Mosaic

Objective: To showcase the minimum number of steps required for tertiary analysis of DNA, CNV and Protein analytes and explore different ways of visualizing the data.

Major questions answered:

1. Can we identify DNA clones based on genotypes (SNVs/Indels)?
2. Do we detect CNV events (e.g., copy number amplification, copy number loss, including LOH)?
3. Can we delineate different cell types from cell surface protein expression data?
4. Do DNA clones (genotypes) correlate with specific protein-defined cell types and/or CNV profiles?

Things not shown:

1. All available methods and options - documented here (https://missionbio.github.io/mosaic/index.html)

Setup

Topics covered

1. Loading required packages and data.
2. Structure and contents of data objects.

Load data

In [1]:
```
# Import mosaic libraries
import missionbio.mosaic as ms

# Import these to display entire dataframes
from IPython.display import display, HTML

# Import graph_objects from the plotly package to display figures when saving the notebook as an HTML
# Import numpy for statistics
import plotly.graph_objects as go
import numpy as np

# Import additional packages for specific visuals
import missionbio.mosaic.utils as mutils
import matplotlib.pyplot as plt

# Import the colors
from missionbio.mosaic.constants import COLORS
import seaborn as sns

# resolve issue of showing up graphs.(import plotly.offline as pyo   pyo.init_notebook_mode() )
# Note: when exporting the notebook as an HTML, plots that use the "go.Figure(fig)" command are saved
```

In [2]:
```
# Check version; this notebook is designed for Mosaic 2.0 or higher
print(ms.__version__)
```

```
2.4.1
```

In [3]:
```
# Any function's parameters and default values can be looked up via the 'help' function
# Here, the function is 'ms.load'
help(ms.load)
```

```
Help on function load in module missionbio.mosaic.io:

load(filepath: Any, apply_filter: bool = True, whitelist: Union[Sequence, NoneType] = None, raw: bool = False, single: bool = F
alse) -> Union[missionbio.mosaic.sample.Sample, missionbio.mosaic.samplegroup.SampleGroup]
    Loading the .h5 file with one or more assays.

    This is the preferred way of loading .h5 files.

    It directly returns a `Sample` object, which
    contains all the assays. Those assays that were
    not present are stored as `None`.

    Parameters
    ----------
    filepath:
        The path to the .h5 multi-omics file.
    apply_filter:
        Whether to load only the filtered dna variants.
    whitelist:
        The specific dna variants to load.
    raw:
        Whether the raw counts are to be loaded.
    single:
        Whether to load as a single sample despite being
        a multi sample h5 file (Not a recommended approach).
        Only use this for debugging issues.

    Returns
    -------
    missionbio.mosaic.sample.Sample / missionbio.mosaic.samplegroup.SampleGroup

    Raises
    ------
    Exception
        When the h5 file format is not supported.
```

In [4]:
```
# Specify the h5 file to be used in this analysis: h5path = '/path/to/h5/file/test.h5'
# If working with Windows, you may need to add an 'r' before the path: h5path = r'/path/to/h5/file/test.h5'
h5path = r'C:\Users\smile\Desktop\UPN517BM_4points_06182023.merged.h5'

# Load the data
sample = ms.load(h5path, raw=False, apply_filter=True, whitelist = ["chr2:25458575:C/T", "chr2:25469502:C/T", "chr17:74732959:G/T

#"DNMT3A C.2597+1 G>A_ chr2:25458575:C/T", "DNMT3A p.L422=_ chr2:25469502:C/T","DNMT3A c.2173+26:C/T_ chr2:25463483:G/A"
#"DNMT3A p.V657=_ chr2:25464542:C/T", "SRSF2 p.P95H_ chr17:74732959:G/T", "RUNX1 p.L98sFS*24_ chr21:36259198:AG/A"

# Always set raw=False; if raw=True, ALL barcodes will be loaded (rather than cell-associated barcodes)
# Always set apply_filter=True unless you can't detect an expected (target) variant. Additional filtering options are included in
# The single=True option loads multi-sample h5 files as a single sample object (compatible with this notebook)
# The whitelist option loads any variant that is in the vcf.gz file (e.g. "chr1:179520511:C/G"); similar to whitelist feature in
```

```
Loading, C:\Users\smile\Desktop\UPN517BM_4points_06182023.merged.h5

C:\Users\smile\anaconda3 2023\envs\mosaic\lib\site-packages\missionbio\mosaic\assay.py:329: ChangeAppliedWarning: Multiple samp
les detected. Appending the sample name to the barcodes to avoid duplicate barcodes. The original barcodes are stored in the or
iginal_barcode row attribute
  warnings.warn(msg, ChangeAppliedWarning)

Loaded in 38.9s.
```

All the interactive plotting functions return a plotly figure. If the layout or the color scheme is not suitable for your data type, they can be changed before creating the final figure.

The colors for the plots are stored either in the individual traces or the layout attributes of the plotly figure.

Mosaic also contains a list of colors that can be used to customize the plots.

In [5]:
```
# Explore the colors
# Additional color palettes: https://seaborn.pydata.org/tutorial/color_palettes.html
# Plot the first few colors
sns.palplot(COLORS[:40])
```

In [6]:
```
help(sns.color_palette)
```

```
Help on function color_palette in module seaborn.palettes:

color_palette(palette=None, n_colors=None, desat=None, as_cmap=False)
    Return a list of colors or continuous colormap defining a palette.

    Possible ``palette`` values include:
        - Name of a seaborn palette (deep, muted, bright, pastel, dark, colorblind)
        - Name of matplotlib colormap
        - 'husl' or 'hls'
        - 'ch:<cubehelix arguments>'
        - 'light:<color>', 'dark:<color>', 'blend:<color>,<color>',
        - A sequence of colors in any format matplotlib accepts

    Calling this function with ``palette=None`` will return the current
    matplotlib color cycle.

    This function can also be used in a ``with`` statement to temporarily
    set the color cycle for a plot or set of plots.

    See the :ref:`tutorial <palette_tutorial>` for more information.

    Parameters
    ----------
    palette : None, string, or sequence, optional
        Name of palette or None to return current palette. If a sequence, input
        colors are used but possibly cycled and desaturated.
    n_colors : int, optional
        Number of colors in the palette. If ``None``, the default will depend
        on how ``palette`` is specified. Named palettes default to 6 colors,
        but grabbing the current palette or passing in a list of colors will
        not change the number of colors unless this is specified. Asking for
        more colors than exist in the palette will cause it to cycle. Ignored
        when ``as_cmap`` is True.
    desat : float, optional
        Proportion to desaturate each color by.
    as_cmap : bool
        If True, return a :class:`matplotlib.colors.ListedColormap`.

    Returns
    -------
    list of RGB tuples or :class:`matplotlib.colors.ListedColormap`

    See Also
    --------
    set_palette : Set the default color cycle for all plots.
    set_color_codes : Reassign color codes like ``"b"``, ``"g"``, etc. to
                      colors from one of the seaborn palettes.

    Examples
    --------

    .. include:: ../docstrings/color_palette.rst
```

In [7]:
```
# Alternatively plot another palette
sns.palplot(sns.color_palette("magma", n_colors=20))
```

In [8]:
```
# Print the corresponding hex codes
print(sns.color_palette("magma", n_colors=20).as_hex())
```

```
['#06051a', '#130d34', '#221150', '#36106b', '#4a1079', '#5f187f', '#721f81', '#842681', '#982d80', '#ab337c', '#c03a76', '#d34
36e', '#e44f64', '#f1605d', '#f8765c', '#fc8e64', '#fea571', '#febb81', '#fed194', '#fde7a9']
```

Data Structure

DNA, CNV, and Protein are sub-classes of the Assay class. The information is stored in four categories, and the user can modify each of those:

```
1. metadata (add_metadata / del_metadata):
    - dictionary containing metrics of the assay

2. row_attrs (add_row_attr / del_row_attr):
    - dictionary which contains 'barcode' as one of the keys
      (where the value is a list of all barcodes)
    - for all other keys, the values must be of the same
      length, i.e. match the number of barcodes
    - this is the attribute where 'label', 'pca',
      and 'umap' values are added

3. col_attrs (add_col_attr / del_col_attr):
    - dictionary which contains 'id' as one of the keys
      (where the value is a list of all ids)
    - for DNA assays, 'ids' are variants; for Protein assays,
      'ids' are antibodies
    - for all other keys, the values must be of the same
      length, i.e. match the number ids
```

In [9]:
```
# Summary of DNA assay
print("\'sample.dna\':", sample.dna, '\n')
print("\'row_attrs\':", "\n\t", list(sample.dna.row_attrs.keys()), '\n')
print("\'col_attrs\':", "\n\t", list(sample.dna.col_attrs.keys()), '\n')
print("\'layers\':", "\n\t", list(sample.dna.layers.keys()), '\n')
print("\'metadata\':", "\n")
for i in list(sample.dna.metadata.keys()):
    print("\t", i, ": ", sample.dna.metadata[i], sep="")
```

```
'sample.dna': dna_variants assay with 7 layers, 17616 rows and 91 columns

'row_attrs':
        ['barcode', 'filtered', 'sample_name', 'original_barcode', 'label']

'col_attrs':
        ['ALT', 'CHROM', 'POS', 'QUAL', 'REF', 'ado_gt_cells', 'ado_rate', 'amplicon', 'filtered', 'id']

'layers':
        ['AF', 'DP', 'FILTER_MASK', 'GQ', 'NGT', 'RGQ', 'AF_MISSING']

'metadata':

       sample_name: [['UPN517BM_preLD']
 ['UPN517BM_C1D28_4thRun']
 ['UPN517BM_C1M12_rerun2']
 ['UPN517BM_C2D28_rerun2']]
       ado_rate: [[0.073 ]
 [0.168 ]
```

In [10]:
```
# Summary of Protein assay
print("\'sample.protein\':", sample.protein, '\n')
print("\'row_attrs\':", "\n\t", list(sample.protein.row_attrs.keys()), '\n')
print("\'col_attrs\':", "\n\t", list(sample.protein.col_attrs.keys()), '\n')
print("\'layers\':", "\n\t", list(sample.protein.layers.keys()), '\n')
print("\'metadata\':", "\n")
for i in list(sample.protein.metadata.keys()):
    print("\t", i, ": ", sample.protein.metadata[i], sep="")
```

```
'sample.protein': protein_read_counts assay with 1 layer, 17616 rows and 45 columns

'row_attrs':
        ['barcode', 'sample_name', 'original_barcode', 'label']

'col_attrs':
        ['antibody_id', 'antibody_sequence', 'id']

'layers':
        ['read_counts']

'metadata':

       sample_name: [['UPN517BM_preLD']
 ['UPN517BM_C1D28_4thRun']
 ['UPN517BM_C1M12_rerun2']
 ['UPN517BM_C2D28_rerun2']]
       n_antibodies: [[45]
 [45]
 [45]]
       n_bases_r1: [[40994584112]
 [29503028897]
 [25513377498]
 [26538673779]]
       n_bases_r1_q30: [[37979067418]
 [27711871897]
 [23035680424]
 [25207237906]]
       n_bases_r2: [[40994584112]
 [29503028897]
 [25513377498]
 [26538673779]]
       n_bases_r2_q30: [[38414548163]
 [27753718074]
 [24380181637]
 [25453275537]]
       n_candidate_barcodes: [[294411]
 [488222]
 [577479]
 [365095]]
       n_cell_barcode_bases: [[13775217193]
 [14783753835]
 [12799918893]
 [12846109639]]
       n_cell_barcode_bases_q30: [[13455386976]
 [14298600972]
 [12088123818]
 [12651247321]]
       n_read_pairs: [[271487312]
 [292109197]
 [252607698]
 [253192335]]
       n_read_pairs_after_candidate_barcode_filtering: [[238492494]
 [252288275]
 [232712088]
 [230199908]]
       n_read_pairs_trimmed: [[271467561]
 [291765612]
 [252566900]
 [253160418]]
       n_read_pairs_valid_ab_barcodes: [[240453672]
 [252475078]
 [233664747]
 [234092227]]
       n_read_pairs_valid_cell_barcodes: [[267718887]
 [284261617]
 [246366726]
 [247633697]]
       panel_name: [['totalseq-d-heme-oncology']
 ['totalseq-d-heme-oncology']
 ['totalseq-d-heme-oncology']
 ['totalseq-d-heme-oncology']]
       pipeline_version: [['2.0.1']
 ['2.0.1']
 ['2.0.1']
 ['2.0.1']]
```

In [11]:
```
# For DNA, ids are variants
# sample.dna.ids() is a shortcut for sample.dna.col_attrs['id']
sample.dna.ids()
```

Out[11]:
```
array(['chr11:32417945:T/C', 'chr11:32417964:A/AAAAGAGTCCAGGGT',
       'chr11:32421725:TG/T', 'chr12:112910880:T/TCATCGCTGTCG',
       'chr12:112926395:AAG/A', 'chr12:112926976:G/C',
       'chr12:25378716:GA/G', 'chr12:25380396:CA/C',
       'chr13:28592705:CG/C', 'chr13:28597571:C/G', 'chr13:28597685:G/T',
       'chr13:28597688:T/TTAA', 'chr13:28597691:TCA/T',
       'chr13:28597694:A/ACT', 'chr13:28597696:C/G', 'chr13:28610183:A/G',
       'chr15:90631917:TC/T', 'chr17:74732959:G/T', 'chr17:7573069:CT/C',
       'chr17:7577219:TA/T', 'chr17:7577241:TC/T', 'chr17:7578115:T/C',
       'chr17:7580028:A/ATTACCTAAGTTACACAATAT', 'chr17:7580091:TTC/T',
       'chr1:115256592:T/A', 'chr1:115256598:T/A', 'chr1:115256600:C/G',
       'chr1:115256601:T/G', 'chr1:115256609:T/G', 'chr1:115256616:T/A',
       'chr1:115256623:G/C', 'chr1:115256669:G/A', 'chr20:31021461:C/T',
       'chr20:31021502:T/C', 'chr20:31022396:G/A', 'chr20:31023642:C/A',
       'chr21:36171690:G/C', 'chr21:44514571:G/A', 'chr2:198267542:AC/A',
       'chr2:25458546:C/T', 'chr2:25458575:C/T', 'chr2:25458741:G/A',
       'chr2:25458741:GC/G', 'chr2:25462116:CA/C', 'chr2:25463322:G/A',
       'chr2:25463325:A/G', 'chr2:25464542:C/T', 'chr2:25467481:CCGT/C',
       'chr2:25468286:CA/C', 'chr2:25469210:T/G', 'chr2:25469502:C/T',
       'chr2:25470604:A/T', 'chr2:25470610:C/CTGAAGACATGA',
       'chr2:25471169:GC/G', 'chr2:25471170:CAAGG/*',
       'chr3:128200821:T/TC', 'chr3:128202725:A/G', 'chr3:128202753:G/A',
       'chr4:106154990:TATAGATAG/T', 'chr4:106155133:A/G',
       'chr4:106155226:A/T', 'chr4:106157302:A/ATTCCCTACTGCT',
       'chr4:106157309:ATC/A', 'chr4:106157993:C/CTAACAATCAGTGAAG',
       'chr4:106193653:TAC/T', 'chr4:106196299:CCAG/C',
       'chr4:106196669:T/C', 'chr4:106196792:T/C', 'chr4:106196951:A/G',
       'chr4:106197505:CA/C', 'chr4:55561606:A/T', 'chr4:55569900:A/T',
       'chr4:55592244:G/A', 'chr4:55599436:T/C', 'chr5:170837513:CTT/C',
       'chr7:148504854:A/AGACTT', 'chr7:148504906:TAA/T',
       'chr7:148504906:TAAAA/T', 'chr7:148506064:A/G',
       'chr7:148506396:A/C', 'chr7:148507557:C/CAGCGTTAATGT',
       'chr7:148507584:G/C', 'chr7:148508833:A/G', 'chr7:148508840:CA/C',
       'chr7:148511250:GT/G', 'chr7:148511251:T/G', 'chr7:148515101:GC/G',
       'chr7:148526923:A/G', 'chr7:148529659:A/C', 'chr7:148543525:A/G',
       'chr7:148543583:G/C'], dtype=object)
```

In [12]:
```
# For Protein, ids are AOCs
# sample.protein.ids() is a shortcut for sample.protein.col_attrs['id']
sample.protein.ids()
```

Out[12]:
```
array(['CD10', 'CD117', 'CD11b', 'CD11c', 'CD123', 'CD13', 'CD138',
       'CD14', 'CD141', 'CD16', 'CD163', 'CD19', 'CD1c', 'CD2', 'CD22',
       'CD25', 'CD3', 'CD30', 'CD303', 'CD304', 'CD33', 'CD34', 'CD38',
       'CD4', 'CD44', 'CD45', 'CD45RA', 'CD45RO', 'CD49d', 'CD5', 'CD56',
       'CD62L', 'CD62P', 'CD64', 'CD69', 'CD7', 'CD71', 'CD8', 'CD83',
       'CD90', 'FcεRIα', 'HLA-DR', 'IgG1', 'IgG2a', 'IgG2b'], dtype=object)
```

DNA Analysis

Topics covered

1. Standard filtering of DNA variants.
2. Subsetting dataset for variants of interest, including whitelisted variants.
3. Addition of annotations to the variants.
4. Manual variant selection. [OPTIONAL]
5. Clustering sub-clones.

Basic filtering

```
There are many options for filtering DNA variants.
Use the help() function to understand the approach listed below.
```

In [ ]:
```
# For additional information visit: https://support.missionbio.com/hc/en-us/articles/360047303654-How-do-the-Advanced-Filters-wor
help(sample.dna.filter_variants)
```

In [13]:
```
# Filter the variants, similar to the "Advanced Filters" in Tapestri Insights v2.2

# One major difference: The filter "REMOVE GENOTYPE IN CELLS WITH ALTERNATE ALLELE FREQ"
# is replaced with the following 3 zygosity-specific filters: vaf_ref, vaf_hom, vaf_het

# In general these additional filters remove additional false-positive variants from the data

# Define dna_vars variable
dna_vars = sample.dna.filter_variants()
dna_vars
```

Out[13]:
```
array(['chr11:32417945:T/C', 'chr12:25378716:GA/G', 'chr13:28610183:A/G',
       'chr17:7578115:T/C', 'chr2:25458546:C/T', 'chr2:25458575:C/T',
       'chr2:25469502:C/T', 'chr2:25471169:GC/G', 'chr4:106196951:A/G',
       'chr4:55599436:T/C', 'chr7:148504854:A/AGACTT',
       'chr7:148506064:A/G', 'chr7:148506396:A/C', 'chr7:148508833:A/G',
       'chr7:148515101:GC/G', 'chr7:148543525:A/G'], dtype='<U23')
```

In [14]:
```
# Check the number of filtered variants. When using the default filters, the number of
# variants is likely smaller compared to the originally loaded variants due to the more
# stringent filtering criteria (e.g., vaf_ref=5, vaf_hom=95, vaf_het=35).
len(dna_vars)
```

Out[14]:
```
16
```

In [ ]:
```
# Adjust filters if needed by overwriting dna_vars
# In this example, we alter the filters to match Tapestri Insights v2.2
dna_vars = sample.dna.filter_variants(
    min_dp=10,
    min_gq=30,
    vaf_ref=100,
    vaf_hom=0,
    vaf_het=20,
    min_prct_cells=50,
    min_mut_prct_cells=1,
)

# List all filtered variants
dna_vars
# differ from initial standard filter_variants below
# (min_dp=10, min_gq=30, vaf_ref=5, vaf_hom=95, vaf_het=35, min_prct_cells=50, min_mut_prct_cells=1)
```

In [15]:
```
# Re-check the number of filtered variants
len(dna_vars)
```

Out[15]:
```
16
```

Subsetting Data for Variants of Interest

First, specify variants of interest using one of the three options below:
1. Use the filtered variants from the above section (dna_vars)
2. Use specific variants of interest (whitelist)
3. Combine 1 & 2: filtered variants plus whitelist

Then, this list (final_vars) is used to reduce the larger data set to only your variants of interest.

In [ ]:
```
## Option 1: filtered variants only (no whitelist)

final_vars=dna_vars
```

In [ ]:
```
## Option 2: whitelisted variants only

# Specify the whitelist; variants may be copy/pasted from Tapestri Insights v2.2,
# but ensure correct nomenclature, ie whitelist = ["chr13:28589657:T/G","chrX:39921424:G/A"]
final_vars = ["chr17:29562734:C/A","chr11:118377211:G/C","chr2:25464542:C/T","chr2:25463322:G/A",
"chr17:7577398:G/T","chr4:106194088:T/C"]
```

In [16]:
```
## Option 3: filtered variants plus whitelisted variants

# Specify the whitelist; variants may be copy/pasted from Tapestri Insights v2.2,
# but ensure correct nomenclature, ie whitelist = ["chr13:28589657:T/G","chrX:39921424:G/A"]
target_variants = ["chr2:25458575:C/T", "chr2:25469502:C/T", "chr17:74732959:G/T",]

#"DNMT3A C.2597+1 G>A_ chr2:25458575:C/T", "DNMT3A p.L422=_ chr2:25469502:C/T","DNMT3A c.2173+26:C/T_ chr2:25463483:G/A"
#"DNMT3A p.V657=_ chr2:25464542:C/T", "SRSF2 p.P95H_ chr17:74732959:G/T", "RUNX1 p.L98sFS*24_ chr21:36259198:AG/A"

# Combine whitelisted and filtered variants
final_vars = list(set(list(dna_vars) + target_variants))
```

In [17]:
```
# Check the length of your final list of variants
len(final_vars)
```

Out[17]: 17

In [18]:
```
# Dimensionality of the original sample.dna dataframe
# First number = number of cells (rows); second number = number of variants (columns)
sample.dna.shape
```

Out[18]: (17616, 91)

In [19]:
```
# Before subsetting, verify that all the chosen variants are in the current sample.dna ids (should return True)
print(set(final_vars).issubset(set(sample.dna.ids())))
```

```
True
```

In [20]:
```
# Subsetting sample.dna (columns) based on reduced variant list. Keeping all cells that passed filtering
sample.dna = sample.dna[sample.dna.barcodes(), final_vars]
```

In [21]:
```
# Check the shape of the final filtered DNA object, i.e. (number of barcodes/cells, number of ids/variants)
sample.dna.shape
```

Out[21]: (17616, 17)

Annotation Addition

In [22]:
```
help(sample.dna.get_annotations)
```

```
Help on method get_annotations in module missionbio.mosaic.dna:

get_annotations(variants: Union[Sequence, NoneType] = None) -> pandas.core.frame.DataFrame method of missionbio.mosaic.dna.Dna
instance
    Annotate DNA variants.

    Returns an annotated string array based on the ids of the assay (of form chrA-B-C-D), where
    A,B,C,D represent the chromosome, position, reference, and alternate variant values of an arbitrary length.

    The new strings have a pathological type (PATH, L.PATH, MISS, NONS), and protein or gene added to the beginning
    of the string. This information is pulled from the MB annotation API.

    Parameters
    ----------
    variants : list
        List of variant ids to get annotation for (e.g. ['chr1:12341234:T/A']).
        By default it will get annotations for all variants from the assay.

    Returns
    -------
    variants : pandas.DataFrame
        The array of strings with pathogenicity and gene/protein appended to the front.
```

In [23]:
```
# Fetch annotations using varsome
# Note: run this on a filtered DNA sample - too many variants (e.g., 100+) are not handled correctly by the method
annotation = sample.dna.get_annotations()

# Store the annotations in the dna assay as a new column attribute
for col, content in annotation.items():
    sample.dna.add_col_attr(col, content.values)
```

In [24]:
```
# Sort the values as desired and display a table of all variants (similar to Tapestri Insights)
ann = annotation.sort_values(by=["DANN", "Coding impact"], ascending=False)

display(HTML(ann.to_html()))
```

Variant ID	Position	Ref allele	Alt allele	Varsome url	Variant type	RefSeq transcript id	Gene	
chr7:148515101:GC/G	148515102	C		https://varsome.com/variant/hg19/chr7:148515102:C:	Deletion	NM_004456.5	EZH2	EZH2:p.R3(
chr12:25378716:GA/G	25378717	A		https://varsome.com/variant/hg19/chr12:25378717:A:	Deletion (homopolymer)	NM_033360.4	KRAS	
chr2:25471169:GC/G	25471170	C		https://varsome.com/variant/hg19/chr2:25471170:C:	Deletion	NM_022552.5	DNMT3A	
chr7:148504854:A/AGACTT	148504855		GACTT	https://varsome.com/variant/hg19/chr7:148504855::GACTT	Insertion	NM_004456.5	EZH2	
chr2:25458575:C/T	25458575	C	T	https://varsome.com/variant/hg19/chr2:25458575:C:T	SNV	NM_022552.5	DNMT3A	
chr17:74732959:G/T	74732959	G	T	https://varsome.com/variant/hg19/chr17:74732959:G:T	SNV	NM_003016.4	SRSF2	SRSF2
chr2:25469502:C/T	25469502	C	T	https://varsome.com/variant/hg19/chr2:25469502:C:T	SNV	NM_022552.5	DNMT3A	DNMT3A
chr7:148506064:A/G	148506064	A	G	https://varsome.com/variant/hg19/chr7:148506064:A:G	SNV	NM_004456.5	EZH2	
chr7:148506396:A/C	148506396	A	C	https://varsome.com/variant/hg19/chr7:148506396:A:C	SNV	NM_004456.5	EZH2	
chr17:7578115:T/C	7578115	T	C	https://varsome.com/variant/hg19/chr17:7578115:T:C	SNV	NM_001276760.2	TP53	
chr7:148543525:A/G	148543525	A	G	https://varsome.com/variant/hg19/chr7:148543525:A:G	SNV	NM_004456.5	EZH2	
chr4:106196951:A/G	106196951	A	G	https://varsome.com/variant/hg19/chr4:106196951:A:G	SNV	NM_001127208.3	TET2	TET2:p
chr7:148508833:A/G	148508833	A	G	https://varsome.com/variant/hg19/chr7:148508833:A:G	SNV	NM_004456.5	EZH2	
chr4:55599436:T/C	55599436	T	C	https://varsome.com/variant/hg19/chr4:55599436:T:C	SNV	NM_000222.3	KIT	
chr2:25458546:C/T	25458546	C	T	https://varsome.com/variant/hg19/chr2:25458546:C:T	SNV	NM_022552.5	DNMT3A	
chr13:28610183:A/G	28610183	A	G	https://varsome.com/variant/hg19/chr13:28610183:A:G	SNV	NM_004119.3	FLT3	
chr11:32417945:T/C	32417945	T	C	https://varsome.com/variant/hg19/chr11:32417945:T:C	SNV	NM_024426.6	WT1	WT1:

In [25]:
```
# The data can also be presented as an interactive table
# There are multiple pages to the table - check top right corner for page number
# Clicking will select a particular row and store it in table.selected_rows
table = ms.Table(ann)
table.draw()
```

Out[25]:

```
FigureWidget({
    'data': [{'hoverinfo': 'none',
              'mode': 'text',
              'name': 'data',
              'text': array(['<b><b>Variant ID</b></b>', '<b>Position</b>', '<b>Ref allele</b>', ...,
                             'rs16754', 'Benign', ''], dtype=object),
              'textfont': {'color': [black, black, black, ..., black, black,
                                     black]},
              'type': 'scatter',
              'uid': '17f4f0a3-b4a4-432c-bde1-623dee9cd8d8',
              'x': array([  7.5,  21. ,  33.5, ..., 196. , 210.5, 224.5]),
              'xaxis': 'x2',
              'y': array([60.1304713 , 60.1304713 , 60.1304713 , ...,  1.74110113,  1.74110113,
                          1.74110113]),
              'yaxis': 'y2'},
             {'hoverinfo': 'none',
              'marker': {'color': 'black', 'line': {'color': 'black', 'width': 2}, 'size': 8, 'symbol': 'arrow-left-open'},
              'mode': 'markers',
              'name': 'Previous',
              'type': 'scatter',
              'uid': '594f276f-a000-46d1-9ac8-54650cfdaa2e',
              'x': [0.915],
              'xaxis': 'x',
              'y': [0.5],
              'yaxis': 'y'},
             {'hoverinfo': 'none',
              'mode': 'text',
              'name': 'PageNumber',
              'text': '1/3',
              'textfont': {'size': 11},
              'textposition': 'middle center',
              'type': 'scatter',
              'uid': 'f9040fd3-ee8a-4d28-9109-9cef20f32677',
              'x': [0.955],
              'xaxis': 'x',
              'y': [0.5],
              'yaxis': 'y'},
             {'hoverinfo': 'none',
              'marker': {'color': 'black', 'line': {'color': 'black', 'width': 2}, 'size': 8, 'symbol': 'arrow-right'},
              'mode': 'markers',
              'name': 'Next',
              'type': 'scatter',
              'uid': 'ba560381-5440-4b16-b4f4-185828888dd1',
              'x': [0.995],
              'xaxis': 'x',
              'y': [0.5],
              'yaxis': 'y'},
             {'hoverinfo': 'none',
              'line': {'color': 'black', 'width': 0.25},
              'mode': 'lines',
              'type': 'scattergl',
              'uid': '6b3858df-3ea7-4cb0-b1e5-0a25772f862b',
              'x': [15, 15, 15, 27, 27, 27, 40, 40, 40, 53, 53, 53, 68, 68, 68,
                    83, 83, 83, 98, 98, 98, 107, 107, 107, 122, 122, 122, 137, 137,
                    137, 152, 152, 152, 167, 167, 167, 182, 182, 182, 189, 189,
                    189, 203, 203, 203, 218, 218, 218, 231, 231, 231, 0, 0, 231, 0,
                    0, 231, 0, 0, 231, 0, 0, 231, 0, 0, 231, 0, 0, 231, 0, 0, 231,
                    0, 0, 231, 0, 0, 231, 0, 0, 231, 0, 0, 231, 0, 0, 231, 0, 0,
                    231, 0, 0, 231, 0, 0, 231, 0, 0, 231, 0, 0, 231, 0, 0, 231, 0],
              'xaxis': 'x2',
              'y': [0, 61.06350428721006, 0, 0, 61.06350428721006, 0, 0,
                    61.06350428721006, 0, 0, 61.06350428721006, 0, 0,
                    61.06350428721006, 0, 0, 61.06350428721006, 0, 0,
                    61.06350428721006, 0, 0, 61.06350428721006, 0, 0,
                    61.06350428721006, 0, 0, 61.06350428721006, 0, 0,
                    61.06350428721006, 0, 0, 61.06350428721006, 0, 0,
                    61.06350428721006, 0, 0, 3.4822022531844965,
                    3.4822022531844965, 3.4822022531844965, 6.964404506368993,
                    6.964404506368993, 6.964404506368993, 10.44660675955349,
                    10.44660675955349, 10.44660675955349, 13.928809012737986,
                    13.928809012737986, 13.928809012737986, 17.411011265922482,
                    17.411011265922482, 17.411011265922482, 20.89321351910698,
                    20.89321351910698, 20.89321351910698, 24.375415772291475,
                    24.375415772291475, 24.375415772291475, 27.857618025475972,
                    27.857618025475972, 27.857618025475972, 31.33982027866047,
                    31.33982027866047, 31.33982027866047, 34.822022531844965,
                    34.822022531844965, 34.822022531844965, 38.30422478502946,
                    38.30422478502946, 38.30422478502946, 41.78642703821396,
                    41.78642703821396, 41.78642703821396, 45.268629291398454,
                    45.268629291398454, 45.268629291398454, 48.75083154458295,
                    48.75083154458295, 48.75083154458295, 52.23303379776745,
                    52.23303379776745, 52.23303379776745, 55.715236050951944,
                    55.715236050951944, 55.715236050951944, 59.19743830413644,
                    59.19743830413644, 59.19743830413644, 61.06350428721006,
```

```
                    61.06350428721006, 61.06350428721006],
                'yaxis': 'y2'}],
     'layout': {'font': {'color': '#7f7f7f', 'family': 'Courier New, monospace', 'size': 14},
                'height': 2011.9051286163017,
                'hoverdistance': -1,
                'showlegend': False,
                'template': '...',
                'width': 1000,
                'xaxis': {'anchor': 'y', 'domain': [0.0, 1.0], 'fixedrange': True, 'range': [0, 1], 'visible': False},
                'xaxis2': {'anchor': 'y2',
                           'domain': [0.0, 1.0],
                           'fixedrange': True,
                           'range': [-0.1, 83.1],
                           'visible': False},
                'yaxis': {'anchor': 'x',
                          'domain': [0.9838874712041344, 1.0],
                          'fixedrange': True,
                          'range': [0, 1],
                          'visible': False},
                'yaxis2': {'anchor': 'x2',
                           'domain': [0.0, 0.9838874712041344],
                           'fixedrange': True,
                           'range': [-0.1, 61.16350428721006],
                           'visible': False}}
})
```

In [ ]:
```
# Check the selected variants
# If desired, this list can be used to subset the data further (similar to the subsetting done above with "final_vars")
table.selected_rows
```

In [26]:
```
# Any of these column values can be added to the id names
sample.dna.col_attrs.keys()
```

Out[26]:
```
dict_keys(['ALT', 'CHROM', 'POS', 'QUAL', 'REF', 'ado_gt_cells', 'ado_rate', 'amplicon', 'filtered', 'id', 'annotated Positio
n', 'annotated Ref allele', 'annotated Alt allele', 'annotated Varsome url', 'annotated Variant type', 'annotated RefSeq transc
ript id', 'annotated Gene', 'annotated Protein', 'annotated cDNA', 'annotated Coding impact', 'annotated Function', 'annotated
Allele Freq (gnomAD)', 'annotated DANN', 'annotated dbSNP rsids', 'annotated ClinVar', 'annotated COSMIC ids', 'Position', 'Ref
allele', 'Alt allele', 'Varsome url', 'Variant type', 'RefSeq transcript id', 'Gene', 'Protein', 'cDNA', 'Coding impact', 'Func
tion', 'Allele Freq (gnomAD)', 'DANN', 'dbSNP rsids', 'ClinVar', 'COSMIC ids'])
```

In [27]:
```
# Add annotation to the id names
sample.dna.set_ids_from_cols(["Gene","CHROM", "cDNA","Protein",])

# Annotations are now added to the variants
sample.dna.ids()

# Use sample.dna.reset_ids() to get the original ids
```

Out[27]:
```
array(['WT1:11:c.1122A>G:WT1:p.R374=', 'KRAS:12:c.291-10del:',
       'FLT3:13:c.1310-3T>C:', 'SRSF2:17:c.284C>A:SRSF2:p.P95H',
       'TP53:17:c.555+62A>G:', 'DNMT3A:2:c.2597+30G>A:',
       'DNMT3A:2:c.2597+1G>A:', 'DNMT3A:2:c.1266G>A:DNMT3A:p.L422=',
       'DNMT3A:2:c.640-49del:', 'TET2:4:c.5284A>G:TET2:p.I1762V',
       'KIT:4:c.2484+78T>C:', 'EZH2:7:c.2196-61_2196-57dup:',
       'EZH2:7:c.2195+99T>C:', 'EZH2:7:c.2110+6T>G:',
       'EZH2:7:c.1852-21T>C:', 'EZH2:7:c.1107del:EZH2:p.R369Sfs*55',
       'EZH2:7:c.246+37T>C:'], dtype=object)
```

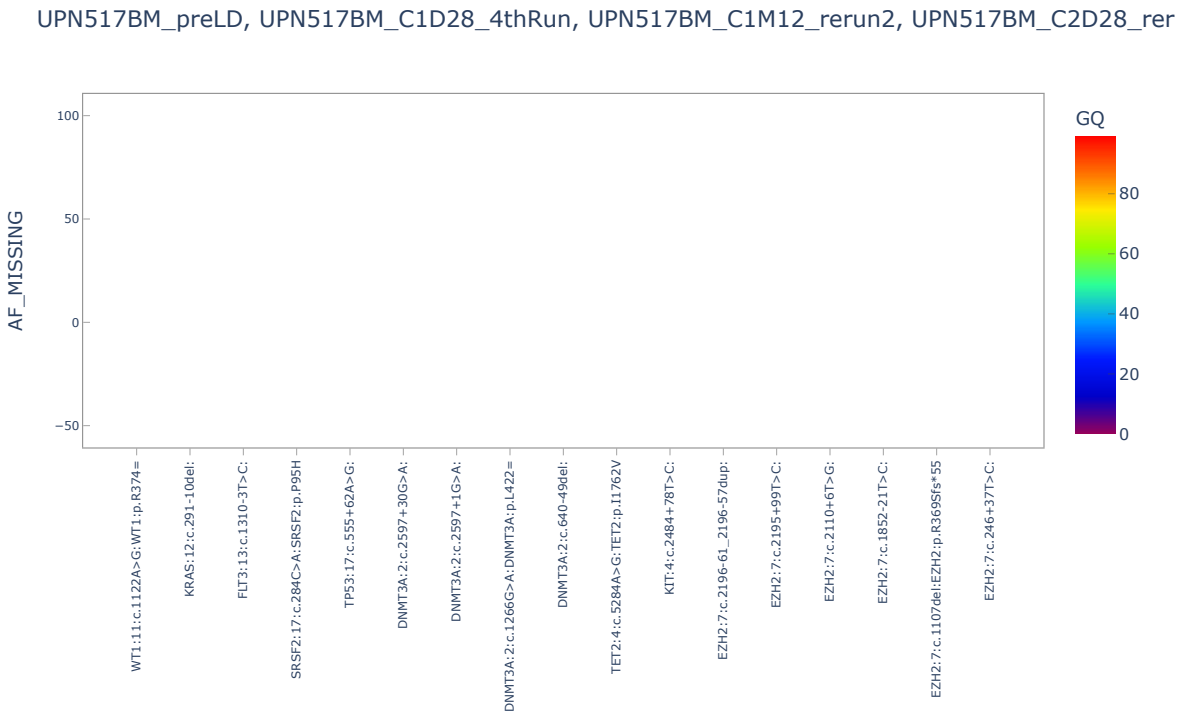
Manual Variant Selection

If using only whitelisted variants (e.g., target variants are already known), this section can be skipped.

This section includes a variety of plots to help assess variant quality.
Heatmaps are interactive. Clicking on a column selects the corresponding id,
whose value is stored in the `selected_ids` attribute. Selected ids can then
be removed from the data set. I.e., click on the low quality variants you wish to discard.

In [ ]:
```
help(sample.dna.stripplot)
```

In [28]:
```
# First diagnostic plot to evaluate variant quality
# The 'attribute' or 'colorby' arguments may be changed
# Variants with the same genotype for every cell are likley germline
# Variants with a HET population distributed below AF=35 are likely false positives (expect HET cells around AF=50)
fig = sample.dna.stripplot(attribute='AF_MISSING', colorby='GQ')
go.Figure(fig)
```



In [29]:
```
# Second diagnostic plot: use this heatmap to (de)-select variants with little variance
# For example, a germline mutation with identical genotype across all cells
# Additionally, variants with excessive missing data can be selected for removal
# Variants may be selected by clicking on the heatmap (variant name changes color from black to red)
# Don't use "fig =" as the heatmap won't become responsive for variant selection
sample.dna.heatmap(attribute='NGT_FILTERED')
```

Out[29]:

```
FigureWidget({
    'data': [{'colorscale': [[0.0, '#1f77b4'], [1.0, '#1f77b4']],
              'customdata': array(['-', '-', '-', ..., '-', '-', '-'], dtype=object),
              'hovertemplate': 'label: %{customdata}<extra></extra>',
              'showlegend': False,
              'showscale': False,
              'type': 'heatmap',
              'uid': 'ace98ed6-c335-4c97-95bc-2fa5e205713d',
              'x': [0, 0, 0, ..., 0, 0, 0],
              'xaxis': 'x',
              'y': array([    0,     1,     2, ..., 17613, 17614, 17615]),
              'yaxis': 'y',
              'z': array([1, 1, 1, ..., 1, 1, 1], dtype=int64)},
             {'coloraxis': 'coloraxis',
              'customdata': array([['-', '-', '-', ..., '-', '-', '-'],
                                   ['-', '-', '-', ..., '-', '-', '-'],
                                   ['-', '-', '-', ..., '-', '-', '-'],
                                   ...,
                                   ['-', '-', '-', ..., '-', '-', '-'],
                                   ['-', '-', '-', ..., '-', '-', '-'],
                                   ['-', '-', '-', ..., '-', '-', '-']], dtype=object),
              'hovertemplate': '%{z:.2f}<br>%{x}<extra>%{customdata}</extra>',
              'showlegend': False,
              'showscale': False,
              'type': 'heatmap',
              'uid': '6aeec360-3758-4bfb-9c3e-684b8e3919b6',
              'x': array(['EZH2:7:c.246+37T>C:', 'EZH2:7:c.1852-21T>C:', 'TP53:17:c.555+62A>G:',
                          'EZH2:7:c.2196-61_2196-57dup:', 'EZH2:7:c.2195+99T>C:',
                          'FLT3:13:c.1310-3T>C:', 'SRSF2:17:c.284C>A:SRSF2:p.P95H',
                          'KIT:4:c.2484+78T>C:', 'TET2:4:c.5284A>G:TET2:p.I1762V',
                          'DNMT3A:2:c.2597+30G>A:', 'EZH2:7:c.2110+6T>G:',
                          'WT1:11:c.1122A>G:WT1:p.R374=', 'DNMT3A:2:c.2597+1G>A:',
                          'DNMT3A:2:c.1266G>A:DNMT3A:p.L422=',
                          'EZH2:7:c.1107del:EZH2:p.R369Sfs*55', 'KRAS:12:c.291-10del:',
                          'DNMT3A:2:c.640-49del:'], dtype=object),
              'xaxis': 'x2',
              'y': array([    0,     1,     2, ..., 17613, 17614, 17615]),
              'yaxis': 'y2',
              'z': array([[2., 3., 3., ..., 3., 0., 3.],
                          [2., 3., 3., ..., 3., 0., 3.],
                          [2., 2., 2., ..., 3., 0., 3.],
                          ...,
                          [2., 2., 2., ..., 3., 0., 3.],
                          [2., 2., 2., ..., 3., 3., 3.],
                          [2., 2., 2., ..., 3., 3., 3.]])}],
    'layout': {'annotations': [{'align': 'left',
                                'font': {'size': 10},
                                'showarrow': False,
                                'text': '17616 cells, 17 features',
                                'x': -0.5,
                                'xanchor': 'left',
                                'xref': 'x2',
                                'y': 17616,
                                'yanchor': 'bottom',
                                'yref': 'y2'}],
               'coloraxis': {'cauto': False,
                             'cmax': 3.0,
                             'cmin': 0.0,
                             'colorbar': {'len': 0.3,
                                          'thickness': 15,
                                          'ticks': 'outside',
                                          'ticktext': [WT, HET, HOM, Missing],
                                          'tickvals': [0.375, 1.125, 1.875, 2.625],
                                          'title': {'text': 'Genotype'},
                                          'y': 0,
                                          'yanchor': 'bottom'},
                             'colorscale': [[0.0, '#3b4d73'], [0.25, '#3b4d73'],
                                            [0.25, '#78a3bc'], [0.5, '#78a3bc'],
                                            [0.5, '#d7ecee'], [0.75, '#d7ecee'],
                                            [0.75, '#000000'], [1.0, '#000000']]},
               'height': 800,
               'legend': {'tracegroupgap': 0},
               'template': '...',
               'title': {'text': 'UPN517BM_preLD, UPN517BM_C1D28_4thRun, UPN517BM_C1M12_rerun2, UPN517BM_C2D28_rerun2'},
               'width': 800,
               'xaxis': {'anchor': 'y',
                         'domain': [0.0, 0.0396],
                         'showticklabels': False,
                         'ticktext': array(['EZH2:7:c.246+37T>C:', 'EZH2:7:c.1852-21T>C:', 'TP53:17:c.555+62A>G:',
                                            'EZH2:7:c.2196-61_2196-57dup:', 'EZH2:7:c.2195+99T>C:',
                                            'FLT3:13:c.1310-3T>C:', 'SRSF2:17:c.284C>A:SRSF2:p.P95H',
                                            'KIT:4:c.2484+78T>C:', 'TET2:4:c.5284A>G:TET2:p.I1762V',
                                            'DNMT3A:2:c.2597+30G>A:', 'EZH2:7:c.2110+6T>G:',
                                            'WT1:11:c.1122A>G:WT1:p.R374=', 'DNMT3A:2:c.2597+1G>A:',
                                            'DNMT3A:2:c.1266G>A:DNMT3A:p.L422=',
                                            'EZH2:7:c.1107del:EZH2:p.R369Sfs*55', 'KRAS:12:c.291-10del:',
```

```
                                        'DNMT3A:2:c.640-49del:'], dtype=object)},
                'xaxis2': {'anchor': 'y2',
                           'domain': [0.049600000000000005, 0.9999999999999999],
                           'tickangle': -90,
                           'tickmode': 'array',
                           'ticktext': array(['EZH2:7:c.246+37T>C:', 'EZH2:7:c.1852-21T>C:', 'TP53:17:c.555+62A>G:',
                                              'EZH2:7:c.2196-61_2196-57dup:', 'EZH2:7:c.2195+99T>C:',
                                              'FLT3:13:c.1310-3T>C:', 'SRSF2:17:c.284C>A:SRSF2:p.P95H',
                                              'KIT:4:c.2484+78T>C:', 'TET2:4:c.5284A>G:TET2:p.I1762V',
                                              'DNMT3A:2:c.2597+30G>A:', 'EZH2:7:c.2110+6T>G:',
                                              'WT1:11:c.1122A>G:WT1:p.R374=', 'DNMT3A:2:c.2597+1G>A:',
                                              'DNMT3A:2:c.1266G>A:DNMT3A:p.L422=',
                                              'EZH2:7:c.1107del:EZH2:p.R369Sfs*55', 'KRAS:12:c.291-10del:',
                                              'DNMT3A:2:c.640-49del:'], dtype=object),
                           'tickvals': array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16])},
                'yaxis': {'anchor': 'x',
                          'domain': [0.0, 1.0],
                          'range': [0, 17616],
                          'tickformat': '{^text}',
                          'ticktext': [<b>-</b> 100.0%],
                          'tickvals': [8808],
                          'type': 'category'},
                'yaxis2': {'anchor': 'x2',
                           'domain': [0.0, 1.0],
                           'matches': 'y',
                           'range': [0, 17616],
                           'showticklabels': False,
                           'type': 'category'}}
})
```

In [30]:
```
# Array that lists all variants that were selected in the heatmap
sample.dna.selected_ids
```

Out[30]:
```
array(['EZH2:7:c.246+37T>C:', 'EZH2:7:c.1852-21T>C:',
       'TP53:17:c.555+62A>G:', 'EZH2:7:c.2196-61_2196-57dup:',
       'EZH2:7:c.2195+99T>C:', 'FLT3:13:c.1310-3T>C:',
       'KIT:4:c.2484+78T>C:', 'TET2:4:c.5284A>G:TET2:p.I1762V'],
      dtype='<U32')
```

In [31]:
```
# Subset the sample.dna variable to removing all selected variants
# Note: if no variants are selected for removal, running this will produce an error (ignore)
sample.dna = sample.dna.drop(sample.dna.selected_ids)
```

In [32]:
```
# Redraw the heatmap with GQ values to select variants that display low quality in majority of cells
sample.dna.heatmap(attribute='GQ')
```

Out[32]:
```
FigureWidget({
    'data': [{'colorscale': [[0.0, '#1f77b4'], [1.0, '#1f77b4']],
              'customdata': array(['-', '-', '-', ..., '-', '-', '-'], dtype=object),
              'hovertemplate': 'label: %{customdata}<extra></extra>',
              'showlegend': False,
              'showscale': False,
              'type': 'heatmap',
              'uid': '0b29f070-9503-42c7-87c5-6e1ade67be65',
              'x': [0, 0, 0, ..., 0, 0, 0],
              'xaxis': 'x',
              'y': array([    0,     1,     2, ..., 17613, 17614, 17615]),
              'yaxis': 'y',
              'z': array([1, 1, 1, ..., 1, 1, 1], dtype=int64)},
             {'coloraxis': 'coloraxis',
              'customdata': array([['-', '-', '-', ..., '-', '-', '-'],
                                   ['-', '-', '-', ..., '-', '-', '-'],
                                   ['-', '-', '-', ..., '-', '-', '-'],
                                   ...,
                                   ['-', '-', '-', ..., '-', '-', '-'],
```

In [ ]:
```
# Array that lists all variants that were selected in the heatmap
sample.dna.selected_ids
```

In [ ]:
```
# Subset the sample.dna variable to removing all selected variants
# Note: if no variants are selected for removal, running this will produce an error (ignore)
sample.dna = sample.dna.drop(sample.dna.selected_ids)
```

In [33]:
```
# Confirm new number of columns (variants)
sample.dna.shape
```

Out[33]:
```
(17616, 9)
```

In [34]:
```
# Redraw heatmap with undesired variants removed
sample.dna.heatmap(attribute='AF_MISSING')
```


```
                        'DNMT3A:2:c.640-49del:', 'DNMT3A:2:c.2597+30G>A:',
                        'SRSF2:17:c.284C>A:SRSF2:p.P95H'], dtype=object),
             'xaxis': 'x2',
             'y': array([    0,     1,     2, ..., 17613, 17614, 17615]),
             'yaxis': 'y2',
             'z': array([[ 18.75      , 100.        , 100.        , ...,   0.        ,
                            4.87804878,   6.66666667],
                         [ 21.05263158,   7.69230769,  66.66666667, ...,  16.66666667,
                           14.28571429,   0.        ],
                         [ 28.57142857,  52.08333333,  47.82608696, ...,   0.        ,
                            0.        , -50.        ],
                         ...,
                         [  0.        ,   1.69491525, -50.        , ..., 100.        ,
                          -50.        , -50.        ],
                         [  0.        ,   0.        , -50.        , ...,  98.03921569,
                          -50.        , -50.        ],
                         [ 12.5       ,   1.75438596, -50.        , ..., -50.        ,
                          -50.        , 100.        ]])}],
 'layout': {'annotations': [{'align': 'left',
                             'font': {'size': 10}
```


Clustering

The DNA assay class comes with three different methods of clustering:
Method 1: f(x) = group_by_genotype (akin to Tapestri Insights v2.2)
Method 2: PCA + UMAP + clustering (customizable)

CLUSTERING METHOD 1

In [ ]:
```
help(sample.dna.group_by_genotype)
```

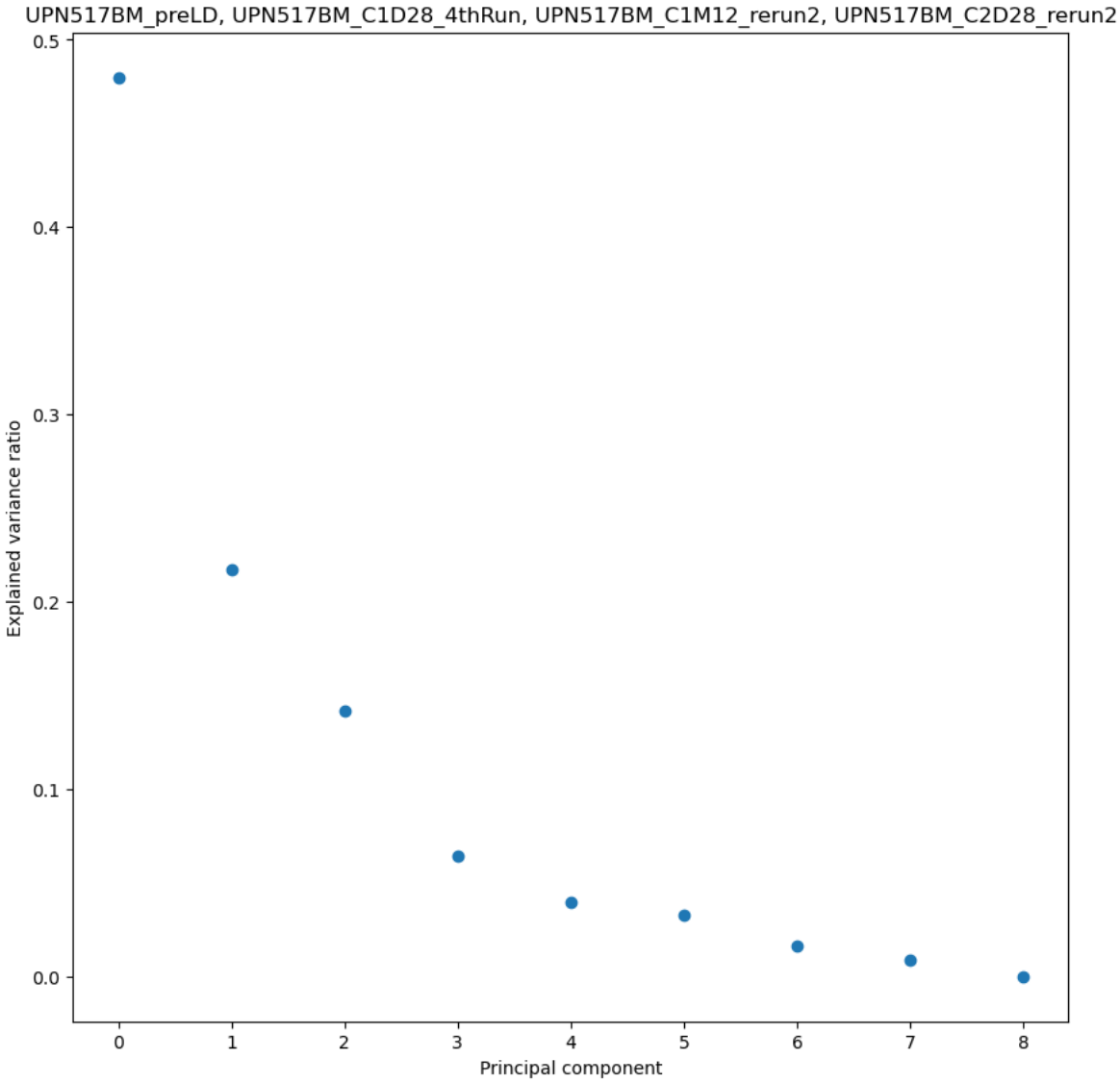
In [ ]:
```
# Clustering Method #1

# Recommended for 1-10 variants
# Cluster with Tapestri Insights v2.2 count-based method

# The created table includes a 'score' row that intends to help with the identification of allelic dropout (ADO) clones
# Scores greater than 0.8 are typically considered artifacts (ADO) and may be labeled as such to be discarded
# Clones are ordered based on their size (1=largest clone, 2=second largest clone, etc.)

clone_data = sample.dna.group_by_genotype(features=sample.dna.ids(), group_missing=True, min_clone_size=0.05, layer="NGT_FILTERED

# Display as a static html or interactive table using ms.Table.
display(HTML(clone_data.to_html()))
```

In [ ]:
```
help(sample.dna.heatmap)
```

In [ ]:
```
# Plot heatmap using NGT_FILTERED.
fig = sample.dna.heatmap(attribute='NGT_FILTERED')
go.Figure(fig)
# NGT_FILTERED, NGT, AF, AF_MISSING
```

CLUSTERING METHOD 2 (PCA + UMAP + Clustering)

In [36]:
```
# CLUSTERING METHOD #2
# An alternative clustering recommended for high-feature/dimensional space (e.g. 10+ variants)

# First: run PCA on filtered data frame (all cells, chosen variants) using the AF_MISSING layer, for instance
# Note: AF_MISSING includes -50 values for all 0 values for which the data was filtered and is missing
# The number of components should equal the number of pre-filtered variants
sample.dna.run_pca(components=91, attribute='AF_MISSING',show_plot=True)

# Assess 'elbow' plot and determine number of PCs where the distribution starts to plateau
# Note that PC1 = 0 in the plot
```



In [47]:
```
# Rerun PCA with optimal number of components based on elbow plot analysis
sample.dna.run_pca(components=8, attribute='AF_MISSING')
```

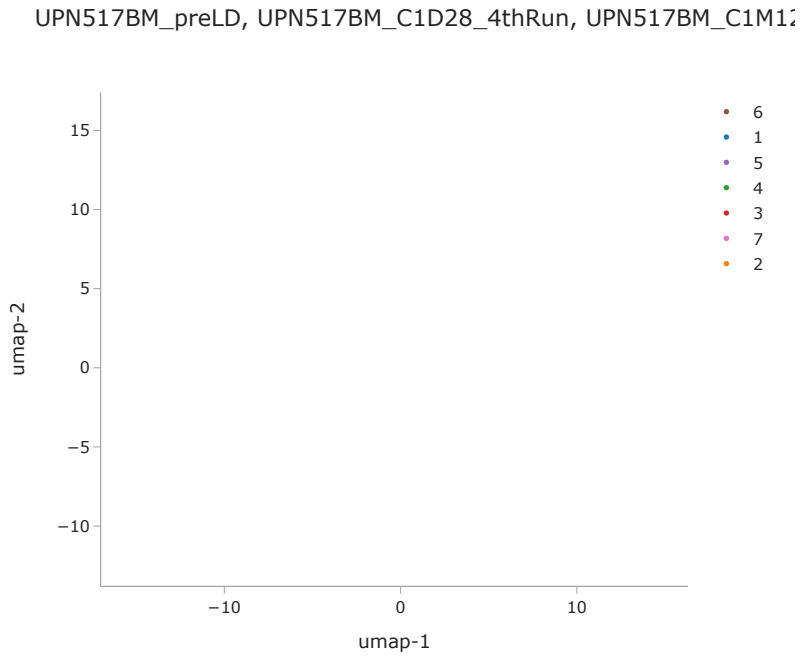
In [48]:
```
# Run UMAP on top of the newly created PC dataframe
# See https://jlmelville.github.io/uwot/abparams.html for appropriate spread and min_dist values
sample.dna.run_umap(attribute='pca', random_state=42) #, min_dist=0.2, spread=1.5)
```

In [ ]:
```
# Review other clustering methods
help(sample.dna.cluster)
```

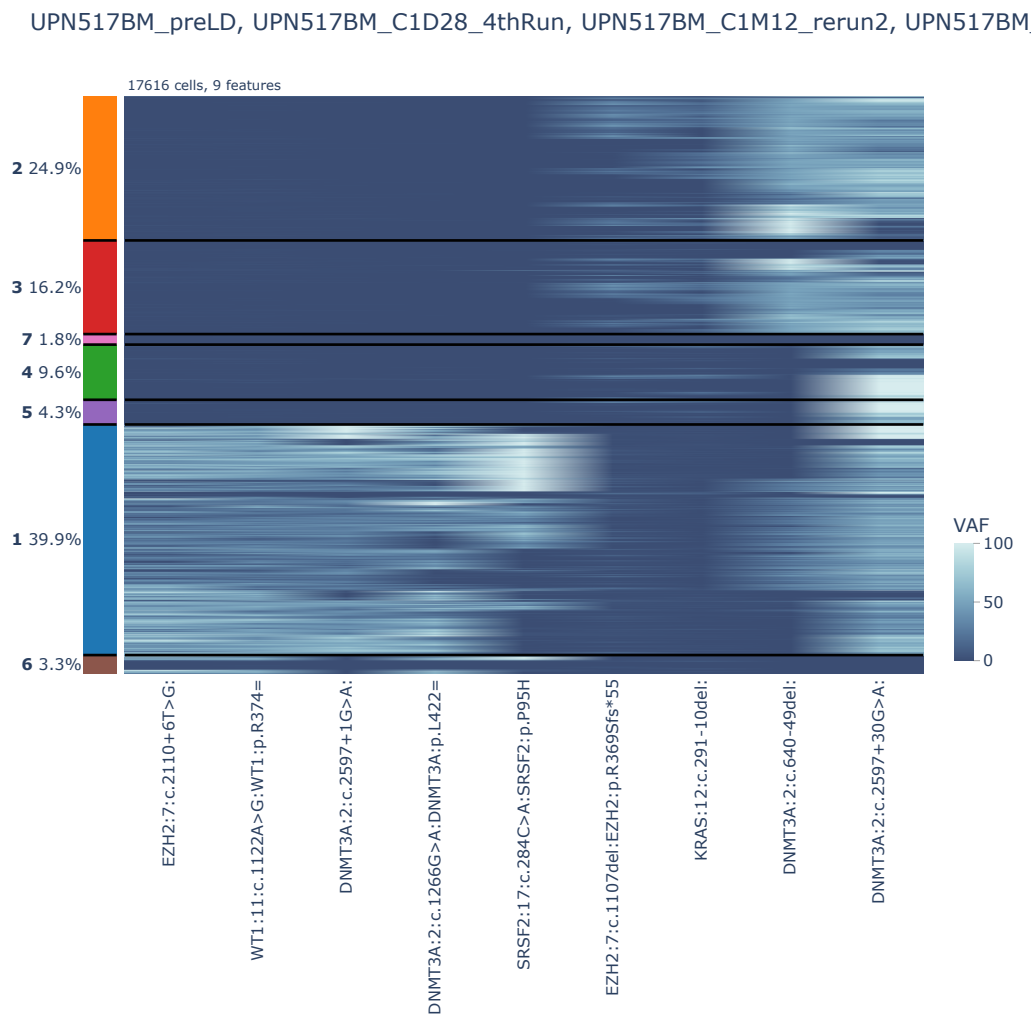
In [49]:
```
# Visualize data and cluster it using different methods
sample.dna.cluster(attribute='umap', method='kmeans', n_clusters=7)
```

```
C:\Users\smile\anaconda3 2023\envs\mosaic\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warnin
g
```

In [50]:
```
# Plot UMAP projection w/ alternative clustering results
fig = sample.dna.scatterplot(attribute='umap', colorby='label')
go.Figure(fig)
```



In [51]:
```
# Plot heatmap using NGT
# NGT, NGT_FILTERED, AF, AF_MISSING
fig = sample.dna.heatmap(attribute='AF')
go.Figure(fig)
```



Renaming Subclones (Labels)

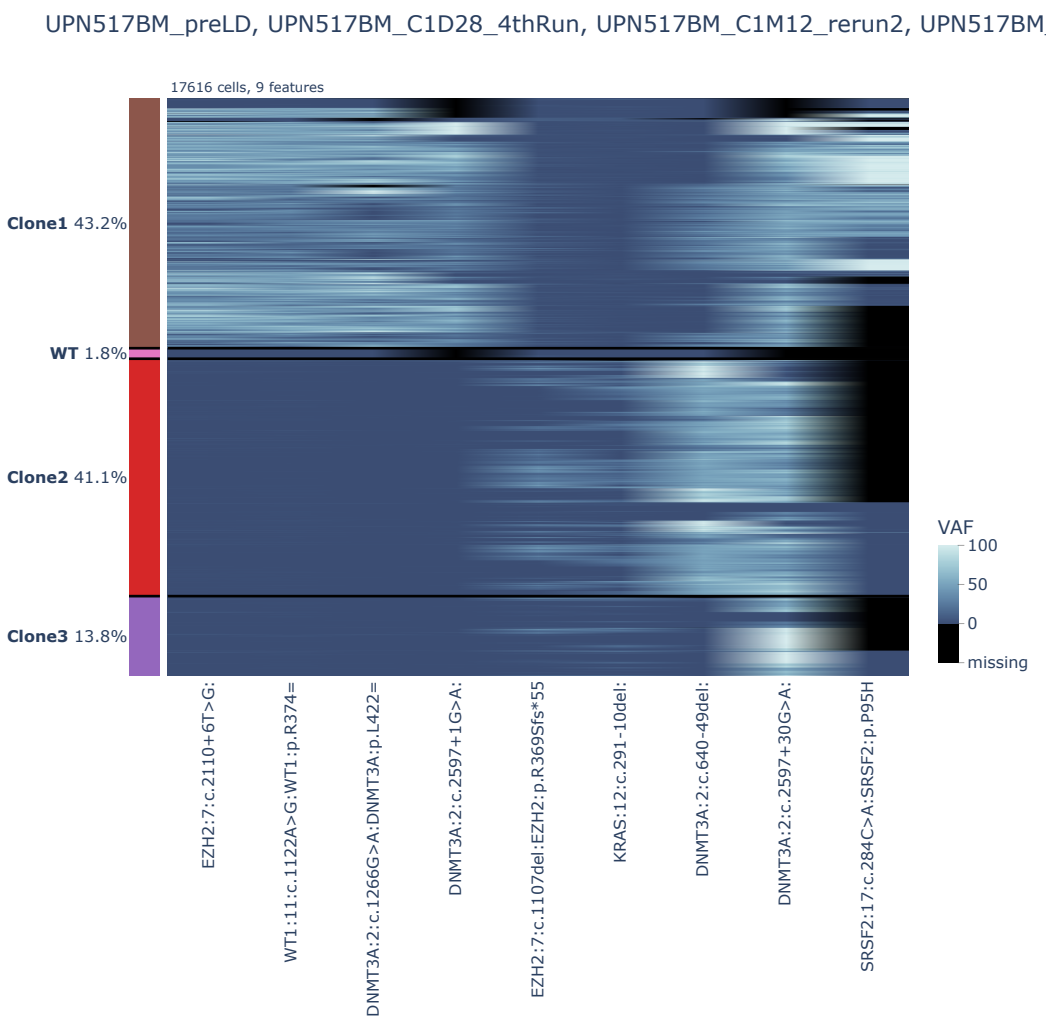
In [52]:
```
# Rename clusters based on genotypes
# Merge clusters by giving them the same name
# Rename clusters identically that are to be merged into one and discarded (e.g. "FP" for false-positive)
sample.dna.rename_labels(
  {
    '1': 'Clone1',
    '2': 'Clone2',
    '3': 'Clone2',
    '4': 'Clone3',
    '5': 'Clone3',
    '6': 'Clone1',
    '7': 'WT',


   # 'missing': 'FP',
   # 'small': 'FP'
  }
)
```

In [53]:
```
# Remove barcodes (clones) from data based on renamed labels
# The reduced data set will now be called 'sample.dna2'
fp_barcodes = sample.dna.barcodes({"FP",})
sample.dna2 = sample.dna.drop(fp_barcodes)
set(sample.dna2.get_labels())
```

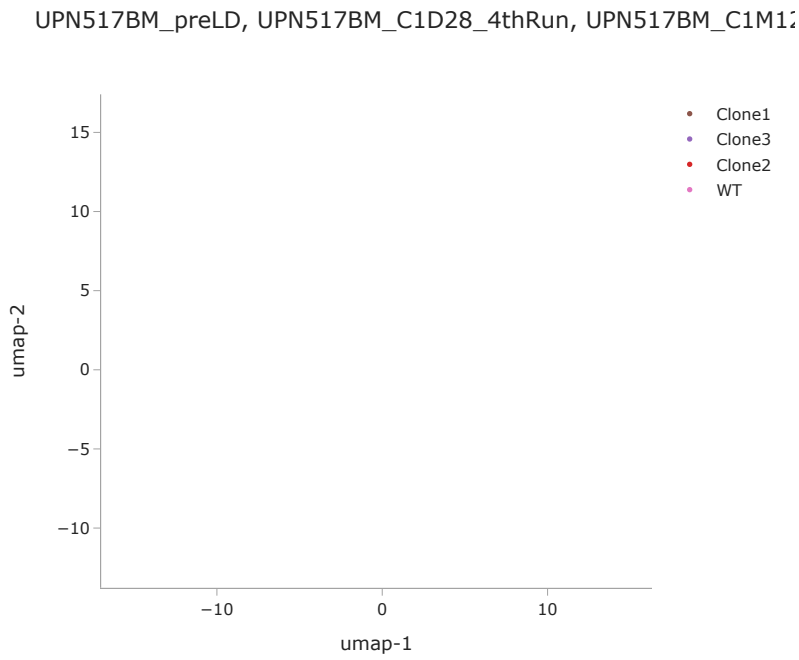
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[53], line 4
      1 # Remove barcodes (clones) from data based on renamed labels
      2 # The reduced data set will now be called 'sample.dna2'
      3 fp_barcodes = sample.dna.barcodes({"FP",})
----> 4 sample.dna2 = sample.dna.drop(fp_barcodes)
      5 set(sample.dna2.get_labels())

File ~\anaconda3 2023\envs\mosaic\lib\site-packages\missionbio\mosaic\assay.py:383, in _Assay.drop(self, values, value_type)
    381         remaining_bars = ~np.isin(self.barcodes(), values)
    382     else:
--> 383         raise ValueError("Not all the given values are present in the assay.")
    384 else:
    385     raise ValueError("value_type must be `id`, `barcode`, or `None`.")

ValueError: Not all the given values are present in the assay.
```

In [54]:
```
# Use only if no barcodes are removed by the above-listed command
sample.dna2 = sample.dna
```

In [55]:
```
# Redraw heatmap
fig = sample.dna2.heatmap(attribute='AF_MISSING')
go.Figure(fig)
```



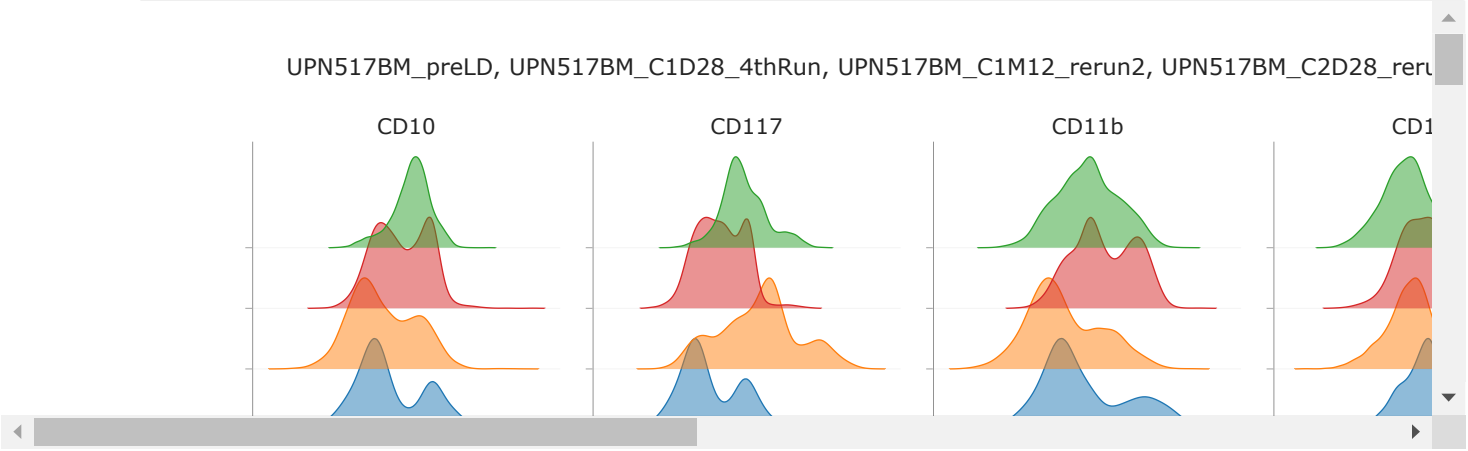
In [56]:
```
# Plot UMAP projection w/ alternative clustering results
fig = sample.dna2.scatterplot(attribute='umap', colorby='label')
go.Figure(fig)
```



In [64]:
```
cellsOfInterest = sample.dna2.row_attrs["sample_name"]=="UPN517BM_C2D28_rerun2"
singleSample = sample.dna2[cellsOfInterest,:]
singleSample.scatterplot(attribute='umap',colorby='label')

# ['UPN517BM_preLD'] ['UPN517BM_C1D28_4thRun'] ['UPN517BM_C1M12_rerun2'] ['UPN517BM_C2D28_rerun2']
```

Out[64]:
```
FigureWidget({
    'data': [{'customdata': array([['label: Clone2<br>AACAACCTACAGCACGTG-UPN517BM_C2D28_rerun2'],
                                   ['label: Clone2<br>AACAACCTACATTCGAAT-UPN517BM_C2D28_rerun2'],
                                   ['label: Clone2<br>AACAATGCACAGGTTGTC-UPN517BM_C2D28_rerun2'],
                                   ...,
                                   ['label: Clone2<br>TTGTCAACCCTATTAAGG-UPN517BM_C2D28_rerun2'],
                                   ['label: Clone2<br>TTGTCAACCTACCTCCTC-UPN517BM_C2D28_rerun2'],
                                   ['label: Clone2<br>TTGTTAGAGAGCTATATG-UPN517BM_C2D28_rerun2']],
                                  dtype=object),
              'hovertemplate': '<b>%{customdata}</b><extra></extra>',
              'legendgroup': 'Clone2',
              'marker': {'color': array(['#d62728', '#d62728', '#d62728', ..., '#d62728', '#d62728', '#d62728'],
                                        dtype=object),
                         'size': 4,
                         'symbol': 'circle'},
              'mode': 'markers',
              'name': 'Clone2',
              'showlegend': True,
              'type': 'scattergl',
              'uid': '8048fccf-f005-4b5e-be40-4e05af419dbf',
```

In [80]:
```
# If analyzing multiple samples, re-plot heatmap with barcodes ordered based on sample, 'NGT_FILTERED', 'AF' 'AF_FILTERED'

fig = sample.protein.ridgeplot(attribute='normalized_counts',splitby="sample_name")
go.Figure(fig)
```



In [ ]:
```
# Adding write_image function saves the picture in high-res
# See help() to change resolution or file type
help(fig.write_image)
```

In [ ]:
```
fig.write_image("heatmap_example.svg",scale=3)
```

In [68]:
```
# Evaluate new total number of cells after the above filtering
sample.dna2.shape
```

Out[68]: (17616, 9)

In [69]:
```
# Compare to original cell number
sample.dna.shape
```

Out[69]: (17616, 9)

CNV Analysis

Topics covered

1. Amplicon filtering and read normalization.
2. Genotype-guided ploidy computation and CNV clustering.
3. Statistical significance analysis.

Filtering And Normalization

In [162]:
```
# Filtering Amplicons
# This returns a table of the reads for each amplicon in each cell
reads = sample.cnv.get_attribute('read_counts', constraint='row+col')
reads
```

Out[162]:

read_counts	TAMPL6114	TAMPL6115	TAMPL6116	TAMPL6117	TAMPL6118	TAMPL6119	TAMPL6120	TAMPL6121	TAMPL6122	TAMPL6123	...
AACAACCTACCTCGCGTT-UPN517BM_preLD	9	92	16	10	0	0	9	12	7	0	...
AACAACCTATGGAACTAA-UPN517BM_preLD	7	71	106	40	4	0	27	108	26	0	...
AACAACTGGCCGATACCG-UPN517BM_preLD	72	68	188	49	6	0	23	116	43	0	...
AACAACTGGCGGACAAGT-UPN517BM_preLD	4	51	23	16	4	0	14	27	13	0	...
AACAACTGGGCGAGCAAG-UPN517BM_preLD	1	47	58	29	3	0	41	61	5	0	...
...	...	...	...	...	...	...	...	...	...	...	...
TTGTCAACCACAGCGGTC-UPN517BM_C2D28_rerun2	0	280	200	551	391	0	233	312	370	261	...
TTGTCAACCCTATAGCTC-UPN517BM_C2D28_rerun2	0	19	42	58	39	8	44	55	54	26	...
TTGTCAACCCTATTAAGG-UPN517BM_C2D28_rerun2	15	641	616	280	565	117	951	378	440	615	...
TTGTCAACCTACCTCCTC-UPN517BM_C2D28_rerun2	0	2	170	155	364	34	431	298	395	174	...
TTGTTAGAGAGCTATATG-UPN517BM_C2D28_rerun2	0	174	183	263	431	0	389	237	74	146	...

17616 rows × 127 columns

In [163]:
```
# Only amplicons found in more than half the cells are analyzed
# The other amplicons are dropped
# Note: optional for samples with expected biological missing data
working_amplicons = (reads.median() > 0).values
sample.cnv = sample.cnv[:, working_amplicons]
```

In [164]:
```
# Additionally, we keep only valid barcodes from the DNA analysis and are storing the data
# in a new variable sample.cnv2 (e.g., in order to not overwrite sample.cnv)
sample.cnv2 = sample.cnv[sample.dna2.barcodes(),:]

# Assign the DNA labels to the CNV assay class
# We want to ensure the labels (e.g., names of cell populations) are identical across assay classes
sample.cnv2.set_labels(sample.dna2.get_labels())
sample.cnv2.set_palette(sample.dna2.get_palette())
```

In [165]:
```
# Compare amplicon number and cell number original
sample.cnv.shape
```

Out[165]: (17616, 127)

In [166]:
```
# Compare amplicon number and cell number
sample.cnv2.shape
```

Out[166]: (17616, 127)

In [ ]:
```
help(sample.cnv2.normalize_reads)
```

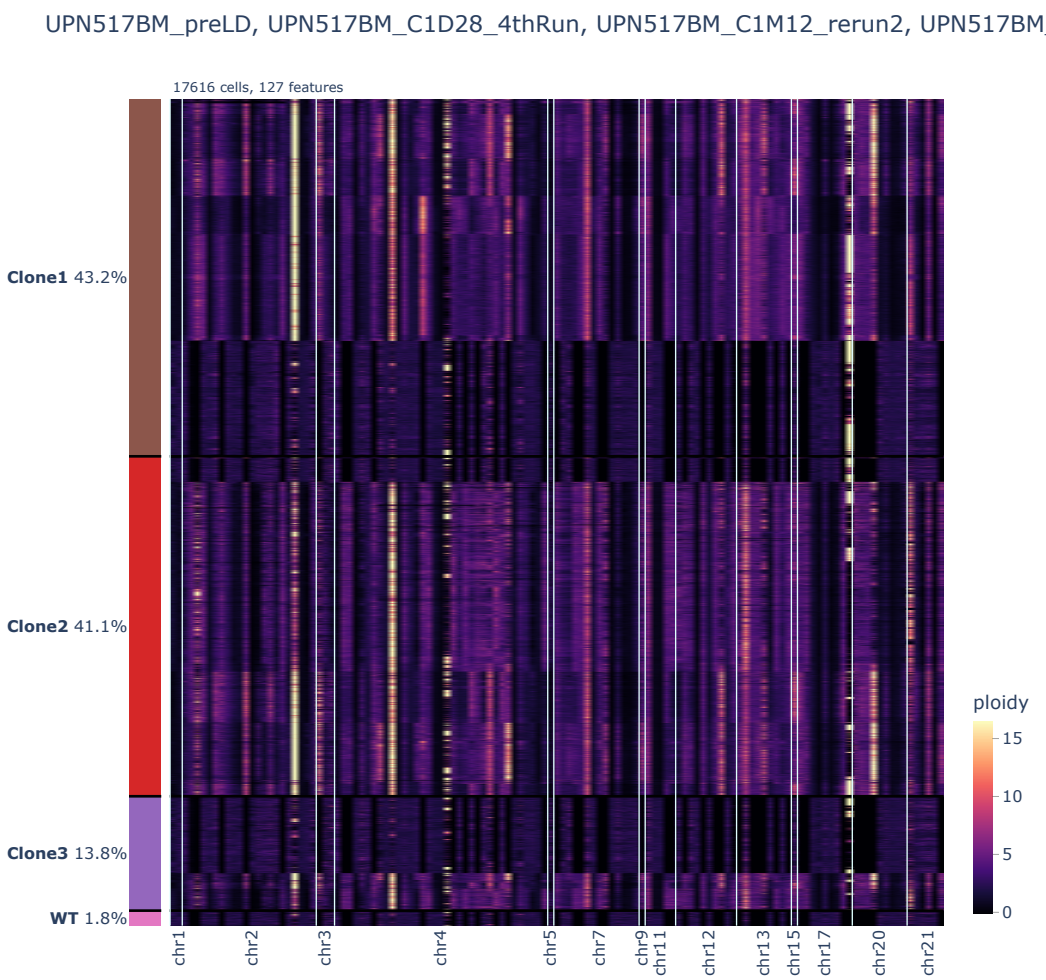
In [167]:
```
# Normalize the reads
sample.cnv2.normalize_reads()
```

In [ ]:
```
# The normalized reads can sometimes reveal CNV events prior to ploidy calculation
fig = sample.cnv2.heatmap('normalized_counts', features='positions')
go.Figure(fig)
```

Genotype-Guided Ploidy Clustering

In [168]:
```
# Assuming the presence of a cell population that is considered diploid across
# all the amplicons, we can compute the ploidy of all other cell populations (previously defined)
sample.cnv2.compute_ploidy(diploid_cells=sample.dna.barcodes('WT'))
```

In [169]:
```
# Heatmap with the features ordered by the default amplicon order
fig = sample.cnv2.heatmap('ploidy', features='positions')
go.Figure(fig)
```

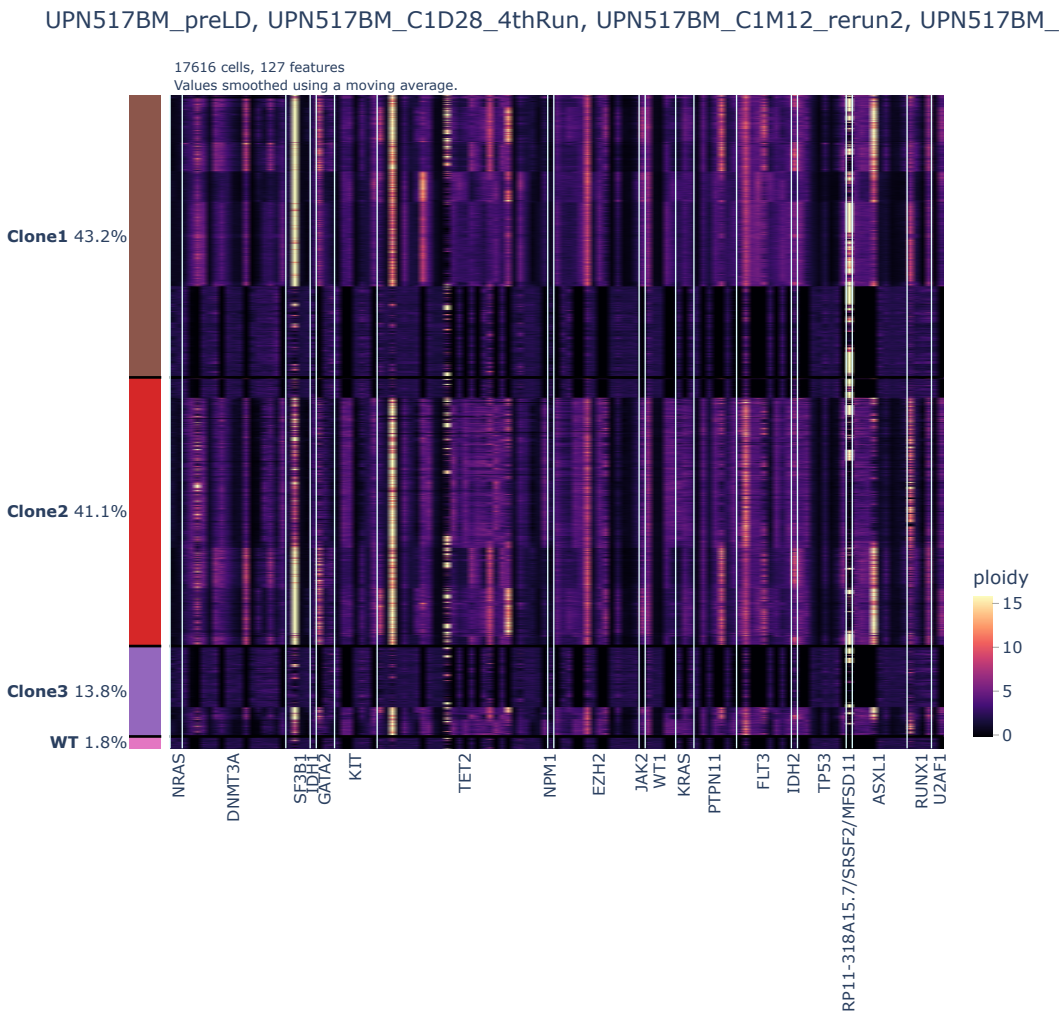


In [ ]:
```
# Scale the figure width and plot as a static image
# Double click on the plot to zoom-in and improve the resolution
fig = sample.cnv2.heatmap('ploidy', features='positions')
fig.layout.width = 1200
mutils.static_fig(fig, figsize=(20, 20))
```

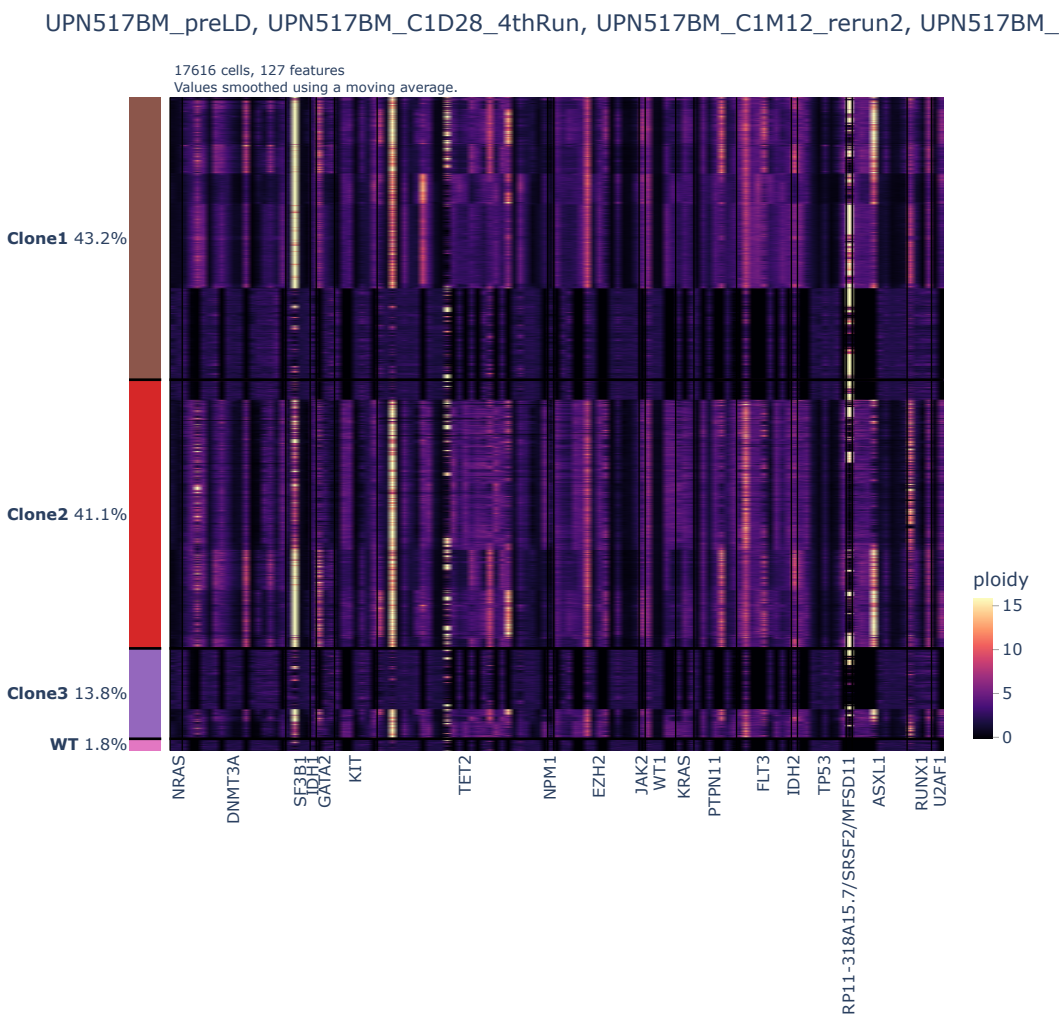
In [ ]:
```
# Heatmap for a subset of the chromosomes
fig = sample.cnv2.heatmap('ploidy', features=['2', '13'])

# Optionally, restrict the range of ploidy values based on observed/expected CNV events (commented out)
#fig.layout.coloraxis.cmax = 4
#fig.layout.coloraxis.cmin = 0

go.Figure(fig)
```

In [171]:
```
# Heatmap with the features grouped by the genes
# The first time this runs, it fetches the gene names from ensembl
# The annotation can also be fetched using sample.cnv.get_gene_names()
# The plots can also be smoothed using a moving average with the convolve parameter

fig = sample.cnv2.heatmap('ploidy', features='genes', convolve=1)
sample.cnv.get_gene_names()

# Optionally, restrict the range of ploidy values based on observed/expected CNV events (commented out)
#fig.layout.coloraxis.cmax = 4
#fig.layout.coloraxis.cmin = 0

go.Figure(fig)
```

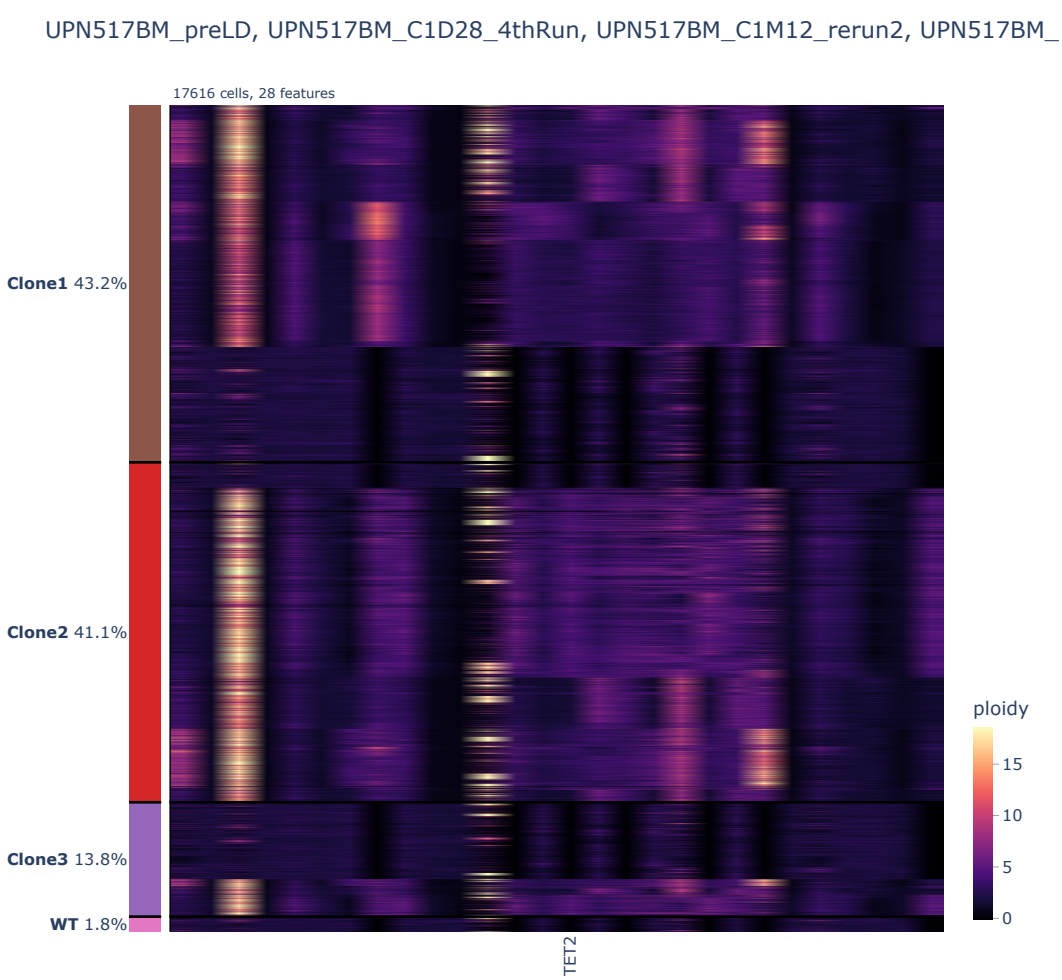
```
Fetching annotation for regions:   0%|          | 0/20 [00:00<?, ?it/s]
```



In [172]:
```
# Change the color scale to "magma" - other suitable options might be "viridis", "plasma", "blues", "blues_r" ...
fig.layout.coloraxis.colorscale = 'magma'

# Update the separating lines to be black
for shape in fig.layout.shapes:
    shape.line.color = '#000000'

# Optionally, restrict the range of ploidy values based on observed/expected CNV events (commented out)
#fig.layout.coloraxis.cmax = 4
#fig.layout.coloraxis.cmin = 0

go.Figure(fig)
```

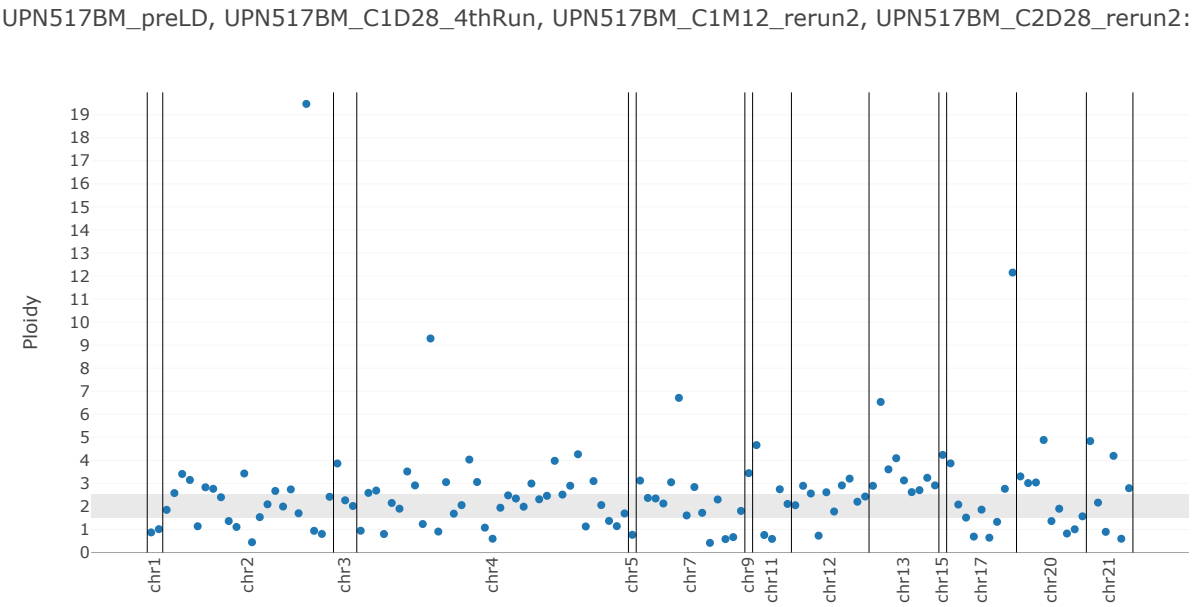


In [173]:
```
# Heatmap for a subset of the genes
fig = sample.cnv2.heatmap('ploidy', features=[ "TET2", ])
fig.layout.coloraxis.colorscale = 'magma'

# Update the separating lines to be black
for shape in fig.layout.shapes:
    shape.line.color = '#000000'

# Optionally, restrict the range of ploidy values based on observed/expected CNV events (commented out)
#fig.layout.coloraxis.cmax = 2
#fig.layout.coloraxis.cmin = 0

go.Figure(fig)
```

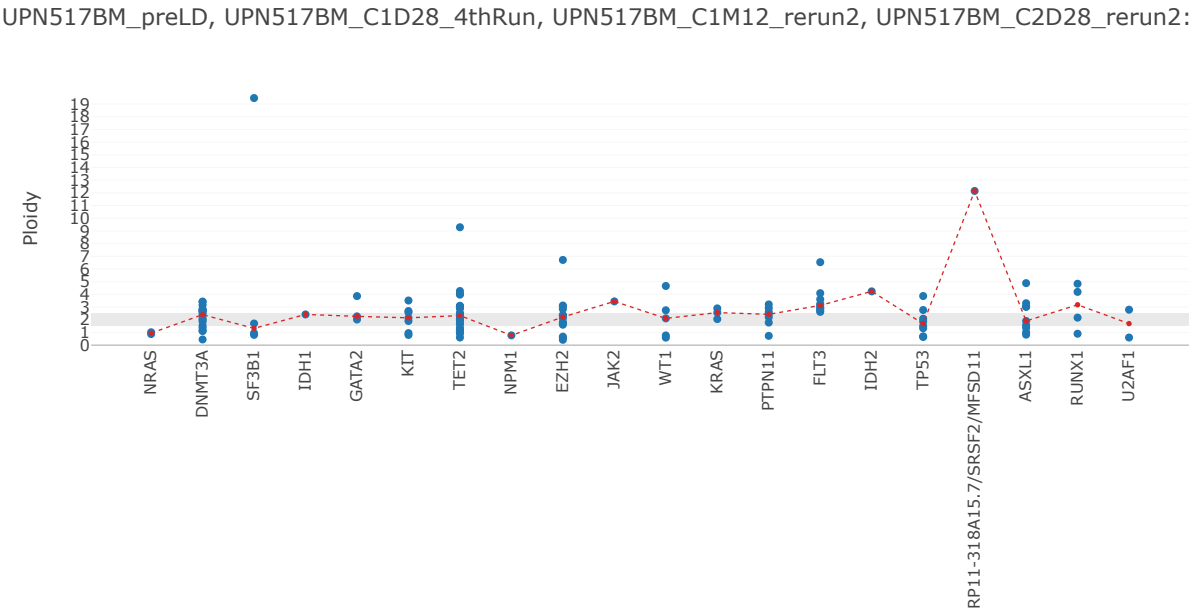


In [ ]:
```
help(sample.cnv.plot_ploidy)
```

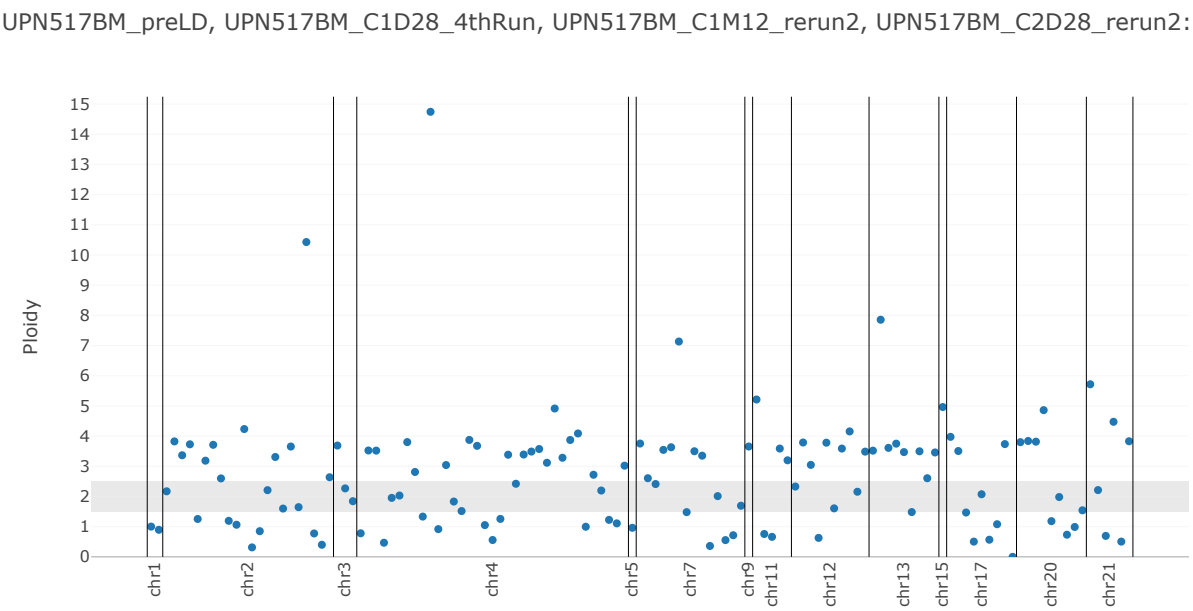
In [174]:
```
# Ploidy line plot across amplicons for all WT cells
# WT clone cells normalized to ploidy = 2 by default
sample.cnv2.plot_ploidy('Clone1')
```



In [175]:
```
# Same as above, only using different feature groupings (genes+amplicons)
# Red dots represent the median per gene, blue dots represent the individual amplicons in each gene
sample.cnv2.plot_ploidy('Clone1', features="genes+amplicons")
```

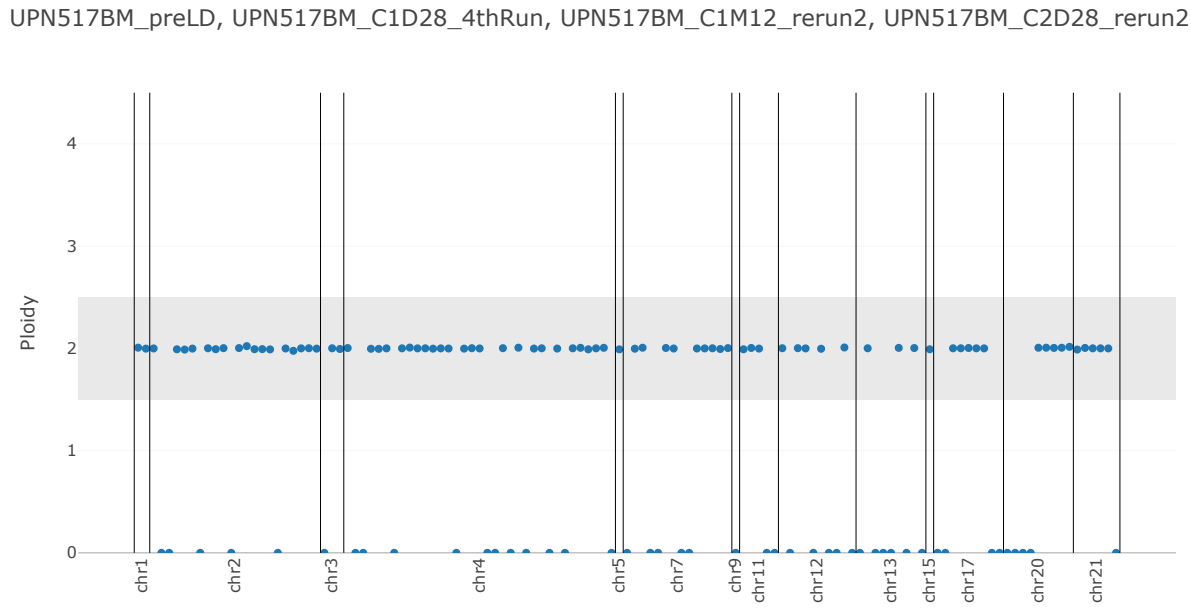


In [176]:
```
# Evaluate ploidy for other genotype-based defined clones by updating the code
sample.cnv2.plot_ploidy('Clone2')
```

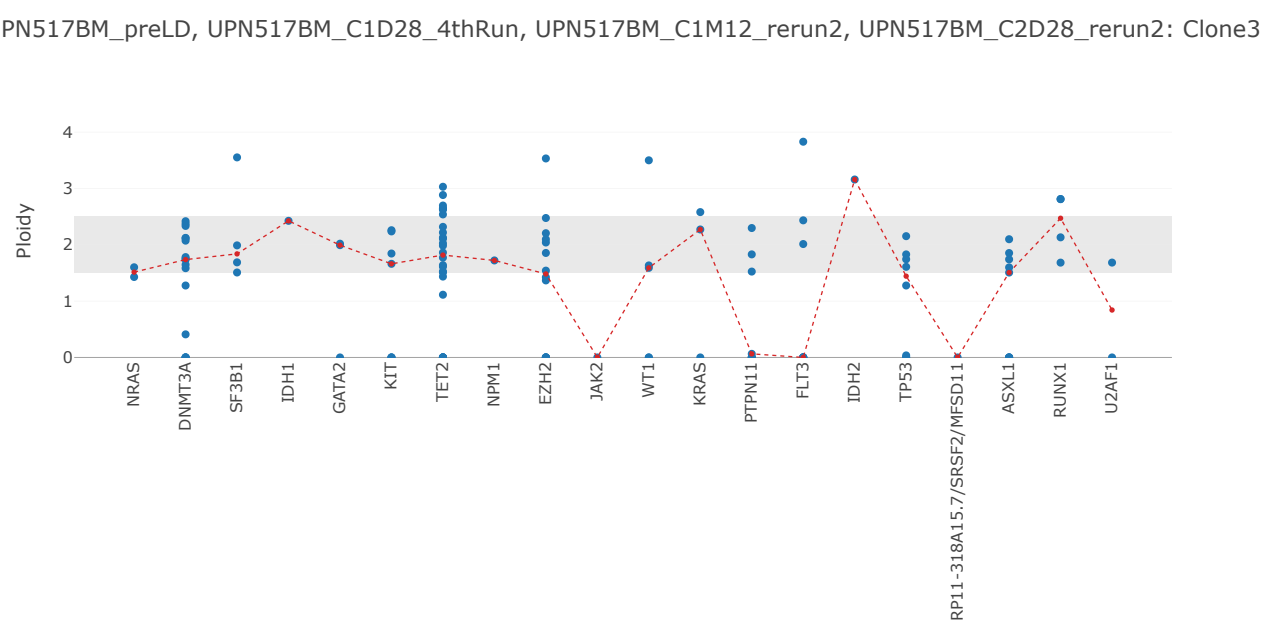


In [ ]:
```
# Evaluate ploidy for other genotype-based defined clones by updating the code
sample.cnv2.plot_ploidy('Clone2', features="genes+amplicons")
```

In [177]:
```
sample.cnv2.plot_ploidy('WT')
```



In [179]:
```
sample.cnv2.plot_ploidy('Clone3', features="genes+amplicons")
```



Statistics

In [180]:
```
pval_cnv, tstat_cnv = sample.cnv2.test_signature("normalized_counts")
```

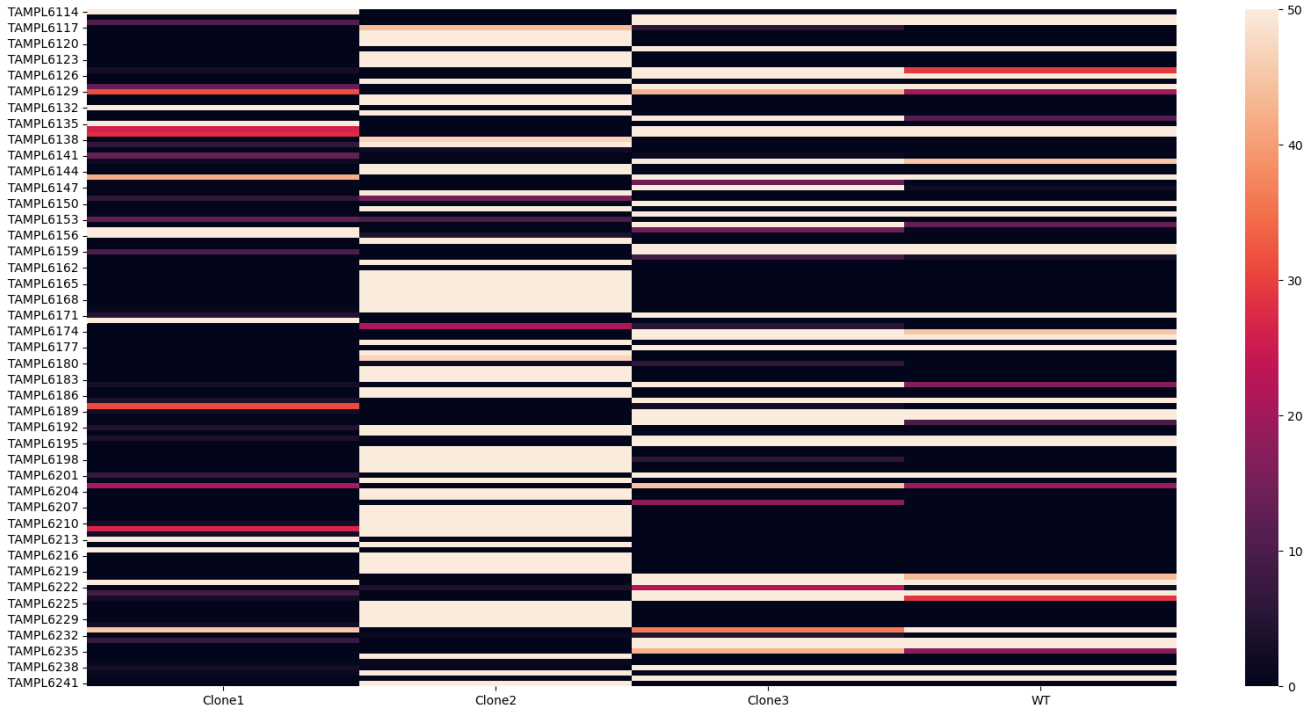
In [181]:
```
pval_cnv
```

Out[181]:

AMPL6122	TAMPL6123	...	TAMPL6232	TAMPL6233	TAMPL6234	TAMPL6235	TAMPL6236	TAMPL6237	TAMPL6238	TAMPL6239	TAMPL6240	TAMPL6241
3.517594e-03	2.971403e-29	...	3.712843e-07	3.006717e-08	4.741168e-01	8.240967e-08	3.521901e-24	0.077112	1.675630e-03	2.788036e-02	3.552776e-01	1.809369e-41
9.916065e-85	0.000000e+00	...	2.394047e-02	0.000000e+00	8.101111e-197	1.856194e-11	5.525300e-145	0.031443	1.344279e-222	1.083048e-72	1.942242e-258	0.000000e+00
2.257366e-82	0.000000e+00	...	1.268944e-05	0.000000e+00	7.977407e-260	9.081003e-44	1.176166e-66	0.000023	1.661347e-256	1.450655e-113	0.000000e+00	1.926613e-266
3.456069e-29	6.506171e-113	...	4.518970e-01	1.750188e-125	1.764430e-78	9.665508e-19	3.117135e-31	0.000419	2.364540e-65	5.177046e-59	5.740400e-83	7.512606e-109

In [182]:
```
pval_cnv = pval_cnv + 10 ** -50 + pval_cnv
pvals_cnv = -np.log10(pval_cnv) * (tstat_cnv > 0)
```

In [183]:
```
# Colored tiles indicate strong association of a ploidy with a particular cluster
plt.figure(figsize=(20, 10))
fig = sns.heatmap(pvals_cnv.T, vmax=50, vmin=0)
```



Protein Analysis

Topics covered

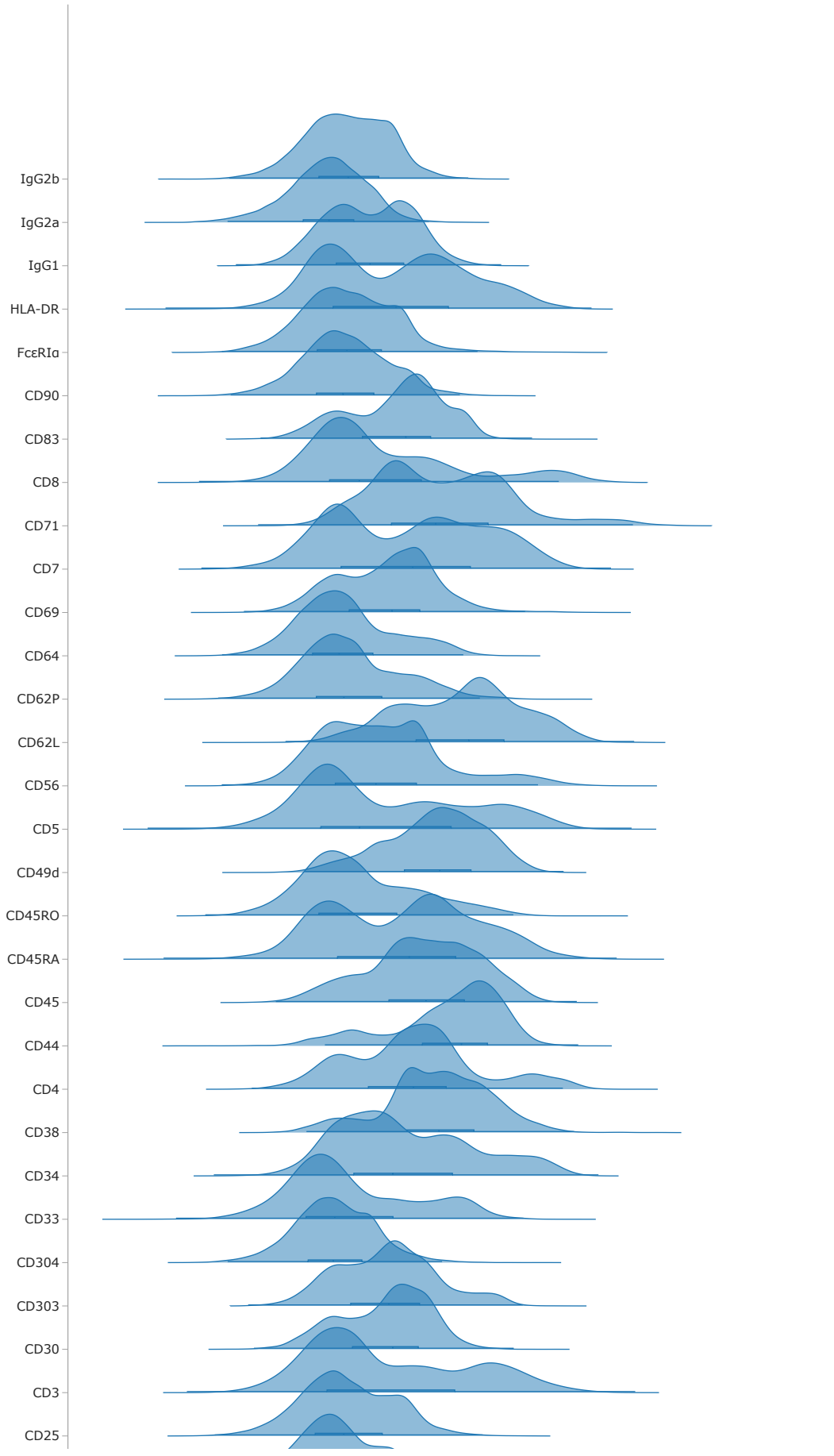
1. Normalization and assessment of AOC abundance
2. Clustering and visualization
3. Statistical significance analysis

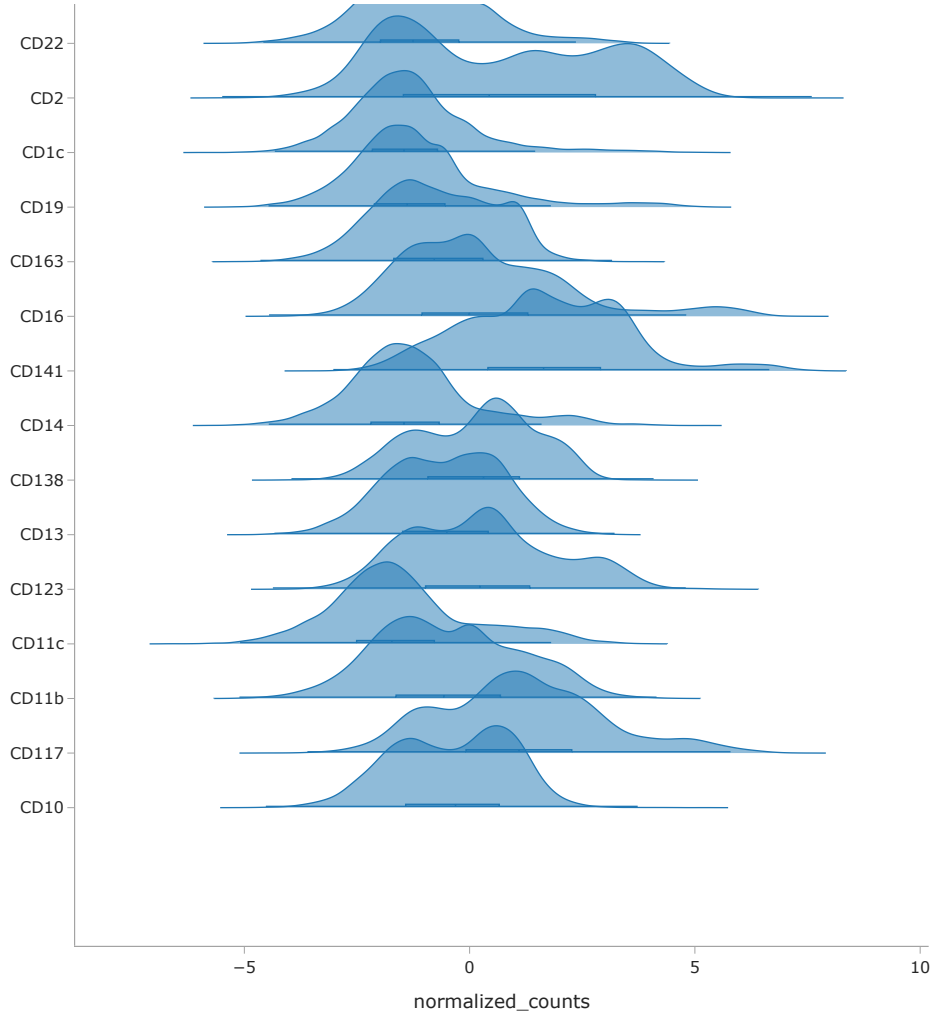
Normalization And Data Inspection

In [ ]:
```
help(sample.protein.normalize_reads)
```

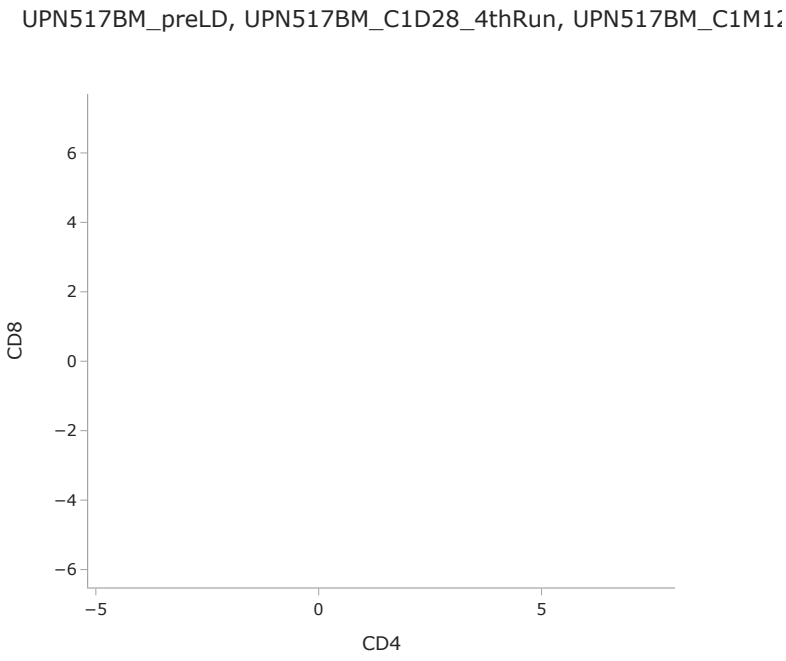
In [70]:
```
# Normalize reads
# Three different methods available: CLR = Centered Log Ratio
sample.protein.normalize_reads('CLR')
```

In [71]:
```
# Ridge plot: shows distribution of AOC counts across all cells (bimodal = positive and negative cell populations)
sample.protein.ridgeplot(attribute='normalized_counts',
                         features=sample.protein.ids())
```

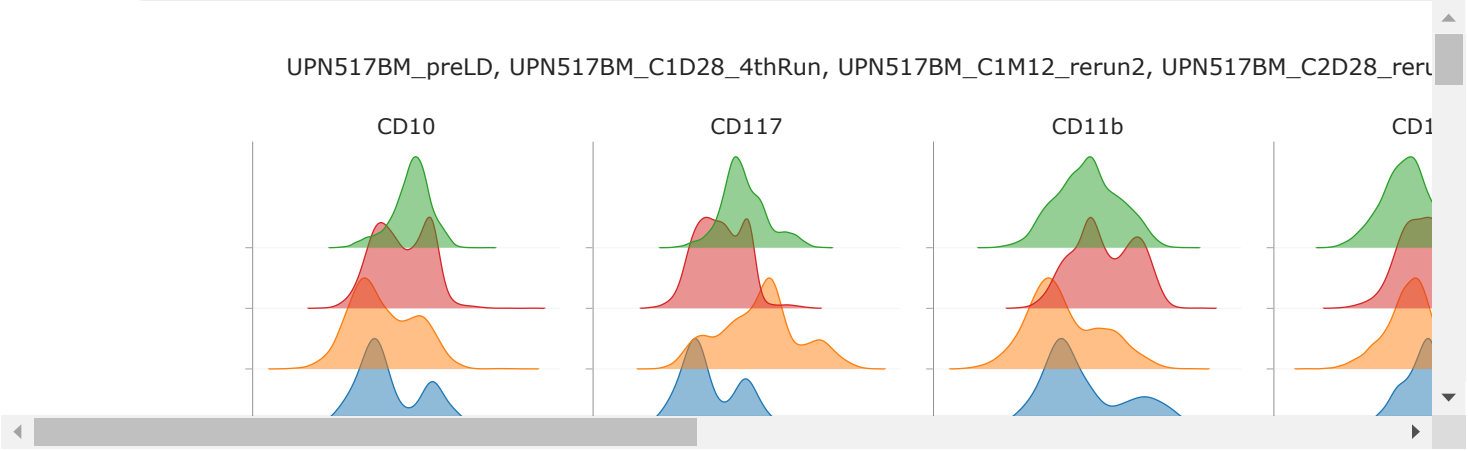

UPN517BM_preLD, UPN517BM_C1D28_4thRun, UPN517BM_C1M12_rerun2, UPN517BM_
IgG2b
IgG2a
IgG1
HLA-DR
FcεRIα
CD90
CD83
CD8
CD71
CD7
CD69
CD64
CD62P
CD62L
CD56
CD5
CD49d
CD45RO
CD45RA
CD45
CD44
CD4
CD38
CD34
CD33
CD304
CD303
CD30
CD3
CD25



In [76]:
```
# Plotting combinations of AOCs on a biaxial scatterplot, to mimic FACS data
fig = sample.protein.feature_scatter(layer='normalized_counts', ids=['CD4', 'CD8'])
go.Figure(fig)
```



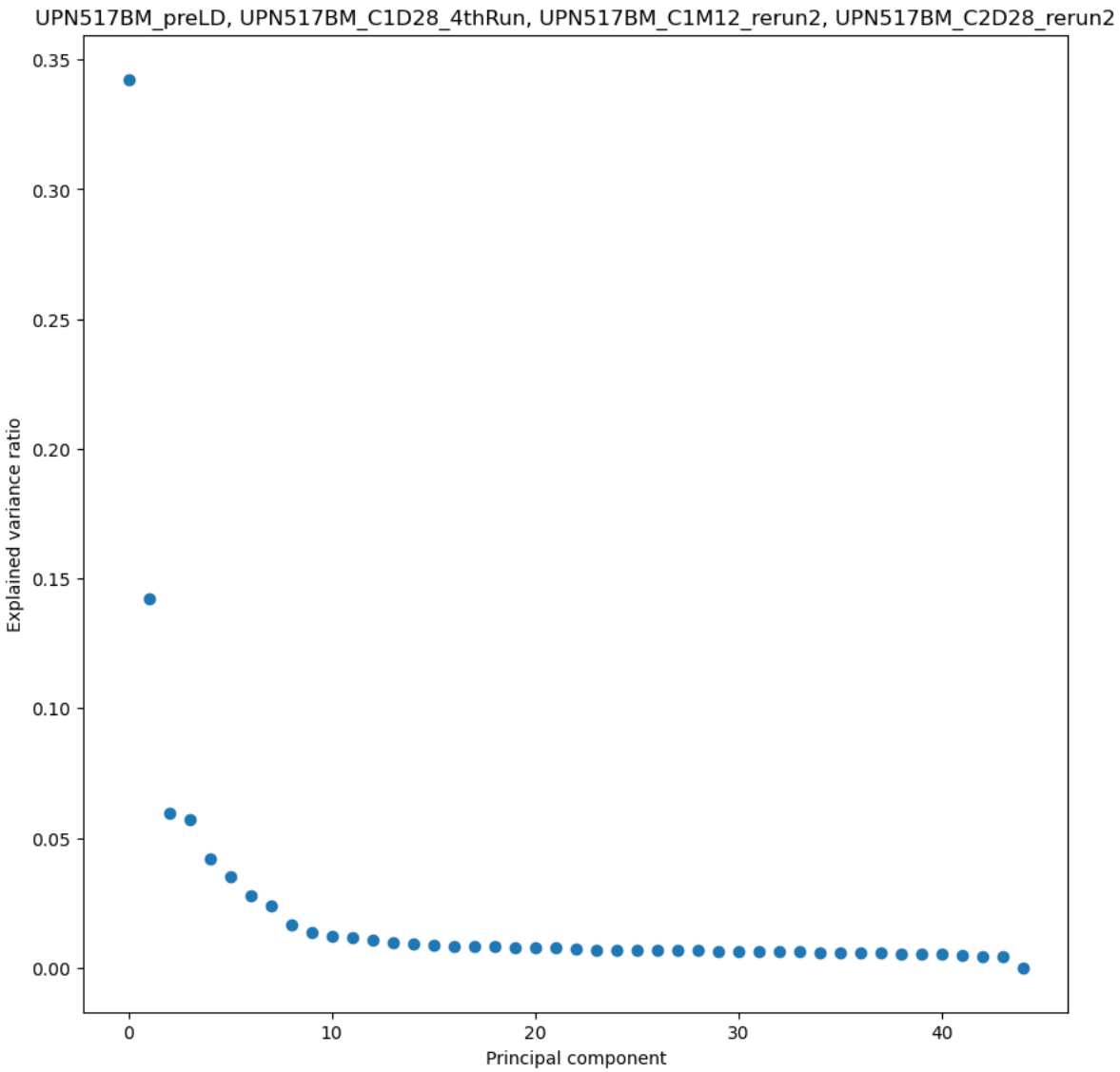
In [81]:
```
# If analyzing multiple samples, re-plot heatmap with barcodes ordered based on sample, 'NGT_FILTERED', 'AF' 'AF_FILTERED'

fig = sample.protein.ridgeplot(attribute='normalized_counts',splitby="sample_name")
go.Figure(fig)
```



In [ ]:
```
# Optionally, remove AOCs from the data set prior to clustering
# For example, those that don't display any signal in any cells
sample.protein = sample.protein.drop(['IgG2a','IgG2b','IgG1'])
```

Clustering And Visualization

In [82]:
```
# Clustering similar to "CLUSTERING METHOD #2" of DNA clustering using genotypes
# Select fewer components when analyzing custom panels with 2-20 AOCs
sample.protein.run_pca(attribute='normalized_counts', components=45,show_plot=True)
```



In [83]:
```
# Rerun PCA with optimal number of components based on elbow plot analysis
# Typically, components = 10 is appropriate for 45-plex Biolegend panel
sample.protein.run_pca(attribute='normalized_counts', components=10, show_plot=False)
```

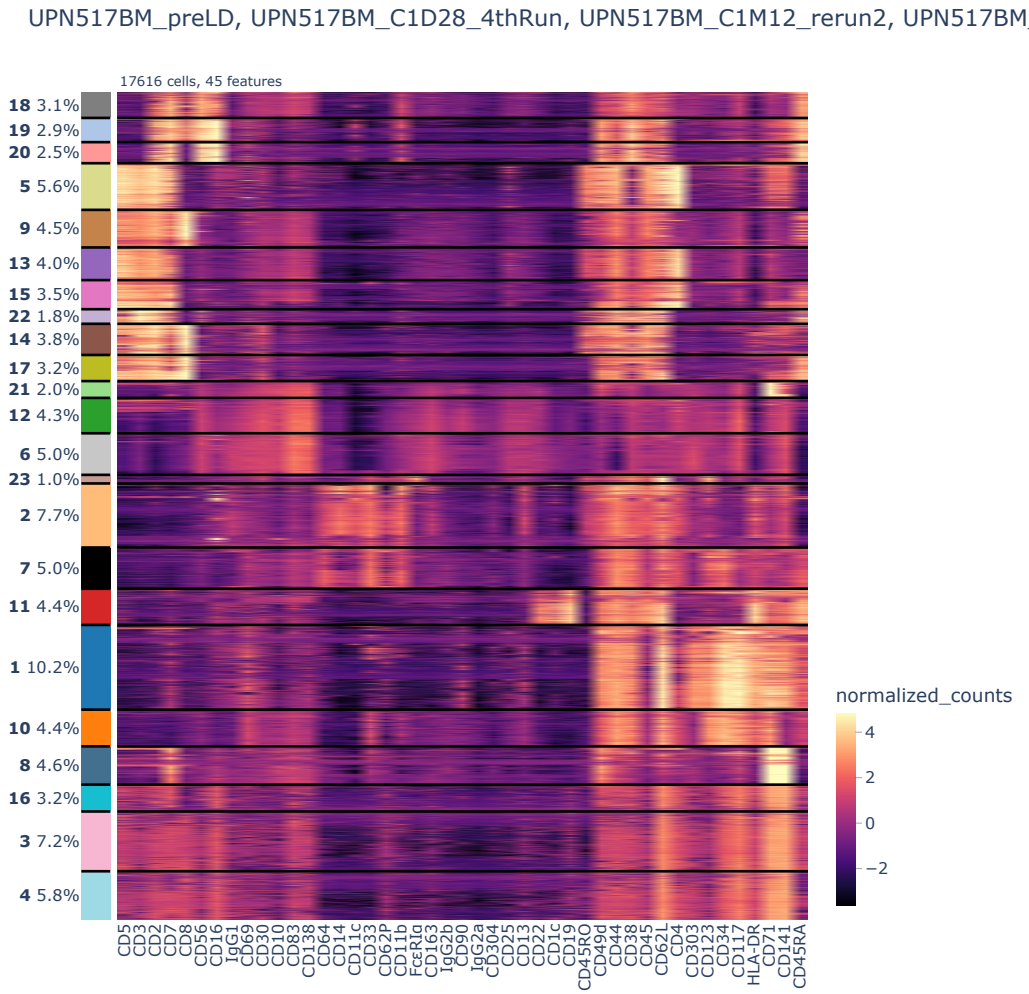
In [84]:
```
# Now run UMAP on the chosen PCs
# UMAPs rely on an initial randomization which leads to different projections every time
# To address this, pass random_state to the run_umap method
# See https://jlmelville.github.io/uwot/abparams.html for appropriate spread/min_dist values
sample.protein.run_umap(attribute='pca', random_state=42) #, spread=, min_dist=
```

In [90]:
```
# Cluster the AOCs. Several options are available, see help info for more options
# Graph-community clustering: the higher the k value, the smaller the number of clusters --> adjust if necessary
sample.protein.cluster(attribute='umap', method='graph-community',k=230)
```

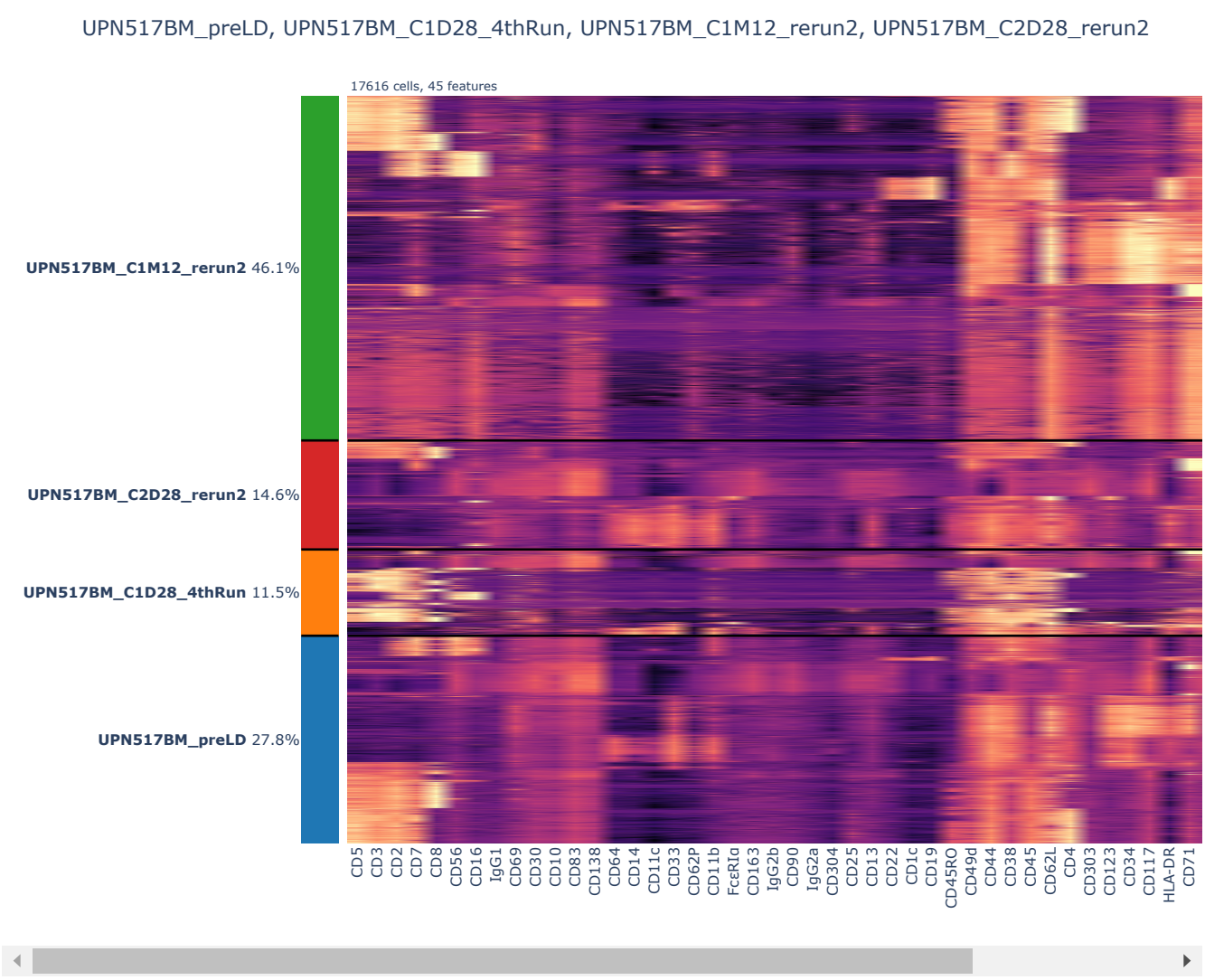
```
Creating the Shared Nearest Neighbors graph.

  0%|          | 0/4051680 [00:00<?, ?it/s]

Identifying clusters using Louvain community detection.

Number of clusters found: 23.
Modularity: 0.925
```

In [ ]:
```
help(sample.protein.heatmap)
```

In [91]:
```
# Plot heatmap of normalized, clustered protein data
fig = sample.protein.heatmap(attribute='normalized_counts')
go.Figure(fig)
```

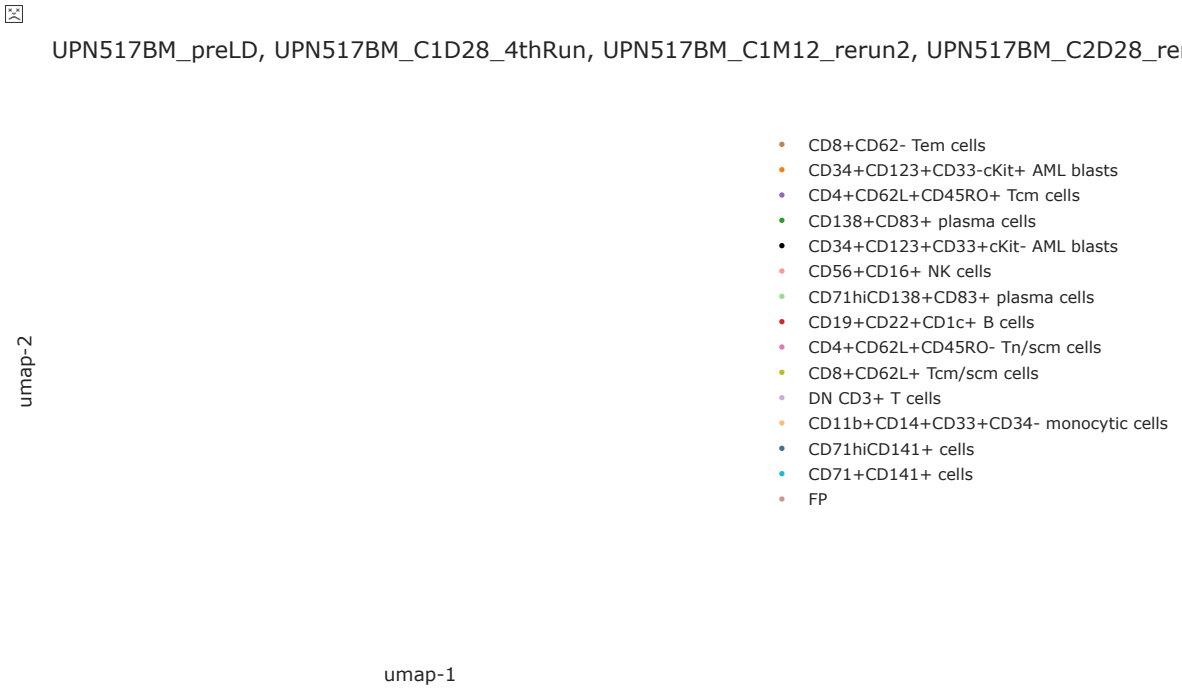


In [92]:
```
# If analyzing multiple samples, order barcodes by sample and then by cluster
fig = sample.protein.heatmap(attribute='normalized_counts',splitby='sample_name')
fig.layout.width = 1200
go.Figure(fig)
```



In [ ]:
```
help(sample.protein.scatterplot)
```

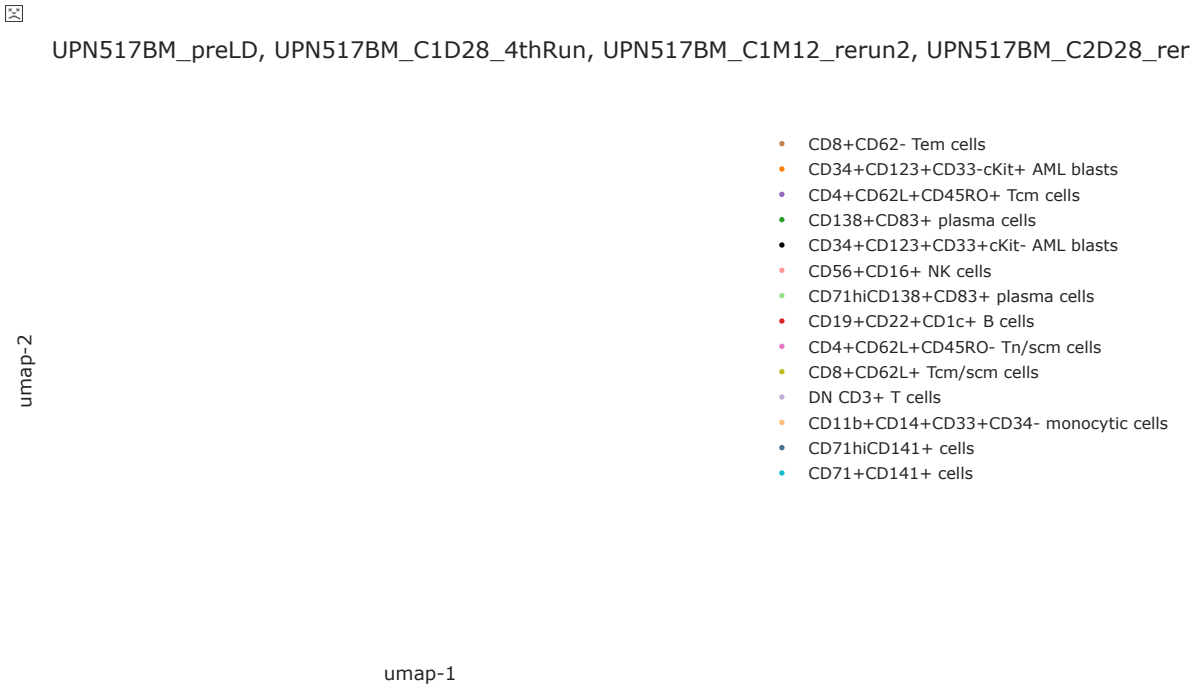
In [95]:
```
# Plot UMAP for visualization of clusters
fig = sample.protein.scatterplot(attribute='umap',colorby='label')
fig.layout.width = 900
go.Figure(fig)
```



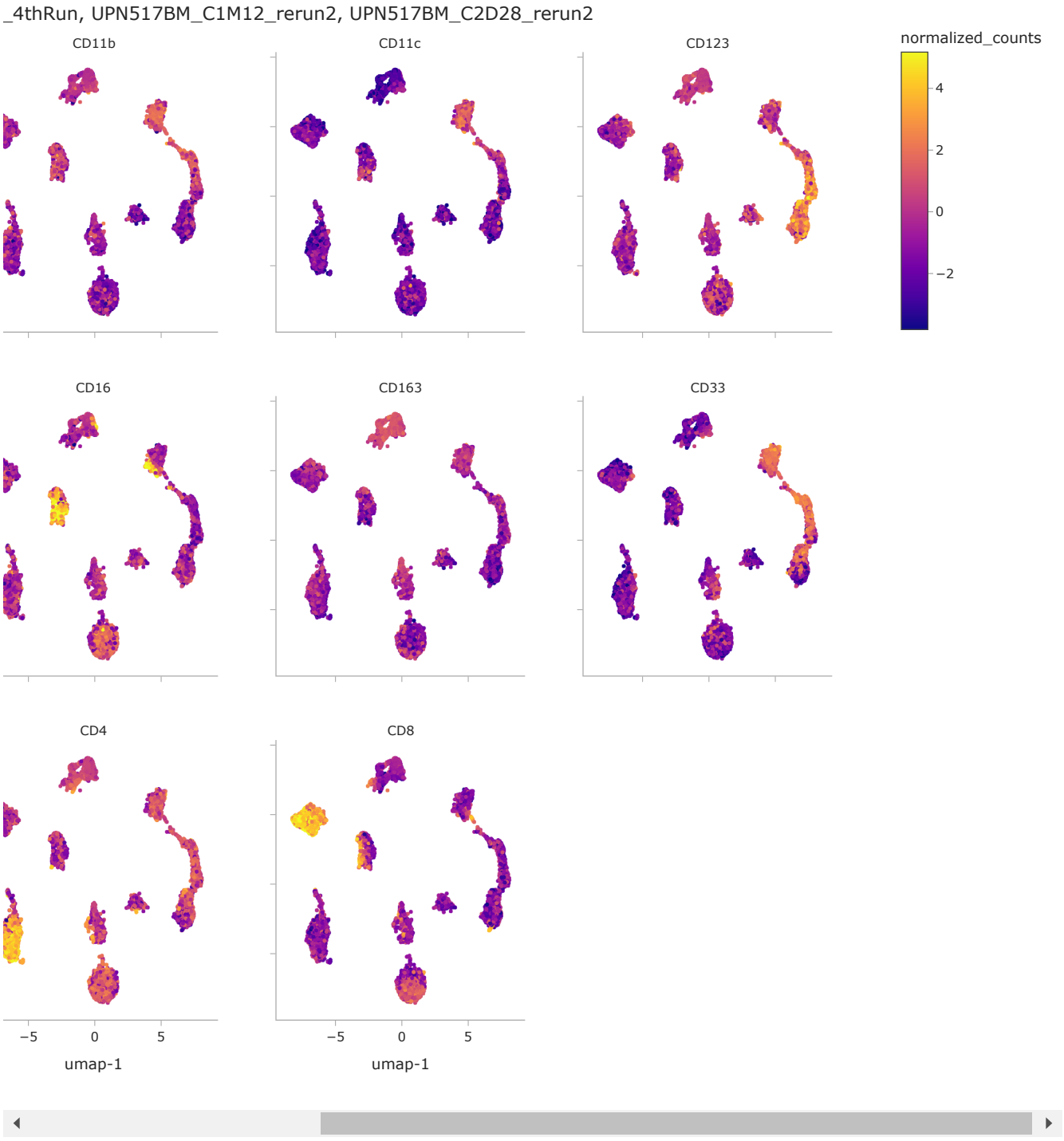
In [96]:
```
# Relabel clone names acccording to biology, combine clones by assigned identical names
# Just an example! Need to fill this in based on knowledge of the sample and markers
sample.protein.rename_labels(
    {
        '1': 'CD34+CD123+CD33-cKit+ AML blasts',
        '2': 'CD11b+CD14+CD33+CD34- monocytic cells',
        '3': 'CD71+CD141+ cells',
        '4': 'CD71+CD141+ cells',
        '5': 'CD4+CD62L+CD45RO+ Tcm cells',
        '6': 'CD138+CD83+ plasma cells',
        '7': 'CD34+CD123+CD33+cKit- AML blasts' ,
        '8': 'CD71hiCD141+ cells',
        '9': 'CD8+CD62- Tem cells',
        '10': 'CD34+CD123+CD33-cKit+ AML blasts' ,
        '11': 'CD19+CD22+CD1c+ B cells' ,
        '12': 'CD138+CD83+ plasma cells',
        '13': 'CD4+CD62L+CD45RO+ Tcm cells',
        '14': 'CD8+CD62L+ Tcm/scm cells',
        '15': 'CD4+CD62L+CD45RO- Tn/scm cells' ,
        '16': 'CD71+CD141+ cells',
        '17': 'CD8+CD62L+ Tcm/scm cells',
        '18': 'CD56+CD16+ NK cells',
        '19': 'CD56+CD16+ NK cells' ,
        '20': 'CD56+CD16+ NK cells',
        '21': 'CD71hiCD138+CD83+ plasma cells',
        '22': 'DN CD3+ T cells',
        '23': 'FP',
               }
)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[96], line 3
      1 # Relabel clone names acccording to biology, combine clones by assigned identical names
      2 # Just an example! Need to fill this in based on knowledge of the sample and markers
----> 3 sample.protein.rename_labels(
      4     {
      5         '1': 'CD34+CD123+CD33-cKit+ AML blasts',
      6         '2': 'CD11b+CD14+CD33+CD34- monocytic cells',
      7         '3': 'CD71+CD141+ cells',
      8         '4': 'CD71+CD141+ cells',
      9         '5': 'CD4+CD62L+CD45RO+ Tcm cells',
     10         '6': 'CD138+CD83+ plasma cells',
     11         '7': 'CD34+CD123+CD33+cKit- AML blasts' ,
     12         '8': 'CD71hiCD141+ cells',
     13         '9': 'CD8+CD62- Tem cells',
     14         '10': 'CD34+CD123+CD33-cKit+ AML blasts' ,
     15         '11': 'CD19+CD22+CD1c+ B cells' ,
     16         '12': 'CD138+CD83+ plasma cells',
     17         '13': 'CD4+CD62L+CD45RO+ Tcm cells',
     18         '14': 'CD8+CD62L+ Tcm/scm cells',
     19         '15': 'CD4+CD62L+CD45RO- Tn/scm cells' ,
     20         '16': 'CD71+CD141+ cells',
     21         '17': 'CD8+CD62L+ Tcm/scm cells',
     22         '18': 'CD56+CD16+ NK cells',
     23         '19': 'CD56+CD16+ NK cells' ,
     24         '20': 'CD56+CD16+ NK cells',
     25         '21': 'CD71hiCD138+CD83+ plasma cells',
     26         '22': 'DN CD3+ T cells',
     27         '23': 'FP',
     28                }
     29 )

File ~\anaconda3 2023\envs\mosaic\lib\site-packages\missionbio\mosaic\assay.py:1018, in _Assay.rename_labels(self, label_map)
   1016 for k in label_map:
   1017     labels = np.where(labels == k, label_map[k], labels)
-> 1018     palette[label_map[k]] = palette[k]
   1019     del palette[k]
   1021 self.add_row_attr(LABEL, labels)

KeyError: '1'
```

In [154]:
```
cellsOfInterest = sample.protein.row_attrs["sample_name"]=="UPN517BM_preLD"
singleSample = sample.protein[cellsOfInterest,:]
singleSample.scatterplot(attribute='umap',colorby='label')

fig = singleSample.scatterplot(attribute='umap',colorby='label')
fig.layout.width = 900
go.Figure(fig)

# ['UPN517BM_preLD'] ['UPN517BM_C1D28_4thRun'] ['UPN517BM_C1M12_rerun2'] ['UPN517BM_C2D28_rerun2']
```

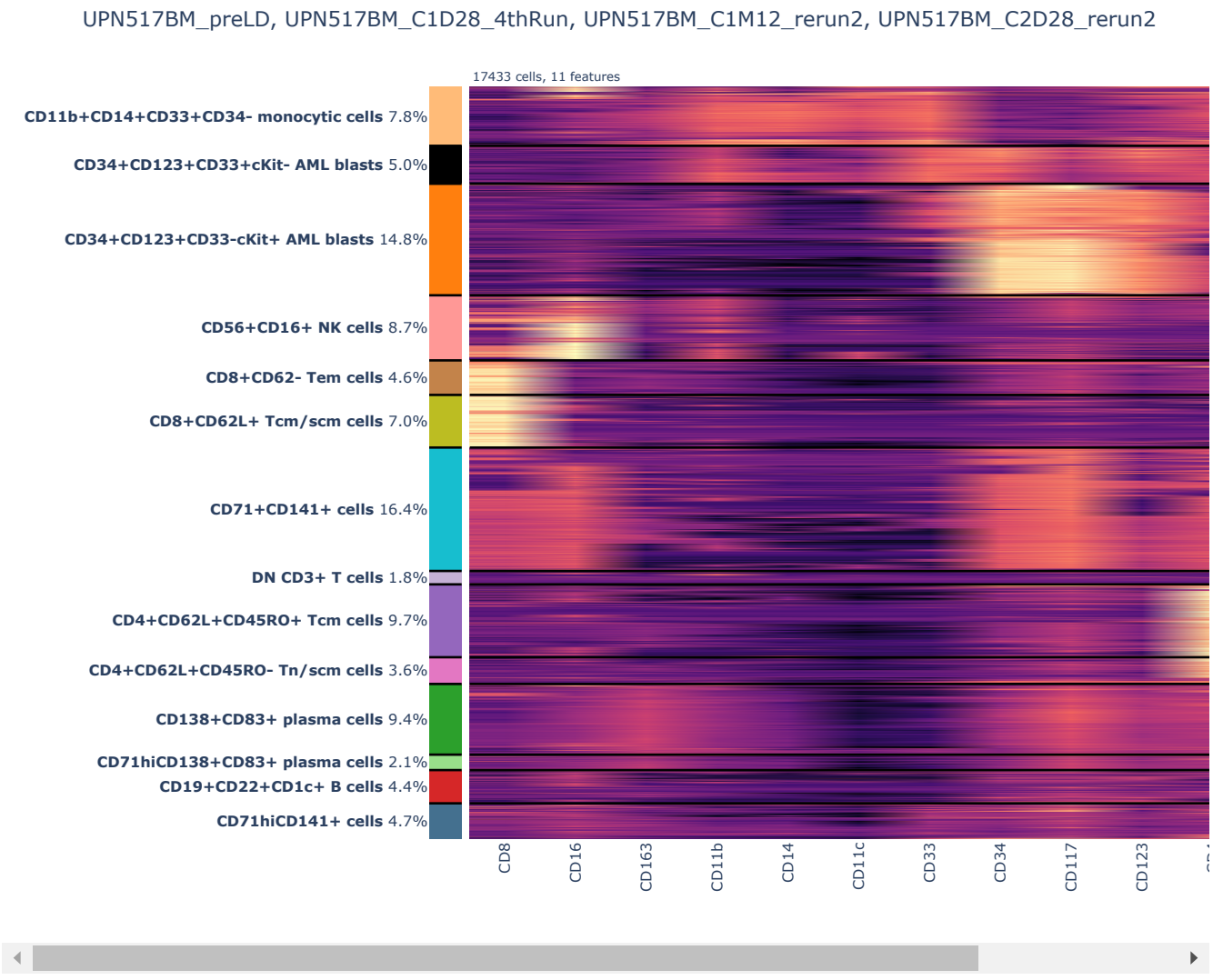


In [155]:
```
# UMAP colored by the expression of each AOC
fig = sample.protein.scatterplot(attribute='umap',
                                 colorby='normalized_counts',
                                 features=sample.protein.ids())
go.Figure(fig)
```

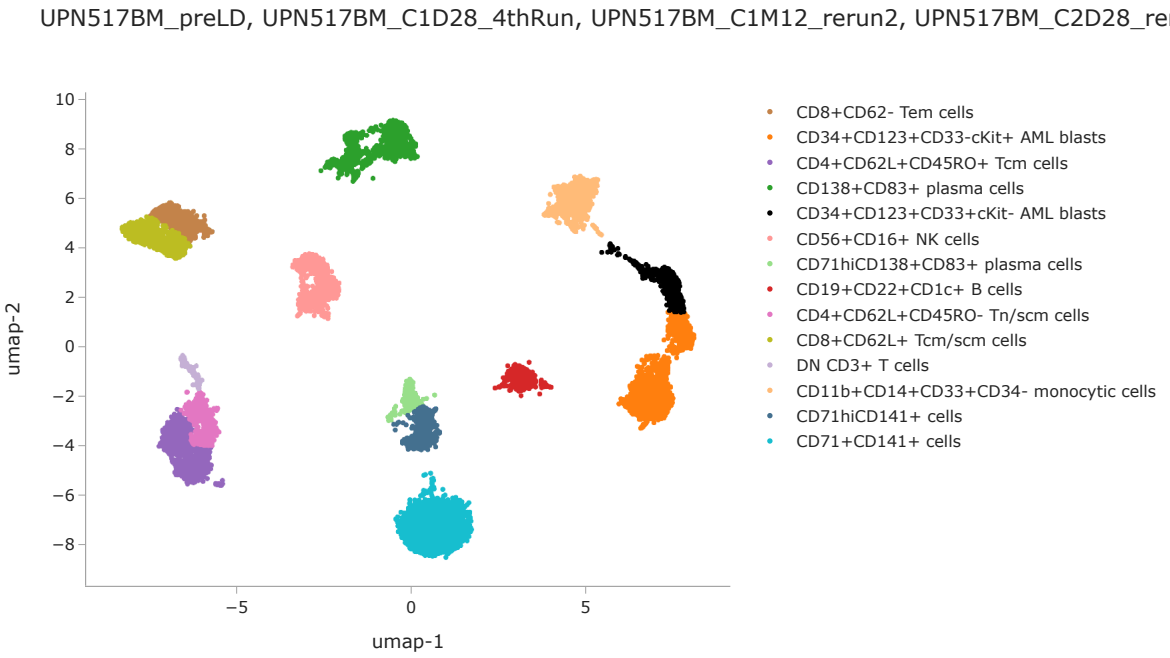


In [ ]:
```
# Relabel clone names acccording to biology, combine clones by assigned identical names
# Just an example! Need to fill this in based on knowledge of the sample and markers
sample.protein.rename_labels(
    {
        '1': 'CD19+CD22+CD1c+ B',
        '2': 'CD62L-CD45RO+CD4+ Tem',
        '3': 'CD45RA+CD62L+CD45RO-CD4+ Tn/scm',
        '4': 'CD62L+CD45RO+CD8+ Tcm',
        '5': 'CD56+CD16+CD3- NK',
        '6': 'CD62L+CD45RO+CD4+ Tcm',
        '7': 'CD14+CD64+CD11b+CD33+CD163+ Myelomonocytic' ,
        '8': 'CD45RA+CD62L+CD45RO-CD8+ Tn/scm',
        '9': 'CD56+CD16+CD3- NK',
        '10': 'CD56+CD16+CD3+ NKT' ,
        '11': 'CD138+CD38+CD30+cKit+ B/Plasma cell' ,
        '12': 'CD16+CD11c+CD123+CD33+ Myelomonocytic',
        '13': 'CD138+CD38+CD30+cKit+ B/Plasma cell',
        '14': 'CD138+CD38+CD30+cKit+ B/Plasma cell',
        '15': 'CD45RA+CD62L+CD45RO-CD8+ Tn/scm' ,
        '16': 'CD62L+CD45RO+CD4+ Tcm',
        '17': 'CD16+CD11c+CD123+CD33+ Myelomonocytic',
        '18': 'FP',


          }
)
```

In [156]:
```
# Run heatmap again with new labels
fig = sample.protein.heatmap(attribute='normalized_counts')
fig.layout.width = 1200
go.Figure(fig)
```



In [157]:
```
fig = sample.protein.scatterplot(attribute='umap',colorby='label')
fig.layout.width = 900
go.Figure(fig)
```



In [ ]:
```
# Drop cells (barcodes) that need to be removed
# Instead of overwriting the sample.protein variable, we define new variable called sample.protein2
# Note: if you do not wish to drop any cells, this code will producs an error (ignore)
fp_barcodes2 = sample.protein.barcodes("unknown")
sample.protein2 = sample.protein.drop(fp_barcodes2)
set(sample.protein2.get_labels())
```

In [ ]:
```
# Use only if no barcodes are removed by the above-listed command
sample.protein2 = sample.protein
```

In [ ]:
```
# Additionally, option to remove AOCs from the data
sample.protein2 = sample.protein2.drop(['CD25'])
```

In [158]:
```
# Shape of cleaned up protein data (number of cells, number of AOCs)
sample.protein2.shape
```

Out[158]: (17433, 45)

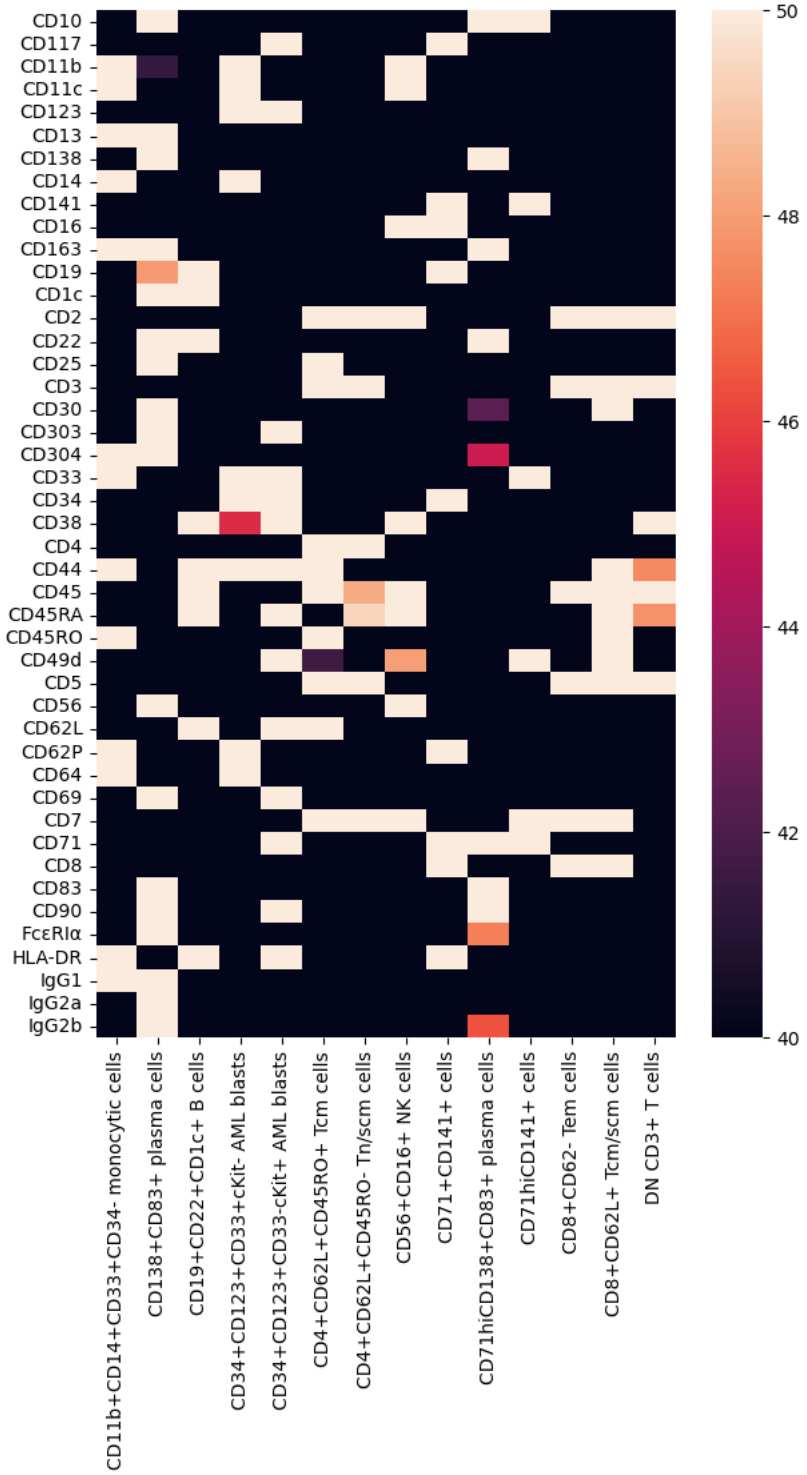
In [ ]:
```
# Re-run heatmap with new filtered data
fig = sample.protein2.heatmap(attribute='normalized_counts')
fig.layout.width = 1200
go.Figure(fig)
```

Statistics

In [159]:
```
pval_protein, tstat_protein = sample.protein2.test_signature("normalized_counts")
```

In [160]:
```
pval_protein = pval_protein + 10 ** -50 + pval_protein
pvals_protein = -np.log10(pval_protein) * (tstat_protein > 0)
```

In [161]:
```
# Colored tiles indicate strong association of a protein with a particular cluster
plt.figure(figsize=(7, 10))
fig = sns.heatmap(pvals_protein.T, vmax=50, vmin=40)
```



Combined visualizations

Topics covered

1. Defining a new sample object by combining the cleaned-up versions of the DNA, CNV and protein data
2. Built-in methods for combined visualizations
3. Additional visualizations with analyte-specific color schemes
4. Quantification of DNA clone and protein cluster overlaps
5. Export and save final data set

Create a new sample object

In [184]:
```
# First, we ensure that all individual analytes are using the same cells.
# Since no cells are removed during CNV analysis, we take the intersect of cells retained in the DNA and protein data.
# In this and the following lines of code, use the final version of each assay,
# which may differ from the defaults listed here (e.g. sample.dna3, sample.protein4, etc.)
x = sample.protein2.barcodes()
y = sample.dna2.barcodes()

z = np.intersect1d(x, y)
len(z)
```

Out[184]: 17433

In [185]:
```
# Subset protein data to include only barcodes that were included in both analytes and store in sample.protein.3 variable
sample.protein3 = sample.protein2[z,sample.protein2.ids()]
```

In [186]:
```
# Subset dna data to include only barcodes that were included in both analytes and store in sample.dna.3 variable
sample.dna3 = sample.dna2[z,sample.dna2.ids()]
```

In [189]:
```
# Subset CNV data to include only barcodes that were included in the protein and dna analysis and store in sample.cnv.3
sample.cnv3 = sample.cnv2[z,sample.cnv2.ids()]
```

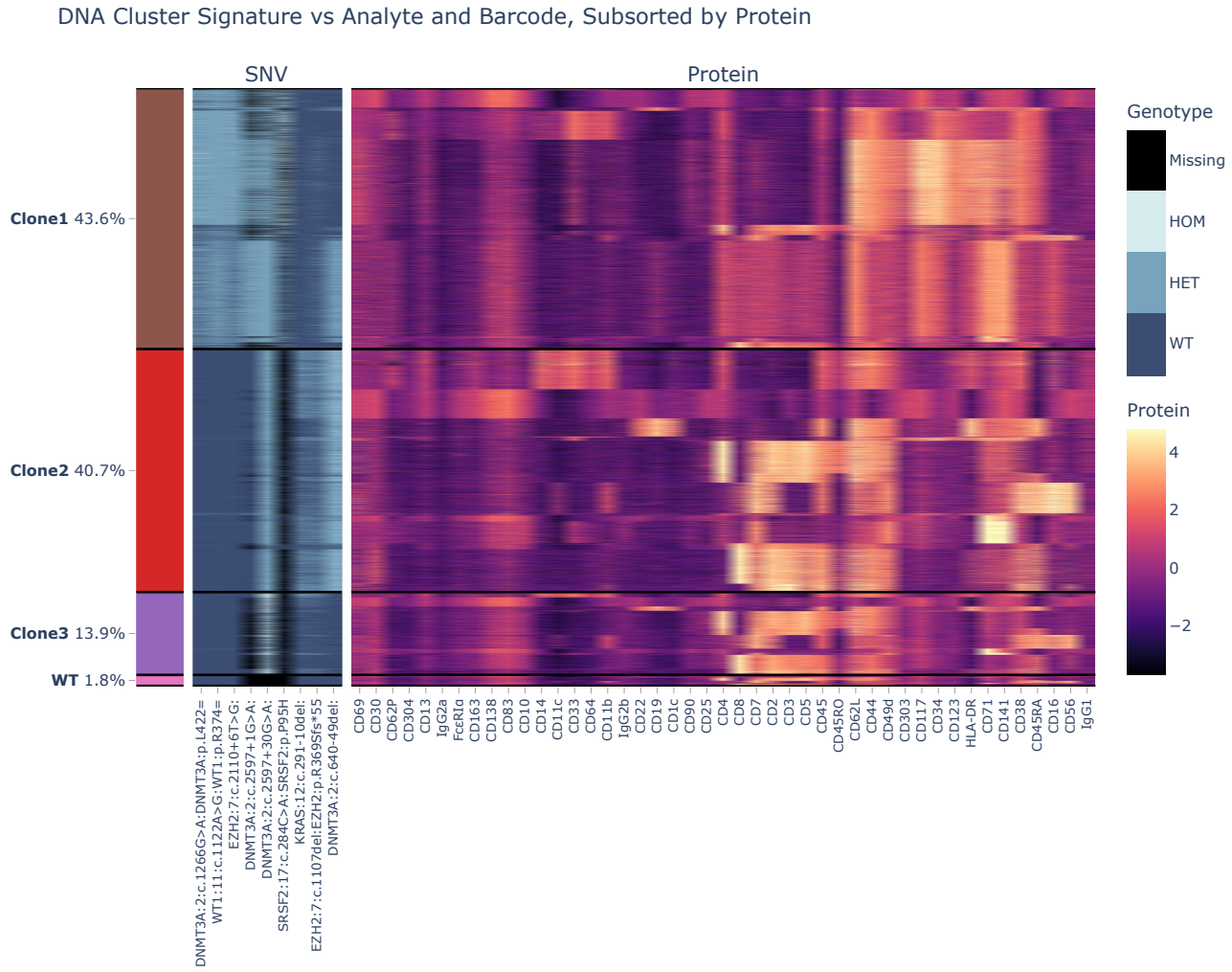
In [190]:
```
#Update each sub-class (e.g., DNA, CNV, Protein) in order to visualize "clean" data only with relevant features and cells.
sample2 = sample
#sub-assays
sample2.dna = sample.dna3
sample2.cnv = sample.cnv3
sample2.protein = sample.protein3
```

In [191]:
```
# Check dimensionality for each subclass; the number of cells (first number) should be the same in each data set
print(sample2.dna.shape,
sample2.cnv.shape,
sample2.protein.shape)
```

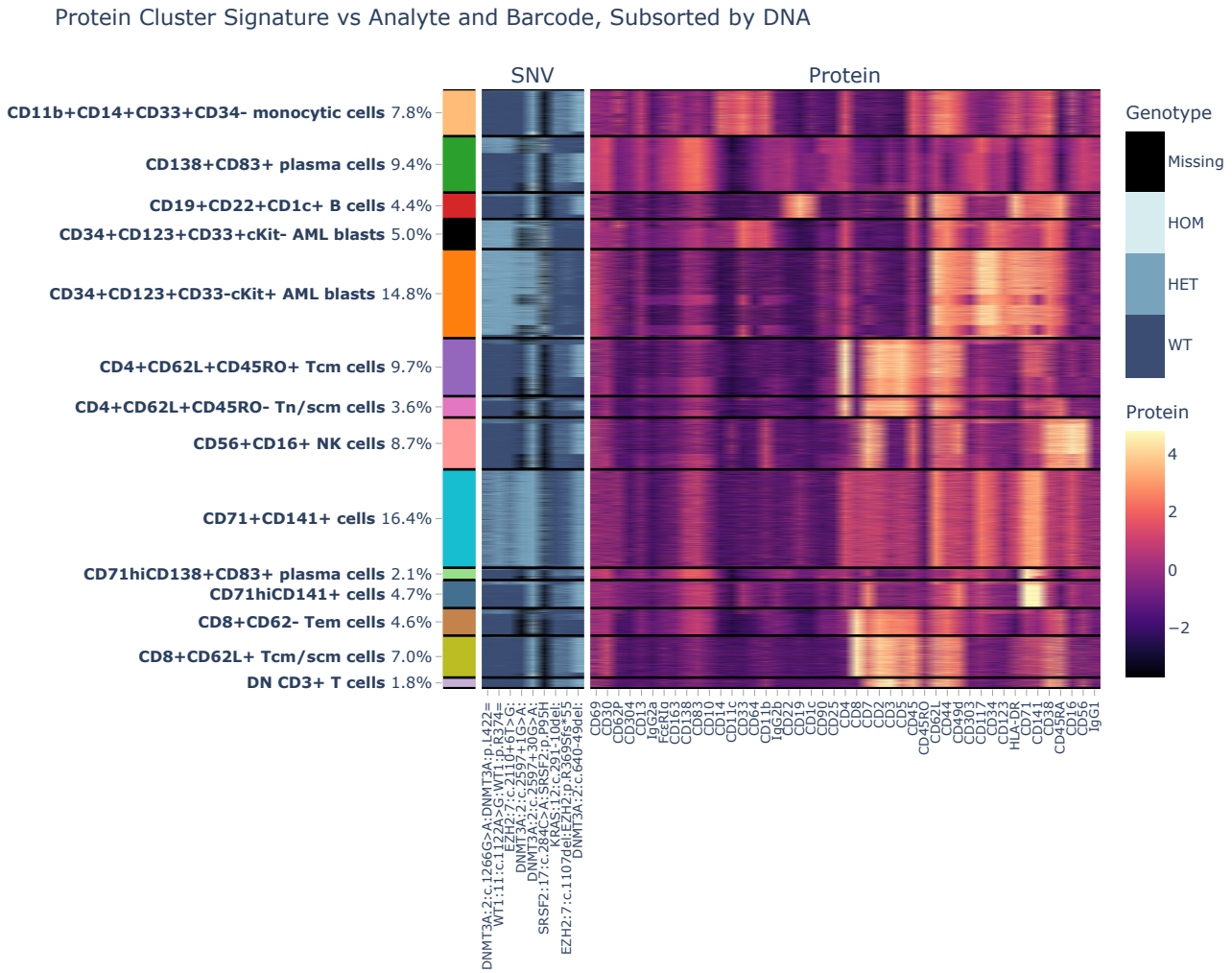
```
(17433, 9) (17433, 127) (17433, 45)
```

Built-in visualizations for combining analytes

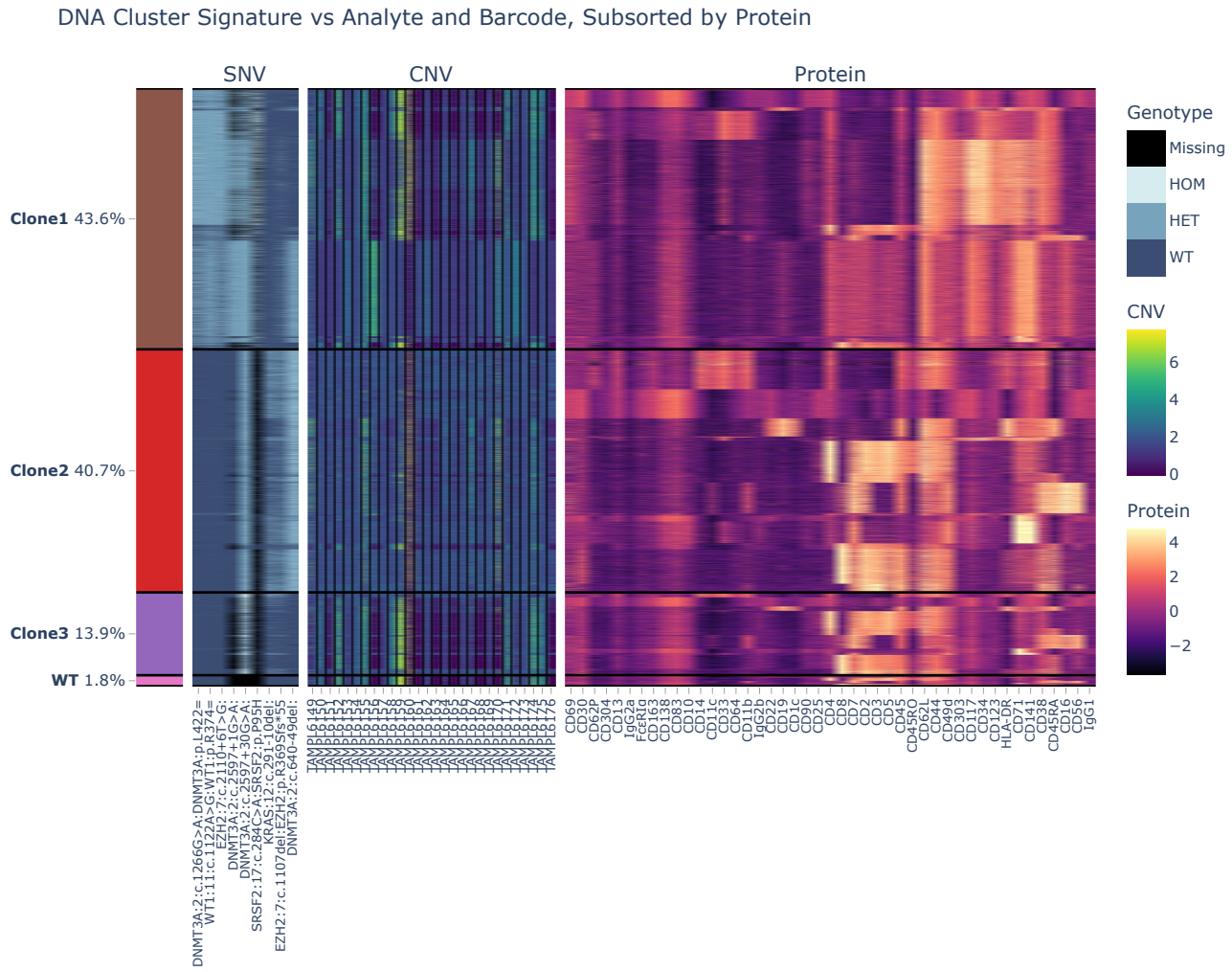
In [192]:
```
# A heatmap presenting the data for up to three analytes
# Note, the default layer in the DNA subclass is NGT, not NGT_FILTERED
# If you wish to use NGT_FILTERED, you must first run the following line (commented out)
# Note that this will erase the NGT layer and replace it with NGT_FILTERED (which includes missing data)
#sample2.dna.layers['NGT'] = sample2.dna.layers['NGT_FILTERED']

sample2.heatmap(clusterby='dna', sortby='protein', drop='cnv', flatten=False)
```



In [193]:
```
sample2.heatmap(clusterby='protein', sortby='dna', drop='cnv', flatten=False)
```



In [197]:
```
# Alternatively, try the following to include the cnv data (but see the note below)
fig = sample2.heatmap(clusterby='dna', sortby='protein', flatten=False)
go.Figure(fig)
```



Often the number of amplicons in the CNV data will take over the heatmap, making the plot uninterpretable. Moreover, there might be certain non-differentiating variants and antibod in the panel. These can be dropped before making the final heatmap.

Note: when subsetting any of the analytes, the assay will "shrink" to contain only those features. To go back to the full data set, you must re-initialize your assay (see below).

In [195]:
```
# Filter the CNV with amplicons only from the relevant genes
genes = sample2.cnv.col_attrs['gene_name'].copy()
relevant_ids = np.isin(genes, ['TET2'])

sample2.cnv = sample2.cnv[:, relevant_ids]
```

In [196]:
```
# Plot again w/ customization (e.g., color code)
fig = sample2.heatmap(clusterby='dna', sortby='cnv',drop='protein', flatten=False)

# Update the width of the plot [See the section on CNV heatmaps]
fig.layout.width = 3000

# Change the CNV colorscale [See the section on CNV heatmaps]
fig.data[2].zmax = 2
fig.data[2].zmin = 0
fig.data[2].colorscale = 'magma'

# Updating the ticktexts to show the gene names instead
fig.layout.xaxis3.ticktext = sample2.cnv.col_attrs['gene_name'].copy()

# Show as a static plot
mutils.static_fig(fig, figsize=(20, 20))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[196], line 16
     13 fig.layout.xaxis3.ticktext = sample2.cnv.col_attrs['gene_name'].copy()
     15 # Show as a static plot
---> 16 mutils.static_fig(fig, figsize=(20, 20))

File ~\anaconda3 2023\envs\mosaic\lib\site-packages\missionbio\mosaic\plotting.py:54, in require_seaborn.<locals>.wrapper(*arg
s, **kwargs)
     50 @functools.wraps(f)
     51 def wrapper(*args, **kwargs):
     52     check_seaborn_imported()
---> 54     return f(*args, **kwargs)

File ~\anaconda3 2023\envs\mosaic\lib\site-packages\missionbio\mosaic\utils.py:39, in static_fig(fig, figsize, scale, ax)
     18 @require_seaborn
     19 def static_fig(fig: go.Figure, figsize: tuple = (7, 7), scale: float = 3, ax: Any = None) -> Any:
     20     """
     21     Convert plotly figure to matplotlib.
     22
   (...)
     36         The axis to show the image on.
     37     """
---> 39     i = fig.to_image(format="png", scale=scale)
     40     i = io.BytesIO(i)
     41     i = mpimg.imread(i, format="png")

File ~\anaconda3 2023\envs\mosaic\lib\site-packages\plotly\basedatatypes.py:3777, in BaseFigure.to_image(self, *args, **kwargs)
   3722 """
   3723 Convert a figure to a static image bytes string
   3724
   (...)
   3773     The image data
   3774 """
   3775 import plotly.io as pio
-> 3777 return pio.to_image(self, *args, **kwargs)

File ~\anaconda3 2023\envs\mosaic\lib\site-packages\plotly\io\_kaleido.py:144, in to_image(fig, format, width, height, scale, v
alidate, engine)
    141 # Validate figure
    142 # ---------------
    143 fig_dict = validate_coerce_fig_to_dict(fig, validate)
--> 144 img_bytes = scope.transform(
    145     fig_dict, format=format, width=width, height=height, scale=scale
    146 )
    148 return img_bytes

File ~\anaconda3 2023\envs\mosaic\lib\site-packages\kaleido\scopes\plotly.py:153, in PlotlyScope.transform(self, figure, forma
t, width, height, scale)
    142     raise ValueError(
    143         "Invalid format '{original_format}'.\n"
    144         "    Supported formats: {supported_formats_str}"
   (...)
    148         )
    149     )
    151 # Transform in using _perform_transform rather than superclass so we can access the full
    152 # response dict, including error codes.
--> 153 response = self._perform_transform(
    154     figure, format=format, width=width, height=height, scale=scale
    155 )
    157 # Check for export error, later can customize error messages for plotly Python users
    158 code = response.get("code", 0)

File ~\anaconda3 2023\envs\mosaic\lib\site-packages\kaleido\scopes\base.py:293, in BaseScope._perform_transform(self, data, **k
wargs)
    284 """
    285 Transform input data using the current scope, returning dict response with error code
    286 whether successful or not.
   (...)
    290 :return: Dict of response from Kaleido executable, whether successful or not
    291 """
    292 # Ensure that kaleido subprocess is running
--> 293 self._ensure_kaleido()
    295 # Perform export
    296 export_spec = self._json_dumps(dict(kwargs, data=data)).encode('utf-8')

File ~\anaconda3 2023\envs\mosaic\lib\site-packages\kaleido\scopes\base.py:198, in BaseScope._ensure_kaleido(self)
    193 if not startup_response_string:
    194     message = (
    195         "Failed to start Kaleido subprocess. Error stream:\n\n" +
    196         self._get_decoded_std_error()
    197     )
--> 198     raise ValueError(message)
    199 else:
    200     startup_response = json.loads(startup_response_string)
```

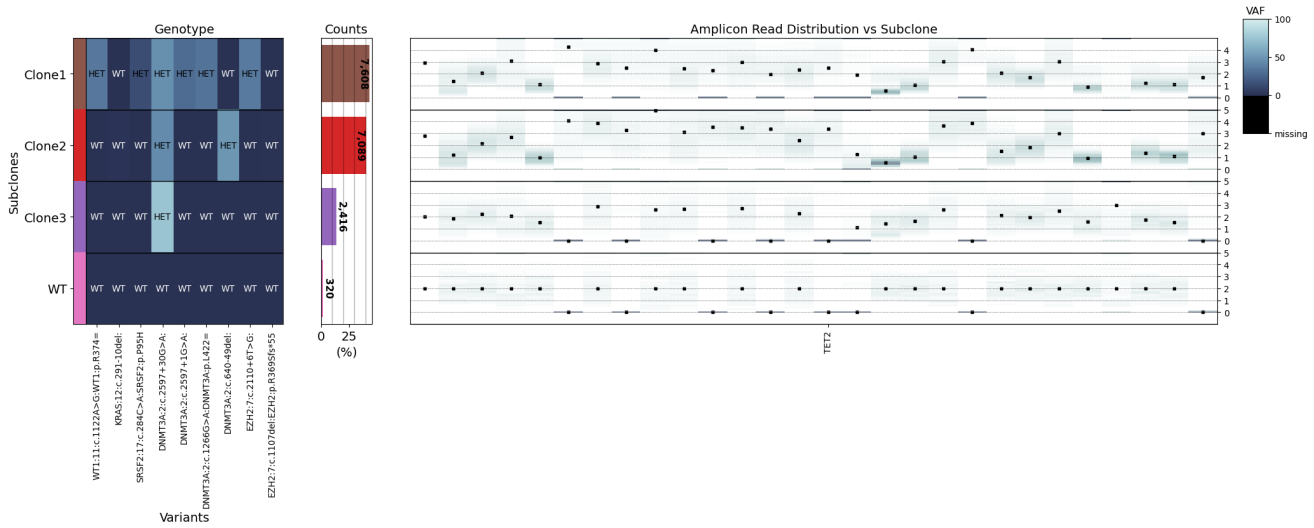
```
ValueError: Failed to start Kaleido subprocess. Error stream:

'C:/Users/smile/anaconda3' is not recognized as an internal or external command,
operable program or batch file.
```

In [ ]:
```
# To reset the CNV data (include all filtered amplicons instead of this new subset), simply re-initialize
sample2.cnv = sample.cnv2
```

In [ ]:
```
help(sample2.clone_vs_analyte)
```

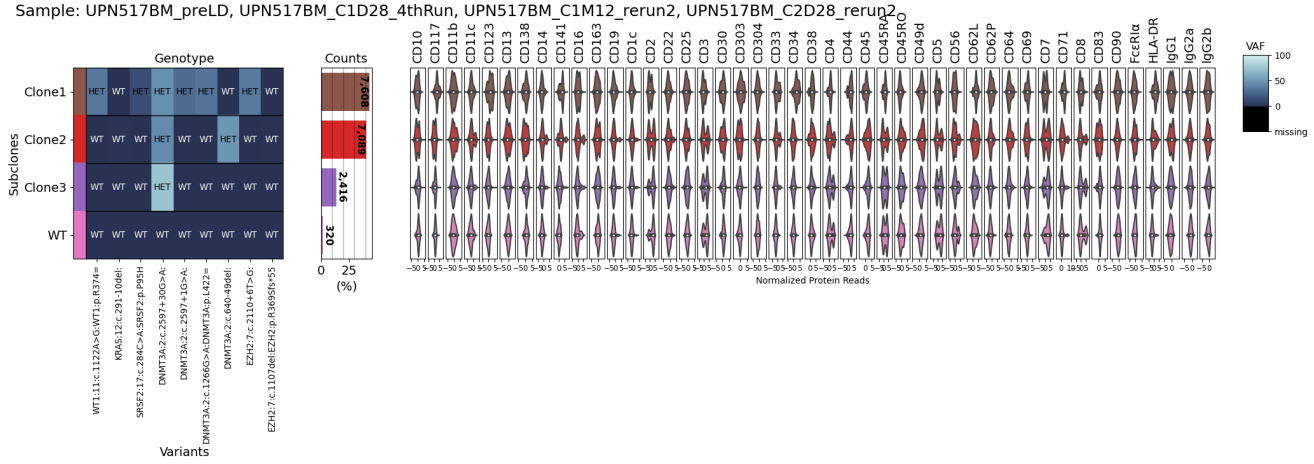
In [198]:
```
# Visualize the CNV data stratified by clone
fig = sample2.clone_vs_analyte('cnv')
fig
#fig.savefig('genotype_cnv.png')
```

Sample: UPN517BM_preLD, UPN517BM_C1D28_4thRun, UPN517BM_C1M12_rerun2, UPN517BM_C2D28_rerun2

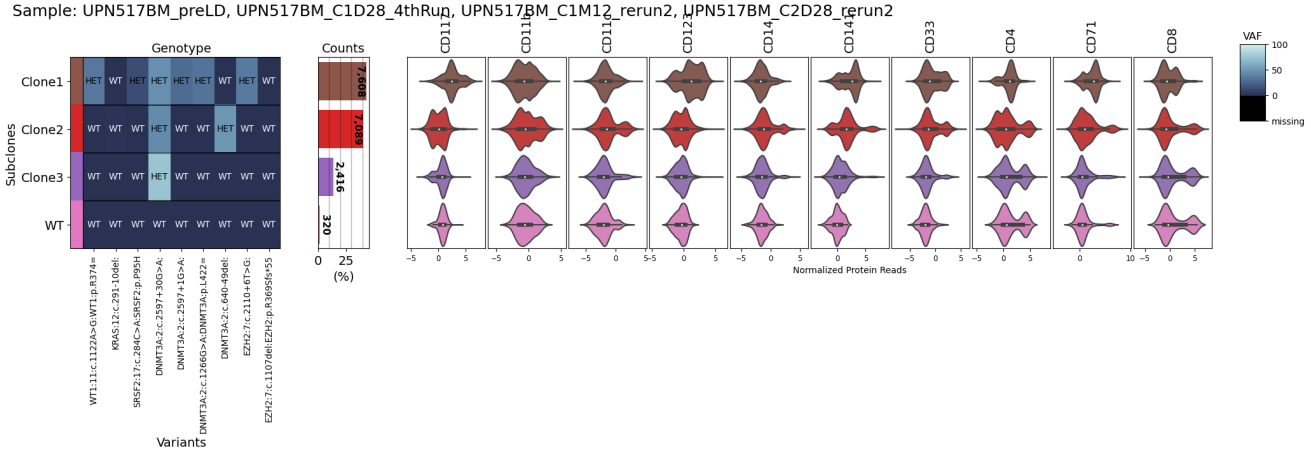


In [199]:
```
# Visualize the protein data (violin plots) stratified by clone
fig = sample2.clone_vs_analyte('protein')
fig
```

Sample: UPN517BM_preLD, UPN517BM_C1D28_4thRun, UPN517BM_C1M12_rerun2, UPN517BM_C2D28_rerun2



In [208]:
```
# To review a subset of features, first subset the assay itself
sample2.protein = sample2.protein[:, ['CD4','CD8','CD123','CD33','CD11b','CD11c','CD14','CD117','CD71','CD141']]
sample2.clone_vs_analyte('protein')
```

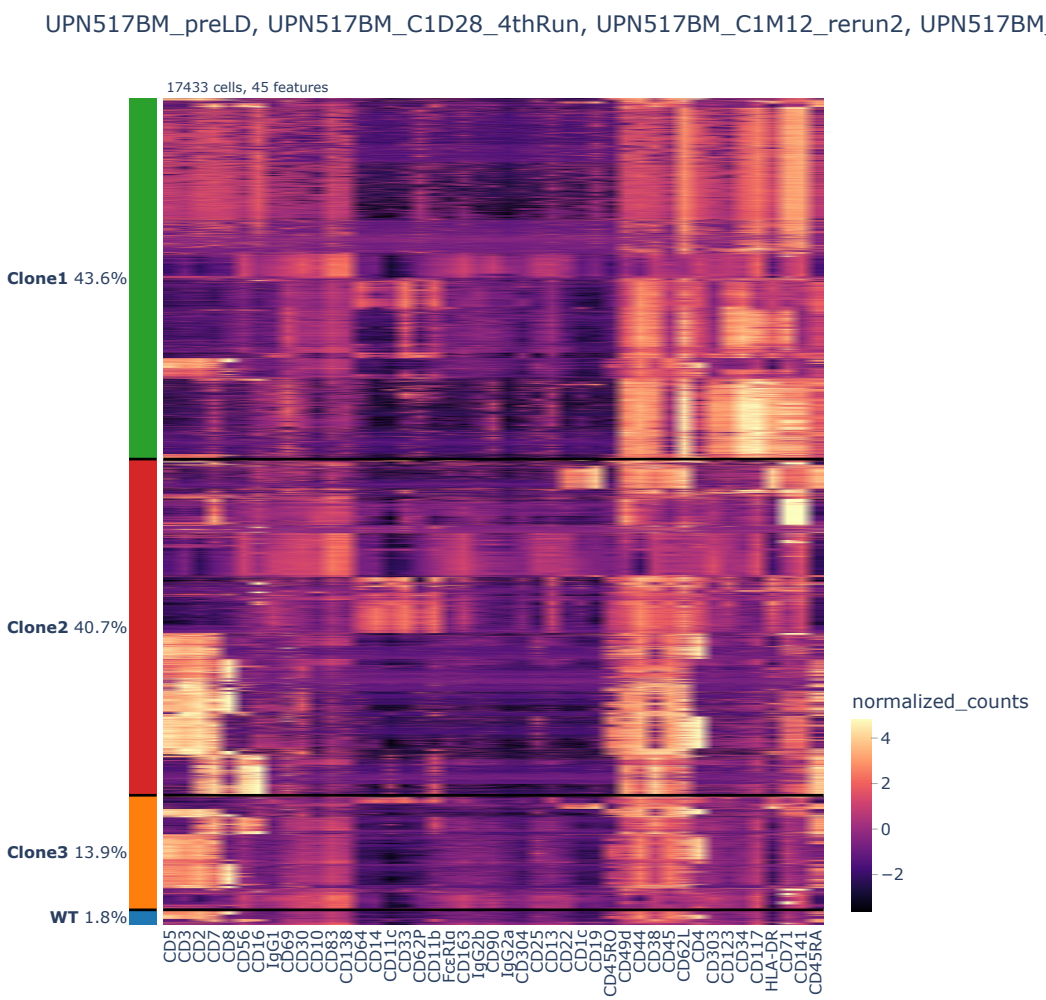


In [209]:
```
# To reset the Protein data (include all AOCs instead of this new subset), simply re-initialize
sample2.protein = sample.protein3
```

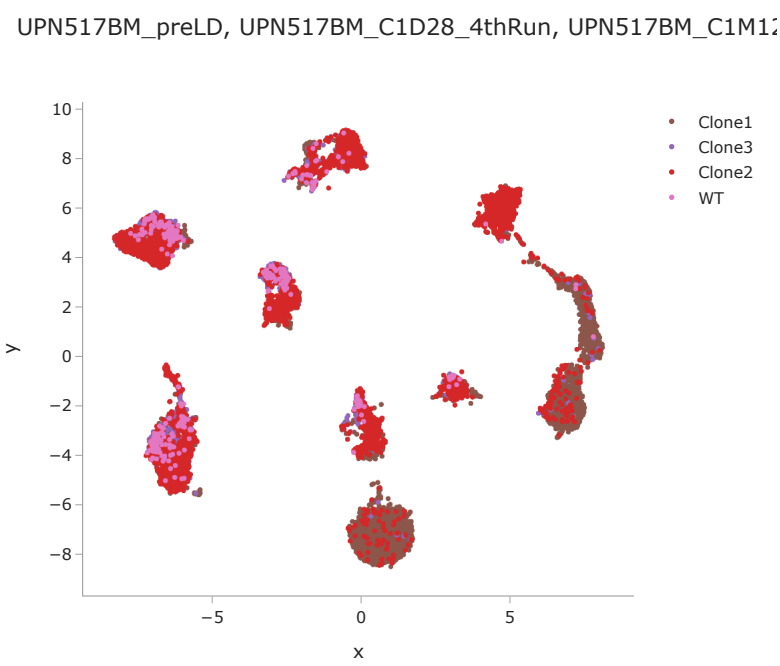
Additional visualizations

In addition to the built-in methods above, the analytes can be intersected by re-plotting some of the single-analyte visualizations, but coloring them by the labels of a different analyte.

In [210]:
```
# Plot the protein heatmap with the DNA clone labels
fig = sample2.protein.heatmap(attribute='normalized_counts',splitby=sample2.dna.row_attrs['label'])
go.Figure(fig)
```



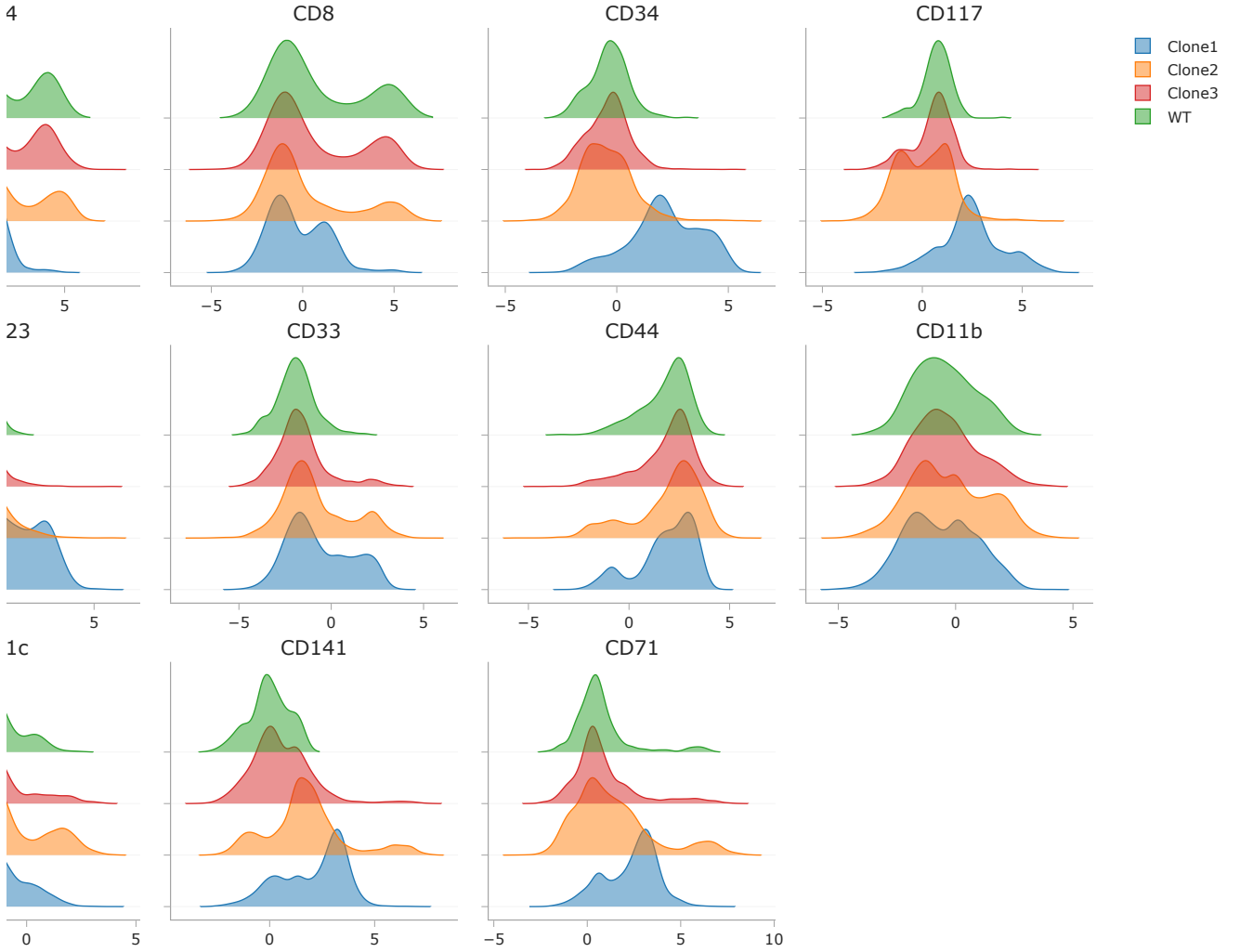
In [211]:
```
# Plot the protein UMAP and color with the DNA clone labels
protein_umap = sample2.protein.row_attrs["umap"]
fig=sample2.dna.scatterplot(attribute=protein_umap, colorby="label")
go.Figure(fig)
```



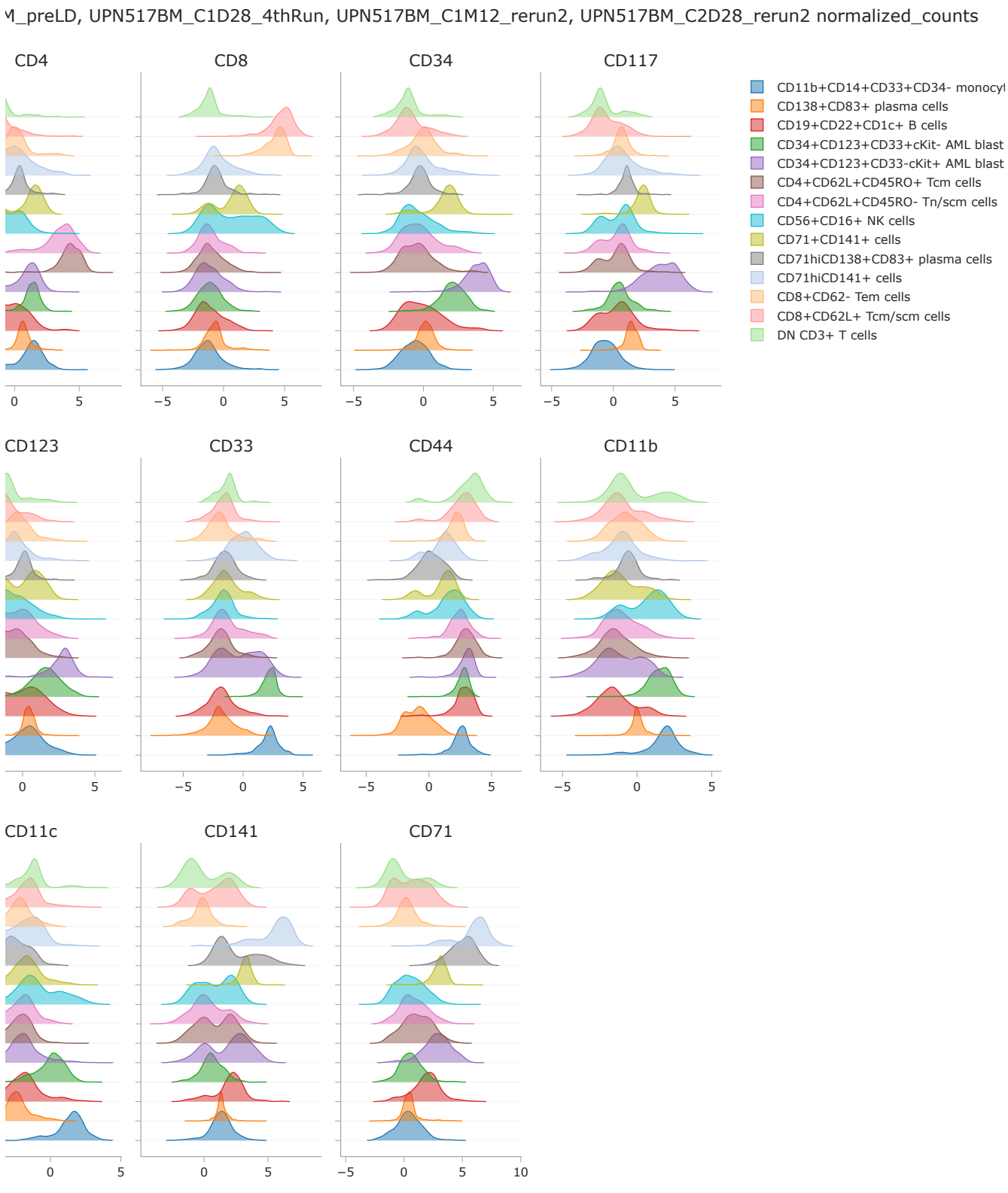
In [ ]:
```
# Plot the protein UMAP and color with variant layers (e.g., VAFs) for specific variants
protein_umap = sample2.protein.row_attrs["umap"]
feats = sample2.dna.ids()[:10] # plot the first 3 variants
sample2.dna.scatterplot(attribute=protein_umap, colorby="NGT", features=feats)
```

In [212]:
```
# Plot the AOC ridge plots split by DNA clone
sample2.protein.ridgeplot(attribute='normalized_counts',
                          features=['CD4','CD8', 'CD34','CD117','CD123','CD33','CD44','CD11b','CD11c','CD141','CD71'],
                          splitby=sample2.dna.row_attrs['label'])
```

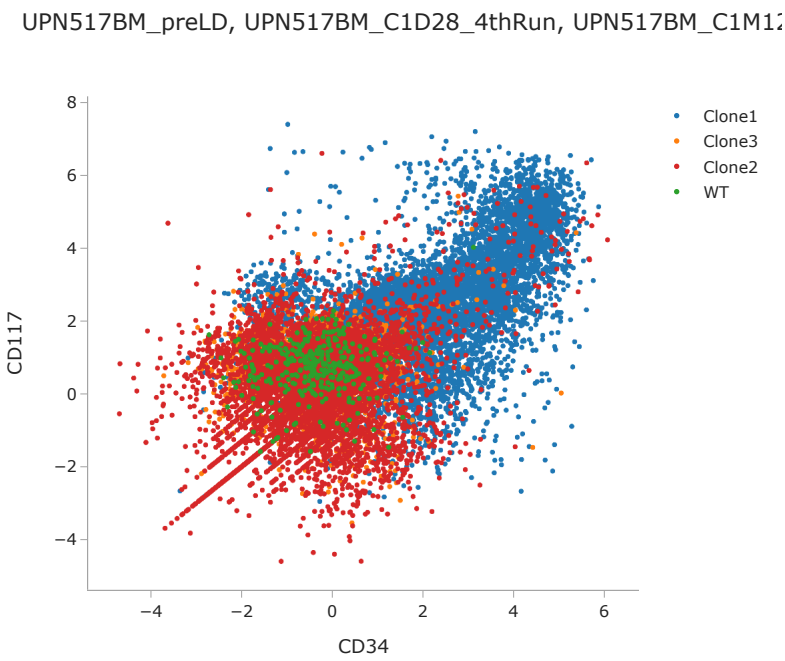
D, UPN517BM_C1D28_4thRun, UPN517BM_C1M12_rerun2, UPN517BM_C2D28_rerun2 normalized_counts



In [213]:
```
# Plot the AOC ridge plots split by protein cluster
sample2.protein.ridgeplot(attribute='normalized_counts',
                          features=['CD4','CD8', 'CD34','CD117','CD123','CD33','CD44','CD11b','CD11c','CD141','CD71'],
                          splitby=sample2.protein.row_attrs['label'])
```



In [217]:
```
# Plot the biaxial AOC plot split by DNA clone
fig = sample2.protein.feature_scatter(layer='normalized_counts', ids=['CD34', 'CD117'], colorby = sample2.dna.row_attrs['label'])
go.Figure(fig)
```

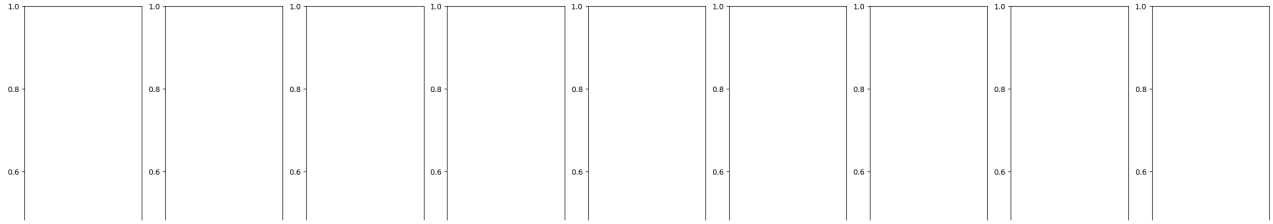


In [218]:
```
sample2.dna.ids()
```

Out[218]:
```
array(['WT1:11:c.1122A>G:WT1:p.R374=', 'KRAS:12:c.291-10del:',
       'SRSF2:17:c.284C>A:SRSF2:p.P95H', 'DNMT3A:2:c.2597+30G>A:',
       'DNMT3A:2:c.2597+1G>A:', 'DNMT3A:2:c.1266G>A:DNMT3A:p.L422=',
       'DNMT3A:2:c.640-49del:', 'EZH2:7:c.2110+6T>G:',
       'EZH2:7:c.1107del:EZH2:p.R369Sfs*55'], dtype=object)
```

In [219]:
```
# Plot the biaxial AOC plot colored by genotype (or AF_MISSING, etc.) for specific variants
variants = [  'WT1:11:c.1122A>G:WT1:p.R374=', 'KRAS:12:c.291-10del:',
       'SRSF2:17:c.284C>A:SRSF2:p.P95H', 'DNMT3A:2:c.2597+30G>A:',
       'DNMT3A:2:c.2597+1G>A:', 'DNMT3A:2:c.1266G>A:DNMT3A:p.L422=',
       'DNMT3A:2:c.640-49del:', 'EZH2:7:c.2110+6T>G:',
       'EZH2:7:c.1107del:EZH2:p.R369Sfs*55']
layer = 'AF_MISSING'
AOCs = ['CD34', 'CD123'] # only two!

# Set the layout and size of the resulting plots (rows, columns) - may need to adjust based on number of variants
fig, axs = plt.subplots(1, len(variants), figsize=(30, 10))

# Generate plots for each variant
counter = 0
for variant in variants:
    color_variant = sample2.dna.layers[layer][:,sample2.dna.ids() == variant]
    fig = sample2.protein.feature_scatter(layer='normalized_counts', ids=AOCs, colorby = color_variant, title=("\t" + variant))
    mutils.static_fig(fig, ax=axs[counter])
    counter = counter + 1

# Show the plot
for i in range(0, counter):
    axs[i].axis('off')

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

```
    199 else:
    200     startup_response = json.loads(startup_response_string)

ValueError: Failed to start Kaleido subprocess. Error stream:

'C:/Users/smile/anaconda3' is not recognized as an internal or external command,
operable program or batch file.
```



Quantify overlaps between DNA clones and protein clusters

In [220]:
```
# Count total cells in each cluster (cell type) and stratify by clone
from collections import Counter
all_cells = sample2.protein.row_attrs['label']
print("Abundance in all clones")
for i in set(all_cells):
    x = Counter(all_cells)[i]
    y = round((x/len(all_cells))*100,2)
    print("    ", i, x, "  ",y, "%")

for i in set(sample2.dna.row_attrs['label']):
    names = sample2.dna.row_attrs['label']
    print("\nAbundance in", i)
    subset = [all_cells[a] for a in range(0, len(all_cells)) if names[a] == i]
    for j in set(subset):
        x = Counter(subset)[j]
        y = round((x/len(subset))*100,2)
        print("    ", j, x, "  ",y, "%")
```

```
Abundance in all clones
     DN CD3+ T cells 311    1.78 %
     CD34+CD123+CD33-cKit+ AML blasts 2588    14.85 %
     CD11b+CD14+CD33+CD34- monocytic cells 1365    7.83 %
     CD34+CD123+CD33+cKit- AML blasts 879    5.04 %
     CD8+CD62L+ Tcm/scm cells 1221    7.0 %
     CD4+CD62L+CD45RO+ Tcm cells 1690    9.69 %
     CD8+CD62- Tem cells 801    4.59 %
     CD138+CD83+ plasma cells 1640    9.41 %
     CD71+CD141+ cells 2863    16.42 %
     CD56+CD16+ NK cells 1510    8.66 %
     CD71hiCD141+ cells 816    4.68 %
     CD71hiCD138+CD83+ plasma cells 358    2.05 %
     CD19+CD22+CD1c+ B cells 771    4.42 %
     CD4+CD62L+CD45RO- Tn/scm cells 620    3.56 %

Abundance in Clone2
     DN CD3+ T cells 261    3.68 %
     CD34+CD123+CD33-cKit+ AML blasts 94    1.33 %
     CD11b+CD14+CD33+CD34- monocytic cells 1166    16.45 %
     CD34+CD123+CD33+cKit- AML blasts 33    0.47 %
     CD8+CD62L+ Tcm/scm cells 1004    14.16 %
     CD4+CD62L+CD45RO+ Tcm cells 947    13.36 %
     CD8+CD62- Tem cells 172    2.43 %
     CD138+CD83+ plasma cells 843    11.89 %
     CD71+CD141+ cells 70    0.99 %
     CD56+CD16+ NK cells 888    12.53 %
     CD71hiCD141+ cells 638    9.0 %
     CD71hiCD138+CD83+ plasma cells 172    2.43 %
     CD19+CD22+CD1c+ B cells 529    7.46 %
     CD4+CD62L+CD45RO- Tn/scm cells 272    3.84 %

Abundance in Clone1
     DN CD3+ T cells 6    0.08 %
     CD34+CD123+CD33-cKit+ AML blasts 2482    32.62 %
     CD11b+CD14+CD33+CD34- monocytic cells 61    0.8 %
     CD34+CD123+CD33+cKit- AML blasts 838    11.01 %
     CD8+CD62L+ Tcm/scm cells 49    0.64 %
     CD4+CD62L+CD45RO+ Tcm cells 175    2.3 %
     CD8+CD62- Tem cells 155    2.04 %
     CD138+CD83+ plasma cells 495    6.51 %
     CD71+CD141+ cells 2788    36.65 %
     CD56+CD16+ NK cells 163    2.14 %
     CD71hiCD141+ cells 117    1.54 %
     CD71hiCD138+CD83+ plasma cells 50    0.66 %
     CD19+CD22+CD1c+ B cells 107    1.41 %
     CD4+CD62L+CD45RO- Tn/scm cells 122    1.6 %

Abundance in WT
     DN CD3+ T cells 2    0.62 %
     CD34+CD123+CD33-cKit+ AML blasts 1    0.31 %
     CD11b+CD14+CD33+CD34- monocytic cells 2    0.62 %
     CD34+CD123+CD33+cKit- AML blasts 2    0.62 %
     CD8+CD62L+ Tcm/scm cells 11    3.44 %
     CD4+CD62L+CD45RO+ Tcm cells 71    22.19 %
     CD8+CD62- Tem cells 68    21.25 %
     CD138+CD83+ plasma cells 37    11.56 %
     CD56+CD16+ NK cells 66    20.62 %
     CD71hiCD141+ cells 2    0.62 %
     CD71hiCD138+CD83+ plasma cells 17    5.31 %
     CD19+CD22+CD1c+ B cells 12    3.75 %
     CD4+CD62L+CD45RO- Tn/scm cells 29    9.06 %

Abundance in Clone3
     DN CD3+ T cells 42    1.74 %
     CD34+CD123+CD33-cKit+ AML blasts 11    0.46 %
     CD11b+CD14+CD33+CD34- monocytic cells 136    5.63 %
     CD34+CD123+CD33+cKit- AML blasts 6    0.25 %
     CD8+CD62L+ Tcm/scm cells 157    6.5 %
     CD4+CD62L+CD45RO+ Tcm cells 497    20.57 %
     CD8+CD62- Tem cells 406    16.8 %
     CD138+CD83+ plasma cells 265    10.97 %
     CD71+CD141+ cells 5    0.21 %
     CD56+CD16+ NK cells 393    16.27 %
     CD71hiCD141+ cells 59    2.44 %
     CD71hiCD138+CD83+ plasma cells 119    4.93 %
     CD19+CD22+CD1c+ B cells 123    5.09 %
     CD4+CD62L+CD45RO- Tn/scm cells 197    8.15 %
```

In [221]:
```
# For each cell type, measure abundance of each DNA clone
all_cells = sample2.protein.row_attrs['label']
for i in set(all_cells):
    names = sample2.dna.row_attrs['label']
    print("\nClonal abundance in", i)
    subset = [names[a] for a in range(0, len(names)) if all_cells[a] == i]
    for j in set(subset):
        x = Counter(subset)[j]
        y = round((x/len(subset))*100,2)
        print("    ", j, x, "  ",y, "%")
```

```
Clonal abundance in DN CD3+ T cells
     Clone2 261    83.92 %
     Clone1 6    1.93 %
     WT 2    0.64 %
     Clone3 42    13.5 %

Clonal abundance in CD34+CD123+CD33-cKit+ AML blasts
     Clone2 94    3.63 %
     Clone1 2482    95.9 %
     WT 1    0.04 %
     Clone3 11    0.43 %

Clonal abundance in CD11b+CD14+CD33+CD34- monocytic cells
     Clone2 1166    85.42 %
     Clone1 61    4.47 %
     WT 2    0.15 %
     Clone3 136    9.96 %

Clonal abundance in CD34+CD123+CD33+cKit- AML blasts
     Clone2 33    3.75 %
     Clone1 838    95.34 %
     WT 2    0.23 %
     Clone3 6    0.68 %

Clonal abundance in CD8+CD62L+ Tcm/scm cells
     Clone2 1004    82.23 %
     Clone1 49    4.01 %
     WT 11    0.9 %
     Clone3 157    12.86 %

Clonal abundance in CD4+CD62L+CD45RO+ Tcm cells
     Clone2 947    56.04 %
     Clone1 175    10.36 %
     WT 71    4.2 %
     Clone3 497    29.41 %

Clonal abundance in CD8+CD62- Tem cells
     Clone2 172    21.47 %
     Clone1 155    19.35 %
     WT 68    8.49 %
     Clone3 406    50.69 %

Clonal abundance in CD138+CD83+ plasma cells
     Clone2 843    51.4 %
     Clone1 495    30.18 %
     WT 37    2.26 %
     Clone3 265    16.16 %

Clonal abundance in CD71+CD141+ cells
     Clone2 70    2.44 %
     Clone1 2788    97.38 %
     Clone3 5    0.17 %

Clonal abundance in CD56+CD16+ NK cells
     Clone2 888    58.81 %
     Clone1 163    10.79 %
     WT 66    4.37 %
     Clone3 393    26.03 %

Clonal abundance in CD71hiCD141+ cells
     Clone2 638    78.19 %
     Clone1 117    14.34 %
     WT 2    0.25 %
     Clone3 59    7.23 %

Clonal abundance in CD71hiCD138+CD83+ plasma cells
     Clone2 172    48.04 %
     Clone1 50    13.97 %
     WT 17    4.75 %
     Clone3 119    33.24 %

Clonal abundance in CD19+CD22+CD1c+ B cells
     Clone2 529    68.61 %
     Clone1 107    13.88 %
     WT 12    1.56 %
     Clone3 123    15.95 %

Clonal abundance in CD4+CD62L+CD45RO- Tn/scm cells
     Clone2 272    43.87 %
     Clone1 122    19.68 %
     WT 29    4.68 %
     Clone3 197    31.77 %
```

In [222]:
```
# If analyzing multiple samples: count total cells in each cluster (cell type) and stratify by sample
all_cells = sample2.protein.row_attrs['label']
print("Abundance in merged data")
for i in set(all_cells):
    x = Counter(all_cells)[i]
    y = round((x/len(all_cells))*100,2)
    print("    ", i, x, "  ",y, "%")

for i in set(sample2.protein.row_attrs['sample_name']):
    names = sample2.protein.row_attrs['sample_name']
    print("\nAbundance in", i)
    subset = [all_cells[a] for a in range(0, len(all_cells)) if names[a] == i]
    for j in set(subset):
        x = Counter(subset)[j]
        y = round((x/len(subset))*100,2)
        print("    ", j, x, "  ",y, "%")
```

```
Abundance in merged data
     DN CD3+ T cells 311    1.78 %
     CD34+CD123+CD33-cKit+ AML blasts 2588    14.85 %
     CD11b+CD14+CD33+CD34- monocytic cells 1365    7.83 %
     CD34+CD123+CD33+cKit- AML blasts 879    5.04 %
     CD8+CD62L+ Tcm/scm cells 1221    7.0 %
     CD4+CD62L+CD45RO+ Tcm cells 1690    9.69 %
     CD8+CD62- Tem cells 801    4.59 %
     CD138+CD83+ plasma cells 1640    9.41 %
     CD71+CD141+ cells 2863    16.42 %
     CD56+CD16+ NK cells 1510    8.66 %
     CD71hiCD141+ cells 816    4.68 %
     CD71hiCD138+CD83+ plasma cells 358    2.05 %
     CD19+CD22+CD1c+ B cells 771    4.42 %
     CD4+CD62L+CD45RO- Tn/scm cells 620    3.56 %

Abundance in UPN517BM_C2D28_rerun2
     DN CD3+ T cells 5    0.2 %
     CD34+CD123+CD33-cKit+ AML blasts 34    1.38 %
     CD11b+CD14+CD33+CD34- monocytic cells 1023    41.65 %
     CD34+CD123+CD33+cKit- AML blasts 13    0.53 %
     CD8+CD62L+ Tcm/scm cells 271    11.03 %
     CD4+CD62L+CD45RO+ Tcm cells 72    2.93 %
     CD8+CD62- Tem cells 21    0.86 %
     CD138+CD83+ plasma cells 570    23.21 %
     CD71+CD141+ cells 5    0.2 %
     CD56+CD16+ NK cells 24    0.98 %
     CD71hiCD141+ cells 292    11.89 %
     CD71hiCD138+CD83+ plasma cells 75    3.05 %
     CD19+CD22+CD1c+ B cells 3    0.12 %
     CD4+CD62L+CD45RO- Tn/scm cells 48    1.95 %

Abundance in UPN517BM_preLD
     DN CD3+ T cells 19    0.39 %
     CD34+CD123+CD33-cKit+ AML blasts 822    16.83 %
     CD11b+CD14+CD33+CD34- monocytic cells 9    0.18 %
     CD34+CD123+CD33+cKit- AML blasts 710    14.54 %
     CD8+CD62L+ Tcm/scm cells 71    1.45 %
     CD4+CD62L+CD45RO+ Tcm cells 665    13.62 %
     CD8+CD62- Tem cells 726    14.87 %
     CD138+CD83+ plasma cells 654    13.39 %
     CD71+CD141+ cells 1    0.02 %
     CD56+CD16+ NK cells 511    10.46 %
     CD71hiCD141+ cells 10    0.2 %
     CD71hiCD138+CD83+ plasma cells 205    4.2 %
     CD19+CD22+CD1c+ B cells 104    2.13 %
     CD4+CD62L+CD45RO- Tn/scm cells 376    7.7 %

Abundance in UPN517BM_C1D28_4thRun
     DN CD3+ T cells 250    12.61 %
     CD34+CD123+CD33-cKit+ AML blasts 53    2.67 %
     CD11b+CD14+CD33+CD34- monocytic cells 231    11.65 %
     CD34+CD123+CD33+cKit- AML blasts 9    0.45 %
     CD8+CD62L+ Tcm/scm cells 443    22.35 %
     CD4+CD62L+CD45RO+ Tcm cells 196    9.89 %
     CD8+CD62- Tem cells 30    1.51 %
     CD138+CD83+ plasma cells 253    12.76 %
     CD71+CD141+ cells 8    0.4 %
     CD56+CD16+ NK cells 270    13.62 %
     CD71hiCD141+ cells 126    6.36 %
     CD71hiCD138+CD83+ plasma cells 49    2.47 %
     CD19+CD22+CD1c+ B cells 14    0.71 %
     CD4+CD62L+CD45RO- Tn/scm cells 50    2.52 %

Abundance in UPN517BM_C1M12_rerun2
     DN CD3+ T cells 37    0.46 %
     CD34+CD123+CD33-cKit+ AML blasts 1679    20.7 %
     CD11b+CD14+CD33+CD34- monocytic cells 102    1.26 %
     CD34+CD123+CD33+cKit- AML blasts 147    1.81 %
     CD8+CD62L+ Tcm/scm cells 436    5.37 %
     CD71+CD141+ cells 2849    35.12 %
     CD4+CD62L+CD45RO+ Tcm cells 757    9.33 %
     CD138+CD83+ plasma cells 163    2.01 %
     CD8+CD62- Tem cells 24    0.3 %
     CD56+CD16+ NK cells 705    8.69 %
     CD71hiCD141+ cells 388    4.78 %
     CD71hiCD138+CD83+ plasma cells 29    0.36 %
     CD19+CD22+CD1c+ B cells 650    8.01 %
     CD4+CD62L+CD45RO- Tn/scm cells 146    1.8 %
```

Export and Save Data

In [223]:
```
# Save new h5 file that includes only the final, cleaned dataset
ms.save(sample2, "UPN517BM_4samples_Filtered.h5")
```

In [ ]:
```
# Export data into csv format
import os

## set directory that does NOT exist
folder_to_save = '/Users/rachelagoglia/Desktop/new-folder'

os.mkdir(folder_to_save)
for assay in [sample2.dna, sample2.cnv, sample2.protein]:
    if assay is not None:
        os.mkdir(f'{folder_to_save}/{assay.name}')
        os.mkdir(f'{folder_to_save}/{assay.name}/layers')
        os.mkdir(f'{folder_to_save}/{assay.name}/rows')

        for layer in assay.layers.keys():
            df = assay.get_attribute(layer, constraint='row+col')
            cols = list(df.columns.values)
            df.loc[:, 'label'] = assay.get_labels()
            df = df.loc[:, ['label'] + cols]
            df.to_csv(f'{folder_to_save}/{assay.name}/layers/{layer}.csv')

        for row in assay.row_attrs.keys():
            df = assay.get_attribute(row, constraint='row')
            cols = list(df.columns.values)
            df.loc[:, 'label'] = assay.get_labels()
            df = df.loc[:, ['label'] + cols]
            df.to_csv(f'{folder_to_save}/{assay.name}/rows/{row}.csv')
```

In [ ]:
```

```