Neural Data Science

Lecturer: Prof. Dr. Philipp Berens

Tutors: Jonas Beck, Ziwei Huang, Rita González Márquez

Summer term 2023

Name: Kathrin Root, Alexander Wendt, Patrick Weygoldt

Coding Lab 7 : Transcriptomics

```
import numpy as np
import pylab as plt
import pandas as pd
import matplotlib.pyplot as plt

# We recommend using openTSNE for experiments with t-SNE
# https://github.com/pavlin-policar/openTSNE
from openTSNE import TSNE

%matplotlib inline

%load_ext jupyter_black

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

```
The jupyter_black extension is already loaded. To reload it, use:
  %reload_ext jupyter_black
The watermark extension is already loaded. To reload it, use:
  %reload_ext watermark
Last updated: 2023-06-20 23:07:53CEST

Python implementation: CPython
Python version       : 3.11.3
IPython version      : 8.11.0

sklearn: 0.0.post1

matplotlib: 3.7.1
numpy     : 1.24.3
```

```
pandas    : 1.5.3
leidenalg : 0.9.1
igraph    : 0.10.4

Watermark: 2.3.1
```

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

Load data

Download the data from ILIAS, move it to the `data` directory and unzip it there.

```
# LOAD HARRIS ET AL DATA

# Load gene counts
data = pd.read_csv("../data/nds_cl_7/harris-data/expression.tsv.gz", sep="\t")
genes = data.values[:, 0]
cells = data.columns[1:-1]
counts = data.values[:, 1:-1].transpose().astype("int")
data = []

# Kick out all genes with all counts = 0
genes = genes[counts.sum(axis=0) > 0]
counts = counts[:, counts.sum(axis=0) > 0]
print(counts.shape)

# Load clustering results
data = pd.read_csv("../data/nds_cl_7/harris-data/analysis_results.tsv", sep="\t")
clusterNames, clusters = np.unique(data.values[0, 1:-1], return_inverse=True)

# Load cluster colors
data = pd.read_csv("../data/nds_cl_7/harris-data/colormap.txt", sep="\s+", header=None)
clusterColors = data.values

# Note: the color order needs to be reversed to match the publication
clusterColors = clusterColors[::-1]

# Taken from Figure 1 - we need cluster order to get correct color order
clusterOrder = [
    "Sst.No",
```

```
"Sst.Npy.C",
"Sst.Npy.Z",
"Sst.Npy.S",
"Sst.Npy.M",
"Sst.Pnoc.Calb1.I",
"Sst.Pnoc.Calb1.P",
"Sst.Pnoc.P",
"Sst.Erbb4.R",
"Sst.Erbb4.C",
"Sst.Erbb4.T",
"Pvalb.Tac1.N",
"Pvalb.Tac1.Ss",
"Pvalb.Tac1.Sy",
"Pvalb.Tac1.A",
"Pvalb.C1ql1.P",
"Pvalb.C1ql1.C",
"Pvalb.C1ql1.N",
"Cacna2d1.Lhx6.R",
"Cacna2d1.Lhx6.V",
"Cacna2d1.Ndnf.N",
"Cacna2d1.Ndnf.R",
"Cacna2d1.Ndnf.C",
"Calb2.Cry",
"Sst.Cry",
"Ntng1.S",
"Ntng1.R",
"Ntng1.C",
"Cck.Sema",
"Cck.Lmo1.N",
"Cck.Calca",
"Cck.Lmo1.Vip.F",
"Cck.Lmo1.Vip.C",
"Cck.Lmo1.Vip.T",
"Cck.Ly",
"Cck.Cxcl14.Calb1.Tn",
"Cck.Cxcl14.Calb1.I",
"Cck.Cxcl14.S",
"Cck.Cxcl14.Calb1.K",
"Cck.Cxcl14.Calb1.Ta",
"Cck.Cxcl14.V",
"Vip.Crh.P",
```

```
    "Vip.Crh.C1",
    "Calb2.Vip.G",
    "Calb2.Vip.I",
    "Calb2.Vip.Nos1",
    "Calb2.Cntnap5a.R",
    "Calb2.Cntnap5a.V",
    "Calb2.Cntnap5a.I",
]

reorder = np.zeros(clusterNames.size) * np.nan
for i, c in enumerate(clusterNames):
    for j, k in enumerate(clusterOrder):
        if c[: len(k)] == k:
            reorder[i] = j
            break
clusterColors = clusterColors[reorder.astype(int)]
```

```
(3663, 17965)
```

1. Data inspection

Before we use t-SNE or any other advanced visualization methods on the data, we first want to have a closer look on the data and plot some statistics. For most of the analysis we will compare the data to a Poisson distribution.

1.1. Relationship between expression mean and fraction of zeros

The higher the average expression of a gene, the smaller fraction of cells will show a 0 count.

(2pt.)

```
# ------------------------------------------------------
# Compute actual and predicted gene expression (1.5 pts)
# ------------------------------------------------------

# Compute the average expression for each gene
avg_expression = np.mean(counts, axis=0)
print(counts.shape, avg_expression.shape)
print(counts)
```

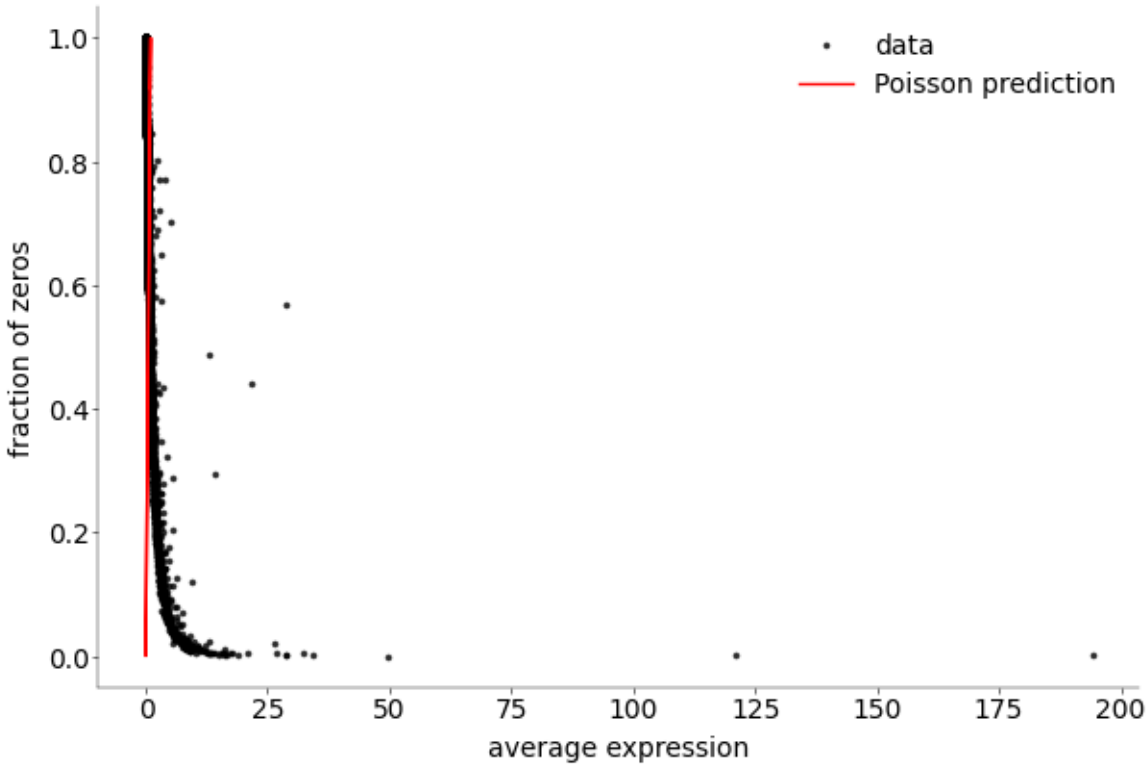
```
# Compute the fraction of zeros for each gene
fraction_zeros = np.sum(counts == 0, axis=0) / counts.shape[0]
```

```
(3663, 17965) (17965,)
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 2 0 0]
 [0 0 0 ... 1 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 2 1 0]
 [0 0 0 ... 2 0 0]]
```

```
# Compute the Poisson prediction

# (what is the expected fraction of zeros in a Poisson distribution with a given mean?)

poisson_prediction = np.exp(-avg_expression)
```

```
# ---------------------------------------------------
# plot the data and the Poisson prediction (0.5 pts)
# ---------------------------------------------------

fig, ax = plt.subplots(figsize=(6, 4))
ax.scatter(avg_expression, fraction_zeros, s=10, alpha=0.7, label="data", c="black")
ax.plot(poisson_prediction, poisson_prediction, label="Poisson prediction", c="red")
ax.set_xlabel("average expression")
ax.set_ylabel("fraction of zeros")
ax.legend()

# add plot
```

```
<matplotlib.legend.Legend at 0x17eacdc90>
```



1.2. Mean-variance relationship

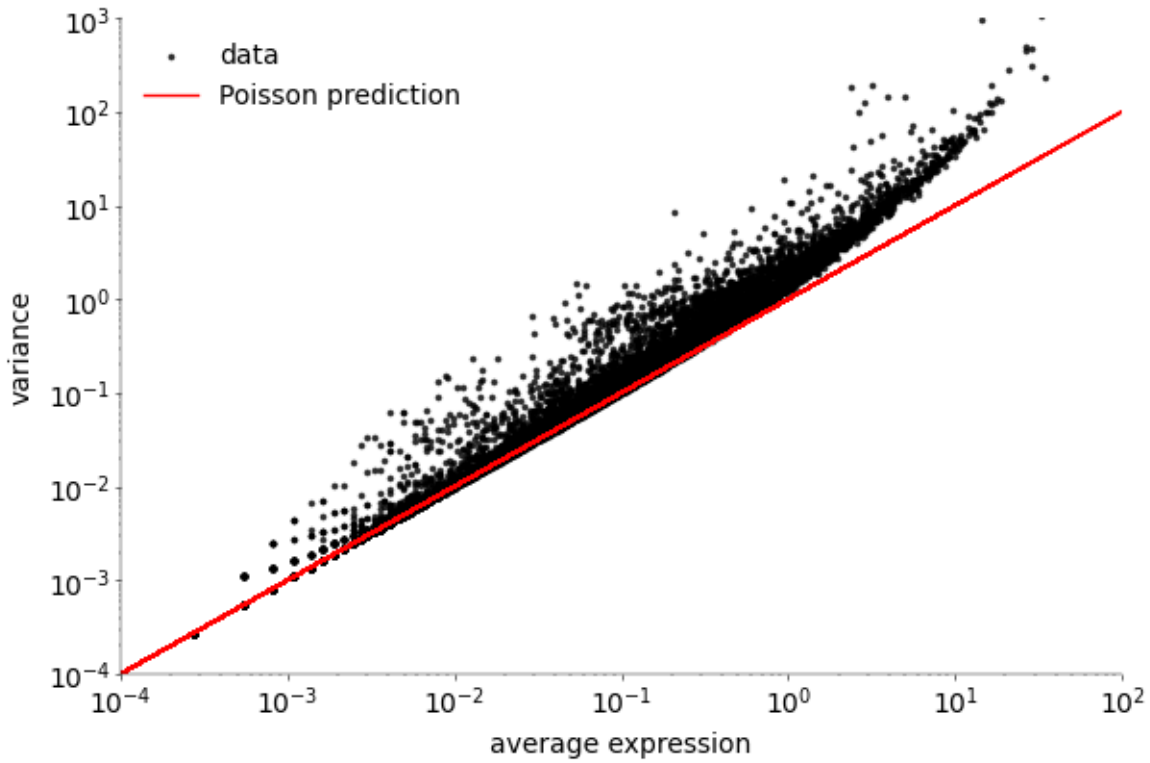
If the expression follows Poisson distribution, then the mean should be equal to the variance. *(1pt.)*

```
# Compute the variance of the expression counts of each gene
variance_expression = np.var(counts, axis=0)
print(variance_expression.shape)
```

```
(17965,)
```

```
# Plot the mean-variance relationship on a log-log plot
# Plot the Poisson prediction as a line
poisson_prediction = np.random.poisson(avg_expression)

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

```
# ----------------------------
# plot variance vs mean (1 pt)
# incl. computing mean and var
# ----------------------------
ax.scatter(
    avg_expression, variance_expression, s=10, alpha=0.7, label="data", c="black"
)

ax.plot(
    poisson_prediction,
    poisson_prediction,
    label="Poisson prediction",
    c="red",
)
ax.set_xlabel("average expression")
ax.set_ylabel("variance")
ax.set_ylim(10e-5, 10e2)
ax.set_xlim(10e-5, 10e1)
ax.legend()
ax.set_xscale("log")
ax.set_yscale("log")
```



1.3. Relationship between the mean and the Fano factor

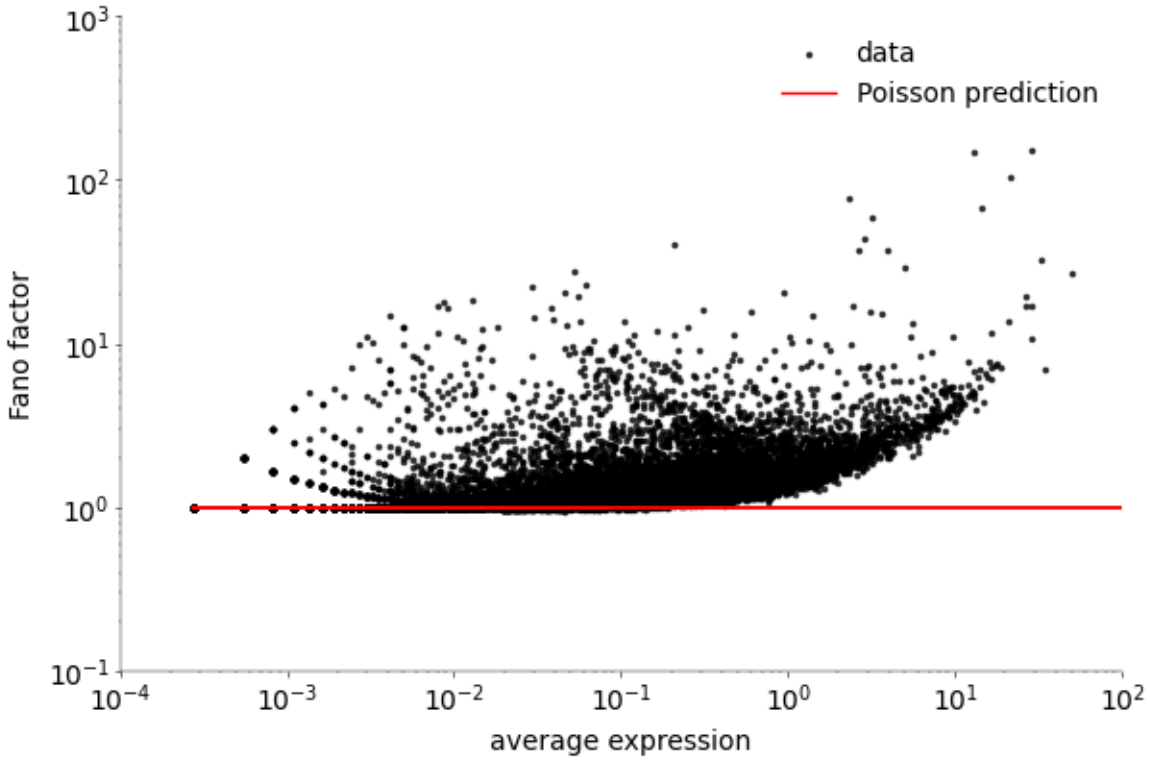
If the expression follows the Poisson distribution, then the Fano factor (variance/mean) should be equal to 1 for all genes.

(1pt.)

```
# Compute the Fano factor for each gene and make a scatter plot
# of expression mean vs. Fano factor in log-log coordinates.
fano = variance_expression / avg_expression
```

```
# Plot a Poisson prediction as line
# Use the same style of plot as above.
fig, ax = plt.subplots(figsize=(6, 4))

# --------------------------------
# plot fano-factor vs mean (1 pt)
# incl. fano factor
```

```
# ------------------------------

ax.scatter(avg_expression, fano, s=10, alpha=0.7, label="data", c="black")
ax.plot(
    avg_expression,
    np.ones_like(avg_expression),
    label="Poisson prediction",
    c="red",
)
ax.set_xlabel("average expression")
ax.set_ylabel("Fano factor")
ax.set_ylim(10e-2, 10e2)
ax.set_xlim(10e-5, 10e1)
ax.legend()
ax.set_xscale("log")
ax.set_yscale("log")
```



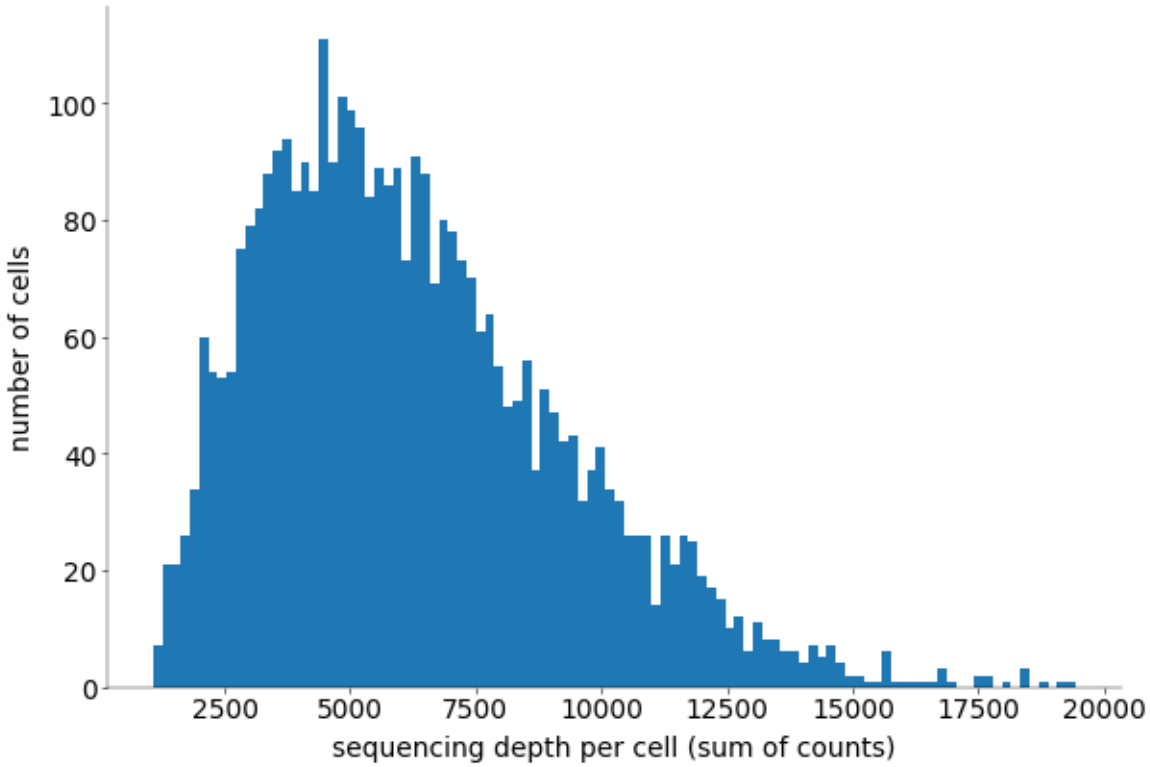
1.4. Histogram of sequencing depths

Different cells have different sequencing depths (sum of counts across all genes) because the efficiency can change from droplet to droplet due to some random expreimental factors.

(1pt.)

```
# Make a histogram of sequencing depths across cells.
# Sequencing depth of each cell is the sum of all counts of this cell

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

# ----------------------------------------
# Plot histogram of sequencing depths (1 pt)
# ----------------------------------------
ax.hist(np.sum(counts, axis=1), bins=100)
ax.set_xlabel("sequencing depth per cell (sum of counts)")
ax.set_ylabel("number of cells")
```

```
Text(0, 0.5, 'number of cells')
```



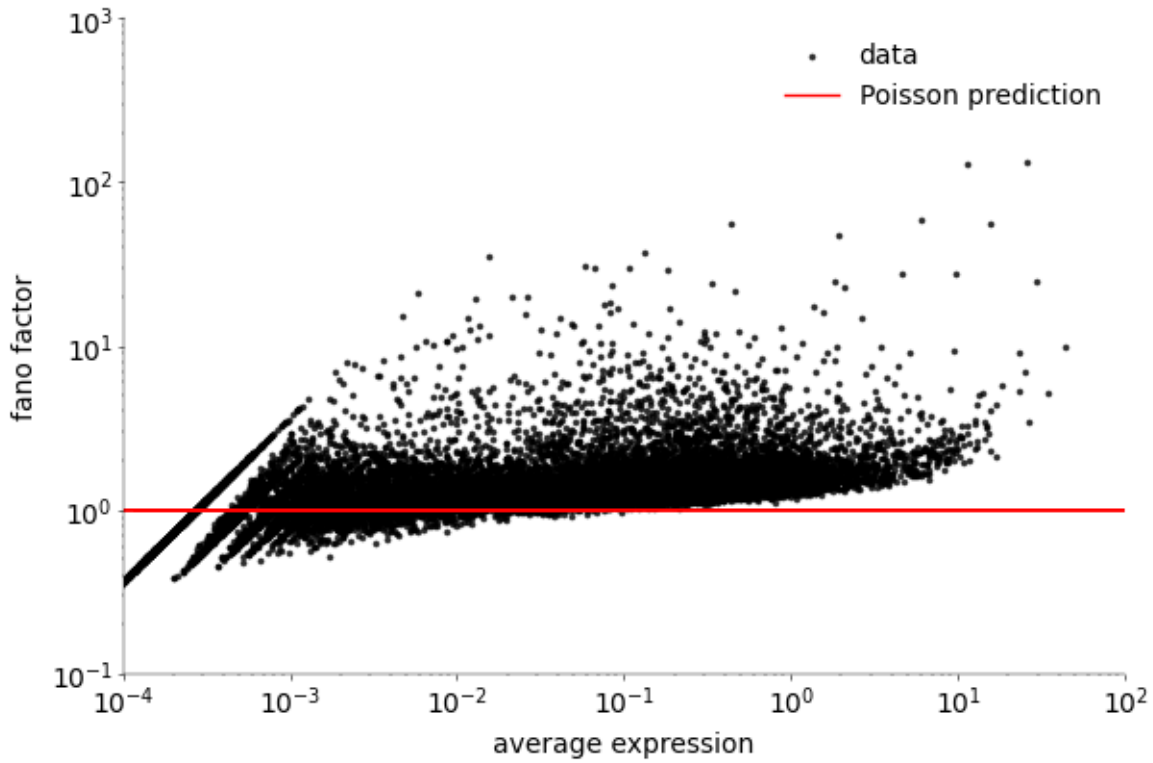
1.5. Fano factors after normalization

After normalization by sequencing depth, Fano factor should be closer to 1 (i.e. variance even more closely following the mean). This can be used for feature selection.

(1pt.)

```
# Normalize counts by the sequencing depth of each cell and multiply by the median sequenc
# Then make the same expression vs Fano factor plot as above

# -----------------------------------
# compute normalized counts (0.5 pts)
# -----------------------------------
norm_counts = (
    counts / np.sum(counts, axis=1)[:, None] * np.median(np.sum(counts, axis=1))
)
fano = np.var(norm_counts, axis=0) / np.mean(norm_counts, axis=0)
```

```
# ------------------------------------------------------------
# plot normalized counts and find the top 10 genes (0.5 pts)
# hint: keep appropriate axis scaling in mind
# ------------------------------------------------------------

fig, ax = plt.subplots(figsize=(6, 4))
ax.scatter(np.mean(norm_counts, axis=0), fano, s=10, alpha=0.7, label="data", c="black")
ax.plot(
    np.mean(norm_counts, axis=0),
    np.ones_like(np.mean(norm_counts, axis=0)),
    label="Poisson prediction",
    c="red",
)
ax.set_xlabel("average expression")
ax.set_ylabel("fano factor")
ax.set_ylim(10e-2, 10e2)
ax.set_xlim(10e-5, 10e1)
ax.legend()
ax.set_xscale("log")
ax.set_yscale("log")

# add plot
```




```
# Find top-10 genes with the highest normalized Fano factor
# Print them sorted by the Fano factor starting from the highest
# Gene names are stored in the `genes` array
sorted_fano_index = np.argsort(fano)[::-1]

print(fano[sorted_fano_index[:10]])
print(genes[sorted_fano_index[:10]])
```

```
[131.14152764 128.39902525  59.11491488  55.65105499  55.22031716
  47.65007481  37.24624683  35.05542058  31.42260973  30.58952939]
['Sst' 'Npy' 'Vip' 'Cck' 'Cpne2' 'Pcp4' 'Ptpn23' 'Pdzd9' 'Malat1' 'Armc2']
```

2. Low dimensional visualization

Here we look at the influence of variance-stabilizing transformations on PCA and t-SNE.

2.1. PCA with and without transformations

Square root is a variance-stabilizing transformation for the Poisson data. Log-transform is also often used in the transcriptomic community. Look at the effect of both.

(1pt.)

```
# ----------------------------------------
# transform data and apply PCA (0.5 pts)
# ----------------------------------------

# Transform the counts into normalized counts (as above)
norm_counts = (
    counts / np.sum(counts, axis=1)[:, None] * np.median(np.sum(counts, axis=1))
)
fano = np.var(norm_counts, axis=0) / np.mean(norm_counts, axis=0)

# Select all genes with the normalized Fano factor above 3 and remove the rest
# extract all genes with fano > 3 and counts matrix
print(norm_counts.shape)
print(fano.shape)
print(fano > 3)
extracted_counts = norm_counts[:, fano > 3]
```

```
(3663, 17965)
(17965,)
[False False False ... False False False]
```

```
# Perform PCA three times: on the resulting matrix as is,
# after np.log2(X+1) transform, and after np.sqrt(X) transform

extracted_counts = norm_counts[:, fano > 3]

# log2(x+1) transform
log_counts = np.log2(extracted_counts + 1)
print(log_counts.shape)
# sqrt(x) transform
sqrt_counts = np.sqrt(extracted_counts)

from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=2, random_state=42)

# PCA on extracted_counts
X_pca = pca.fit_transform(extracted_counts)
print(X_pca.shape)
# PCA on log_counts
X_pca_log = pca.fit_transform(log_counts)
# PCA on sqrt_counts
X_pca_sqrt = pca.fit_transform(sqrt_counts)
```

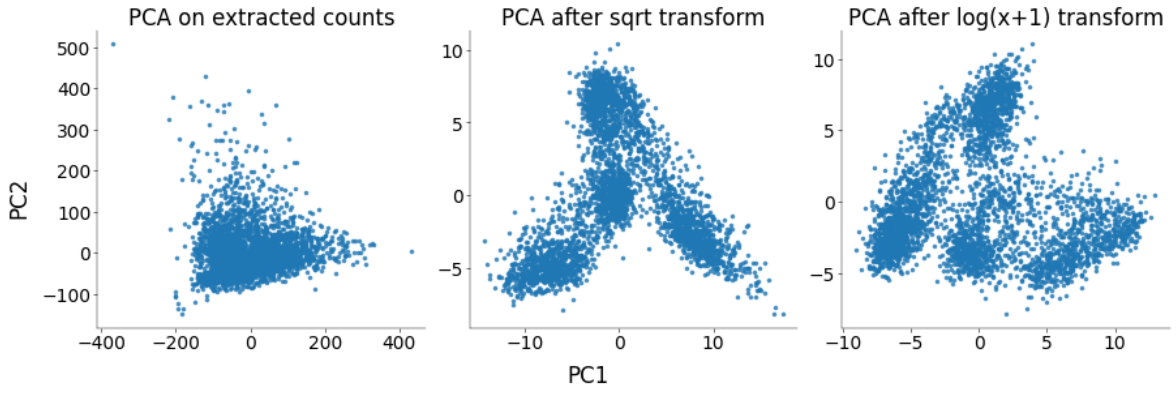
```
(3663, 707)
(3663, 2)
```

```
# ------------------------------------------
# plot first 2 PCs for each dataset (0.5 pts)
# ------------------------------------------

fig, axs = plt.subplots(1, 3, figsize=(9, 3))
print(X_pca.shape)
print(X_pca)

# add plot
axs[0].scatter(
    X_pca[:, 0],
    X_pca[:, 1],
    s=10,
    alpha=0.7,
    label="data",
)
axs[0].set_title("PCA on extracted counts")

axs[1].scatter(
    X_pca_sqrt[:, 0],
    X_pca_sqrt[:, 1],
    s=10,
    alpha=0.7,
    label="data",
)
axs[1].set_title("PCA after sqrt transform")
```

```
axs[2].scatter(
    X_pca_log[:, 0],
    X_pca_log[:, 1],
    s=10,
    alpha=0.7,
    label="data",
)
axs[2].set_title("PCA after log(x+1) transform")


fig.supxlabel("PC1")
fig.supylabel("PC2")
```

```
(3663, 2)
[[  98.25882891  -18.32369821]
 [   9.17280852  -14.20738483]
 [  39.28224513   -6.69135392]
 ...
 [ -99.11108152  -13.48731125]
 [-136.22189058   19.74332623]
 [-118.71500399   43.06724293]]
```

```
Text(0.02, 0.5, 'PC2')
```



2.2. tSNE with and without transformations

Do these transformations have any effect on t-SNE?

(1pt.)

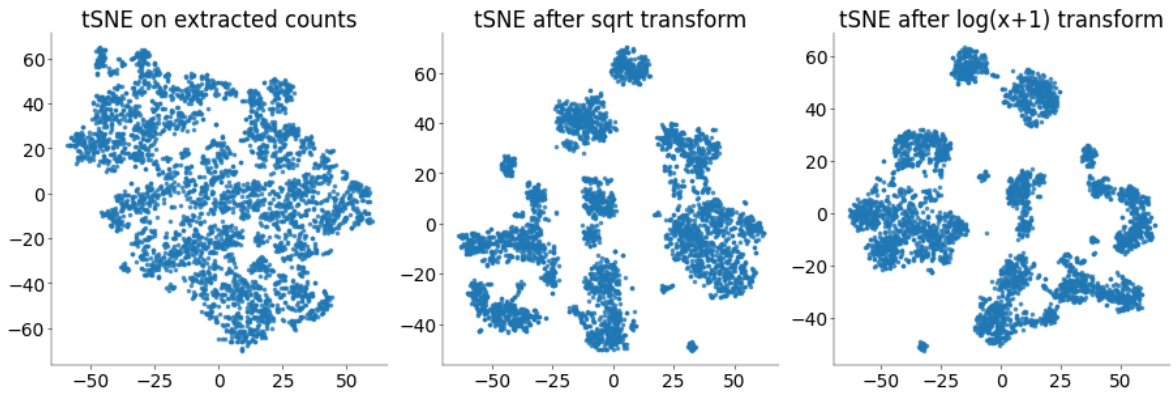
```
# ------------------------------------------------------------------------
# Perform tSNE three times: on the resulting matrix as is,
# after np.log2(X+1) transform, and after np.sqrt(X) transform. (0.5 pts)
# ------------------------------------------------------------------------

# Apply t-SNE to the 50 PCs

# Use default settings of openTSNE
tsne = TSNE(random_state=42)
# 50 PCs
# tSNE on extracted_counts
pca = PCA(n_components=50, random_state=42)

X_pca = pca.fit_transform(extracted_counts)
X_tsne = tsne.fit(X_pca)
print(X_tsne.shape)
# tSNE on log_counts
X_pca_log = pca.fit_transform(log_counts)
X_tsne_log = tsne.fit(X_pca_log)
# tSNE on sqrt_counts
X_pca_sqrt = pca.fit_transform(sqrt_counts)
X_tsne_sqrt = tsne.fit(X_pca_sqrt)


# You can also use sklearn if you want
```

```
(3663, 2)
```

```
# ------------------------------------------------
# plot t-SNE embedding for each dataset (0.5 pts)
# ------------------------------------------------

fig, axs = plt.subplots(1, 3, figsize=(9, 3))
axs[0].scatter(X_tsne[:, 0], X_tsne[:, 1], s=10, alpha=0.7, label="data")
axs[0].set_title("tSNE on extracted counts")
axs[1].scatter(X_tsne_sqrt[:, 0], X_tsne_sqrt[:, 1], s=10, alpha=0.7, label="data")
axs[1].set_title("tSNE after sqrt transform")
axs[2].scatter(X_tsne_log[:, 0], X_tsne_log[:, 1], s=10, alpha=0.7, label="data")
axs[2].set_title("tSNE after log(x+1) transform")
```

```
# add plot
```

```
Text(0.5, 1.0, 'tSNE after log(x+1) transform')
```



2.3. Leiden clustering

This dataset is small and can be clustered in many different ways. We will apply Leiden clustering (closely related to the Louvain clustering), which is standard in the field and works well even for very large datasets.

(1pt.)

```
# To run this code you need to install leidenalg and igraph
# conda install -c conda-forge python-igraph leidenalg

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

```
# Define some contrast colors

clusterCols = [
    "#FFFF00",
    "#1CE6FF",
    "#FF34FF",
    "#FF4A46",
    "#008941",
```

```
    "#006FA6",
    "#A30059",
    "#FFDBE5",
    "#7A4900",
    "#0000A6",
    "#63FFAC",
    "#B79762",
    "#004D43",
    "#8FB0FF",
    "#997D87",
    "#5A0007",
    "#809693",
    "#FEFFE6",
    "#1B4400",
    "#4FC601",
    "#3B5DFF",
    "#4A3B53",
    "#FF2F80",
    "#61615A",
    "#BA0900",
    "#6B7900",
    "#00C2A0",
    "#FFAA92",
    "#FF90C9",
    "#B903AA",
    "#D16100",
    "#DDEFFF",
    "#000035",
    "#7B4F4B",
    "#A1C299",
    "#300018",
    "#0AA6D8",
    "#013349",
    "#00846F",
    "#372101",
    "#FFB500",
    "#C2FFED",
    "#A079BF",
    "#CC0744",
    "#C0B9B2",
    "#C2FF99",
```

```
    "#001E09",
    "#00489C",
    "#6F0062",
    "#0CBD66",
    "#EEC3FF",
    "#456D75",
    "#B77B68",
    "#7A87A1",
    "#788D66",
    "#885578",
    "#FAD09F",
    "#FF8A9A",
    "#D157A0",
    "#BEC459",
    "#456648",
    "#0086ED",
    "#886F4C",
    "#34362D",
    "#B4A8BD",
    "#00A6AA",
    "#452C2C",
    "#636375",
    "#A3C8C9",
    "#FF913F",
    "#938A81",
    "#575329",
    "#00FECF",
    "#B05B6F",
    "#8CD0FF",
    "#3B9700",
    "#04F757",
    "#C8A1A1",
    "#1E6E00",
    "#7900D7",
    "#A77500",
    "#6367A9",
    "#A05837",
    "#6B002C",
    "#772600",
    "#D790FF",
    "#9B9700",
```

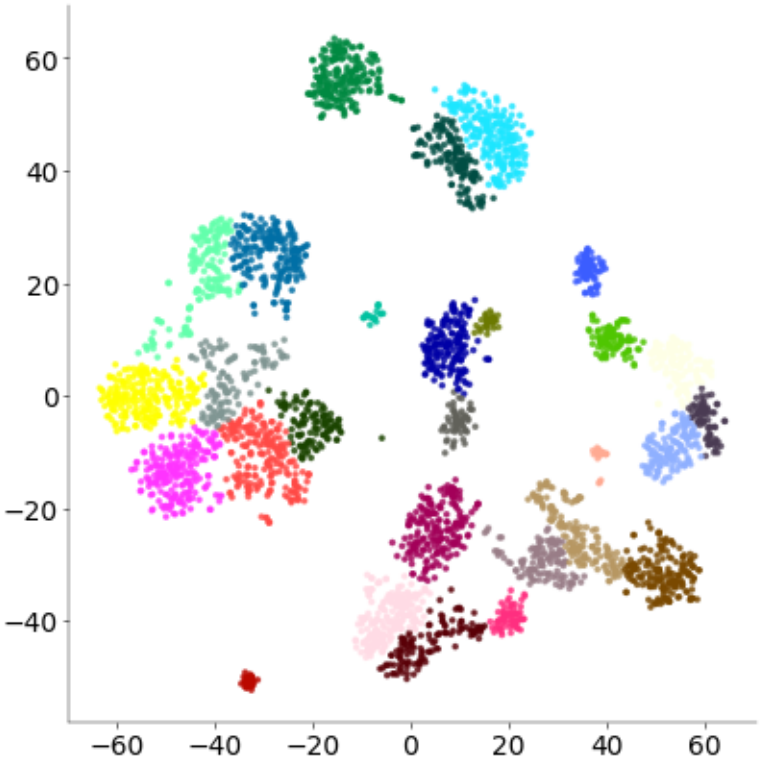
```
    "#549E79",
    "#FFF69F",
    "#201625",
    "#72418F",
    "#BC23FF",
    "#99ADC0",
    "#3A2465",
    "#922329",
    "#5B4534",
    "#FDE8DC",
    "#404E55",
    "#0089A3",
    "#CB7E98",
    "#A4E804",
    "#324E72",
    "#6A3A4C",
    "#83AB58",
    "#001C1E",
    "#D1F7CE",
    "#004B28",
    "#C8D0F6",
    "#A3A489",
    "#806C66",
    "#222800",
    "#BF5650",
    "#E83000",
    "#66796D",
    "#DA007C",
    "#FF1A59",
    "#8ADBB4",
    "#1E0200",
    "#5B4E51",
    "#C895C5",
    "#320033",
    "#FF6832",
    "#66E1D3",
    "#CFCDAC",
    "#D0AC94",
    "#7ED379",
    "#012C58",
]
```

```
# ------------------------------------------------------
# create graph and run leiden clustering on it (0.5 pts)
# hint: use `la?`, `la.find_partition?` and `ig.Graph?`
# to find out more about the provided packages.
# ------------------------------------------------------

# Construct kNN graph with k=15
k = 15
A = kneighbors_graph(X_tsne_log, k, mode="connectivity", include_self=True)
sources, targets = A.nonzero()
m = A.nonzero()

# Transform it into an igraph object
G = ig.Graph()
num_vertices = max(max(sources), max(targets)) + 1
G.add_vertices(range(len(X_tsne_log)))
# G.add_edges(list(zip(sources, targets)))
G.add_edges(list(zip(m[0], m[1])))
```

```
# Run Leiden clustering
# you can use `la.RBConfigurationVertexPartition` as the partition type

partition = la.find_partition(G, la.RBConfigurationVertexPartition, seed=42)
```

```
# --------------------------
# Plot the results (0.5 pts)
# --------------------------

fig, ax = plt.subplots(figsize=(4, 4))
# Get the community labels assigned by the algorithm
community_labels = partition.membership
# Plot the communities
ax.scatter(
    X_tsne_log[:, 0],
    X_tsne_log[:, 1],
    s=10,
    alpha=0.7,
    label="data",
    c=[clusterCols[i] for i in community_labels],
)
```

```
<matplotlib.collections.PathCollection at 0x2bb2c6550>
```



2.4. Change the clustering resolution

The number of clusters can be changed by modifying the resolution parameter.

(1pt.)

```
# How many clusters did we get?
print("Number of clusters: ", np.unique(community_labels).size)
# Change the resolution parameter to yield 2x more and 2x fewer clusters
# Plot all three results as tSNE overlays (as above)

# ------------------------------------------------------------------
# run the clustering for 3 different resolution parameters (0.5 pts)
# ------------------------------------------------------------------

fig, ax = plt.subplots(1, 3, figsize=(12, 4))
```

```
resolution_parameters = [0.11, 1, 2.88]
for i, res in enumerate(resolution_parameters):
    partition = la.find_partition(
        G, la.RBConfigurationVertexPartition, resolution_parameter=res, seed=42
    )
    community_labels = partition.membership

    ax[i].scatter(
        X_tsne_log[:, 0],
        X_tsne_log[:, 1],
        s=10,
        alpha=0.7,
        label="data",
        c=[clusterCols[i] for i in community_labels],
    )
    print("Number of clusters: ", np.unique(community_labels).size)
    ax[i].set_title(
        f"Resolution parameter:{res}\n Cluster size: {np.unique(community_labels).size}"
    )
```

```
Number of clusters:  28
Number of clusters:  14
Number of clusters:  28
Number of clusters:  55
```