```
#%matplotlib notebook
#%run imports.py

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from matplotlib.gridspec import GridSpec
import string
import sys
import os
import scipy as sp
from scipy import sparse
import sklearn

## add your packages ##

import time
import pickle
#import memory_profiler
from packaging.version import parse as parse_version
from memory_profiler import profile

# Load the memory_profiler extension
#get_ipython().run_line_magic('load_ext', 'memory_profiler')

from pathlib import Path

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.mixture import GaussianMixture as GMM
from scipy import stats
import seaborn as sns
import umap
from sklearn.metrics import pairwise_distances
from scipy.stats import pearsonr
from scipy.spatial.distance import pdist, squareform
from sklearn.metrics import accuracy_score, recall_score
from sklearn.neighbors import KNeighborsClassifier, NearestNeighbors,
NeighborhoodComponentsAnalysis
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import umap.umap_ as umap

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
import igraph as ig
from sklearn.neighbors import NearestNeighbors, kneighbors_graph
import leidenalg as la
from scipy.stats import ttest_ind
```

```
from statsmodels.stats.multitest import multipletests

from Utils import *
```

```
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\tqdm\auto.py:21: TqdmWarning: IProgress not found.
Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

```
import black
import jupyter_black

%load_ext jupyter_black

%load_ext watermark
%watermark --time --date --timezone --updated --python --iversions --
watermark -p sklearn

jupyter_black.load(line_length=79)
```

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

Last updated: 2024-07-29 23:39:33Mitteleuropäische Sommerzeit

Python implementation: CPython
Python version       : 3.10.14
IPython version      : 8.26.0

sklearn: 1.5.1

igraph       : 0.11.6
seaborn      : 0.13.2
black        : 24.4.2
matplotlib   : 3.9.1
scipy        : 1.14.0
sklearn      : 1.5.1
leidenalg    : 0.10.2
jupyter_black: 0.3.4
umap         : 0.5.6
sys          : 3.10.14 | packaged by conda-forge | (main, Mar 20 2024,
12:40:08) [MSC v.1938 64 bit (AMD64)]
numpy        : 2.0.0
pandas       : 2.2.2

Watermark: 2.4.3
```

```
variables_path = Path("../results/variables")
figures_path = Path("../results/figures")
data_path = Path("../data")
```

```
plt.style.use("../matplotlib_style.txt")

np.random.seed(42)
```

Data

We are going to use the multimodal data from the paper Scala et al. 2021 (https://www.nature.com/articles/s41586-020-2907-3#Sec7). In particular, you will work with transcriptomics and electrophysiological data. From the transcriptomics gene counts, we will only work with the exon counts for simplicity. Some of the electrophysiological features are not high-quality recordings, therefore we will also filter them out for the project.

Import

Meta data

```
# META DATA

meta = pd.read_csv(data_path / "m1_patchseq_meta_data.csv", sep="\t")

cells = meta["Cell"].values

layers = meta["Targeted layer"].values.astype("str")
cre = meta["Cre"].values
yields = meta["Yield (pg/µl)"].values
yields[yields == "?"] = np.nan
yields = yields.astype("float")
depth = meta["Soma depth (µm)"].values
depth[depth == "Slice Lost"] = np.nan
depth = depth.astype(float)
thickness = meta["Cortical thickness (µm)"].values
thickness[thickness == 0] = np.nan
thickness = thickness.astype(float)
traced = meta["Traced"].values == "y"
exclude = meta["Exclusion reasons"].values.astype(str)
exclude[exclude == "nan"] = ""

mice_names = meta["Mouse"].values
mice_ages = meta["Mouse age"].values
mice_cres = np.array(
    [c if c[-1] != "+" and c[-1] != "-" else c[:-1] for c in
meta["Cre"].values]
)
mice_ages_dict = dict(zip(mice_names, mice_ages))
mice_cres = dict(zip(mice_names, mice_cres))

print("Number of cells with measured depth:    ",
np.sum(~np.isnan(depth)))
```

```
print("Number of cells with measured thickness:",
np.sum(~np.isnan(thickness)))
print("Number of reconstructed cells:          ", np.sum(traced))

sliceids = meta["Slice"].values
a, b = np.unique(sliceids, return_counts=True)
assert np.all(b <= 2)
print("Number of slices with two cells:        ", np.sum(b == 2))

# Some consistency checks
assert np.all(
    [np.unique(meta["Date"].values[mice_names == m]).size == 1 for m
in mice_names]
)
assert np.all(
    [np.unique(meta["Mouse age"].values[mice_names == m]).size == 1
for m in mice_names]
)
assert np.all(
    [
        np.unique(meta["Mouse gender"].values[mice_names == m]).size
== 1
        for m in mice_names
    ]
)
assert np.all(
    [
        np.unique(meta["Mouse genotype"].values[mice_names == m]).size
== 1
        for m in mice_names
    ]
)
assert np.all(
    [np.unique(meta["Mouse"].values[sliceids == s]).size == 1 for s in
sliceids]
)
```

```
Number of cells with measured depth:     1284
Number of cells with measured thickness: 1284
Number of reconstructed cells:           646
Number of slices with two cells:         69
```

Transcriptomic data

```
# READ COUNTS

data_exons = pd.read_csv(
    data_path / "m1_patchseq_exon_counts.csv.gz", na_filter=False,
index_col=0
)
```

```
exonCounts = data_exons.values.transpose()

assert all(cells == data_exons.columns)
genes = np.array(data_exons.index)

print("Count matrix shape (exon):  ", exonCounts.shape)
```

```
Count matrix shape (exon):   (1329, 42466)
```

```
data_exons
```

	20171204_sample_2	20171204_sample_4	20171204_sample_5
0610005C13Rik	0	0	0
0610006L08Rik	0	0	0
0610009B22Rik	0	68	1291
0610009E02Rik	0	0	0
0610009L18Rik	0	0	0
...	...	...	...
n-R5s96	0	0	0
n-R5s97	0	0	0
n-R5s98	0	0	0
n-TSaga9	0	0	0
n-TStga1	0	0	0

	20171204_sample_6	20171207_sample_1	20171207_sample_2
0610005C13Rik	0	0	1
0610006L08Rik	0	0	13
0610009B22Rik	0	0	0
0610009E02Rik	0	30	80
0610009L18Rik	0	0	99
...	...	...	...
n-R5s96	0	0	0

```
n-R5s97                          0                  0                   0

n-R5s98                          0                  0                   0

n-TSaga9                         0                  0                   0

n-TStga1                         0                  0                   0


               20171207_sample_6  20171207_sample_7  20171219_sample_1
\
0610005C13Rik                    0                  0                   0

0610006L08Rik                    0                  0                   0

0610009B22Rik                  227                  0                   7

0610009E02Rik                  205                  0                   0

0610009L18Rik                    0                380                   0

...                            ...                ...                 ...

n-R5s96                          0                  0                   0

n-R5s97                          0                  0                   0

n-R5s98                          0                  0                   0

n-TSaga9                         0                  0                   0

n-TStga1                         0                  0                   0


               20171219_sample_2  ...  20191114_sample_9
20200106_sample_1  \
0610005C13Rik                    0  ...                  0
0
0610006L08Rik                    0  ...                  0
0
0610009B22Rik                   10  ...                  0
271
0610009E02Rik                   14  ...                  0
0
0610009L18Rik                    0  ...                  0
0
...                            ...  ...                ...
...
n-R5s96                          0  ...                  0
0
n-R5s97                          0  ...                  0
```

```
0
n-R5s98                          0  ...                   0
0
n-TSaga9                         0  ...                   0
0
n-TStga1                         0  ...                   0
0

                20200106_sample_4  20200106_sample_5  20200106_sample_6
\
0610005C13Rik                    0                  0                  1

0610006L08Rik                    0                  0                  0

0610009B22Rik                    0                  0                  0

0610009E02Rik                    0                  0                  0

0610009L18Rik                    0                  0                  0

...                            ...                ...                ...

n-R5s96                          0                  0                  0

n-R5s97                          0                  0                  0

n-R5s98                          0                  0                  0

n-TSaga9                         0                  0                  0

n-TStga1                         0                  0                  0


                20200225_sample_2  20200225_sample_5  20200316_sample_1
\
0610005C13Rik                    0                  0                  0

0610006L08Rik                    0                  0                  0

0610009B22Rik                    0                138                  0

0610009E02Rik                    0                  1                  0

0610009L18Rik                    0                  0                  0

...                            ...                ...                ...

n-R5s96                          0                  0                  0

n-R5s97                          0                  0                  0
```

```
n-R5s98                          0                   0                   0

n-TSaga9                         0                   0                   0

n-TStga1                         0                   0                   0


                20200316_sample_2  20200316_sample_3
0610005C13Rik                   0                  0
0610006L08Rik                   0                  0
0610009B22Rik                  78                 89
0610009E02Rik                   0                  0
0610009L18Rik                   0                  0
...                           ...                ...
n-R5s96                         0                  0
n-R5s97                         0                  0
n-R5s98                         0                  0
n-TSaga9                        0                  0
n-TStga1                        0                  0

[42466 rows x 1329 columns]
```

```
# GENE LENGTH

data = pd.read_csv(data_path / "gene_lengths.txt")
assert all(data["GeneID"] == genes)
exonLengths = data["exon_bp"].values
intronLengths = data["intron_bp"].values
```

Cluster colors

```
cluster_colors = np.load(data_path / "cluster_colors.npy")

cluster_colors.shape
print(len(np.unique(cluster_colors)))
```

```
78
```

```
rna_type = np.load(data_path / "rna_type.npy", allow_pickle=True)

ephysData = pd.read_csv(data_path / "m1_patchseq_ephys_features.csv")

print(rna_type.shape)
print(rna_type)
```

```
(1329,)
['L5 ET_1' 'L5 IT_2' 'L5 IT_1' ... 'L5 ET_1' nan 'Pvalb Gpr149']
```

```
pickle_in = open(data_path / "dict_rna_type_colors.pkl", "rb")
dict_rna_type_colors = pickle.load(pickle_in)
```

```
print(dict_rna_type_colors)
rna_type_colors_array = np.array(list(dict_rna_type_colors.values()))
print(len(rna_type_colors_array))
print(len(np.unique(rna_type_colors_array)))
```

```
{'Lamp5 Pax6': '#DDACC9', 'Lamp5 Egln3_1': '#FF88AD', 'Lamp5 Egln3_2':
'#DD8091', 'Lamp5 Egln3_3': '#F08E98', 'Lamp5 Pdlim5_1': '#FF7290',
'Lamp5 Pdlim5_2': '#FF8C97', 'Lamp5 Slc35d3': '#FFA388', 'Lamp5 Lhx6':
'#C77963', 'Sncg Col14a1': '#7E0ACB', 'Sncg Slc17a8': '#9440F3', 'Sncg
Calb1_1': '#9611B6', 'Sncg Calb1_2': '#9900B3', 'Sncg Npy2r':
'#7A0099', 'Vip Sncg': '#AA4DB5', 'Vip Serpinf1_1': '#A720FF', 'Vip
Serpinf1_2': '#AB1DFF', 'Vip Serpinf1_3': '#FF5FCD', 'Vip Htr1f':
'#FF4DC1', 'Vip Gpc3': '#B09FFF', 'Vip C1ql1': '#BD3D9A', 'Vip
Mybpc1_2': '#992E81', 'Vip Mybpc1_1': '#F70CF3', 'Vip Chat_1':
'#FF00FF', 'Vip Mybpc1_3': '#AB379C', 'Vip Chat_2': '#B3128A', 'Vip
Igfbp6_1': '#7779BF', 'Vip Igfbp6_2': '#626EB8', 'Sst Chodl':
'#FFFF00', 'Sst Penk': '#FF8011', 'Sst Myh8_1': '#FF9F2C', 'Sst
Myh8_2': '#FFB307', 'Sst Myh8_3': '#FFBF09', 'Sst Htr1a': '#BFAF00',
'Sst Etv1': '#FFB22B', 'Sst Pvalb Etv1': '#D9C566', 'Sst Crhr2_1':
'#BE8652', 'Sst Crhr2_2': '#B0993C', 'Sst Hpse': '#CDB115', 'Sst
Calb2': '#D2A328', 'Sst Pappa': '#635821', 'Sst Pvalb Calb2':
'#784F14', 'Sst C1ql3_1': '#802600', 'Sst C1ql3_2': '#8A2B1A', 'Sst
Tac2': '#804600', 'Sst Th_1': '#8C6012', 'Sst Th_2': '#A81111', 'Sst
Th_3': '#9B211B', 'Pvalb Gabrg1': '#ED4C50', 'Pvalb Egfem1':
'#C05661', 'Pvalb Gpr149': '#E62A5D', 'Pvalb Kank4': '#BC4B11', 'Pvalb
Calb1_1': '#B6411E', 'Pvalb Calb1_2': '#BC2D71', 'Pvalb Reln':
'#9C4165', 'Pvalb Il1rapl2': '#BC2C41', 'Pvalb Vipr2_1': '#FF197F',
'Pvalb Vipr2_2': '#F4358B', 'L2/3 IT_1': '#00FF34', 'L2/3 IT_2':
'#07D945', 'L2/3 IT_3': '#2EB934', 'L4/5 IT_1': '#09CCC6', 'L4/5
IT_2': '#52B8AA', 'L5 IT_1': '#58D2C1', 'L5 IT_2': '#4A9F93', 'L5
IT_3': '#4EAC9C', 'L5 IT_4': '#52B4B8', 'L6 IT_1': '#B2AD23', 'L6
IT_2': '#81791F', 'L6 IT Car3': '#5100FF', 'L5/6 NP CT': '#1AAB99',
'L6 CT Gpr139': '#168577', 'L6 CT Cpa6': '#338C5E', 'L6 CT Grp':
'#2FBCE5', 'L6 CT Pou3f2': '#3E766C', 'L6 CT Kit_1': '#516B78', 'L6 CT
Kit_2': '#557361', 'L6b Col6a1': '#69419D', 'L6b Shisa6_1': '#46306A',
'L6b Shisa6_2': '#464576', 'L6b Ror1': '#7044AA', 'L6b Kcnip1':
'#573D90', 'L5/6 NP_1': '#48CB80', 'L5/6 NP_2': '#3C78BC', 'L5/6
NP_3': '#47867A', 'L5 ET_1': '#0D5D7E', 'L5 ET_2': '#0B77A5', 'L5
ET_3': '#0B8AA5', 'L5 ET_4': '#0A75B1'}
88
88
```

Electrophysiological features

```
# EPHYS DATA

ephysData = pd.read_csv(data_path / "m1_patchseq_ephys_features.csv")
ephysNames = np.array(ephysData.columns[1:]).astype(str)
ephysCells = ephysData["cell id"].values
ephysData = ephysData.values[:, 1:].astype("float")
```

```
names2ephys = dict(zip(ephysCells, ephysData))
ephysData = np.array(
    [names2ephys[c] if c in names2ephys else ephysData[0] * np.nan for
c in cells]
)

print("Number of cells with ephys data:", np.sum(np.isin(cells,
ephysCells)))

assert np.sum(~np.isin(ephysCells, cells)) == 0
```

```
Number of cells with ephys data: 1328
```

```
ephys_pd = pd.read_csv(data_path / "m1_patchseq_ephys_features.csv")

# print(ephys_pd["cell id"])
# print(ephys_pd.head())
# print(cells)

gene_data = pd.read_csv(data_path / "gene_lengths.txt")
print(gene_data.head())
```

```
          GeneID  exon_bp  intron_bp  gene_bp
0  0610005C13Rik     3583       3951     7534
1  0610006L08Rik     2128      32868    34996
2  0610009B22Rik      998       2491     3489
3  0610009E02Rik     1803      11892    13695
4  0610009L18Rik      619       1894     2513
```

```
# Filtering ephys data

features_exclude = [
    "Afterdepolarization (mV)",
    "AP Fano factor",
    "ISI Fano factor",
    "Latency @ +20pA current (ms)",
    "Wildness",
    "Spike frequency adaptation",
    "Sag area (mV*s)",
    "Sag time (s)",
    "Burstiness",
    "AP amplitude average adaptation index",
    "ISI average adaptation index",
    "Rebound number of APs",
]
features_log = [
    "AP coefficient of variation",
    "ISI coefficient of variation",
    "ISI adaptation index",
    "Latency (ms)",
]
```

```
X = ephysData

print(X.shape)
for e in features_log:
    X[:, ephysNames == e] = np.log(X[:, ephysNames == e])
X = X[:, ~np.isin(ephysNames, features_exclude)]

keepcells = ~np.isnan(np.sum(X, axis=1))
X = X[keepcells, :]
print(X.shape)

X = X - X.mean(axis=0)
ephysData_filtered = X / X.std(axis=0)
```

```
(1329, 29)
(1320, 17)
```

```
ephysData_filtered
```

```
array([[ 0.98608392,  0.64834158, -0.93881053, ...,  3.04737527,
         0.16776081,  0.63710672],
       [ 0.58728941,  0.45548548, -0.80161871, ...,  2.96590616,
        -0.76260223,  0.99778569],
       [ 0.36748484,  0.036109  , -0.11310118, ...,  3.04737527,
        -0.31069552,  1.05910185],
       ...,
       [ 0.66987131, -0.0487444 , -0.34647255, ...,  4.75822642,
         0.21211713,  0.73777371],
       [ 0.41921208,  0.55079968, -0.32063559, ...,  3.61765898,
         0.10267565,  1.50108896],
       [-0.50165036, -0.14969752, -0.01064697, ...,  0.76624039,
        -1.16822945, -1.10823434]])
```

```
np.sum(np.isnan(ephysData_filtered))
```

```
np.int64(0)
```

Research questions to investigate

1) Inspect the data computing different statistics. Keep in mind that the data is read counts, not UMI, so it is not supposed to follow a Poisson distribution.

2) Normalize and transform the data. There are several ways of normalizing the data (Raw, CPM, CPMedian, RPKM, see https://www.reneshbedre.com/blog/expression_units.html, https://translational-medicine.biomedcentral.com/articles/10.1186/s12967-021-02936-w). Take into account that there are certain normalizations that only make sense for UMI data. You also explored different transformations in the assignment (none, log, sqrt). Compare how the different transformations change the two-dimensional visualization.

3) Two-dimensional visualization. Try different methods (t-SNE, UMAP) / parameters (exagg., perplex.) for visualizations. Compare them using quantitative metrics (e.g., distance correlation, kNN accuracy/recall in high-dim vs. two-dim). Think about also using the electrophysiological features for different visualizations.

4) Clustering. Try different clustering methods (leiden, GMM). Implement a negative binomial mixture model. For that you can follow a similar method that what is described in Harris et al. 2018 (https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.2006387#abstract0), with fixed r (r=2) and S (set of important genes). Evaluate your clustering results (metrics, compare number of clusters to original labels,...).

5) Correlation in between electrophysiological features and genes/PCs. Find correlations and a way of visualizing them.

```
import json

# Convert rna_type to a Pandas Series if it is not already
rna_type = pd.Series(rna_type) if not isinstance(rna_type, pd.Series)
else rna_type

# Remove NaNs from rna_type and count occurrences of each RNA type
rna_type_clean = rna_type.dropna()
rna_type_counts = rna_type_clean.value_counts().to_dict()

# Add a 'Not identified' key if there were NaNs in the original
rna_type array
if rna_type.isnull().sum() > 0:
    rna_type_counts["Not identified"] = rna_type.isnull().sum()

# Sort the dictionary by keys
rna_type_counts = dict(sorted(rna_type_counts.items()))
# Move 'Not identified' to the end if it exists
if "Not identified" in rna_type_counts:
    rna_type_counts["Not identified"] = rna_type_counts.pop("Not
identified")

# Create a DataFrame for cell types and their colors
cell_types = list(rna_type_counts.keys())
mapping_db = pd.DataFrame({"cell_types": cell_types})

# Assign colors to cell types
mapping_db["color"] = mapping_db["cell_types"].apply(
    lambda cell_type: dict_rna_type_colors.get(cell_type, "gray")
)

# Define marker lists based on the paper
excitatory_markers = [
    "L2/3 IT",
    "L4/5 IT",
    "L5 IT",
```

```
    "L6 IT",
    "L6b",
    "L6",
    "L5/6",
    "L5",
    "L6",
    "L2/3",
    "L4",
    "L5 IT",
    "L6 IT",
    "L6 CT",
    "L6b",
    "L6 CT",
]
cge_markers = ["Lamp5", "Sncg", "Vip"]
mge_markers = ["Sst", "Pvalb"]


# Assign types to cell types based on markers
def assign_type(cell_type):
    if any(marker in cell_type for marker in excitatory_markers):
        return "Excitatory neurons"
    if any(marker in cell_type for marker in cge_markers):
        return "CGE-derived interneurons"
    if any(marker in cell_type for marker in mge_markers):
        return "MGE-derived interneurons"
    return None


mapping_db["type"] = mapping_db["cell_types"].apply(assign_type)

# Add counts to the DataFrame
mapping_db["count"] = mapping_db["cell_types"].map(rna_type_counts)

# Sort the DataFrame by type and cell types
mapping_db = mapping_db.sort_values(by=["type", "cell_types"])

# Define groups
neuron_groups = [
    "Excitatory neurons",
    "CGE-derived interneurons",
    "MGE-derived interneurons",
]

def calculate_top_fano_factor(gene_expression_df, top_genes=500):
    """
    Calculate the top genes with the highest Fano factors.

    Parameters
    ----------
```

```
    gene_expression_df : pandas.DataFrame (n_genes, n_cells)
        The gene expression matrix.

    top_genes : int
        The number of top genes to return.

    Returns
    -------
    top_fano_factors : numpy.ndarray (top_genes,)
        The top Fano factors.

    top_gene_expression : pandas.DataFrame (n_genes, top_genes)

    """
    # Calculate the mean and variance of gene expression
    mean_expression = gene_expression_df.mean(axis=1)
    variance_expression = gene_expression_df.var(axis=1)

    # Calculate the Fano factor
    fano_factor = variance_expression / mean_expression

    # Exclude NaNs from fano_factor
    valid_indices = ~np.isnan(fano_factor)

    # Get non-NaN Fano factors and their corresponding indices
    valid_fano_factors = fano_factor[valid_indices]
    valid_gene_expression = gene_expression_df[valid_indices]

    # Sort the non-NaN Fano factors in descending order
    sorted_indices = np.argsort(-valid_fano_factors)

    # Select the top genes
    top_indices = sorted_indices[:top_genes]

    # Get the top genes and their Fano factors
    top_fano_factors = valid_fano_factors.iloc[top_indices]
    top_gene_expression = valid_gene_expression.iloc[top_indices]

    return top_fano_factors.values, top_gene_expression


# Calculate the top genes with the highest Fano factors

top_fano_factors, top_gene_expression = calculate_top_fano_factor(
    data_exons, top_genes=10
)


def calculate_top_fano_factor_array(gene_expression_array,
top_genes=500):
    """
```

```
    Calculate the top genes with the highest Fano factors.

    Parameters
    ----------
    gene_expression_array : numpy.ndarray (n_cells, n_genes)
        The transposed gene expression matrix.

    top_genes : int
        The number of top genes to return.

    Returns
    -------
    top_fano_factors : numpy.ndarray (top_genes,)
        The top Fano factors.

    top_gene_expression : numpy.ndarray (top_genes, n_cells)
        The gene expression matrix for the top genes.
    """
    # Calculate the mean and variance of gene expression
    mean_expression = np.mean(gene_expression_array, axis=0)
    variance_expression = np.var(gene_expression_array, axis=0)

    # Calculate the Fano factor
    fano_factor = variance_expression / mean_expression

    # Exclude NaNs from fano_factor
    valid_indices = ~np.isnan(fano_factor)

    # Get non-NaN Fano factors and their corresponding indices
    valid_fano_factors = fano_factor[valid_indices]
    valid_gene_expression = gene_expression_array[:, valid_indices]

    # Sort the non-NaN Fano factors in descending order
    sorted_indices = np.argsort(-valid_fano_factors)

    # Select the top genes
    top_indices = sorted_indices[:top_genes]

    # Get the top genes and their Fano factors
    top_fano_factors = valid_fano_factors[top_indices]
    top_gene_expression = valid_gene_expression[:, top_indices]

    return top_fano_factors, top_gene_expression


# Print the top genes with the highest Fano factors and their
corresponding fanos factors
# print(zip(top_gene_expression.index, top_fano_factors))

print(mice_names.shape)
print(exonCounts.shape)
```

```
mice_names_keep = mice_names[keepcells]
mice_ages_keep = mice_ages[keepcells]
cre_keep = cre[keepcells]
rna_type_keep = rna_type[keepcells]
exonCounts_keep = exonCounts[keepcells, :]
# compare variance of gene expression between mice to variance within
mice
anova_results = []
# for each unique mice name get the indices of the cells
unique_mice_names = np.unique(mice_names_keep)
mice_cells = {}

print("Number of individual mice: ", len(unique_mice_names))
# print(unique_mice_names.shape)
for m in unique_mice_names:
    # Get indices of the cells for the current mouse
    mice_cell_indices = np.where(mice_names_keep == m)[0]

    # Get gene expression data for these cells
    gene_expression_data = exonCounts_keep[mice_cell_indices, :]

    # Store gene expression data in the dictionary
    mice_cells[m] = {
        "Gene_expression": gene_expression_data,
        "Variance": np.var(gene_expression_data, axis=0),
    }

# Assuming mice_cells is already populated with gene expression and
variance data
# Step 1: Prepare the data for ANOVA
variances = []
group_labels = []

for mouse, data in mice_cells.items():
    gene_expression_var = np.var(data["Gene_expression"], axis=0)
    variances.extend(gene_expression_var)
    group_labels.extend([mouse] * gene_expression_var.shape[0])

# Create a DataFrame for easier handling
df = pd.DataFrame({"Variance": variances, "Mouse": group_labels})

# Step 2: Perform ANOVA
anova_result = stats.f_oneway(
    *(df[df["Mouse"] == mouse]["Variance"] for mouse in
unique_mice_names)
)
anova_results.append(anova_result)
print(f"ANOVA result: F={anova_result.statistic},
p={anova_result.pvalue}")
```

```
# get samples with the same ages
unique_ages = np.unique(mice_ages_keep)
print("Number of unique ages: ", len(unique_ages))
age_cells = {}

for a in unique_ages:
    # Get indices of the cells for the current mouse
    age_cell_indices = np.where(mice_ages_keep == a)[0]

    # Get gene expression data for these cells
    gene_expression_data = exonCounts_keep[age_cell_indices, :]

    # Store gene expression data in the dictionary
    age_cells[a] = {
        "Gene_expression": gene_expression_data,
        "Variance": np.var(gene_expression_data, axis=0),
    }

# Assuming age_cells is already populated with gene expression and
variance data
# Step 1: Prepare the data for ANOVA
variances_age = []
group_labels_age = []

for age, data in age_cells.items():
    gene_expression_var = np.var(data["Gene_expression"], axis=0)
    variances_age.extend(gene_expression_var)
    group_labels_age.extend([age] * gene_expression_var.shape[0])

# Create a DataFrame for easier handling
df_age = pd.DataFrame({"Variance": variances_age, "Age":
group_labels_age})

# Step 2: Perform ANOVA
anova_result_age = stats.f_oneway(
    *(df_age[df_age["Age"] == age]["Variance"] for age in unique_ages)
)

print(f"ANOVA result: F={anova_result_age.statistic},
p={anova_result_age.pvalue}")
anova_results.append(anova_result_age)

# get samples with the same cre type
unique_cre_types = np.unique(cre_keep)
print("Number of unique cre types: ", len(unique_cre_types))
cre_cells = {}

print(cre_keep.shape)
print(exonCounts_keep.shape)
```

```
for c in unique_cre_types:
    # Get indices of the cells for the current cre type
    cre_cell_indices = np.where(cre_keep == c)[0]

    # Get gene expression data for these cells
    gene_expression_data = exonCounts_keep[cre_cell_indices, :]

    # Store gene expression data in the dictionary
    cre_cells[c] = {
        "Gene_expression": gene_expression_data,
        "Variance": np.var(gene_expression_data, axis=0),
    }

# Assuming cre_cells is already populated with gene expression and
variance data
# Step 1: Prepare the data for ANOVA
variances_cre = []
group_labels_cre = []

for cre_type, data in cre_cells.items():
    gene_expression_var = np.var(data["Gene_expression"], axis=0)
    variances_cre.extend(gene_expression_var)
    group_labels_cre.extend([cre_type] * gene_expression_var.shape[0])

# Create a DataFrame for easier handling
df_cre = pd.DataFrame({"Variance": variances_cre, "Cre":
group_labels_cre})

# Step 2: Perform ANOVA
anova_result_cre = stats.f_oneway(
    *(df_cre[df_cre["Cre"] == cre_type]["Variance"] for cre_type in
unique_cre_types)
)

print(f"ANOVA result: F={anova_result_cre.statistic},
p={anova_result_cre.pvalue}")
anova_results.append(anova_result_cre)

# get samples with the same rna type
# reset rna_type_keep index
rna_type_keep = rna_type_keep.reset_index(drop=True)

# print number of nan
print("Number of NaN values in rna_type: ",
rna_type_keep.isnull().sum())

nan_mask = rna_type_keep.notna()

# Apply the mask to rna_type_array and exonCounts_keep
rna_type_filtered = rna_type_keep[nan_mask]
```

```
# reset rna_type_filtered index
rna_type_filtered = rna_type_filtered.reset_index(drop=True)

exonCounts_filtered = exonCounts_keep[nan_mask.values, :]

# Get unique RNA types
unique_rna_types = rna_type_filtered.unique()
print("Number of unique RNA types: ", len(unique_rna_types))

# Dictionary to store gene expression data and variances for each RNA
type
rna_type_cells = {}

print(exonCounts_filtered.shape)
print(rna_type_filtered.shape)
for r in unique_rna_types:
    # Get indices of the cells for the current RNA type
    rna_type_cell_indices = rna_type_filtered[rna_type_filtered ==
r].index

    # Get gene expression data for these cells
    gene_expression_data =
exonCounts_filtered[rna_type_cell_indices, :]

    # Store gene expression data and variance in the dictionary
    rna_type_cells[r] = {
        "Gene_expression": gene_expression_data,
        "Variance": np.var(gene_expression_data, axis=0),
    }

# Prepare data for ANOVA
variances_rna_type = []
group_labels_rna_type = []

for rna_type, data in rna_type_cells.items():
    gene_expression_var = data["Variance"]
    variances_rna_type.extend(gene_expression_var)
    group_labels_rna_type.extend([rna_type] *
gene_expression_var.shape[0])

# Create a DataFrame for easier handling
df_rna_type = pd.DataFrame(
    {"Variance": variances_rna_type, "RNA type":
group_labels_rna_type}
)

# Perform ANOVA
anova_result_rna_type = stats.f_oneway(
    *(
```

```
        df_rna_type[df_rna_type["RNA type"] == rna_type]["Variance"]
        for rna_type in unique_rna_types
    )
)

print(
    f"ANOVA result: F={anova_result_rna_type.statistic},
p={anova_result_rna_type.pvalue}"
)
anova_results.append(anova_result_rna_type)

(1329,)
(1329, 42466)
Number of individual mice:  266
ANOVA result: F=1.8181509337239867, p=8.707167539633704e-15
Number of unique ages:  102
ANOVA result: F=1.6501309746297719, p=4.289434383754619e-05
Number of unique cre types:  18
(1320,)
(1320, 42466)
ANOVA result: F=2.3163657955246326, p=0.0015829130359728523
Number of NaN values in rna_type:  96
Number of unique RNA types:  76
(1224, 42466)
(1224,)
ANOVA result: F=1.7226515321740707, p=0.00010219613389086996

# Data
from matplotlib.table import Table

categories = ["Mice", "Ages", "Cre Types", "RNA Types"]

f_values = []
p_values = []

for i in range(len(anova_results)):
    f_values.append(anova_results[i].statistic)
    p_values.append(anova_results[i].pvalue)


# Create DataFrame for table
data = {"Category": categories, "F-value": f_values, "p-value":
p_values}
df = pd.DataFrame(data)

# Plot
fig, ax = plt.subplots(figsize=(10, 2))  # Set figure size
ax.axis("off")  # Hide axes

# Create table
```

```
table = ax.table(
    cellText=df.values,
    colLabels=df.columns,
    cellLoc="center",
    loc="center",
    bbox=[0, 0, 1, 1],
)

# Style the table
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

plt.title("ANOVA Results Summary")
plt.show()
```

ANOVA Results Summary

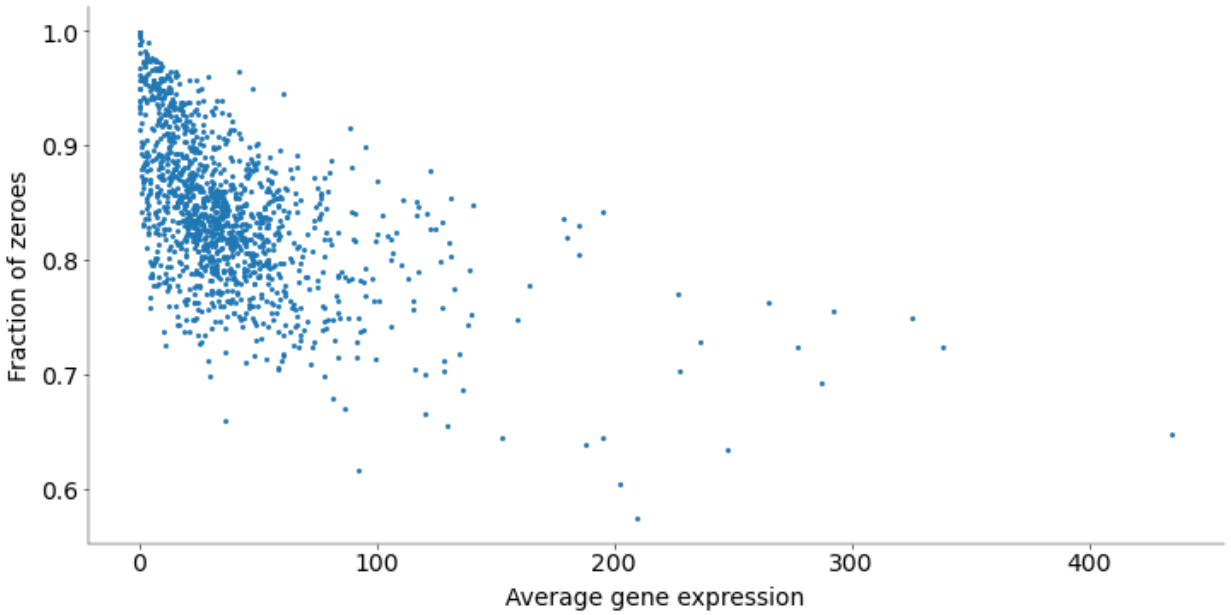
Category	F-value	p-value
Mice	1.8181509337239867	8.707167539633704e-15
Ages	1.6501309746297719	4.289434383754619e-05
Cre Types	2.3163657955246326	0.0015829130359728523
RNA Types	1.7226515321740707	0.00010219613389086996

Transcriptomics data exploration

For the transcriptomics data we calculated the mean expression levels (in counts) per cell and plotted against the fraction of zero expression genes in said cell.

```
# calculate the average gene expression for each cell
average_gene_expression = np.mean(exonCounts, axis=1)
print(average_gene_expression)
# compute the fraction of zeroes for each cell
fraction_zeroes = np.sum(exonCounts == 0, axis=1) /
exonCounts.shape[1]

# plot the average gene expression against the fraction of zeroes
plt.figure()
plt.scatter(average_gene_expression, fraction_zeroes, s=5)
plt.xlabel("Average gene expression")
plt.ylabel("Fraction of zeroes")
plt.show()
```

```
[264.97393209 138.57627278 435.15259266 ...   9.30473791  11.06779541
   9.80271276]
```




```
# normalisation using bioinfotoolkit
from bioinfokit.analys import norm, get_data

# print head of the data
# print("raw data: ", data_exons.head(2))

# using cpm normalization
nm = norm()
nm.cpm(df=data_exons)
# get CPM normalized dataframe
cpm_df = nm.cpm_norm
cpm_df.head(2)


# using rpkm normalization
# print(exonLengths, exonLengths.shape)

# add gene length to the corresponding gene id
# find the matching gene id in data exons and gene data and add the
gene length to the data exons
data_genes = pd.read_csv(data_path / "gene_lengths.txt")

data_exons.index.name = "GeneID"

# Perform the merge
merged_exon_data = pd.merge(data_exons, data_genes, on="GeneID",
how="left")
merged_exon_data.set_index("GeneID", inplace=True)
# Print the head of the merged data
# print("merged: ", merged_exon_data.head(2))
```

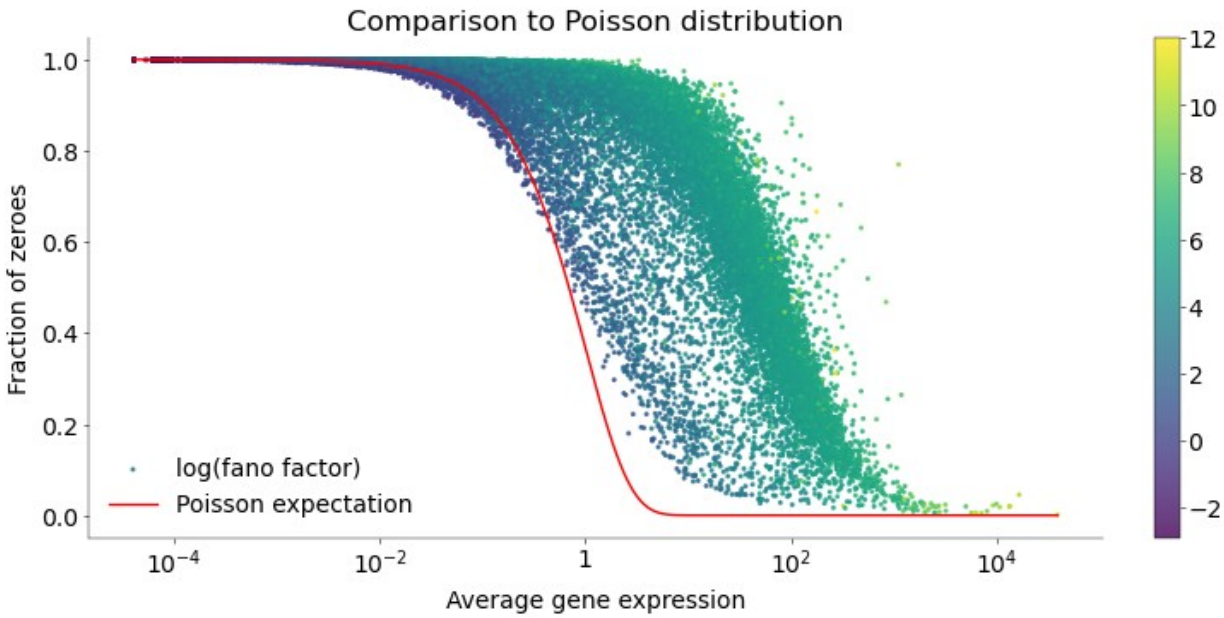
```
nm.rpkm(df=merged_exon_data, gl="exon_bp")

# get RPKM normalized dataframe
rpkm_df = nm.rpkm_norm

# print("RPKM normalized data: ", rpkm_df.head(2))

# using tpm normalization

nm.tpm(df=merged_exon_data, gl="exon_bp")

# get TPM normalized dataframe
tpm_df = nm.tpm_norm

# print("TPM normalized data: ", tpm_df.head(2))
```

Next we normalized with CPM (counts per million) and plotted fraction of zeros and variance of expression and the fano factor against the average gene expression. We color coded the dots by the log fano factor and plotted the Poisson expectation for the zero count; we can see that the data is shifted rightwards in comparison.

```
### for the cpm data
cpm_exonCounts = cpm_df.values.transpose()
average_gene_expression_cpm = np.mean(cpm_exonCounts, axis=0)
variance_gene_expression = np.var(cpm_exonCounts, axis=0)
fano_factor = variance_gene_expression / average_gene_expression_cpm


# calculate the average gene expression for each cell


# compute the fraction of zeroes for each cell
fraction_zeroes_cpm = np.mean(cpm_exonCounts == 0, axis=0)

# Add Hertie logo in sign of gratitude of our exellent lecturers and
Exersize class support.
plt.figure()
# img =
plt.imread("We_love_Hertie_so_much_we_cant_even_express_it_in_words.pn
g")
# plt.imshow(img, extent=[-4, 4, 0, 1], aspect="auto")

# plot the average gene expression against the fraction of zeroes
plt.scatter(
    np.log10(average_gene_expression_cpm),
    fraction_zeroes_cpm,
    s=5,
    c=np.log(fano_factor),
    label="log(fano factor)",
    alpha=0.8,
```

```
)  # np.arange(len(average_gene_expression_cpm)))

y = np.exp(-np.unique(average_gene_expression_cpm))
plt.plot(
    np.log10(np.unique(average_gene_expression_cpm)),
    y,
    color="red",
    label="Poisson expectation",
)
plt.legend()
plt.xlabel("Average gene expression")
plt.ylabel("Fraction of zeroes")
plt.title("Comparison to Poisson distribution")
plt.colorbar()

# make the plot loglog
# plt.xscale("log")
# plt.yscale("log")
plt.xticks([-4, -2, 0, 2, 4], ["$10^{-4}$", "$10^{-2}$", "1",
"$10^{2}$", "$10^{4}$"])
plt.show()
```

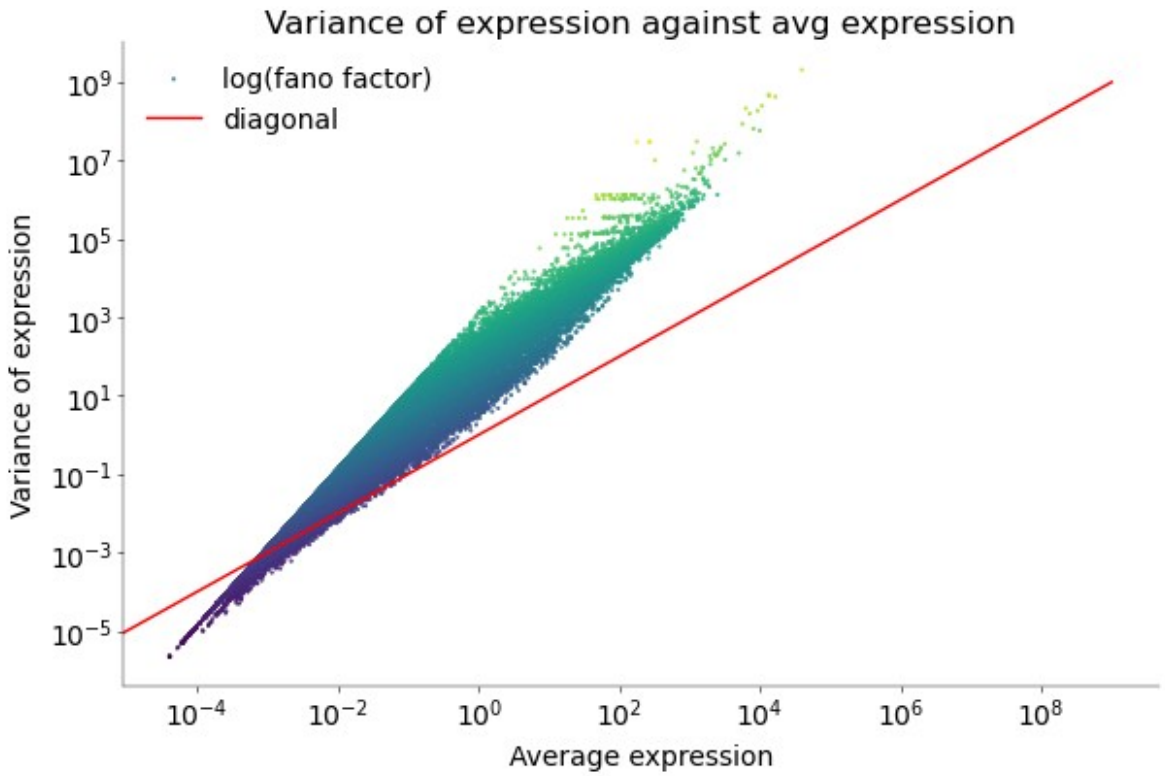
```
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\2904957403.py:5: RuntimeWarning: invalid value
encountered in divide
  fano_factor = variance_gene_expression / average_gene_expression_cpm
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\2904957403.py:21: RuntimeWarning: divide by zero
encountered in log10
  np.log10(average_gene_expression_cpm),
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\2904957403.py:31: RuntimeWarning: divide by zero
encountered in log10
  np.log10(np.unique(average_gene_expression_cpm)),
```



Mean-variance analysis.

Scatter plot of expression variance against average expression and compare to the diagonal.

```
fig, ax = plt.subplots(figsize=(6, 4))

ax.scatter(
    average_gene_expression_cpm,
    variance_gene_expression,
    s=1,
    c=np.log(fano_factor),
    label="log(fano factor)",
)
ax.plot([0, 10**9], [0, 10**9], color="red", label="diagonal")

# add labels
ax.set_xlabel("Average expression")
ax.set_ylabel("Variance of expression")
# add legend
ax.legend()
plt.title("Variance of expression against avg expression")
# make axis logarithmix
ax.set_xscale("log")
ax.set_yscale("log")


plt.show()
```

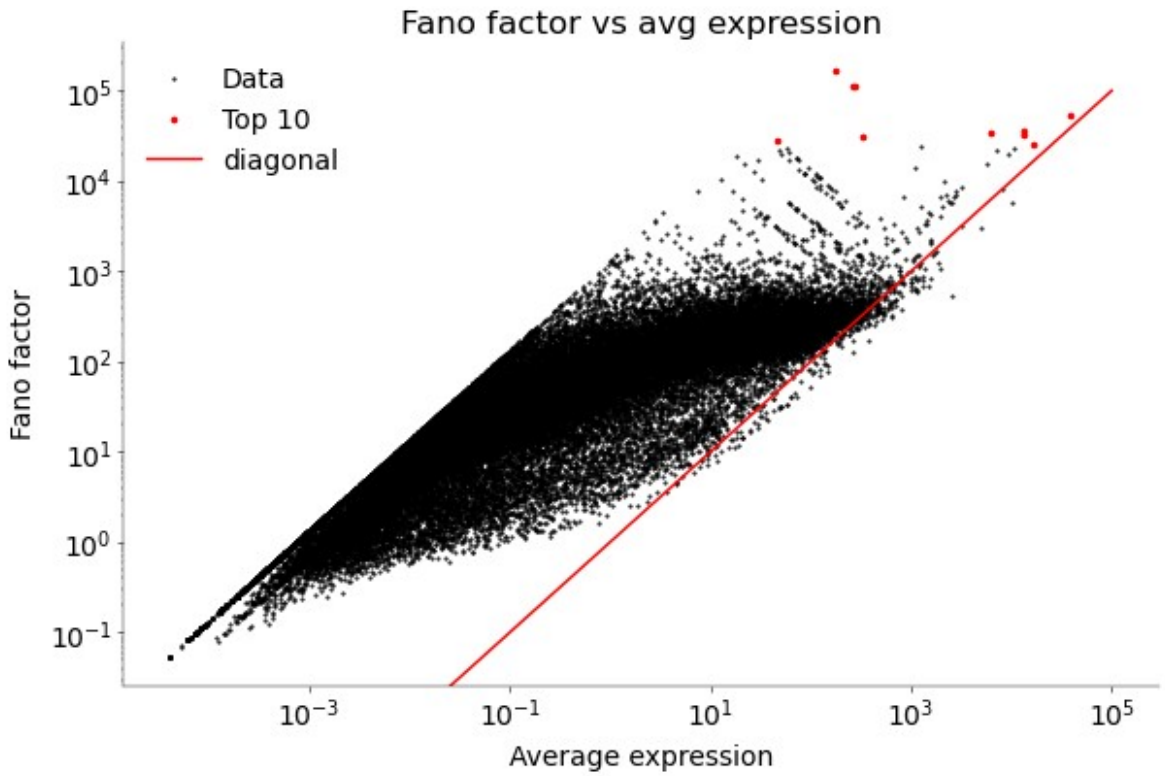


Scattering the Variance of expression against the average expression level we see a deviation form the diagonal. One again the scattered data was color coded by the fano factor.

Plotting the Fano factor against the average expression level

```
fig, ax = plt.subplots(figsize=(6, 4))
fano_nan = fano_factor[~np.isnan(fano_factor)]
avg_nan = average_gene_expression_cpm[~np.isnan(fano_factor)]
ax.scatter(average_gene_expression_cpm, fano_factor, s=1,
color="black", label="Data")
top10 = np.argsort(fano_nan)[-10:]
# print(top10)

# mark the top 10 fanofactors red
ax.scatter(
    avg_nan[top10],
    fano_nan[top10],
    s=10,
    color="red",
    label="Top 10",
)
ax.plot([0, 10**5], [0, 10**5], color="red", label="diagonal")

# add labels
ax.set_xlabel("Average expression")
ax.set_ylabel("Fano factor")
```

```
# add legend
ax.legend()
ax.set_title("Fano factor vs avg expression")

# make axis logarithmic
ax.set_xscale("log")
ax.set_yscale("log")
plt.show()
```



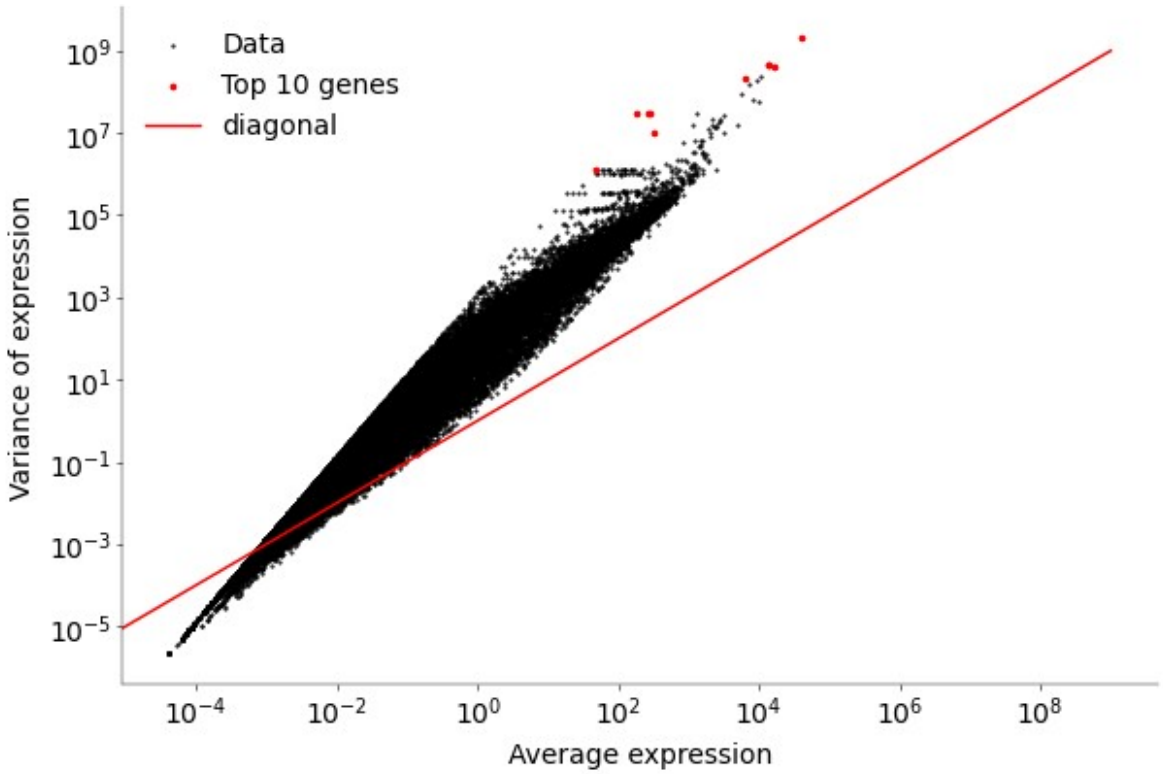
Plotting the fano factor agains the average expression we see that most values lie above the diagonal, ie for most genes $F \geq E[Expression]$. Marked with red dots are the 10 genes with the highest Fano factor.

Histogram of sequencing depth

Next we plotted the distibution of sequencing depth and the variance of expression and printed out which genes corresponded to the highest fano factor, therefore promising a potential role in cellular differentiation between the invesitgated neurons.

```
# Exclude NaNs from fano_factor
valid_indices = ~np.isnan(fano_factor)

# Get non-NaN Fano factors and their corresponding indices
valid_fano_factors = fano_factor[valid_indices]
valid_genes = np.array(genes)[valid_indices]
```

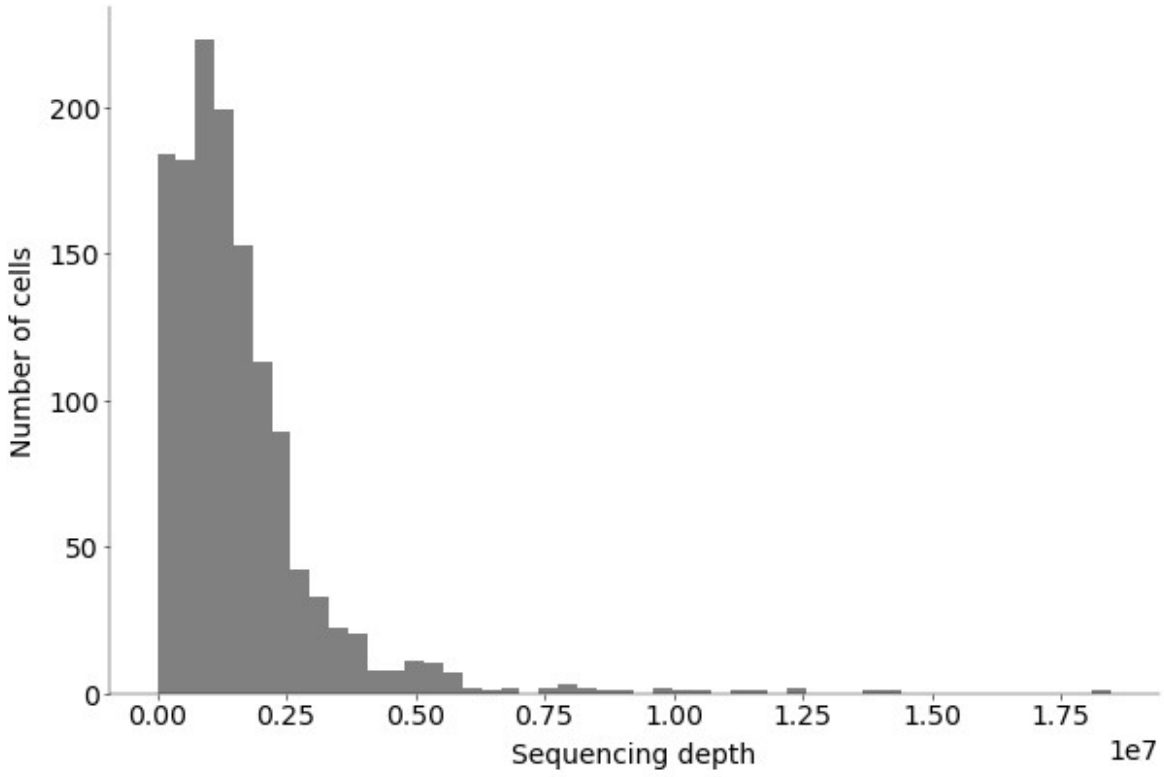
```
valid_average_gene_expression_cpm =
average_gene_expression_cpm[valid_indices]
valid_variance_gene_expression =
variance_gene_expression[valid_indices]

# Sort the non-NaN Fano factors in descending order
sorted_indices = np.argsort(-valid_fano_factors)

# Select the top 10 genes
top10_indices = sorted_indices[:10]

# Create the plot
fig, ax = plt.subplots(figsize=(6, 4))


# Scatter plot for all data
ax.scatter(
    valid_average_gene_expression_cpm,
    valid_variance_gene_expression,
    s=1,
    color="black",
    label="Data",
)

# Highlight the top 10 genes
ax.scatter(
    valid_average_gene_expression_cpm[top10_indices],
    valid_variance_gene_expression[top10_indices],
    s=10,
    color="red",
    label="Top 10 genes",
)


ax.plot([0, 10**9], [0, 10**9], color="red", label="diagonal")

# Add labels
ax.set_xlabel("Average expression")
ax.set_ylabel("Variance of expression")
# Add legend
ax.legend()

# Make axis logarithmic
ax.set_xscale("log")
ax.set_yscale("log")
plt.show()
```




```
# Find top-10 genes with the highest normalized Fano factor
# Print them sorted by the Fano factor starting from the highest

# Find indices of non-NaN values
valid_indices = ~np.isnan(fano_factor)

# Get non-NaN Fano factors and their corresponding indices
valid_fano_factors = fano_factor[valid_indices]
valid_genes = np.array(genes)[valid_indices]

# Sort the non-NaN Fano factors in descending order
sorted_indices = np.argsort(-valid_fano_factors)

# Print the top 10 genes with the highest normalized Fano factor
print("Top 10 genes with the highest normalized Fano factor:")
for i in range(10):
    print(
        f"{valid_genes[sorted_indices[i]]}:
{valid_fano_factors[sorted_indices[i]]:.2f}"
    )

# Give max value of the Fano factor excluding NaNs
max_fano = np.nanmax(fano_factor)
print("Max Fano factor:", max_fano)
```

```
Top 10 genes with the highest normalized Fano factor:
Nectin2: 169717.99
Scai: 113449.80
Thoc2: 109742.37
CT010467.1: 52284.37
ERCC-00130: 35841.41
Rn7s1: 34081.25
ERCC-00074: 32902.02
Mir6236: 31180.47
Atg10: 27783.79
Gm26917: 25380.96
Max Fano factor: 169717.99145220438
```

```
# plot sequencing depth
fig, ax = plt.subplots(figsize=(6, 4))

ax.hist(exonCounts.sum(axis=1), bins=50, color="grey")
# add labels
ax.set_xlabel("Sequencing depth")
ax.set_ylabel("Number of cells")
plt.show()
```



Two-dimensional visualisation of the transcriptomics data

For the upcoming analysis we use normallised counts, log counts and sqrt counts with the different respective normalization techniques.For log counts the data isshifted by 1 to avoid NaN entries in consequence of zero counts:

```
# # # get exon count data
# # exonCounts = data_exons
# # gene_length = gene_data
# # normalise the data using cpm

from Utils import normalize_count_data, perform_pca,
plot_pca_embedding

# normalise the data using cpm, housekeeping method, tpm and rpkm
norm_methods = ["cpm", "tpm", "rpkm", "housekeeping", "total_counts"]
# Include raw data
norm_data = {"Raw": exonCounts[keepcells, :]}
transform_labels = ["log", "sqrt", "log_sqrt", "no"]
transform_methods = ["log", "sqrt", "log_sqrt", [None]]

pca_results = {}

for i, n_method in enumerate(norm_methods):
    norm_exonCounts = normalize_count_data(
        data_exons, data_genes=gene_data, method=n_method
    )
    norm_data[n_method] = norm_exonCounts
    print(norm_exonCounts.shape)

    for j, t_method in enumerate(transform_methods):
        pca_norm, pca_norm_model = perform_pca(
            norm_data[n_method][keepcells, :],
            transformation=t_method,
            standardize=False,
        )
        pca_results[(n_method, transform_labels[j])] = pca_norm,
pca_norm_model

for j, t_method in enumerate(transform_methods):
    pca_raw, pca_raw_model = perform_pca(
        exonCounts[keepcells, :], transformation=t_method,
standardize=False
    )
    pca_results[("Raw", transform_labels[j])] = pca_raw, pca_raw_model

# plot the pca embedding
print(cluster_colors[keepcells].shape)
```

```
# get the 1000 highest fano factor genes
top_fano_factors, top_gene_expression = calculate_top_fano_factor(
    data_exons, top_genes=1000
)
print(top_gene_expression.shape)
pca_results_fano = {}


for i, n_method in enumerate(norm_methods):
    norm_exonCounts = normalize_count_data(
        data_exons, data_genes=gene_data, method=n_method
    )
    norm_data[n_method] = norm_exonCounts
    print(norm_exonCounts.shape)

    for j, t_method in enumerate(transform_methods):
        pca_norm, pca_norm_model = perform_pca(
            norm_data[n_method][keepcells, :],
            transformation=t_method,
            standardize=False,
        )
        pca_results_fano[(n_method, transform_labels[j])] = pca_norm,
pca_norm_model
```

```
(1329, 42466)
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Performing PCA...
Transformation:  sqrt
Square root-transforming data...
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]
(1329, 42466)
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Performing PCA...
Transformation:  sqrt
Square root-transforming data...
```

```
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]
(1329, 42466)
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Performing PCA...
Transformation:  sqrt
Square root-transforming data...
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]
Housekeeping genes: ['Actb', 'Gapdh', 'Ubc', 'Hprt'] Index(['Actb',
'Gapdh', 'Ubc', 'Hprt'], dtype='object', name='GeneID')
Warning: All housekeeping genes have zero counts in sample 58,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 73,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 132,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 409,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 419,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 504,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 1201,
skipping normalization.
(1329, 42466)
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Performing PCA...
Transformation:  sqrt
```

```
Square root-transforming data...
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]
(1329, 42466)
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Performing PCA...
Transformation:  sqrt
Square root-transforming data...
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0. Adding small offset.
Performing PCA...
Transformation:  sqrt
Square root-transforming data...
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]
(1320,)
(1000, 1329)
(1329, 42466)
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
```

```
of 0.0. Adding small offset.
Performing PCA...
Transformation:  sqrt
Square root-transforming data...
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]
(1329, 42466)
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Performing PCA...
Transformation:  sqrt
Square root-transforming data...
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]
(1329, 42466)
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Performing PCA...
Transformation:  sqrt
Square root-transforming data...
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]
Housekeeping genes: ['Actb', 'Gapdh', 'Ubc', 'Hprt'] Index(['Actb',
'Gapdh', 'Ubc', 'Hprt'], dtype='object', name='GeneID')
Warning: All housekeeping genes have zero counts in sample 58,
```

```
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 73,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 132,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 409,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 419,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 504,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 1201,
skipping normalization.
(1329, 42466)
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Performing PCA...
Transformation:  sqrt
Square root-transforming data...
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]
(1329, 42466)
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Performing PCA...
Transformation:  sqrt
Square root-transforming data...
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]
```

```
print(norm_data.keys())
```

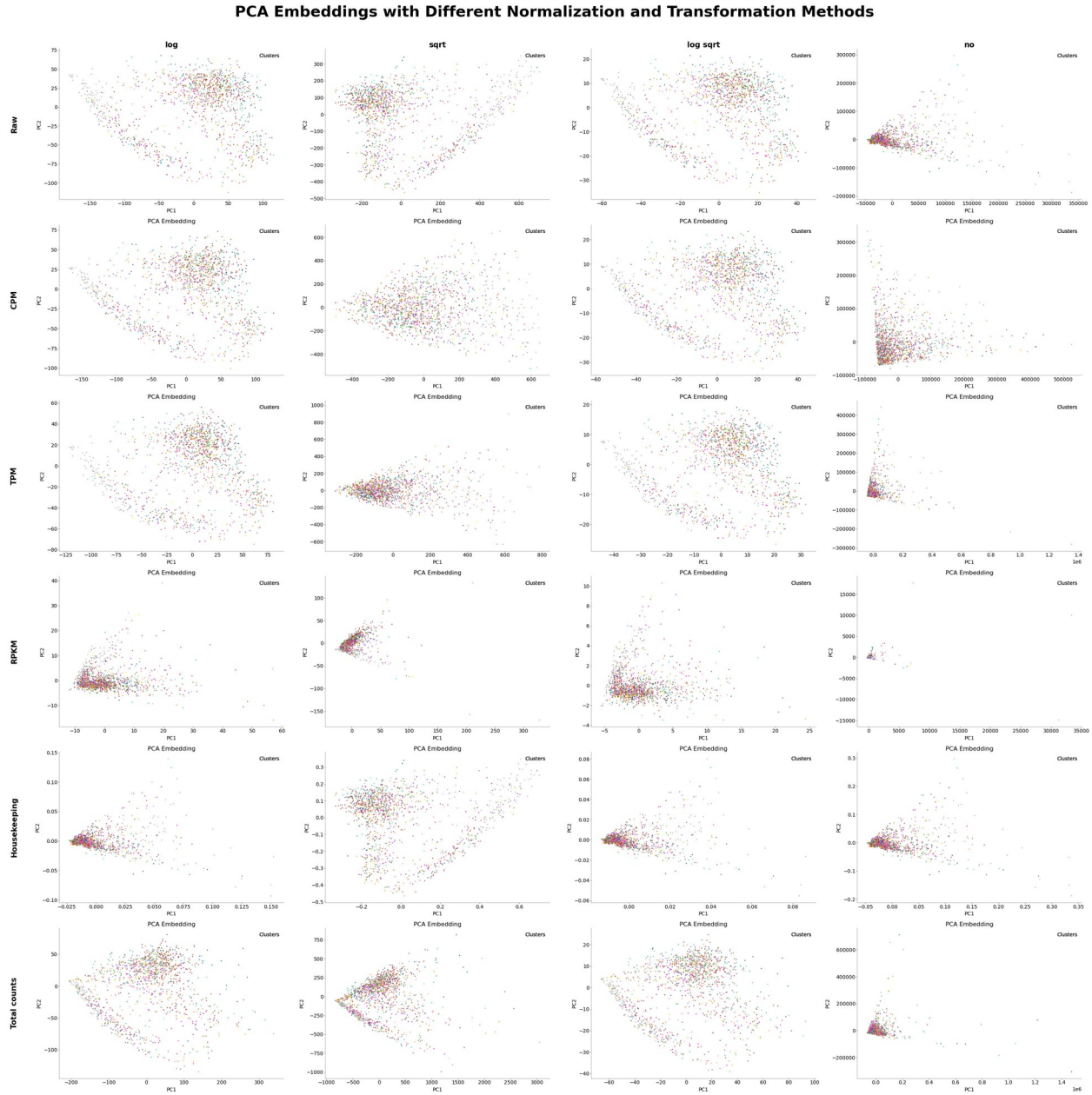
```
dict_keys(['Raw', 'cpm', 'tpm', 'rpkm', 'housekeeping',
'total_counts'])

# Create subplots with 2 rows and 3 columns
fig, axs = plt.subplots(6, 4, figsize=(30, 30))

# Flatten the axs array for easier indexing
axs = axs.flatten()

norm_methods_ = ["Raw", "CPM", "TPM", "RPKM", "Housekeeping", "Total
counts"]
transformations_ = ["log", "sqrt", "log sqrt", "no"]


# Iterate through the PCA results and plot the embeddings and
explained variance
for i, (key, value) in enumerate(pca_results.items()):
    # Plot PCA embedding
    plot_pca_embedding(
        value[0],
        labels=cluster_colors[keepcells],
        # title=f"PCA Embedding with {key[0]} Normalization and
{key[1]} Transformation, explained variance: PC1:
{value[1].explained_variance_ratio_[0]:.2f}, PC2:
{value[1].explained_variance_ratio_[1]:.2f}",
        # keepcells=keepcells,
        ax=axs[i],
    )
    (
        axs[i].set_title(
            transformations_[i],
            fontweight="bold",
            fontsize=15,
        )
        if i < 4
        else None
    )

    # print text next to y axis
    (
        axs[i].text(
            -0.2,
            0.5,
            norm_methods_[int(i // 4)],
            transform=axs[i].transAxes,
            fontsize=15,
            rotation=90,
            va="center",
            ha="center",
            fontweight="bold",
```

```
        )
        if i % 4 == 0
        else None
    )


plt.suptitle(
    "PCA Embeddings with Different Normalization and Transformation
Methods",
    x=0.5,
    y=1.01,
    ha="center",
    fontsize=30,
    fontweight="bold",
)
plt.tight_layout()


plt.show()

fig, axs = plt.subplots(6, 4, figsize=(15, 15))
axs = axs.flatten()

# Iterate through the PCA results again and plot the explained
variance
for i, (key, value) in enumerate(pca_results.items()):
    explained_variance_ratio = value[1]
    cumulative_variance = np.cumsum(explained_variance_ratio)

    # Plot the explained variance of all the principal components
    axs[i].plot(explained_variance_ratio)  # , marker='o',
label='Explained Variance')

    # Add vertical lines for 50%, 70%, and 90% cumulative variance
    for threshold in [0.5, 0.7, 0.9]:
        # Find the index where cumulative variance exceeds the
threshold
        index = np.argmax(cumulative_variance >= threshold)
        axs[i].axvline(
            x=index,
            linestyle="--",
            color="r",
            label=f"{int(threshold * 100)}% Variance",
        )
        (
            axs[i].set_title(
                transformations_[i],
                fontweight="bold",
                fontsize=15,
            )
```

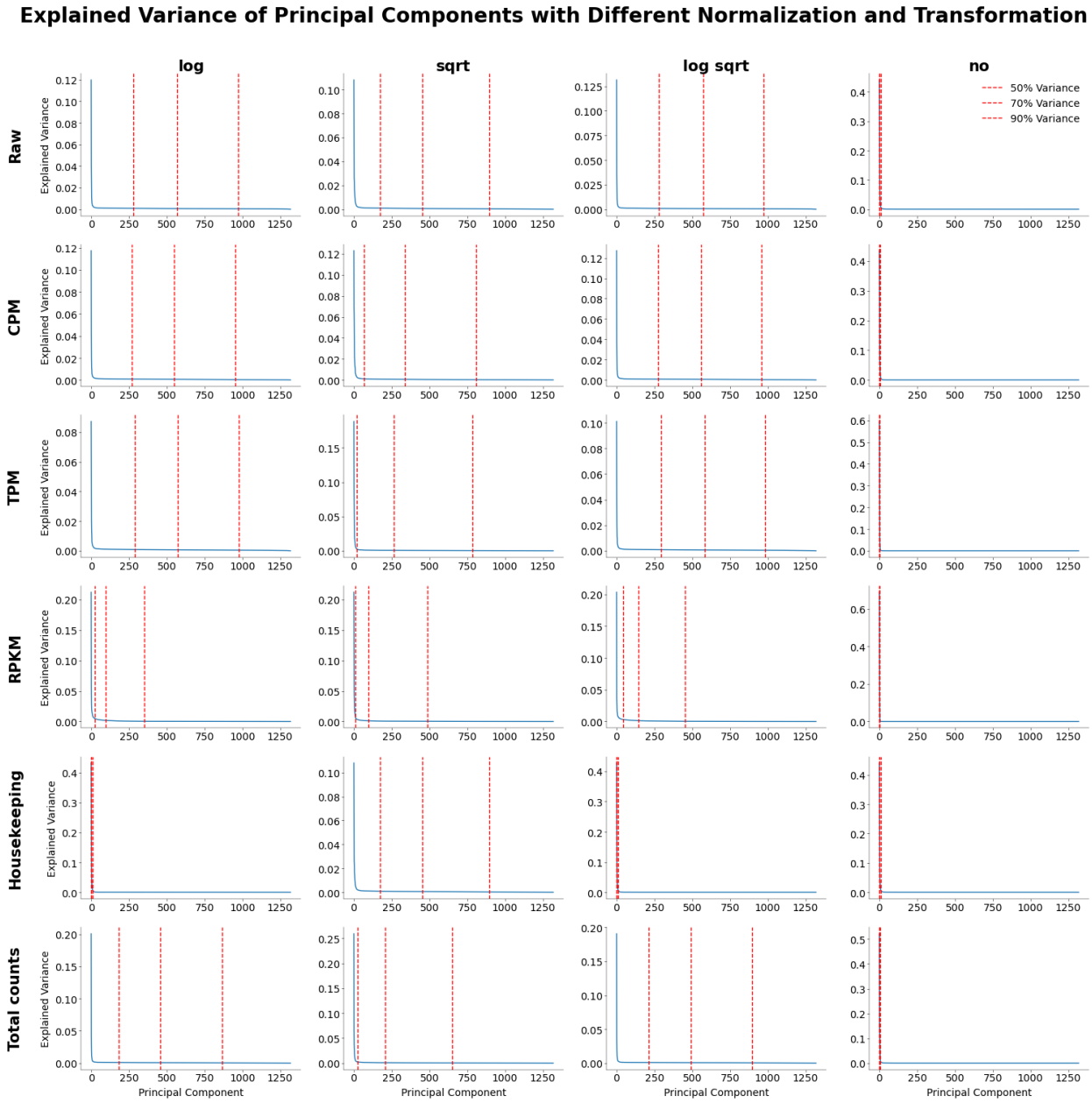
```
            if i < 4
            else None
        )

    (
        axs[i].text(
            -0.3,
            0.5,
            norm_methods_[i // 4],
            transform=axs[i].transAxes,
            fontsize=15,
            rotation=90,
            va="center",
            ha="center",
            fontweight="bold",
        )
        if i % 4 == 0
        else None
    )

    # axs[i].set_title(f"Explained Variance: {key[0]} Norm & {key[1]}
Trans")
    axs[i].set_xlabel("Principal Component") if i >= 20 else None
    axs[i].set_ylabel("Explained Variance") if i % 4 == 0 else None
    axs[i].legend() if i == 3 else None


plt.suptitle(
    "Explained Variance of Principal Components with Different
Normalization and Transformation",
    x=0.5,
    y=1.01,
    ha="center",
    fontsize=20,
    fontweight="bold",
)
plt.tight_layout()
plt.show()
```

```
c:\Users\Arne.Gittel\Documents\Neuro Master\Neural data science\
practicals\Final project\Final_Project\NDS_project\notebooks\
Utils.py:698: UserWarning: No data for colormapping provided via 'c'.
Parameters 'cmap' will be ignored
  scatter = ax.scatter(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\matplotlib\collections.py:1109: UserWarning: Collection
without array used. Make sure to specify the values to be colormapped
via the `c` argument.
  warnings.warn("Collection without array used. Make sure to "
C:\Users\Arne.Gittel\AppData\Local\Temp\
```

```
ipykernel_2668\1898982888.py:57: UserWarning: The figure layout has
changed to tight
  plt.tight_layout()
```




```
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\1898982888.py:123: UserWarning: The figure layout has
changed to tight
  plt.tight_layout()
```



In the overview we can see that the most efficient PCA projection is achieved by the log (or lig sqrt) transformation on the Housekeeping normalized data. As well as for, surprisingly, the untransformed data. We repeated the procedure for the subset of top 1000 genes by their Fano factor.

```
# get the 1000 highest fano factor genes from exoncounts
top_fano_factors, top_gene_expression =
calculate_top_fano_factor_array(
    exonCounts[keepcells, :], top_genes=1000
)

norm_data_fano = {"Raw": top_gene_expression}
```

```
print(top_gene_expression.shape)
pca_results_fano = {}


for i, n_method in enumerate(norm_methods):
    norm_exonCounts = normalize_count_data(
        data_exons, data_genes=gene_data, method=n_method
    )
    top_fano_factors, top_gene_expression =
calculate_top_fano_factor_array(
        norm_exonCounts, top_genes=1000
    )
    norm_data_fano[n_method] = top_gene_expression
    print("TGE:, ", top_gene_expression.shape)

    for j, t_method in enumerate(transform_methods):
        pca_norm, pca_norm_model = perform_pca(
            norm_data_fano[n_method][keepcells, :],
            transformation=t_method,
            standardize=False,
        )
        pca_results_fano[(n_method, transform_labels[j])] = pca_norm,
pca_norm_model


# Create subplots with 2 rows and 3 columns
fig, axs = plt.subplots(5, 4, figsize=(30, 30))

# Flatten the axs array for easier indexing
axs = axs.flatten()

# Iterate through the PCA results and plot the embeddings and
explained variance
for i, (key, value) in enumerate(pca_results_fano.items()):
    norm_methods = ["Raw", "CPM", "TPM", "RPKM", "Housekeeping",
"Total counts"]
    transformations = ["log", "sqrt", "log sqrt", "no"]
    # Plot PCA embedding
    plot_pca_embedding(
        value[0],
        labels=cluster_colors[keepcells],
        # title=f"PCA Embedding with {key[0]} Normalization and
{key[1]} Transformation, explained variance: PC1:
{value[1].explained_variance_ratio_[0]:.2f}, PC2:
{value[1].explained_variance_ratio_[1]:.2f}",
        # keepcells=keepcells,
        ax=axs[i],
    )
    (
```

```
        axs[i].text(
            -0.2,
            0.5,
            norm_methods[int(i // 4)],
            transform=axs[i].transAxes,
            fontsize=15,
            rotation=90,
            va="center",
            ha="center",
            fontweight="bold",
        )
        if i % 4 == 0
        else None
    )
    (
        axs[i].set_title(
            transformations[i],
            fontweight="bold",
            fontsize=15,
        )
        if i < 4
        else None
    )
plt.suptitle(
    "PCA Embeddings with Different Normalization and Transformation
Methods (Top 1000 Fano Factor Genes)",
    x=0.5,
    y=1.01,
    ha="center",
    fontsize=30,
    fontweight="bold",
)

plt.tight_layout()
plt.show()

fig, axs = plt.subplots(5, 4, figsize=(15, 15))
axs = axs.flatten()

# Iterate through the PCA results again and plot the explained
variance
for i, (key, value) in enumerate(pca_results_fano.items()):
    explained_variance_ratio = value[1]
    cumulative_variance = np.cumsum(explained_variance_ratio)

    # Plot the explained variance of all the principal components
    axs[i].plot(explained_variance_ratio)  # , marker='o',
label='Explained Variance')

    # Add vertical lines for 50%, 70%, and 90% cumulative variance
```

```
    for threshold in [0.5, 0.7, 0.9]:
        # Find the index where cumulative variance exceeds the
threshold
        index = np.argmax(cumulative_variance >= threshold)
        axs[i].axvline(
            x=index,
            linestyle="--",
            color="r",
            label=f"{int(threshold * 100)}% Variance",
        )

    # axs[i].set_title(f"Explained Variance: {key[0]} Norm & {key[1]}
Trans")
    axs[i].set_xlabel("Principal Component")
    axs[i].set_ylabel("Explained Variance")
    axs[i].legend()
    (
        axs[i].text(
            -0.3,
            0.5,
            norm_methods[i // 4],
            transform=axs[i].transAxes,
            fontsize=15,
            rotation=90,
            va="center",
            ha="center",
            fontweight="bold",
        )
        if i % 4 == 0
        else None
    )
    (
        axs[i].set_title(
            transformations[i],
            fontweight="bold",
            fontsize=15,
        )
        if i < 4
        else None
    )

# Adjust layout for better spacing
# set normalisation method as the column label

# set transformation method as the y-axis label
# set the title as the explained variance

plt.suptitle(
    "Explained Variance of Principal Components with Different
Normalization and Transformation",
```

```
    x=0.5,
    y=1.01,
    ha="center",
    fontsize=20,
    fontweight="bold",
)

plt.tight_layout()
plt.show()
```

```
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\690074641.py:80: RuntimeWarning: invalid value
encountered in divide
  fano_factor = variance_expression / mean_expression
```

```
(1320, 1000)
```

```
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\690074641.py:80: RuntimeWarning: invalid value
encountered in divide
  fano_factor = variance_expression / mean_expression
```

```
TGE:,  (1329, 1000)
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Performing PCA...
Transformation:  sqrt
Square root-transforming data...
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]
```

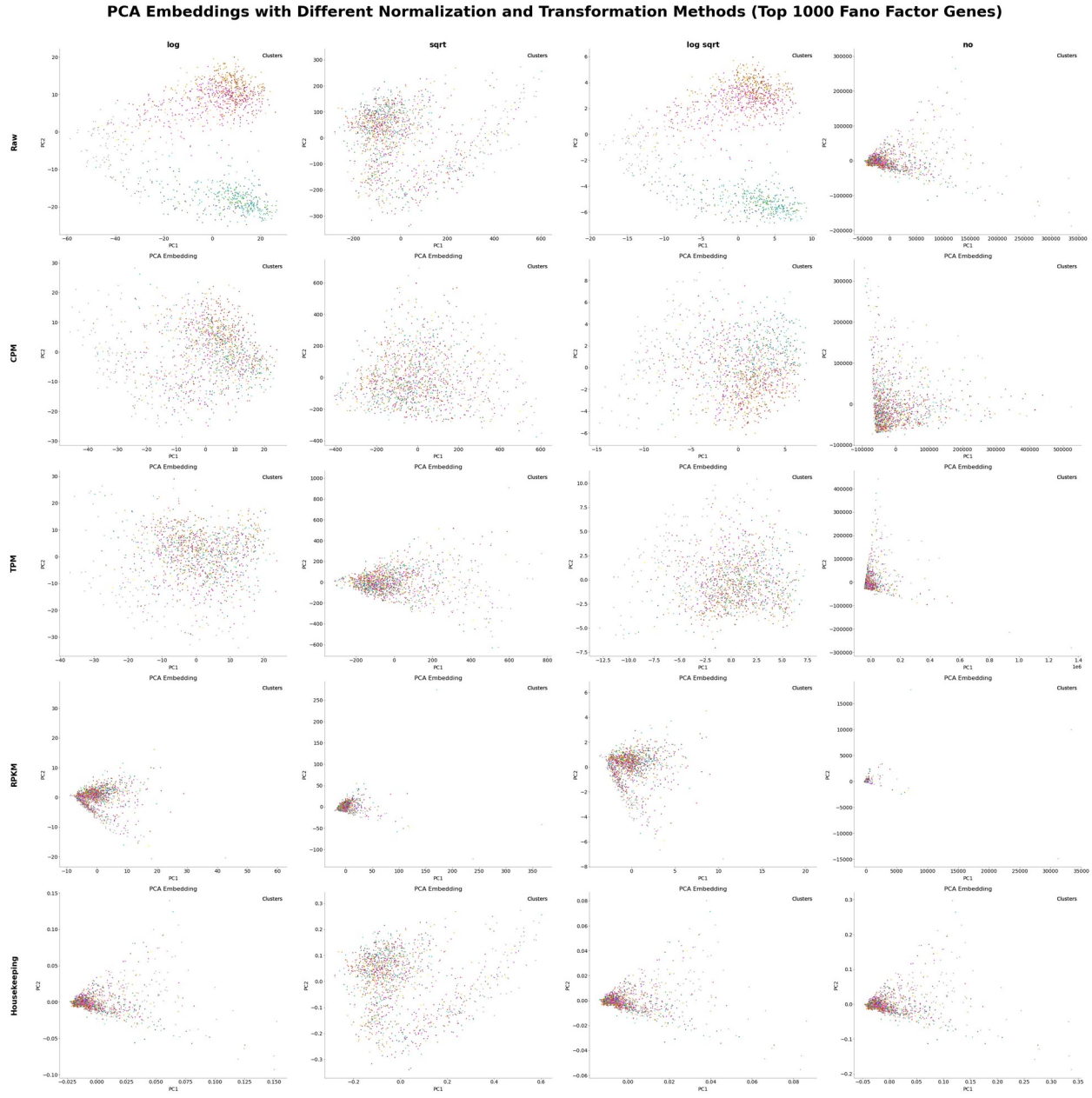
```
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\690074641.py:80: RuntimeWarning: invalid value
encountered in divide
  fano_factor = variance_expression / mean_expression
```

```
TGE:,  (1329, 1000)
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
```

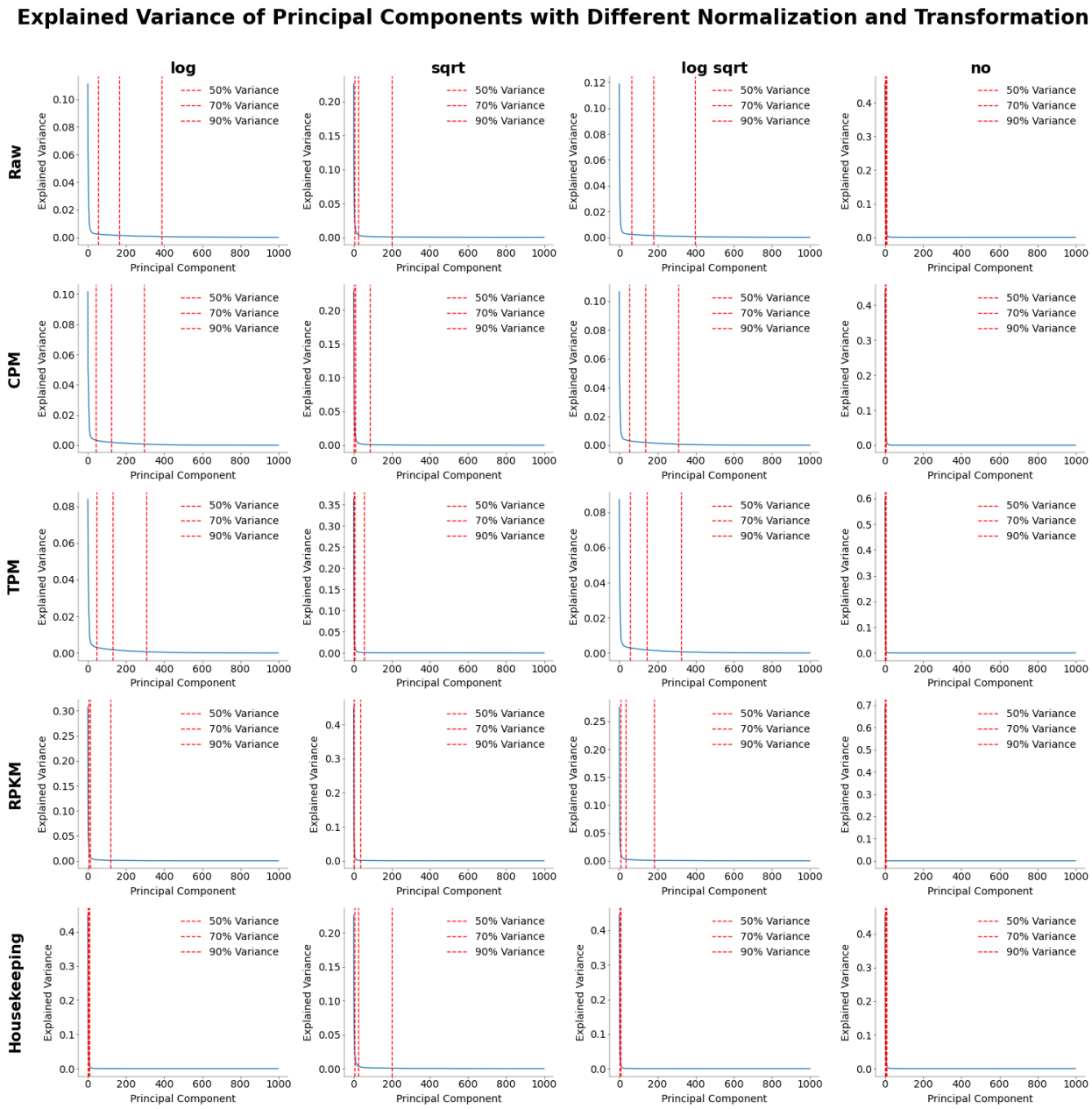
```
Performing PCA...
Transformation:  sqrt
Square root-transforming data...
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]

C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\690074641.py:80: RuntimeWarning: invalid value
encountered in divide
  fano_factor = variance_expression / mean_expression

TGE:,  (1329, 1000)
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Performing PCA...
Transformation:  sqrt
Square root-transforming data...
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]
Housekeeping genes: ['Actb', 'Gapdh', 'Ubc', 'Hprt'] Index(['Actb',
'Gapdh', 'Ubc', 'Hprt'], dtype='object', name='GeneID')
Warning: All housekeeping genes have zero counts in sample 58,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 73,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 132,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 409,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 419,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 504,
skipping normalization.
Warning: All housekeeping genes have zero counts in sample 1201,
skipping normalization.
```

```
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\690074641.py:80: RuntimeWarning: invalid value
encountered in divide
  fano_factor = variance_expression / mean_expression

TGE:,  (1329, 1000)
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Performing PCA...
Transformation:  sqrt
Square root-transforming data...
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]

C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\690074641.py:80: RuntimeWarning: invalid value
encountered in divide
  fano_factor = variance_expression / mean_expression

TGE:,  (1329, 1000)
Performing PCA...
Transformation:  log
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Performing PCA...
Transformation:  sqrt
Square root-transforming data...
Performing PCA...
Transformation:  log_sqrt
Log-transforming data...
Warning: log-transforming data with zero or negative values with a min
of 0.0. Adding small offset.
Square root-transforming data...
Performing PCA...
Transformation:  [None]

c:\Users\Arne.Gittel\Documents\Neuro Master\Neural data science\
practicals\Final project\Final_Project\NDS_project\notebooks\
Utils.py:698: UserWarning: No data for colormapping provided via 'c'.
Parameters 'cmap' will be ignored
```

```
  scatter = ax.scatter(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\matplotlib\collections.py:1109: UserWarning: Collection
without array used. Make sure to specify the values to be colormapped
via the `c` argument.
  warnings.warn("Collection without array used. Make sure to "
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\2414870450.py:82: UserWarning: The figure layout has
changed to tight
  plt.tight_layout()
```



```
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\2414870450.py:151: UserWarning: The figure layout has
changed to tight
  plt.tight_layout()
```



PCA embedding looks generally better. Like above, we can see that the most efficient PCA projection is achieved by the log (or lig sqrt) transformation on the Housekeeping normalized data.

```
for key, value in pca_results.items():
```

```
    # print the number of pcas that explain 70% of the variance for
each normalisation method
    print(
        f"Number of PCAs that explain 70% of the variance for {key[0]}
Normalization and {key[1]} Transformation: {np.sum(np.cumsum(value[1])
< 0.7)}"
    )

# print the method with the min number of pc that explain 70% of the
variance
min_pca = min(
    pca_results.items(),
    key=lambda x: np.sum(np.cumsum(x[1][1]) < 0.7),
)
print(
    f"Method with the min number of PCAs that explain 70% of the
variance: {min_pca[0][0]} Normalization and {min_pca[0][1]}
Transformation"
)
```

```
Number of PCAs that explain 70% of the variance for cpm Normalization
and log Transformation: 568
Number of PCAs that explain 70% of the variance for cpm Normalization
and sqrt Transformation: 454
Number of PCAs that explain 70% of the variance for cpm Normalization
and log_sqrt Transformation: 571
Number of PCAs that explain 70% of the variance for cpm Normalization
and no Transformation: 2
Number of PCAs that explain 70% of the variance for tpm Normalization
and log Transformation: 549
Number of PCAs that explain 70% of the variance for tpm Normalization
and sqrt Transformation: 335
Number of PCAs that explain 70% of the variance for tpm Normalization
and log_sqrt Transformation: 559
Number of PCAs that explain 70% of the variance for tpm Normalization
and no Transformation: 1
Number of PCAs that explain 70% of the variance for rpkm Normalization
and log Transformation: 574
Number of PCAs that explain 70% of the variance for rpkm Normalization
and sqrt Transformation: 263
Number of PCAs that explain 70% of the variance for rpkm Normalization
and log_sqrt Transformation: 582
Number of PCAs that explain 70% of the variance for rpkm Normalization
and no Transformation: 1
Number of PCAs that explain 70% of the variance for housekeeping
Normalization and log Transformation: 97
Number of PCAs that explain 70% of the variance for housekeeping
Normalization and sqrt Transformation: 95
Number of PCAs that explain 70% of the variance for housekeeping
Normalization and log_sqrt Transformation: 144
```

```
Number of PCAs that explain 70% of the variance for housekeeping
Normalization and no Transformation: 1
Number of PCAs that explain 70% of the variance for total_counts
Normalization and log Transformation: 2
Number of PCAs that explain 70% of the variance for total_counts
Normalization and sqrt Transformation: 454
Number of PCAs that explain 70% of the variance for total_counts
Normalization and log_sqrt Transformation: 2
Number of PCAs that explain 70% of the variance for total_counts
Normalization and no Transformation: 2
Number of PCAs that explain 70% of the variance for Raw Normalization
and log Transformation: 457
Number of PCAs that explain 70% of the variance for Raw Normalization
and sqrt Transformation: 208
Number of PCAs that explain 70% of the variance for Raw Normalization
and log_sqrt Transformation: 491
Number of PCAs that explain 70% of the variance for Raw Normalization
and no Transformation: 2
Method with the min number of PCAs that explain 70% of the variance:
tpm Normalization and no Transformation
```

```
for key, value in pca_results_fano.items():

    # print the number of pcas that explain 70% of the variance for
each normalisation method
    print(
        f"Number of PCAs that explain 70% of the variance for {key[0]}
Normalization and {key[1]} Transformation: {np.sum(np.cumsum(value[1])
< 0.7)}"
    )

# print the method with the min number of pc that explain 70% of the
variance
min_pca = min(
    pca_results_fano.items(),
    key=lambda x: np.sum(np.cumsum(x[1][1]) < 0.7),
)
print(
    f"Method with the min number of PCAs that explain 70% of the
variance: {min_pca[0][0]} Normalization and {min_pca[0][1]}
Transformation"
)
```

```
Number of PCAs that explain 70% of the variance for cpm Normalization
and log Transformation: 166
Number of PCAs that explain 70% of the variance for cpm Normalization
and sqrt Transformation: 24
Number of PCAs that explain 70% of the variance for cpm Normalization
and log_sqrt Transformation: 178
Number of PCAs that explain 70% of the variance for cpm Normalization
```

```
and no Transformation: 2
Number of PCAs that explain 70% of the variance for tpm Normalization
and log Transformation: 124
Number of PCAs that explain 70% of the variance for tpm Normalization
and sqrt Transformation: 8
Number of PCAs that explain 70% of the variance for tpm Normalization
and log_sqrt Transformation: 138
Number of PCAs that explain 70% of the variance for tpm Normalization
and no Transformation: 1
Number of PCAs that explain 70% of the variance for rpkm Normalization
and log Transformation: 129
Number of PCAs that explain 70% of the variance for rpkm Normalization
and sqrt Transformation: 4
Number of PCAs that explain 70% of the variance for rpkm Normalization
and log_sqrt Transformation: 146
Number of PCAs that explain 70% of the variance for rpkm Normalization
and no Transformation: 1
Number of PCAs that explain 70% of the variance for housekeeping
Normalization and log Transformation: 13
Number of PCAs that explain 70% of the variance for housekeeping
Normalization and sqrt Transformation: 3
Number of PCAs that explain 70% of the variance for housekeeping
Normalization and log_sqrt Transformation: 35
Number of PCAs that explain 70% of the variance for housekeeping
Normalization and no Transformation: 1
Number of PCAs that explain 70% of the variance for total_counts
Normalization and log Transformation: 2
Number of PCAs that explain 70% of the variance for total_counts
Normalization and sqrt Transformation: 24
Number of PCAs that explain 70% of the variance for total_counts
Normalization and log_sqrt Transformation: 2
Number of PCAs that explain 70% of the variance for total_counts
Normalization and no Transformation: 2
Method with the min number of PCAs that explain 70% of the variance:
tpm Normalization and no Transformation
```

```
print(ephysData_filtered.shape)
print(exonCounts[keepcells, :].shape)

# print pca result keys
print(pca_results.keys())

# print the pca results ('housekeeping', 'no')
print(pca_results[("housekeeping", "no")][0].shape)

# check if the pca results contain nan values
print(np.sum(np.isnan(pca_results[("housekeeping", "no")][0])))

# get normalised exon counts with housekeeping method
print(norm_data.keys())
```

```
norm_exonCounts = norm_data["housekeeping"][keepcells, :]


norm_exonCounts_fano = norm_data_fano["housekeeping"][keepcells, :]
# normalize_count_data(data_exons, data_genes=gene_data, method="cpm")
[
#     keepcells, :
# ]
# check if the normalised exon counts contain nan values
print(np.sum(np.isnan(norm_exonCounts)))
```

```
(1320, 17)
(1320, 42466)
dict_keys([('cpm', 'log'), ('cpm', 'sqrt'), ('cpm', 'log_sqrt'),
('cpm', 'no'), ('tpm', 'log'), ('tpm', 'sqrt'), ('tpm', 'log_sqrt'),
('tpm', 'no'), ('rpkm', 'log'), ('rpkm', 'sqrt'), ('rpkm',
'log_sqrt'), ('rpkm', 'no'), ('housekeeping', 'log'), ('housekeeping',
'sqrt'), ('housekeeping', 'log_sqrt'), ('housekeeping', 'no'),
('total_counts', 'log'), ('total_counts', 'sqrt'), ('total_counts',
'log_sqrt'), ('total_counts', 'no'), ('Raw', 'log'), ('Raw', 'sqrt'),
('Raw', 'log_sqrt'), ('Raw', 'no')])
(1320, 1320)
0
dict_keys(['Raw', 'cpm', 'tpm', 'rpkm', 'housekeeping',
'total_counts'])
0
```

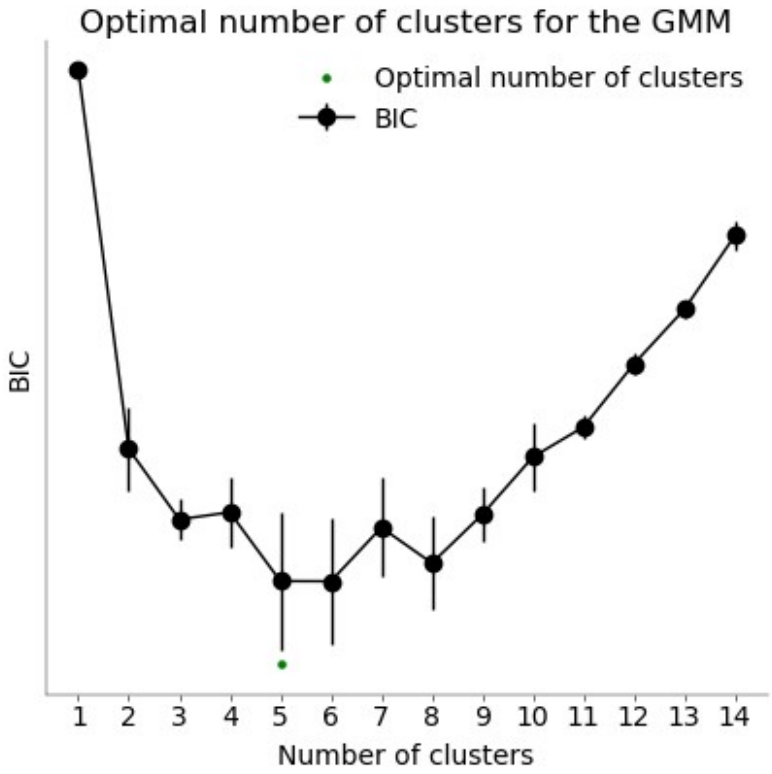
We tried the hierarchical clustering on top 1000 genes (by Fano factor) after normalization. We obtain 3 major clusters. Globally the clustering does not correspond very well to originally identified t-types. Only on a very local scale some parts are recovered.

Gaussian mixture model clustering of the transcriptomic data

```
import warnings

# set the range of possible cluster number
possible_clusters = np.arange(1, 15, 1)
num_seeds = 10

best_pcs = pca_results_fano[("rpkm", "log")][0][:, 0:15]
# get the optimal cluster number using BIC
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    Transcriptomic_cluster_n = get_optimal_cluster_number(
        possible_clusters,
        num_seeds,
        best_pcs,
        bic_mode="sklearn",
        plot=True,
    )
```

```
    # fit the GMM model
    Transcriptomic_gmm = GMM(
        n_components=Transcriptomic_cluster_n, random_state=42
    ).fit(best_pcs)

# get the cluster labels, means and covariances
Trans_clusters = Transcriptomic_gmm.predict(best_pcs)
Trans_clusters_means = Transcriptomic_gmm.means_
Trans_clusters_covariances = Transcriptomic_gmm.covariances_
```

```
Optimal number of clusters: 5
```




```
# get class labels
Transcriptomic_gmm_labels = Transcriptomic_gmm.predict(best_pcs)

reduced_data_rna = dim_reduction(norm_exonCounts,
cluster_colors[keepcells])
```

```
PCA accuracy: 0.09
PCA recall: 0.09
NCA accuracy: 0.05
NCA recall: 0.05
LDA accuracy: 0.05
LDA recall: 0.05
```

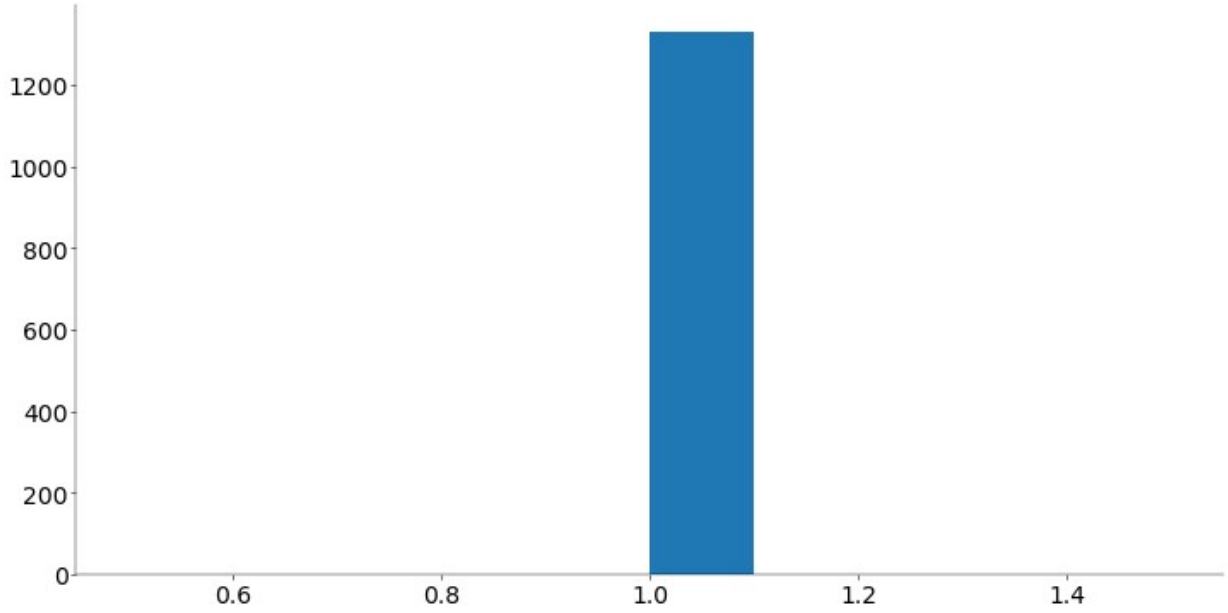
Negative binomial clustering

Our implementation of negative binomial mixture model only finds one cluster and is not clustering our data

```
# negative binomial clustering

MM_cluster_ids_fit, MM_means_fit, S, p = NBMM(4).fit_nbmm(exonCounts)


print("cluster ID", MM_cluster_ids_fit)
print("means", MM_means_fit.shape)
print("S:", S)
print("p", p)

plt.figure()
plt.hist(MM_cluster_ids_fit)
plt.show()
```

```
LL -793303.1280832111
0:327, 1:357, 2:336, 3:309
LL -1746.936224947197
0:0, 1:1329, 2:0, 3:0

c:\Users\Arne.Gittel\Documents\Neuro Master\Neural data science\
practicals\Final project\Final_Project\NDS_project\notebooks\
Utils.py:729: RuntimeWarning: divide by zero encountered in log
  return np.log(p) + np.sum([x[:, g] * np.log(p_gk[:,g]) + r *
np.log(1 - p_gk[:,g]) for g in S], axis=0)

LL -0.009992481026800327
0:0, 1:1329, 2:0, 3:0
cluster ID [1 1 1 ... 1 1 1]
means (4, 42466)
S: [19383 29494 17506 32947 42462    48 19239 16917 33627 33086 30959
8544
 33396 33394 16759 17672 18873 20356 18540 30923 18513 16771 41522
19952
  7889 33168 17100 32894 30900 32021 17740 15977 39179 20523 18579
17201
 32412 32165 16855  8338   467 16168 15984 15795 17127 17138 16052
1704
 33332 16041  1712  1464 11429 20522  6084 18010 16082 17606  7894
38656
 38655 31970 17699 17698 16018 17729  6536 19063  9727 30210 17263
16916
 16310  9836 31190 32608 19270 32542 20068  9982 28824 17307  4372
29660
 19591 33537  3079 18355 16151 15949 15799   622  3307  6782  3306
19671
```

```
 19077  8349 17627  1717]
p [0. 1. 0. 0.]
```




```
print(Trans_clusters_means.shape)
```

```
(5, 15)
```

```
# performing t-SNE on the ePhys data
Transcriptomic_tsne = TSNE(
    init="pca", early_exaggeration=4, random_state=42
).fit_transform(best_pcs)

Transcriptomic_umap =
umap.UMAP(random_state=42).fit_transform(best_pcs)
```

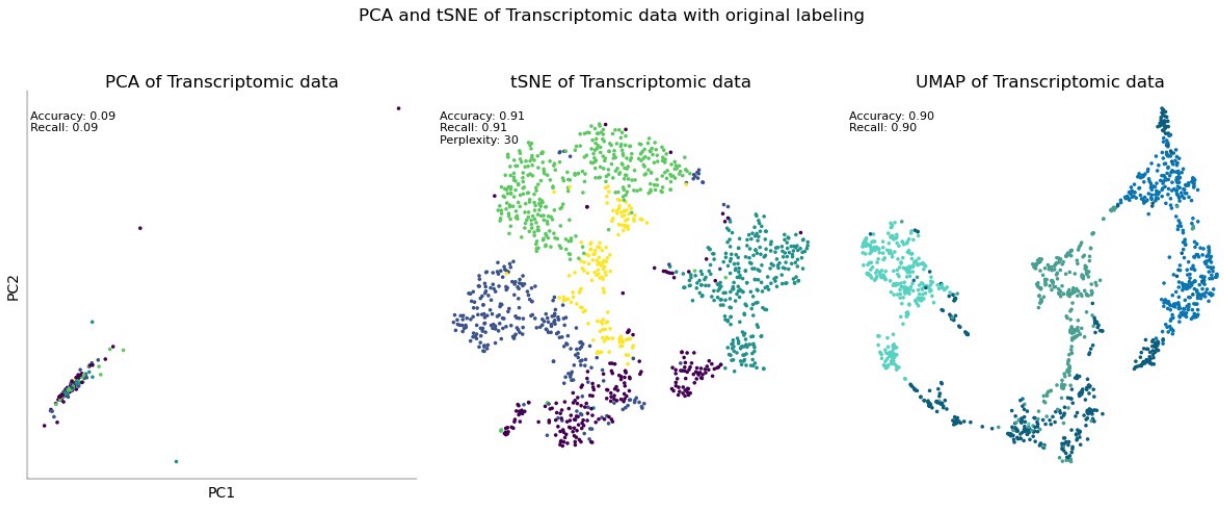
```
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\umap\umap_.py:1945: UserWarning: n_jobs value 1
overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting
random_state. Use no seed for parallelism.")
```

```
# pca_ePhys = pca_ePhys / np.std(pca_ePhys[:, 0]) * 0.0001

fig = plt.figure(figsize=(15, 8))

gs = GridSpec(2, 3, figure=fig, wspace=0.05, height_ratios=[2, 1])
# Create subplots with the specified gridspec
ax0 = fig.add_subplot(gs[0, 0])
ax1 = fig.add_subplot(gs[0, 1])
ax2 = fig.add_subplot(gs[0, 2])
# ax3 = fig.add_subplot(gs[1, :])
```

```
plot_dim1_dim2(
    ax0,
    reduced_data_rna,
    Transcriptomic_gmm_labels,
    "PCA of Transcriptomic data",
    type="PCA",
    display_stats=True,
)
plot_TSNE(
    ax1,
    Transcriptomic_tsne,
    Transcriptomic_gmm_labels,
    "tSNE of Transcriptomic data",
    display_accuracy=True,
)
plot_umap(
    ax2,
    Transcriptomic_umap,
    cluster_colors[Transcriptomic_gmm_labels],
    "UMAP of Transcriptomic data",
    display_accuracy=True,
)
# plot_cell_types(neuron_groups, ax3, base_font_size=8)

# plot_ellipse(
#     Trans_clusters_means,
#     Trans_clusters_covariances,
#     ax=ax0,
#     color="black",
#     cluster_n=Transcriptomic_cluster_n,
# )
ax0.legend()
plt.tight_layout()
plt.suptitle("PCA and tSNE of Transcriptomic data with original
labeling", fontsize=12)
```

```
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\4015354557.py:44: UserWarning: No artists with labels
found to put in legend.  Note that artists whose label start with an
underscore are ignored when legend() is called with no argument.
  ax0.legend()
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\4015354557.py:45: UserWarning: This figure includes
Axes that are not compatible with tight_layout, so results might be
incorrect.
  plt.tight_layout()
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\4015354557.py:45: UserWarning: The figure layout has
```

```
changed to tight
  plt.tight_layout()

Text(0.5, 0.98, 'PCA and tSNE of Transcriptomic data with original
labeling')
```



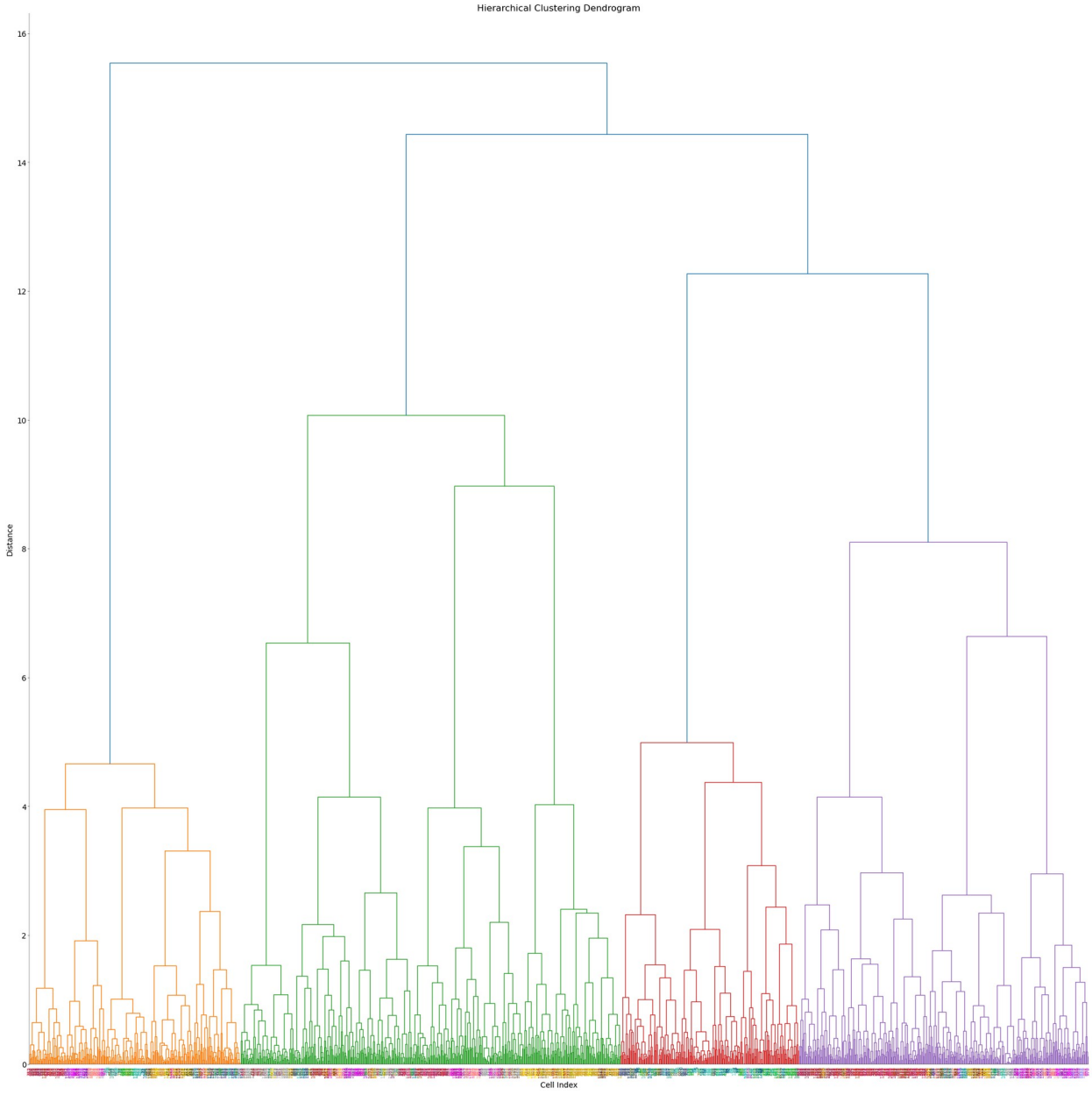
Hierarchical clustering using pearson distance

```
import numpy as np
import scipy.cluster.hierarchy as sch
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.spatial.distance import pdist, squareform


def pearson_distance(X):
    """
    Compute the Pearson distance matrix for a given array.

    Parameters:
    - X (np.ndarray): A 2D array with shape (cells, genes).

    Returns:
    - dist (np.ndarray): The Pearson distance matrix.
    """
    # Compute the correlation matrix
    corr = np.corrcoef(X)

    # Convert correlation to distance
    dist = 1 - corr

    # Ensure the distance matrix is condensed as required by scipy
linkage
    return squareform(dist, checks=False)
```

```
def hierarchical_clustering(rna_seq_array, cluster_colors,
method="ward", plot=True):
    """
    Perform hierarchical clustering on cells based on gene counts
using Pearson distance.

    Parameters:
    - rna_seq_array (np.ndarray): A 2D array with shape (cells,
genes).
    - cluster_colors (list or np.ndarray): Array of colors
corresponding to each sample.
    - method (str): Linkage method to use for clustering (default is
'ward').
    - plot (bool): If True, plot the dendrogram (default is True).

    Returns:
    - linkage_matrix (np.ndarray): The linkage matrix containing the
hierarchical clustering.
    """

    # Calculate the Pearson distance matrix
    distance_matrix = pearson_distance(rna_seq_array)

    # Perform hierarchical/agglomerative clustering
    linkage_matrix = sch.linkage(distance_matrix, method=method)

    if plot:
        # Plot the dendrogram
        plt.figure(figsize=(20, 20))
        dendro = sch.dendrogram(
            linkage_matrix,
            labels=np.arange(len(rna_seq_array)),
            leaf_rotation=90,
            leaf_font_size=8,
        )

        # Apply the colors to the labels
        ax = plt.gca()
        x_labels = ax.get_xmajorticklabels()
        for lbl in x_labels:
            lbl.set_color(cluster_colors[int(lbl.get_text())])
            lbl.set_fontsize(6)

        plt.title("Hierarchical Clustering Dendrogram")
        plt.xlabel("Cell Index")
        plt.ylabel("Distance")
        plt.show()

    return linkage_matrix
```

```
# Example usage with your data and cluster_colors array:
# hierarchical_clustering(norm_exonCounts_fano, cluster_colors)

print(cluster_colors[keepcells].shape, norm_exonCounts_fano.shape)
hierarchical_clustering(best_pcs,
cluster_colors=cluster_colors[keepcells])
```

```
(1320,) (1320, 1000)
```



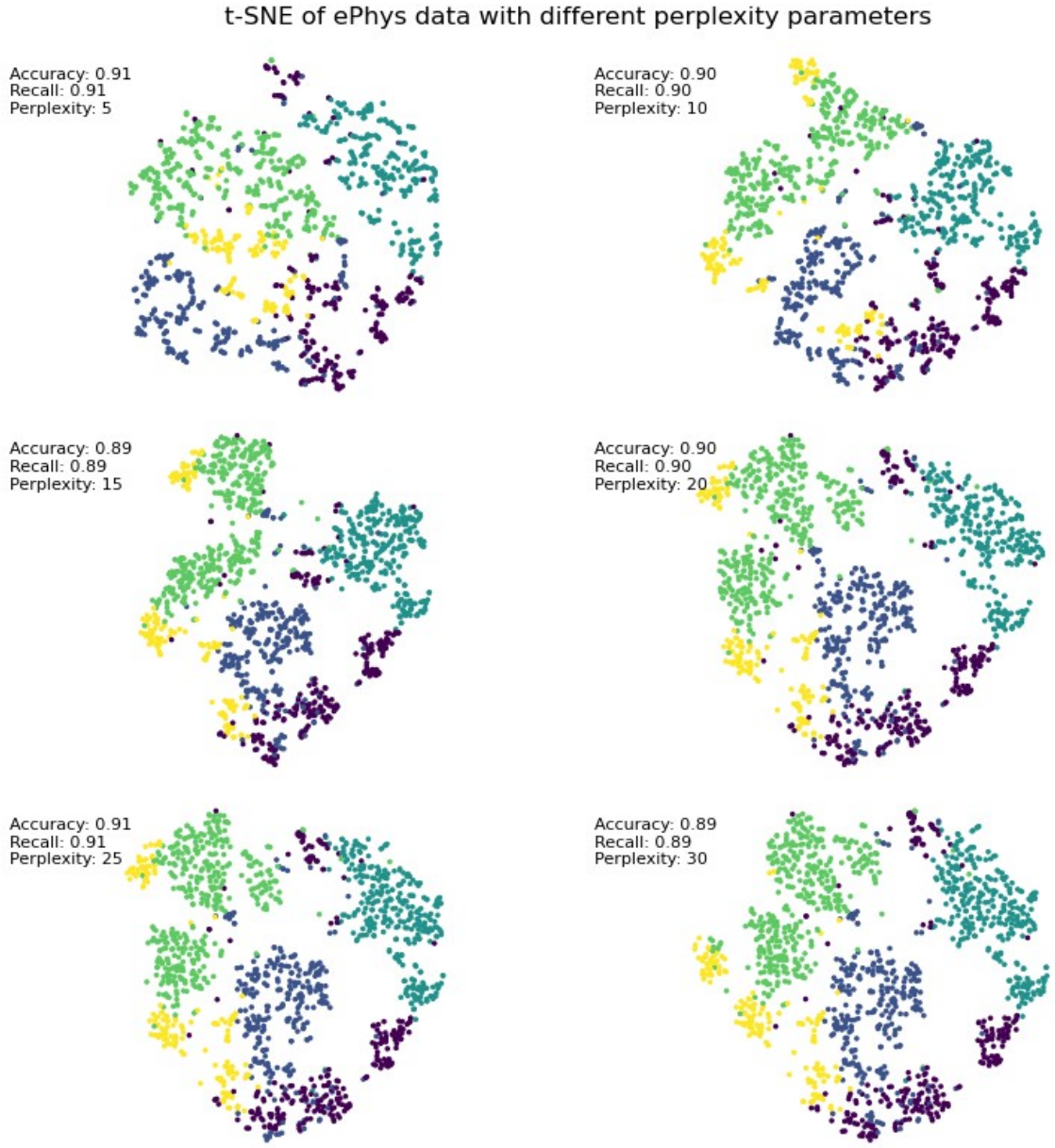
```
array([[1.36000000e+02, 1.73000000e+02, 3.50476973e-04,
2.00000000e+00],
       [7.40000000e+01, 9.80000000e+01, 1.85561387e-03,
2.00000000e+00],
       [1.57000000e+02, 1.65000000e+02, 2.27689761e-03,
2.00000000e+00],
       ...,
       [2.63000000e+03, 2.63300000e+03, 1.22683856e+01,
5.83000000e+02],
       [2.63500000e+03, 2.63600000e+03, 1.44359901e+01,
1.05700000e+03],
       [2.62900000e+03, 2.63700000e+03, 1.55387420e+01,
1.32000000e+03]])
```

The GMM yields 4 to 5 clusters when performed on the PCA-processed transcriptomic data. This is concordant with the hierarchical clustering performed above. However obtained clusters do not match well original clusters as presented in the paper.

```
# Initialize list to store t-SNE results
TSNES = []

# Define range of perplexity values to test
perplexities = np.arange(5, 35, 5)

# Calculate number of rows and columns for subplots grid
num_plots = len(perplexities)
ncols = 2
nrows = (num_plots + ncols - 1) // ncols  # Ensure enough rows for the
number of plots

# Set up subplots grid
fig, axs = plt.subplots(nrows, ncols, figsize=(8, 8))

# Flatten the axs array if it's 2D
if nrows * ncols > 1:
    axs = axs.flatten()

# Loop through perplexity values and perform t-SNE
for i, p in enumerate(perplexities):
    tsne = TSNE(perplexity=p, random_state=42).fit_transform(best_pcs)
    TSNES.append(tsne)
    ax = axs[i]
    plot_TSNE(
        ax,
        tsne,
        Transcriptomic_gmm_labels,
        title="",
        display_accuracy=True,
        perplexity=p,
```

```
    )

# Remove any empty subplots
for j in range(i + 1, len(axs)):
    fig.delaxes(axs[j])

# Show the plot

plt.suptitle("t-SNE of ePhys data with different perplexity
parameters", fontsize=12)
plt.tight_layout()
```

```
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\1495139830.py:40: UserWarning: The figure layout has
changed to tight
  plt.tight_layout()
```

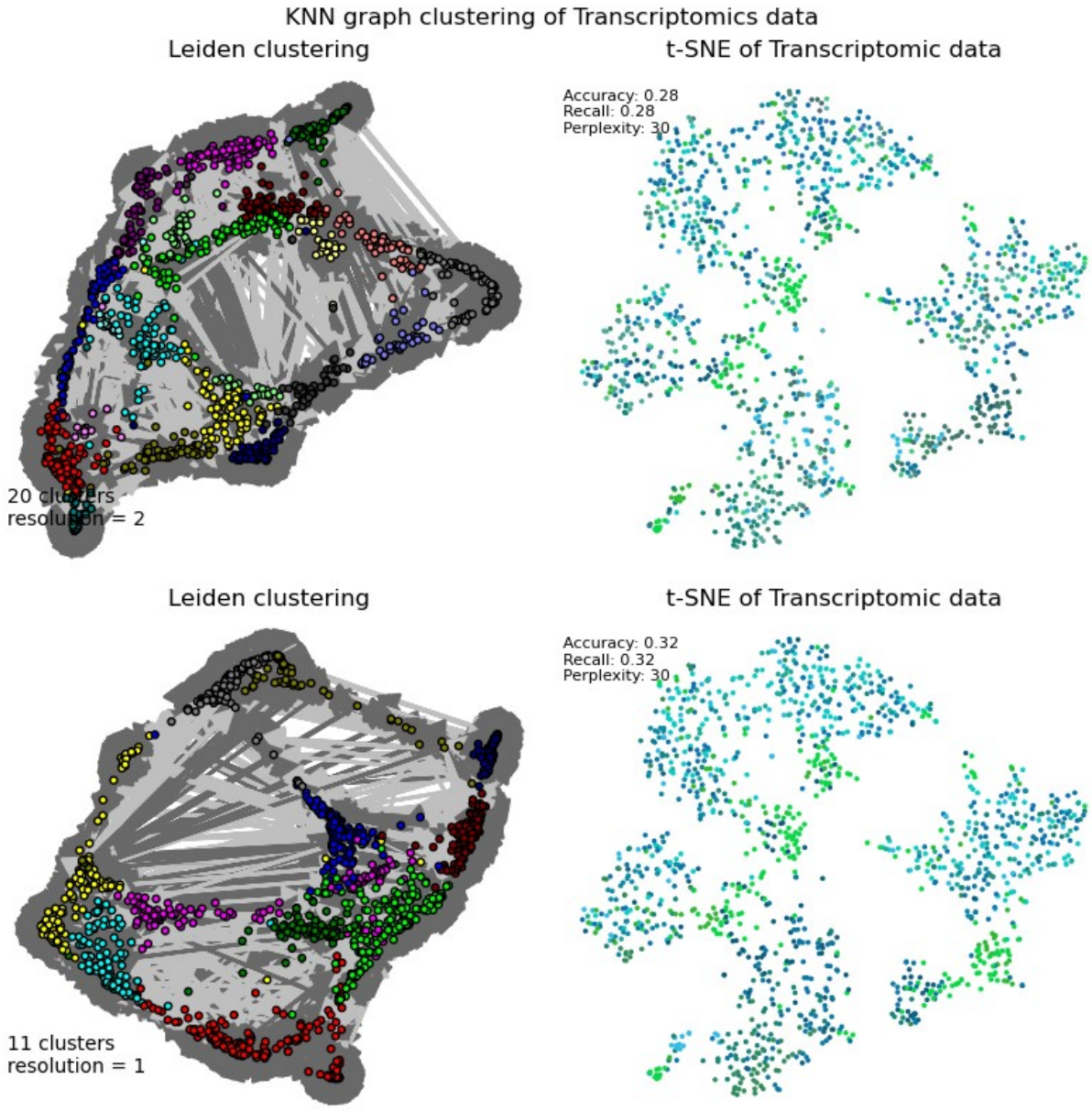


Here we run TSNE-embeddings with different perplexity parameters and assess accuracy with the help of k-nearest neighbors. The figure shows that according to the k-nearest neighbors classification, TSNE with the perplexity parameter 25 has the highest accuracy. This is also the default parameter, which was used in other plots in this study

K-nearest neigbhors clustering and Leiden graph visualization

```
import igraph as ig
from sklearn.neighbors import NearestNeighbors, kneighbors_graph
import leidenalg as la


clusterCols = np.unique(cluster_colors)
```

```
A = kneighbors_graph(norm_exonCounts, 10, mode="connectivity",
include_self=False)
sources, targets = A.nonzero()
G = ig.Graph(directed=True)
G.add_vertices(A.shape[0])
edges = list(zip(sources, targets))
G.add_edges(edges)

resolutions = [2, 1]

fig, ax = plt.subplots(2, 2, figsize=(8, 8))
for i in range(2):
    partition = la.find_partition(
        G,
        la.RBConfigurationVertexPartition,
        resolution_parameter=resolutions[i],
        seed=42,
    )

    ig.plot(
        partition,
        vertex_size=5,
        edge_curved=False,
        edgecolor="black",
        colors=clusterCols[partition.membership],
        target=ax[i][0],
        alpha=0.5,
    )
    ax[i][0].set_title("Leiden clustering")
    ax[i][0].text(
        0.025,
        0.1,
        f"{len(np.unique(partition.membership))} clusters\nresolution
= {resolutions[i]}",
        transform=ax[i][0].transAxes,
    )

    plot_TSNE(
        ax[i][1],
        Transcriptomic_tsne,
        clusterCols[partition.membership],
        "t-SNE of Transcriptomic data",
        display_accuracy=True,
    )
    plt.suptitle("KNN graph clustering of Transcriptomics data")
```



Leiden clustering finds 20 or 11 clusters depending on the resolution parameter.

Data analysis and visualisation

```
# Load morphology data
morphologyData = pd.read_csv(data_path /
"m1_patchseq_morph_features.csv")

data_exons.rename_axis("GeneID", axis="index", inplace=True)

# # Ensure 'GeneID' is in the columns or index of data_exons
# if "GeneID" in data_exons.columns:
#     data_exons.set_index("GeneID", inplace=True)
```

```
# elif "GeneID" in data_exons.index.names:
#     pass
# else:
#     raise KeyError(
#         "'GeneID' not found in the index or columns of the gene
expression dataframe."
#     )

# Filter gene expression dataframe to include only cells present in
the morphological dataframe
cells_with_morph_data = morphologyData["cell id"].unique()
df_genes_filtered = data_exons.loc[:, cells_with_morph_data]

# Split based on cell class
df_morph = morphologyData.set_index("cell id")
cell_classes = df_morph.loc[cells_with_morph_data, "cell class"]
excitatory_cells = cell_classes[cell_classes == "exc"].index
inhibitory_cells = cell_classes[cell_classes == "inh"].index

df_genes_excitatory = df_genes_filtered[excitatory_cells]
df_genes_inhibitory = df_genes_filtered[inhibitory_cells]

# Print shapes
print("Excitatory cells data shape:", df_genes_excitatory.shape)
print("Inhibitory cells data shape:", df_genes_inhibitory.shape)

# Calculate mean gene expression for each gene in each cell class
mean_gene_expression_excitatory = df_genes_excitatory.mean(axis=1)
mean_gene_expression_inhibitory = df_genes_inhibitory.mean(axis=1)

# Drop NA values
df_genes_excitatory = df_genes_excitatory.dropna()
df_genes_inhibitory = df_genes_inhibitory.dropna()

# Check for zero and NaN values
zero_values_excitatory =
df_genes_excitatory.columns[(df_genes_excitatory == 0).all()]
nan_values_excitatory =
df_genes_excitatory.columns[df_genes_excitatory.isnull().all()]
zero_values_inhibitory =
df_genes_inhibitory.columns[(df_genes_inhibitory == 0).all()]
nan_values_inhibitory =
df_genes_inhibitory.columns[df_genes_inhibitory.isnull().all()]

print("Excitatory cells with all zero values:",
zero_values_excitatory)
print("Excitatory cells with all NaN values:", nan_values_excitatory)
print("Inhibitory cells with all zero values:",
zero_values_inhibitory)
print("Inhibitory cells with all NaN values:", nan_values_inhibitory)
```

```
# Perform unpaired t-test for each gene
t_stat, p_val = ttest_ind(df_genes_excitatory.T,
df_genes_inhibitory.T, axis=0)

# Check for NaNs or Infs in p-values
if np.any(np.isnan(p_val)) or np.any(np.isinf(p_val)):
    print("P-values contain NaNs or Infs.")
    print("Number of NaNs in p-values:", np.sum(np.isnan(p_val)))
    print("Number of Infs in p-values:", np.sum(np.isinf(p_val)))


# Create a dataframe for the t-test results
t_test_results = pd.DataFrame(
    {
        "Gene": df_genes_filtered.index,
        "t_stat": t_stat,
        "p_val": p_val,
        "mean_expression_excitatory": mean_gene_expression_excitatory,
        "mean_expression_inhibitory": mean_gene_expression_inhibitory,
    }
)

# select all t_test results with p-value == NaN
nan_p_values = t_test_results[t_test_results["p_val"].isna()]
# print(nan_p_values)

# delete all rows with p-value  == NaN
t_test_results = t_test_results.dropna(subset=["p_val"])


# Correct for multiple testing on p-values from t_test_result df using
Benjamini-Hochberg method
_, pvals_corrected, _, _ = multipletests(t_test_results["p_val"],
method="fdr_bh")

# Add corrected p-values to the t_test_results dataframe for the
correct genes
t_test_results["p_val_corrected"] = pvals_corrected

# Print the first few rows of the t_test_results dataframe
print(t_test_results.head())
print(t_test_results.shape)

# print minimal p value
min_p_val = t_test_results["p_val"].min()
min_p_val_idx = t_test_results["p_val"].idxmin()
min_p_val_gene = t_test_results.loc[min_p_val_idx, "Gene"]

print(f"Minimal p-value: {min_p_val} for gene {min_p_val_gene}")
```

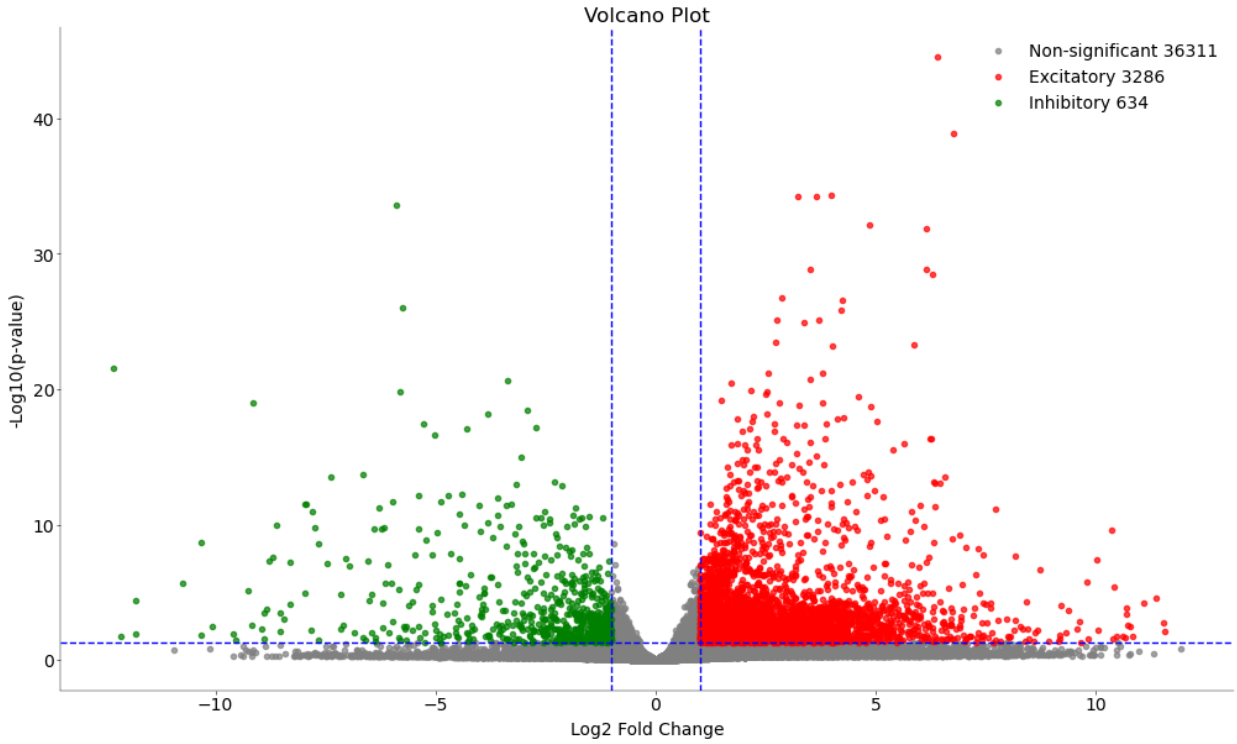
```
# print minimal corrected p value
min_p_val_corrected = t_test_results["p_val_corrected"].min()
min_p_val_corrected_idx = t_test_results["p_val_corrected"].idxmin()
min_p_val_corrected_gene = t_test_results.loc[min_p_val_corrected_idx,
"Gene"]

print(
    f"Minimal corrected p-value: {min_p_val_corrected} for gene
{min_p_val_corrected_gene}"
)


t_test_results.set_index("Gene", inplace=True)

# Ensure fold change is calculated correctly
t_test_results["fold_change"] = np.log2(
    t_test_results["mean_expression_excitatory"]
    / t_test_results["mean_expression_inhibitory"]
)

# Thresholds
fold_change_threshold = 1
p_val_threshold = 0.05

# Identify significant genes for excitatory and inhibitory cells
significant_genes_excitatory = t_test_results[
    (t_test_results["fold_change"] > fold_change_threshold)
    & (t_test_results["p_val_corrected"] < p_val_threshold)
]

significant_genes_inhibitory = t_test_results[
    (t_test_results["fold_change"] < -fold_change_threshold)
    & (t_test_results["p_val_corrected"] < p_val_threshold)
]

# Print the number of significant genes
print(
    "Number of significant genes (excitatory):",
significant_genes_excitatory.shape[0]
)
print(
    "Number of significant genes (inhibitory):",
significant_genes_inhibitory.shape[0]
)

# Plot the volcano plot
plt.figure(figsize=(10, 6))

# Plot non-significant genes
```

```
plt.scatter(
    t_test_results.loc[
        ~t_test_results.index.isin(significant_genes_excitatory.index)
        &
~t_test_results.index.isin(significant_genes_inhibitory.index),
        "fold_change",
    ],
    -np.log10(
        t_test_results.loc[

~t_test_results.index.isin(significant_genes_excitatory.index)
            &
~t_test_results.index.isin(significant_genes_inhibitory.index),
            "p_val_corrected",
        ]
    ),
    color="grey",
    alpha=0.7,
    label=f"Non-significant {t_test_results.shape[0] -
significant_genes_excitatory.shape[0] -
significant_genes_inhibitory.shape[0]}",
)

# Highlight significant genes (excitatory)
plt.scatter(
    significant_genes_excitatory["fold_change"],
    -np.log10(significant_genes_excitatory["p_val_corrected"]),
    color="red",
    alpha=0.7,
    label=f"Excitatory {significant_genes_excitatory.shape[0]}",
)

# Highlight significant genes (inhibitory)
plt.scatter(
    significant_genes_inhibitory["fold_change"],
    -np.log10(significant_genes_inhibitory["p_val_corrected"]),
    color="green",
    alpha=0.7,
    label=f"Inhibitory {significant_genes_inhibitory.shape[0]}",
)
# mark the top 10 genes with the highest significance and fold change
top_genes_excitatory = significant_genes_excitatory.nlargest(10,
"fold_change")
top_genes_inhibitory = significant_genes_inhibitory.nsmallest(10,
"fold_change")


# Adding labels and title
plt.xlabel("Log2 Fold Change")
plt.ylabel("-Log10(p-value)")
```

```
plt.title("Volcano Plot")

# Adding threshold lines
plt.axhline(y=-np.log10(p_val_threshold), color="blue",
linestyle="--")
plt.axvline(x=-fold_change_threshold, color="blue", linestyle="--")
plt.axvline(x=fold_change_threshold, color="blue", linestyle="--")

plt.legend()
plt.show()


# mark the top 10 genes with the highest significance and fold change
top_genes_excitatory = significant_genes_excitatory.nlargest(10,
"fold_change")
top_genes_inhibitory = significant_genes_inhibitory.nsmallest(10,
"fold_change")
```

```
Excitatory cells data shape: (42466, 275)
Inhibitory cells data shape: (42466, 371)
Excitatory cells with all zero values: Index([], dtype='object')
Excitatory cells with all NaN values: Index([], dtype='object')
Inhibitory cells with all zero values: Index([], dtype='object')
Inhibitory cells with all NaN values: Index([], dtype='object')
P-values contain NaNs or Infs.
Number of NaNs in p-values: 2235
Number of Infs in p-values: 0
                        Gene    t_stat     p_val
mean_expression_excitatory  \
GeneID

0610005C13Rik  0610005C13Rik  0.754662  0.450728
1.025455
0610006L08Rik  0610006L08Rik  1.161819  0.245740
0.047273
0610009B22Rik  0610009B22Rik  0.967456  0.333679
46.985455
0610009E02Rik  0610009E02Rik  1.444250  0.149155
2.992727
0610009L18Rik  0610009L18Rik  2.069276  0.038919
9.825455

               mean_expression_inhibitory  p_val_corrected
GeneID
0610005C13Rik                    0.711590         0.608217
0610006L08Rik                    0.000000         0.461727
0610009B22Rik                   38.398922         0.548541
0610009E02Rik                    1.644205         0.412388
0610009L18Rik                    4.517520         0.189740
(40231, 6)
```

```
Minimal p-value: 6.751770531145768e-50 for gene Slc17a7
Minimal corrected p-value: 2.716304802385254e-45 for gene Slc17a7
Number of significant genes (excitatory): 3286
Number of significant genes (inhibitory): 634

c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\pandas\core\arraylike.py:399: RuntimeWarning: divide by
zero encountered in log2
  result = getattr(ufunc, method)(*inputs, **kwargs)
```



Here we scan for the fraction of significantly active genes with significant fold change in mRNA in Excitatory (red) and Inhibitory (green cells) assessed by multiple testing correections using Benjamini - Hochberg method

```
# create a list of the gene names of the significant genes for
excitatory cells
significant_genes_excitatory_list =
significant_genes_excitatory.index.tolist()
print(significant_genes_excitatory_list)
# create a list of the gene names of the significant genes for
inhibitory cells
significant_genes_inhibitory_list =
significant_genes_inhibitory.index.tolist()

['1110002E22Rik', '1110008P14Rik', '1110018N20Rik', '1110032F04Rik',
'1600010M07Rik', '1700001K23Rik', '1700001L19Rik', '1700003D09Rik',
'1700007F19Rik', '1700016K19Rik', '1700019D03Rik', '1700021N21Rik',
```

```
'1700022I11Rik', '1700047F07Rik', '1700047G03Rik', '1700066M21Rik',
'1700101I11Rik', '1700105P06Rik', '1700110K17Rik', '1810037I17Rik',
'2010109A12Rik', '2010300C02Rik', '2310075C17Rik', '2410017I17Rik',
'2410018L13Rik', '2600014E21Rik', '2610306M01Rik', '2810029C07Rik',
'2810433D01Rik', '2810455O05Rik', '2810468N07Rik', '2900026A02Rik',
'2900060B14Rik', '2900079G21Rik', '3110082I17Rik', '4833412C05Rik',
'4921511C10Rik', '4921524J17Rik', '4921539H07Rik', '4930419G24Rik',
'4930426L09Rik', '4930447C04Rik', '4930447M23Rik', '4930449C09Rik',
'4930455M05Rik', '4930480K15Rik', '4930483P17Rik', '4930518C09Rik',
'4930533N22Rik', '4930556I23Rik', '4930578G10Rik', '4930590L20Rik',
'4930599N23Rik', '4932441J04Rik', '4933412E12Rik', '4933421D24Rik',
'4933425L06Rik', '4933427G23Rik', '4933432I09Rik', '4933432K03Rik',
'5031425E22Rik', '5330416C01Rik', '5530401A14Rik', '6330403K07Rik',
'6430562O15Rik', '8030423F21Rik', '8030453O22Rik', '8430429K09Rik',
'9130017K11Rik', '9130024F11Rik', '9130230N09Rik', '9330159F19Rik',
'9430037O13Rik', '9530059O14Rik', '9830132P13Rik', '9830144P21Rik',
'9830166K06Rik', '9930014A18Rik', 'A2ml1', 'A330009N23Rik',
'A330032B11Rik', 'A330072L02Rik', 'A430090L17Rik', 'A530076I17Rik',
'A630023P12Rik', 'A630072M18Rik', 'A830009L08Rik', 'A830036E02Rik',
'A830092H15Rik', 'A930001M01Rik', 'A930015D03Rik', 'A930030B08Rik',
'AC183097.1', 'AI115009', 'AL844494.1', 'AL844494.3', 'AU023762',
'AW822252', 'Aagab', 'Abca17', 'Abcg2', 'Abhd2', 'Abi3bp', 'Abracl',
'Acer2', 'Acta1', 'Actn1', 'Actn4', 'Actr3b', 'Acvr1c', 'Adad1',
'Adam18', 'Adamts2', 'Adcy9', 'Adcyap1', 'Adgrb1', 'Adgrb2', 'Adgrd1',
'Adgrg6', 'Adgrl2', 'Adora1', 'Adra1d', 'Adrb1', 'Afap1l1', 'Agbl4',
'Agr3', 'Agxt2', 'Aifm3', 'Ak3l2-ps', 'Ak4', 'Ak5', 'Ak8', 'Akap13',
'Akr1c19', 'Akr1d1', 'Akt2', 'Akt2-ps', 'Aldh1a3', 'Aldh1l1', 'Aldh2',
'Aldh3b1', 'Aldh3b3', 'Aldoa-ps4', 'Aldoart1', 'Aldoart2', 'Alms1-
ps1', 'Alms1-ps2', 'Alpk1', 'Amd-ps6', 'Amd-ps7', 'Amer1', 'Amer3',
'Amigo2', 'Ampd1', 'Angptl6', 'Ankrd13b', 'Ankrd33b', 'Ankrd34c',
'Ankrd36', 'Ankrd40', 'Ankrd63', 'Anks1b', 'Ano3', 'Antxr1', 'Anxa11',
'Anxa4', 'Anxa5', 'Aopep', 'Aox2', 'Ap1m2', 'Ap2s1', 'Apba2', 'Apbb1',
'Arap2', 'Arc', 'Arf3', 'Arg2', 'Arhgap1', 'Arhgap10', 'Arhgap17',
'Arhgap20', 'Arhgap20os', 'Arhgap25', 'Arhgap32', 'Arhgap33',
'Arhgap42', 'Arhgdia', 'Arhgef15', 'Arhgef25', 'Arhgef9', 'Arid3b',
'Arl4d', 'Arntl', 'Arpc3', 'Arpc5', 'Arpp19', 'Arpp21', 'Arr3',
'Asap1', 'Asap2', 'Asb11', 'Asb5', 'Ascl5', 'Aspg', 'Atad2', 'Atf6',
'Atl1', 'Atox1', 'Atp10a', 'Atp1a1', 'Atp23', 'Atp2b1', 'Atp2b4',
'Atp5e', 'Atp5k', 'Atp5k-ps2', 'Atp5l2-ps', 'Atp8b1', 'Axin2',
'B230208B08Rik', 'B230334C09Rik', 'B230369F24Rik', 'B3galt2',
'B3gnt2', 'B530045E10Rik', 'BC006965', 'BC037704', 'BC048507',
'Bace2', 'Baiap2', 'Basp1', 'Batf2', 'Baz1a', 'Bbs9', 'Bc1', 'Bcl10',
'Bcl11a', 'Bcl2', 'Bcor', 'Bcorl1', 'Bdnf', 'Bhlhe22', 'Bhlhe40',
'Bicc1', 'Bin3', 'Blnk', 'Bmi1', 'Bmp1', 'Bmpr1b', 'Bmt2', 'Boc',
'Bok', 'Bola2', 'Bop1', 'Brinp1', 'Brk1', 'Brpf3', 'Brsk2', 'Bsn',
'Btbd10', 'Btbd19', 'Btbd9', 'Btg2', 'C030004G16Rik', 'C130074G19Rik',
'C1ql3', 'C1qtnf12', 'C1ra', 'C1rb', 'C1s1', 'C230062I16Rik',
'C330002G04Rik', 'C4b', 'C530025M09Rik', 'C630043F03Rik', 'C7',
'C730002L08Rik', 'C77080', 'CT009536.1', 'Cabp1', 'Cacna1e',
```

'Cacna1h', 'Cacna2d1', 'Cacnb1', 'Cacnb3', 'Cacng3', 'Cadps2',
'Calm1', 'Calm2', 'Calm5', 'Camk2a', 'Camk4', 'Camkk1', 'Camkk2',
'Camkv', 'Cap1', 'Cap2', 'Capn1', 'Capn12', 'Car1', 'Car10', 'Car12',
'Car7', 'Catspere2', 'Catsperz', 'Cav1', 'Cavin3', 'Cbfa2t3', 'Cblif',
'Ccdc112', 'Ccdc113', 'Ccdc116', 'Ccdc155', 'Ccdc181', 'Ccdc24',
'Ccdc27', 'Ccdc3', 'Ccdc34', 'Ccdc7b', 'Ccdc88c', 'Cck', 'Cckbr',
'Ccl19', 'Ccl21a', 'Ccl21b', 'Ccl27a', 'Ccl27b', 'Ccn1', 'Ccn2',
'Ccn4', 'Ccnb1ip1', 'Ccnd1', 'Ccnd2', 'Ccnjl', 'Ccno', 'Ccnyl1',
'Ccpg1os', 'Ccsap', 'Cd248', 'Cd34', 'Cd6', 'Cd7', 'Cdc40', 'Cdh11',
'Cdh22', 'Cdh6', 'Cdk14', 'Cdk17', 'Cdk18', 'Cdk2ap1', 'Cdkl4',
'Cdkl5', 'Cdkn1a', 'Cdv3-ps', 'Cdyl2', 'Cebpb', 'Cecr2', 'Celf2',
'Celf4', 'Celf5', 'Cemip2', 'Cenpw', 'Cenpx', 'Cep85l', 'Cep89',
'Cerkl', 'Cetn2', 'Cetn4', 'Cfap100', 'Cfap206', 'Cfap44', 'Cfap45',
'Cfap54', 'Cfap65', 'Cfl1', 'Chchd6', 'Chd1l', 'Chn1', 'Chn1os3',
'Chrd', 'Chrm1', 'Chrm4', 'Chrna1', 'Chst1', 'Chst8', 'Chsy3', 'Cinp',
'Cisd3', 'Cish', 'Cited2', 'Cited4', 'Ckb-ps1', 'Ckb-ps2', 'Clba1',
'Clca3a2', 'Clec18a', 'Cmip', 'Cmpk2', 'Cmya5', 'Cnih3', 'Cnksr2',
'Cnksr3', 'Cntln', 'Coa4', 'Coasy', 'Cobl', 'Coch', 'Col12a1',
'Col1a1', 'Col22a1', 'Col4a1', 'Col5a1', 'Col6a1', 'Col6a2', 'Cop1',
'Coq10b', 'Coro1a', 'Coro2a', 'Cox20b', 'Cox6c2', 'Cphx3', 'Cplx2',
'Cpne5', 'Cpne8', 'Cpne9', 'Cpped1', 'Creg2', 'Crhr1', 'Crim1',
'Crip2', 'Crtc1', 'Cryab', 'Crybb3', 'Crybg2', 'Crym', 'Csgalnact1',
'Csmd2', 'Csmd2os', 'Csnk2a1-ps', 'Cspg5', 'Cst6', 'Cstad', 'Cstdc4',
'Ctla4', 'Ctnnal1', 'Ctnnd1', 'Ctsk', 'Ctxn1', 'Cutc', 'Cwh43',
'Cxcl12', 'Cxxc5', 'Cyb5a', 'Cycs-ps2', 'Cyp11a1', 'Cyp2ab1',
'Cyp3a13', 'Cyren', 'Cysrt1', 'D030068K23Rik', 'D130043K22Rik',
'D430041D05Rik', 'D630045J12Rik', 'D930019O06Rik', 'Dab2ip', 'Dact2',
'Dapk2', 'Dbf4', 'Dcbld1', 'Dclk3', 'Dclre1a', 'Dctpp1', 'Ddit4l',
'Ddn', 'Dennd3', 'Dennd4a', 'Deptor', 'Dgat2', 'Dgcr6', 'Dgkb',
'Dgkh', 'Dgkz', 'Diaph2', 'Diaph3', 'Diras2', 'Dkk3', 'Dkkl1', 'Dlc1',
'Dlg1', 'Dlgap4', 'Dlk2', 'Dmkn', 'Dnah12', 'Dnah17', 'Dnah9',
'Dnajb1', 'Dnajb5', 'Dnajb7', 'Dnajc21', 'Dnajc25', 'Doc2a', 'Dpf1',
'Dpm3', 'Dpp10', 'Dpp4', 'Dpy19l1', 'Drc3', 'Dsel', 'Dtl', 'Dtx4',
'Duoxa1', 'Dusp14', 'Dusp18', 'Dusp3', 'Dusp5', 'Dusp6', 'Dusp7',
'Dynll1', 'E030013I19Rik', 'E2f1', 'E330009J07Rik', 'ERCC-00051',
'Echdc2', 'Ecm1', 'Edn2', 'Eef1a2', 'Efcab1', 'Efcab12', 'Efcab6',
'Efcc1', 'Efna1', 'Efna2', 'Efna3', 'Efna5', 'Efnb2', 'Egfl7', 'Egr1',
'Egr2', 'Egr3', 'Egr4', 'Eif1-ps2', 'Eif1-ps3', 'Eif2b3', 'Eif3s6-
ps3', 'Eif4ebp1', 'Eif5al3-ps', 'Elavl4', 'Elk4', 'Elmo1', 'Elmo2',
'Elp6', 'Emb', 'Eme1', 'Emx1', 'Enc1', 'Enpp2', 'Enthd1', 'Epha4',
'Epha7', 'Ephb1', 'Ephb2', 'Ephb3', 'Ephb6', 'Ephx1', 'Ephx4', 'Epn2',
'Epop', 'Epor', 'Erc2', 'Erf', 'Erlin1', 'Esam', 'Etnk2', 'Ets2',
'Etv1', 'Etv5', 'Exd2', 'Exoc6', 'Exph5', 'Ext1', 'Extl1', 'F10',
'F630206G17Rik', 'Fabp5l2', 'Fam129b', 'Fam131a', 'Fam131b', 'Fam136b-
ps', 'Fam155a', 'Fam160a2', 'Fam163b', 'Fam189a1', 'Fam205a2',
'Fam205a3', 'Fam205a4', 'Fam205c', 'Fam20a', 'Fam221b', 'Fam241a',
'Fam241b', 'Fam76a', 'Fam81a', 'Fancd2', 'Fancl', 'Fank1', 'Fap',
'Fat3', 'Fat4', 'Fbxl19', 'Fbxo27', 'Fbxo32', 'Fbxo33', 'Fbxw7',
'Fcho2', 'Fcor', 'Fdft1', 'Fermt2', 'Fezf2', 'Ffar1', 'Fgf5', 'Fhad1',

'Fhl2', 'Fhod3', 'Fjx1', 'Fkbp1a', 'Fkbp1a-ps1', 'Flg', 'Flot2',
'Flrt1', 'Flrt2', 'Fmnl1', 'Fosl2', 'Foxg1', 'Foxk1', 'Foxp1',
'Foxp2', 'Foxp4', 'Fras1', 'Frmd6', 'Frrs1l', 'Frzb', 'Fscn1', 'Fst',
'Fstl4', 'Fxyd1', 'Fxyd5', 'Fxyd7', 'Fyb2', 'Fzd3', 'G2e3',
'G430095P16Rik', 'Gabra2', 'Gabra5', 'Gabrb1', 'Gabrg1', 'Gadd45a',
'Gadd45b', 'Gadd45g', 'Galnt16', 'Galnt17', 'Galnt18', 'Galnt9',
'Gap43', 'Gar1', 'Gas7', 'Gata3', 'Gca', 'Gcnt2', 'Gcnt4', 'Gda',
'Gdpd5', 'Gfra2', 'Gfra4', 'Gipc1', 'Git1', 'Glns-ps1', 'Glp2r',
'Glt8d2', 'Gm10012', 'Gm10020', 'Gm10033', 'Gm10043', 'Gm10054',
'Gm10076', 'Gm10080', 'Gm10086', 'Gm10088', 'Gm10126', 'Gm10154',
'Gm10167', 'Gm10180', 'Gm10237', 'Gm10263', 'Gm10312', 'Gm10334',
'Gm10358', 'Gm10382', 'Gm10421', 'Gm10432', 'Gm10443', 'Gm10540',
'Gm10561', 'Gm10570', 'Gm10591', 'Gm10593', 'Gm10601', 'Gm10635',
'Gm10754', 'Gm10784', 'Gm10819', 'Gm10827', 'Gm10874', 'Gm10913',
'Gm10916', 'Gm11203', 'Gm11221', 'Gm11223', 'Gm11227', 'Gm11242',
'Gm11243', 'Gm11245', 'Gm11246', 'Gm11270', 'Gm11285', 'Gm11307',
'Gm11343', 'Gm11363', 'Gm11367', 'Gm11384', 'Gm11405', 'Gm11410',
'Gm11416', 'Gm11417', 'Gm11418', 'Gm11425', 'Gm11438', 'Gm11443',
'Gm11453', 'Gm11456', 'Gm11460', 'Gm11488', 'Gm11500', 'Gm11549',
'Gm11557', 'Gm11578', 'Gm11581', 'Gm11582', 'Gm11598', 'Gm11599',
'Gm11604', 'Gm11619', 'Gm11625', 'Gm11633', 'Gm11689', 'Gm11759',
'Gm11762', 'Gm11794', 'Gm11803', 'Gm11814', 'Gm11826', 'Gm11836',
'Gm11857', 'Gm11873', 'Gm11888', 'Gm11902', 'Gm11910', 'Gm11912',
'Gm11913', 'Gm11930', 'Gm11949', 'Gm11950', 'Gm11956', 'Gm11980',
'Gm12002', 'Gm12003', 'Gm12005', 'Gm12009', 'Gm12020', 'Gm12024',
'Gm12027', 'Gm12031', 'Gm12034', 'Gm12035', 'Gm12038', 'Gm12109',
'Gm12129', 'Gm12151', 'Gm12155', 'Gm12168', 'Gm12201', 'Gm12216',
'Gm12230', 'Gm12242', 'Gm12285', 'Gm12318', 'Gm12328', 'Gm12337',
'Gm12341', 'Gm12362', 'Gm12371', 'Gm12389', 'Gm12394', 'Gm12419',
'Gm12428', 'Gm12453', 'Gm12460', 'Gm12466', 'Gm12468', 'Gm12469',
'Gm12492', 'Gm12498', 'Gm12504', 'Gm12513', 'Gm12535', 'Gm12537',
'Gm12565', 'Gm12587', 'Gm12632', 'Gm12643', 'Gm12666', 'Gm12697',
'Gm12722', 'Gm12728', 'Gm12732', 'Gm12733', 'Gm12751', 'Gm12770',
'Gm12838', 'Gm12859', 'Gm12868', 'Gm12895', 'Gm12896', 'Gm12900',
'Gm12901', 'Gm12920', 'Gm12936', 'Gm12963', 'Gm12967', 'Gm12983',
'Gm12986', 'Gm12988', 'Gm13006', 'Gm13009', 'Gm13015', 'Gm13029',
'Gm13047', 'Gm13075', 'Gm13077', 'Gm13092', 'Gm13121', 'Gm13149',
'Gm13186', 'Gm13196', 'Gm13215', 'Gm13216', 'Gm13249', 'Gm13268',
'Gm13303', 'Gm13306', 'Gm13320', 'Gm13321', 'Gm13339', 'Gm13342',
'Gm13359', 'Gm13365', 'Gm13384', 'Gm13389', 'Gm13407', 'Gm13433',
'Gm13437', 'Gm13489', 'Gm13494', 'Gm13500', 'Gm13503', 'Gm13532',
'Gm13541', 'Gm13545', 'Gm13549', 'Gm13572', 'Gm13588', 'Gm13600',
'Gm13614', 'Gm13623', 'Gm13679', 'Gm13684', 'Gm13690', 'Gm13716',
'Gm13726', 'Gm13736', 'Gm13743', 'Gm13744', 'Gm13754', 'Gm13768',
'Gm13770', 'Gm13771', 'Gm13777', 'Gm13803', 'Gm13809', 'Gm13810',
'Gm13812', 'Gm13842', 'Gm13862', 'Gm13870', 'Gm13872', 'Gm13875',
'Gm13879', 'Gm13915', 'Gm13921', 'Gm13935', 'Gm13977', 'Gm13981',
'Gm14000', 'Gm14008', 'Gm14037', 'Gm14039', 'Gm14053', 'Gm14094',
'Gm14101', 'Gm14107', 'Gm14108', 'Gm14131', 'Gm14140', 'Gm14156',

'Gm14173', 'Gm14214', 'Gm14236', 'Gm14245', 'Gm14263', 'Gm14267',
'Gm14269', 'Gm14277', 'Gm14278', 'Gm14285', 'Gm14303', 'Gm14328',
'Gm14472', 'Gm14480', 'Gm14506', 'Gm14512', 'Gm14513', 'Gm14516',
'Gm14529', 'Gm14532', 'Gm14536', 'Gm14539', 'Gm14567', 'Gm14612',
'Gm14630', 'Gm14633', 'Gm14636', 'Gm14648', 'Gm14676', 'Gm14706',
'Gm14723', 'Gm14753', 'Gm14775', 'Gm14794', 'Gm14807', 'Gm14813',
'Gm14830', 'Gm14840', 'Gm14856', 'Gm14859', 'Gm14881', 'Gm14892',
'Gm14897', 'Gm14925', 'Gm14932', 'Gm14984', 'Gm14989', 'Gm15026',
'Gm15066', 'Gm15067', 'Gm15148', 'Gm15151', 'Gm15156', 'Gm15178',
'Gm15190', 'Gm15196', 'Gm15198', 'Gm15212', 'Gm15238', 'Gm15294',
'Gm15349', 'Gm15365', 'Gm15393', 'Gm15403', 'Gm15408', 'Gm15411',
'Gm15432', 'Gm15467', 'Gm15493', 'Gm15512', 'Gm15532', 'Gm15536',
'Gm15568', 'Gm15575', 'Gm15609', 'Gm15621', 'Gm15638', 'Gm15707',
'Gm15721', 'Gm15728', 'Gm15737', 'Gm15738', 'Gm15794', 'Gm15810',
'Gm15812', 'Gm15912', 'Gm15928', 'Gm15974', 'Gm16000', 'Gm16005',
'Gm16020', 'Gm16056', 'Gm16061', 'Gm16072', 'Gm16089', 'Gm16103',
'Gm16106', 'Gm16148', 'Gm16183', 'Gm16214', 'Gm16335', 'Gm16339',
'Gm16351', 'Gm16372', 'Gm16373', 'Gm16379', 'Gm16385', 'Gm16412',
'Gm16416', 'Gm16418', 'Gm16423', 'Gm16494', 'Gm16505', 'Gm16581',
'Gm16582', 'Gm16897', 'Gm17056', 'Gm17138', 'Gm1715', 'Gm17150',
'Gm17151', 'Gm17167', 'Gm17212', 'Gm17322', 'Gm17334', 'Gm17349',
'Gm17501', 'Gm17511', 'Gm17541', 'Gm17799', 'Gm17802', 'Gm17816',
'Gm17819', 'Gm17831', 'Gm17891', 'Gm18024', 'Gm18076', 'Gm18078',
'Gm18079', 'Gm18103', 'Gm18119', 'Gm18134', 'Gm18169', 'Gm18406',
'Gm18432', 'Gm18555', 'Gm18671', 'Gm18705', 'Gm18735', 'Gm18769',
'Gm18859', 'Gm18867', 'Gm18892', 'Gm18894', 'Gm18904', 'Gm18959',
'Gm19141', 'Gm19183', 'Gm19243', 'Gm19246', 'Gm19399', 'Gm19410',
'Gm1943', 'Gm19503', 'Gm19563', 'Gm19566', 'Gm19587', 'Gm19620',
'Gm19739', 'Gm1976', 'Gm19764', 'Gm19786', 'Gm19807', 'Gm19931',
'Gm19932', 'Gm2000', 'Gm20056', 'Gm20063', 'Gm20072', 'Gm20082',
'Gm20091', 'Gm20147', 'Gm20150', 'Gm20177', 'Gm20302', 'Gm20336',
'Gm20362', 'Gm20369', 'Gm20428', 'Gm20479', 'Gm20519', 'Gm20568',
'Gm20570', 'Gm20594', 'Gm20621', 'Gm20717', 'Gm20745', 'Gm2077',
'Gm20878', 'Gm20946', 'Gm20953', 'Gm21093', 'Gm21319', 'Gm21399',
'Gm21541', 'Gm21559', 'Gm21586', 'Gm2164', 'Gm21748', 'Gm21860',
'Gm21863', 'Gm21936', 'Gm21953', 'Gm21955', 'Gm21963', 'Gm21985',
'Gm2199', 'Gm2213', 'Gm22184', 'Gm2225', 'Gm22498', 'Gm22567',
'Gm2260', 'Gm22750', 'Gm22814', 'Gm22971', 'Gm23153', 'Gm23505',
'Gm23566', 'Gm2395', 'Gm24105', 'Gm2412', 'Gm2415', 'Gm24187',
'Gm24207', 'Gm24245', 'Gm24260', 'Gm24340', 'Gm24462', 'Gm24514',
'Gm24601', 'Gm2464', 'Gm2477', 'Gm24787', 'Gm25018', 'Gm25203',
'Gm25578', 'Gm25687', 'Gm25814', 'Gm2607', 'Gm26190', 'Gm26445',
'Gm26571', 'Gm26600', 'Gm26629', 'Gm26652', 'Gm26658', 'Gm26737',
'Gm26740', 'Gm26743', 'Gm26789', 'Gm26793', 'Gm26815', 'Gm26871',
'Gm26936', 'Gm2694', 'Gm26953', 'Gm27167', 'Gm27178', 'Gm27243',
'Gm2736', 'Gm27572', 'Gm2791', 'Gm27910', 'Gm2805', 'Gm2824',
'Gm28386', 'Gm28392', 'Gm28580', 'Gm28651', 'Gm28659', 'Gm28688',
'Gm2869', 'Gm28800', 'Gm28905', 'Gm28911', 'Gm28927', 'Gm29041',
'Gm29103', 'Gm2912', 'Gm29124', 'Gm29138', 'Gm29164', 'Gm29228',

'Gm29242', 'Gm29519', 'Gm29602', 'Gm29610', 'Gm2965', 'Gm29676',
'Gm29739', 'Gm29745', 'Gm29753', 'Gm29770', 'Gm29787', 'Gm29808',
'Gm2981', 'Gm29966', 'Gm30023', 'Gm30177', 'Gm30382', 'Gm30648',
'Gm30652', 'Gm30708', 'Gm30731', 'Gm31205', 'Gm31479', 'Gm3148',
'Gm31518', 'Gm31520', 'Gm31693', 'Gm31763', 'Gm32051', 'Gm3213',
'Gm32158', 'Gm32444', 'Gm32926', 'Gm32939', 'Gm33050', 'Gm33121',
'Gm33178', 'Gm33228', 'Gm33312', 'Gm33508', 'Gm33533', 'Gm336',
'Gm33707', 'Gm3375', 'Gm33952', 'Gm34248', 'Gm34304', 'Gm34583',
'Gm34655', 'Gm34730', 'Gm34783', 'Gm34788', 'Gm34866', 'Gm35013',
'Gm3508', 'Gm35106', 'Gm35161', 'Gm35248', 'Gm3527', 'Gm35405',
'Gm3571', 'Gm35769', 'Gm35853', 'Gm36198', 'Gm36595', 'Gm36756',
'Gm36758', 'Gm3678', 'Gm36816', 'Gm36899', 'Gm36964', 'Gm36985',
'Gm37009', 'Gm37070', 'Gm37098', 'Gm37102', 'Gm37108', 'Gm3712',
'Gm37143', 'Gm37164', 'Gm37170', 'Gm37194', 'Gm37233', 'Gm37240',
'Gm37259', 'Gm37306', 'Gm37340', 'Gm37416', 'Gm37419', 'Gm37421',
'Gm37424', 'Gm37459', 'Gm37530', 'Gm37670', 'Gm37696', 'Gm37824',
'Gm37832', 'Gm3785', 'Gm3786', 'Gm3788', 'Gm37974', 'Gm38015',
'Gm38022', 'Gm38040', 'Gm38207', 'Gm38247', 'Gm3829', 'Gm38295',
'Gm38305', 'Gm38392', 'Gm38413', 'Gm38433', 'Gm38562', 'Gm38574',
'Gm3883', 'Gm3888', 'Gm39147', 'Gm39197', 'Gm3940', 'Gm3963',
'Gm39653', 'Gm39822', 'Gm4012', 'Gm40271', 'Gm4045', 'Gm40493',
'Gm40518', 'Gm4052', 'Gm4057', 'Gm40893', 'Gm4131', 'Gm41414',
'Gm41505', 'Gm41764', 'Gm42161', 'Gm4217', 'Gm42303', 'Gm42453',
'Gm42456', 'Gm42513', 'Gm4258', 'Gm42665', 'Gm42694', 'Gm42742',
'Gm42776', 'Gm42786', 'Gm42814', 'Gm42845', 'Gm42848', 'Gm42875',
'Gm42898', 'Gm42901', 'Gm42947', 'Gm42977', 'Gm43000', 'Gm43002',
'Gm43008', 'Gm43014', 'Gm43028', 'Gm43240', 'Gm43305', 'Gm43487',
'Gm43507', 'Gm43510', 'Gm43578', 'Gm43646', 'Gm43670', 'Gm43672',
'Gm4374', 'Gm43803', 'Gm43948', 'Gm44010', 'Gm44018', 'Gm44085',
'Gm44156', 'Gm44163', 'Gm44199', 'Gm44210', 'Gm44227', 'Gm44254',
'Gm44279', 'Gm44389', 'Gm44402', 'Gm44419', 'Gm44431', 'Gm44609',
'Gm44643', 'Gm44644', 'Gm44645', 'Gm44649', 'Gm44705', 'Gm44723',
'Gm44736', 'Gm4478', 'Gm44805', 'Gm44843', 'Gm44844', 'Gm44978',
'Gm44987', 'Gm45027', 'Gm45070', 'Gm4510', 'Gm45193', 'Gm4525',
'Gm45252', 'Gm45346', 'Gm45405', 'Gm45420', 'Gm45422', 'Gm45436',
'Gm4544', 'Gm45456', 'Gm45458', 'Gm45481', 'Gm45512', 'Gm45551',
'Gm45588', 'Gm45638', 'Gm45660', 'Gm45716', 'Gm45717', 'Gm45733',
'Gm45740', 'Gm45756', 'Gm45774', 'Gm45827', 'Gm45850', 'Gm45856',
'Gm4587', 'Gm45883', 'Gm45901', 'Gm45926', 'Gm45928', 'Gm45930',
'Gm4602', 'Gm46123', 'Gm46142', 'Gm4617', 'Gm46190', 'Gm46192',
'Gm46312', 'Gm46344', 'Gm46349', 'Gm46367', 'Gm46369', 'Gm46499',
'Gm46522', 'Gm46546', 'Gm46562', 'Gm46577', 'Gm46611', 'Gm46618',
'Gm46649', 'Gm47050', 'Gm47057', 'Gm47079', 'Gm47134', 'Gm47135',
'Gm47184', 'Gm47272', 'Gm47337', 'Gm47338', 'Gm47339', 'Gm47355',
'Gm47381', 'Gm47399', 'Gm47402', 'Gm4742', 'Gm47437', 'Gm47438',
'Gm4751', 'Gm47538', 'Gm47540', 'Gm47553', 'Gm4757', 'Gm4761',
'Gm47612', 'Gm47661', 'Gm47678', 'Gm47706', 'Gm47715', 'Gm47724',
'Gm4773', 'Gm47748', 'Gm47767', 'Gm4777', 'Gm47808', 'Gm47838',
'Gm47885', 'Gm47894', 'Gm47938', 'Gm47942', 'Gm47951', 'Gm4796',

'Gm4798', 'Gm48003', 'Gm48012', 'Gm48016', 'Gm48045', 'Gm48068',
'Gm48088', 'Gm48159', 'Gm48200', 'Gm48228', 'Gm48229', 'Gm4824',
'Gm48296', 'Gm4832', 'Gm4837', 'Gm48392', 'Gm48458', 'Gm48513',
'Gm4852', 'Gm4853', 'Gm48623', 'Gm48635', 'Gm48705', 'Gm4875',
'Gm48761', 'Gm48795', 'Gm4883', 'Gm48906', 'Gm48920', 'Gm48953',
'Gm48957', 'Gm48960', 'Gm48974', 'Gm4899', 'Gm48990', 'Gm49021',
'Gm49079', 'Gm49083', 'Gm49105', 'Gm49168', 'Gm49263', 'Gm49264',
'Gm49301', 'Gm4935', 'Gm49353', 'Gm4936', 'Gm49374', 'Gm49379',
'Gm4938', 'Gm49381', 'Gm4939', 'Gm49441', 'Gm4945', 'Gm49467',
'Gm49487', 'Gm4949', 'Gm49504', 'Gm49534', 'Gm49552', 'Gm49564',
'Gm49578', 'Gm49601', 'Gm4963', 'Gm49656', 'Gm49657', 'Gm49676',
'Gm49678', 'Gm4968', 'Gm49685', 'Gm49689', 'Gm49692', 'Gm49717',
'Gm49720', 'Gm49739', 'Gm49857', 'Gm4986', 'Gm49866', 'Gm49869',
'Gm4987', 'Gm49932', 'Gm49948', 'Gm49954', 'Gm49958', 'Gm4997',
'Gm49996', 'Gm49999', 'Gm50024', 'Gm50040', 'Gm50045', 'Gm50109',
'Gm50111', 'Gm50156', 'Gm50174', 'Gm50178', 'Gm50185', 'Gm50286',
'Gm50307', 'Gm50334', 'Gm50355', 'Gm50370', 'Gm50371', 'Gm50381',
'Gm50394', 'Gm5040', 'Gm50420', 'Gm50422', 'Gm50432', 'Gm5055',
'Gm5100', 'Gm5106', 'Gm5135', 'Gm5147', 'Gm5160', 'Gm5176', 'Gm5177',
'Gm5185', 'Gm5216', 'Gm5217', 'Gm5221', 'Gm5223', 'Gm5224', 'Gm5228',
'Gm5231', 'Gm5237', 'Gm5239', 'Gm5240', 'Gm5251', 'Gm5254', 'Gm5256',
'Gm5259', 'Gm5264', 'Gm5265', 'Gm5276', 'Gm5277', 'Gm5283', 'Gm5288',
'Gm5291', 'Gm5292', 'Gm5308', 'Gm5309', 'Gm5311', 'Gm5312', 'Gm5315',
'Gm5321', 'Gm5335', 'Gm5358', 'Gm5364', 'Gm5380', 'Gm5384', 'Gm5389',
'Gm5396', 'Gm5406', 'Gm5415', 'Gm5425', 'Gm5426', 'Gm5436', 'Gm5449',
'Gm5454', 'Gm5461', 'Gm5468', 'Gm5471', 'Gm5473', 'Gm5479', 'Gm5507',
'Gm5527', 'Gm5528', 'Gm5529', 'Gm5558', 'Gm5559', 'Gm5609', 'Gm5612',
'Gm5614', 'Gm5616', 'Gm5652', 'Gm5654', 'Gm5676', 'Gm5678', 'Gm5686',
'Gm5687', 'Gm5688', 'Gm5693', 'Gm5711', 'Gm5735', 'Gm5738', 'Gm5745',
'Gm5761', 'Gm5764', 'Gm5771', 'Gm5780', 'Gm5781', 'Gm5787', 'Gm5788',
'Gm5791', 'Gm5809', 'Gm5822', 'Gm5823', 'Gm5826', 'Gm5828', 'Gm5829',
'Gm5832', 'Gm5864', 'Gm5865', 'Gm5867', 'Gm5870', 'Gm5876', 'Gm5881',
'Gm5915', 'Gm5928', 'Gm5946', 'Gm5948', 'Gm5951', 'Gm5963', 'Gm5967',
'Gm5969', 'Gm5978', 'Gm5981', 'Gm5983', 'Gm5995', 'Gm5996', 'Gm5997',
'Gm6007', 'Gm6023', 'Gm6028', 'Gm6029', 'Gm6035', 'Gm6043', 'Gm6059',
'Gm6060', 'Gm6064', 'Gm6065', 'Gm6068', 'Gm6069', 'Gm6071', 'Gm6075',
'Gm6098', 'Gm6100', 'Gm6112', 'Gm6128', 'Gm6134', 'Gm6155', 'Gm6158',
'Gm6169', 'Gm6175', 'Gm6177', 'Gm6180', 'Gm6197', 'Gm6203', 'Gm6226',
'Gm6240', 'Gm6263', 'Gm6272', 'Gm6291', 'Gm6311', 'Gm6322', 'Gm6332',
'Gm6368', 'Gm6373', 'Gm6378', 'Gm6390', 'Gm6407', 'Gm6410', 'Gm6419',
'Gm6426', 'Gm6434', 'Gm6440', 'Gm6474', 'Gm6520', 'Gm6533', 'Gm6550',
'Gm6554', 'Gm6562', 'Gm6564', 'Gm6576', 'Gm6581', 'Gm6599', 'Gm6607',
'Gm6616', 'Gm6625', 'Gm6637', 'Gm6643', 'Gm6686', 'Gm6693', 'Gm6695',
'Gm6702', 'Gm6706', 'Gm6717', 'Gm6722', 'Gm6781', 'Gm6788', 'Gm6794',
'Gm6822', 'Gm6851', 'Gm6867', 'Gm6869', 'Gm6881', 'Gm6884', 'Gm6913',
'Gm6919', 'Gm6929', 'Gm6939', 'Gm6942', 'Gm6944', 'Gm6946', 'Gm6947',
'Gm6951', 'Gm6954', 'Gm6964', 'Gm6975', 'Gm6985', 'Gm6989', 'Gm6995',
'Gm6997', 'Gm7002', 'Gm7004', 'Gm7042', 'Gm7049', 'Gm7059', 'Gm7063',
'Gm7065', 'Gm7075', 'Gm7082', 'Gm7083', 'Gm7085', 'Gm7091', 'Gm7102',

'Gm7105', 'Gm7123', 'Gm7160', 'Gm7182', 'Gm7198', 'Gm7199', 'Gm7204',
'Gm7213', 'Gm7239', 'Gm7253', 'Gm7258', 'Gm7275', 'Gm7290', 'Gm7308',
'Gm7312', 'Gm7329', 'Gm7334', 'Gm7336', 'Gm7357', 'Gm7363', 'Gm7367',
'Gm7369', 'Gm7380', 'Gm7382', 'Gm7384', 'Gm7407', 'Gm7419', 'Gm7422',
'Gm7424', 'Gm7426', 'Gm7429', 'Gm7434', 'Gm7436', 'Gm7445', 'Gm7459',
'Gm7464', 'Gm7477', 'Gm7482', 'Gm7492', 'Gm7502', 'Gm7504', 'Gm7507',
'Gm7516', 'Gm7517', 'Gm7539', 'Gm7556', 'Gm7557', 'Gm7558', 'Gm7589',
'Gm7595', 'Gm7614', 'Gm7619', 'Gm7621', 'Gm7626', 'Gm7642', 'Gm7667',
'Gm7670', 'Gm7680', 'Gm7694', 'Gm7695', 'Gm7730', 'Gm7768', 'Gm7781',
'Gm7784', 'Gm7820', 'Gm7839', 'Gm7840', 'Gm7847', 'Gm7851', 'Gm7855',
'Gm7857', 'Gm7859', 'Gm7863', 'Gm7870', 'Gm7911', 'Gm7926', 'Gm7927',
'Gm7933', 'Gm7936', 'Gm7962', 'Gm7986', 'Gm7997', 'Gm8000', 'Gm8019',
'Gm8034', 'Gm8051', 'Gm8062', 'Gm8069', 'Gm8100', 'Gm8106', 'Gm8115',
'Gm8118', 'Gm8129', 'Gm8130', 'Gm8134', 'Gm8152', 'Gm8163', 'Gm8168',
'Gm8173', 'Gm8174', 'Gm8186', 'Gm8199', 'Gm8202', 'Gm8213', 'Gm8219',
'Gm8242', 'Gm8251', 'Gm8260', 'Gm8264', 'Gm8268', 'Gm8290', 'Gm8302',
'Gm8317', 'Gm8326', 'Gm8343', 'Gm8345', 'Gm8353', 'Gm8363', 'Gm8373',
'Gm8394', 'Gm8420', 'Gm8421', 'Gm8444', 'Gm8464', 'Gm8483', 'Gm8508',
'Gm8514', 'Gm8515', 'Gm8517', 'Gm8523', 'Gm8534', 'Gm8543', 'Gm8545',
'Gm8566', 'Gm8574', 'Gm8588', 'Gm8606', 'Gm8618', 'Gm8625', 'Gm8648',
'Gm8649', 'Gm8659', 'Gm8662', 'Gm8666', 'Gm8696', 'Gm8697', 'Gm8712',
'Gm8722', 'Gm8724', 'Gm8731', 'Gm8737', 'Gm8767', 'Gm8796', 'Gm8801',
'Gm8802', 'Gm8804', 'Gm8805', 'Gm8822', 'Gm8823', 'Gm8835', 'Gm8849',
'Gm8860', 'Gm8896', 'Gm8918', 'Gm8920', 'Gm8925', 'Gm8929', 'Gm8940',
'Gm8948', 'Gm8971', 'Gm8973', 'Gm8982', 'Gm8990', 'Gm9001', 'Gm9009',
'Gm9022', 'Gm9078', 'Gm9088', 'Gm9095', 'Gm9097', 'Gm9109', 'Gm9143',
'Gm9158', 'Gm9164', 'Gm9172', 'Gm9176', 'Gm9202', 'Gm9234', 'Gm9238',
'Gm9267', 'Gm9278', 'Gm9280', 'Gm9285', 'Gm9379', 'Gm9386', 'Gm9391',
'Gm9399', 'Gm9411', 'Gm9416', 'Gm9434', 'Gm9435', 'Gm9436', 'Gm9540',
'Gm9550', 'Gm9613', 'Gm9644', 'Gm9645', 'Gm9662', 'Gm9700', 'Gm9728',
'Gm9780', 'Gm9844', 'Gm9899', 'Gm9925', 'Gm9954', 'Gm9989', 'Gnaz',
'Gnb3', 'Gng12', 'Gng13', 'Gng3', 'Golga7b', 'Golt1a', 'Got2-ps1',
'Gp1bb', 'Gpd1', 'Gpd2', 'Gpm6b', 'Gpr150', 'Gpr180', 'Gpr22',
'Gpr26', 'Gpr45', 'Gpr55', 'Gpr85', 'Gpr88', 'Gprin3', 'Gramd2',
'Grasp', 'Grb14', 'Greb1l', 'Grhl1', 'Grik4', 'Grik5', 'Grm2', 'Grm3',
'Grm7', 'Grm8', 'Grp', 'Gsg1l', 'Gsn', 'Gsta4', 'Gstp1', 'Gstp2',
'Gstt1', 'Gtdc1', 'Gucy1a1', 'Gucy1a2', 'Gucy2g', 'H1fx', 'H2-DMa',
'H2-M6-ps', 'H2-T-ps', 'H2-T23', 'H2afj', 'H3f3c', 'Hace1', 'Haus6',
'Hbb-bh1', 'Hdc', 'Hecw1', 'Herc6', 'Hes1', 'Hey2', 'Hgf', 'Hhla1',
'Hipk4', 'Hist1h2ab', 'Hist1h2aj', 'Hist1h2ak', 'Hist1h2an',
'Hist1h2ao', 'Hist1h2bb', 'Hist1h3f', 'Hist1h3i', 'Hist1h4d',
'Hist1h4h', 'Hist1h4i', 'Hist1h4k', 'Hist1h4m', 'Hist1h4n',
'Hist3h2ba', 'Hist3h2bb-ps', 'Hivep1', 'Hivep2', 'Hivep3', 'Hk1',
'Hkdc1', 'Hmgb1-ps11', 'Hmgb1-ps9', 'Hmgb1-rs17', 'Hmgcr', 'Homer1',
'Hpca', 'Hpcal4', 'Hrh1', 'Hs3st2', 'Hs3st4', 'Hs6st1', 'Hs6st3',
'Hsd17b7', 'Hspa1a', 'Hspa1b', 'Hspb3', 'Hspd1-ps4', 'Hspd1-ps5',
'Htr1b', 'Htr1f', 'Htr2a', 'Htr5a', 'Hydin', 'Icam5', 'Id2', 'Ier5',
'Igf1r', 'Igf2bp2', 'Igfbp4', 'Igfbp6', 'Igfn1', 'Ighg2c', 'Ighm',
'Igkv14-134-1', 'Igsf21', 'Igsf9b', 'Il11ra1', 'Il11ra2', 'Il12a',

'Il15ra', 'Il17ra', 'Il31', 'Il34', 'Ildr2', 'Imp3', 'Inf2', 'Inhba',
'Inka2', 'Ints7', 'Ipcef1', 'Iqca', 'Iqch', 'Iqcn', 'Iqgap1',
'Iqschfp', 'Irf1', 'Irf2', 'Islr', 'Ism1', 'Itgav', 'Itgb8', 'Itm2c',
'Itpk1', 'Itpka', 'Itpr1', 'Jak2', 'Jam3', 'Jcad', 'Jph1', 'Jpt1',
'Jsrp1', 'Junb', 'Jup', 'Kalrn', 'Katnal2', 'Kcna4', 'Kcna5', 'Kcnb1',
'Kcnc4', 'Kcnf1', 'Kcng1', 'Kcnh1', 'Kcnh3', 'Kcnh4', 'Kcnh5',
'Kcnh7', 'Kcnip3', 'Kcnj2', 'Kcnj4', 'Kcnj5', 'Kcnj6', 'Kcnk4',
'Kcnk9', 'Kcnma1', 'Kcnmb4', 'Kcns2', 'Kcnt2', 'Kcnv1', 'Kctd1',
'Kctd13', 'Kctd4', 'Kdm7a', 'Kel', 'Khdrbs2', 'Khdrbs3', 'Kif17',
'Kitl', 'Klf10', 'Klf13', 'Klf16', 'Klf4', 'Klf7', 'Klf9', 'Klhl2',
'Klhl21', 'Klhl23', 'Klhl29', 'Klhl3', 'Klhl38', 'Kmt5a', 'Kng2',
'Kremen1', 'Krt10', 'Krt12', 'Krt20', 'Krt80', 'L3mbtl1', 'L3mbtl3',
'Lamb2', 'Lamc2', 'Lamp5', 'Larp6', 'Lasp1', 'Lax1', 'Lct', 'Ldb2',
'Ldha', 'Ldlr', 'Ldlrad2', 'Ldlrad4', 'Leng9', 'Lgi1', 'Lhfp', 'Lhx2',
'Limd1', 'Limd2', 'Lin7a', 'Lin7b', 'Lingo1', 'Lingo3', 'Lipg',
'Lipt2', 'Lix1', 'Llgl2', 'Llph-ps1', 'Lmcd1', 'Lmo1', 'Lmo4', 'Lmo7',
'Lnx2', 'Lonrf1', 'Loxhd1', 'Lpar1', 'Lpcat4', 'Lratd2', 'Lrfn2',
'Lrg1', 'Lrp4', 'Lrrc32', 'Lrrc42', 'Lrrc55', 'Lrrc57', 'Lrrc6',
'Lrrc7', 'Lrrc75a', 'Lrriq1', 'Lsm11', 'Lsm8', 'Ltk', 'Lxn', 'Ly6e',
'Ly6e-ps1', 'Ly6g5b', 'Lypd1', 'Lyrm2', 'Lyzl4', 'Lzts1', 'Lzts3',
'M1ap', 'Macc1', 'Mak', 'Maml3', 'Man2a1', 'Map1lc3a', 'Map2k1',
'Map3k15', 'Map3k5', 'Map3k6', 'Map3k8', 'Mapk1', 'Mapk11', 'Mapk12',
'Mapk4', 'Marc2', 'March1', 'March10', 'Mark2', 'Mas1', 'Mast3',
'Mast4', 'Mat2b', 'Matk', 'Mblac1', 'Mc5r', 'Mcee', 'Mctp2', 'Mcu',
'Mdga1', 'Me1', 'Med10', 'Med27', 'Mef2c', 'Mef2d', 'Megf11', 'Mei1',
'Meis2', 'Meltf', 'Mesp2', 'Metrnl', 'Mettl21c', 'Mettl24',
'Mettl7a1', 'Mettl7a3', 'Mex3b', 'Mfrp', 'Mfsd13a', 'Mfsd13b',
'Mfsd4b3-ps', 'Mgp', 'Miat', 'Micu1', 'Mien1', 'Minar2', 'Mindy3',
'Mir1934', 'Mir221', 'Mir5099', 'Mir670hg', 'Mir684-1', 'Mir692-1',
'Mir7240', 'Mkrn1-ps1', 'Mlip', 'Mmaa', 'Mmd', 'Mmd2', 'Mmp17', 'Mnt',
'Mocos', 'Mov10', 'Mpped2', 'Mras', 'Mroh3', 'Mroh5', 'Mrps18c',
'Mrps28', 'Mrtfa', 'Mrtfb', 'Mrto4-ps1', 'Mrto4-ps2', 'Mt2', 'Mt3',
'Mtfr2', 'Mtmr10', 'Mtmr12', 'Mtpn', 'Muc15', 'Muc19', 'Muc3a',
'Mup5', 'Mybpc3', 'Myc', 'Myh10', 'Myh3', 'Myh7', 'Myh9', 'Myhas',
'Myl4', 'Mylip', 'Mylk3', 'Myo15b', 'Myo18b', 'Myo1d', 'Naa38',
'Naa80', 'Nab1', 'Nab2', 'Naf1', 'Naif1', 'Nanog', 'Nat14', 'Nbl1',
'Ncam2', 'Ncan', 'Nck2', 'Ndrg1', 'Ndufa11b', 'Ndufaf8', 'Ndufs6b',
'Neat1', 'Neb', 'Necab3', 'Nectin3', 'Nedd4l', 'Nefl', 'Nefm',
'Nell2', 'Neo1', 'Net1', 'Neu2', 'Neurl1a', 'Neurl1b', 'Neurod2',
'Neurod6', 'Nf2', 'Nfatc1', 'Nfe2l3', 'Nfia', 'Nfib', 'Nfix', 'Ngef',
'Nhp2', 'Nhsl2', 'Nin', 'Nkain2', 'Nkpd1', 'Nlgn1', 'Nlk', 'Nlk-ps1',
'Nlrc4', 'Nme5', 'Nos1ap', 'Npas2', 'Npas4', 'Nphp1', 'Nphs1', 'Npr3',
'Nptx1', 'Nptx2', 'Nptxr', 'Npy1r', 'Nr2c2ap', 'Nr3c1', 'Nr4a1',
'Nr4a2', 'Nr4a3', 'Nrcam', 'Nrg1', 'Nrgn', 'Nrl', 'Nrn1', 'Nrp1',
'Nrp2', 'Nrxn1', 'Ntn5', 'Ntng1', 'Ntng2', 'Ntsr1', 'Ntsr2', 'Nuak1',
'Nudt3', 'Numb', 'Numbl', 'Nwd2', 'Nxf7', 'Nxph3', 'Nxph4', 'Obscn',
'Ociad2', 'Olfm1', 'Olfml2b', 'Olfr1167', 'Olfr1279', 'Olfr1323',
'Olfr1441', 'Olfr1506', 'Olfr155', 'Olfr177', 'Olfr275', 'Olfr63',
'Olfr78', 'Olfr802', 'Olfr901', 'Olfr905', 'Onecut2', 'Orai2',

'Osbp2', 'Osbpl10', 'Osbpl1a', 'Osbpl5', 'Osr1', 'Otop2', 'Otud1',
'Otulinl', 'Otx1', 'Ovol2', 'Oxgr1', 'P4ha1', 'P4ha3', 'Pacsin3',
'Pak1', 'Pak7', 'Palm', 'Palm3', 'Palmd', 'Pamr1', 'Panct2', 'Parp10',
'Pbdc1', 'Pcdh9', 'Pced1b', 'Pcsk1', 'Pcsk2', 'Pcsk2os2', 'Pcsk5',
'Pdap1', 'Pde10a', 'Pde1a', 'Pde4b', 'Pde4d', 'Pde7b', 'Pdgfb',
'Pdgfc', 'Pdia5', 'Pdk1', 'Pdlim1', 'Pdp1', 'Pdzrn3', 'Pdzrn4',
'Pef1', 'Pet100', 'Pex5l', 'Pfkfb3', 'Pfkl', 'Pgam2', 'Pgk1-ps1',
'Pglyrp3', 'Pgm2l1', 'Pgr', 'Phactr1', 'Phf20-ps', 'Phf21b', 'Phf24',
'Phyhip', 'Pianp', 'Piezo2', 'Pih1d1', 'Pik3r1', 'Pik3r6', 'Pim3',
'Pin1', 'Pip4k2a', 'Pip4k2c', 'Pip5k1c', 'Pira2', 'Pitpnm2', 'Pkn1',
'Pkp2', 'Pla2g12b', 'Plat', 'Platr29', 'Plcl2', 'Plcxd2', 'Plekha2',
'Plekha6', 'Plekha7', 'Plekhg5', 'Plk2', 'Plk5', 'Pln', 'Plp2',
'Plppr2', 'Pls1', 'Pls3', 'Plxdc1', 'Plxnd1', 'Pmepa1', 'Pnma1',
'Poc1a', 'Podxl', 'Polq', 'Polr1d', 'Polr2j', 'Polr2l', 'Pop5',
'Pou1f1', 'Pou2f2', 'Pou3f1', 'Pou3f2', 'Pou6f2', 'Ppfibp1', 'Ppm1k',
'Ppme1', 'Ppp1r10', 'Ppp1r1a', 'Ppp1r1b', 'Ppp1r7', 'Ppp1r9a',
'Ppp2r5a', 'Ppp3ca', 'Ppp3r1', 'Pradc1', 'Prag1', 'Pramel4', 'Prcp',
'Prdm8', 'Prdx2-ps1', 'Prdx2-ps2', 'Prdx6-ps2', 'Prdx6b', 'Prex1',
'Prickle1', 'Prickle3', 'Prkab2', 'Prkag2os1', 'Prkcb', 'Prkce',
'Prkcg', 'Prkcz2', 'Prkg1', 'Prr16', 'Prr36', 'Prrt1', 'Prss12',
'Prss22', 'Prss3', 'Prss30', 'Prss35', 'Psd3', 'Psmb5-ps', 'Psme4',
'Psrc1', 'Ptch2', 'Ptchd1', 'Pter', 'Ptger1', 'Ptger3', 'Ptgs2',
'Ptgs2os', 'Pth2', 'Ptk2', 'Ptk2b', 'Ptms', 'Ptn', 'Ptp4a3', 'Ptpn13',
'Ptpn3', 'Ptprd', 'Ptprf', 'Ptprh', 'Ptprj', 'Ptprk', 'Pwp2', 'Pxdn',
'Qrfprl', 'R3hdm1', 'R3hdm2', 'R3hdm4', 'Rab11fip4', 'Rab15', 'Rab26',
'Rab33a', 'Rab36', 'Rab39', 'Rab3d', 'Rab3ip', 'Rab40b', 'Rab40c',
'Rad51ap2', 'Radil', 'Raet1e', 'Rai14', 'Raly', 'Ralyl', 'Ramp1',
'Ramp3', 'Rap1gap2', 'Rap2a', 'Rap2b', 'Rap2c', 'Rapgefl1', 'Raph1',
'Rara', 'Rasal1', 'Rasd2', 'Rasgef1b', 'Rasgef1c', 'Rasgrp1',
'Rasl10a', 'Rasl11b', 'Rbfox3', 'Rbm12b1', 'Rbm15', 'Rbm22', 'Rbm24',
'Rcc2', 'Rcor2', 'Rdh5', 'Reep6', 'Rell1', 'Rem2', 'Rep15', 'Retn',
'Rgl1', 'Rgs14', 'Rgs19', 'Rgs2', 'Rgs20', 'Rgs4', 'Rhag', 'Riiad1',
'Rilpl1', 'Rin1', 'Riox1', 'Rmnd5b', 'Rmst', 'Rn7s6', 'Rnd1',
'Rnf112', 'Rnf130', 'Rnf144a', 'Rnf150', 'Rnf169', 'Rnf38', 'Rnf39',
'Rnf43', 'Rny3', 'Robo3', 'Romo1', 'Rorb', 'Rpl10-ps4', 'Rpl10a-ps2',
'Rpl10a-ps3', 'Rpl13a-ps1', 'Rpl15-ps1', 'Rpl15-ps5', 'Rpl17-ps1',
'Rpl17-ps7', 'Rpl19-ps10', 'Rpl19-ps12', 'Rpl19-ps2', 'Rpl19-ps4',
'Rpl19-ps9', 'Rpl21-ps1', 'Rpl21-ps11', 'Rpl21-ps3', 'Rpl21-ps4',
'Rpl21-ps5', 'Rpl23a-ps1', 'Rpl23a-ps5', 'Rpl26-ps2', 'Rpl26-ps5',
'Rpl27-ps1', 'Rpl27-ps2', 'Rpl29-ps5', 'Rpl30-ps10', 'Rpl30-ps11',
'Rpl30-ps6', 'Rpl30-ps8', 'Rpl31-ps1', 'Rpl31-ps10', 'Rpl31-ps11',
'Rpl31-ps15', 'Rpl31-ps16', 'Rpl31-ps17', 'Rpl31-ps18', 'Rpl31-ps20',
'Rpl31-ps21', 'Rpl31-ps22', 'Rpl31-ps25', 'Rpl31-ps8', 'Rpl32-ps',
'Rpl34-ps2', 'Rpl35a-ps5', 'Rpl35a-ps6', 'Rpl36-ps10', 'Rpl36-ps4',
'Rpl36-ps8', 'Rpl36-ps9', 'Rpl36a-ps1', 'Rpl37rt', 'Rpl38', 'Rpl38-
ps1', 'Rpl38-ps2', 'Rpl7a-ps1', 'Rpl7a-ps8', 'Rpl9-ps1', 'Rplp1-ps1',
'Rprm', 'Rprml', 'Rps11-ps4', 'Rps12-ps1', 'Rps12-ps10', 'Rps12-ps13',
'Rps12-ps19', 'Rps12-ps21', 'Rps12-ps5', 'Rps13-ps1', 'Rps15a-ps1',
'Rps15a-ps4', 'Rps15a-ps8', 'Rps19-ps4', 'Rps19-ps5', 'Rps19-ps7',

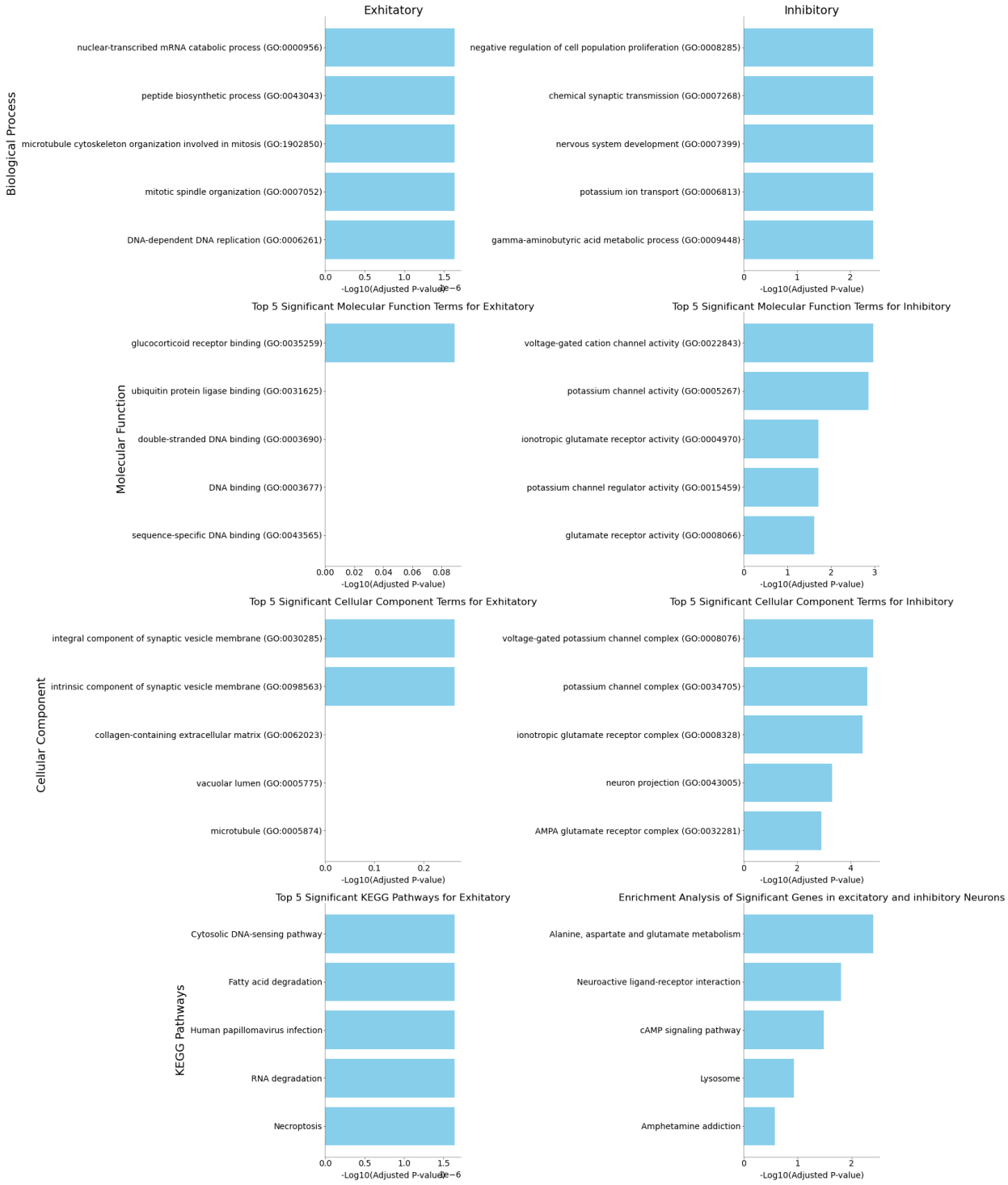
'Rps19-ps8', 'Rps19-ps9', 'Rps2-ps10', 'Rps2-ps2', 'Rps2-ps7', 'Rps24-ps2', 'Rps26-ps1', 'Rps27-ps1', 'Rps29', 'Rps6ka3', 'Rpsa-ps12', 'Rpsa-ps5', 'Rpusd3', 'Rragd', 'Rsbn1', 'Rskr', 'Rsph1', 'Rsph9', 'Rspo2', 'Rspo3', 'Rtn4', 'Rtn4r', 'Rtn4rl2', 'Ryr3', 'S100a10', 'S100a6', 'S100b', 'Sac3d1', 'Samd15', 'Satb2', 'Scn4b', 'Scube1', 'Sdhaf1', 'Sdk2', 'Sebox', 'Secisbp2l', 'Selenok-ps4', 'Selenok-ps5', 'Selenok-ps6', 'Selenok-ps7', 'Selenow', 'Sema4a', 'Sema4c', 'Sema7a', 'Sept11', 'Sept7', 'Serinc2', 'Serpinb6a', 'Serpinb8', 'Sertad1', 'Sertad2', 'Sgcz', 'Sgk3', 'Sh2d4b', 'Sh3bgrl3', 'Sh3gl2', 'Sh3rf1', 'Shank2', 'Shf', 'Shisa4', 'Shisa5', 'Siae', 'Sidt1', 'Sigmar1', 'Skiv2l-ps1', 'Sla', 'Slc13a4', 'Slc13a5', 'Slc16a10', 'Slc17a5', 'Slc17a6', 'Slc17a7', 'Slc1a2', 'Slc20a1', 'Slc22a15', 'Slc24a4', 'Slc25a42', 'Slc25a5-ps', 'Slc26a10', 'Slc26a4', 'Slc26a8', 'Slc2a1', 'Slc30a1', 'Slc31a2', 'Slc34a2', 'Slc37a1', 'Slc39a1', 'Slc39a10', 'Slc39a5', 'Slc6a11', 'Slc7a4', 'Slc8a1', 'Slc8a2', 'Slc9b2', 'Slco1a6', 'Slco2a1', 'Slco4a1', 'Slco4c1', 'Slit1', 'Slit3', 'Slitrk1', 'Slk', 'Smad3', 'Smad7', 'Smap1', 'Smim8', 'Smoc2', 'Smt3h2-ps2', 'Smurf1', 'Smyd3', 'Snca', 'Snn', 'Snorc', 'Snord104', 'Snrp1c-ps1', 'Snrpc', 'Snrpf', 'Snta1', 'Sntb2', 'Snw1', 'Socs2', 'Sorbs2', 'Sorcs1', 'Sorl1', 'Sowaha', 'Sowahb', 'Sox30', 'Sox4', 'Sox5', 'Sox8', 'Sox9', 'Spa17', 'Spag5', 'Spag6', 'Spata2l', 'Spcs2-ps', 'Speg', 'Spin3-ps', 'Spr', 'Sprn', 'Sprtn', 'Spry2', 'Spryd3', 'Spsb1', 'Sptbn2', 'Sptbn5', 'Srebf2', 'Srf', 'Srr', 'Ssbp3', 'Ssh3', 'Sstr2', 'Sstr3', 'Sstr4', 'St14', 'St3gal5', 'St6gal2', 'St6galnac3', 'Stac3', 'Stam2', 'Stard10', 'Stard8', 'Stim2', 'Stk10', 'Stmn1', 'Stmn4', 'Stn1', 'Stox1', 'Stpg4', 'Stum', 'Stx1a', 'Stx4a', 'Stx6', 'Stxbp5l', 'Sulf1', 'Sult2b1', 'Sun1', 'Supt4a', 'Susd2', 'Suv39h2', 'Sv2b', 'Syn1', 'Syndig1', 'Syngap1', 'Synj2', 'Synpo', 'Synrg', 'Syt10', 'Syt12', 'Syt16', 'Syt4', 'Syt5', 'Syt6', 'Syt7', 'Sytl2', 'Tacc1', 'Tafa1', 'Tagln3', 'Tapbpl', 'Tbata', 'Tbc1d2', 'Tbc1d22a', 'Tbc1d32', 'Tbr1', 'Tbrg1', 'Tceal6', 'Tceanc2', 'Tcf20', 'Tchh', 'Tdo2', 'Tdrd7', 'Tead2', 'Tecpr2', 'Tek', 'Tekt1', 'Tenm2', 'Tesc', 'Tet3', 'Tex2', 'Tex261', 'Tex30', 'Tgfb1i1', 'Theg', 'Themis', 'Thg1l', 'Tiam2', 'Timm8b', 'Tjp3', 'Tktl1', 'Tle4', 'Tle6', 'Tmeff1', 'Tmem116', 'Tmem121b', 'Tmem132a', 'Tmem132d', 'Tmem14a', 'Tmem150c', 'Tmem158', 'Tmem159', 'Tmem160', 'Tmem163', 'Tmem181b-ps', 'Tmem181c-ps', 'Tmem19', 'Tmem196', 'Tmem200c', 'Tmem215', 'Tmem232', 'Tmem245', 'Tmem254a', 'Tmem254b', 'Tmem254c', 'Tmem50a', 'Tmem64', 'Tmem74', 'Tmem88b', 'Tmem94', 'Tmod1', 'Tmprss7', 'Tmsb10', 'Tmtc1', 'Tnfaip6', 'Tnfrsf10b', 'Tnfrsf19', 'Tnks1bp1', 'Tnnc1', 'Tnnt2', 'Tnpo1', 'Tom1l1', 'Tomm6', 'Tox', 'Tpd52', 'Tpd52-ps', 'Tpd52l1', 'Tpi-rs10', 'Trabd2b', 'Traf5', 'Traip', 'Trbc1', 'Trbc2', 'Trbj2-1', 'Trbj2-2', 'Trbj2-3', 'Trbj2-4', 'Trbj2-5', 'Trbj2-6', 'Trbj2-7', 'Trib1', 'Trim68', 'Trp53i11', 'Trpm3', 'Trpm4', 'Trpv2', 'Trpv5', 'Trpv6', 'Tsacc', 'Tsga13', 'Tsnax', 'Tsnaxip1', 'Tspan11', 'Tspan14', 'Tspan17', 'Ttc7', 'Ttc7b', 'Ttc9b', 'Tuba4a', 'Tubb4a', 'Tubb4b-ps2', 'Tusc3', 'Twf2', 'Txn-ps1', 'Tyro3', 'Ubash3a', 'Ube2l3-ps1', 'Ubtd2', 'Ubxn11', 'Uchl1', 'Uck2', 'Ulk4', 'Umad1', 'Unc13b', 'Unc5a', 'Unc80', 'Uqcc2', 'Uqcr10', 'Uqcr11', 'Urah', 'Ush2a', 'Usp3', 'Usp43', 'Usp46', 'Usp50', 'Utp4', 'Uxs1', 'Vash1', 'Vegfd', 'Vgll4',

```
'Vhl', 'Vinac1', 'Vipr1', 'Vmn1r-ps145', 'Vmn1r218', 'Vmn1r72',
'Vmn1r73', 'Vmn1r87', 'Vmn2r110', 'Vmn2r59', 'Vmn2r63', 'Vmn2r76',
'Vmn2r84', 'Vmn2r85', 'Vmn2r86', 'Vmn2r87', 'Vps13a', 'Vps37b',
'Vps37c', 'Vps45', 'Vsig2', 'Vti1a', 'Vwa3a', 'Vxn', 'Wasf1', 'Wasl',
'Wdr24', 'Wdr66', 'Wdr90', 'Wdr95', 'Wfs1', 'Wnt1', 'Wnt10a', 'Wnt4',
'Wnt7b', 'Wnt9a', 'Wwc1', 'Xirp2', 'Xrcc4', 'Xrra1', 'Xylb', 'Xylt1',
'Yap1', 'Ykt6', 'Ypel1', 'Ywhah', 'Ywhaq-ps2', 'Zbtb1', 'Zbtb18',
'Zbtb44', 'Zbtb46', 'Zbtb7c', 'Zcchc2', 'Zdbf2', 'Zdhhc12', 'Zdhhc23',
'Zdhhc24', 'Zfand3', 'Zfhx4', 'Zfp189', 'Zfp239', 'Zfp296', 'Zfp365',
'Zfp428', 'Zfp46', 'Zfp503', 'Zfp575', 'Zfp652os', 'Zfp668', 'Zfp703',
'Zfp827', 'Zfp831', 'Zfp945', 'Zfp978', 'Zfpm2', 'Zfyve28',
'Zkscan16', 'Zmat2', 'Zmiz1', 'Zmiz1os1', 'Znhit1', 'Znrf3', 'Zxdc',
'mt-Ta', 'mt-Tc', 'mt-Te', 'mt-Tf', 'mt-Th', 'mt-Tl1', 'mt-Tm', 'mt-
Tn', 'mt-Tp', 'mt-Tq', 'mt-Tr', 'mt-Ts2', 'mt-Tt', 'mt-Tv', 'n-
R5s115', 'n-R5s151', 'n-R5s194']

import gseapy as gp


# Function to plot the top 5 significant results
def plot_top5(ax, data, title):
    # Sort the data and get the top 5 results
    top5 = data.sort_values(by="Adjusted P-value").head(5)

    # Create the bar plot on the provided ax
    ax.barh(top5["Term"], -np.log10(top5["Adjusted P-value"]),
color="skyblue")
    ax.set_xlabel("-Log10(Adjusted P-value)")
    ax.set_title(title)
    ax.invert_yaxis()

    return ax


# Create a subplot with 2 columns and 4 rows
fig, axs = plt.subplots(4, 2, figsize=(15, 20))

# Define criteria and labels
criteria = ["Exhitatory", "Inhibitory"]
analysis_types = [
    "Biological Process",
    "Molecular Function",
    "Cellular Component",
    "KEGG Pathways",
]

# Define your gene lists
gene_lists = [significant_genes_excitatory_list,
significant_genes_inhibitory_list]
```

```
for i, gene_list in enumerate(gene_lists):
    # Perform enrichment analysis for Biological Process
    enr_bp = gp.enrichr(
        gene_list=gene_list,
        gene_sets="GO_Biological_Process_2021",
        organism="Mouse",  # Change to 'human' if your genes are human
        outdir=None,  # Set to a path if you want to save results to
files
    )

    # Perform enrichment analysis for Molecular Function
    enr_mf = gp.enrichr(
        gene_list=gene_list,
        gene_sets="GO_Molecular_Function_2021",
        organism="Mouse",
        outdir=None,
    )

    # Perform enrichment analysis for Cellular Component
    enr_cc = gp.enrichr(
        gene_list=gene_list,
        gene_sets="GO_Cellular_Component_2021",
        organism="Mouse",
        outdir=None,
    )

    # Perform enrichment analysis for KEGG Pathways
    enr_kegg = gp.enrichr(
        gene_list=gene_list,
        gene_sets="KEGG_2019_Mouse",
        organism="Mouse",
        outdir=None,
    )

    # Assign each subplot to the appropriate analysis
    plot_top5(
        axs[0, i],
        enr_bp.results,
        f"Top 5 Significant Biological Process Terms for
{criteria[i]}",
    )
    plot_top5(
        axs[1, i],
        enr_mf.results,
        f"Top 5 Significant Molecular Function Terms for
{criteria[i]}",
    )
    plot_top5(
        axs[2, i],
        enr_cc.results,
```

```
        f"Top 5 Significant Cellular Component Terms for
{criteria[i]}",
    )
    plot_top5(
        axs[3, i],
        enr_kegg.results,
        f"Top 5 Significant KEGG Pathways for {criteria[i]}",
    )

# Add column labels
for ax, criterion in zip(axs[0], criteria):
    ax.set_title(criterion, fontsize=14)

# Add row labels
for ax, analysis_type in zip(axs[:, 0], analysis_types):
    ax.set_ylabel(analysis_type, fontsize=14)

# Adjust layout
plt.tight_layout()
plt.title(
    "Enrichment Analysis of Significant Genes in excitatory and
inhibitory Neurons"
)
plt.show()
```

```
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\1445115026.py:97: UserWarning: The figure layout has
changed to tight
  plt.tight_layout()
```

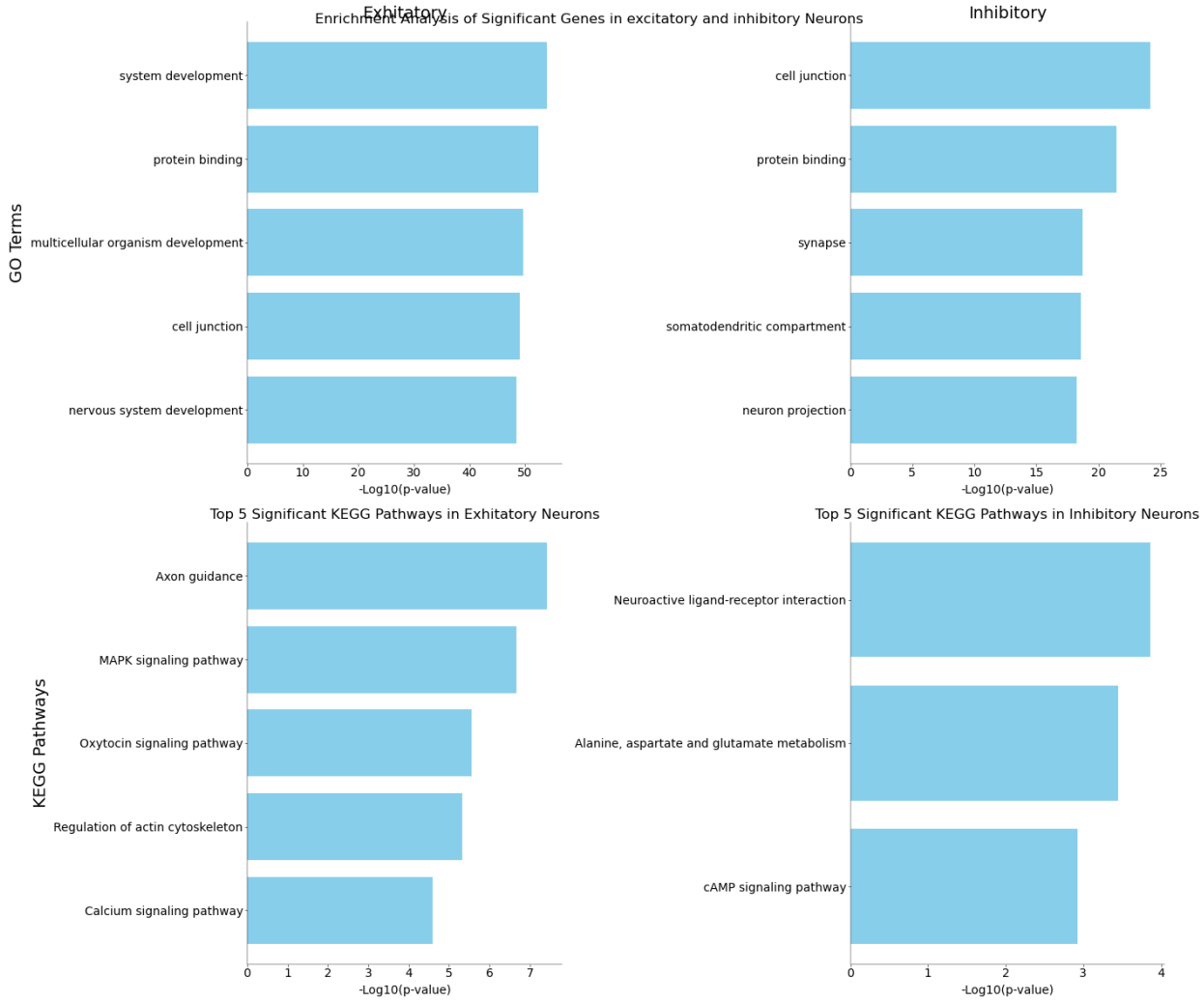



```
from gprofiler import GProfiler


# Function to plot horizontal bar chart
def plot_horizontal_bar(ax, data, title):
    # Sort the data and get the top 5 results
```

```
    top5 = data.sort_values(by="p_value").head(5)

    # Create the bar plot on the provided ax
    ax.barh(top5["name"], -np.log10(top5["p_value"]), color="skyblue")
    ax.set_xlabel("-Log10(p-value)")
    ax.set_title(title)
    ax.invert_yaxis()


# Initialize the GProfiler object
gp = GProfiler(return_dataframe=True)

# Create a subplot with 2 columns and 2 rows
fig, axs = plt.subplots(2, 2, figsize=(14, 12))

criteria = ["Exhitatory", "Inhibitory"]
for i, genes in enumerate(
    [significant_genes_excitatory_list,
significant_genes_inhibitory_list]
):
    # Perform the enrichment analysis
    results = gp.profile(organism="mmusculus", query=genes)

    # Filter results to include only GO terms
    go_results = results[results["source"].str.contains("GO")]

    # Filter results to include only KEGG pathways
    kegg_results = results[results["source"].str.contains("KEGG")]

    # Apply multiple testing correction threshold (e.g., FDR < 0.05)
    significant_go_results = go_results[go_results["p_value"] < 0.05]
    significant_kegg_results = kegg_results[kegg_results["p_value"] <
0.05]

    # Sort the results by p-value to get the most significant results
    sorted_go_results =
significant_go_results.sort_values(by="p_value").head(5)
    sorted_kegg_results =
significant_kegg_results.sort_values(by="p_value").head(5)

    # Plot top 5 significant GO terms
    plot_horizontal_bar(
        axs[0, i],
        sorted_go_results,
        f"Top 5 Significant GO Terms in {criteria[i]} Neurons",
    )

    # Plot top 5 significant KEGG pathways
    plot_horizontal_bar(
        axs[1, i],
```

```
        sorted_kegg_results,
        f"Top 5 Significant KEGG Pathways in {criteria[i]} Neurons",
    )

# Add column labels
for ax, criterion in zip(axs[0], criteria):
    ax.set_title(criterion, fontsize=14)

# Add row labels
axs[0, 0].set_ylabel("GO Terms", fontsize=14)
axs[1, 0].set_ylabel("KEGG Pathways", fontsize=14)

# Adjust layout
plt.tight_layout()
plt.suptitle(
    "Enrichment Analysis of Significant Genes in excitatory and
inhibitory Neurons"
)
plt.show()
```

```
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\436738076.py:66: UserWarning: The figure layout has
changed to tight
  plt.tight_layout()
```



Compile the cell types pandas dataframe

```
# stack depth array and rna type array
depth_keep = depth[keepcells]
# rna_type_keep = rna_type[keepcells]
thickness_keep = thickness[keepcells]
# normalize depth via thickness
depth_keep = depth_keep / thickness_keep
# print min and max of depth
print(np.nanmin(depth_keep))
print(np.nanmax(depth_keep))

# normalize depth
depth_keep = (depth_keep - np.nanmin(depth_keep)) / (
    np.nanmax(depth_keep) - np.nanmin(depth_keep)
)

print(depth_keep.shape)
print(rna_type_keep.shape)
```

```
# Create a DataFrame for the cell types and depths
df_depth_per_type = pd.DataFrame({"rna_type": rna_type_keep, "depth":
depth_keep})

print(df_depth_per_type.head())
```

```
0.030842794478131853
0.9805202018837277
(1320,)
(1320,)
  rna_type     depth
0  L5 ET_1  0.492092
1  L5 IT_2  0.475048
2  L5 IT_1  0.442118
3  L5 ET_4  0.402793
4  L5 ET_1  0.585180
```

Plotting function for cell types and classes

```
def plot_cell_types(
    groups: list,
    ax: plt.Axes,
    base_font_size: int = 8,
    df_depth_per_type=pd.DataFrame({"rna_type": rna_type, "depth":
depth}),
) -> None:
    """
    Plot cell types of the mapping database

    Parameters
    ----------

    groups: list
        List of groups to plot

    ax: plt.Axes
        Axis to plot on

    base_font_size: int
        Base font size for plotting

    Return
    ------
    None

    """
    global mapping_db
    x_base = 0
    xticks = []
```

```
    xticklabels = []

    max_height = max(mapping_db["count"])

    figsize = ax.get_figure().get_size_inches()
    width_factor = figsize[0] / 15
    height_factor = figsize[1] / 5
    font_size = calculate_font_size(base_font_size, min(width_factor,
height_factor))

    for i, group in enumerate(groups):
        group_data = mapping_db[mapping_db["type"] == group]
        print(group_data.shape)
        x_positions = np.arange(len(group_data)) + x_base

        ax.bar(x_positions, group_data["count"],
color=group_data["color"], width=0.8)

        ax.text(
            x_base + (len(group_data) - 1) / 2,
            max_height - 10,
            group,
            ha="center",
            va="bottom",
            fontsize=font_size * 1.2,
            fontweight="bold",
        )

        xticks.extend(x_positions)
        xticklabels.extend(group_data["cell_types"])

        x_base += len(group_data) + 1  # Increment x_base for next
group

    ax.set_xticks(xticks)
    ax.set_xticklabels(xticklabels, rotation=90, fontsize=font_size)
    # (L1, 0.07; L2/3, 0.29; L5, 0.73)
    ax.set_ylabel("Count")


def plot_cell_types_depth(
    groups: list,
    base_font_size: int = 10,  # Manually set base font size
    figsize: tuple = (15, 10),  # Manually set figure size
    df_depth_per_type=pd.DataFrame({"rna_type": rna_type, "depth":
depth}),
) -> None:
    """
    Plot cell types of the mapping database and violin plot of depths
```

```
    Parameters
    ----------
    groups: list
        List of groups to plot
    base_font_size: int
        Base font size for plotting
    df_depth_per_type: pd.DataFrame
        DataFrame containing the depth information for each cell type

    Return
    ------
    None
    """
    global mapping_db

    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=figsize, sharex=True)

    x_base = 0
    xticks = []
    xticklabels = []
    colors = []

    max_height = max(mapping_db["count"])

    for i, group in enumerate(groups):
        group_data = mapping_db[mapping_db["type"] == group]
        x_positions = np.arange(len(group_data)) + x_base

        bars = ax1.bar(
            x_positions, group_data["count"],
color=group_data["color"], width=0.8
        )

        # Collect colors to use in violin plot
        colors.extend([bar.get_facecolor() for bar in bars])

        ax1.text(
            x_base + (len(group_data) - 1) / 2,
            max_height - 10,
            group,
            ha="center",
            va="bottom",
            fontsize=base_font_size * 1.2,
            fontweight="bold",
        )

        xticks.extend(x_positions)
        xticklabels.extend(group_data["cell_types"])

        x_base += len(group_data) + 1  # Increment x_base for next
```

```
group

    ax1.set_xticks(xticks)
    ax1.set_xticklabels(xticklabels, rotation=90,
fontsize=base_font_size)
    ax1.set_ylabel("Count", fontsize=base_font_size)

    # Prepare data for the violin plot
    violin_data =
df_depth_per_type[df_depth_per_type["rna_type"].isin(xticklabels)]
    sns.violinplot(
        x="rna_type",
        y="depth",
        data=violin_data,
        ax=ax2,
        order=xticklabels,
        palette=colors,
        alpha=0.5,
    )

    ax2.set_xlabel("Cell Types", fontsize=base_font_size)
    ax2.set_ylabel("Depth", fontsize=base_font_size)
    ax2.set_xticklabels(xticklabels, rotation=90,
fontsize=base_font_size)
    ax2.set_ylim(0, 1)

    # Add horizontal lines and labels
    horizontal_lines = {"L1": 0.07, "L2/3": 0.29, "L5": 0.73}

    # add y ticks for the horizontal lines
    for label, y in horizontal_lines.items():
        ax2.axhline(y=y, color="grey", linestyle="--")
        ax2.text(
            len(xticklabels) + 0.5,
            y,
            label,
            color="grey",
            va="center",
            fontsize=base_font_size,
        )

    for label, y in horizontal_lines.items():
        ax2.axhline(y=y, color="grey", linestyle="--")
        ax2.text(
            len(xticklabels) + 0.5,
            y,
            label,
            color="grey",
            va="center",
            fontsize=base_font_size,
```

```
        )

    plt.tight_layout()
    plt.show()
```

Analisys of ePhys Data

```
# Use knn_classification to classify the original data

Data = ephysData_filtered
labels = cluster_colors[keepcells]
print(cluster_colors[keepcells].shape)
# print(np.unique(cluster_colors[keepcells]))
accuracy, recall = knn_classification(Data, labels)
print(f"Original data label accuracy: {accuracy:.2f}")
print(f"Original data label recall: {recall:.2f}")
```

```
(1320,)
Original data label accuracy: 0.34
Original data label recall: 0.34
```

Exploring and plotting different dimensionality reduction methods

```
# perform different dimensionality reduction techniques
reduced_data = dim_reduction(ephysData_filtered,
cluster_colors[keepcells])
```

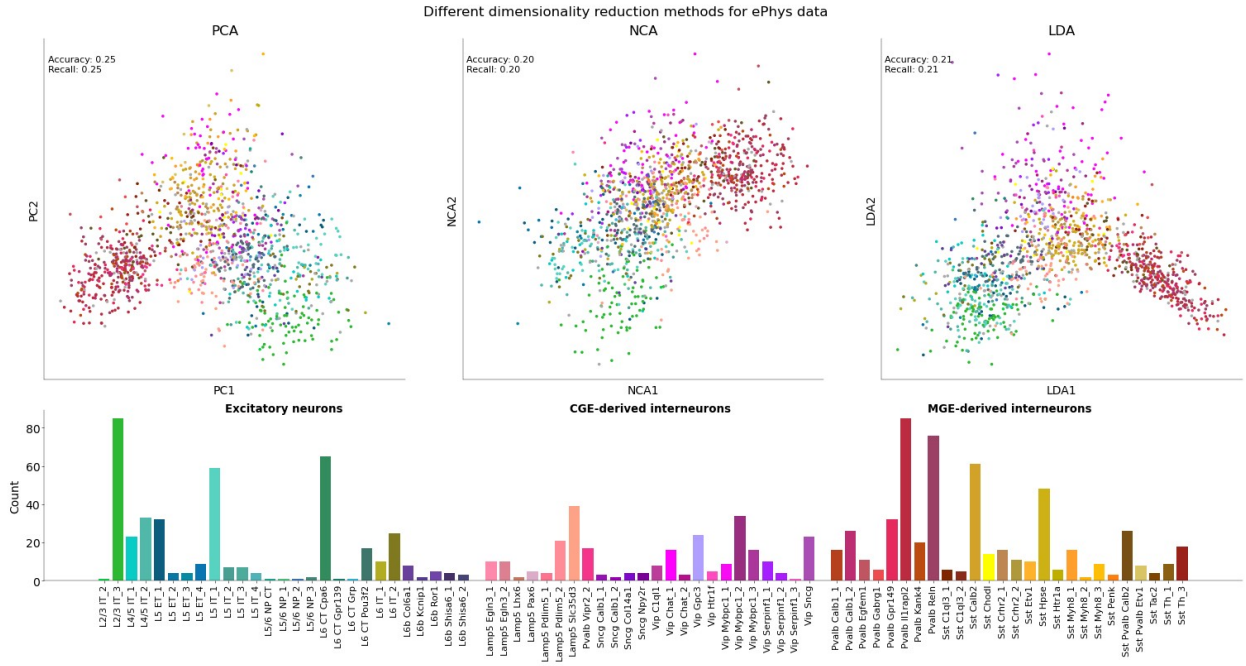
```
PCA accuracy: 0.25
PCA recall: 0.25
NCA accuracy: 0.20
NCA recall: 0.20
LDA accuracy: 0.21
LDA recall: 0.21
```

```
fig = plt.figure(figsize=(15, 8))

gs = GridSpec(2, 3, figure=fig, wspace=0.05, height_ratios=[2, 1])
# Create subplots with the specified gridspec
ax0 = fig.add_subplot(gs[0, 0])
ax1 = fig.add_subplot(gs[0, 1])
ax2 = fig.add_subplot(gs[0, 2])
ax3 = fig.add_subplot(gs[1, :])


axs = [ax0, ax1, ax2]
for i, (key, value) in enumerate(reduced_data.items()):
    plot_dim1_dim2(
        ax=axs[i],
        data=reduced_data,
        labels=cluster_colors[keepcells],
```

```
        type=key,
        title=key,
        display_stats=True,
    )
plot_cell_types(neuron_groups, ax3)
plt.suptitle("Different dimensionality reduction methods for ePhys
data")
```

```
(27, 4)
(24, 4)
(26, 4)
```

```
Text(0.5, 0.98, 'Different dimensionality reduction methods for ePhys
data')
```



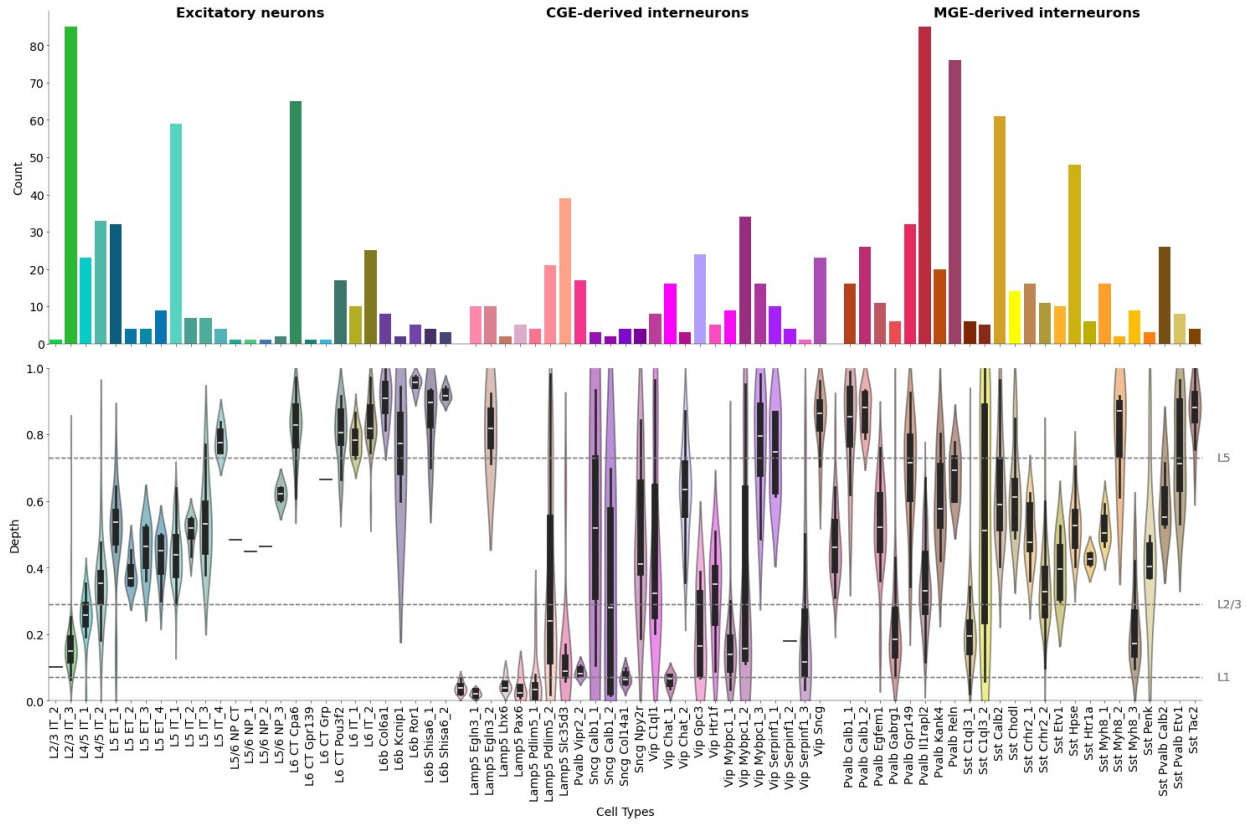
The figure demonstrates that NCA and LDA methods for dimensionality reduction seem to be slightly more accurate than PCA. Since the difference is not dramatic, we will use PCA (being the most common method) further on.

Representing ePhys data in PCA, TSNE and UMAP spaces. Performing and visuzalizing GMM-clustering on raw data.

```
plot_cell_types_depth(neuron_groups,
df_depth_per_type=df_depth_per_type)
```

```
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\583724656.py:66: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
```

```
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.violinplot(
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\583724656.py:107: UserWarning: The figure layout has
changed to tight
  plt.tight_layout()
```



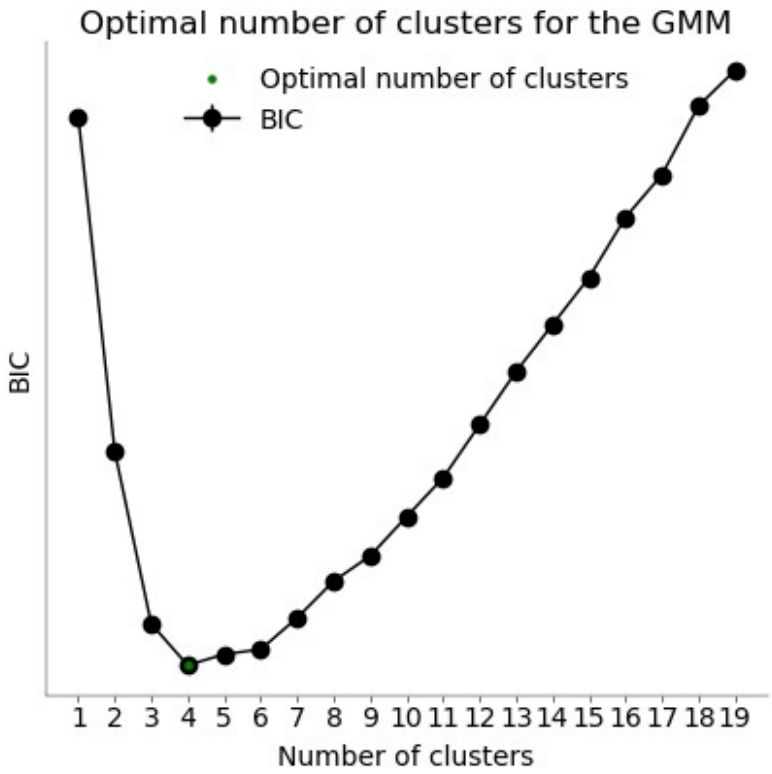

```
### GMM CLUSTERING ###
# set the range of possible cluster number
possible_clusters = np.arange(1, 20, 1)
num_seeds = 1

# get the optimal cluster number using BIC
ePhys_cluster_n = get_optimal_cluster_number(
    possible_clusters, num_seeds, ephysData_filtered,
bic_mode="sklearn", plot=True
)

# fit the GMM model
ePhys_gmm = GMM(n_components=ePhys_cluster_n,
random_state=42).fit(ephysData_filtered)
```

```python
# get the cluster labels, means and covariances
ePhys_clusters = ePhys_gmm.predict(ephysData_filtered)
ePhys_clusters_means = ePhys_gmm.means_
ePhys_clusters_covariances = ePhys_gmm.covariances_
```

```
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
```

```
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
```

```
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(

Optimal number of clusters: 4
```




```
# transform cluster means and covariances to PCA space for plotting
later
pca = make_pipeline(StandardScaler(), PCA(n_components=2))
pca_ePhys = pca.fit(Data)
ePhys_clusters_means = pca[pca.steps[-1]
[0]].transform(ePhys_clusters_means)
components = pca[pca.steps[-1][0]].components_
ePhys_clusters_covariances = [
    components @ cov @ components.T for cov in
ePhys_clusters_covariances
]
```

t-SNE and UMAP

```
### t-SNE ###
# performing t-SNE on the ePhys data
ePhys_tsne = TSNE(init="pca", early_exaggeration=4,
random_state=42).fit_transform(
    ephysData_filtered
)

### UMAP ###
# performing UMAP on the ePhys data
umap_ePhys =
umap.UMAP(random_state=42).fit_transform(ephysData_filtered)
```

```
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\umap\umap_.py:1945: UserWarning: n_jobs value 1
overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting
random_state. Use no seed for parallelism.")
```

Plot PCA-processed data with GMM clusters overlay and the t-SNE plot. Here we use original color labels from the paper

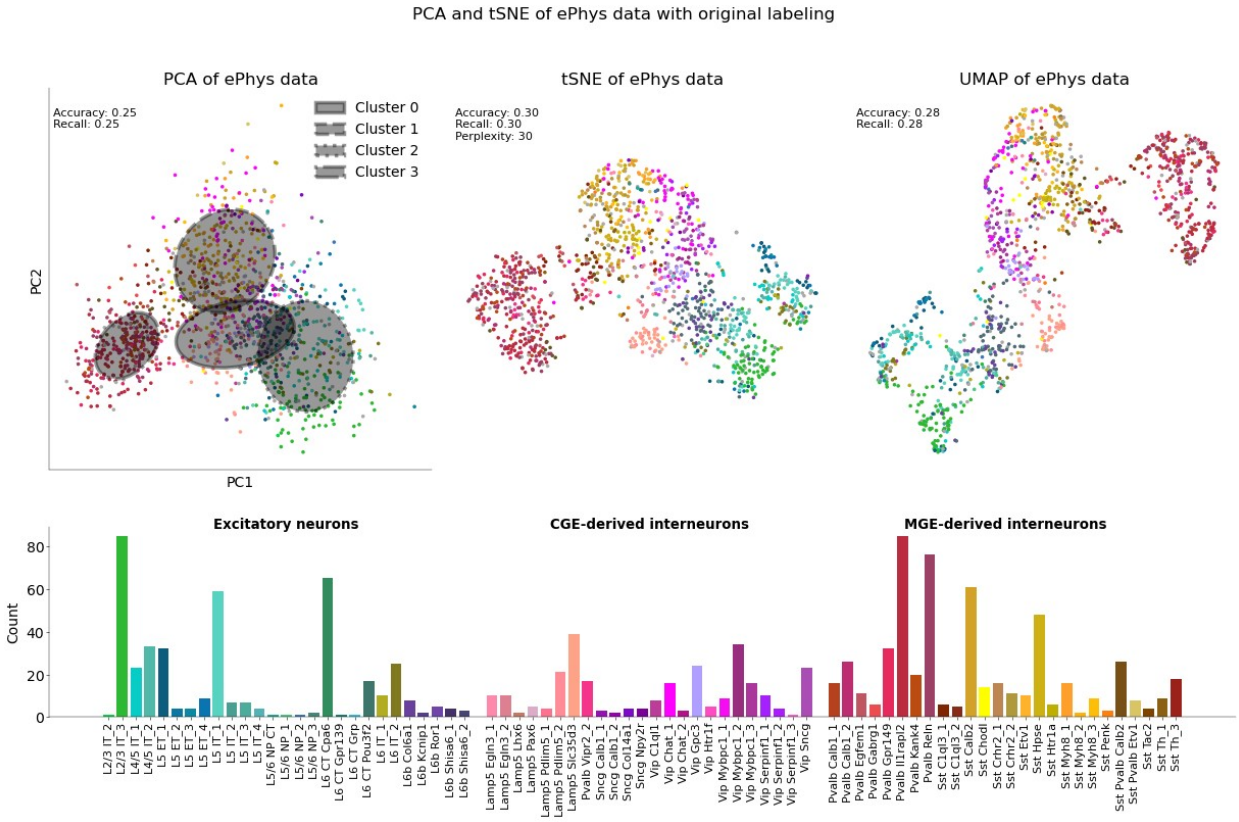
```
# pca_ePhys = pca_ePhys / np.std(pca_ePhys[:, 0]) * 0.0001

fig = plt.figure(figsize=(15, 8))

gs = GridSpec(2, 3, figure=fig, wspace=0.05, height_ratios=[2, 1])
# Create subplots with the specified gridspec
ax0 = fig.add_subplot(gs[0, 0])
ax1 = fig.add_subplot(gs[0, 1])
ax2 = fig.add_subplot(gs[0, 2])
ax3 = fig.add_subplot(gs[1, :])


plot_dim1_dim2(
    ax0,
    reduced_data,
    cluster_colors[keepcells],
    "PCA of ePhys data",
    type="PCA",
    display_stats=True,
)
plot_TSNE(
    ax1,
    ePhys_tsne,
    cluster_colors[keepcells],
    "tSNE of ePhys data",
    display_accuracy=True,
)
plot_umap(
    ax2,
    umap_ePhys,
    cluster_colors[keepcells],
    "UMAP of ePhys data",
    display_accuracy=True,
)
plot_cell_types(neuron_groups, ax3, base_font_size=8)

plot_ellipse(
    ePhys_clusters_means,
    ePhys_clusters_covariances,
    ax=ax0,
    color="black",
```

```
    cluster_n=ePhys_cluster_n,
)
ax0.legend()
plt.tight_layout()
plt.suptitle("PCA and tSNE of ePhys data with original labeling",
fontsize=12)
```

```
(27, 4)
(24, 4)
(26, 4)
```

```
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\3099095034.py:45: UserWarning: This figure includes
Axes that are not compatible with tight_layout, so results might be
incorrect.
  plt.tight_layout()
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\3099095034.py:45: UserWarning: The figure layout has
changed to tight
  plt.tight_layout()
```

```
Text(0.5, 0.98, 'PCA and tSNE of ePhys data with original labeling')
```



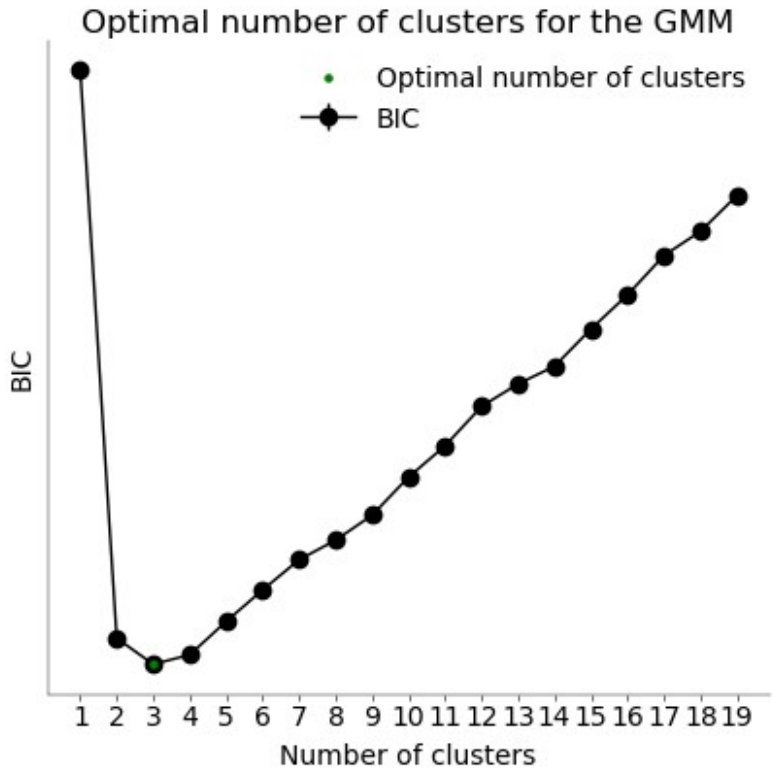
Representing ePhys data in PCA, TSNE and UMAP spaces. Performing and visuzalizing GMM-clustering on PCA-processed data.

```
### Dimensionality reduction###
reduced_data_ = dim_reduction(ephysData_filtered,
cluster_colors[keepcells])
pca_ePhys = reduced_data_["PCA"]["data"]
pca_ePhys_cluster_n = get_optimal_cluster_number(
    possible_clusters, num_seeds, pca_ePhys, bic_mode="sklearn",
plot=True
)

### GMM CLUSTERING ###
pca_ePhys_gmm = GMM(n_components=pca_ePhys_cluster_n,
random_state=42).fit(pca_ePhys)
pca_ePhys_clusters = pca_ePhys_gmm.predict(pca_ePhys)
pca_ePhys_clusters_means = pca_ePhys_gmm.means_
pca_ePhys_clusters_covariances = pca_ePhys_gmm.covariances_

PCA accuracy: 0.25
PCA recall: 0.25
NCA accuracy: 0.20
NCA recall: 0.20
LDA accuracy: 0.21
LDA recall: 0.21
```

```
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
```

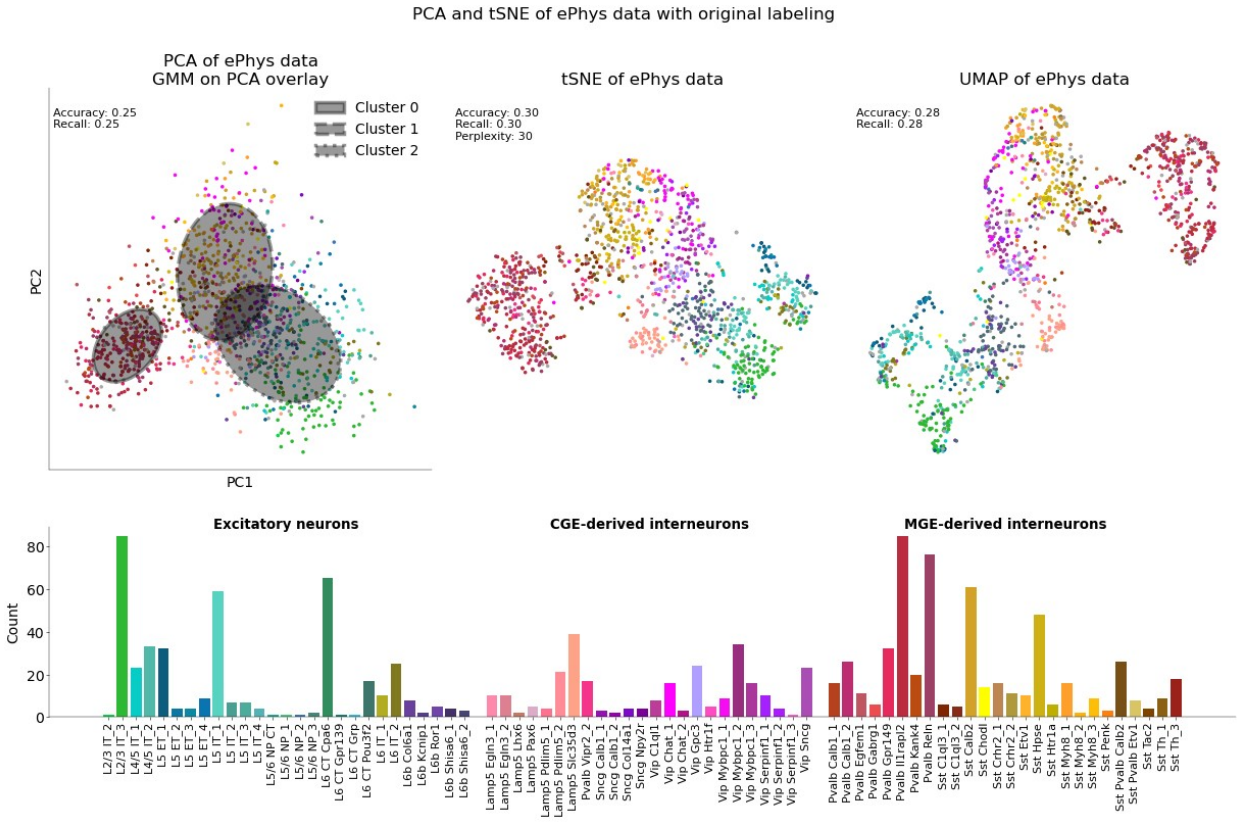
```
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
```

```
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(

Optimal number of clusters: 3

c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is
known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=6.
  warnings.warn(
```



```
### PLOTTING ###
fig = plt.figure(figsize=(15, 8))

gs = GridSpec(2, 3, figure=fig, wspace=0.05, height_ratios=[2, 1])
# Create subplots with the specified gridspec
ax0 = fig.add_subplot(gs[0, 0])
ax1 = fig.add_subplot(gs[0, 1])
ax2 = fig.add_subplot(gs[0, 2])
ax3 = fig.add_subplot(gs[1, :])


plot_dim1_dim2(
    ax0,
    reduced_data_,
    cluster_colors[keepcells],
    "PCA of ePhys data\nGMM on PCA overlay",
    type="PCA",
    display_stats=True,
)
plot_TSNE(
    ax1,
    ePhys_tsne,
    cluster_colors[keepcells],
    "tSNE of ePhys data",
    display_accuracy=True,
)
plot_umap(
    ax2,
    umap_ePhys,
    cluster_colors[keepcells],
    "UMAP of ePhys data",
    display_accuracy=True,
)
plot_cell_types(neuron_groups, ax3, base_font_size=8)

plot_ellipse(
    pca_ePhys_clusters_means,
    pca_ePhys_clusters_covariances,
    ax=ax0,
    color="black",
    cluster_n=pca_ePhys_cluster_n,
)
ax0.legend()
plt.tight_layout()
plt.suptitle("PCA and tSNE of ePhys data with original labeling",
fontsize=12)

(27, 4)
(24, 4)
(26, 4)
```

```
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\3889043698.py:44: UserWarning: This figure includes
Axes that are not compatible with tight_layout, so results might be
incorrect.
  plt.tight_layout()
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\3889043698.py:44: UserWarning: The figure layout has
changed to tight
  plt.tight_layout()

Text(0.5, 0.98, 'PCA and tSNE of ePhys data with original labeling')
```



Summary

Our analysis demonstrates that Pvalb MGE-derived interneurons are distinctly separated from other cell types in both PCA, t-SNE, and UMAP projections. This finding is consistent with the original study, aligning with the known characteristic of high firing rates in these interneurons. The Gaussian Mixture Model (GMM) clustering further validates this separation, both in PCA-reduced and raw data spaces.

Moreover, excitatory neurons form distinct clusters in all three dimensionality reduction techniques. The GMM applied to the raw data (with an optimal cluster number of 4) effectively segregates excitatory neurons. Conversely, the GMM applied to PCA-reduced data (optimal cluster number of 3) exhibits slightly reduced performance, misclassifying some CGE-derived interneurons within the excitatory neuron cluster.

In contrast, the CGE-derived and MGE-derived interneurons do not exhibit clear separation as whole families within the electrophysiological space. They are consistently clustered together, indicating overlapping electrophysiological properties that complicate their distinct classification using the methods applied.

K-nearest neighbors classification yields some information loss after PCA dimensionality reduction (0.33 accuracy in original data vs 0.22 in the PCA-space), which is a common trade-off in such analyses. TSNE and UMAP show better accuracy: 0.30 and 0.28 respectively.

As for the dimensionality reduction, the raw data might have more complex structures that GMM can capture better, leading to more clusters (4 vs 3). PCA simplifies the data, which might merge some of these structures into fewer clusters. Overall PCA-based clustering is more

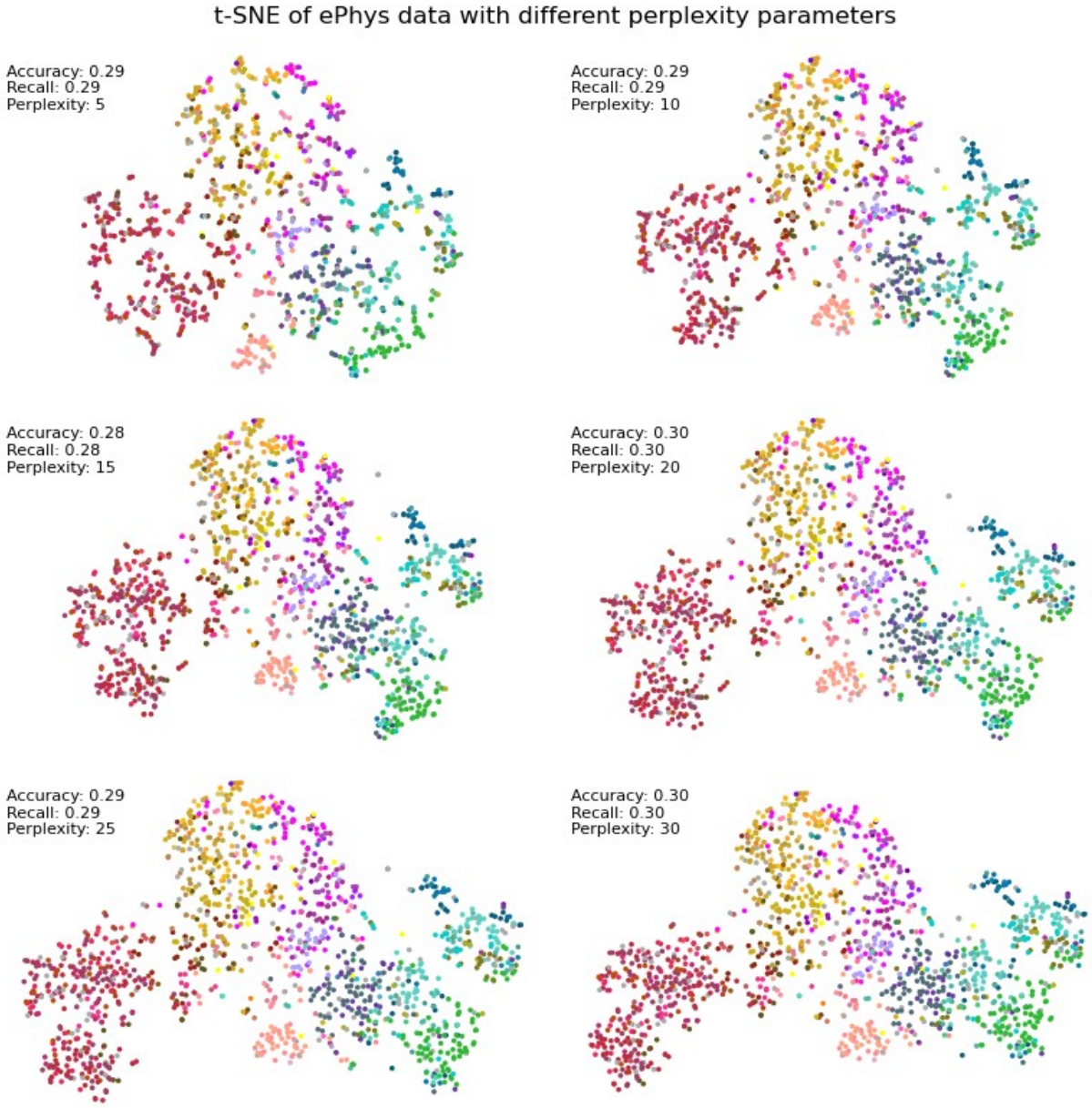
efficient and less prone to overfitting, but it might lose some finer details present in the raw data clustering.

Plotting t-SNE wit different parameters

```
# Initialize list to store t-SNE results
TSNES = []

# Define range of perplexity values to test
perplexities = np.arange(5, 35, 5)

# Calculate number of rows and columns for subplots grid
num_plots = len(perplexities)
ncols = 2
nrows = (num_plots + ncols - 1) // ncols  # Ensure enough rows for the
number of plots

# Set up subplots grid
fig, axs = plt.subplots(nrows, ncols, figsize=(8, 8))

# Flatten the axs array if it's 2D
if nrows * ncols > 1:
    axs = axs.flatten()

# Loop through perplexity values and perform t-SNE
for i, p in enumerate(perplexities):
    tsne = TSNE(perplexity=p,
random_state=42).fit_transform(ephysData_filtered)
    TSNES.append(tsne)
    ax = axs[i]
    plot_TSNE(
        ax,
        tsne,
        cluster_colors[keepcells],
        title="",
        display_accuracy=True,
        perplexity=p,
    )

# Remove any empty subplots
for j in range(i + 1, len(axs)):
    fig.delaxes(axs[j])

# Show the plot

plt.suptitle("t-SNE of ePhys data with different perplexity
parameters", fontsize=12)
plt.tight_layout()
```

```
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\metrics\_classification.py:1531:
```

```
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\262685583.py:40: UserWarning: The figure layout has
changed to tight
  plt.tight_layout()
```



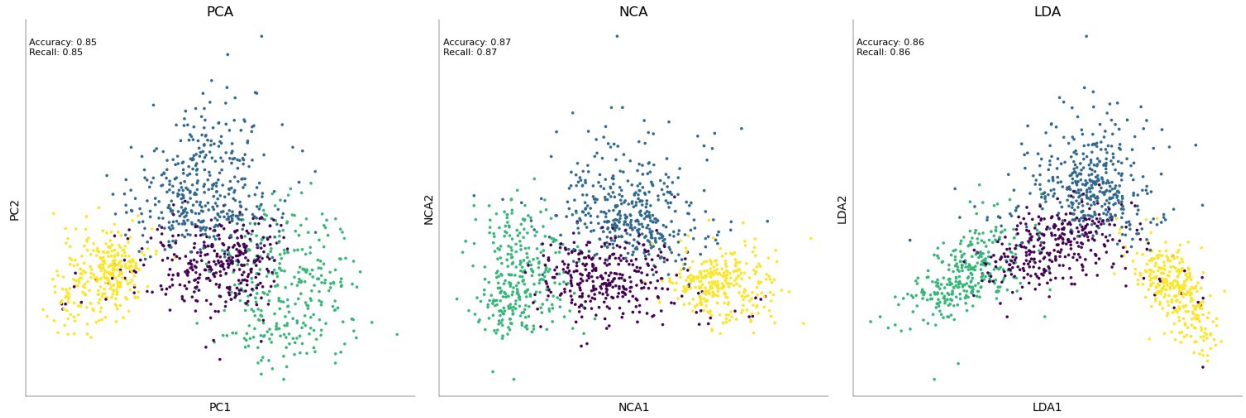
The figure shows that according to the k-nearest neighbors classification, TSNE with the perplexity parameter 30 has the highest accuracy. This is also the default parameter, which was used in other plots in this study

Representing ePhys data in PCA, TSNE and UMAP spaces with previously obtained GMM-labeling

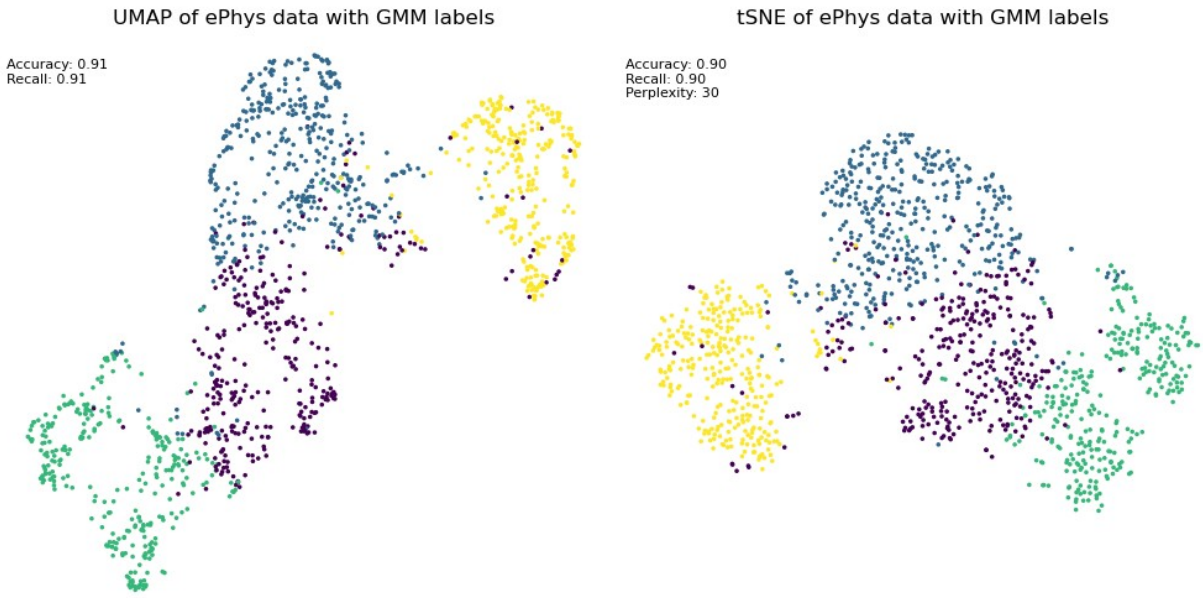
```
gmm_reduced_data = dim_reduction(ephysData_filtered, ePhys_clusters)

PCA accuracy: 0.85
PCA recall: 0.85
NCA accuracy: 0.87
NCA recall: 0.87
```

```
LDA accuracy: 0.86
LDA recall: 0.86
```

```
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
for i, (key, value) in enumerate(gmm_reduced_data.items()):
    plot_dim1_dim2(
        ax[i],
        gmm_reduced_data,
        ePhys_clusters,
        type=key,
        title=key,
        display_stats=True,
    )
```




```
fig, ax = plt.subplots(1, 2, figsize=(10, 5))

plot_umap(
    ax[0],
    umap_ePhys,
    ePhys_clusters,
    "UMAP of ePhys data with GMM labels",
    display_accuracy=True,
)
plot_TSNE(
    ax[1],
    ePhys_tsne,
    ePhys_clusters,
    "tSNE of ePhys data with GMM labels",
    display_accuracy=True,
)
```



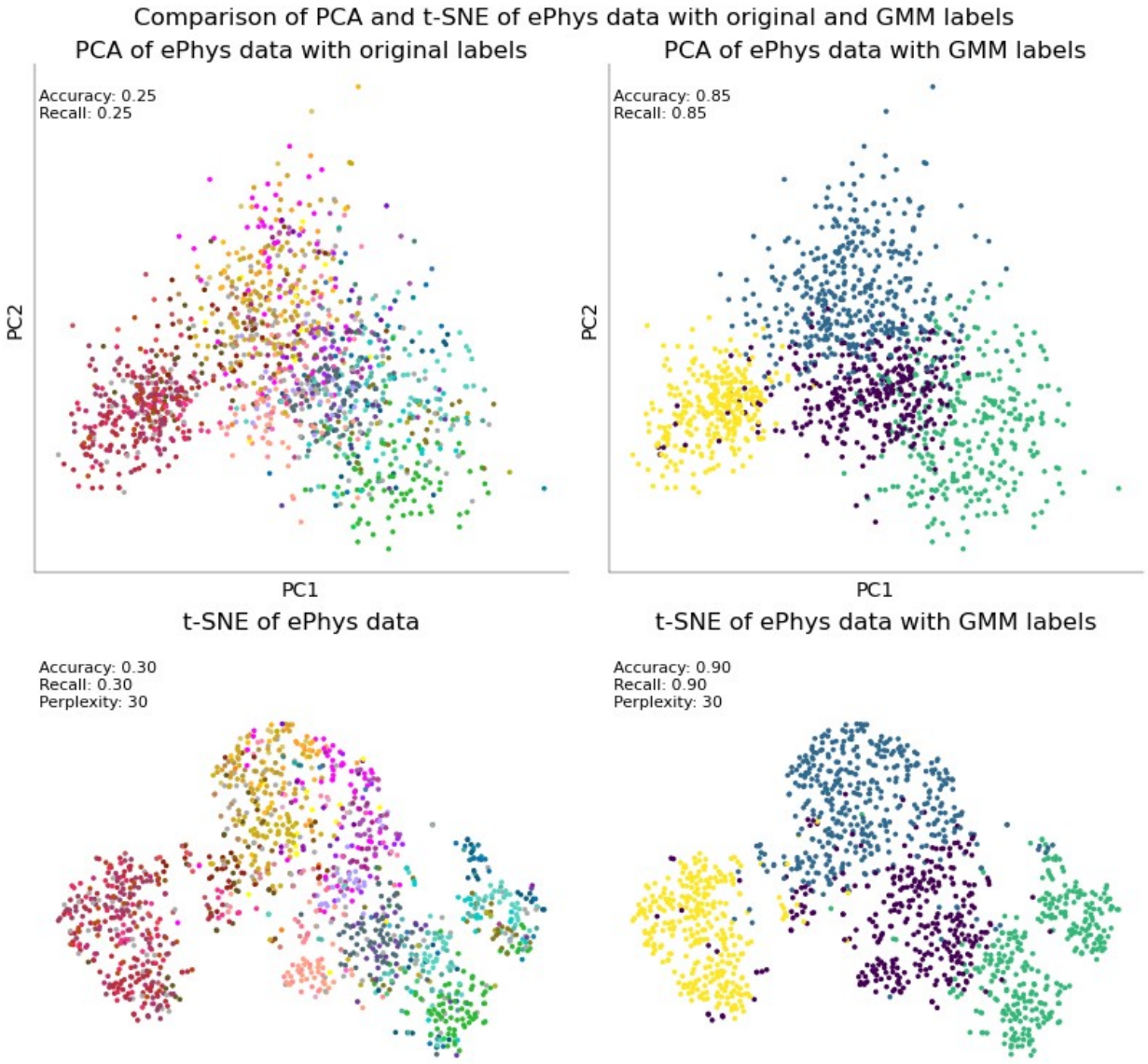
We can compare GMM labeling with original lableing

```
fig, ax = plt.subplots(2, 2, figsize=(8, 8))

plot_dim1_dim2(
    ax[0, 0],
    reduced_data,
    cluster_colors[keepcells],
    "PCA of ePhys data with original labels",
    type="PCA",
    display_stats=True,
)
plot_dim1_dim2(
    ax[0, 1],
    gmm_reduced_data,
    ePhys_clusters,
    "PCA of ePhys data with GMM labels",
    type="PCA",
    display_stats=True,
)


plot_TSNE(
    ax[1, 0],
    ePhys_tsne,
    cluster_colors[keepcells],
    "t-SNE of ePhys data",
    display_accuracy=True,
)
plot_TSNE(
    ax[1, 1],
```

```
    ePhys_tsne,
    ePhys_clusters,
    "t-SNE of ePhys data with GMM labels",
    display_accuracy=True,
)
plt.suptitle("Comparison of PCA and t-SNE of ePhys data with original
and GMM labels")
```

```
c:\Users\Arne.Gittel\AppData\Local\anaconda3\envs\final_nds_env\lib\
site-packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

```
Text(0.5, 0.98, 'Comparison of PCA and t-SNE of ePhys data with
original and GMM labels')
```



Summary

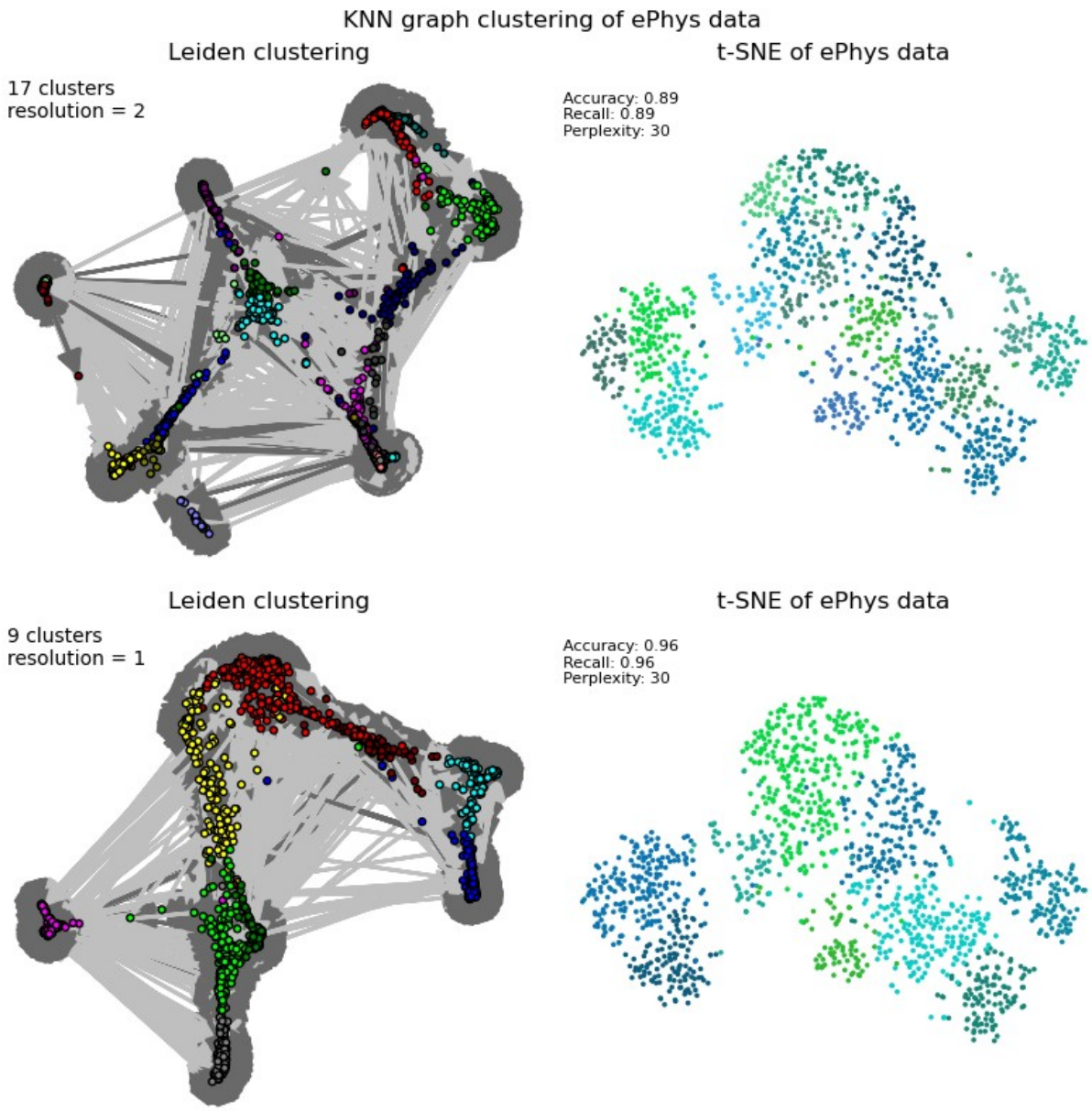
As discussed before, using GMM-clustering on raw data yields 4 clusters with high accuracy (assessed by k-nearest neighbors classification). These clusters are well separated in PCA, TSNE and UMAP spaces. Again, yellow cluster distinctively represents the Pvalb cells, while the green cluster corresponds to the excitatory cells. Blue and Purple clusters include a mixture of MGE- and CGE-derived interneurons, which are not well separable by their electrophysiological features.

Creating K-nearest neighbors graph and Leiden partition for ePhys data

```
import igraph as ig
from sklearn.neighbors import NearestNeighbors, kneighbors_graph
```

```
import leidenalg as la


clusterCols = np.unique(cluster_colors)

data = ephysData_filtered

A = kneighbors_graph(data, 10, mode="connectivity",
include_self=False)
sources, targets = A.nonzero()
G = ig.Graph(directed=True)
G.add_vertices(A.shape[0])
edges = list(zip(sources, targets))
G.add_edges(edges)

resolutions = [2, 1]

fig, ax = plt.subplots(2, 2, figsize=(8, 8))
for i in range(2):
    partition = la.find_partition(
        G,
        la.RBConfigurationVertexPartition,
        resolution_parameter=resolutions[i],
        seed=42,
    )

    ig.plot(
        partition,
        vertex_size=5,
        edge_curved=False,
        edgecolor="black",
        colors=clusterCols[partition.membership],
        target=ax[i][0],
    )
    ax[i][0].set_title("Leiden clustering")
    ax[i][0].text(
        0.025,
        0.9,
        f"{len(np.unique(partition.membership))} clusters\nresolution
= {resolutions[i]}",
        transform=ax[i][0].transAxes,
    )

    plot_TSNE(
        ax[i][1],
        ePhys_tsne,
        clusterCols[partition.membership],
        "t-SNE of ePhys data",
        display_accuracy=True,
```

```
    )
    plt.suptitle("KNN graph clustering of ePhys data")
```



KNN clustering finds 17 and 9 clusters depending on the resolution parameter.

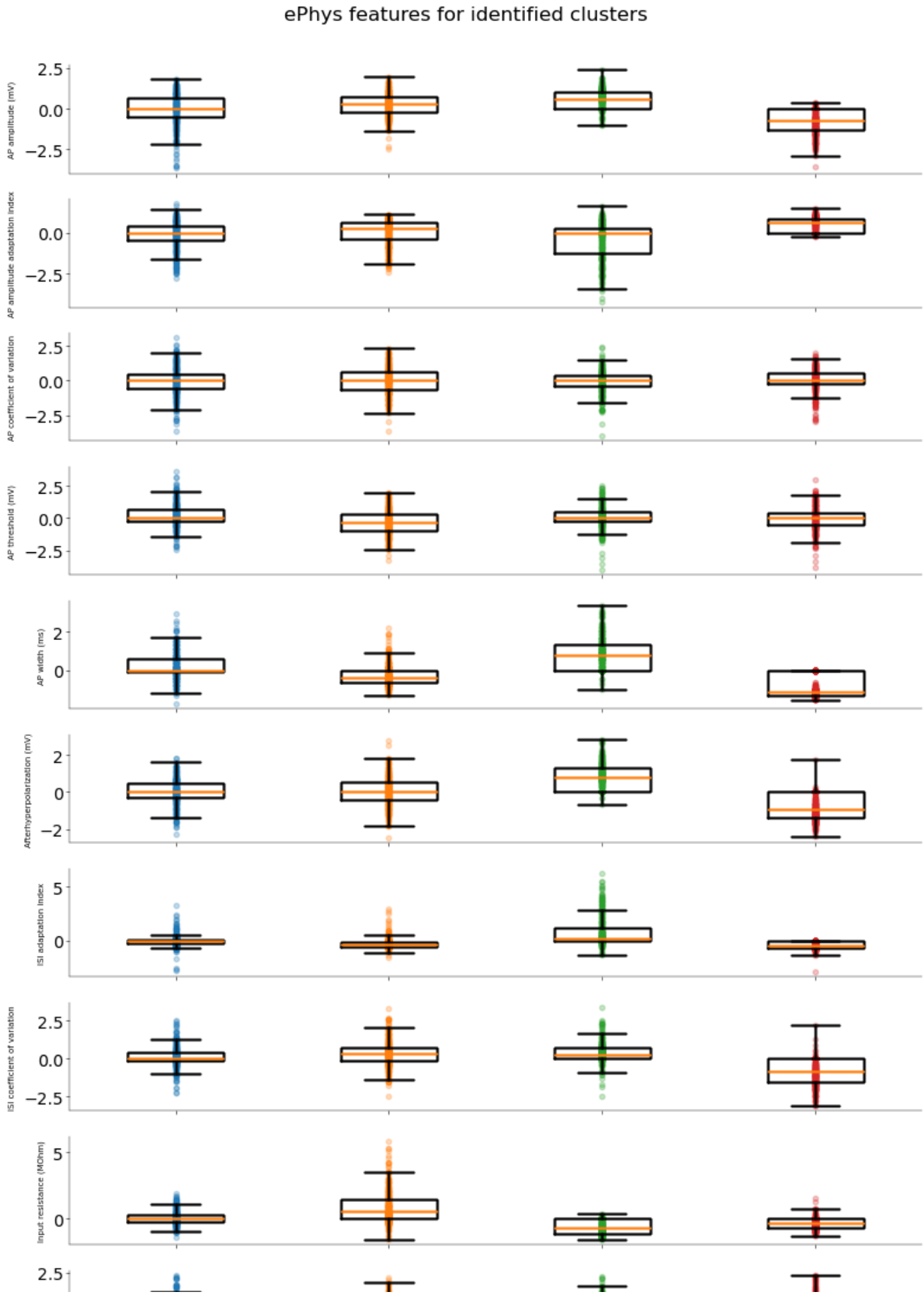
Plot selected electrophysiological features for the cells, clustered by GMM

```
data = ephysData_filtered
features = ephysNames[~np.isin(ephysNames, features_exclude)]
mean_Per_group = np.zeros((len(features),
len(np.unique(ePhys_clusters))))
std_Per_group = np.zeros((len(features),
```

```
len(np.unique(ePhys_clusters))))
for i, c in enumerate(np.unique(ePhys_clusters)):
    mean_Per_group[:, i] = np.mean(data[ePhys_clusters == c, :],
axis=0)
    std_Per_group[:, i] = np.std(data[ePhys_clusters == c, :], axis=0)


n_features = features.size
clusters_data = np.zeros((len(features),
len(np.unique(ePhys_clusters)), data.shape[0]))

unique_clusters = np.unique(ePhys_clusters)
# Determine the max number of samples in any cluster
max_samples = max(np.sum(ePhys_clusters == c) for c in
unique_clusters)

clusters_data = np.zeros((len(features), len(unique_clusters),
max_samples))

for i, c in enumerate(unique_clusters):
    cluster_data = data[ePhys_clusters == c, :].T
    clusters_data[:, i, : cluster_data.shape[1]] = cluster_data

fig, ax = plt.subplots(len(features), 1, figsize=(8, 20), sharex=True)
for i, feature in enumerate(features):
    for j, c in enumerate(np.unique(ePhys_clusters)):
        ax[i].scatter(
            np.zeros(clusters_data.shape[2]) + j,
            clusters_data[i, j, :],
            # use the same colors as for gmm clusters
            color=plt.cm.tab10.colors[j],
            alpha=0.25,
        )
        ax[i].boxplot(
            clusters_data[i, :, :].T,
            positions=np.arange(len(np.unique(ePhys_clusters))),
            showfliers=False,
            patch_artist=False,
        )

    ax[i].set_xticks(np.arange(len(np.unique(ePhys_clusters))))
    ax[i].set_xticklabels(np.unique(ePhys_clusters))
    ax[i].set_ylabel(feature, fontsize=5)
    ax[i].set_xlabel("Cluster") if i == len(features) - 1 else None
plt.suptitle("ePhys features for identified clusters", y=1.001)
plt.tight_layout()
```

```
C:\Users\Arne.Gittel\AppData\Local\Temp\
ipykernel_2668\1137455490.py:45: UserWarning: The figure layout has
```

```
changed to tight
  plt.tight_layout()
```


ePhys features for identified clusters
AP amplitude (mV)
AP amplitude adaptation index
AP coefficient of variation
AP threshold (mV)
AP width (ms)
Afterhyperpolarization (mV)
ISI adaptation index
ISI coefficient of variation
Input resistance (MOhm)

Final Remarks

Comparing ePhys and transcriptomics space we can state that we found from 3 to 4 clusters using different dimensionality reduction and normalization techniques. However not in all cases obtained clusters allign well with the t-types. Furthermore we were able to pin down some genes that might be responsible for the diversity of the observed cells.