In [ ]:

```
"""Broadly useful python packages"""
import pandas as pd
import os
import numpy as np
import pickle
from copy import deepcopy
from shutil import move
import warnings

"""Machine learning and single cell packages"""
import sklearn.metrics as metrics
from sklearn.metrics import adjusted_rand_score as ari, normalized_mutual_info_s
import scanpy as sc
from anndata import AnnData

"""CarDEC Package"""
from CarDEC import CarDEC_API
```

In [ ]:

```
"""Miscellaneous useful functions"""

def read_macaque(path):
    """A function to read and preprocess the macaque data"""
    adata = sc.read(path)
    sc.pp.filter_cells(adata, min_genes=0)
    sc.pp.filter_genes(adata, min_cells=30)

    adata = adata[adata.obs['n_genes'] < 2500, :]

    return(adata)

def purity_score(y_true, y_pred):
    """A function to compute cluster purity"""
    # compute contingency matrix (also called confusion matrix)
    contingency_matrix = metrics.cluster.contingency_matrix(y_true, y_pred)

    return np.sum(np.amax(contingency_matrix, axis=0)) / np.sum(contingency_matr

def find_resolution(adata_, n_clusters, random = 0):
    adata = adata_.copy()
    obtained_clusters = -1
    iteration = 0
    resolutions = [0., 1000.]

    while obtained_clusters != n_clusters and iteration < 50:
        current_res = sum(resolutions)/2
        sc.tl.louvain(adata, resolution = current_res, random_state = random)
        labels = adata.obs['louvain']
        obtained_clusters = len(np.unique(labels))

        if obtained_clusters < n_clusters:
            resolutions[0] = current_res
        else:
            resolutions[1] = current_res

        iteration = iteration + 1

    return current_res
```

```
metrics_ = [ari, nmi, purity_score]
```

In the following cell, we read in the data.

In [ ]:
```
"""Read and normalize the data"""
adata = read_macaque("macaque_bc.h5ad")
```

Now, intialize the CarDEC class. Doing so will normalize the dataset. The results will be stored in an anndata object, referenced by CarDEC.dataset

In [ ]:
```
"""Args:
    1. adata is the dataframe to work on
    2. weights_dir: A directory in which to save weights for both the autoencode
    CarDEC model. Weights are also loaded from this directory. If the directory
    created from scratch.
    3. batch_key is the key in adata.obs that identifies the vector of cell batc
    4. n_high_var is the number of features to treat as highly variable. These f
       n_high_var highly variable genes are identified with scanpy, using within
    5. LVG: If True, then denoise low variance features too. Else, only denoise
"""

CarDEC = CarDEC_API(adata, weights_dir = "weights_dir/CarDEC_LVG Weights", batch
```

```
Trying to set attribute `.var` of view, copying.
/Users/jlakkis/anaconda3/envs/test/lib/python3.7/site-packages/scanpy/preprocess
ing/_simple.py:848: UserWarning: Revieved a view of an AnnData. Making a copy.
  view_to_actual(adata)
```

Fit the CarDEC Model

Now, build the model. If weights for the autoencoder do not exist in the weights directory, the autoencoder will be pretrained and its weights will be saved.

In [ ]:
```
CarDEC.build_model(n_clusters = 11)
```

```
Pretrain weight index file not detected, pretraining autoencoder weights.

Epoch 000: Training Loss: 0.956, Validation Loss: 0.947, Time: 4.6 s
Epoch 001: Training Loss: 0.932, Validation Loss: 0.931, Time: 4.5 s
Epoch 002: Training Loss: 0.922, Validation Loss: 0.926, Time: 5.0 s
Epoch 003: Training Loss: 0.917, Validation Loss: 0.923, Time: 5.2 s
Epoch 004: Training Loss: 0.914, Validation Loss: 0.920, Time: 4.6 s
Epoch 005: Training Loss: 0.911, Validation Loss: 0.918, Time: 4.5 s
Epoch 006: Training Loss: 0.910, Validation Loss: 0.917, Time: 4.5 s
Epoch 007: Training Loss: 0.908, Validation Loss: 0.916, Time: 4.5 s
Epoch 008: Training Loss: 0.908, Validation Loss: 0.915, Time: 4.5 s
Epoch 009: Training Loss: 0.906, Validation Loss: 0.914, Time: 4.4 s
Epoch 010: Training Loss: 0.905, Validation Loss: 0.913, Time: 4.4 s
Epoch 011: Training Loss: 0.904, Validation Loss: 0.914, Time: 4.5 s
Epoch 012: Training Loss: 0.903, Validation Loss: 0.912, Time: 4.7 s
Epoch 013: Training Loss: 0.903, Validation Loss: 0.912, Time: 4.5 s
Epoch 014: Training Loss: 0.901, Validation Loss: 0.912, Time: 4.5 s
Epoch 015: Training Loss: 0.901, Validation Loss: 0.911, Time: 4.5 s

Decaying Learning Rate to: 3.3333334e-05
Epoch 016: Training Loss: 0.899, Validation Loss: 0.910, Time: 4.4 s
```

```
Epoch 017: Training Loss: 0.899, Validation Loss: 0.912, Time: 4.4 s
Epoch 018: Training Loss: 0.899, Validation Loss: 0.911, Time: 4.4 s
Epoch 019: Training Loss: 0.899, Validation Loss: 0.911, Time: 4.4 s

Decaying Learning Rate to: 1.1111111e-05
Epoch 020: Training Loss: 0.899, Validation Loss: 0.910, Time: 4.5 s
Epoch 021: Training Loss: 0.899, Validation Loss: 0.910, Time: 4.4 s
Epoch 022: Training Loss: 0.898, Validation Loss: 0.912, Time: 4.4 s

Decaying Learning Rate to: 3.703704e-06
Epoch 023: Training Loss: 0.898, Validation Loss: 0.913, Time: 4.4 s
Epoch 024: Training Loss: 0.898, Validation Loss: 0.912, Time: 4.5 s
Epoch 025: Training Loss: 0.898, Validation Loss: 0.913, Time: 4.5 s

Training Completed
Total training time: 117.72 seconds

Initializing cluster centroids using the louvain method.
/Users/jlakkis/anaconda3/envs/test/lib/python3.7/site-packages/numba/np/ufunc/pa
rallel.py:355: NumbaWarning: The TBB threading layer requires TBB version 2019.5
or later i.e., TBB_INTERFACE_VERSION >= 11005. Found TBB_INTERFACE_VERSION = 110
00. The TBB threading layer is disabled.
  warnings.warn(problem)
 11 clusters detected.


-----------------------CarDEC Architecture-----------------------

Model: "car_dec__model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder (Sequential)         multiple                  260256
_________________________________________________________________
decoder (Sequential)         multiple                  262224
_________________________________________________________________
encoderLVG (Sequential)      multiple                  2062880
_________________________________________________________________
decoderLVG (Sequential)      multiple                  2083027
_________________________________________________________________
clustering (ClusteringLayer) multiple                  352
=================================================================
Total params: 4,668,739
Trainable params: 4,668,739
Non-trainable params: 0
_________________________________________________________________

--------------------Encoder Sub-Architecture--------------------

Model: "encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_0 (Dense)            multiple                  256128
_________________________________________________________________
embedding (Dense)            multiple                  4128
=================================================================
Total params: 260,256
Trainable params: 260,256
Non-trainable params: 0
_________________________________________________________________

------------------Base Decoder Sub-Architecture------------------

Model: "decoder"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
decoder0 (Dense)             multiple                  4224      
_________________________________________________________________
output (Dense)               multiple                  258000    
=================================================================
Total params: 262,224
Trainable params: 262,224
Non-trainable params: 0
_________________________________________________________________

------------------LVG Encoder Sub-Architecture------------------

Model: "encoderLVG"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
encoder0 (Dense)             multiple                  2058752   
_________________________________________________________________
embedding (Dense)            multiple                  4128      
=================================================================
Total params: 2,062,880
Trainable params: 2,062,880
Non-trainable params: 0
_________________________________________________________________

----------------LVG Base Decoder Sub-Architecture----------------

Model: "decoderLVG"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
decoderLVG0 (Dense)          multiple                  8320      
_________________________________________________________________
outputLVG (Dense)            multiple                  2074707   
=================================================================
Total params: 2,083,027
Trainable params: 2,083,027
Non-trainable params: 0
_________________________________________________________________
```

Now, call the make_inference method to finetune CarDEC. Doing so will finetune the model, and then produce denoised features on the zscore scale. If weights for the full model are already saved in the weights directory, these weights will be loaded, rather than training the full model.

In [ ]:

```
CarDEC.make_inference()
```

```
CarDEC Model Weights not detected. Training full model.

Iter 000 Loss: [Training: 0.970, Validation Cluster: 0.948, Validation AE: 0.91
5], Label Change: 0.011, Time: 32.9 s
Iter 001 Loss: [Training: 1.054, Validation Cluster: 1.001, Validation AE: 0.91
7], Label Change: 0.002, Time: 31.7 s
Iter 002 Loss: [Training: 1.143, Validation Cluster: 1.075, Validation AE: 0.91
9], Label Change: 0.002, Time: 30.2 s
Iter 003 Loss: [Training: 1.199, Validation Cluster: 1.146, Validation AE: 0.92
0], Label Change: 0.001, Time: 29.7 s

Decaying Learning Rate to: 3.3333334e-05
Iter 004 Loss: [Training: 1.219, Validation Cluster: 1.209, Validation AE: 0.92
1], Label Change: 0.000, Time: 31.6 s
Iter 005 Loss: [Training: 1.215, Validation Cluster: 1.207, Validation AE: 0.92
```

```
0], Label Change: 0.000, Time: 30.1 s
Iter 006 Loss: [Training: 1.210, Validation Cluster: 1.204, Validation AE: 0.92
0], Label Change: 0.000, Time: 31.9 s

Decaying Learning Rate to: 1.1111111e-05

Autoencoder_loss  0.9204132 not improving.
Proportion of Labels Changed:  0.00023103835236649284  is less than tolerance of
0.005

Reached tolerance threshold. Stop training.

The final cluster assignments are:
0     6136
1     4474
2     4039
3     3482
4     3001
5     2627
6     2248
7     1834
8     1168
9      661
10     628
dtype: int64

Total Runtime is 424.5527148246765

The CarDEC model is now making inference on the data matrix.
Inference completed, results added.
```

To get denoised features on the count scale, call the model_counts method.

In [ ]:

```
CarDEC.model_counts()
```

```
Weight files for count models not detected. Training HVG count model.

Epoch 000: Training Loss: 0.266, Validation Loss: 0.231, Time: 10.2 s
Epoch 001: Training Loss: 0.228, Validation Loss: 0.230, Time: 10.2 s
Epoch 002: Training Loss: 0.227, Validation Loss: 0.229, Time: 10.1 s
Epoch 003: Training Loss: 0.227, Validation Loss: 0.229, Time: 10.2 s
Epoch 004: Training Loss: 0.226, Validation Loss: 0.228, Time: 10.2 s
Epoch 005: Training Loss: 0.226, Validation Loss: 0.228, Time: 10.2 s
Epoch 006: Training Loss: 0.226, Validation Loss: 0.228, Time: 10.4 s
Epoch 007: Training Loss: 0.225, Validation Loss: 0.228, Time: 10.3 s

Decaying Learning Rate to: 0.00033333336
Epoch 008: Training Loss: 0.225, Validation Loss: 0.227, Time: 10.1 s
Epoch 009: Training Loss: 0.225, Validation Loss: 0.227, Time: 10.1 s
Epoch 010: Training Loss: 0.225, Validation Loss: 0.227, Time: 10.3 s
Epoch 011: Training Loss: 0.225, Validation Loss: 0.227, Time: 10.2 s

Decaying Learning Rate to: 0.00011111112
Epoch 012: Training Loss: 0.224, Validation Loss: 0.227, Time: 10.2 s
Epoch 013: Training Loss: 0.225, Validation Loss: 0.227, Time: 10.1 s
Epoch 014: Training Loss: 0.224, Validation Loss: 0.227, Time: 10.3 s

Decaying Learning Rate to: 3.703704e-05
Epoch 015: Training Loss: 0.224, Validation Loss: 0.227, Time: 10.2 s
Epoch 016: Training Loss: 0.224, Validation Loss: 0.227, Time: 10.3 s
Epoch 017: Training Loss: 0.224, Validation Loss: 0.227, Time: 10.1 s

Training Completed
```

```
Total training time: 183.81 seconds




Training LVG count model.

Epoch 000: Training Loss: 0.180, Validation Loss: 0.172, Time: 70.4 s
Epoch 001: Training Loss: 0.170, Validation Loss: 0.171, Time: 75.6 s
Epoch 002: Training Loss: 0.170, Validation Loss: 0.171, Time: 77.2 s
Epoch 003: Training Loss: 0.169, Validation Loss: 0.171, Time: 85.1 s

Decaying Learning Rate to: 0.00033333336
Epoch 004: Training Loss: 0.169, Validation Loss: 0.171, Time: 75.1 s
Epoch 005: Training Loss: 0.169, Validation Loss: 0.171, Time: 71.3 s
Epoch 006: Training Loss: 0.168, Validation Loss: 0.171, Time: 68.9 s
Epoch 007: Training Loss: 0.168, Validation Loss: 0.171, Time: 79.9 s

Decaying Learning Rate to: 0.00011111112
Epoch 008: Training Loss: 0.168, Validation Loss: 0.171, Time: 76.3 s
Epoch 009: Training Loss: 0.168, Validation Loss: 0.171, Time: 85.1 s
Epoch 010: Training Loss: 0.168, Validation Loss: 0.171, Time: 85.1 s

Decaying Learning Rate to: 3.703704e-05
Epoch 011: Training Loss: 0.168, Validation Loss: 0.171, Time: 79.2 s
Epoch 012: Training Loss: 0.168, Validation Loss: 0.171, Time: 75.2 s
Epoch 013: Training Loss: 0.168, Validation Loss: 0.171, Time: 76.1 s

Training Completed
Total training time: 1080.68 seconds
```

As mentioned before, the output is accessed via CarDEC.dataset. Let's look at the output structure.

In [ ]:

```
print("The overall structure of the output is: \n")
print(CarDEC.dataset)

CarDEC.dataset.X #The main layer of the output object contains the original coun
CarDEC.dataset.layers['denoised'] #These are the denoised features, on the zscor
CarDEC.dataset.layers['denoised counts'] #These are the denoised features, on th
CarDEC.dataset.var['Variance Type'] #This is a vector that informs which genes a
CarDEC.dataset.obsm['embedding'] #This is the CarDEC low-dimensional embedding a
CarDEC.dataset.obsm['precluster denoised'] #This is the matrix of feature zscore
CarDEC.dataset.obsm['precluster embedding'] #This is the latent embedding from t

"""Example, this is how to get the matrix of denoised counts for only high varia
HVG_denoised = deepcopy(CarDEC.dataset.layers['denoised counts'][:, CarDEC.datas

"""Example, this is how to get the matrix of denoised counts for only low varian
LVG_denoised = deepcopy(CarDEC.dataset.layers['denoised counts'][:, CarDEC.datas
```

```
The overall structure of the output is:

AnnData object with n_obs × n_vars = 30298 × 18083
    obs: 'batch', 'sample', 'macaque_id', 'nGene', 'nTranscripts', 'cluster', 'r
egion', 'class', 'n_genes', 'n_counts', 'size factors'
    var: 'n_cells', 'n_counts', 'Variance Type'
    uns: 'log1p', 'num_batch'
    obsm: 'cluster memberships', 'embedding', 'LVG embedding', 'precluster denoi
sed', 'precluster embedding', 'initial assignments'
    layers: 'denoised', 'denoised counts'
```

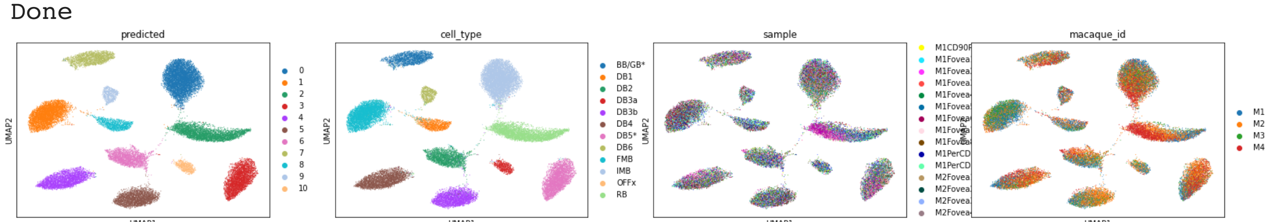
Working with the embedding and cluster assignments

Here, I demonstrate how to access the latent embedding of CarDEC and how to use it for UMAP visualization. I also demonstrate how to get the CarDEC cluster assignments.

In [ ]:

```
"""Get the predicted cluster assignments and compute cluster accuracy metrics"""

embedded = deepcopy(CarDEC.dataset.obsm['embedding']) #The latent embedding nump

q = deepcopy(CarDEC.dataset.obsm['cluster memberships']) #The cluster membership
labels = np.argmax(q, axis=1)
labels = [str(x) for x in labels]

true_celltype = list(CarDEC.dataset.obs['cluster']) #Note: all obs properties fr

print("CarDEC Clustering Results")
ARI, NMI, Purity = [metric(CarDEC.dataset.obs['cluster'], labels) for metric in

print ("ARI = {0:.4f}".format(ARI))
print ("NMI = {0:.4f}".format(NMI))
print ("Purity = {0:.4f}".format(Purity))

"""Create a scanpy AnnData object with the latent embedding as the matrix, to pe
formatting = AnnData(embedded)
formatting.obs["cell_type"] = list(CarDEC.dataset.obs['cluster'])
formatting.obs["predicted"] = list(labels)
formatting.obs["sample"] = list(CarDEC.dataset.obs['sample'])
formatting.obs["macaque_id"] = list(CarDEC.dataset.obs['macaque_id'])

sc.pp.neighbors(formatting, n_neighbors = 15, use_rep = 'X')
sc.tl.umap(formatting)
sc.pl.umap(formatting, color = ["predicted", "cell_type", "sample", "macaque_id"
print("Done")
```

```
CarDEC Clustering Results
ARI = 0.9772
NMI = 0.9629
Purity = 0.9850
... storing 'cell_type' as categorical
... storing 'predicted' as categorical
... storing 'sample' as categorical
... storing 'macaque_id' as categorical
Done
```

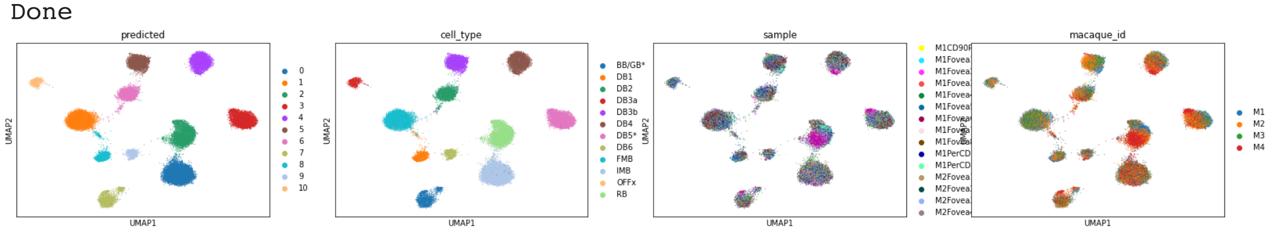


In [ ]:

```
"""Get the predicted labels and compute adjusted rand score for the precluster e

preclust_emb = deepcopy(CarDEC.dataset.obsm['precluster embedding'])

formatting = AnnData(preclust_emb)
sc.pp.neighbors(formatting, n_neighbors = 15, use_rep = 'X')
res = find_resolution(formatting, 11)
sc.tl.louvain(formatting, resolution = res)
```

```
print(str(len(np.unique(labels))) + " Clusters Detected")

labels = formatting.obs['louvain']
type_strings = list(CarDEC.dataset.obs['cluster'])

ARI, NMI, Purity = [metric(CarDEC.dataset.obs['cluster'], labels) for metric in

print("Pretrained Autoencoder Clustering Results")
print ("ARI = {0:.4f}".format(ARI))
print ("NMI = {0:.4f}".format(NMI))
print ("Purity = {0:.4f}".format(Purity))

formatting.obs["cell_type"] = list(CarDEC.dataset.obs['cluster'])
formatting.obs["predicted"] = list(labels)
formatting.obs["sample"] = list(CarDEC.dataset.obs['sample'])
formatting.obs["region"] = list(CarDEC.dataset.obs['region'])
formatting.obs["macaque_id"] = list(CarDEC.dataset.obs['macaque_id'])
sc.tl.umap(formatting)
sc.pl.umap(formatting, color = ["predicted", "cell_type", "sample", "macaque_id"
print("Done")
```

```
11 Clusters Detected
Pretrained Autoencoder Clustering Results
ARI = 0.9658
NMI = 0.9492
Purity = 0.9805
... storing 'cell_type' as categorical
... storing 'predicted' as categorical
... storing 'sample' as categorical
... storing 'region' as categorical
... storing 'macaque_id' as categorical
Done
```



Working with the denoised counts

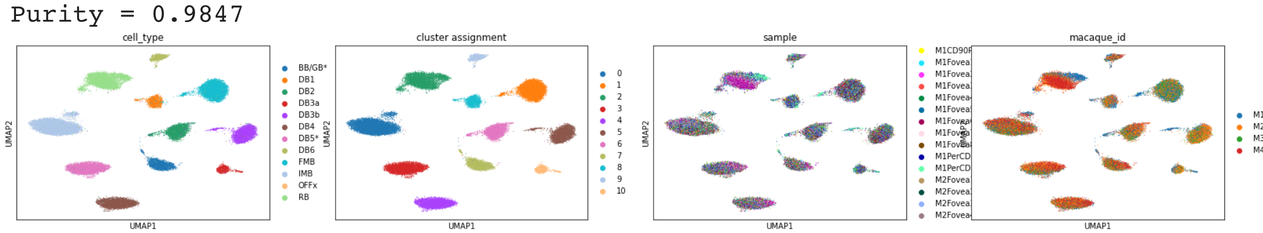
Here I work with the denoised counts. I demonstrate the use of the denoised counts for UMAP embedding and louvain clustering with scanpy.

In [ ]:

```
"""Assessing denoised Counts"""

temporary = AnnData(deepcopy(CarDEC.dataset.layers['denoised counts']))
temporary.obs = CarDEC.dataset.obs
temporary.obs['cell_type'] = temporary.obs['cluster']

sc.pp.normalize_total(temporary)
sc.pp.log1p(temporary)
sc.pp.scale(temporary)

sc.tl.pca(temporary, svd_solver='arpack')
sc.pp.neighbors(temporary, n_neighbors = 15)

res = find_resolution(temporary, 11)
```

```
sc.tl.louvain(temporary, resolution = res)
temporary.obs['cluster assignment'] = temporary.obs['louvain']

sc.tl.umap(temporary)
sc.pl.umap(temporary, color = ["cell_type", "cluster assignment", "sample", "mac

ARI, NMI, Purity = [metric(temporary.obs['cell_type'], temporary.obs['cluster as

print("CarDEC Denoising Results using all denoised counts")
print ("ARI = {0:.4f}".format(ARI))
print ("NMI = {0:.4f}".format(NMI))
print ("Purity = {0:.4f}".format(Purity))
```

```
CarDEC Denoising Results using all denoised counts
ARI = 0.9760
NMI = 0.9625
Purity = 0.9847
```

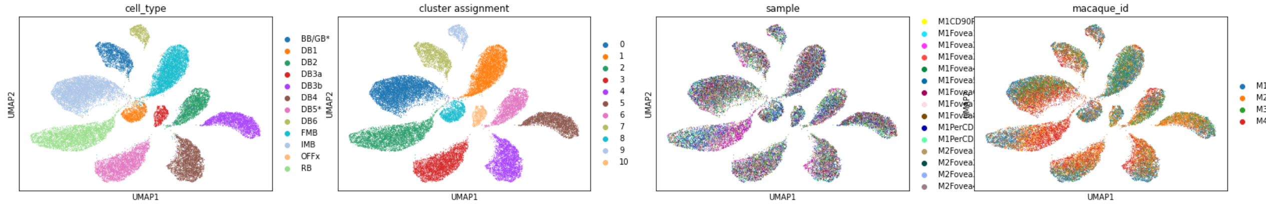


Working with only the high variance denoised counts

In [ ]:

```
"""Assessing HVG denoised Counts"""

temporary = AnnData(deepcopy(CarDEC.dataset.layers['denoised counts'][:, CarDEC.
temporary.obs = CarDEC.dataset.obs
temporary.obs['cell_type'] = temporary.obs['cluster']

sc.pp.normalize_total(temporary)
sc.pp.log1p(temporary)
sc.pp.scale(temporary)

sc.tl.pca(temporary, svd_solver='arpack')
sc.pp.neighbors(temporary, n_neighbors = 15)

res = find_resolution(temporary, 11)
sc.tl.louvain(temporary, resolution = res)
temporary.obs['cluster assignment'] = temporary.obs['louvain']

sc.tl.umap(temporary)
sc.pl.umap(temporary, color = ["cell_type", "cluster assignment", "sample", "mac

ARI, NMI, Purity = [metric(temporary.obs['cell_type'], temporary.obs['cluster as

print("Clustering high variance denoised counts")
print ("ARI = {0:.4f}".format(ARI))
print ("NMI = {0:.4f}".format(NMI))
print ("Purity = {0:.4f}".format(Purity))
```

```
Clustering high variance denoised counts
ARI = 0.9766
NMI = 0.9628
Purity = 0.9849
```

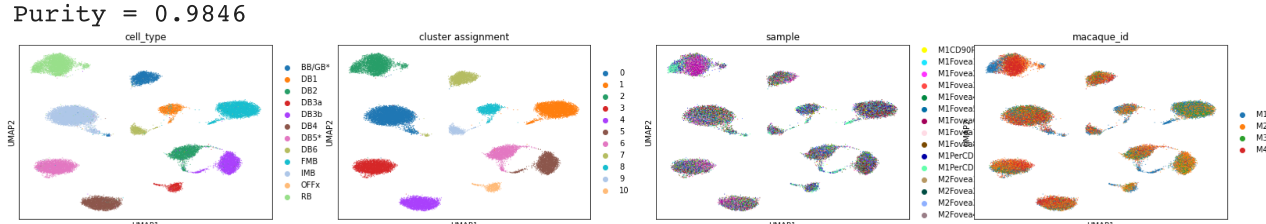


Working with only the low variance denoised counts

In [ ]:

```
"""Assessing LVG denoised Counts"""

temporary = AnnData(deepcopy(CarDEC.dataset.layers['denoised counts'][:, CarDEC.
temporary.obs = CarDEC.dataset.obs
temporary.obs['cell_type'] = temporary.obs['cluster']

sc.pp.normalize_total(temporary)
sc.pp.log1p(temporary)
sc.pp.scale(temporary)

sc.tl.pca(temporary, svd_solver='arpack')
sc.pp.neighbors(temporary, n_neighbors = 15)

res = find_resolution(temporary, 11)
sc.tl.louvain(temporary, resolution = res)
temporary.obs['cluster assignment'] = temporary.obs['louvain']

sc.tl.umap(temporary)
sc.pl.umap(temporary, color = ["cell_type", "cluster assignment", "sample", "mac

ARI, NMI, Purity = [metric(temporary.obs['cell_type'], temporary.obs['cluster as

print("Clustering low variance denoised counts")
print ("ARI = {0:.4f}".format(ARI))
print ("NMI = {0:.4f}".format(NMI))
print ("Purity = {0:.4f}".format(Purity))
```

```
Clustering low variance denoised counts
ARI = 0.9762
NMI = 0.9619
Purity = 0.9846
```



Working with the denoised counts on the zscore scale

In [ ]:

```
"""Assessing denoised zscore features"""

temporary = AnnData(deepcopy(CarDEC.dataset.layers['denoised']))
temporary.obs = CarDEC.dataset.obs
temporary.obs['cell_type'] = temporary.obs['cluster']

sc.tl.pca(temporary, svd_solver='arpack')
```

```
sc.pp.neighbors(temporary, n_neighbors = 15)

res = find_resolution(temporary, 11)
sc.tl.louvain(temporary, resolution = res)
temporary.obs['cluster assignment'] = temporary.obs['louvain']

sc.tl.umap(temporary)
sc.pl.umap(temporary, color = ["cell_type", "cluster assignment", "sample", "mac

ARI, NMI, Purity = [metric(temporary.obs['cell_type'], temporary.obs['cluster as

print("CarDEC Denoising Results using all denoised features")
print ("ARI = {0:.4f}".format(ARI))
print ("NMI = {0:.4f}".format(NMI))
print ("Purity = {0:.4f}".format(Purity))
```

```
CarDEC Denoising Results using all denoised features
ARI = 0.9769
NMI = 0.9630
Purity = 0.9849
```