing_hsa_r-hsa-72172
80554         mrna splicing - major pathway_hsa_r-hsa-72163
80555    transport of mature transcript to cytoplasm_hs...
Name: term_name, dtype: object


80551    processing of capped intron-containing pre-mrna
80552                                    gene expression
```

```
80553                                           mrna splicing
80554                            mrna splicing - major pathway
80555         transport of mature transcript to cytoplasm
Name: term_name, dtype: object
```

[11]:

```
# we can use a word cloud to view what terms are enriched
word_cloud = create_wordcloud(reactome_only.sig)
word_cloud.plot();
```



[12]:

```
word_cloud.data.head(20)
```

[12]:

```
                      words  counts
181              cell cycle     325
208         dna replication     160
249              rho gtpase     142
243              dna damage     139
179        transport mature     138
586               apc cdc20     132
587        cdc20 degradation     122
175       intron containing     121
353           immune system      98
199              life cycle      97
228          cycle mitotic      95
```

```
381           g1 transition      94
355        sister chromatid      86
176          containing pre      80
236           polymerase ii      79
400           g2 transition      79
5                 infection      77
429  translation initiation      77
237        ii transcription      75
261          post elongation      73
```

[ ]:

```
```

3.1 Phospho-SILAC enrichment

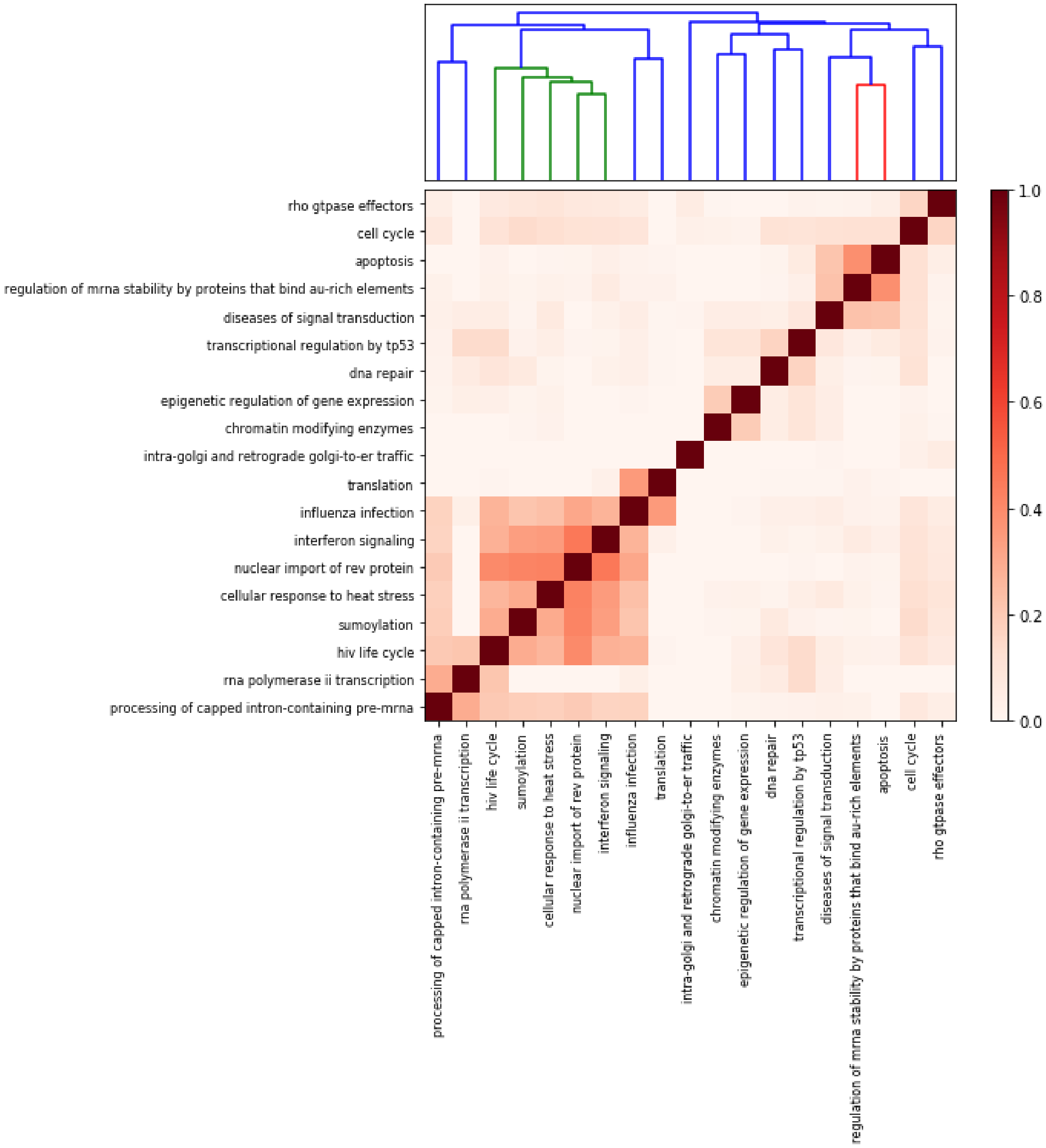
3.1.1 Filtering enrichment output

[13]:

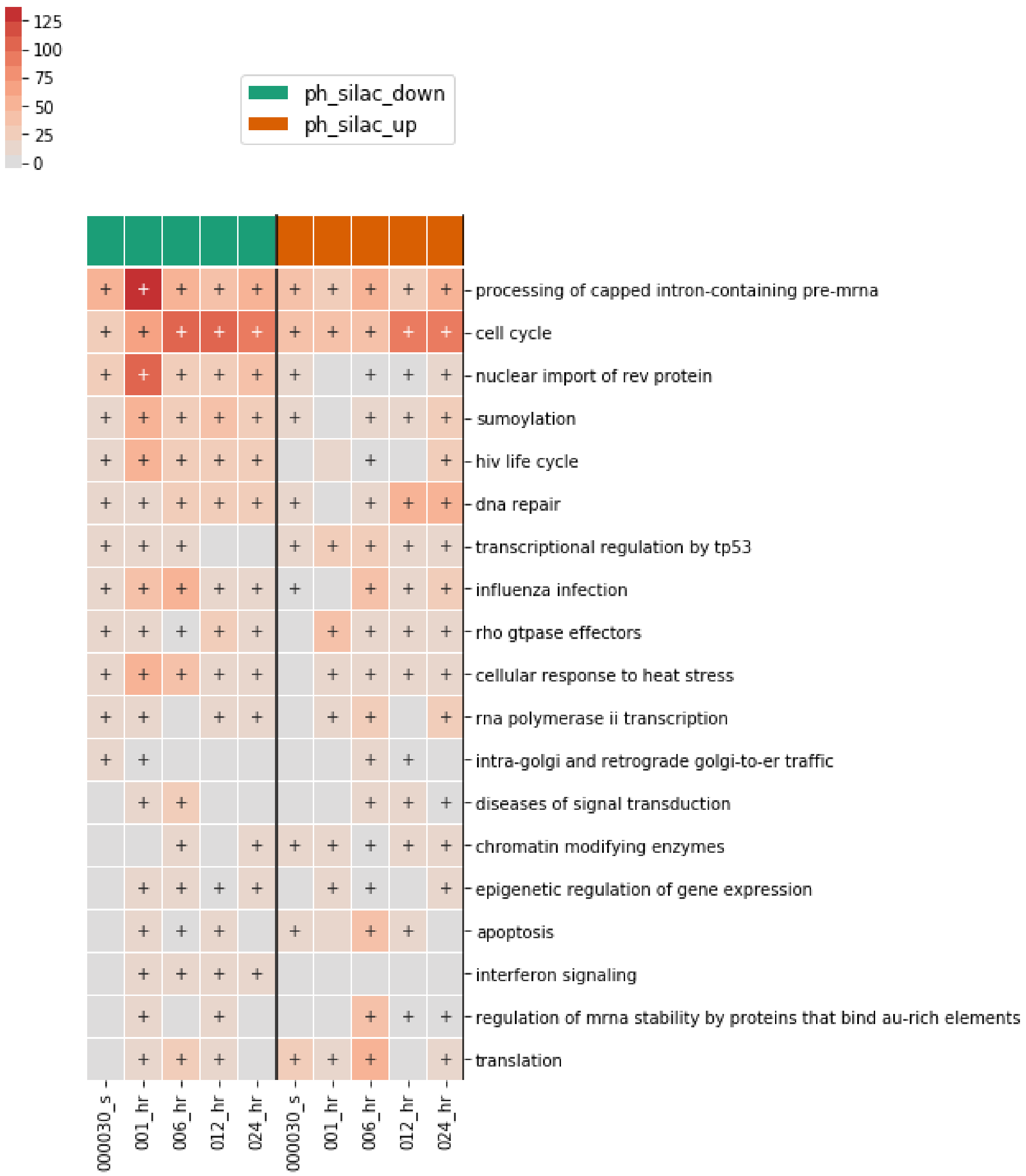
```python
# subset the data to only look at ph-silac data.
# Later on we will look at label-free, then both.

ph_silac = reactome_only.filter_multi(
    category=['ph_silac_up', 'ph_silac_down'],
)

# arbitrary value that can be easily changed.
ph_silac.require_n_sig(n_sig=4, inplace=True)

not_useful = [
    'gene expression', 'translation',
    'immune system',
    'disease', 'diseases of signal transduction',
    'infectious disease',
    'influenza infection', 'influenza life cycle',
    'influenza viral rna transcription and replication',
]
# ph_silac = ph_silac.loc[~ph_silac['term_name'].isin(not_useful)]
ph_silac_copy = ph_silac.copy()

ph_silac.remove_redundant(
    threshold=.5,
    level='dataframe',
    sort_by='combined_score',
    inplace=True
)
ph_silac.remove_redundant(
    threshold=.5,
    level='sample',
    sort_by='combined_score',
    inplace=True
)
```

```
Number of rows went from 110 to 19
Number of rows went from 19 to 19
```

[14]:
```
ph_silac.dist_matrix();
```



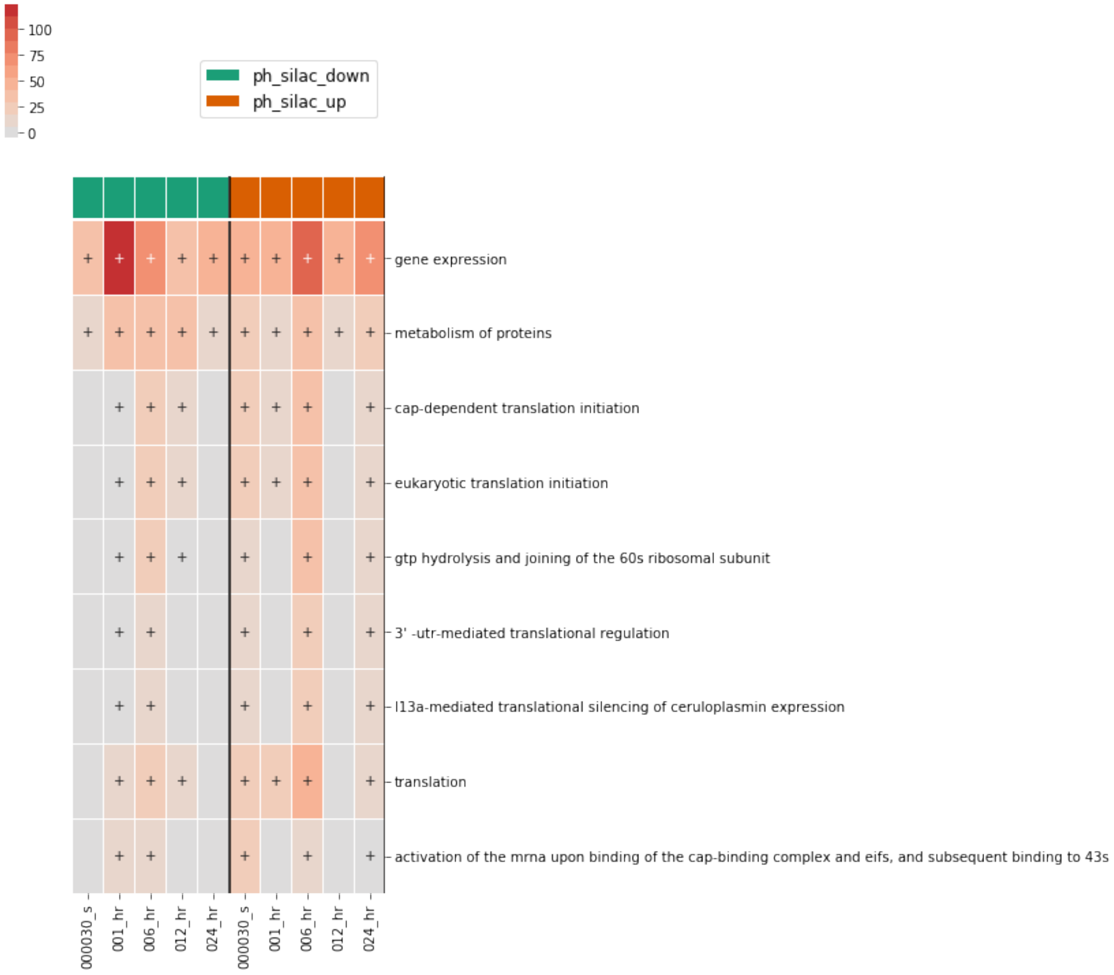
[16]:
```
ph_silac.heatmap(
    convert_to_log=False,
    cluster_by_set=False,
    cluster_row=False,
    values='combined_score',
    columns=['category', 'sample_id'],
    annotate_sig=True,
    div_colors=True,
    linewidths=.005,
    figsize=(5,12)
);
plt.savefig("ph_silac_enriched.png", dpi=300, bbox_inches='tight')
```



[17]:

```
# This figure shows the process of discovering "too general/not useful term".
# Translation involves various genes that make up individual process
ph_silac_copy.show_terms_below("translation").heatmap(
    convert_to_log=False,
    cluster_by_set=False,
    cluster_row=False,
    values='combined_score',
    columns=['category', 'sample_id'],
    annotate_sig=True,
    div_colors=True,
    linewidths=.005,
    figsize=(5, 12)
);
```

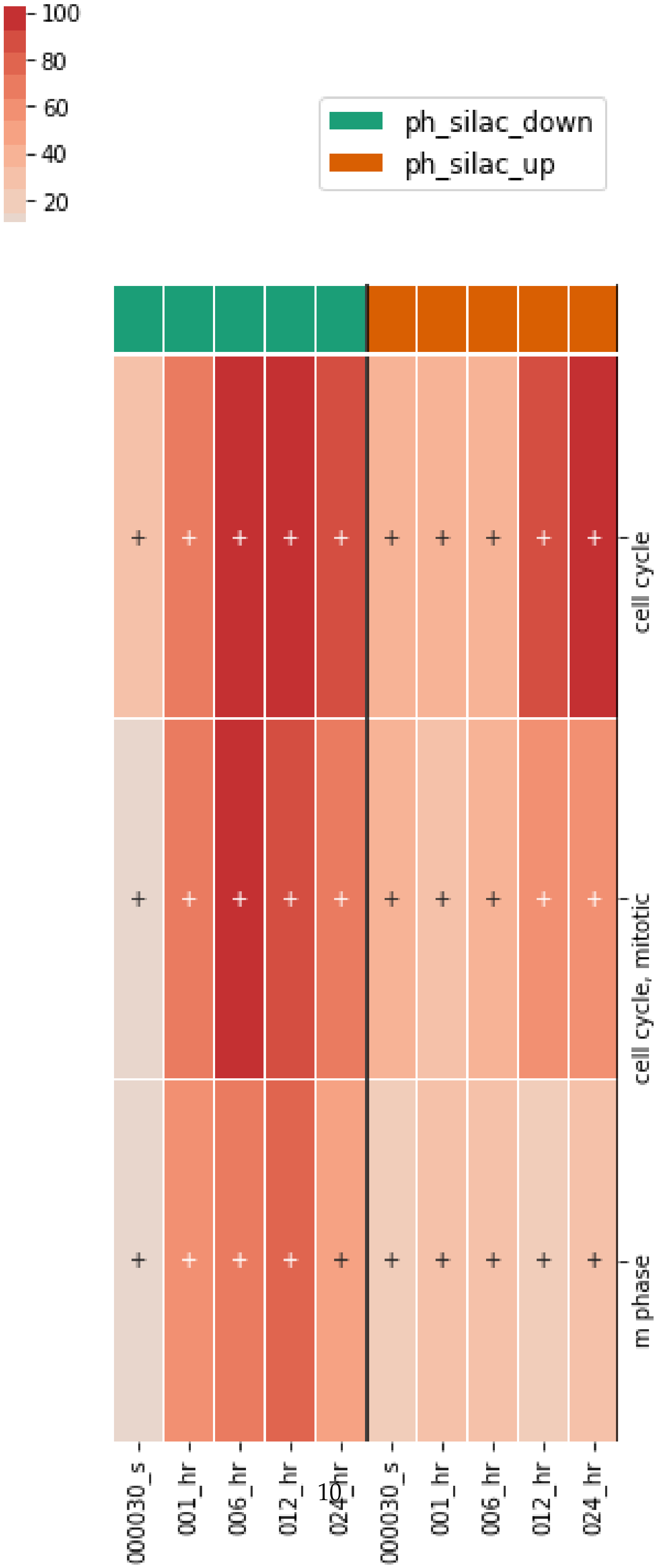
```
Number of rows went from 110 to 23
```



[18]:

```
# We can use the same thought process to find more specific terms
ph_silac_copy.show_terms_below("cell cycle", remove_subset=False, threshold=.5).
 ↪require_n_sig(n_sig=1).heatmap(
    convert_to_log=False,
    cluster_by_set=False,
    cluster_row=False,
    values='combined_score',
    columns=['category', 'sample_id'],
    annotate_sig=True,
    div_colors=True,
    linewidths=.005,
    figsize=(5, 12)
);
plt.savefig("below_cell_cycle.png", bbox_inches='tight', dpi=300)
```

```
Number of rows went from 110 to 19
```


100
80
60
40
20
ph_silac_down
ph_silac_up
cell cycle
cell cycle, mitotic
m phase
000030_s
001_hr
006_hr
012_hr
024_hr
000030_s
001_hr
006_hr
012_hr
024_hr

3.1.2 Creating an annotated set network.

[19]:

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import ndimage
from scipy import misc
import matplotlib.image as mpimg
import imageio

def trip_photo(im_location, title=None):
    """
    Removes whitespace and adds title to image


    Parameters
    ----------
    im_location: str
        location of file, will be used for output as well
    title : str
        title to provide to add to image
        Will be the sample id

    Returns
    -------

    """
    img = mpimg.imread(im_location)
    # mask = img[:, :, 0] < 1

    # img = img[np.ix_(mask.any(1), mask.any(0))]

    plt.imshow(img, interpolation='none')
    plt.xticks([])
    plt.yticks([])
    if title is not None:
        plt.title(title, fontsize=18)
    plt.axis('off')
    out = im_location.replace('.png', '_formatted.png')
    out2 = im_location.replace('.png', '_formatted.svg')
    plt.savefig(out, dpi=1000, bbox_inches='tight', transparent=True)
    plt.savefig(out2, bbox_inches='tight', transparent=True)
    plt.close()
```

```
def create_gif(prefix, timepoints, save_name):
    f_names = ["{}_{}.png".format(prefix, i) for i in tps]
    for i,j in zip(f_names, tps):
        trip_photo(i, j)

    kargs = {'duration': 8 / len(f_names)}

    imageio.mimsave(
        '{}.gif'.format(save_name),
        [imageio.imread(i.replace('.png', '_formatted.png')) for i in f_names],
        **kargs
    )
    return Image('{}.gif'.format(save_name),)
```

[20]:
```
prefix = 'ph_silac_ags'
tps = ['000030s', '001hr', '006hr', '012hr', '024hr']
term_net, mol_net = create_subnetwork(
    ph_silac,
    network=network,
    save_name='ph_silac_ags',
    use_cytoscape=True,
    use_fdr=True, use_threshold=True, min_edges=20,
    scale=150
)
create_gif(prefix, tps, 'scale_150')
```

```
Creating ontology network


      ␣
↪---------------------------------------------------------------------------

       ConnectionRefusedError                    Traceback (most recent call␣
↪last)

       ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\connection.py in␣
↪_new_conn(self)
       158                conn = connection.create_connection(
   --> 159                    (self._dns_host, self.port), self.timeout,␣
↪**extra_kw)
       160


       ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\util\connection.py␣
↪in create_connection(address, timeout, source_address, socket_options)
        79      if err is not None:
   ---> 80          raise err
        81
       ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\util\connection.py␣
↪in create_connection(address, timeout, source_address, socket_options)
        69                 sock.bind(source_address)
   ---> 70             sock.connect(sa)
        71             return sock


       ConnectionRefusedError: [WinError 10061] No connection could be made␣
↪because the target machine actively refused it


   During handling of the above exception, another exception occurred:


       NewConnectionError                        Traceback (most recent call␣
↪last)

       ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\connectionpool.py␣
↪in urlopen(self, method, url, body, headers, retries, redirect,␣
↪assert_same_host, timeout, pool_timeout, release_conn, chunked, body_pos,␣
↪**response_kw)
       599                                                      body=body,␣
↪headers=headers,
   --> 600                                                      chunked=chunked)
       601


       ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\connectionpool.py␣
↪in _make_request(self, conn, method, url, timeout, chunked,␣
↪**httplib_request_kw)
       353             else:
   --> 354                 conn.request(method, url, **httplib_request_kw)
       355


       ~\miniconda3\envs\magine_37\lib\http\client.py in request(self, method,␣
↪url, body, headers, encode_chunked)
      1228             """Send a complete request to the server."""
   -> 1229             self._send_request(method, url, body, headers,␣
↪encode_chunked)
      1230


       ~\miniconda3\envs\magine_37\lib\http\client.py in _send_request(self,␣
↪method, url, body, headers, encode_chunked)
```

```
      1274                 body = _encode(body, 'body')
   -> 1275             self.endheaders(body, encode_chunked=encode_chunked)
      1276


       ~\miniconda3\envs\magine_37\lib\http\client.py in endheaders(self,␣
↪message_body, encode_chunked)
      1223                 raise CannotSendHeader()
   -> 1224             self._send_output(message_body,␣
↪encode_chunked=encode_chunked)
      1225


       ~\miniconda3\envs\magine_37\lib\http\client.py in _send_output(self,␣
↪message_body, encode_chunked)
      1015             del self._buffer[:]
   -> 1016             self.send(msg)
      1017


       ~\miniconda3\envs\magine_37\lib\http\client.py in send(self, data)
       955                 if self.auto_open:
   --> 956                     self.connect()
       957                 else:


       ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\connection.py in␣
↪connect(self)
       180         def connect(self):
   --> 181             conn = self._new_conn()
       182             self._prepare_conn(conn)


       ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\connection.py in␣
↪_new_conn(self)
       167                 raise NewConnectionError(
   --> 168                     self, "Failed to establish a new connection: %s" % e)
       169


       NewConnectionError: <urllib3.connection.HTTPConnection object at␣
↪0x000001D71B544710>: Failed to establish a new connection: [WinError 10061] No␣
↪connection could be made because the target machine actively refused it


   During handling of the above exception, another exception occurred:
    MaxRetryError                             Traceback (most recent call␣
↪last)

    ~\miniconda3\envs\magine_37\lib\site-packages\requests\adapters.py in␣
↪send(self, request, stream, timeout, verify, cert, proxies)
    448                     retries=self.max_retries,
--> 449                     timeout=timeout
    450                 )


    ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\connectionpool.py␣
↪in urlopen(self, method, url, body, headers, retries, redirect,␣
↪assert_same_host, timeout, pool_timeout, release_conn, chunked, body_pos,␣
↪**response_kw)
    637             retries = retries.increment(method, url, error=e,␣
↪_pool=self,
--> 638                                         _stacktrace=sys.
↪exc_info()[2])
    639             retries.sleep()


    ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\util\retry.py in␣
↪increment(self, method, url, response, error, _pool, _stacktrace)
    398         if new_retry.is_exhausted():
--> 399             raise MaxRetryError(_pool, url, error or␣
↪ResponseError(cause))
    400


    MaxRetryError: HTTPConnectionPool(host='localhost', port=1234): Max␣
↪retries exceeded with url: /v1/styles/visualproperties (Caused by␣
↪NewConnectionError('<urllib3.connection.HTTPConnection object at␣
↪0x000001D71B544710>: Failed to establish a new connection: [WinError 10061] No␣
↪connection could be made because the target machine actively refused it'))


During handling of the above exception, another exception occurred:


    ConnectionError                           Traceback (most recent call␣
↪last)

    <ipython-input-20-6f0dfd01020f> in <module>
      7     use_cytoscape=True,
      8     use_fdr=True, use_threshold=True, min_edges=20,
----> 9     scale=150
```

```
        10 )
        11 create_gif(prefix, tps, 'scale_150')


       E:\PycharmProjects\PycharmProjects\Magine\magine\networks\annotated_set.
↪py in create_subnetwork(df, network, terms, save_name, draw_png,␣
↪remove_isolated, use_cytoscape, merge, out_dir, use_threshold, use_fdr,␣
↪min_edges, scale)
       310     if use_cytoscape:
       311         from magine.networks.visualization.cytoscape import␣
↪RenderModel
   --> 312         rm = RenderModel(term_g, layout='force-directed')
       313         rm.visualize_by_list_of_time(labels,
       314                                      prefix=save_name,


       E:
↪\PycharmProjects\PycharmProjects\Magine\magine\networks\visualization\cytoscape.
↪py in __init__(self, graph, layout, style)
        76
        77         self.graph = graph
   ---> 78         self.cy = CyRestClient()
        79         self.cy.session.delete()
        80         self.cy.layout2 = LayoutClient()


       E:
↪\PycharmProjects\PycharmProjects\py2cytoscape\py2cytoscape\data\cyrest_client.
↪py in __init__(self, ip, port, version)
        17
        18         self.network = NetworkClient(self.__url)
   ---> 19         self.style = StyleClient(self.__url)
        20         self.layout = LayoutClient(self.__url)
        21         self.edgebundling = EdgeBundlingClient(self.__url)


       E:
↪\PycharmProjects\PycharmProjects\py2cytoscape\py2cytoscape\data\style_client.
↪py in __init__(self, url)
        14         self.__url_apply = url + 'apply/styles/'
        15
   ---> 16         self.vps = VisualProperties(url)
        17
        18     def create(self, name=None, original_style=None):
```

```
      E:
↪\PycharmProjects\PycharmProjects\py2cytoscape\py2cytoscape\data\style_client.
↪py in __init__(self, url)
       75     def __init__(self, url):
       76         self.__url = url + 'styles/visualproperties'
  ---> 77         self.__convert_to_dict()
       78
       79     def __convert_to_dict(self):


      E:
↪\PycharmProjects\PycharmProjects\py2cytoscape\py2cytoscape\data\style_client.
↪py in __convert_to_dict(self)
       78
       79     def __convert_to_dict(self):
  ---> 80         vps = requests.get(self.__url).json()
       81         vp_dict = {}
       82         node_vps = []


      ~\miniconda3\envs\magine_37\lib\site-packages\requests\api.py in␣
↪get(url, params, **kwargs)
       73
       74     kwargs.setdefault('allow_redirects', True)
  ---> 75     return request('get', url, params=params, **kwargs)
       76
       77


      ~\miniconda3\envs\magine_37\lib\site-packages\requests\api.py in␣
↪request(method, url, **kwargs)
       58     # cases, and look like a memory leak in others.
       59     with sessions.Session() as session:
  ---> 60         return session.request(method=method, url=url, **kwargs)
       61
       62


      ~\miniconda3\envs\magine_37\lib\site-packages\requests\sessions.py in␣
↪request(self, method, url, params, data, headers, cookies, files, auth,␣
↪timeout, allow_redirects, proxies, hooks, stream, verify, cert, json)
      531         }
      532         send_kwargs.update(settings)
  --> 533         resp = self.send(prep, **send_kwargs)
      534
      535         return resp
```

```
    ~\miniconda3\envs\magine_37\lib\site-packages\requests\sessions.py in␣
↪send(self, request, **kwargs)
    644
    645         # Send the request
--> 646         r = adapter.send(request, **kwargs)
    647
    648         # Total elapsed time of the request (approximately)


    ~\miniconda3\envs\magine_37\lib\site-packages\requests\adapters.py in␣
↪send(self, request, stream, timeout, verify, cert, proxies)
    514                 raise SSLError(e, request=request)
    515
--> 516             raise ConnectionError(e, request=request)
    517
    518         except ClosedPoolError as e:


    ConnectionError: HTTPConnectionPool(host='localhost', port=1234): Max␣
↪retries exceeded with url: /v1/styles/visualproperties (Caused by␣
↪NewConnectionError('<urllib3.connection.HTTPConnection object at␣
↪0x000001D71B544710>: Failed to establish a new connection: [WinError 10061] No␣
↪connection could be made because the target machine actively refused it'))
```

[26]:

```python
term_net, mol_net = create_subnetwork(
    ph_silac,
    network=network,
    save_name='ph_silac_ags',
    use_cytoscape=False,
    use_fdr=True, use_threshold=True, min_edges=20,
    scale=200
)
create_gif(prefix, tps, 'scale_200')
```

```
Creating ontology network
```

[26]:

```
<IPython.core.display.Image object>
```

[ ]:

```python
term_net, mol_net = create_subnetwork(
    ph_silac,
    network=network,
    save_name='ph_silac_ags',
    use_cytoscape=True,
    use_fdr=True, use_threshold=True, min_edges=20,
    scale=300
)
```

```
create_gif(prefix, tps, 'scale_300')
```

[27]:

```
vis.draw_cyjs(term_net, default_color='white', layout='concentric',␣
 ↪spacingFactor=2.8)
```

```
<IPython.core.display.HTML object>
```

3.1.3 Known mechanisms of bendamustine

Next, we will demonstrate how to explore known mechanisms. Bendamustine is known to cause dna damage, cell cycle arrest, and apoptosis, so we will start there.

[ ]:

```
selected_terms = ['dna repair', 'cell cycle', 'apoptosis']

# extract out lists of genes for each term
g_sets = [ph_silac.sig.term_to_genes(i) for i in selected_terms]

# visualize the overlap
create_venn3(*g_sets+selected_terms, save_name='venn_canonical');
```

[ ]:

```
def print_numbers(term_name):
    genes = reactome_only.sig.term_to_genes(term_name)
    n_sig = len(genes)
    if n_sig:
        n_sig_ptms = len(exp_data.subset(genes).sig.label_list)
        print("{} : {} : {}".format(term_name, n_sig, n_sig_ptms))
print_numbers('cell cycle')
print_numbers('dna repair')
print_numbers('apoptosis')
```

[ ]:

```
# though we focused on ph-silac originally, we can see if the terms of interest␣
 ↪are in other datasets
subset = reactome_only.loc[reactome_only.term_name.isin(selected_terms)].copy()

subset.filter_multi(
    category=['ph_silac_down', 'ph_silac_up',
              'label_free_down', 'label_free_up',
              'silac_up', 'silac_down',
              'rna_up', 'rna_down'],
    inplace=True,
)

# remove time points and samples that don't contain a significant term
subset.require_n_sig(
    n_sig=1,
    index=['category', 'sample_id'],
    columns='term_name',
    inplace=True
)
subset.require_n_sig(
    n_sig=1,
    index=['sample_id', 'category'],
    columns='term_name',
    inplace=True
)

fig = subset.heatmap(
    convert_to_log=False,
    cluster_by_set=False,
    annotate_sig=True,
    columns=['category', 'sample_id'],
    div_colors=True,
    linewidths=.005,
    figsize=(12, 16)
);
```

[ ]:

```
# subset the data to only include these terms
subset = reactome_only.sig.loc[reactome_only.sig.term_name.
 ↪isin(selected_terms)].copy()

term_net, mol_net = create_subnetwork(
    subset,
    network=network,
    save_name='apop_dna_cell_cycle',
    use_cytoscape=False,
    use_fdr=True,
    use_threshold=True,
    min_edges=10
)
print(len(mol_net.nodes))
print(len(mol_net.edges))
```

[ ]:

```
nx.set_node_attributes(term_net, 'white', 'color')
vis.draw_cyjs(term_net, layout='concentric', spacingFactor=2.8)
```

DNA damage response

[ ]:

```
# extract dna repair genes
dna_repair_genes = reactome_only.sig.term_to_genes('dna repair')


exp_data.ph_silac.heatmap(
    dna_repair_genes,
    convert_to_log=True,
    subset_index='identifier',
    index='label',
    cluster_row=True,
    rank_index=True,
    min_sig=2,
    num_colors=13,
    linewidths=0.01
);


exp_data.label_free.heatmap(
    dna_repair_genes,
    convert_to_log=True,
    subset_index='identifier',
    index='label',
    cluster_row=True,
    rank_index=True,
    min_sig=2,
    num_colors=13,
    linewidths=0.01
);


exp_data.rna_seq.heatmap(
    dna_repair_genes,
    convert_to_log=True,
    subset_index='identifier',
    index='label',
    cluster_row=True,
    rank_index=True,
    min_sig=2,
    num_colors=13,
    linewidths=0.01
);
```

[ ]:

```
# look at only first time point
first_tp = exp_data.species.subset(dna_repair_genes, sample_ids=['000030_s']).
 ↪sig

first_tp.heatmap(figsize=(3,6), index='label', linewidths=0.01);
```

[ ]:

```
# Create a network based on the first time point, include dna damage nodes
dna_repair_subnet = net_sub.expand_neighbors(
    nodes=first_tp.id_list,
    upstream=False,
    downstream=True,
    add_interconnecting_edges=True,
    max_dist=1,
    include_only=dna_repair_genes
)
```

```
# removes disconnected nodes in network
dna_repair_subnet = utils.delete_disconnected_network(dna_repair_subnet)

colors = dict()
for i in dna_repair_subnet.nodes:
    if i in first_tp.id_list:
        colors[i] = 'lightgreen'
    else:
        colors[i] = 'lightblue'
nx.set_node_attributes(dna_repair_subnet, colors, 'color')
```

[ ]:

```
vis.draw_cyjs(dna_repair_subnet, layout='cose-bilkent', spacingFactor=2.)
```

[ ]:

```
vis.draw_cyjs(dna_repair_subnet, layout='concentric', spacingFactor=1.)
```

Cell cycle

[ ]:

```
cell_cycle_genes = reactome_only.sig.term_to_genes('cell cycle')



exp_data.ph_silac.heatmap(
    cell_cycle_genes,
    subset_index='identifier',
    index='label',
    cluster_row=True,
    rank_index=True,
    min_sig=2,
    num_colors=13,
    linewidths=0.01,
    figsize=(6, 20),
    y_tick_labels=True
);

exp_data.label_free.heatmap(
    cell_cycle_genes,
    subset_index='identifier',
    index='label',
    cluster_row=True,
    rank_index=True,
    min_sig=2,
    num_colors=13,
    linewidths=0.01,
    figsize=(6,16)
);

exp_data.rna_seq.heatmap(
    cell_cycle_genes,
    subset_index='identifier',
    index='label',
    cluster_row=True,
    rank_index=True,
    min_sig=1,
    num_colors=13,
    linewidths=0.01,
    figsize=(6,10)
);
```

[ ]:

```
apoptosis_genes = reactome_only.sig.term_to_genes('apoptosis')

exp_data.ph_silac.heatmap(
    apoptosis_genes,
    subset_index='identifier',
    index='label',
    cluster_row=False,
    rank_index=True,
    min_sig=1,
    num_colors=13,
    linewidths=0.01,
    figsize=(6,16)
);


exp_data.label_free.heatmap(
    apoptosis_genes,
    subset_index='identifier',
    index='label',
    cluster_row=False,
    rank_index=True,
    min_sig=1,
    num_colors=13,
    linewidths=0.01,
    figsize=(6,16)
);
exp_data.rna_seq.heatmap(
    apoptosis_genes,
    subset_index='identifier',
    index='label',
    cluster_row=True,
    rank_index=True,
    min_sig=1,
    num_colors=13,
    linewidths=0.01,
    figsize=(6,10)
);
```

[ ]:

```
print("DNA repair")
for i in exp_data.exp_methods:
    print("\t", i, len(exp_data[i].subset(dna_repair_genes).sig.id_list),
          len(exp_data[i].subset(dna_repair_genes).sig.label_list))
print("Cell cycle")
for i in exp_data.exp_methods:
    print("\t", i, len(exp_data[i].subset(cell_cycle_genes).sig.id_list),
          len(exp_data[i].subset(cell_cycle_genes).sig.label_list))
print("Apoptosis")
for i in exp_data.exp_methods:
    print("\t", i, len(exp_data[i].subset(apoptosis_genes).sig.id_list),
          len(exp_data[i].subset(apoptosis_genes).sig.label_list))
```

[ ]:

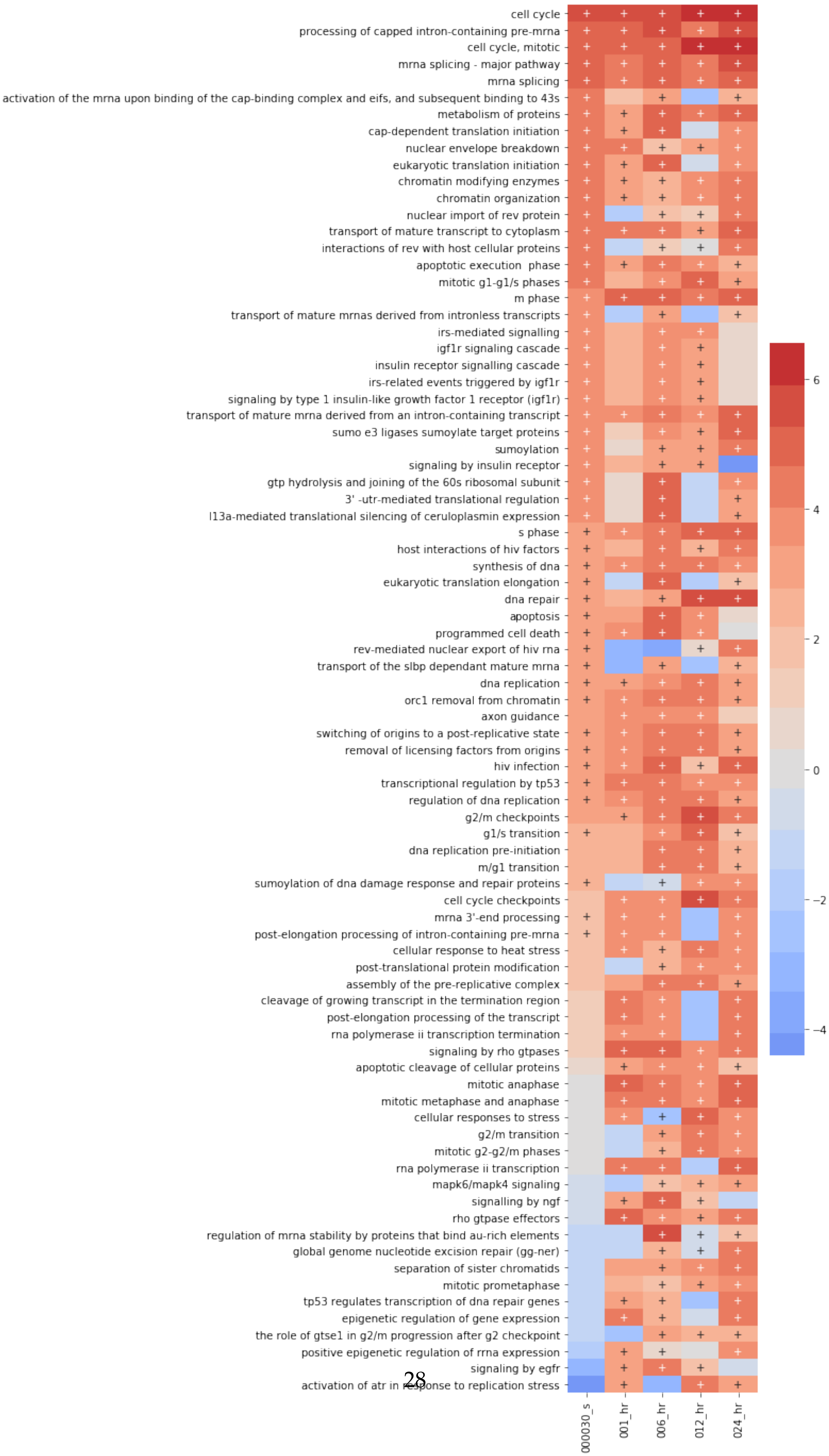
```
# create plots grouping together data for each platform

genes_in_labels = utils.create_dict_from_node_attributes(mol_net, 'termName')

# phosph-silac
heatmap_by_terms(
    exp_data.ph_silac,
    convert_to_log=True,
    index='label',
    term_labels=list(genes_in_labels.keys()),
    term_sets=list(genes_in_labels.values()),
    div_colors=True,
    linewidths=0.01,
    min_sig=3,
    annotate_sig=True,
    cluster_col=False,
    cluster_row=False,
    y_tick_labels=True,
    figsize=(6, 12)
);

# label free
heatmap_by_terms(
    exp_data.label_free,
    convert_to_log=True,
    index='label',
    term_labels=list(genes_in_labels.keys()),
    term_sets=list(genes_in_labels.values()),
    div_colors=True,
    linewidths=0.01,
    min_sig=3,
    annotate_sig=True,
    cluster_col=False,
    cluster_row=False,
    y_tick_labels=True,
    figsize=(8, 12)
);

# silac
heatmap_by_terms(
    exp_data.silac,
    convert_to_log=True,
    index='label',
    term_labels=list(genes_in_labels.keys()),
    term_sets=list(genes_in_labels.values()),
    div_colors=True,
    linewidths=0.01,
    min_sig=2,
    annotate_sig=True,
    cluster_col=False,
    cluster_row=False,
    y_tick_labels=True,
    figsize=(4, 4)
);



# rna
heatmap_by_terms(
    exp_data.rna_seq,
    convert_to_log=True,
    index='label',
    term_labels=list(genes_in_labels.keys()),
    term_sets=list(genes_in_labels.values()),
    div_colors=True,
    linewidths=0.01,
    min_sig=1,
    annotate_sig=True,
    cluster_col=False,
    cluster_row=False,
    y_tick_labels=True,
    figsize=(5, 10)
);
```

3.1.4 Lets look at only up-regulated ph-silac

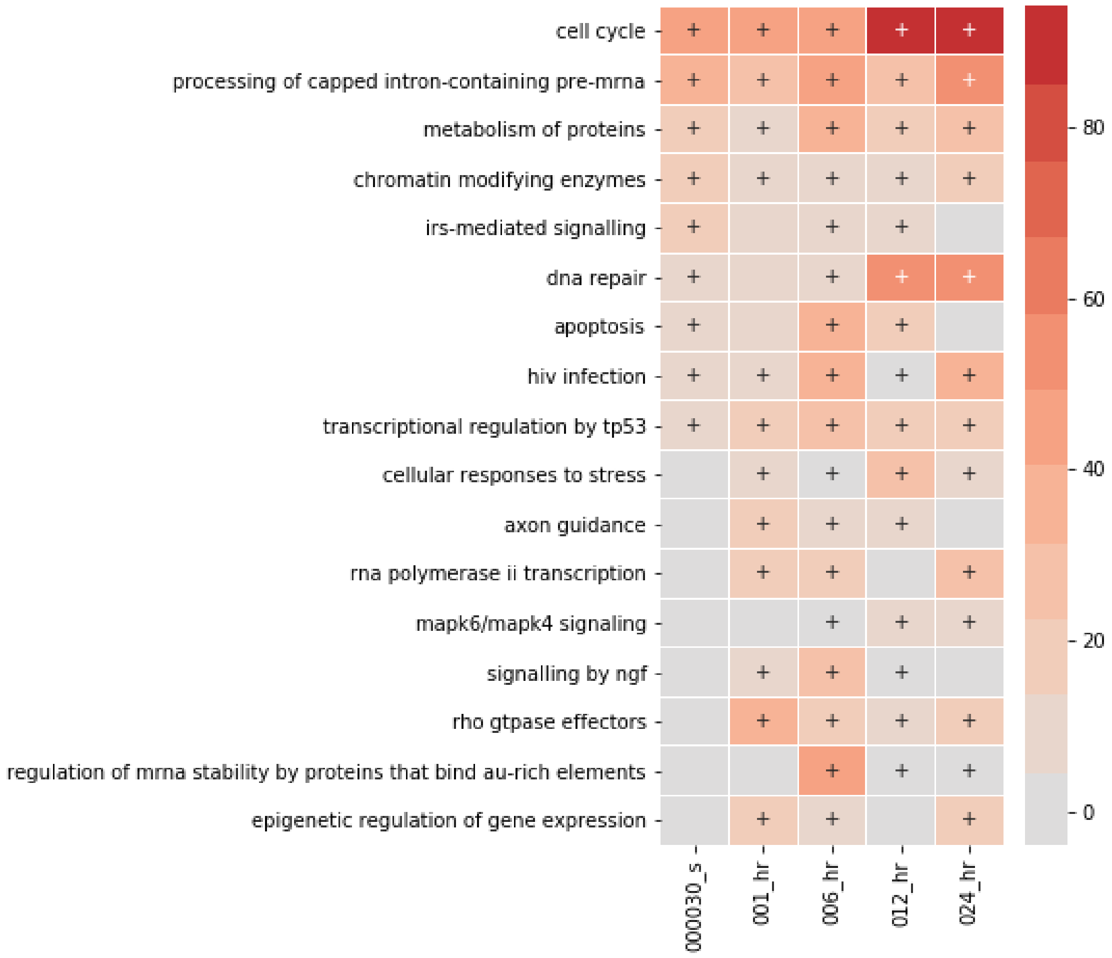
[22]:

```
ph_silac_up = reactome_only.filter_multi(category=['ph_silac_up'])

not_useful = [
    'gene expression', 'translation',
    'immune system',
    'disease', 'diseases of signal transduction',
    'infectious disease',
    'influenza infection', 'influenza life cycle',
    'influenza viral rna transcription and replication',
]

ph_silac_up = ph_silac_up.loc[~ph_silac_up['term_name'].isin(not_useful)]

print("Number of terms before filtering base on minimum time points : {}\
      ".format(len(ph_silac_up.term_name.unique())))

ph_silac_up.require_n_sig(
    index='term_name',
    columns='sample_id',
    n_sig=3,
    inplace=True
)
ph_silac_up_copy = ph_silac_up.copy()
print("Number of terms after filtering base on minimum time points : {}\
      ".format(len(ph_silac_up.term_name.unique())))

fig = ph_silac_up_copy.heatmap(figsize=(4, 24));
fig.savefig('ph_silac.png', dpi=300, bbox_inches='tight')

ph_silac_up.remove_redundant(
    threshold=.7,
    level='sample',
    inplace=True,
    sort_by='combined_score'
)

ph_silac_up.remove_redundant(
    threshold=.7,
    level='dataframe',
    inplace=True,
    sort_by='combined_score'
)
print("Number of genes before : {}".format(len(ph_silac_up_copy.
 ↪all_genes_from_df())))
print("Number of genes after : {}".format(len(ph_silac_up.all_genes_from_df())))
```

```
Number of terms before filtering base on minimum time points : 560
Number of terms after filtering base on minimum time points : 83
Number of rows went from 83 to 30
Number of rows went from 30 to 17
Number of genes before : 570
Number of genes after : 550
```

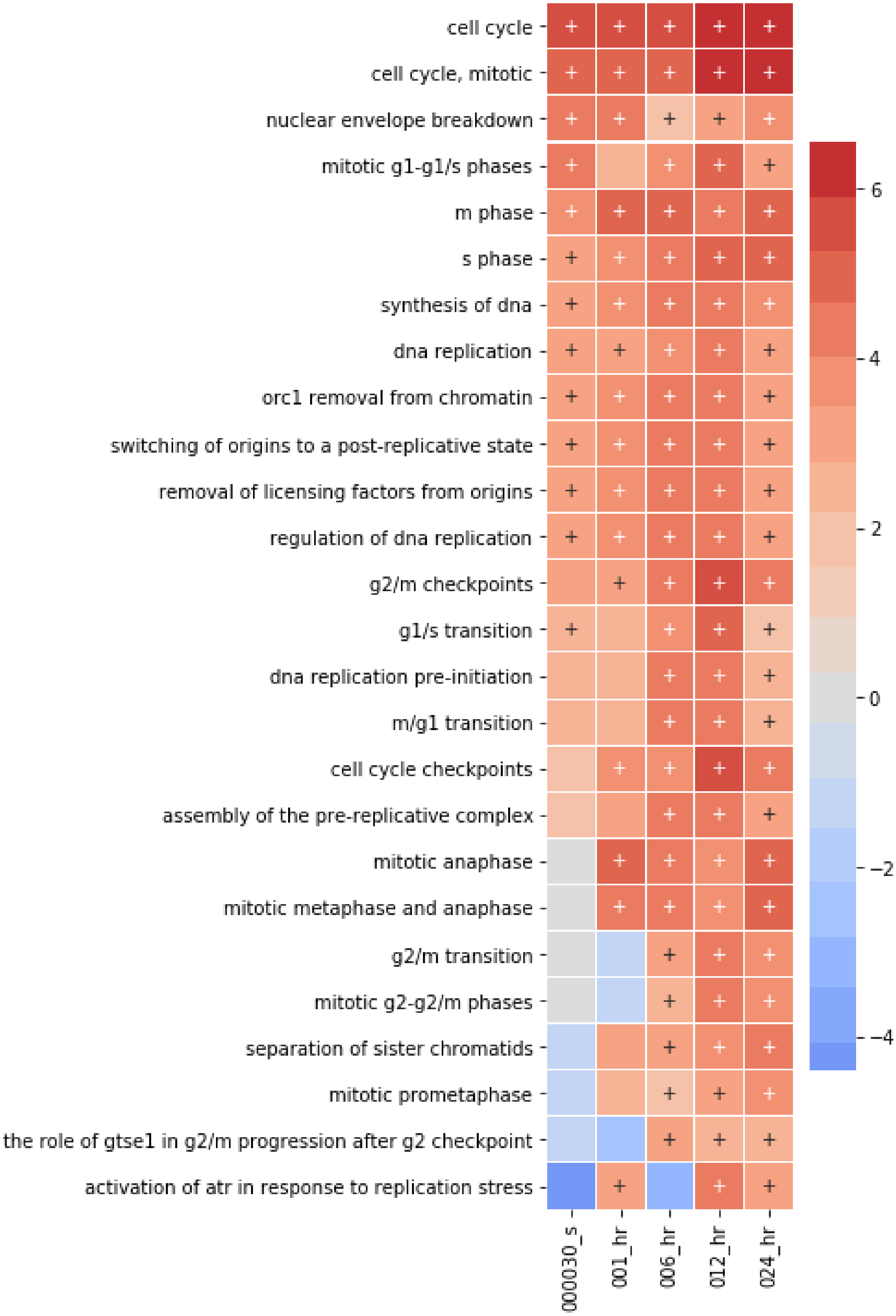

cell cycle
processing of capped intron-containing pre-mrna
cell cycle, mitotic
mrna splicing - major pathway
mrna splicing
activation of the mrna upon binding of the cap-binding complex and eifs, and subsequent binding to 43s
metabolism of proteins
cap-dependent translation initiation
nuclear envelope breakdown
eukaryotic translation initiation
chromatin modifying enzymes
chromatin organization
nuclear import of rev protein
transport of mature transcript to cytoplasm
interactions of rev with host cellular proteins
apoptotic execution phase
mitotic g1-g1/s phases
m phase
transport of mature mrnas derived from intronless transcripts
irs-mediated signalling
igf1r signaling cascade
insulin receptor signalling cascade
irs-related events triggered by igf1r
signaling by type 1 insulin-like growth factor 1 receptor (igf1r)
transport of mature mrna derived from an intron-containing transcript
sumo e3 ligases sumoylate target proteins
sumoylation
signaling by insulin receptor
gtp hydrolysis and joining of the 60s ribosomal subunit
3' -utr-mediated translational regulation
l13a-mediated translational silencing of ceruloplasmin expression
s phase
host interactions of hiv factors
synthesis of dna
eukaryotic translation elongation
dna repair
apoptosis
programmed cell death
rev-mediated nuclear export of hiv rna
transport of the slbp dependant mature mrna
dna replication
orc1 removal from chromatin
axon guidance
switching of origins to a post-replicative state
removal of licensing factors from origins
hiv infection
transcriptional regulation by tp53
regulation of dna replication
g2/m checkpoints
g1/s transition
dna replication pre-initiation
m/g1 transition
sumoylation of dna damage response and repair proteins
cell cycle checkpoints
mrna 3'-end processing
post-elongation processing of intron-containing pre-mrna
cellular response to heat stress
post-translational protein modification
assembly of the pre-replicative complex
cleavage of growing transcript in the termination region
post-elongation processing of the transcript
rna polymerase ii transcription termination
signaling by rho gtpases
apoptotic cleavage of cellular proteins
mitotic anaphase
mitotic metaphase and anaphase
cellular responses to stress
g2/m transition
mitotic g2-g2/m phases
rna polymerase ii transcription
mapk6/mapk4 signaling
signalling by ngf
rho gtpase effectors
regulation of mrna stability by proteins that bind au-rich elements
global genome nucleotide excision repair (gg-ner)
separation of sister chromatids
mitotic prometaphase
tp53 regulates transcription of dna repair genes
epigenetic regulation of gene expression
the role of gtse1 in g2/m progression after g2 checkpoint
positive epigenetic regulation of rrna expression
signaling by egfr
activation of atr in response to replication stress
000030_s
001_hr
006_hr
012_hr
024_hr
6
4
2
0
−2
−4

[23]:
```
fig = ph_silac_up.heatmap(
    convert_to_log=False,
    cluster_by_set=False,
    cluster_row=False,
    values='combined_score',
    annotate_sig=True,
    div_colors=True,
    linewidths=.005,
    figsize=(4,8)
);
fig.savefig('ph_silac_slimmed.png', dpi=300, bbox_inches='tight')
```



[24]:
```
# note that we can still get back terms that were compressed
ph_silac_up_copy.show_terms_below(
    'cell cycle',
    threshold=.7,
    remove_subset=True
).heatmap(
    figsize=(3, 12),
    cluster_by_set=False,
    linewidths=0.01,
);
```

```
Number of rows went from 83 to 17
```


cell cycle
cell cycle, mitotic
nuclear envelope breakdown
mitotic g1-g1/s phases
m phase
s phase
synthesis of dna
dna replication
orc1 removal from chromatin
switching of origins to a post-replicative state
removal of licensing factors from origins
regulation of dna replication
g2/m checkpoints
g1/s transition
dna replication pre-initiation
m/g1 transition
cell cycle checkpoints
assembly of the pre-replicative complex
mitotic anaphase
mitotic metaphase and anaphase
g2/m transition
mitotic g2-g2/m phases
separation of sister chromatids
mitotic prometaphase
the role of gtse1 in g2/m progression after g2 checkpoint
activation of atr in response to replication stress
000030_s
001_hr
006_hr
012_hr
024_hr
6
4
2
0
−2
−4

[25]:
```
term_net, mol_net = create_subnetwork(
    ph_silac_up,
    network=network,
    save_name='time_series_agn',
    use_cytoscape=True,
    use_fdr=True, use_threshold=True, min_edges=25, scale=150
)
for i in ph_silac_up.term_name.unique():
    if i not in term_net.nodes:
        print(i)
```

```
Creating ontology network


      ␣
↪---------------------------------------------------------------------------

       ConnectionRefusedError                    Traceback (most recent call␣
↪last)

       ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\connection.py in␣
↪_new_conn(self)
       158             conn = connection.create_connection(
   --> 159                 (self._dns_host, self.port), self.timeout,␣
↪**extra_kw)
       160


       ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\util\connection.py␣
↪in create_connection(address, timeout, source_address, socket_options)
        79     if err is not None:
   ---> 80         raise err
        81


       ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\util\connection.py␣
↪in create_connection(address, timeout, source_address, socket_options)
        69                 sock.bind(source_address)
   ---> 70             sock.connect(sa)
        71             return sock


       ConnectionRefusedError: [WinError 10061] No connection could be made␣
↪because the target machine actively refused it


   During handling of the above exception, another exception occurred:
       NewConnectionError                        Traceback (most recent call␣
↪last)

       ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\connectionpool.py␣
↪in urlopen(self, method, url, body, headers, retries, redirect,␣
↪assert_same_host, timeout, pool_timeout, release_conn, chunked, body_pos,␣
↪**response_kw)
       599                                                   body=body,␣
↪headers=headers,
   --> 600                                                   chunked=chunked)
       601


       ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\connectionpool.py␣
↪in _make_request(self, conn, method, url, timeout, chunked,␣
↪**httplib_request_kw)
       353             else:
   --> 354                 conn.request(method, url, **httplib_request_kw)
       355


       ~\miniconda3\envs\magine_37\lib\http\client.py in request(self, method,␣
↪url, body, headers, encode_chunked)
      1228             """Send a complete request to the server."""
   -> 1229             self._send_request(method, url, body, headers,␣
↪encode_chunked)
      1230


       ~\miniconda3\envs\magine_37\lib\http\client.py in _send_request(self,␣
↪method, url, body, headers, encode_chunked)
      1274                 body = _encode(body, 'body')
   -> 1275             self.endheaders(body, encode_chunked=encode_chunked)
      1276


       ~\miniconda3\envs\magine_37\lib\http\client.py in endheaders(self,␣
↪message_body, encode_chunked)
      1223                 raise CannotSendHeader()
   -> 1224             self._send_output(message_body,␣
↪encode_chunked=encode_chunked)
      1225
```

```
    ~\miniconda3\envs\magine_37\lib\http\client.py in _send_output(self,␣
↪message_body, encode_chunked)
   1015         del self._buffer[:]
-> 1016         self.send(msg)
   1017


    ~\miniconda3\envs\magine_37\lib\http\client.py in send(self, data)
    955             if self.auto_open:
--> 956                 self.connect()
    957             else:


    ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\connection.py in␣
↪connect(self)
    180     def connect(self):
--> 181         conn = self._new_conn()
    182         self._prepare_conn(conn)


    ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\connection.py in␣
↪_new_conn(self)
    167             raise NewConnectionError(
--> 168                 self, "Failed to establish a new connection: %s" % e)
    169


    NewConnectionError: <urllib3.connection.HTTPConnection object at␣
↪0x000001D71B658898>: Failed to establish a new connection: [WinError 10061] No␣
↪connection could be made because the target machine actively refused it


During handling of the above exception, another exception occurred:


    MaxRetryError                             Traceback (most recent call␣
↪last)

    ~\miniconda3\envs\magine_37\lib\site-packages\requests\adapters.py in␣
↪send(self, request, stream, timeout, verify, cert, proxies)
    448                     retries=self.max_retries,
--> 449                     timeout=timeout
    450                 )
```

```
       ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\connectionpool.py␣
↪in urlopen(self, method, url, body, headers, retries, redirect,␣
↪assert_same_host, timeout, pool_timeout, release_conn, chunked, body_pos,␣
↪**response_kw)
       637             retries = retries.increment(method, url, error=e,␣
↪_pool=self,
   --> 638                                         _stacktrace=sys.
↪exc_info()[2])
       639             retries.sleep()


       ~\miniconda3\envs\magine_37\lib\site-packages\urllib3\util\retry.py in␣
↪increment(self, method, url, response, error, _pool, _stacktrace)
       398         if new_retry.is_exhausted():
   --> 399             raise MaxRetryError(_pool, url, error or␣
↪ResponseError(cause))
       400


       MaxRetryError: HTTPConnectionPool(host='localhost', port=1234): Max␣
↪retries exceeded with url: /v1/styles/visualproperties (Caused by␣
↪NewConnectionError('<urllib3.connection.HTTPConnection object at␣
↪0x000001D71B658898>: Failed to establish a new connection: [WinError 10061] No␣
↪connection could be made because the target machine actively refused it'))


   During handling of the above exception, another exception occurred:


       ConnectionError                           Traceback (most recent call␣
↪last)

       <ipython-input-25-3897a1d095d0> in <module>
         4     save_name='time_series_agn',
         5     use_cytoscape=True,
   ----> 6     use_fdr=True, use_threshold=True, min_edges=25, scale=150
         7 )
         8 for i in ph_silac_up.term_name.unique():


       E:\PycharmProjects\PycharmProjects\Magine\magine\networks\annotated_set.
↪py in create_subnetwork(df, network, terms, save_name, draw_png,␣
↪remove_isolated, use_cytoscape, merge, out_dir, use_threshold, use_fdr,␣
↪min_edges, scale)
       310     if use_cytoscape:
       311         from magine.networks.visualization.cytoscape import␣
↪RenderModel
```

```
  --> 312         rm = RenderModel(term_g, layout='force-directed')
      313         rm.visualize_by_list_of_time(labels,
      314                                      prefix=save_name,


      E:
↪\PycharmProjects\PycharmProjects\Magine\magine\networks\visualization\cytoscape.
↪py in __init__(self, graph, layout, style)
       76
       77         self.graph = graph
  ---> 78         self.cy = CyRestClient()
       79         self.cy.session.delete()
       80         self.cy.layout2 = LayoutClient()


      E:
↪\PycharmProjects\PycharmProjects\py2cytoscape\py2cytoscape\data\cyrest_client.
↪py in __init__(self, ip, port, version)
       17
       18         self.network = NetworkClient(self.__url)
  ---> 19         self.style = StyleClient(self.__url)
       20         self.layout = LayoutClient(self.__url)
       21         self.edgebundling = EdgeBundlingClient(self.__url)


      E:
↪\PycharmProjects\PycharmProjects\py2cytoscape\py2cytoscape\data\style_client.
↪py in __init__(self, url)
       14         self.__url_apply = url + 'apply/styles/'
       15
  ---> 16         self.vps = VisualProperties(url)
       17
       18     def create(self, name=None, original_style=None):


      E:
↪\PycharmProjects\PycharmProjects\py2cytoscape\py2cytoscape\data\style_client.
↪py in __init__(self, url)
       75     def __init__(self, url):
       76         self.__url = url + 'styles/visualproperties'
  ---> 77         self.__convert_to_dict()
       78
       79     def __convert_to_dict(self):
```

```
      E:
↪\PycharmProjects\PycharmProjects\py2cytoscape\py2cytoscape\data\style_client.
↪py in __convert_to_dict(self)
       78
       79     def __convert_to_dict(self):
  ---> 80         vps = requests.get(self.__url).json()
       81         vp_dict = {}
       82         node_vps = []


      ~\miniconda3\envs\magine_37\lib\site-packages\requests\api.py in␣
↪get(url, params, **kwargs)
       73
       74     kwargs.setdefault('allow_redirects', True)
  ---> 75     return request('get', url, params=params, **kwargs)
       76
       77


      ~\miniconda3\envs\magine_37\lib\site-packages\requests\api.py in␣
↪request(method, url, **kwargs)
       58     # cases, and look like a memory leak in others.
       59     with sessions.Session() as session:
  ---> 60         return session.request(method=method, url=url, **kwargs)
       61
       62


      ~\miniconda3\envs\magine_37\lib\site-packages\requests\sessions.py in␣
↪request(self, method, url, params, data, headers, cookies, files, auth,␣
↪timeout, allow_redirects, proxies, hooks, stream, verify, cert, json)
      531         }
      532         send_kwargs.update(settings)
  --> 533         resp = self.send(prep, **send_kwargs)
      534
      535         return resp


      ~\miniconda3\envs\magine_37\lib\site-packages\requests\sessions.py in␣
↪send(self, request, **kwargs)
      644
      645         # Send the request
  --> 646         r = adapter.send(request, **kwargs)
      647
      648         # Total elapsed time of the request (approximately)
```

```
       ~\miniconda3\envs\magine_37\lib\site-packages\requests\adapters.py in␣
↪send(self, request, stream, timeout, verify, cert, proxies)
       514                     raise SSLError(e, request=request)
       515
   --> 516                 raise ConnectionError(e, request=request)
       517
       518             except ClosedPoolError as e:


       ConnectionError: HTTPConnectionPool(host='localhost', port=1234): Max␣
↪retries exceeded with url: /v1/styles/visualproperties (Caused by␣
↪NewConnectionError('<urllib3.connection.HTTPConnection object at␣
↪0x000001D71B658898>: Failed to establish a new connection: [WinError 10061] No␣
↪connection could be made because the target machine actively refused it'))
```

[21]:

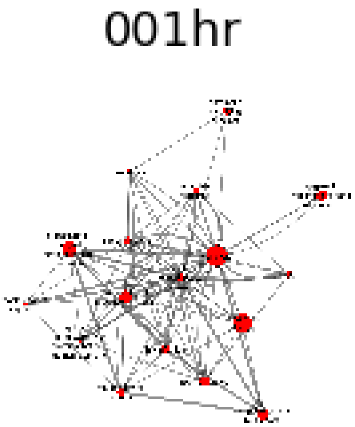
```
create_gif('time_series_agn', tps, 'time_series_agn')
```

```
      ␣
↪---------------------------------------------------------------------------

       KeyboardInterrupt                         Traceback (most recent call␣
↪last)

       <ipython-input-21-d45f88c5bf26> in <module>
   ----> 1 create_gif('time_series_agn', tps, 'time_series_agn')


       <ipython-input-19-35fad6325ce4> in create_gif(prefix, timepoints,␣
↪save_name)
        44     f_names = ["{}_{}.png".format(prefix, i) for i in tps]
        45     for i,j in zip(f_names, tps):
   ---> 46         trip_photo(i, j)
        47
        48     kargs = {'duration': 8 / len(f_names)}


       <ipython-input-19-35fad6325ce4> in trip_photo(im_location, title)
        36     out = im_location.replace('.png', '_formatted.png')
        37     out2 = im_location.replace('.png', '_formatted.svg')
   ---> 38     plt.savefig(out, dpi=1000, bbox_inches='tight', transparent=True)
        39     plt.savefig(out2, bbox_inches='tight', transparent=True)
        40     plt.close()


       ~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\pyplot.py in␣
↪savefig(*args, **kwargs)
```

```
    714 def savefig(*args, **kwargs):
    715     fig = gcf()
--> 716     res = fig.savefig(*args, **kwargs)
    717     fig.canvas.draw_idle()   # need this if 'transparent=True' to␣
↪reset colors
    718     return res


    ~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\figure.py in␣
↪savefig(self, fname, transparent, **kwargs)
   2178             self.patch.set_visible(frameon)
   2179
-> 2180         self.canvas.print_figure(fname, **kwargs)
   2181
   2182         if frameon:


    ~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\backend_bases.
↪py in print_figure(self, filename, dpi, facecolor, edgecolor, orientation,␣
↪format, bbox_inches, **kwargs)
   2080                     orientation=orientation,
   2081                     bbox_inches_restore=_bbox_inches_restore,
-> 2082                     **kwargs)
   2083             finally:
   2084                 if bbox_inches and restore_bbox:


      ␣
↪~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\backends\backend_agg.
↪py in print_png(self, filename_or_obj, metadata, pil_kwargs, *args, **kwargs)
    525
    526         else:
--> 527             FigureCanvasAgg.draw(self)
    528             renderer = self.get_renderer()
    529             with cbook._setattr_cm(renderer, dpi=self.figure.dpi), \


      ␣
↪~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\backends\backend_agg.
↪py in draw(self)
    386         self.renderer = self.get_renderer(cleared=True)
    387         with RendererAgg.lock:
--> 388             self.figure.draw(self.renderer)
    389             # A GUI class may be need to update a window using this␣
↪draw, so
    390             # don't forget to call the superclass.
```

```
       ~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\artist.py in␣
↪draw_wrapper(artist, renderer, *args, **kwargs)
        36                 renderer.start_filter()
        37 
   ---> 38             return draw(artist, renderer, *args, **kwargs)
        39         finally:
        40             if artist.get_agg_filter() is not None:


       ~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\figure.py in␣
↪draw(self, renderer)
      1707             self.patch.draw(renderer)
      1708             mimage._draw_list_compositing_images(
   -> 1709                 renderer, self, artists, self.suppressComposite)
      1710 
      1711             renderer.close_group('figure')


       ~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\image.py in␣
↪_draw_list_compositing_images(renderer, parent, artists, suppress_composite)
       133     if not_composite or not has_images:
       134         for a in artists:
   --> 135             a.draw(renderer)
       136     else:
       137         # Composite any adjacent images together


       ~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\artist.py in␣
↪draw_wrapper(artist, renderer, *args, **kwargs)
        36                 renderer.start_filter()
        37 
   ---> 38             return draw(artist, renderer, *args, **kwargs)
        39         finally:
        40             if artist.get_agg_filter() is not None:


       ~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\axes\_base.py␣
↪in draw(self, renderer, inframe)
      2643             renderer.stop_rasterizing()
      2644 
   -> 2645         mimage._draw_list_compositing_images(renderer, self, artists)
      2646 
      2647         renderer.close_group('axes')
```

```
    ~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\image.py in _draw_list_compositing_images(renderer, parent, artists, suppress_composite)
    133     if not_composite or not has_images:
    134         for a in artists:
--> 135             a.draw(renderer)
    136     else:
    137         # Composite any adjacent images together


    ~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\artist.py in draw_wrapper(artist, renderer, *args, **kwargs)
     36                 renderer.start_filter()
     37 
---> 38             return draw(artist, renderer, *args, **kwargs)
     39         finally:
     40             if artist.get_agg_filter() is not None:


    ~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\image.py in draw(self, renderer, *args, **kwargs)
    617         else:
    618             im, l, b, trans = self.make_image(
--> 619                 renderer, renderer.get_image_magnification())
    620             if im is not None:
    621                 renderer.draw_image(gc, l, b, im)


    ~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\image.py in make_image(self, renderer, magnification, unsampled)
    872         return self._make_image(
    873             self._A, bbox, transformed_bbox, self.axes.bbox, magnification,
--> 874             unsampled=unsampled)
    875 
    876     def _check_unsampled_image(self, renderer):


    ~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\image.py in _make_image(self, A, in_bbox, out_bbox, clip_bbox, magnification, unsampled, round_to_pixel_border)
    505 
    506                 #resample rgb channels
--> 507                 A = _rgb_to_rgba(A[..., :3])
    508                 _image.resample(
    509                     A, output, t,
~\miniconda3\envs\magine_37\lib\site-packages\matplotlib\image.py in _rgb_to_rgba(A)
    167     """
    168     rgba = np.zeros((A.shape[0], A.shape[1], 4), dtype=A.dtype)
--> 169     rgba[:, :, :3] = A
    170     if rgba.dtype == np.uint8:
    171         rgba[:, :, 3] = 255


KeyboardInterrupt:
```



[ ]:
```
nx.set_node_attributes(term_net, 'red', 'color')
vis.draw_cyjs(term_net, layout='cose-bilkent', spacingFactor=1.4)
```

3.1.5 Exploring terms outside canonical

Since enrichment analysis is performed over all terms in a gene set, some of the terms might not make sense in certain context. This could be due to genes being classified in multiple sets, bad assignment, or possible new biology. The time it takes to explore each term in understanding `why` it is enriched can be time consuming, which is perhaps why people disregard terms frequently. We decided to use MAGINE to explore one of these terms. `HIV infection`

[ ]:
```
reactome_only.sig.show_terms_below(
    'hiv infection',
    level='dataframe',
    threshold=.25,
    remove_subset=False
).heatmap(
    figsize=(6,16),
    linewidths=0.05,# cluster_by_set=True,
);
```

[ ]:

```
reactome_only.sig.find_similar_terms('hiv infection', level='dataframe',␣
 ↪remove_subset=False )
```

Since HIV infection hijacks dna repair and regulates cell cycle, lets check to see if that explains why HIV infection is enriched.

[ ]:

```
hits = [
    'cell cycle',
    'dna repair',
    'hiv infection'
]
subset = reactome_only.sig.loc[reactome_only.sig.term_name.isin(hits)].copy()

g_sets = [reactome_only.sig.term_to_genes(i) for i in hits]
create_venn3(*g_sets+hits);
plt.savefig('venn_hiv.png', dpi=450)
```

[ ]:

```
hiv_only= g_sets[-1]
hiv_only.difference_update(g_sets[0]) # remove dna repair genes
hiv_only.difference_update(g_sets[1]) # removes cell cycle genes

exp_data.label_free.heatmap(
    hiv_only,
    cluster_row=True,
    subset_index='identifier',
    index='label',
    min_sig=1,
    linewidths=0.01
);
exp_data.ph_silac.heatmap(
    hiv_only,
    subset_index='identifier',
    index='label',
    cluster_row=True,
    rank_index=True,
    min_sig=1,
    figsize=(6, 16),
    linewidths=0.01
);
```

[ ]:

```
term_net, mol_net = create_subnetwork(
    subset,
    network=network,
    save_name='hiv',
    use_cytoscape=False,
    use_fdr=True, use_threshold=True, min_edges=10
)
print(len(mol_net.edges))
print(len(mol_net.nodes))
```

[ ]:

```
vis.draw_cyjs(term_net)
```

[ ]:

```
term_to_gene = subset.term_to_genes_dict()

heatmap_by_terms(
    exp_data.label_free,
    convert_to_log=False,
    index='label',
    term_labels=list(term_to_gene.keys()),
    term_sets=list(term_to_gene.values()),
    div_colors=True,
    linewidths=0.01,
    min_sig=2,
    annotate_sig=True,
    cluster_col=False,
    figsize=(6,12),
    y_tick_labels=True
);
heatmap_by_terms(
    exp_data.ph_silac,
    convert_to_log=False,
    index='label',
    term_labels=list(term_to_gene.keys()),
    term_sets=list(term_to_gene.values()),
    div_colors=True,
    linewidths=0.01,
    min_sig=2,
    annotate_sig=True,
    cluster_col=False,
    figsize=(6,12),
    y_tick_labels=True
);
```

3.1.6 Exploring a subset of terms based on a keyword

[ ]:

```
# Filter by terms
damage_terms = enrichment_array.sig.filter_based_on_words(['damage'])
print(len(damage_terms.all_genes_from_df()))
damage_terms.head(10)
damage_terms_all = damage_terms.filter_multi(category='ph_silac_both')
first_damage = damage_terms_all.filter_multi(sample_id='000030_s')
display(first_damage[cols].head(20))

fig = first_damage.heatmap(
    convert_to_log=False,
    cluster_by_set=False,
    annotate_sig=True,
    div_colors=True,
    linewidths=.01,
    num_colors=21,
    figsize=(3,5)
);
fig.savefig('dna_damage_terms_30s.png', dpi=300, bbox_inches='tight')

damage_terms.require_n_sig(inplace=True, columns='sample_id', n_sig=2)
damage_terms.remove_redundant(inplace=True, threshold=.5, level='sample')
fig = damage_terms.heatmap(
    convert_to_log=False,
    cluster_by_set=False,
    annotate_sig=True,
    div_colors=True,
    linewidths=.01,
    num_colors=21,
    figsize=(4,8)
);
fig.savefig('dna_damage_terms_all.png', bbox_inches='tight')

dna_gene_df = exp_data.subset(first_damage.all_genes_from_df())
dna_gene_df = dna_gene_df.loc[dna_gene_df.sample_id.isin(['000030_s'])]

dna_gene_df.sig.heatmap(
    index='label',
    rank_index=False,
    convert_to_log=True,
    annotate_sig=True,
    div_colors=True,
    linewidths=.01,
    num_colors=21,
    figsize=(2,22)
);
```

3.1.7 Exploring species of interest

[ ]:

```
g2_m = ['CDK1', 'CCNB1']
cdk1_inhibitors = ['GADD45A', 'GADD45B', 'GADD45G', 'CDKN1A']

exp_data.species.heatmap(
    g2_m,
    index='label',
    subset_index='identifier',
    figsize=(4,8),
    linewidths=0.01,
    cluster_row=True,
    min_sig=1
);

exp_data.species.heatmap(
    cdk1_inhibitors,
    index='label',
    subset_index='identifier',
    figsize=(4, 4),
    linewidths=0.01,
    cluster_row=True,
    min_sig=1
);
```

[ ]:

```
expand_neigh = net_sub.expand_neighbors(
    network=None,
    nodes=g2_m,
    downstream=True,
    upstream=True,
    max_dist=1,
    include_only=reactome_only.filter_multi(category=['label_free_up']).
 ↪term_to_genes('cell cycle'),
    add_interconnecting_edges=False,
)

expand_neigh = net_sub.paths_between_two_lists(
    reactome_only.filter_multi(category=['ph_silac_up']).term_to_genes('dna␣
 ↪repair'),
    g2_m,
    max_length=3,
    include_only=reactome_only.filter_multi(category=['label_free_up']).
 ↪term_to_genes('cell cycle'),
    add_interconnecting_edges=True
)
print(len(expand_neigh.nodes))
print(len(expand_neigh.edges))
expand_neigh = utils.delete_disconnected_network(expand_neigh)
exp_data.label_free.heatmap(
    expand_neigh.nodes,
    subset_index='identifier',
    index='label',
    rank_index=False,
    cluster_row=True,
    min_sig=3,
    annotate_sig=True,
    linewidths=0.01
);
```

[ ]:

```
#expand_neigh = utils.add_attribute_to_network(expand_neigh, g2_m, 'color', 'red', 'white')
vis.draw_cyjs(
    expand_neigh, layout='dagre',
    spacingFactor=1.,
    nodeRepulsion=100, gravity=.1,
    rankDir='TB', nodeSep=5, rankSep=25, ranker='longest-path'
)
```

[ ]:

```
expand_neigh = net_sub.paths_between_two_lists(
    reactome_only.filter_multi(category=['ph_silac_up']).term_to_genes('dna repair'),
    g2_m,
    reverse=True,
    max_length=3,
    include_only=exp_data.species.sig,
    add_interconnecting_edges=False
)

expand_neigh = utils.delete_disconnected_network(expand_neigh)

print(len(expand_neigh.nodes))
print(len(expand_neigh.edges))
```

[ ]:

```
vis.draw_cyjs(
    expand_neigh, layout='dagre',
    spacingFactor=1.,
    nodeRepulsion=100, gravity=.1,
    rankDir='TB', nodeSep=5, rankSep=25, ranker='longest-path'
)
```

[ ]:

```
exp_data.ph_silac.heatmap(
    expand_neigh.nodes,
    subset_index='identifier',
    index='label',
    rank_index=False,
    cluster_row=True,
    min_sig=2,
    linewidths=0.01
);

exp_data.label_free.heatmap(
    expand_neigh.nodes,
    subset_index='identifier',
    index='label',
    rank_index=False,
    cluster_row=True,
    min_sig=2,
    linewidths=0.01
);
```

[ ]:

```
down_casp3 = net_sub.expand_neighbors(
    network=None,
    nodes=['CASP3'],
    upstream=False, downstream=True,
    max_dist=1,
    include_only=exp_data.label_free.sig.require_n_sig(index='label', n_sig=2).
 ↪id_list
)
down_casp3 = utils.delete_disconnected_network(down_casp3)
print(len(down_casp3.nodes))
print(len(down_casp3.edges))
exp_data.label_free.heatmap(
    down_casp3.nodes,
    subset_index='identifier', index='label',
    min_sig=2, linewidths=0.01,
    cluster_row=True
);

exp_data.label_free.require_n_sig(index='label', n_sig=2).heatmap(
    down_casp3.nodes, subset_index='identifier',
    #index='label',
    rank_index=True,
    figsize=(8, 2),
    index='sample_id', columns='label',
    min_sig=2, linewidths=0.01, cluster_row=False, cluster_col=False
);
plt.yticks(rotation=0)
plt.savefig('down_from_casp3.png', dpi=300, bbox_inches='tight')
```

[ ]:

```
vis.draw_cyjs(down_casp3)
```

[ ]:

```
def show_neighbors(node, df, upstream=True, downstream=False, max_dist=1,
                   include_only=None, figsize=None):

    df_copy = df.copy()
    df_copy.require_n_sig(n_sig=1, inplace=True)

    neighbors = net_sub.expand_neighbors(
        network=None,
        nodes=[node],
        upstream=upstream,
        downstream=downstream,
        max_dist=max_dist,
        include_only=include_only
    )
    neighbors = utils.delete_disconnected_network(neighbors)
    s_name =  'node_{}.png'.format(node)
    exporters.export_to_dot(neighbors, s_name, image_format='png',␣
 ↪engine='circo')
    display(Image(s_name, width=400))

    g = df_copy.heatmap(
        sorted(neighbors.nodes),
        subset_index='identifier',
        index='label',
        min_sig=1,
        rank_index=True,
        linewidths=0.01,
        figsize=figsize
    );

    g.savefig('{}_heatmap.png'.format(node), bbox_inches='tight', dpi=300)
show_neighbors('CASP3', exp_data.label_free, False, True, max_dist=1,
              include_only=exp_data.label_free.sig.require_n_sig(n_sig=2).
 ↪id_list)
```

[ ]:

```
show_neighbors('BAX', exp_data.species, False, True,  max_dist=1,
              include_only=exp_data.species.sig.require_n_sig(n_sig=1).id_list)
```

[ ]:

[ ]: