Welcome to Group 2's MED263 tutorial on jointly analyzing scRNA-seq and CITE-seq data from different donors.

Group 2: Jenny Dong, Rebecca Kirby, Alexander Monell, Evelyn Xu

Github: @jennydongwx, @rkirby44479, @amonell, @EvelynXzy

Introduction and Significance/Background:

Single cell RNA sequencing (scRNA-seq) is a powerful technology in which RNA transcripts from individual cells are quantified in an isoform aware manner. In the biomedical field, scRNA-seq is used for several applications, including validating genetic hits in GWAS studies by cell-type specific eQTL mapping[1]. CITE-seq is a modified version of scRNA-seq that includes measuring RNA abundances, as well as surface protein quantification by tagging cells with oligo-conjugated antibodies before sequencing. In this tutorial, we will explore how to impute protein expression values into scRNA-seq data from the same sample type through joint embedding and integration. Furthermore, we will conduct an analysis on the joint data to find which proteins and genes are expressed in each cell type.

We are putting a spin on a standard scRNA-seq analysis tutorial by incorporating joint integration and an introduction to CITE-seq data. The use of multimodal data integration has shown benefits in biomedical research, including one study we found, where the authors used an approach to find genes to prioritize in diseases that have no clear candidate genes by using integrated protein and gene expression data [2]. They found that the integrated approach produced better results than either independently, and applied this to find genes of interest in Alzheimer's disease.

While the findings of our tutorial won't be biologically novel, the tutorial will serve as a pipeline that students can follow in their own biomedical research. As single cell sequencing becomes more prevalent, researchers may often find themselves with data collected through multiple single cell modalities across hundreds of individuals. This data needs to be integrated in a way where meaningful analyses can be performed downstream. In GWAS studies, certain SNPs may be implicated in causing a disease, but only in a fraction of the population. Single cell data collection can help us understand why genetic mutations have differing effects across demographics, and also inform what is happening at the RNA and protein level (druggable targets).

Tutorial Overview

We break this tutorial down into 4 aims. The biological implications and details of each aim are addressed as you progress through the tutorial. The aims are as follows:

Aim 1: Preprocess scRNA-seq and CITE-seq data separately.

Aim 2: Integration of CITE-seq and RNA-seq data.

Aim 3: Imputing protein quantification values into scRNA-seq data.

Aim 4: Correlating RNA and Protein expression in CITE-seq data.

Methods

Software

All analysis was conducted in a Jupyter notebook in a Python 3.9 environment. For preprocessing, we used pandas and scanpy. For integration, we used scanpy, scvi-tools, and ucell. To impute protein expression values, we used scipy-spatial and numpy. To plot coexpression, we used scanpy, mygene, matplotlib, and seaborn.

You can install packages with the following code block:

In [ ]:

```
#Run this to install packages.
!pip install pandas==2.1.1
!pip install git+https://github.com/maximilian-heeg/UCell.git
!pip install scanpy
!pip install numpy
!pip install scvi-tools
!pip install --upgrade setuptools
!pip install mygene
```

You can import packages with the following code block:

In [ ]:

```
import os
import scanpy as sc
import numpy as np
import pandas as pd
from scipy.spatial import KDTree
import seaborn as sns
import matplotlib.pyplot as plt
import ucell
import mygene
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

You can check that you have the correct pacakge versions with the following code block:

In [ ]:

```
# run this block to check that your installation is successful
import pkg_resources

required_packages = {
    'numpy': '>=1.25.0',
    'pandas': '>=1.5.3',
    'scanpy': '>=1.9.8',
    'scvi-tools': '>=1.1.2',
    'matplotlib': '>=3.7.1',
    'ucell': '>=0.1',
    'seaborn': '>=0.13.0',
    'mygene': '>=3.2.2',
}

for package, version_range in required_packages.items():
    try:
        installed_version = pkg_resources.get_distribution(package).version
        version_requirement = pkg_resources.Requirement.parse(f"{package}{version_range}
")
        assert pkg_resources.parse_version(installed_version) in version_requirement, \
            f"{package} version must be within {version_range}, found version {installed
_version}"
    except pkg_resources.DistributionNotFound:
        assert False, f"{package} is not installed."

print("All required packages are installed and within the specified version ranges.")
```

Data

This tutorial uses two publicly available datasets PBMC3k and PBMC5k, a scRNA-seq and CITE-seq dataset respectively. In these datasets, Peripheral Blood Mononuclear Cells were measured using the 10x Genomics single cell platform. The main difference between the datasets is that PBMC5k contains protein expression data since it was measured with CITE-seq. These datasets are easy to download from the 10x Genomics website, and importantly have already been mapped to the genome.

Raw data was accessed from 10x Genomics at the following URLs: PBMC3K:

http://cf.10xgenomics.com/samples/cell-exp/1.1.0/pbmc3k/pbmc3k_filtered_gene_bc_matrices.tar.gz
PBMC5K CITE-seq: http://cf.10xgenomics.com/samples/cell-exp/3.1.0/5k_pbmc_protein_v3_nextgem/5k_pbmc_protein_v3_nextgem_filtered_feature_bc_matrix.h5

Integrated data for the in-class tutorial starting at step 3 can be accessed via google drive: scRNA: https://drive.google.com/file/d/1L-cGY75HJoPmiDF7q5hRjXMas3YbNUmD/view?usp=share_link
CITE-Seq: https://drive.google.com/file/d/1kksWD7atJlxbrOMJYYkKyY1RM59QZ2Ej/view?usp=share_link

Tutorial

Aim 1: Preprocess scRNA-seq and CITE-seq data separately.

This section serves to read in and filter the 2 data sets for downstream integration and analysis. This is not part of the hands-on in-class tutorial, but we will go over the steps we took to prepare the data. The main aim of preprocessing is to remove outliers; for example, dead cells, possible doublets, cells with very low read count, and genes that are very lowly expressed. This allows us to focus on the biologically relevant data. Additionally, normalization steps help account for differences in read depth between cells and will allow for accurate differential expression analysis. We move forward with the cleaned files in the proceeding aims.

Biological Interpetation:

High mitochondrial gene count in cells can be an indicator of dead cells, and cells with very high read counts are possible doublets, an artifact of the scRNA-seq technology. We want to focus on genes that are highly expressed, as these are likely to be biologically relevant.

Mathematical/statistical meaning of analysis

For filtering, we stuck to common practices of scRNA-seq, as well as double checking our parameters by visualizing the distribution of the different variables to confirm we were not filtering out meaningful cells. For example, this is why our mitochondrial gene percentage is set a <5% for PBMC3K, and <15% for PBMC5k.

Common pitfalls

If using Python version 3.11, you may run into a kernel crashing when running "sc.pp.calculate_qc_metrics". This can be fixed by changing to an environment using Python 3.9. To check which version you are using, put python --version into the terminal. To make a new environment, refer to the steps in our README.md. If you get an error when reading in the data, make sure that you are in the correct working directory and/or that your file path is correct.

Read in PMBC3k data

- **This is scRNA-seq data from 3k peripheral blood mononuclear cells (PBMCs) from a healthy donor.**

In [4]:

```
%%bash
mkdir data #make new directory
wget http://cf.10xgenomics.com/samples/cell-exp/1.1.0/pbmc3k/pbmc3k_filtered_gene_bc_matrices.tar.gz -O data/pbmc3k_filtered_gene_bc_matrices.tar.gz #pull data from internet via url
cd data; tar -xzf pbmc3k_filtered_gene_bc_matrices.tar.gz

#taken from scanpy cite-seq tutorial
```

```
mkdir: data: File exists
--2024-03-11 23:06:26--  http://cf.10xgenomics.com/samples/cell-exp/1.1.0/pbmc3k/pbmc3k_filtered_gene_bc_matrices.tar.gz
Resolving cf.10xgenomics.com (cf.10xgenomics.com)... 2606:4700::6812:1ad, 2606:4700::6812:ad, 104.18.0.173, ...
Connecting to cf.10xgenomics.com (cf.10xgenomics.com)|2606:4700::6812:1ad|:80...
```

```
Process is interrupted.
```

In [8]:

```
#read in pbmc3k data
pbmc3k = sc.read_10x_mtx(
    "data/filtered_gene_bc_matrices/hg19/",  # the directory with the `.mtx` file
    var_names="gene_symbols",  # use gene symbols for the variable names (variables-axis index)
    cache=True,  # write a cache file for faster subsequent reading
)
```

Lets look at our data briefly to see what we are working with!

First we need to create 2 observations, n_counts (number of reads per cell) and n_genes (number of genes per cell).

In [9]:

```
# create n_counts and n_genes
pbmc3k.obs['n_counts'] = pbmc3k.X.sum(1)
pbmc3k.obs['n_genes'] = (pbmc3k.X > 0).sum(1)
```

In [11]:

```
pbmc3k.obs.head() #look at the data
```

Out[11]:

	n_counts	n_genes
AAACATACAACCAC-1	2421.0	781
AAACATTGAGCTAC-1	4903.0	1352
AAACATTGATCAGC-1	3149.0	1131
AAACCGTGCTTCCG-1	2639.0	960
AAACCGTGTATGCG-1	981.0	522

Each row represents a singular cell, and here we can see the number of reads and number of genes for each cell.

Using .head() we only look at the first 5 observations, so lets see how big our adata object actually is.

In [12]:

```
pbmc3k.shape
```

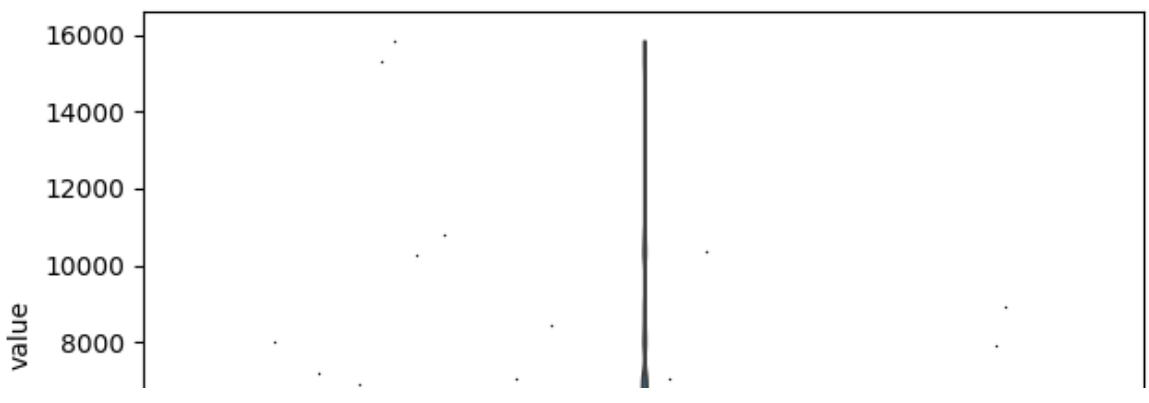
Out[12]:

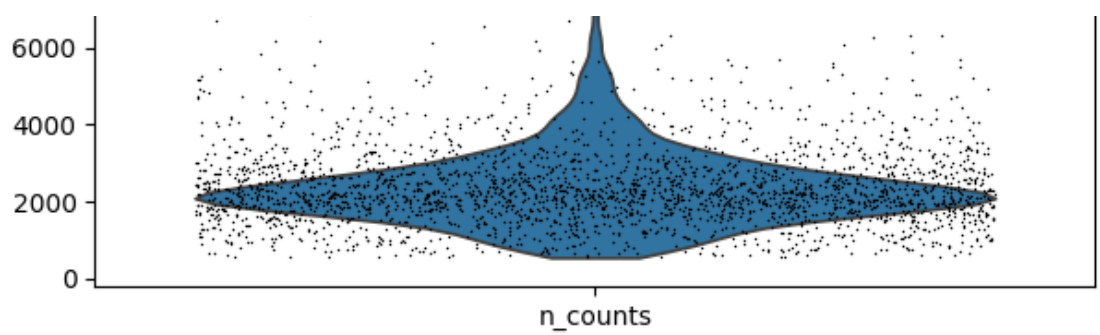
```
(2700, 32738)
```

To get a better sense of the distribution of n_counts and n_genes, we can visualize using violin plots.

In [13]:

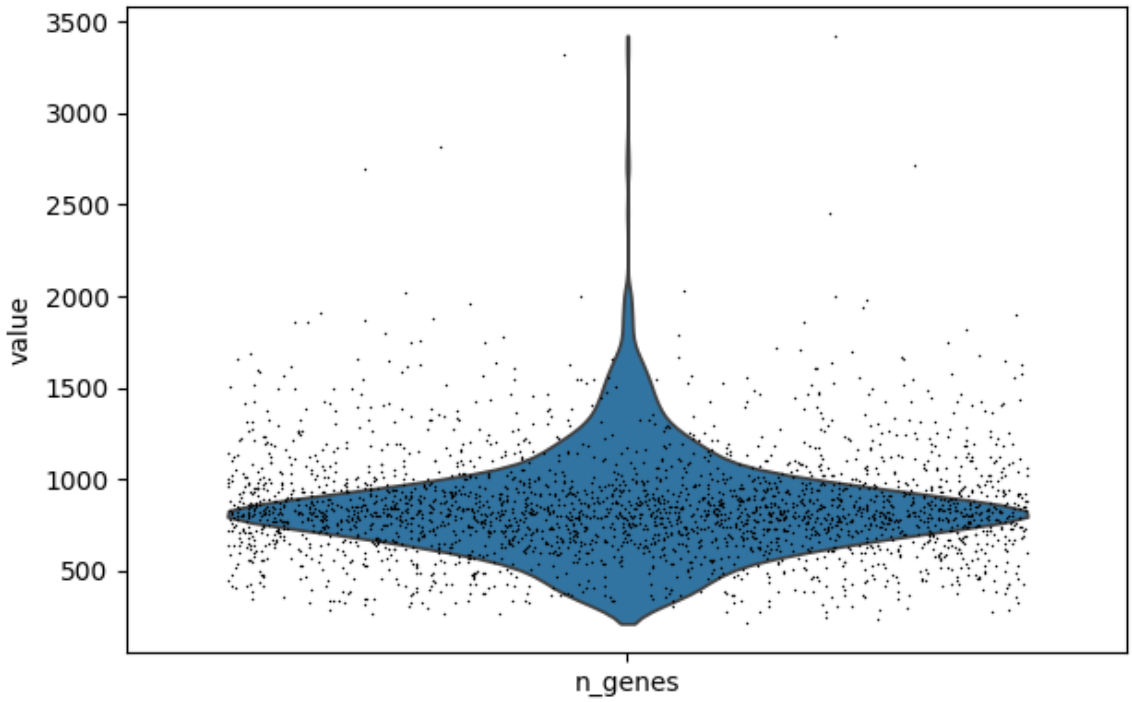
```
sc.pl.violin(pbmc3k, keys = 'n_counts',jitter=0.4) #make a violin plot of counts
```





In [14]:

```
sc.pl.violin(pbmc3k, keys = 'n_genes',jitter=0.4) #violin plot of genes
```



Now we want to filter the data to get rid of poor reads.

To start, cells with high expression of mitochondrial should be filtered out.

First, we need to make a list of genes that are mitochondrial.

In [15]:

```
mito_genes = pbmc3k.var_names.str.startswith('MT-') #create a list of genes that are mito
chondrial
```

In [16]:

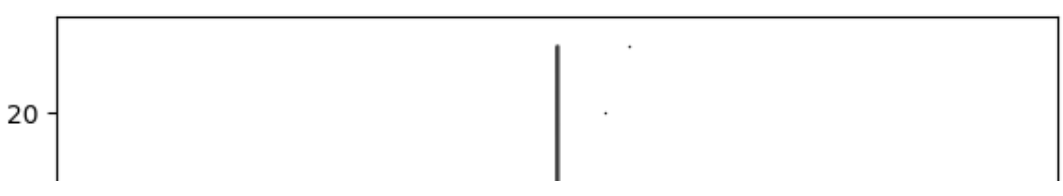
```
pbmc3k.var["mito"] = pbmc3k.var_names.str.startswith("MT-") #create a variable saying if
genes are mitochondrial
```

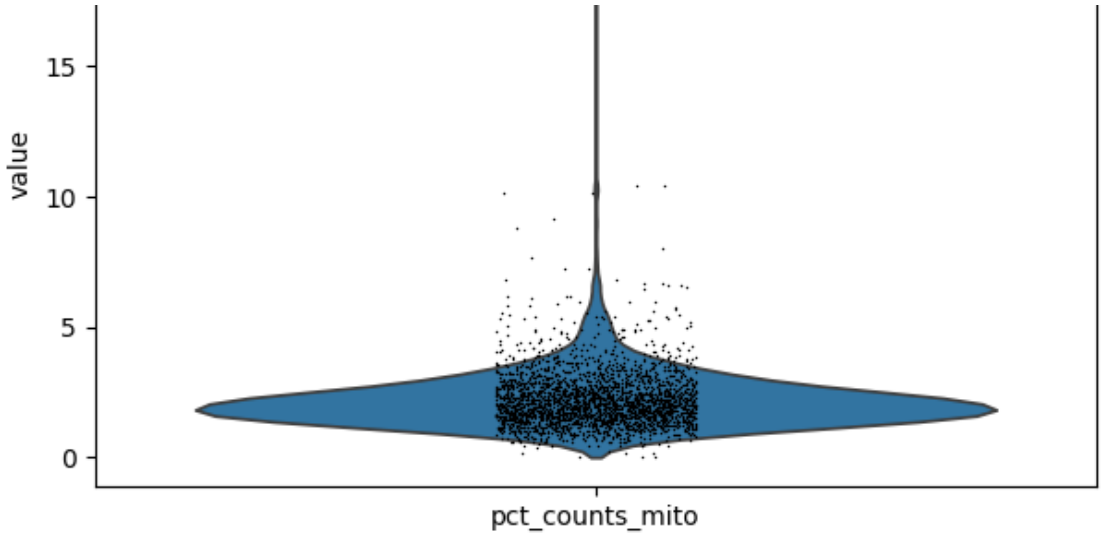
In [17]:

```
sc.pp.calculate_qc_metrics(pbmc3k, qc_vars=["mito"], inplace=True)
```

In [18]:

```
sc.pl.violin(pbmc3k, 'pct_counts_mito') #see what the mito percent looks like
```





In [19]:

```
pbmc3k = pbmc3k[pbmc3k.obs.pct_counts_mito < 5, :]
```

In [20]:

```
pbmc3k.shape
```

Out[20]:

```
(2643, 32738)
```

Next we'll filter out cells with exceptionally low or high gene counts, and genes that are expressed in < 5 cells.

In [21]:

```
sc.pp.filter_cells(pbmc3k, min_genes = 100) #filter out cells with < 100 genes
sc.pp.filter_cells(pbmc3k, max_genes = 3000) #filter out possible doublets
sc.pp.filter_genes(pbmc3k, min_cells=5) #filter out genes with expressed in< 5 cells
```

```
/Users/rebeccakirby/miniconda3/envs/python3.9_env/lib/python3.9/site-packages/scanpy/prep
rocessing/_simple.py:139: ImplicitModificationWarning: Trying to modify attribute `.obs`
of view, initializing view as actual.
  adata.obs['n_genes'] = number
```

In [22]:

```
pbmc3k.shape
```

Out[22]:

```
(2641, 12535)
```

Next we want to normalize the data to account for cell-to-cell differences such as read depth. This will allow us to do accurate differential expression analysis.

In [24]:

```
pbmc3k.layers["counts"] = pbmc3k.X.copy() #make a copy before normalizing
```

In [25]:

```
sc.pp.normalize_per_cell(pbmc3k) #normalize for total counts per cell
sc.pp.log1p(pbmc3k) # log transform
```

We will also filter for highly variable genes, as these are likely to be biologically relevant.

In [26]:

```
sc.pp.highly_variable_genes(pbmc3k, min_mean=0.01, max_mean=5, min_disp=0.2) #make a var
iable that says if the gene is highly variable
```

In [27]:

```
pbmc3k.var.highly_variable.sum() #see how many highly variable genes there are
```

Out[27]:

```
2132
```

Now we want to save this file to be used for integration.

In [28]:

```
pbmc3k.write("data/pbmc3k_preprocessed.h5ad") #write the file as a h5ad to working direct
ory
```

Read in PMBC5k data

- **This is CITE-seq data from 5k peripheral blood mononuclear cells (PBMCs) from a healthy donor.**

  CITE-seq measures both surface protein and RNA expression in single cells. In this dataset, 32 proteins were quantified.

In [ ]:

```
%%bash
wget http://cf.10xgenomics.com/samples/cell-exp/3.1.0/5k_pbmc_protein_v3_nextgem/5k_pbmc_
protein_v3_nextgem_filtered_feature_bc_matrix.h5 -O data/5k_pbmc_protein_v3_nextgem_filte
red_feature_bc_matrix.h5
```

In [38]:

```
pbmc5k = sc.read_10x_h5("data/5k_pbmc_protein_v3_nextgem_filtered_feature_bc_matrix.h5",
gex_only=False)
pbmc5k.var_names_make_unique() #make variable names unique
pbmc5k.layers["counts"] = pbmc5k.X.copy() # make a copy like we did for pbmck3
pbmc5k
```

```
/Users/rebeccakirby/miniconda3/envs/python3.9_env/lib/python3.9/site-packages/anndata/_co
re/anndata.py:1820: UserWarning: Variable names are not unique. To make them unique, call
`.var_names_make_unique`.
  utils.warn_names_duplicates("var")
```

Out[38]:

```
AnnData object with n_obs × n_vars = 5527 × 33570
    var: 'gene_ids', 'feature_types', 'genome'
    layers: 'counts'
```

In [39]:

```
#filter for number of genes
sc.pp.filter_genes(pbmc5k, min_counts=1)
```

In [40]:

```
pbmc5k.var["feature_types"].value_counts() #see what the observations are
```

Out[40]:

```
feature_types
Gene Expression     21421
Antibody Capture       32
Name: count, dtype: int64
```

Now we are going to make 2 separate objects, one with RNA data and one with protein data.

In [41]:

```
pbmc5k_protein = pbmc5k[:, pbmc5k.var["feature_types"] == "Antibody Capture"].copy()
pbmc5k_rna = pbmc5k[:, pbmc5k.var["feature_types"] == "Gene Expression"].copy()
```

We need to filter the rna and protein data, starting with rna.

In [42]:

```
pbmc5k_rna
```

Out[42]:

```
AnnData object with n_obs × n_vars = 5527 × 21421
    var: 'gene_ids', 'feature_types', 'genome', 'n_counts'
    layers: 'counts'
```

In [43]:

```
sc.pp.filter_genes(pbmc5k_rna, min_counts=5) #filter out genes that are present in less than 5 cells
```

In [44]:

```
pbmc5k_rna.var["mito"] = pbmc5k_rna.var_names.str.startswith("MT-")
sc.pp.calculate_qc_metrics(pbmc5k_rna, qc_vars=["mito"], inplace=True)
```

In [45]:

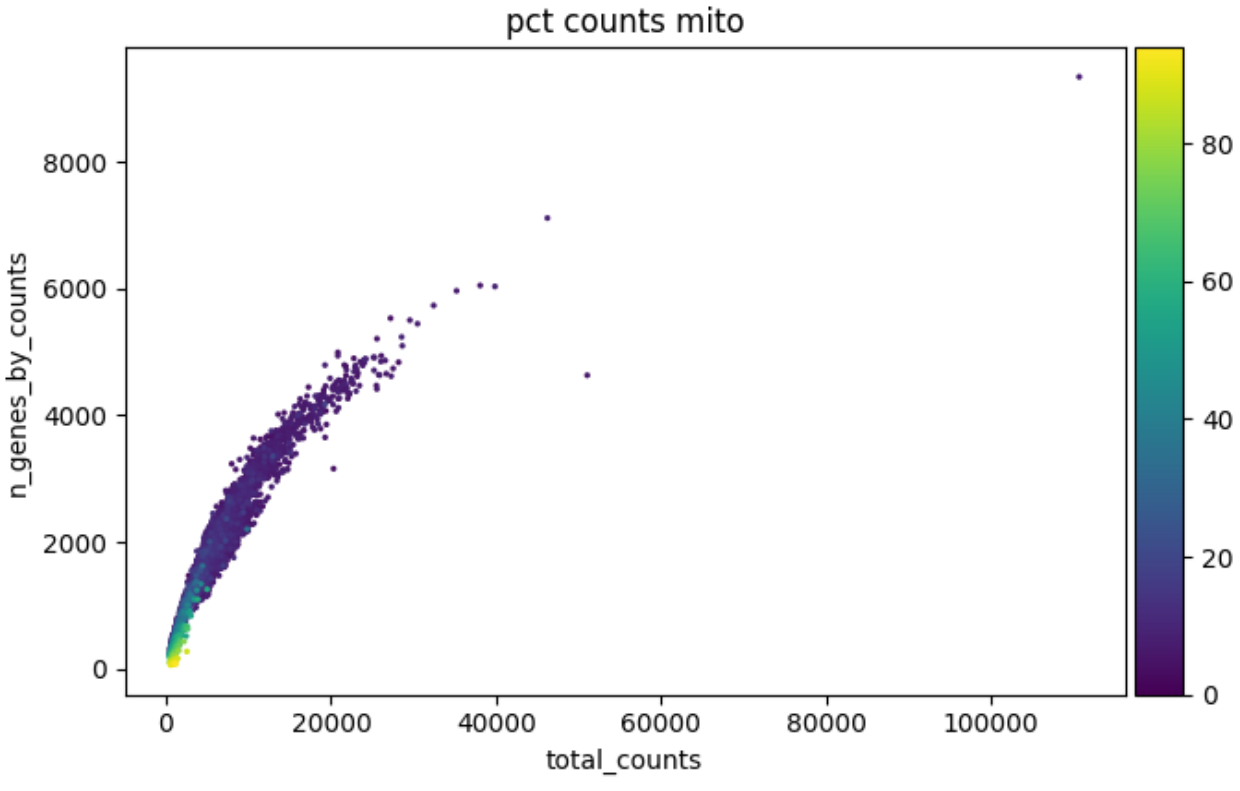
```
pbmc5k_rna
```

Out[45]:

```
AnnData object with n_obs × n_vars = 5527 × 17097
    obs: 'n_genes_by_counts', 'log1p_n_genes_by_counts', 'total_counts', 'log1p_total_counts', 'pct_counts_in_top_50_genes', 'pct_counts_in_top_100_genes', 'pct_counts_in_top_200_genes', 'pct_counts_in_top_500_genes', 'total_counts_mito', 'log1p_total_counts_mito', 'pct_counts_mito'
    var: 'gene_ids', 'feature_types', 'genome', 'n_counts', 'mito', 'n_cells_by_counts', 'mean_counts', 'log1p_mean_counts', 'pct_dropout_by_counts', 'total_counts', 'log1p_total_counts'
    layers: 'counts'
```

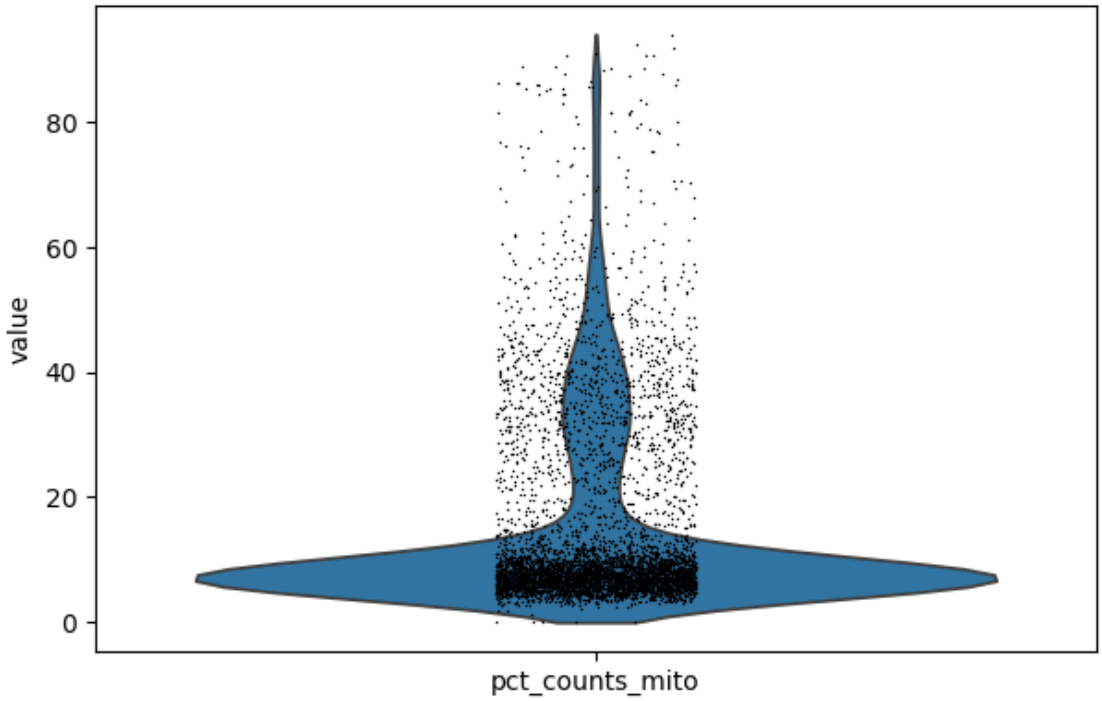
In [46]:

```
sc.pl.scatter(pbmc5k_rna, "total_counts", "n_genes_by_counts", color="pct_counts_mito")
```



In [47]:

```
sc.pl.violin(pbmc5k_rna, 'pct_counts_mito') #see what the mito percent looks like
```



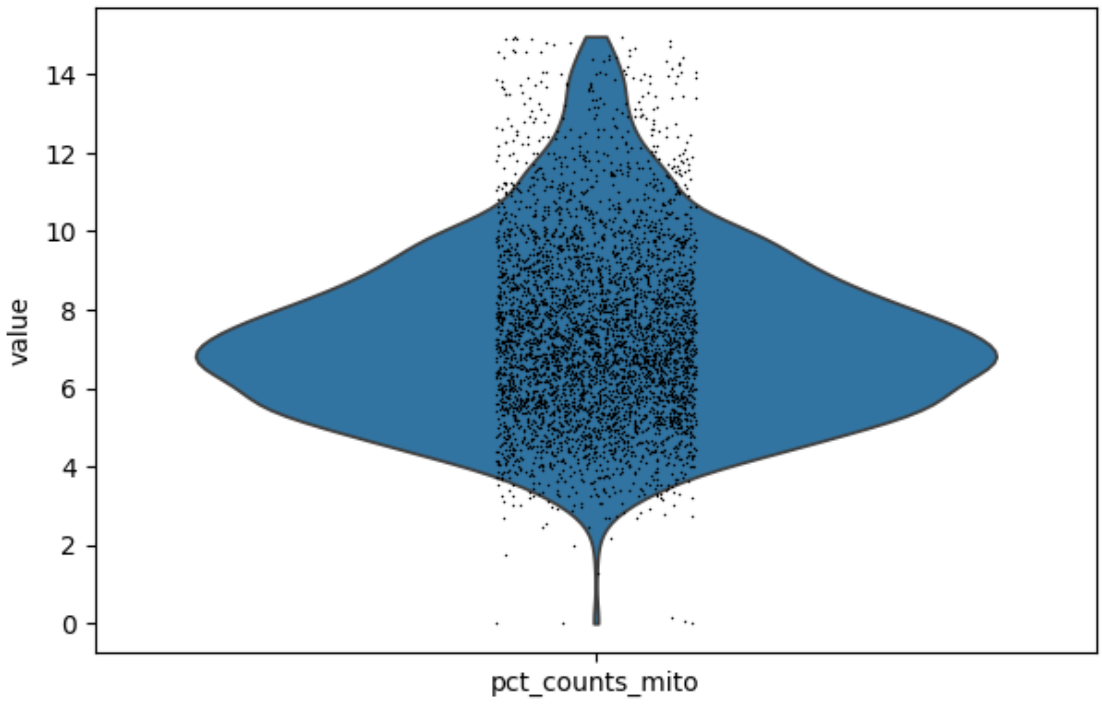
The bulk look to be under 15%, so I will set the filter to that.

In [48]:

```
pbmc5k_rna1 = pbmc5k_rna[pbmc5k_rna.obs.pct_counts_mito < 15, :].copy() #filter out cell
s where mitochondrial genes are > 20
```

In [50]:

```
sc.pl.violin(pbmc5k_rna1, 'pct_counts_mito') #see what the mito percent looks like now
```



We will normalize as we did with pbmc3k.

In [51]:

```
sc.pp.normalize_per_cell(pbmc5k_rna) #normalize for total counts per cell
```

```
sc.pp.log1p(pbmc5k_rna) # log transform
```

Preprocess the protein object

We normalize the protein object a bit differently than the RNA object. We use sc.pp.normalize_geometric.

In [52]:

```
pbmc5k_protein.var["control"] = pbmc5k_protein.var_names.str.contains("control")
sc.pp.calculate_qc_metrics(
    pbmc5k_protein,
    percent_top=(5, 10, 15),
    var_type="antibodies",
    qc_vars=("control",),
    inplace=True,
)
```

In [53]:

```
pbmc5k_protein.layers["counts"] = pbmc5k_protein.X.copy() #make a copy before normalizat
ion
```

In [54]:

```
def geometric_transform(df):
    '''
    implements the scanpy transform originating from ivirshup:multimodal
    '''
    from scipy.stats.mstats import gmean
    T_geometric = np.divide(df, gmean(df + 1, axis=0))
    return T_geometric

pbmc5k_protein.X = geometric_transform(pbmc5k_protein.X.A)
```

In [55]:

```
sc.pp.log1p(pbmc5k_protein)
```

In [56]:

```
pbmc5k_rna1.write("data/pbmc5k_rna_preprocessed.h5ad") #write the file as a h5ad to the d
ata directory
pbmc5k_protein.write("data/pbmc5k_protein_preprocessed.h5ad")
```

Aim 2: Integration of CITE-seq and RNA-seq data.

The code for this aim is not run by students in the in-class section of the tutorial, however, we verbally walk through it step-by-step during class since it is an important part of the pipeline. The goal of this section is to use the RNA expression in both datasets to integrate both modalities into a shared latent space. This is important for our third aim, where we use neighbors in the shared latent space to impute protein expression into scRNA-seq data (a modality that does not measure protein expression).

Biological Interpetation:

Many studies in that take large numbers of measurments from single cells make use of computational integration between similar experiments performed in separate studies. Sometimes this is done because one dataset contains measurements that the other dataset does not have. In our case, we are integrating CITE-seq and scRNA-seq data to impute protein expression into the scRNA-seq data. Protein expression is not quantified in scRNA-seq data, so by integrating the two datasets on their shared feature set (RNA expression), we can later predict protein expression in the scRNA-seq data. While this tutorial is not a novel biological study, we wanted to show how to perform the integration of multimodal data, since there are several cases where it might give biological insight. For instance, (1) imputing scRNA-seq pseudotime into spatial transcriptomics datasets where less genes are measured and pseudotime is hard to calculate reliably. (2) Transferring cell type marker labels from one dataset to another without manual intervention.

Mathematical/statistical meaning of analysis

The integration is performed with scVI, which makes use of a Variational Autoencoder (VAE) to project cells with similar RNA expression into an organized and shared latent space. A covariate defined by the observation 'batch' is used to regress out batch effects between datasets, while still trying to preserve the underlying biological differences.

Common pitfalls

scVI works best when integrating datasets based on their RNA expression taken from the same measurement platform. Different platforms have varying capture efficiency rates, and should not be modeled under the same distribution. For instance, integrating scRNA-seq and spatial data will usually not work well with scVI, as they have different capture efficiencies.

Performing joint integration of the scRNA and CITE-seq data on the RNA modality.

In [65]:

```
#!pip install scvi-tools
import scvi
```

In [66]:

```
cite_rna = sc.read_h5ad("data/pbmc5k_rna_preprocessed.h5ad")
cite_protein = sc.read_h5ad("data/pbmc5k_protein_preprocessed.h5ad")

sc_rna = sc.read_h5ad("data/pbmc3k_preprocessed.h5ad")
```

In [ ]:

```
cite_protein = cite_protein[cite_rna.obs.index, :] #align the protein and rna data
```

In [ ]:

```
cite_rna.obsm["protein"] = cite_protein.to_df()
```

In [ ]:

```
sc_rna.obs['batch'] = 'scrna'
cite_rna.obs['batch'] = 'cite'
```

In [ ]:

```
combined_adata = sc.concat([cite_rna, sc_rna], merge='first')
```

In [ ]:

```
scvi.model.SCVI.setup_anndata(combined_adata, layer="counts", batch_key="batch")
vae = scvi.model.SCVI(combined_adata, n_layers=2, n_latent=20, gene_likelihood="nb")
vae.train()
```

In [ ]:

```
combined_adata.obsm["X_scVI"] = vae.get_latent_representation()
sc.pp.neighbors(combined_adata, use_rep="X_scVI")
```

In [ ]:

```
sc.tl.umap(combined_adata)
```

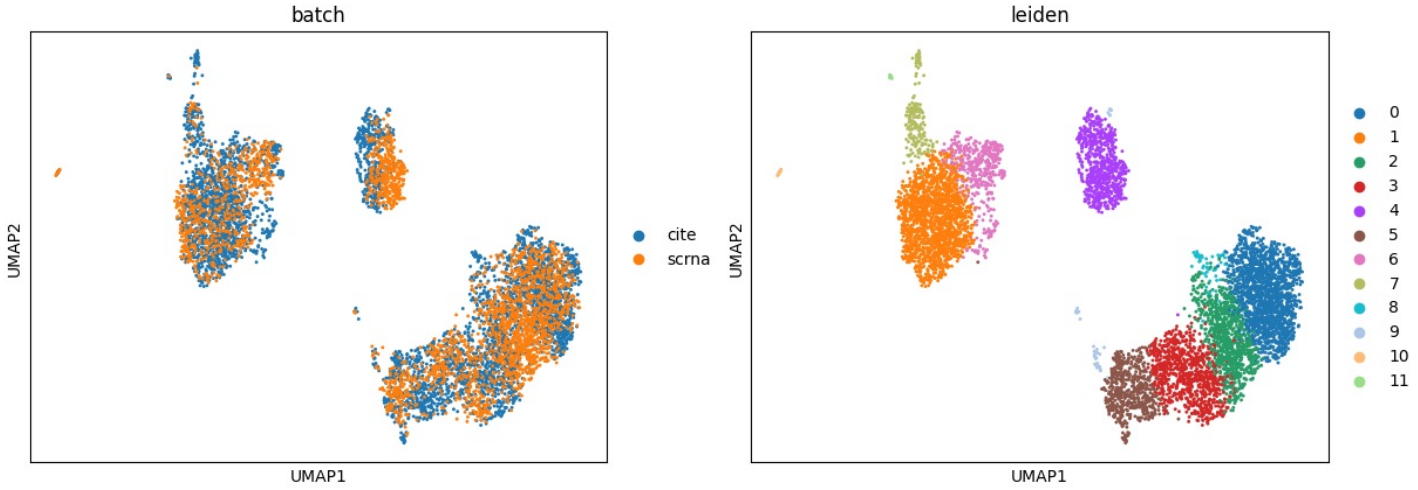
In [ ]:

```
sc.tl.leiden(combined_adata)
```

In [ ]:

```
sc.pl.umap(combined_adata, color=["batch", "leiden"])
```

```
/home/amonell/mambaforge/envs/scvi-env_gpu/lib/python3.9/site-packages/scanpy/plotting/_tools/scatterplots.py:1251: FutureWarning: The default value of 'ignore' for the `na_action` parameter in pandas.Categorical.map is deprecated and will be changed to 'None' in a future version. Please set na_action to the desired value to avoid seeing this warning
  color_vector = pd.Categorical(values.map(color_map))
/home/amonell/mambaforge/envs/scvi-env_gpu/lib/python3.9/site-packages/scanpy/plotting/_tools/scatterplots.py:394: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  cax = scatter(
/home/amonell/mambaforge/envs/scvi-env_gpu/lib/python3.9/site-packages/scanpy/plotting/_tools/scatterplots.py:1251: FutureWarning: The default value of 'ignore' for the `na_action` parameter in pandas.Categorical.map is deprecated and will be changed to 'None' in a future version. Please set na_action to the desired value to avoid seeing this warning
  color_vector = pd.Categorical(values.map(color_map))
/home/amonell/mambaforge/envs/scvi-env_gpu/lib/python3.9/site-packages/scanpy/plotting/_tools/scatterplots.py:394: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  cax = scatter(
```



Using the integrated datasets, annotate cell types using UCell

In [ ]:

```
# Calculate and add signatures
signatures = {
    "CD4 cell": ['CD3E', 'CD4', 'CD8A-', 'CD19-', 'IL7R'],
    "CD8 cell": ['CD3E', 'CD4-', 'CD8A+', 'CD19-', 'IL7R-', 'GZMB', 'NCAM1-', 'FCGR3A-'],
    "B cells": ['CD19', 'CD3E-', 'CD37', 'CD8A-', 'CD4-', 'ITGAX-'],
    "NK cells": ['NKG7', 'EOMES', 'NCAM1', 'CD3E-'],
    "Dendritic cells": ['ITGAX', 'XCR1', 'CD11B', 'CD11C'],
    "Monocytes": ['CD14', 'FCGR3A', 'NKG7-']
}

ucell.add_scores(combined_adata, signatures=signatures, maxRank=1000)
```

```
/home/amonell/mambaforge/envs/scvi-env_gpu/lib/python3.9/site-packages/ucell/ucell.py:39: UserWarning: Some genes were not found in signature: Dendritic cells: ['CD11C', 'CD11B', 'XCR1']. Missing genes will be removed from the list.
  signatures = _check_signatures(signatures=signatures, indices=m.columns)
```

In [ ]:

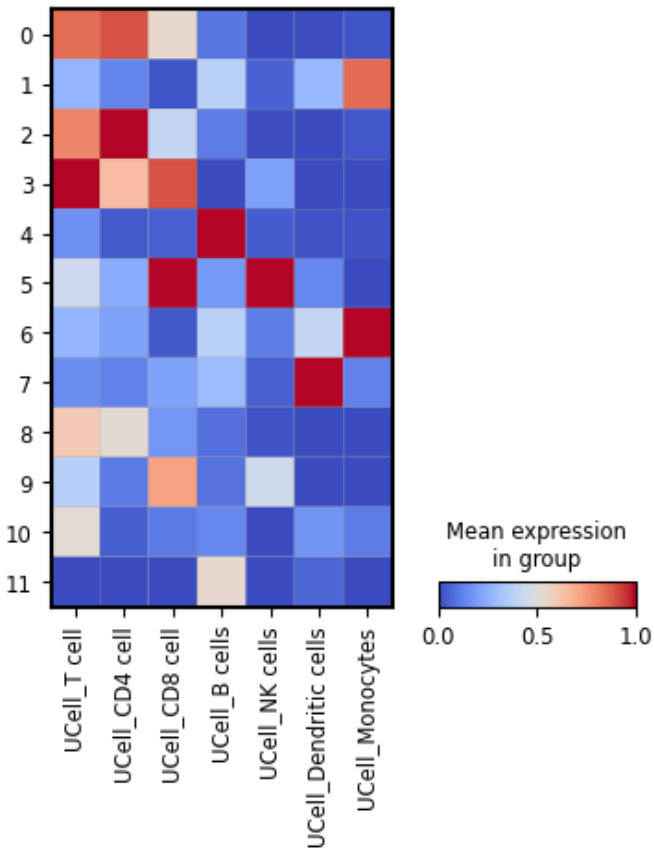
```
ucell_features = [i for i in combined_adata.obs.columns if "UCell" in i]
```

In [ ]:

```
sc.pl.matrixplot(combined_adata, groupby='leiden', var_names=ucell_features, standard_scale='var', cmap='coolwarm')
```

```
/home/amonell/mambaforge/envs/scvi-env_gpu/lib/python3.9/site-packages/scanpy/plotting/_matrixplot.py:143: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current beha
```

```
changed to True in a future version of pandas. Pass observed=False to retain current beha
vior or observed=True to adopt the future default and silence this warning.
  values_df = self.obs_tidy.groupby(level=0).mean()
```



In [ ]:

```
cell_type_annotations = {'0': 'CD4 T cell', '1': 'Monocyte', '2':'CD4 T cell', '3':'CD8
T cell', '4': 'B cell', '5': 'NK cell', '6': 'Monocyte', '7': 'DC', '8': 'Unknown', '9':
'Unknown', '10': 'Unknown', '11': 'Unknown'}
```

In [ ]:

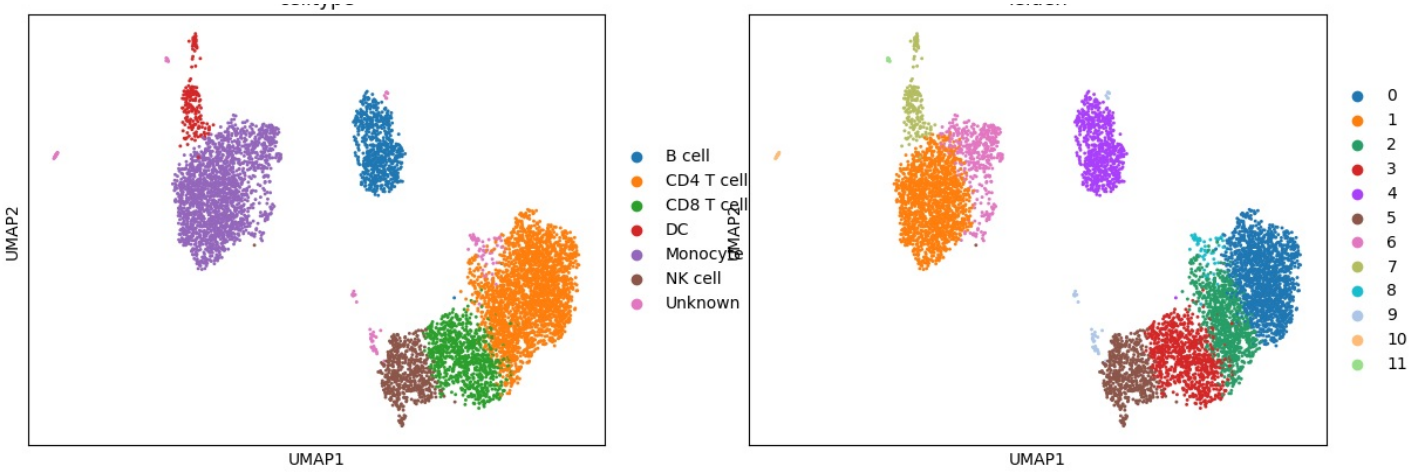
```
combined_adata.obs['celltype'] = [cell_type_annotations.get(i) for i in combined_adata.o
bs['leiden']]
```

In [ ]:

```
sc.pl.umap(combined_adata, color=["celltype", "leiden"])
```

```
/home/amonell/mambaforge/envs/scvi-env_gpu/lib/python3.9/site-packages/scanpy/plotting/_t
ools/scatterplots.py:1251: FutureWarning: The default value of 'ignore' for the `na_actio
n` parameter in pandas.Categorical.map is deprecated and will be changed to 'None' in a f
uture version. Please set na_action to the desired value to avoid seeing this warning
  color_vector = pd.Categorical(values.map(color_map))
/home/amonell/mambaforge/envs/scvi-env_gpu/lib/python3.9/site-packages/scanpy/plotting/_t
ools/scatterplots.py:394: UserWarning: No data for colormapping provided via 'c'. Paramet
ers 'cmap' will be ignored
  cax = scatter(
/home/amonell/mambaforge/envs/scvi-env_gpu/lib/python3.9/site-packages/scanpy/plotting/_t
ools/scatterplots.py:1251: FutureWarning: The default value of 'ignore' for the `na_actio
n` parameter in pandas.Categorical.map is deprecated and will be changed to 'None' in a f
uture version. Please set na_action to the desired value to avoid seeing this warning
  color_vector = pd.Categorical(values.map(color_map))
/home/amonell/mambaforge/envs/scvi-env_gpu/lib/python3.9/site-packages/scanpy/plotting/_t
ools/scatterplots.py:394: UserWarning: No data for colormapping provided via 'c'. Paramet
ers 'cmap' will be ignored
  cax = scatter(
```

celltype leiden



In [ ]:

```
# store integrated objects
cite_rna.obsm["X_scVI"] = combined_adata[cite_rna.obs.index].obsm["X_scVI"]
sc_rna.obsm["X_scVI"] = combined_adata[sc_rna.obs.index].obsm["X_scVI"]

cite_rna.obsm["X_umap"] = combined_adata[cite_rna.obs.index].obsm["X_umap"]
sc_rna.obsm["X_umap"] = combined_adata[sc_rna.obs.index].obsm["X_umap"]

cite_rna.obs["leiden"] = combined_adata[cite_rna.obs.index].obs["leiden"]
sc_rna.obs["leiden"] = combined_adata[sc_rna.obs.index].obs["leiden"]

cite_rna.obs["celltype"] = combined_adata[cite_rna.obs.index].obs["celltype"]
sc_rna.obs["celltype"] = combined_adata[sc_rna.obs.index].obs["celltype"]

cite_rna.write("data/cite_seq_integrated.h5ad")
sc_rna.write("data/scrna_integrated.h5ad")
```

In-class tutorial starts here!

Make sure packages are installed and imported correctly

If you didn't import the packages earlier in the code, import them now

In [1]:

```
import os
import scanpy as sc
import numpy as np
import pandas as pd
from scipy.spatial import KDTree
import seaborn as sns
import matplotlib.pyplot as plt
import ucell
import mygene
```

In [2]:

```
# run this block to check that your installation is successful
import pkg_resources

required_packages = {
    'numpy': '>=1.25.0',
    'pandas': '>=1.5.3',
    'scanpy': '>=1.9.3',
    'scrublet': '>=0.2.3',
    'scvi-tools': '>=1.1.2',
    'matplotlib': '>=3.7.1',
    'ucell': '>=0.1',
    'seaborn': '>=0.13.0',
    'mygene': '>=3.2.2',
}
```

```
for package, version_range in required_packages.items():
    try:
        installed_version = pkg_resources.get_distribution(package).version
        version_requirement = pkg_resources.Requirement.parse(f"{package}{version_range}
")
        assert pkg_resources.parse_version(installed_version) in version_requirement, \
            f"{package} version must be within {version_range}, found version {installed
_version}"
    except pkg_resources.DistributionNotFound:
        assert False, f"{package} is not installed."

print("All required packages are installed and within the specified version ranges.")
```

```
All required packages are installed and within the specified version ranges.
```

```
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1040047929.py:2: Depreca
tionWarning: pkg_resources is deprecated as an API. See https://setuptools.pypa.io/en/lat
est/pkg_resources.html
  import pkg_resources
```

Aim 3: Imputing protein quantification values into scRNA-seq data.

The aim of imputing protein quantification values into scRNA-seq data is motivated by the inherent limitations associated with each dataset when analyzed in isolation: scRNA-seq provides a deep, yet indirect, measurement of cellular function through mRNA levels, while CITE-seq offers direct measurement of surface protein abundances but on a much smaller scale. Taking advantage of the integration, we could locate the expression neighbors of scRNA-seq in the joint latent space, and take the average expression of the protein expression of these neighbors as an imputation for the protein of interest.

Biological Interpetation:

We discussed some of the biological interpretation for this section as part of Aim 2. One other point that is worth mentioning is integration followed by imputation can help to tie together information for visualization in publication. For instance, in a study utilizing scRNA-seq to look at a disease setting, imputing motif enrichment, protein expression, or psuedotime may indicate different cell states resulting from disease effects.

Mathematical/statistical meaning of analysis

The imputation of protein expression values into scRNA-seq data is a form of K-nearest neighbors (KNN) imputation. The KNN algorithm is a non-parametric method used for classification and regression. In this case, we are using KNN to impute protein expression values into scRNA-seq data based on the shared latent space representation from the VAE.

Common pitfalls

The KNN algorithm is sensitive to the choice of K, the number of neighbors to consider. A small K may result in overfitting, while a large K may result in underfitting. The choice of K is often determined by cross-validation.

Use KNN to impute protein expression values

In [3]:

```
## read in the integrated scanpy objects that we have pre calculated
cite_rna = sc.read_h5ad('data/cite_seq_integrated.h5ad')
sc_rna = sc.read_h5ad('data/scrna_integrated.h5ad')
```

In [4]:

```
# Ensure the joint latent space is present and aligned
assert 'X_scVI' in cite_rna.obsm.keys(), "X_scvi not found in CITE-seq object"
assert 'X_scVI' in sc_rna.obsm.keys(), "X_scvi not found in scRNA-seq object"
```

In [5]:

```
#find the nearest cite seq neighbor cells of each scRNA cell using the 'X scVI' latent sp
```

```
ace
kdtree = KDTree(cite_rna.obsm['X_scVI'])
distances, indices = kdtree.query(sc_rna.obsm['X_scVI'], k=30)
```

In [6]:

```
# Initialize a place to store imputed values for the scRNA-seq dataset
num_proteins = cite_rna.obsm['protein'].shape[1]
imputed_proteins = np.zeros((sc_rna.shape[0], num_proteins)) # Assuming protein expressi
on is stored here
```

In [7]:

```
# Impute protein expression for each scRNA-seq cell based on CITE-seq neighbors
for cell_idx in range(sc_rna.shape[0]):  # Iterate through scRNA-seq cells
    imputed_proteins[cell_idx] = np.mean(cite_rna[indices[cell_idx]].obsm['protein'], ax
is=0)
```

In [8]:

```
# Add the imputed protein expression to the scRNA-seq AnnData object
sc_rna.obsm['imputed_protein_expression'] = imputed_proteins
```

plot imputed protein level on scRNA

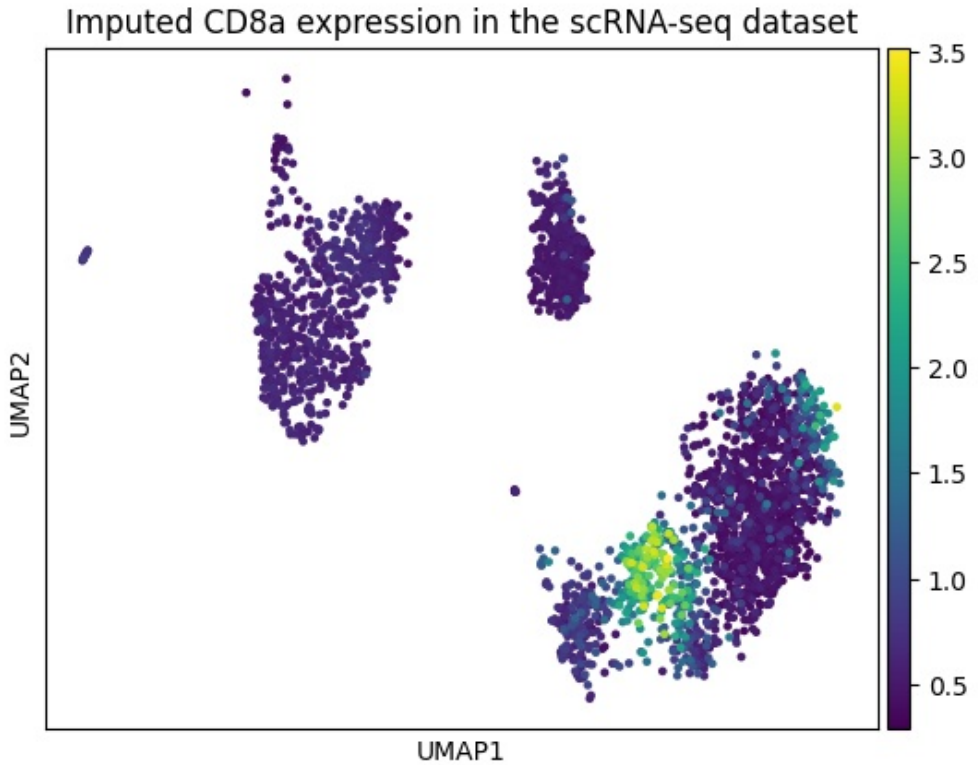
In [9]:

```
# add each protein to the scrna adata
protein_names = list(cite_rna.obsm['protein'].columns)
for i, protein in enumerate(protein_names):
    # Extract the protein expression values from .obsm and add to .obs for plotting
    sc_rna.obs[protein] = sc_rna.obsm['imputed_protein_expression'][:, i]
```

In [10]:

```
# plot imputed CD8a protein expression as an example using sc.pl.umap
protein = 'CD8a_TotalSeqB'
sc.pl.umap(sc_rna, color=protein,
           title=f"Imputed {protein.split('_')[0]} expression in the scRNA-seq dataset")
```

```
/Users/rebeccakirby/miniconda3/envs/python3.9_env/lib/python3.9/site-packages/scanpy/plot
ting/_tools/scatterplots.py:1208: FutureWarning: is_categorical_dtype is deprecated and w
ill be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if not is_categorical_dtype(values):
```



In [114]:

```
# save the result
sc_rna.write("data/scrna_integrated_imputed.h5ad")
```

Aim 4: Correlating RNA and Protein expression in CITE-seq data.

In this aim, our goal is to show that the coexpression between RNA and protein of the same gene can vary. While some genes have highly correlated protein and RNA expression, others do not, indicating post-transcriptional activity and variability in the rate of translation. Observing which genes have a large disconnect between their measured protein and RNA expression can help us better evaluate the results of scRNA-seq ligand-receptor interaction prediction tools that make their predictions solely off of RNA expression, rather than surface proteins which are true interacting molecules.

Biological Interpetation:

The coexpression between RNA and protein can be used to infer the rate of translation, and the stability of the protein. For instance, if a gene has high RNA expression but low protein expression, it may be due to post-transcriptional regulation, such as microRNA binding or protein degradation.

Mathematical/statistical meaning of analysis

The coexpression between RNA and protein is calculated using the Pearson correlation coefficient. The Pearson correlation coefficient measures the linear relationship between two variables. A value of 1 indicates a perfect positive linear relationship, -1 indicates a perfect negative linear relationship, and 0 indicates no linear relationship.

Common pitfalls

The Pearson correlation coefficient is sensitive to outliers. If the data contains outliers, the correlation coefficient may be skewed. It is important to check for outliers before calculating the correlation coefficient.

Plotting coexpression between RNA and protein

In [11]:

```
sc_rna = sc.read("data/scrna_integrated_imputed.h5ad")
cite_rna = sc.read("data/cite_seq_integrated.h5ad")
```

```
/Users/rebeccakirby/miniconda3/envs/python3.9_env/lib/python3.9/site-packages/anndata/__i
nit__.py:55: FutureWarning: `anndata.read` is deprecated, use `anndata.read_h5ad` instead
. `ad.read` will be removed in mid 2024.
  warnings.warn(
/Users/rebeccakirby/miniconda3/envs/python3.9_env/lib/python3.9/site-packages/anndata/__i
nit__.py:55: FutureWarning: `anndata.read` is deprecated, use `anndata.read_h5ad` instead
. `ad.read` will be removed in mid 2024.
  warnings.warn(
```

In [12]:

```
# string processing for the protein name
protein_names = [i.split('_')[0].upper() for i in cite_rna.obsm['protein'].columns]
```

In [13]:

```
import mygene

# Initialize a MyGeneInfo object
mg = mygene.MyGeneInfo()

# List of gene symbols or IDs for which you want to retrieve aliases
genes = protein_names

gene_aliases = []
```

```
# Querying MyGene.info for aliases
for g in genes:
    query_list = mg.query(g)['hits']
    try:
        symbols = [j['symbol'].upper() for j in query_list]
        symbols_in = [sym for sym in symbols if sym in cite_rna.var.index.values]
        gene_aliases.append(symbols_in)
    except:
        gene_aliases.append([])
```

In [14]:

```
overlapping_names = [i[0] if len(i) > 0 else '' for i in gene_aliases]
```

Specify which gene to plot protein vs rna

In [15]:

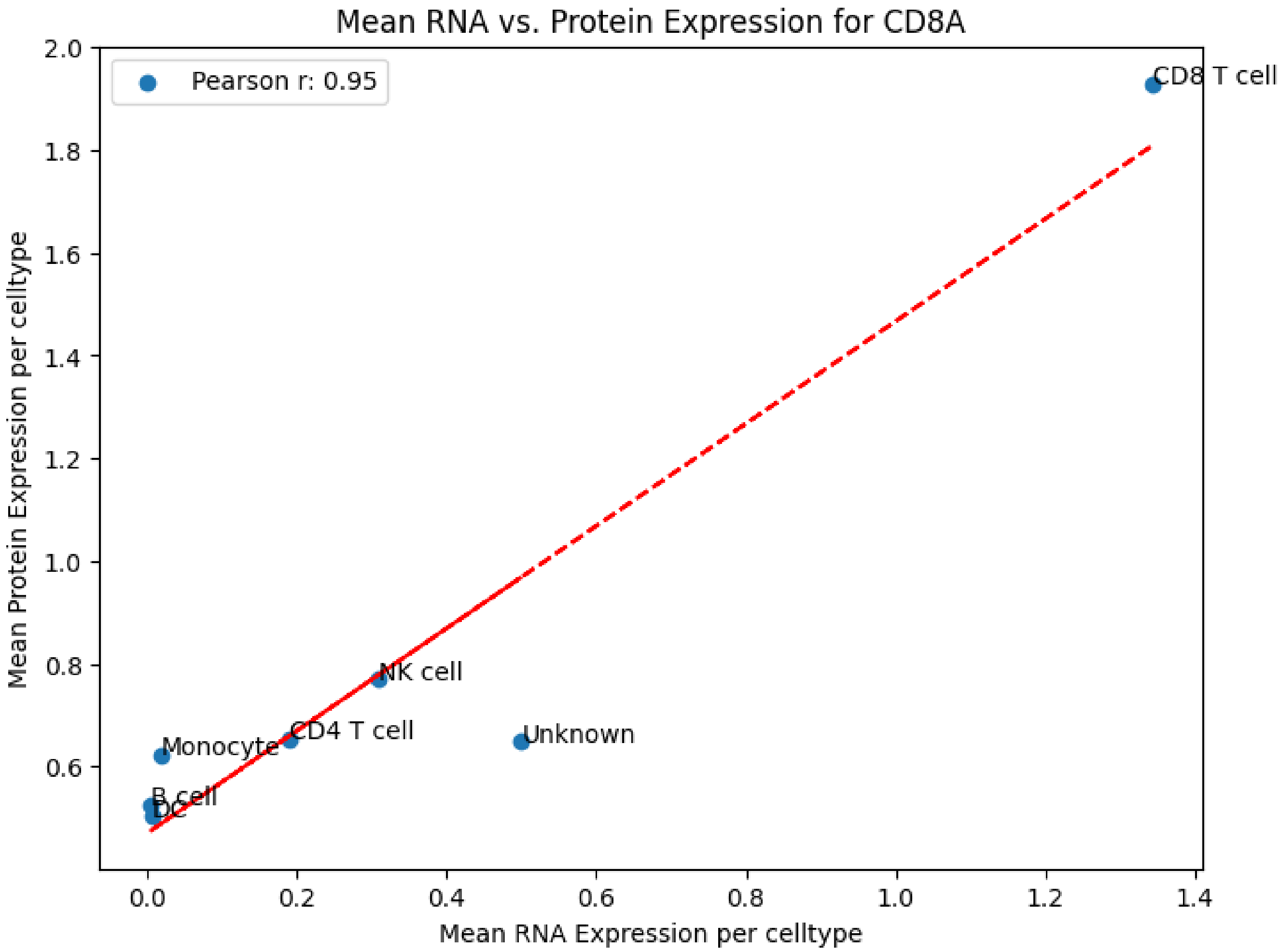
```
gene_to_plot = overlapping_names[2]
```

In [16]:

```
correlation_coefficients = []
gene_names = []

for i, e in enumerate(overlapping_names):
    if e != '':
        # Extract RNA and protein expressions
        rna_expression = cite_rna[:, cite_rna.var.index == e].X.toarray().flatten()
        protein_expression = cite_rna.obsm['protein'][cite_rna.obsm['protein'].columns[i
]].values

        # Create a DataFrame with RNA, protein expressions, and cell type
        df = pd.DataFrame({'rna': rna_expression, 'protein': protein_expression, 'cellty
pe': cite_rna.obs['celltype']})

        # Group by cell type and calculate the mean
        group_means = df.groupby('celltype').mean()

        # Calculate Pearson correlation coefficient
        corr_coef = np.corrcoef(group_means['rna'], group_means['protein'])[0, 1]

        correlation_coefficients.append(corr_coef)
        gene_names.append(e)

        if e == gene_to_plot:
            # Plotting
            plt.figure(figsize=(8, 6))
            plt.scatter(group_means['rna'], group_means['protein'], label=f'Pearson r: {
corr_coef:.2f}')

            # Line of best fit
            m, b = np.polyfit(group_means['rna'], group_means['protein'], 1)
            plt.plot(group_means['rna'], m * group_means['rna'] + b, 'r--')

            # Annotate each point with cell type
            for j, txt in enumerate(group_means.index):
                plt.annotate(txt, (group_means['rna'][j], group_means['protein'][j]))

            plt.title(f"Mean RNA vs. Protein Expression for {e}")
            plt.xlabel('Mean RNA Expression per celltype')
            plt.ylabel('Mean Protein Expression per celltype')
            plt.legend()
            plt.grid(False)
            plt.show()
```

```
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
```

```
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:33: Future
Warning: Series.__getitem__ treating keys as positions is deprecated. In a future version
, integer keys will always be treated as labels (consistent with DataFrame behavior). To
access a value by position, use `ser.iloc[pos]`
  plt.annotate(txt, (group_means['rna'][j], group_means['protein'][j]))
```




```
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
```

```
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
```

```
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
/var/folders/83/d9ggj2kd17d_n5vsy452034r0000gn/T/ipykernel_39276/1423477184.py:14: Future
Warning: The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_means = df.groupby('celltype').mean()
```

Which genes have highly correlated protein and RNA expression?

In [121]:

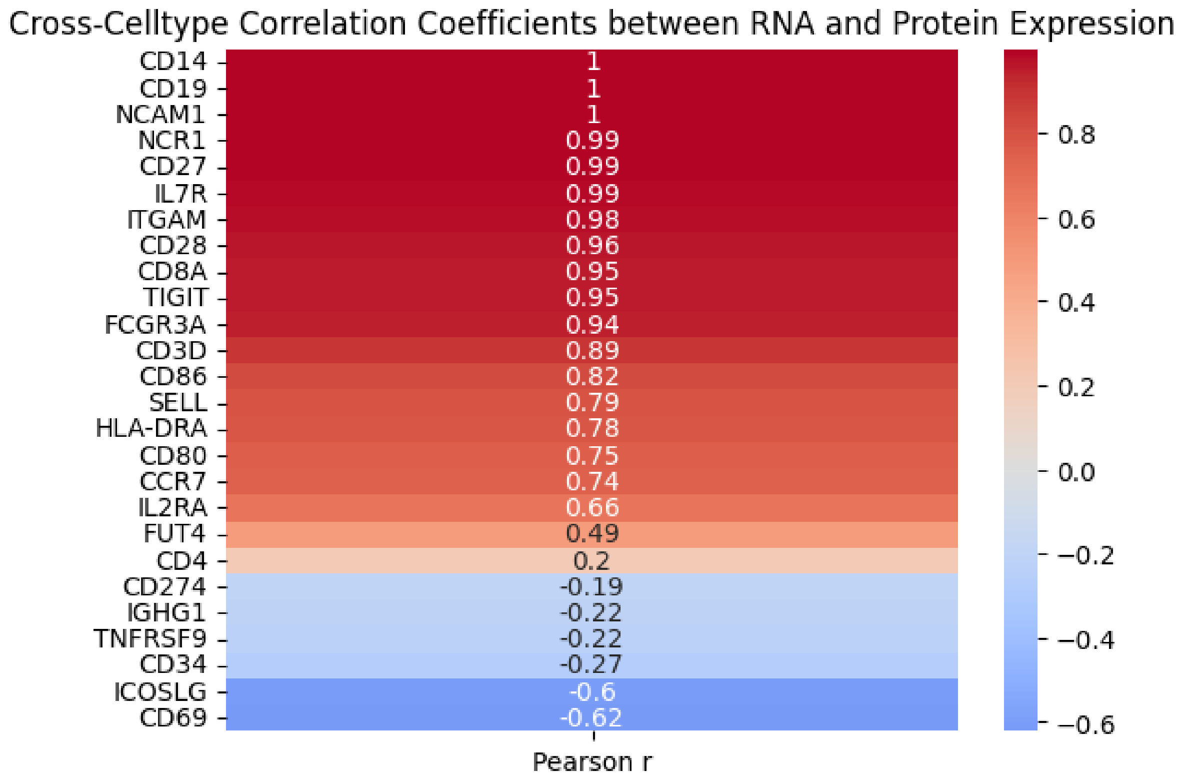
```
whole_df = pd.DataFrame(correlation_coefficients, index=gene_names, columns=['correlation']).sort_values(by='correlation', ascending=False)
```

In [122]:

```
sns.heatmap(whole_df, annot=True, yticklabels=whole_df.index, xticklabels=['Pearson r'], cmap='coolwarm', center=0)
plt.title('Cross-Celltype Correlation Coefficients between RNA and Protein Expression')
```

Out[122]:

```
Text(0.5, 1.0, 'Cross-Celltype Correlation Coefficients between RNA and Protein Expression')
```



References

References for software tools

Scanpy: Wolf, F., Angerer, P. & Theis, F. SCANPY: large-scale single-cell gene expression data analysis. Genome Biol 19, 15 (2018). https://doi.org/10.1186/s13059-017-1382-0

Pandas: McKinney, W., & others. Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference Vol. 445, 51–56 (2010). Jupyter Lab:

Numpy: Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2.

SCVI-Tools: Gayoso, A., Lopez, R., Xing, G. et al. A Python library for probabilistic analysis of single-cell omics data. Nat Biotechnol 40, 163–166 (2022). https://doi.org/10.1038/s41587-021-01206-w.

Mygene: Wu C, Macleod I, Su AI. BioGPS and MyGene.info: organizing online, gene-centric information. Nucleic Acids Res. 2013 Jan;41(Database issue):D561-5. doi: 10.1093/nar/gks1114. Epub 2012 Nov 21. PMID: 23175613; PMCID: PMC3531157.

Setuptools: https://github.com/pypa/setuptools

UCell: Accessed from https://github.com/maximilian-heeg/UCell.git.

References for data

https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.1.0/pbmc3k

https://support.10xgenomics.com/single-cell-gene-expression/datasets/3.0.2/5k_pbmc_protein_v3

References for background

[1] Maria M, Pouyanfar N, Örd T, Kaikkonen MU. The Power of Single-Cell RNA Sequencing in eQTL Discovery. Genes (Basel). 2022 Mar 12;13(3):502. doi: 10.3390/genes13030502. PMID: 35328055; PMCID: PMC8949403.

[2] Xiaotu Ma, Hyunju Lee, Li Wang, Fengzhu Sun, CGI: a new approach for prioritizing genes by combining gene expression and protein–protein interaction data, Bioinformatics, Volume 23, Issue 2, January 2007, Pages 215–221, https://doi.org/10.1093/bioinformatics/btl569