Conversion-seq analysis with GrandPy - CHEAT SHEET

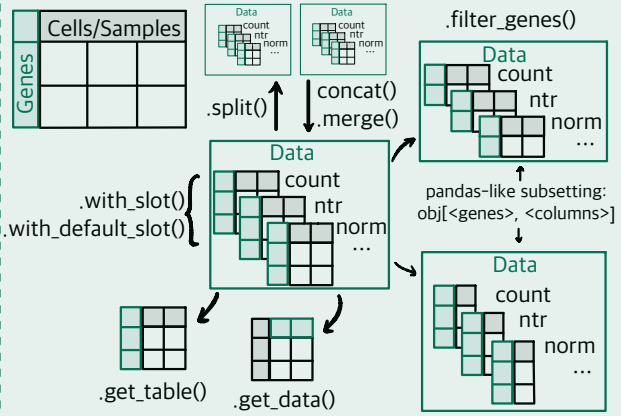
GrandPy OBJECT

Metadata
Metadata for samples/ cells and genes

Data
Matrices for counts, ntrs, etc.

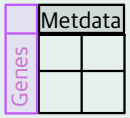
Analyses
Half-lives, fold changes, p value

DATA



METADATA

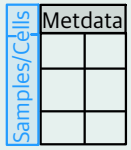
Gene metadata



.gene_info

Stores information per Gene such as gene IDs, gene symbols, transcript length, type, etc.
Access a list of gene names: .genes

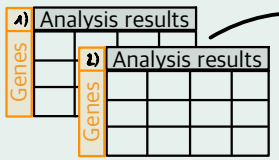
Columns metadata



.coldata

Stores information per sample/cell such as labeling duration, experimental condition, replicate, genotype etc.
Access a list of conditions: .condition

ANALYSES



.get_analysis_table()

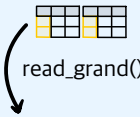
Access a list of Analyses:
.analyses

WORKFLOW

General

Defining samples/cells metadata:
- Using systematic sample names:
Mock.2h.A
read_grand(prefix, design = ('Condition', 'duration.4sU', 'Replicate'), ⋯)
- Using a metadata table

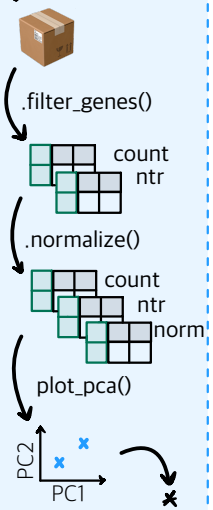
Load Data



Preprocessing

obj.filter_genes(mode_slot = 'count', min_expression = 100, min_columns = 4)
>= 100 counts in 4 samples/cells

obj.filter_genes(mode_slot = 'tpm', min_expression = 10, min_condition = 1)
>= 10 TPM in 1 condition

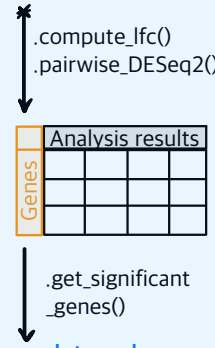


normalize(): size factor normalization (e.g., DESeq2)
Alternatives: .normalize_tpm(), .normalize_fpkm(), .normalize_rpm(), .normalize_baseline()

Quality control

plot_pca(obj): visualize sample clustering and detect outliers based on expression or NTR values -> identify global trends and batch effects

Differential Expression



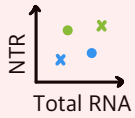
Generate Contrast Matrix for pairwise DE Analysis:
.get_contrasts(data, contrast = ['Condition', ⋯], group = 'Timepoint')
contrast = ['Condition']: All pairwise compa-risons among condition
contrast = ['Condition', 'Control']: Each other condition vs. control
contrast = ['Condition', 'Infected', 'Control']: Infected vs. control
group = 'Timepoint': Comparisons per timepoint

⋯ plots and more

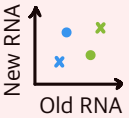
obj.get_significant_genes(criteria = 'Q < 0.05 and abs(LFC) > 1')
Gene names (significant, > 2-fold upregulated)
obj.get_significant_genes(criteria = 'abs(LFC) > 1', as_table = True)
Gene table (> 2-fold regulated)
obj.get_significant_genes(criteria = 'LFC')
All gene names (ordered by fold change)

VISUALIZATION

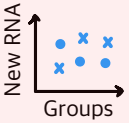
Gene-wise



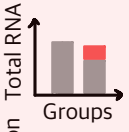
plot_gene_total_vs_ntr()



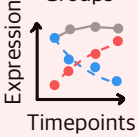
plot_gene_old_vs_ntr()



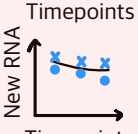
plot_gene_groups_points()



plot_gene_groups_bars()

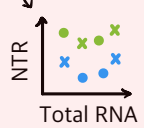


plot_gene_progressive
_timecourse()

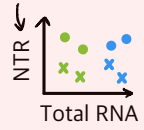


plot_gene_snapshot
_timecourse()

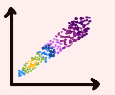
Adapt aesthetic mapping using Coldata columns:
plot_gene_total_vs_ntr(data, 'gene', aest = {'color': 'Condition', 'shape': 'Replicate'})



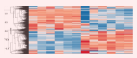
plot_gene_total_vs_ntr(data, 'gene', aest = {'color': 'Genotype', 'shape': 'Condition'})

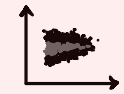


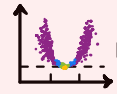
Global



Scatter two variables (expression values, analysis results). Genes can be highlighted (highlight = 'UHMK1') and labeled (label = 'MYC').







Kinetic modeling

.fit_kinetics()

obj.fit_kinetics(fit_type = 'nlls')
fit_type = 'nlls': Non-linear least square fit (steady state and non steady state)
fit_type = 'ntr': Bayesian fit (only steady state)

Calibrate Times

.calibrate_effective_labeling_time_kinetic_fit()
For progressive labeling experiments, infer effective labeling times by jointly optimizing kinetic fits for all genes.