Q1) 1) Read in the data and perform filtering and normalisation: Explain the reason for your chosen threshold for filtering with appropriate figures and rational for normalisation?

1. Define the Data Directory and Sample Folders

In [1]:
```
import os                     # For handling file paths and directory operations.
import scanpy as sc           # Main package for single-cell RNA-seq analysis.
import harmonypy as hm        # For batch integration using Harmony.
import matplotlib.pyplot as plt  # For plotting figures.
import pandas as pd           # For data manipulation and creating dataframes.
import seaborn as sns         # For advanced plotting (e.g., bar plots).
import anndata as ad
import numpy as np
import igraph

# Define data directory
base_path = "/data/home/ha24967/Assignment/"


# List directories
sample_folders = [folder for folder in os.listdir(base_path) if os.path.isdir(os.pa
sample_folders = sorted(sample_folders)  # sort the list.


print("Sorted sample folders:", sample_folders)
```

```
Sorted sample folders: ['P2_cSCC', 'P2_normal', 'P3_cSCC1', 'P3_cSCC2', 'P3_normal',
'P4_cSCC1', 'P4_cSCC2', 'P4_normal', 'P5_cSCC', 'P5_normal']
```

2. Group Sample Folders by Patient and Condition

In [2]:
```
#Create a dictionary called sample_groups that contains keys that are combinations
#the patient ID and the condition (e.g., "P3_cSCC"), and the values are lists of th
#folder names associated with that patient and condition.
sample_groups = {}  # Dictionary
for folder in sample_folders:
    # Extract patient ID from the folder name (e.g., "P3" from "P3_cSCC1")
    patient = folder.split('_')[0]
    # Determine the condition: if "normal" is in the folder name, it's normal; othe
    condition = "normal" if "normal" in folder.lower() else "cSCC"
    # Create a key to group by, e.g., "P3_cSCC"
    key = f"{patient}_{condition}"
    # Initialize the group if it doesn't exist yet.
    if key not in sample_groups:
        sample_groups[key] = []
    # Append the folder name to the corresponding group.
    sample_groups[key].append(folder)
```

```
print(sample_groups)
```

```
{'P2_cSCC': ['P2_cSCC'], 'P2_normal': ['P2_normal'], 'P3_cSCC': ['P3_cSCC1', 'P3_cSC
C2'], 'P3_normal': ['P3_normal'], 'P4_cSCC': ['P4_cSCC1', 'P4_cSCC2'], 'P4_normal':
['P4_normal'], 'P5_cSCC': ['P5_cSCC'], 'P5_normal': ['P5_normal']}
```

3. Load and Merge Data for Each Sample Group

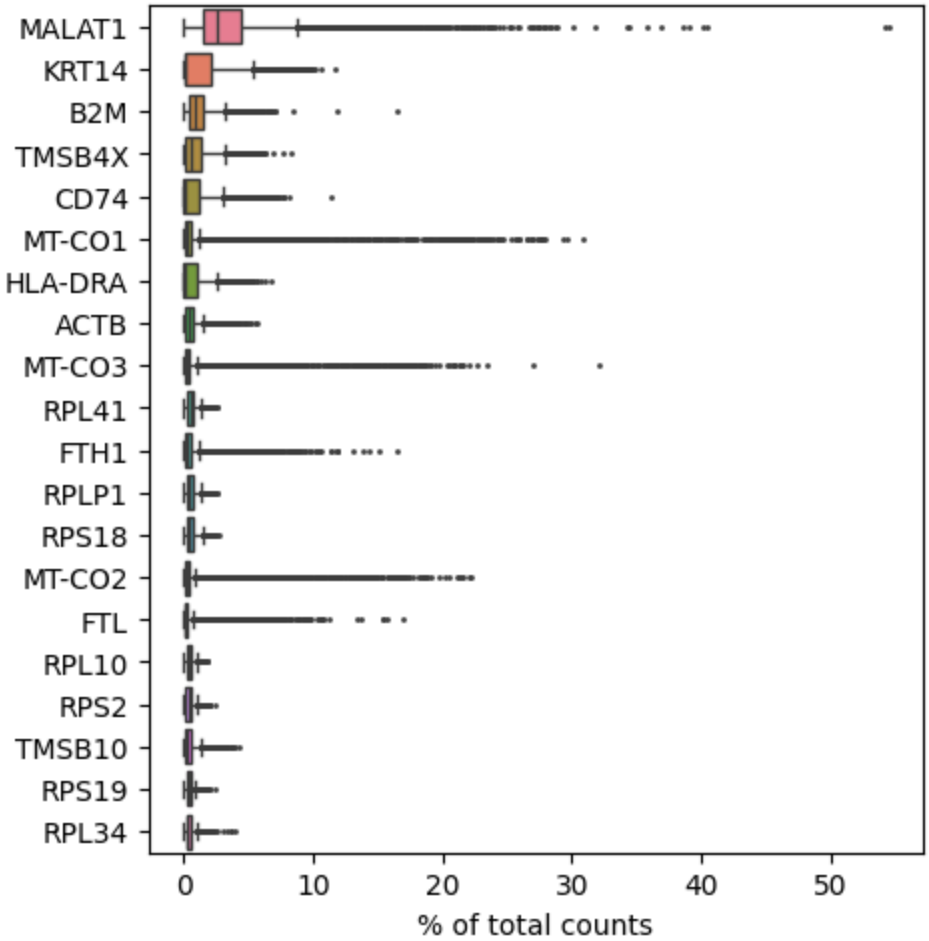
In [3]:
```
adata_dict = {}  #Dictionary to store one AnnData object per sample group

for group_key, folders in sample_groups.items():
    if len(folders) > 1:
        # More than one folder for this group: merge them
        adata_list = []  # list to store each AnnData object loaded from the differ
        for folder in folders: #Loops through each folder in the current group,
            #constructs the file path and loads the AnnData object from the file
            file_path = os.path.join(data_dir, folder, "filtered_feature_bc_matrix.
            adata = sc.read_h5ad(file_path)
            #Metadata Extraction and Annotation from Folder Names
            patient = folder.split('_')[0]
            condition = "normal" if "normal" in folder.lower() else "cSCC"
            adata.obs["sample"] = folder
            adata.obs["patient"] = patient
            adata.obs["condition"] = condition
            adata_list.append(adata)
        # Merge all parts using ad.concat
        merged_adata = ad.concat(adata_list, join='outer', index_unique=None)
        # Update the "sample" annotation for the merged object
        merged_adata.obs["sample"] = group_key
        #Stores the merged AnnData object in the dictionary adata_dict with the key
        adata_dict[group_key] = merged_adata
    else:
        # Only one folder in this group: load normally.
        folder = folders[0]
        file_path = os.path.join(data_dir, folder, "filtered_feature_bc_matrix.h5ad
        adata = sc.read_h5ad(file_path)
        patient = folder.split('_')[0]
        condition = "normal" if "normal" in folder.lower() else "cSCC"
        adata.obs["sample"] = group_key
        adata.obs["patient"] = patient
        adata.obs["condition"] = condition
        adata_dict[group_key] = adata
```

4.Merge Data from All Sample Groups

In [4]:
```
# Define the desired order of sample groups alphabetically.
ordered_keys = sorted(adata_dict.keys())
```

```python
# Create an ordered list of AnnData objects using the sorted keys.
ordered_adata_list = [adata_dict[key] for key in ordered_keys]

# Concatenate in the specified order.
adata_all = ad.concat(ordered_adata_list, join='outer', index_unique='-')

print(adata_all)
adata_all.obs.head()
print(adata_all.obs['sample'].unique())
print(adata_all.obs['sample'].value_counts())
adata_all.obs.describe()

#Genes that yield the highest fraction of counts in each single cell, across all ce
sc.pl.highest_expr_genes(adata_all, n_top=20)
```

```
AnnData object with n_obs × n_vars = 31456 × 36601
    obs: 'sample', 'patient', 'condition'
['P2_cSCC' 'P2_normal' 'P3_cSCC' 'P3_normal' 'P4_cSCC' 'P4_normal'
 'P5_cSCC' 'P5_normal']
sample
P4_cSCC      12866
P2_normal     6619
P2_cSCC       5365
P5_cSCC       2358
P3_normal     1765
P5_normal     1639
P3_cSCC        525
P4_normal      319
Name: count, dtype: int64
```

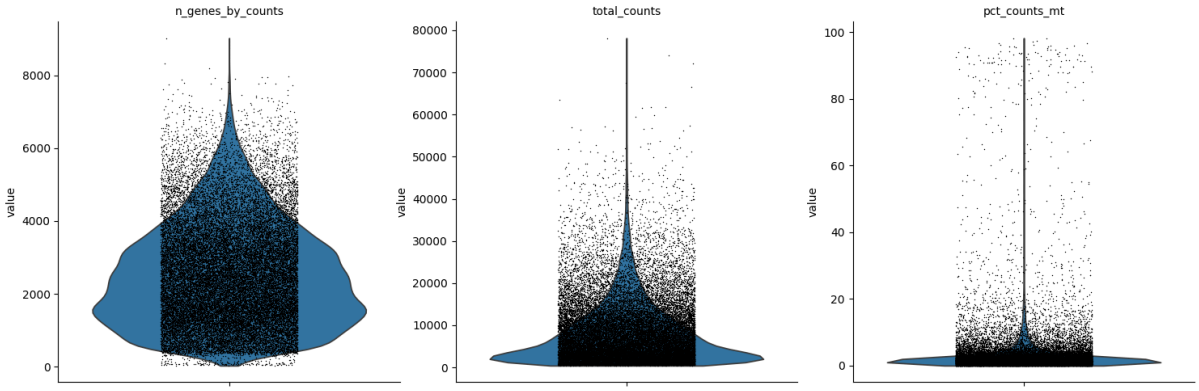


5.Quality Control (QC)

In [5]:
```
## Identify mitochondrial and ribosomal genes
adata_all.var["mt"] = adata_all.var_names.str.startswith("MT-")
adata_all.var["ribo"] = adata_all.var_names.str.startswith(("RPS", "RPL"))

# Compute QC metrics including % mitochondrial and ribosomal counts
sc.pp.calculate_qc_metrics(
    adata_all, qc_vars=["mt", "ribo"], inplace=True, log1p=False, percent_top=None
)

# Display a summary of QC metrics in the observation dataframe:
print(adata_all.obs[["n_genes_by_counts", "total_counts", "pct_counts_mt", "pct_cou


print(adata_all.var.describe())

# Generate a violin plot for selected QC metrics and save the figure
sc.pl.violin(
    adata_all,
    keys=["n_genes_by_counts", "total_counts", "pct_counts_mt"],
    jitter=0.2,
    multi_panel=True,
    rotation=30,
```

```
    save="QC_covariates.pdf",
    show=True,
)

# Filter cells with >20% mitochondrial gene counts
print("Total number of cells: {:d}".format(adata_all.n_obs))
adata_all = adata_all[adata_all.obs.pct_counts_mt < 20, :].copy()
print("Number of cells after mitochondrial filter: {:d}".format(adata_all.n_obs))

# Filter out cells with fewer than 300 genes expressed
sc.pp.filter_cells(adata_all, min_genes=300)
print("Number of cells after min gene filter: {:d}".format(adata_all.n_obs))

# Remove mitochondrial and ribosomal genes from the dataset
print("Number of genes before removal: ", adata_all.shape[1])
adata_all = adata_all[:, ~adata_all.var["mt"].values]
print("After mitochondrial gene removal: ", adata_all.shape[1])
adata_all = adata_all[:, ~adata_all.var["ribo"].values]
print("After ribosomal gene removal: ", adata_all.shape[1])
```

```
       n_genes_by_counts  total_counts  pct_counts_mt  pct_counts_ribo
count       31456.000000  31456.000000   31456.000000     31456.000000
mean         2510.983882   8185.354492       3.066280        15.605822
std          1410.586024   7541.133301       7.380524         8.226326
min            27.000000    436.000000       0.000000         0.000000
25%          1396.000000   2720.000000       0.911259         9.563622
50%          2331.000000   5969.500000       1.524718        15.636131
75%          3418.000000  11276.500000       2.646164        20.808786
max          9027.000000  78150.000000      98.264641        58.778625
       n_cells_by_counts   mean_counts  pct_dropout_by_counts  total_counts
count       36601.000000  36601.000000           36601.000000  3.660100e+04
mean         2158.015054      0.223637              93.139576  7.034740e+03
std          4182.682159      2.022532              13.296930  6.362774e+04
min             0.000000      0.000000               0.480036  0.000000e+00
25%             7.000000      0.000223              92.526068  7.000000e+00
50%           123.000000      0.004323              99.608978  1.360000e+02
75%          2351.000000      0.095530              99.977747  3.005000e+03
max         31305.000000    203.930542             100.000000  6.414839e+06
WARNING: saving figure to file figures/violinQC_covariates.pdf
```



```
Total number of cells: 31456
Number of cells after mitochondrial filter: 30935
Number of cells after min gene filter: 30927
Number of genes before removal:  36601
After mitochondrial gene removal:  36588
After ribosomal gene removal:  36485
```

6.Data Normalisation, identification of HVG and Scaling

In [6]:
```
# Make a copy
adata_all = adata_all.copy()

# Save raw counts
adata_all.layers["raw_counts"] = adata_all.X.copy()

# Normalize + log1p
sc.pp.normalize_total(adata_all, target_sum=1e4,inplace=True)
adata_all.layers["normalised"] = adata_all.X.copy()
sc.pp.log1p(adata_all)
adata_all.layers["log1p"] = adata_all.X.copy()

# Save the normalized + log1p data for DE etc.
adata_all.raw = adata_all

# HVG selection using raw counts
sc.pp.highly_variable_genes(
    adata_all,
    flavor='seurat_v3',
    layer='raw_counts',
    subset=True,
    n_top_genes=2000,
    span=0.4,
    min_disp=0.5,
    min_mean=0.0125,
    max_mean=3,
    batch_key='sample'
)

# Regress out unwanted variation
sc.pp.regress_out(adata_all, ['total_counts', 'pct_counts_mt'])

# Scale the data
sc.pp.scale(adata_all, max_value=10)
```

```
OMP: Info #276: omp_set_nested routine deprecated, please use omp_set_max_active_lev
els instead.
```

In [7]:
```
print(adata_all.X.shape)
print(adata_all.raw.X.shape)
adata_all.var.describe()
```

```
(30927, 2000)
(30927, 36485)
```

Out[7]:

	n_cells_by_counts	mean_counts	pct_dropout_by_counts	total_counts	highly_variabl
count	2000.000000	2000.000000	2000.000000	2.000000e+03	2000.
mean	4242.342000	0.938358	86.513409	2.951700e+04	608.
std	5191.220188	4.983627	16.503116	1.567650e+05	263.
min	2.000000	0.000477	4.104145	1.500000e+01	5.
25%	506.750000	0.034254	79.152626	1.077500e+03	406.
50%	1962.500000	0.143852	93.761127	4.525000e+03	657.
75%	6557.750000	0.521951	98.389020	1.641850e+04	836.
max	30165.000000	158.644806	99.993642	4.990331e+06	985.

Explain the reason for your chosen threshold for filtering with appropriate figures and rational for normalisation? Filtering Thresholds:

------ Low-quality cells are removed based on:

---> Number of genes per cell (e.g., keep cells with 200–2,500 genes): Removes empty droplets and potential multiplets.

---> Mitochondrial gene percentage (e.g., exclude cells with >5–10% mitochondrial genes): High mitochondrial content may indicate dying or stressed cells.

---> Total UMI counts: Very low UMI counts suggest low-quality cells; very high counts might indicate doublets.

Rationale: Filtering ensures that technical noise and biologically irrelevant cells (e.g., dead/dying cells) do not distort downstream analyses like clustering or differential expression.

Normalization:

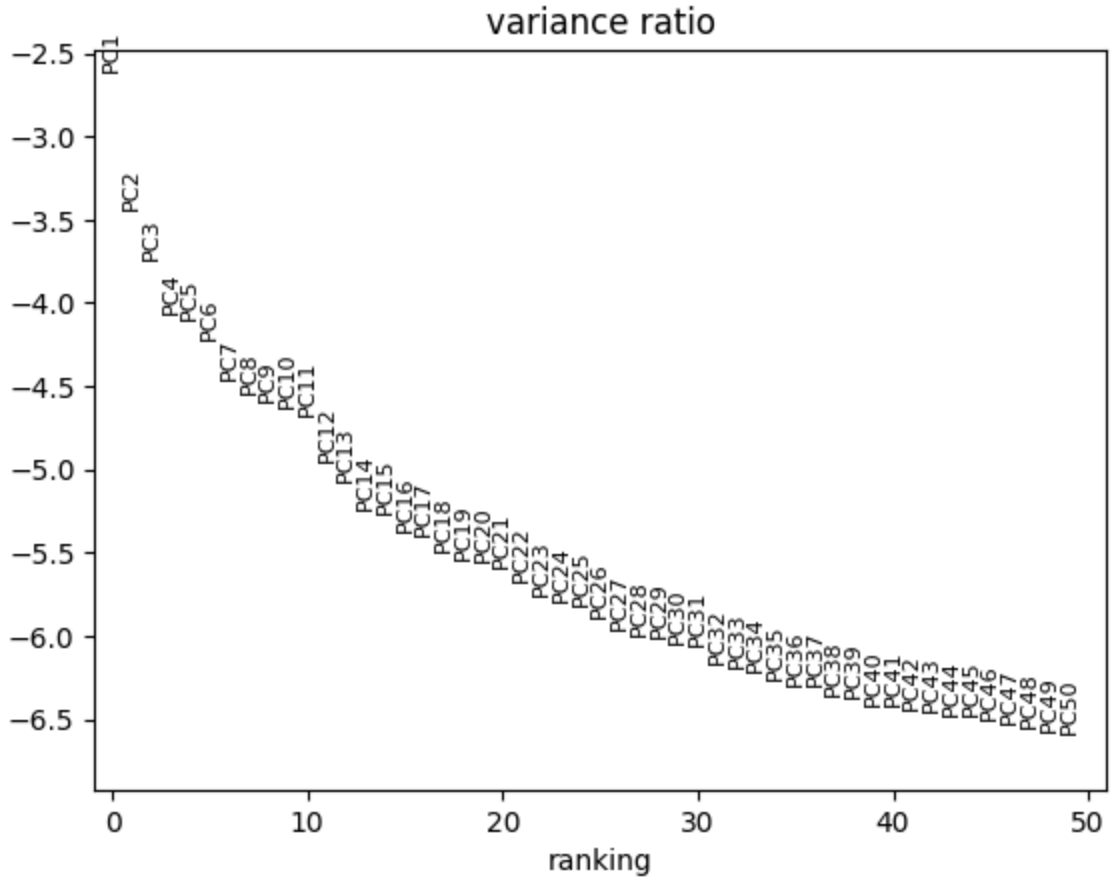
Each cell's gene counts are scaled to a total (e.g., 10,000 reads) and log-transformed to make gene expression levels comparable across cells by removing depth/coverage bias and stabilizing variance.

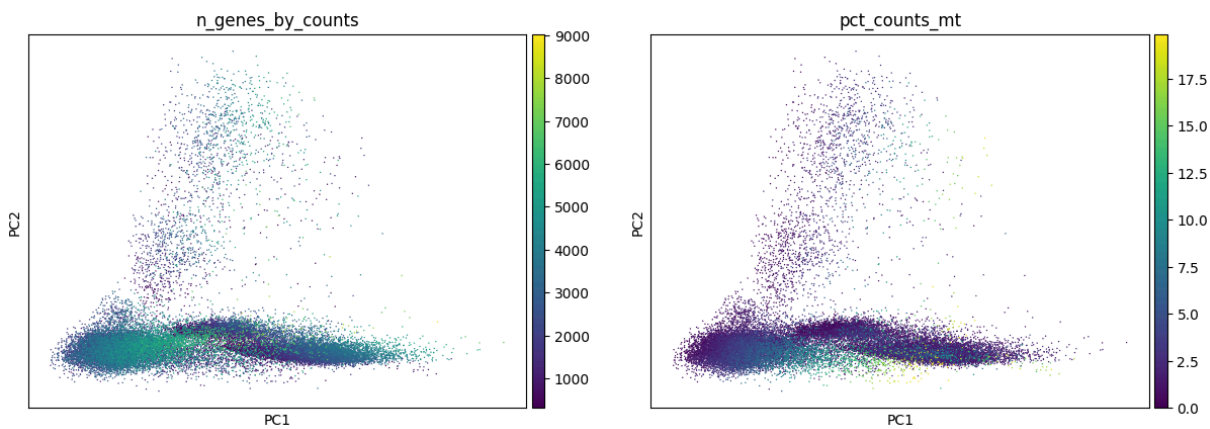
7.PCA

In [8]:

```
# Run PCA restricting to HVGs using the new mask_var argument; setting random_state
sc.tl.pca(adata_all, svd_solver='arpack', mask_var="highly_variable", random_state=

# Generate an elbow plot to visualize the variance explained by the first 50 PCs
```

```
sc.pl.pca_variance_ratio(adata_all, log=True, n_pcs=50, show=True)

#Scatter plot
sc.pl.pca(adata_all, color=['n_genes_by_counts','pct_counts_mt'], color_map='viridi
```





In [9]:
```
#Inspect adata_all
print(adata_all)
print("obs columns:", list(adata_all.obs.columns))
print("obsp keys:", list(adata_all.obsp.keys()))   #empty
print("obsm keys:", list(adata_all.obsm.keys()))   #'X_pca'
print("uns keys:", list(adata_all.uns.keys()))     #PCA settings
```

```
AnnData object with n_obs × n_vars = 30927 × 2000
    obs: 'sample', 'patient', 'condition', 'n_genes_by_counts', 'total_counts', 'tot
al_counts_mt', 'pct_counts_mt', 'total_counts_ribo', 'pct_counts_ribo', 'n_genes'
    var: 'mt', 'ribo', 'n_cells_by_counts', 'mean_counts', 'pct_dropout_by_counts',
'total_counts', 'highly_variable', 'highly_variable_rank', 'means', 'variances', 'va
riances_norm', 'highly_variable_nbatches', 'mean', 'std'
    uns: 'log1p', 'hvg', 'pca'
    obsm: 'X_pca'
    varm: 'PCs'
    layers: 'raw_counts', 'normalised', 'log1p'
obs columns: ['sample', 'patient', 'condition', 'n_genes_by_counts', 'total_counts',
'total_counts_mt', 'pct_counts_mt', 'total_counts_ribo', 'pct_counts_ribo', 'n_gene
s']
obsp keys: []
obsm keys: ['X_pca']
uns keys: ['log1p', 'hvg', 'pca']
```

Q2) If you are using dimensionality reduction and integration methods in your pipeline, explain why and how it will affect your downstream analysis?

Dimensionality Reduction (e.g., PCA, UMAP):

---> Reduces high-dimensional gene expression data to lower dimensions (e.g., 2D) for visualization and clustering.

---> Removes noise and highlights biologically relevant variation.

Impact: Helps separate distinct cell populations and reveals underlying structure in the data.

Integration (Scanpy)

---> Corrects for batch effects (e.g., donor-specific, sample-specific artifacts).

Impact: Improves clustering accuracy and ensures that biological signals, not technical artifacts, drive downstream analysis.

In [ ]:

```
Q3)     Perform neighbourhood, clustering and UMAP analysis:
a)      Compare at least three different resolutions and which one you will choose
b)      Comment how these steps are affecting your Seurat / Anndata object

Hint: look at the metadata / explore the  Anndata object

Ans- a) Resolution Comparison
Clustering is sensitive to resolution in graph-based methods:

---> Low resolution (e.g., 0.2): Few broad clusters.

---> Medium (e.g., 0.6-1.0): Balanced resolution, captures major and some subpopula

---> High (e.g., 1.2-2.0): More granular clustering, may over-split biologically si

Choice Rationale:

---> The optimal resolution balances biological interpretability and cluster separa
```

```
---> Chosen based on cluster marker expression, silhouette scores, and corresponden

b) Effects on Anndata Object
These steps add:

---> A neighbors graph to the object (e.g., neighbors in AnnData).

---> A clustering assignment (leiden).
```

4. Read the data again and perform all your steps again excluding integration step and compare the UMAP with and without integration. If the UMAP looks different, explain briefly why?

8.Neighbourhood, Clustering, and UMAP Analysis with integration

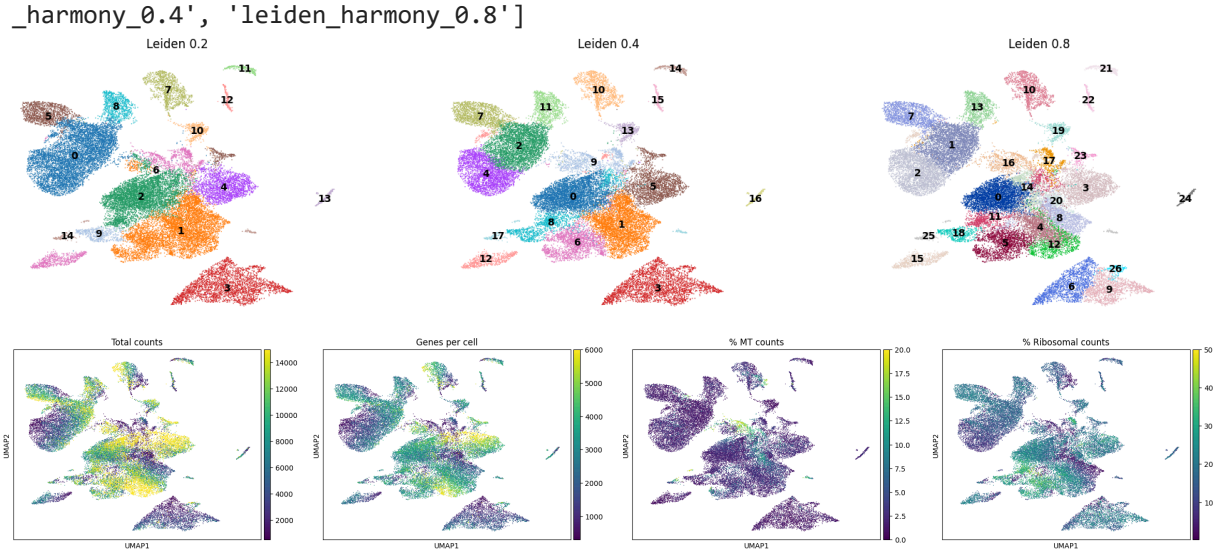
In [10]:
```
#Harmony integration on the PCA space
# Specify the batch variable(s) you want to correct for:
cat_vars_to_regress = ['sample']
ho = hm.run_harmony(
    adata_all.obsm['X_pca'],  #the PCA coordinates
    adata_all.obs,            #the DataFrame of cell metadata
    cat_vars_to_regress,      #list of metadata columns to regress out
    max_iter_harmony=20       #number of Harmony iterations (default is 10-20)
)
#Store the corrected PCs back into the AnnData object
adata_all.obsm['X_pca_harmony'] = ho.Z_corr.T  #transpose

#Build neighbor graph using Harmony PCs
sc.pp.neighbors(
    adata_all,
    use_rep='X_pca_harmony',
    n_neighbors=15,
    n_pcs=20,
    random_state=0
)

#Inspect adata_all after neighbors, before clustering
print("---After sc.pp.neighbors()---")

print("obs columns:", list(adata_all.obs.columns))
print("obsm keys:", list(adata_all.obsm.keys()))
print("obsp keys:", list(adata_all.obsp.keys()))
print("uns keys:", list(adata_all.uns.keys()))

#Compute UMAP after Harmony and neighbor
sc.tl.umap(adata_all)

#Inspect adata_all after umap
print("---After sc.tl.umap()---")
```

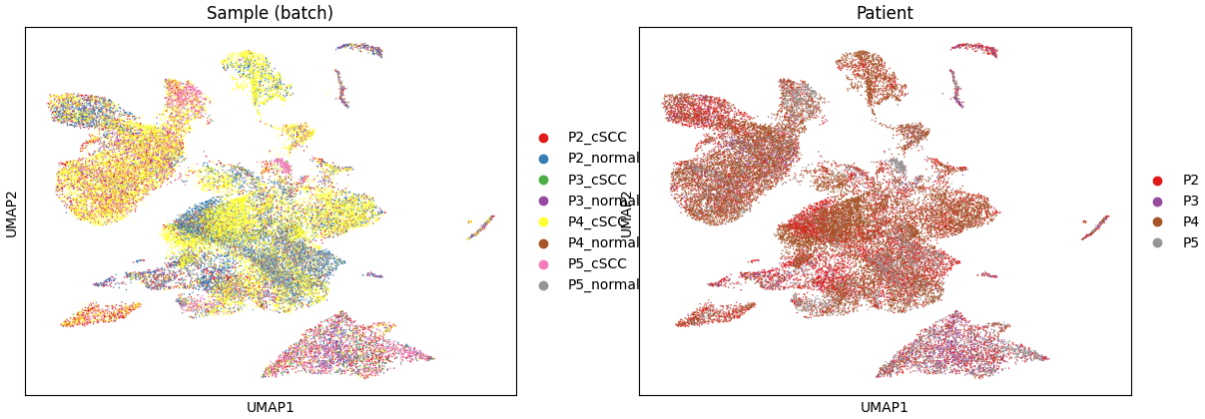
```
print("obs columns:", list(adata_all.obs.columns))
print("obsm keys:", list(adata_all.obsm.keys()))
print("obsp keys:", list(adata_all.obsp.keys()))
print("uns keys:", list(adata_all.uns.keys()))

#Leiden clustering at 3 resolutions
for res in [0.2,0.4, 0.8]:
    sc.tl.leiden(
        adata_all,
        resolution=res,
        key_added=f'leiden_harmony_{res}',
        random_state=0
    )

#Inspect adata_all after clustering (chosen 0.4)
print("---After sc.tl.leiden()---")

print("obs columns:", list(adata_all.obs.columns))
print("obsm keys:", list(adata_all.obsm.keys()))
print("obsp keys:", list(adata_all.obsp.keys()))
print("uns keys:", list(adata_all.uns.keys()))

#Plot UMAPs colored by Leiden clustering at different resolutions
sc.pl.umap(
    adata_all,
    color=['leiden_harmony_0.2','leiden_harmony_0.4', 'leiden_harmony_0.8'],
    ncols=3,
    legend_loc='on data',
    frameon=False,
    title=['Leiden 0.2','Leiden 0.4', 'Leiden 0.8']
)


#Plot UMAPs with QC metrics to check quality after integration
sc.pl.umap(
    adata_all,
    color=['total_counts', 'n_genes_by_counts', 'pct_counts_mt', 'pct_counts_ribo']
    color_map='viridis',
    vmax=[15000, 6000, 20, 50],
    ncols=4,
    title=["Total counts", "Genes per cell", "% MT counts", "% Ribosomal counts"]
)

#Plot UMAPs by sample and patient to assess batch effect removal
sc.pl.umap(
    adata_all,
    color=["sample", "patient"],
    palette='Set1',
    title=["Sample (batch)", "Patient"]
)

# Print cluster sizes to help decide best resolution
for res in [0.2,0.4, 0.8]:
    print(f"\nCluster sizes at resolution {res}:")
```

```
    print(adata_all.obs[f'leiden_harmony_{res}'].value_counts())
```

```
2025-04-21 16:48:06,826 - harmonypy - INFO - Computing initial centroids with sklear
n.KMeans...
2025-04-21 16:48:28,376 - harmonypy - INFO - sklearn.KMeans initialization complete.
2025-04-21 16:48:28,745 - harmonypy - INFO - Iteration 1 of 20
2025-04-21 16:48:52,179 - harmonypy - INFO - Iteration 2 of 20
2025-04-21 16:49:16,407 - harmonypy - INFO - Iteration 3 of 20
2025-04-21 16:49:41,150 - harmonypy - INFO - Converged after 3 iterations
---After sc.pp.neighbors()---
obs columns: ['sample', 'patient', 'condition', 'n_genes_by_counts', 'total_counts',
'total_counts_mt', 'pct_counts_mt', 'total_counts_ribo', 'pct_counts_ribo', 'n_gene
s']
obsm keys: ['X_pca', 'X_pca_harmony']
obsp keys: ['distances', 'connectivities']
uns keys: ['log1p', 'hvg', 'pca', 'neighbors']
---After sc.tl.umap()---
obs columns: ['sample', 'patient', 'condition', 'n_genes_by_counts', 'total_counts',
'total_counts_mt', 'pct_counts_mt', 'total_counts_ribo', 'pct_counts_ribo', 'n_gene
s']
obsm keys: ['X_pca', 'X_pca_harmony', 'X_umap']
obsp keys: ['distances', 'connectivities']
uns keys: ['log1p', 'hvg', 'pca', 'neighbors', 'umap']
```

```
/var/folders/y2/h05c6vzs57b04_t_871d62wm0000gn/T/ipykernel_60548/848252827.py:43: Fu
tureWarning: In the future, the default backend for leiden will be igraph instead of
leidenalg.

 To achieve the future defaults please pass: flavor="igraph" and n_iterations=2.  di
rected must also be False to work with igraph's implementation.
  sc.tl.leiden(
---After sc.tl.leiden()---
obs columns: ['sample', 'patient', 'condition', 'n_genes_by_counts', 'total_counts',
'total_counts_mt', 'pct_counts_mt', 'total_counts_ribo', 'pct_counts_ribo', 'n_gene
s', 'leiden_harmony_0.2', 'leiden_harmony_0.4', 'leiden_harmony_0.8']
obsm keys: ['X_pca', 'X_pca_harmony', 'X_umap']
obsp keys: ['distances', 'connectivities']
uns keys: ['log1p', 'hvg', 'pca', 'neighbors', 'umap', 'leiden_harmony_0.2', 'leiden
_harmony_0.4', 'leiden_harmony_0.8']
```




Sample (batch)
UMAP2
UMAP1
P2_cSCC
P2_normal
P3_cSCC
P3_normal
P4_cSCC
P4_normal
P5_cSCC
P5_normal
Patient
UMAP2
UMAP1
P2
P3
P4
P5

```
Cluster sizes at resolution 0.2:
leiden_harmony_0.2
0     6865
1     6579
2     5673
3     2989
4     1943
5     1524
6     1490
7     1154
8      894
9      556
10     434
11     233
12     222
13     195
14     176
Name: count, dtype: int64

Cluster sizes at resolution 0.4:
leiden_harmony_0.4
0     4488
1     4366
2     3747
3     2990
4     2615
5     2207
6     2087
7     1574
8     1510
9     1196
10    1155
11     961
12     782
13     423
14     233
15     222
16     195
17     176
Name: count, dtype: int64

Cluster sizes at resolution 0.8:
leiden_harmony_0.8
0     3561
1     3541
2     2785
3     1958
4     1892
5     1757
6     1584
7     1569
8     1400
9     1271
10    1157
11    1088
12    1016
```

```
13     958
14     821
15     786
16     756
17     508
18     497
19     440
20     408
21     233
22     222
23     213
24     195
25     177
26     134
Name: count, dtype: int64
```

9.Neighbourhood, Clustering, and UMAP Analysis without integration

In [ ]:

```
# Copy the integrated AnnData to create a non-integrated branch
adata_no_int = adata_all.copy()

#Neighborhood graph (raw PCA)
sc.pp.neighbors(
    adata_no_int,
    use_rep='X_pca',
    n_neighbors=15,
    n_pcs=20,
    random_state=0
)

#UMAP embedding (non-integrated)
sc.tl.umap(adata_no_int, random_state=0)

#Leiden clustering at the same three resolutions: 0.2, 0.4, 0.8
for res in [0.2, 0.4, 0.8]:
    sc.tl.leiden(
        adata_no_int,
        resolution=res,
        key_added=f'leiden_ni_{res}',
        random_state=0,
        flavor='igraph'   # optional, to use the future default backend
    )

#Plot UMAPs colored by those Leiden clusters
sc.pl.umap(
    adata_no_int,
    color=['leiden_ni_0.2', 'leiden_ni_0.4', 'leiden_ni_0.8'],
    ncols=3,
    legend_loc='on data',
    frameon=False,
    title=['Leiden 0.2 (no int)', 'Leiden 0.4 (no int)', 'Leiden 0.8 (no int)']
)
```

```
#QC-metric UMAP(non-integrated)
sc.pl.umap(
    adata_no_int,
    color=['total_counts', 'n_genes_by_counts', 'pct_counts_mt', 'pct_counts_ribo']
    color_map='viridis',
    vmax=[15000, 6000, 20, 50],
    ncols=4,
    title=["Total counts", "Genes per cell", "% MT counts", "% Ribosomal counts"]
)

#Batch UMAP (non-integrated)
sc.pl.umap(
    adata_no_int,
    color=["sample", "patient"],
    palette='Set1',
    title=["Sample (no int)", "Patient (no int)"]
)
```

Ans 4). UMAP With and Without Integration Without Integration:

---> UMAP may show dataset-specific clustering, reflecting batch effects rather than true biological variation.

With Integration:

---> Similar cell types from different batches are aligned.

UMAP shows clustering based on shared cell states.

Reason for Difference:

Integration corrects technical variation, allowing clustering and UMAP to reflect biological, not technical, similarity.

Q5) Perform differential expression gene analysis on integrated object and annotate as many clusters as you can. Confirm your cluster annotation with the markers expression by using feature plots.

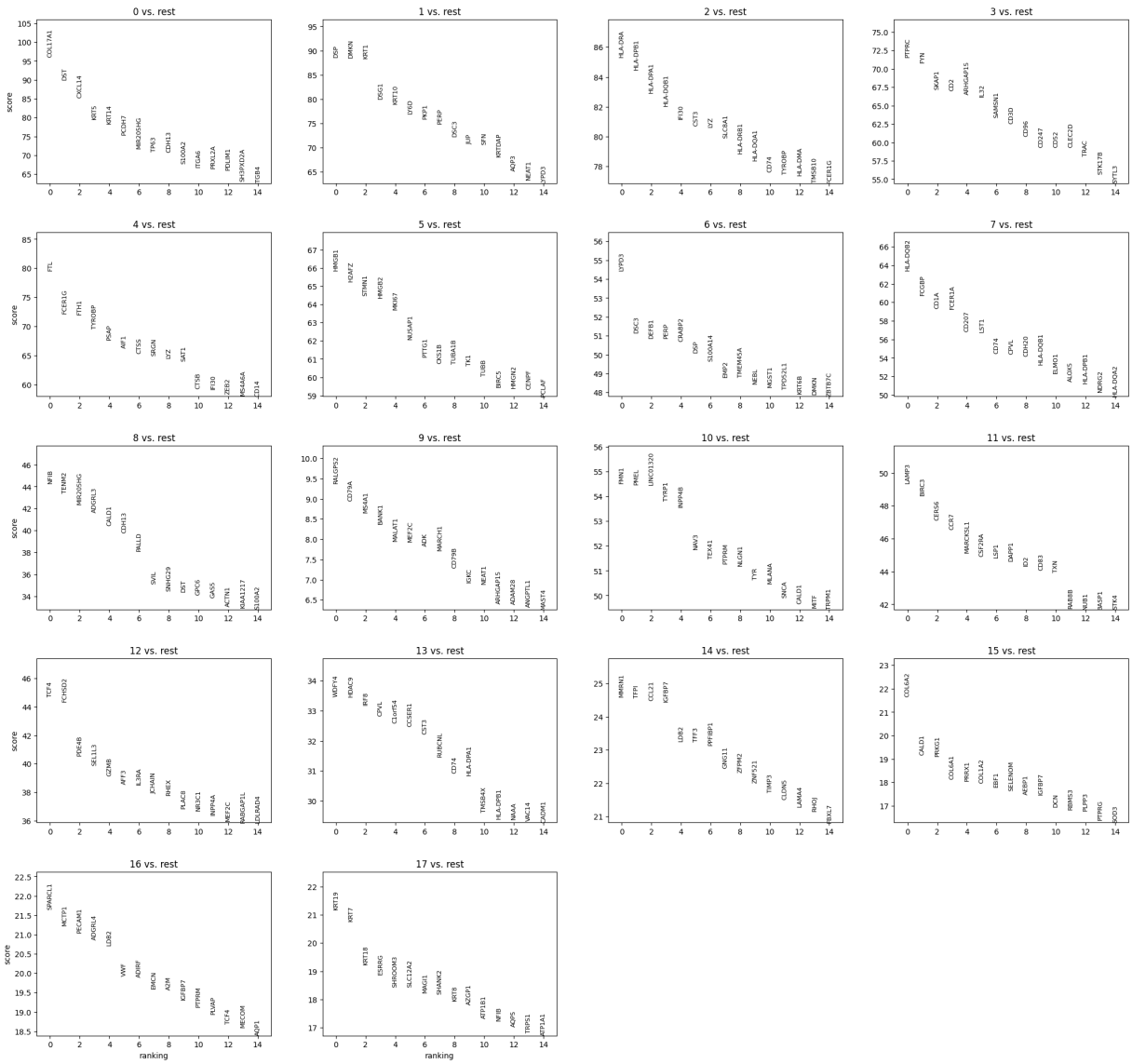
Q6) Finally, compare the cell type differences between normal and cSCC and comment on how these changes can lead to cSCC development? a) Explore immune and non-immune cells and plot the proportion of cells in normal and cSCC. Are there any significant changes in the composition of cell types between normal and cSCC? b) For the immune cell population, describe T cell, B cell and myeloid cell population and their role in cSCC development Hint: use below markers to identify respective cell type

10.Differential Expression & Cluster Annotation

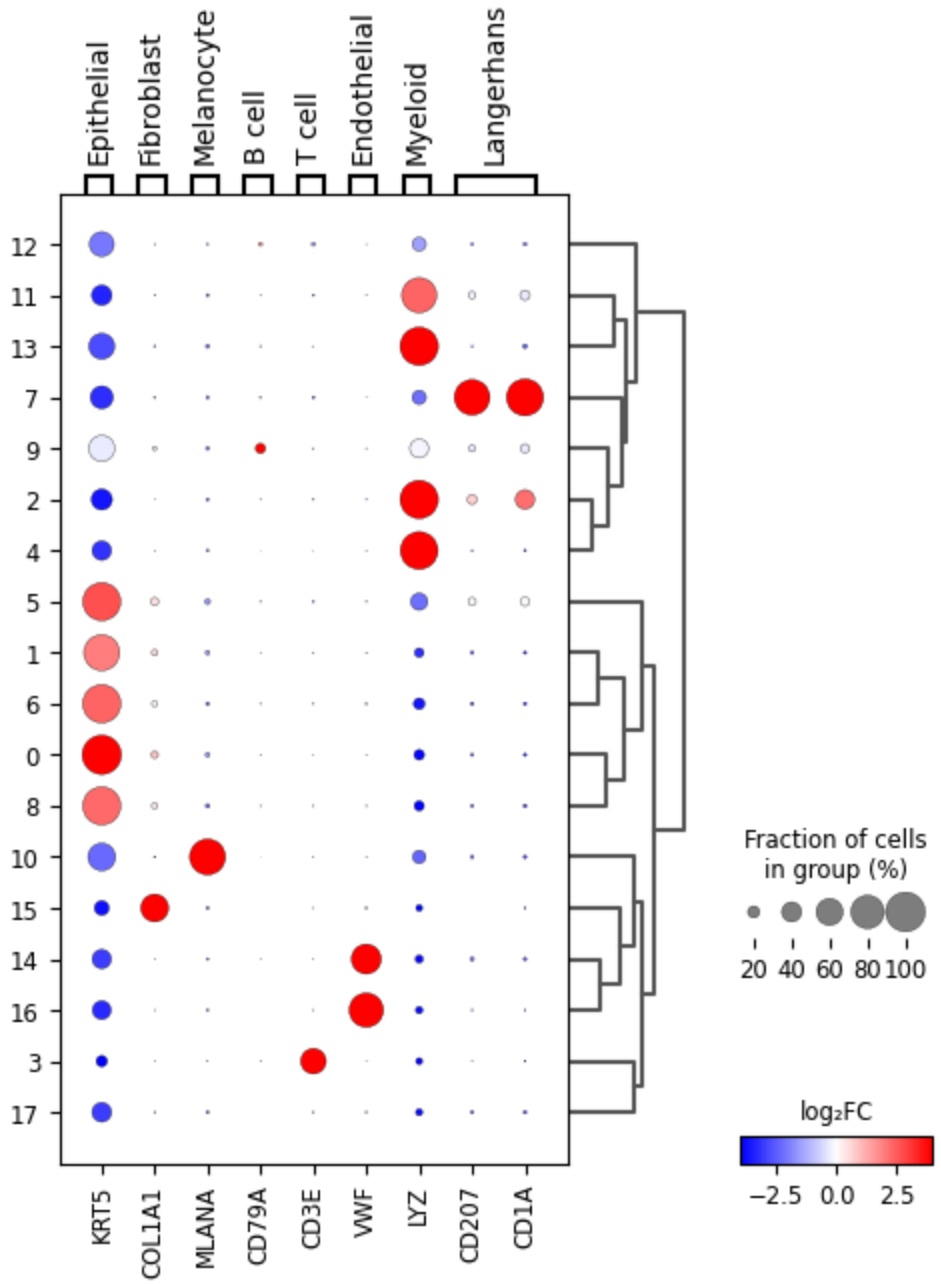
In [89]:
```
import itertools


cluster_key   = 'leiden_harmony_0.4'
pval_thresh   = 0.05
log2fc_thresh = 1
pct_cutoff    = 0.10    # keep only genes in >10% of cells

#Run DE with Wilcoxon & BH correction
sc.tl.rank_genes_groups(
    adata_all,
    groupby=cluster_key,
    reference='rest',
    method='wilcoxon',
    corr_method='benjamini-hochberg',
    key_added='rank_genes_wilcoxon',
    use_raw=True,
    pts=True
)

#Plot top 15 DE genes per cluster
sc.pl.rank_genes_groups(
    adata_all,
    key='rank_genes_wilcoxon',
    n_genes=15,
    sharey=False
)

#Delete old dendrogram so it will be recalculated
dend_key = f'dendrogram_{cluster_key}'
if dend_key in adata_all.uns:
    del adata_all.uns[dend_key]

#Recompute dendrogram for this clustering
sc.tl.dendrogram(adata_all, groupby=cluster_key)

#Dot-plot of specified markers with fresh dendrogram
marker_dict = {
    'Epithelial': ['KRT5'],
    'Fibroblast': ['COL1A1'],
    'Melanocyte': ['MLANA'],
    'B cell': ['CD79A'],
    'T cell': ['CD3E'],
    'Endothelial': ['VWF'],
    'Myeloid': ['LYZ'],
    'Langerhans': ['CD207', 'CD1A']
}
sc.pl.rank_genes_groups_dotplot(
    adata_all,
    groupby=cluster_key,
    key='rank_genes_wilcoxon',
    var_names=marker_dict,
    values_to_plot='logfoldchanges',
    cmap='bwr',
    dendrogram=True,
```

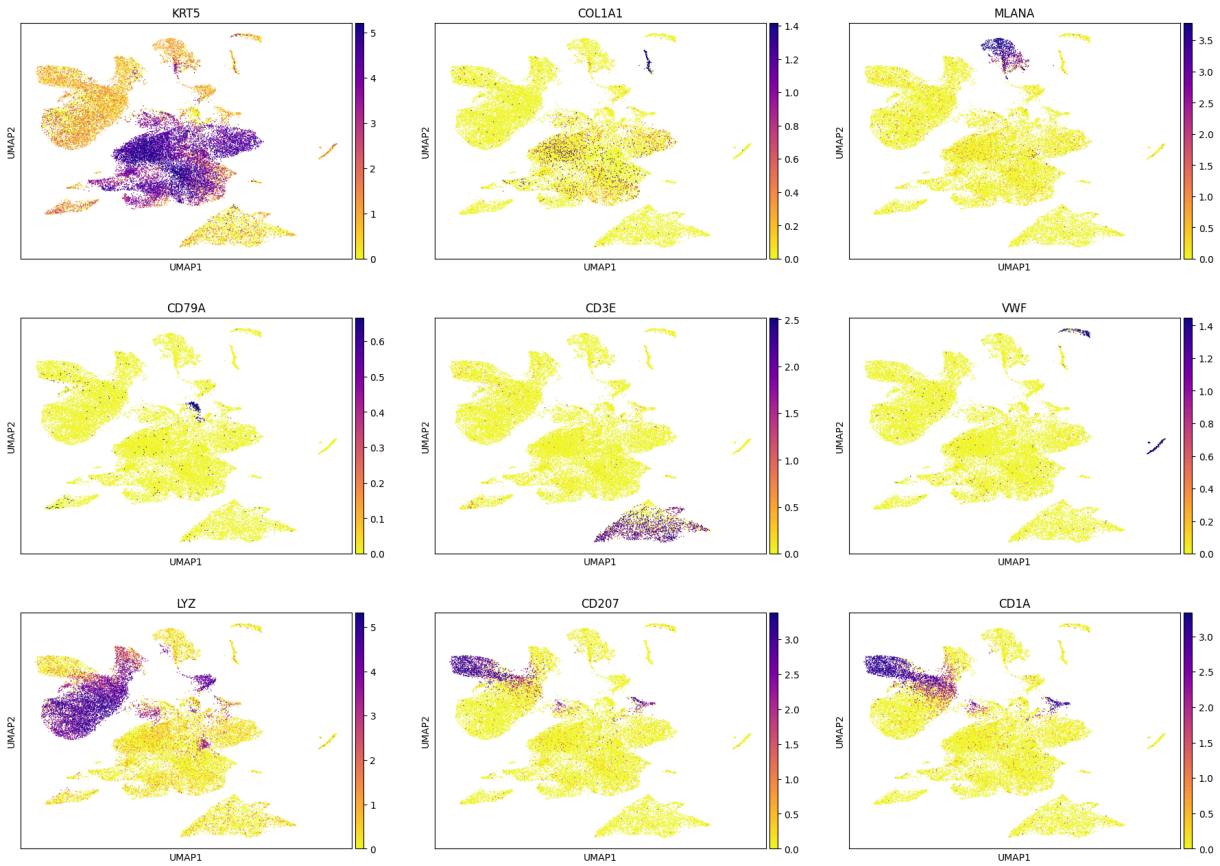
```
    min_logfoldchange=1,
    vmin=-4,
    vmax=4,
    colorbar_title='log₂FC'
)

#Feature-plot just those same markers on UMAP
all_markers = list(itertools.chain(*marker_dict.values()))
sc.pl.umap(
    adata_all,
    color=all_markers,
    use_raw=True,
    vmin=0,
    vmax='p99',
    color_map='plasma_r',
    ncols=3
)

#Collect & filter per-cluster DE tables
cluster_de_genes = {}
for cl in adata_all.obs[cluster_key].cat.categories:
    # pull DE results
    df = sc.get.rank_genes_groups_df(
        adata_all,
        group=cl,
        key='rank_genes_wilcoxon',
        pval_cutoff=pval_thresh,
        log2fc_min=log2fc_thresh
    )
    # filter by percent-expressed
    df = df[df['pct_nz_group'] > pct_cutoff].copy()
    # sort by log-fold change
    df = df.sort_values('logfoldchanges', ascending=False).reset_index(drop=True)
    cluster_de_genes[cl] = df

#Export to Excel: one sheet per cluster
output_file = 'DE_results_skin_clusters.xlsx'
with pd.ExcelWriter(output_file) as writer:
    for cl, df in cluster_de_genes.items():
        df.to_excel(writer, sheet_name=f'cluster_{cl}', index=False)
print(f"Saved per-cluster DE results to {output_file}")

#Annotate clusters to cell_type & plot
cluster2type = {
    '0':'Epithelial','1':'Epithelial','5':'Epithelial','6':'Epithelial','8':'Epithe
    '10':'Melanocyte','15':'Fibroblast','9':'B cell','14':'Endothelial','16':'Endot
    '2':'Myeloid','4':'Myeloid','11':'Myeloid','13':'Myeloid','7':'Langerhans','3':
}
adata_all.obs['cell_type'] = (
    adata_all.obs[cluster_key]
    .astype(str)
    .map(cluster2type)
    .fillna('Other')
    .astype('category')
)
pal = {
```

```
    'Epithelial':    '#1f77b4',
    'Fibroblast':    '#ff7f0e',
    'Melanocyte':    '#2ca02c',
    'B cell':        '#d62728',
    'T cell':        '#17becf',
    'Myeloid':       '#8c564b',
    'Endothelial':   '#e377c2',
    'Langerhans':    '#7f7f7f',
    'Other':         '#c7c7c7'
}
adata_all.uns['cell_type_colors'] = [pal[c] for c in adata_all.obs['cell_type'].cat
sc.pl.umap(
    adata_all,
    color='cell_type',
    legend_loc='on data',
    title='Cluster → Cell Type'
)
```

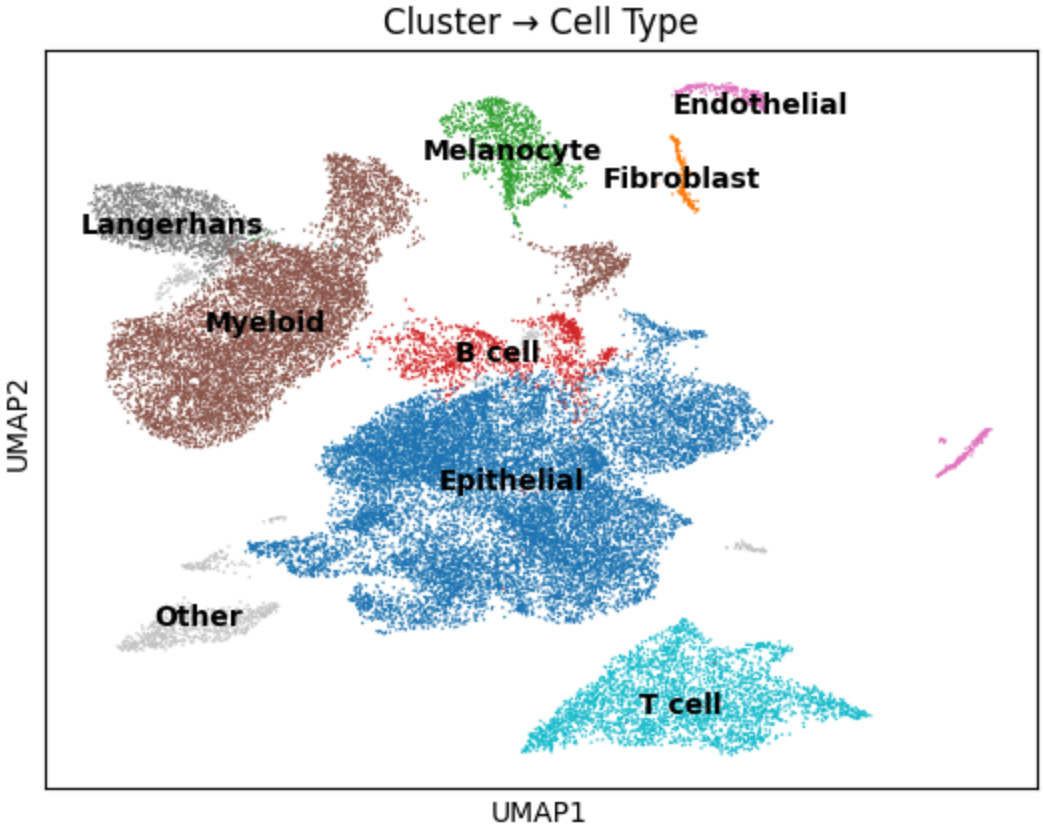



```
WARNING: Groups are not reordered because the `groupby` categories and the `var_grou
p_labels` are different.
categories: 0, 1, 2, etc.
var_group_labels: Epithelial, Fibroblast, Melanocyte, etc.
```


Epithelial
Fibroblast
Melanocyte
B cell
T cell
Endothelial
Myeloid
Langerhans
12
11
13
7
9
2
4
5
1
6
0
8
10
15
14
16
3
17
KRT5
COL1A1
MLANA
CD79A
CD3E
VWF
LYZ
CD207
CD1A
Fraction of cells in group (%)
20 40 60 80 100
log₂FC
−2.5 0.0 2.5




```
Saved per-cluster DE results to DE_results_skin_clusters.xlsx
```



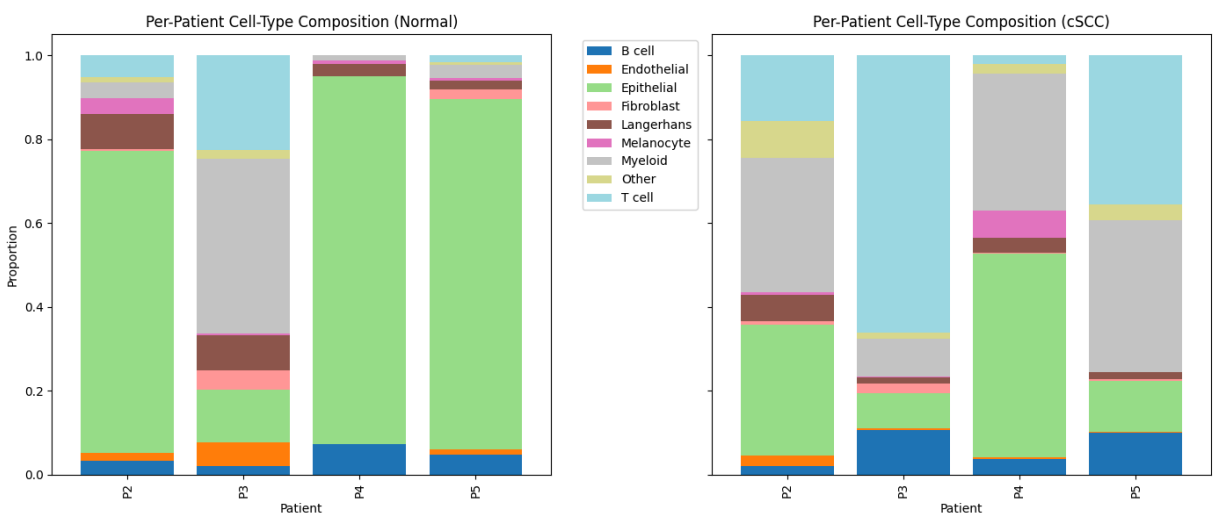
Cell type composition

In [122…

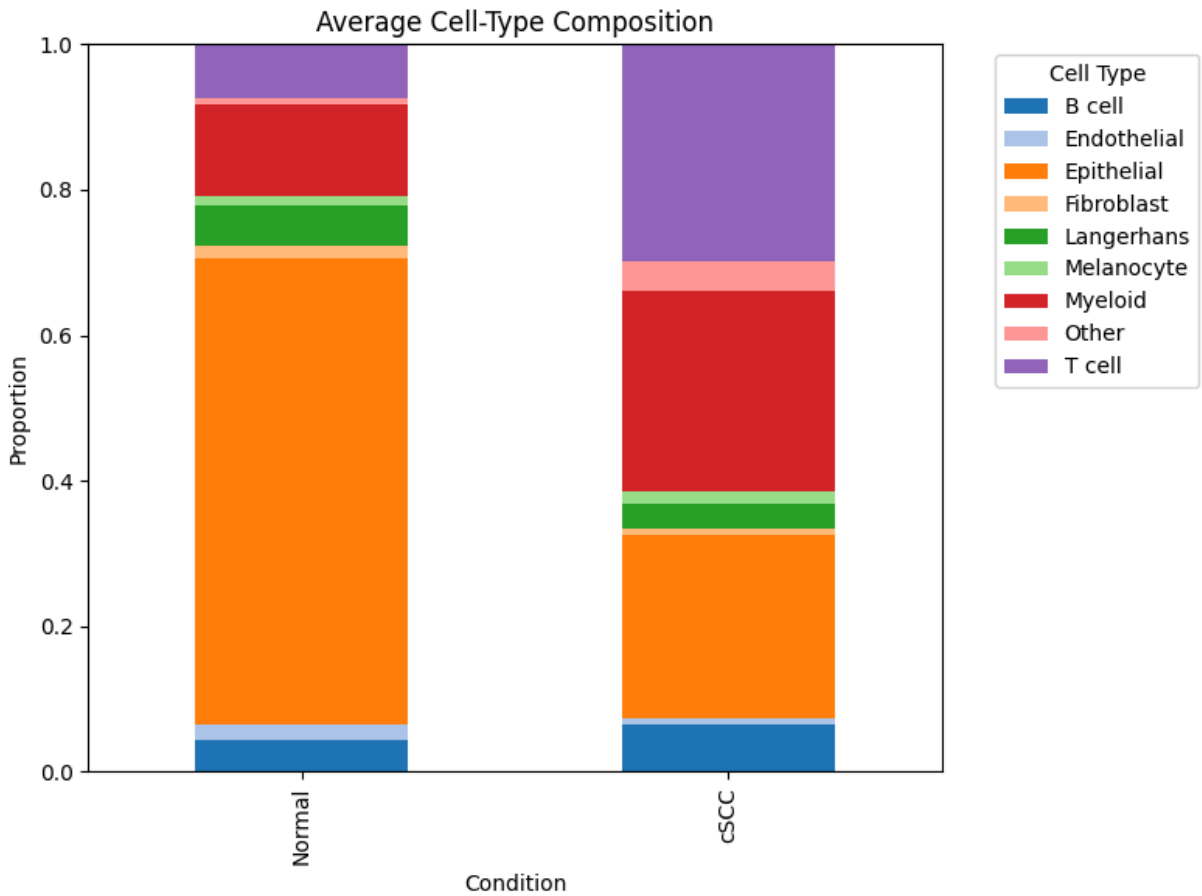
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from scipy.stats import wilcoxon
from statsmodels.stats.multitest import multipletests

# Compute per-patient proportions
def compute_props(adata):
    counts = (
        adata.obs
        .groupby(['patient','cell_type'])
        .size()
        .unstack(fill_value=0)
    )
    return counts.div(counts.sum(axis=1), axis=0)

# Split by condition
adata_norm  = adata_all[adata_all.obs['condition']=='normal']
adata_tumor = adata_all[adata_all.obs['condition']=='cSCC']

# Compute per-patient props
props_norm  = compute_props(adata_norm)
props_tumor = compute_props(adata_tumor)

# 1) Per-patient stacked-bar plots (Normal vs cSCC)
fig, (axn, axt) = plt.subplots(1, 2, figsize=(14,6), sharey=True)
props_norm.plot.bar(
    stacked=True,
    ax=axn,
    colormap='tab20',
    width=0.8
)
axn.set_title('Per-Patient Cell-Type Composition (Normal)')
axn.set_xlabel('Patient')
axn.set_ylabel('Proportion')
axn.legend(bbox_to_anchor=(1.05,1), loc='upper left')

props_tumor.plot.bar(
    stacked=True,
    ax=axt,
    colormap='tab20',
    width=0.8,
    legend=False
)
axt.set_title('Per-Patient Cell-Type Composition (cSCC)')
axt.set_xlabel('Patient')
plt.tight_layout()
plt.show()

# 2) Average composition + set up for annotation
mean_norm  = props_norm.mean(axis=0)
```

```
mean_tumor = props_tumor.mean(axis=0)
mean_df = pd.DataFrame({'Normal': mean_norm, 'cSCC': mean_tumor}).T
fig, ax = plt.subplots(figsize=(8,6))
mean_df.plot(
    kind='bar',
    stacked=True,
    ax=ax,
    color=plt.cm.tab20.colors
)
ax.set_title('Average Cell-Type Composition')
ax.set_ylabel('Proportion')
ax.set_xlabel('Condition')
ax.legend(bbox_to_anchor=(1.05,1), loc='upper left', title='Cell Type')
ax.set_ylim(0, 1)

# 3) Build long DataFrame and run paired Wilcoxon (two-sided)
dfs = []
for cond, props in [('normal', props_norm), ('cSCC', props_tumor)]:
    df = props.reset_index().melt(
        id_vars='patient',
        var_name='cell_type',
        value_name='proportion'
    )
    df['condition'] = cond
    df['condition'] = pd.Categorical(
        df['condition'],
        categories=['normal','cSCC'],
        ordered=True
    )
    dfs.append(df)

df_long = pd.concat(dfs, ignore_index=True)

tests = []
for ct in df_long['cell_type'].unique():
    sub = df_long[df_long['cell_type'] == ct]
    norm_vals = (
        sub.loc[sub.condition=='normal', ['patient','proportion']]
        .set_index('patient')
        .sort_index()
    )
    tum_vals = (
        sub.loc[sub.condition=='cSCC', ['patient','proportion']]
        .set_index('patient')
        .sort_index()
    )
    # two-sided test: any difference
    stat, p_unc = wilcoxon(
        norm_vals.proportion,
        tum_vals.proportion
    )
    med_diff = (norm_vals.proportion - tum_vals.proportion).median()
    tests.append({
        'cell_type':     ct,
        'W_stat':        stat,
        'p_uncorrected': p_unc,
```

```
        'median_diff':   med_diff
    })

results = pd.DataFrame(tests).set_index('cell_type')
results['p_adj'] = multipletests(results['p_uncorrected'], method='fdr_bh')[1]
results['significant'] = results['p_adj'] < 0.05
print("\nPer-cell-type Wilcoxon Results (two-sided):")
print(results.sort_values('p_adj'))

# 4) Annotate average-composition bar plot with significance stars
cell_types = mean_df.columns
for i, ct in enumerate(cell_types):
    p = results.loc[ct, 'p_adj']
    if p < 0.001:
        star = '***'
    elif p < 0.01:
        star = '**'
    elif p < 0.05:
        star = '*'
    else:
        star = ''
    top = max(mean_norm.loc[ct], mean_tumor.loc[ct])
    ax.text(i, top + 0.03, star, ha='center', va='bottom', fontsize=14)

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




```
Per-cell-type Wilcoxon Results (two-sided):
             W_stat  p_uncorrected  median_diff     p_adj  significant
cell_type
Epithelial      0.0           0.125     0.399705  0.562500        False
T cell          0.0           0.125    -0.222079  0.562500        False
B cell          3.0           0.625    -0.019669  0.703125        False
Endothelial     3.0           0.625     0.003879  0.703125        False
Fibroblast      3.0           0.625     0.008005  0.703125        False
Langerhans      2.0           0.375     0.012650  0.703125        False
Myeloid         3.0           0.625    -0.298239  0.703125        False
Other           1.0           0.250    -0.027120  0.703125        False
Melanocyte      4.0           0.875     0.003609  0.875000        False
```



Ans 5) Differential Expression and Cluster Annotation Differential Expression Analysis:

Identifies genes significantly upregulated in each cluster.

---> Tools:rank_genes_groups() in Scanpy.

Annotation:

Clusters are assigned cell identities using:

Known marker genes (e.g., CD3D for T cells, MS4A1 for B cells).

Reference datasets or manual curation.

Feature Plots:

Visualize marker expression on UMAP.

Confirm that expression patterns align with expected biology (e.g., T cell markers enriched in T cell cluster).

Ans 6 a) Explore Immune and Non-Immune Cells – Cell Proportion Analysis ---> Plotting Cell Proportions:

```
a) After annotating cell types using known markers, calculate the
relative proportion of each cell type in normal vs cSCC samples.

b)Use bar plots or stacked bar graphs to visualize immune (e.g., T
cells, B cells, macrophages) and non-immune (e.g., keratinocytes,
fibroblasts) composition.
```

---> Significant Changes Often Observed:

```
a)Increased immune infiltration in cSCC (inflammation and immune
surveillance).

b)Expansion of specific immune cells:

 ---> Tregs or exhausted T cells, which suppress immune responses.

 ---> Myeloid-derived suppressor cells (MDSCs) or tumor-associated
macrophages (TAMs), which can promote tumor progression.

c)Changes in keratinocyte subtypes, with more proliferative or
dysplastic states in cSCC.
```

These compositional shifts reflect a tumor-promoting microenvironment, immune evasion, and stromal remodeling associated with cancer development.

b) Description of Immune Cell Populations in cSCC ---> T Cells Markers: CD3D, CD3E, CD4, CD8A

Roles:

a) CD8+ cytotoxic T cells may attempt to kill tumor cells but are often exhausted in cSCC.

b) CD4+ T helper cells can support or suppress anti-tumor immunity.

c) Regulatory T cells (FOXP3+) often increase in tumors and suppress anti-tumor responses, enabling immune escape.

---> B Cells Markers: MS4A1 (CD20), CD79A, CD19

Roles:

a) Can produce antibodies and present antigens.

b) Their role in cSCC is context-dependent: they may support anti-tumor responses or, in some cases, promote tumor progression via cytokine secretion or immunosuppression.

---> Myeloid Cells Markers: CD14, LYZ, ITGAM (CD11b), CD68, MRC1 (CD206)

Roles:

a) Include monocytes, macrophages, and dendritic cells.

b) In tumors, macrophages often polarize toward an M2-like state, promoting tissue remodeling, angiogenesis, and immune suppression.

--> Myeloid-derived suppressor cells (MDSCs) also contribute to immunosuppression and tumor progression.

Summary of Impact on cSCC Development:-

a) The immune microenvironment in cSCC shifts toward suppression and tolerance, allowing tumor cells to evade immune destruction.

b) Non-immune changes like altered keratinocyte proliferation and stromal remodeling support tumor growth.

These changes collectively facilitate tumor initiation, progression, and metastasis.