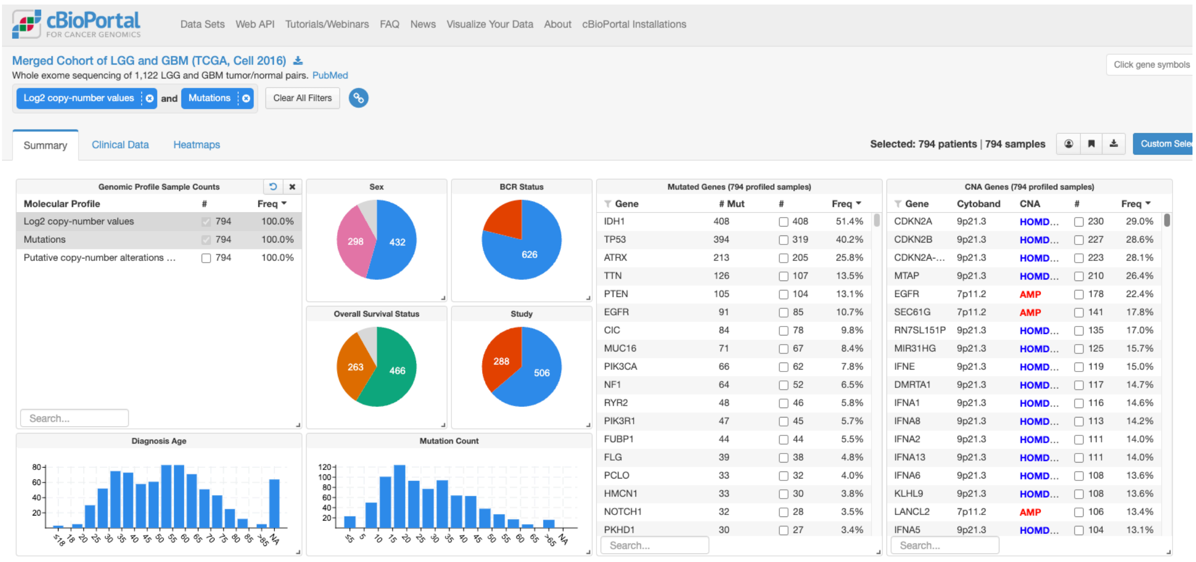
In [1]:
```
import os
os.environ["OMP_NUM_THREADS"] = "1" #the number of threads to use for parallel regions
import flexynesis
import torch
import numpy as np
import seaborn as sns
import pandas as pd
import random
import lightning as pl
```

```
Seed set to 42
```

Finding Survival Markers in Lower Grade Glioma (LGG) and Glioblastoma Multiforme (GBM)seed



Merged Cohort of LGG and GBM (TCGA, Cell 2016)

Whole exome sequencing of 1,122 LGG and GBM tumor/normal pairs. PubMed

Selected: 794 patients | 794 samples

Genomic Profile Sample Counts

Molecular Profile	#	Freq
Log2 copy-number values	794	100.0%
Mutations	794	100.0%
Putative copy-number alterations ...	794	100.0%

Mutated Genes (794 profiled samples)

Gene	# Mut	#	Freq
IDH1	408	408	51.4%
TP53	394	319	40.2%
ATRX	213	205	25.8%
TTN	126	107	13.5%
PTEN	105	104	13.1%
EGFR	91	85	10.7%
CIC	84	78	9.8%
MUC16	71	67	8.4%
PIK3CA	66	62	7.8%
NF1	64	52	6.5%
RYR2	48	46	5.8%
PIK3R1	47	45	5.7%
FUBP1	44	44	5.5%
FLG	39	38	4.8%
PCLO	33	32	4.0%
HMCN1	33	30	3.8%
NOTCH1	32	28	3.5%
PKHD1	30	27	3.4%

CNA Genes (794 profiled samples)

Gene	Cytoband	CNA	#	Freq
CDKN2A	9p21.3	HOMD...	230	29.0%
CDKN2B	9p21.3	HOMD...	227	28.6%
CDKN2A-...	9p21.3	HOMD...	223	28.1%
MTAP	9p21.3	HOMD...	210	26.4%
EGFR	7p11.2	AMP	178	22.4%
SEC61G	7p11.2	AMP	141	17.8%
RN7SL151P	9p21.3	HOMD...	135	17.0%
MIR31HG	9p21.3	HOMD...	125	15.7%
IFNE	9p21.3	HOMD...	119	15.0%
DMRTA1	9p21.3	HOMD...	117	14.7%
IFNA1	9p21.3	HOMD...	116	14.6%
IFNA8	9p21.3	HOMD...	113	14.2%
IFNA2	9p21.3	HOMD...	111	14.0%
IFNA13	9p21.3	HOMD...	111	14.0%
IFNA6	9p21.3	HOMD...	108	13.6%
KLHL9	9p21.3	HOMD...	108	13.6%
LANCL2	7p11.2	AMP	106	13.4%
IFNA5	9p21.3	HOMD...	104	13.1%

Here, we demonstrate the capabilities of `flexynesis` on a multi-omic dataset of 506 Brain Lower Grade Glioma (LGG) and 288 Glioblastoma Multiforme (GBM) samples with matching mutation and copy number alteration data downloaded from the cbioportal. The data was split into `train` (70% of the samples) and `test` (30% of the samples) data folders. The data files were processed to follow the same nomenclature.

- `cna.csv` contains "copy number alteration" data
- `mut.csv` contains "mutation" data, which is a binary matrix of genes versus samples.
- `clin.csv` contains "clinical/sample metatada", which is a table of clinical parameters such as age, sex, disease type, histological diagnosis, and overall survival time and status.

Data Download

The data can be downloaded as follows:

In [2]:
```
if not os.path.exists("lgggbm_tcga_pub_processed"):
    !wget -O lgggbm_tcga_pub_processed.tgz "https://bimsbstatic.mdc-berlin.de/akalin/buyar/flexynesis-benchmark
```

Importing Train and Test Datasets

We import train and test datasets including mutations and copy number alterations. We rank genes by Laplacian Scores and pick top 10% of the genes, while removing highly redundant genes with a correlation score threshold of 0.8 and a variance threshold of 50%. By setting `concatenate` to `False`, we will be doing an `intermediate` fusion of omic layers.

In [3]:
```
data_importer = flexynesis.DataImporter(path ='lgggbm_tcga_pub_processed',
                                        data_types = ['mut', 'cna'], log_transform=False,
                                        concatenate=False, top_percentile=10, min_features=1000, correlation_th
                                       variance_threshold=0.5)
train_dataset, test_dataset = data_importer.import_data()
```

```
[INFO] ================= Importing Data =================
[INFO] Validating data folders...

[INFO] ----------------- Reading Data -----------------
[INFO] Importing lgggbm_tcga_pub_processed/train/mut.csv...
[INFO] Importing lgggbm_tcga_pub_processed/train/clin.csv...
[INFO] Importing lgggbm_tcga_pub_processed/train/cna.csv...

[INFO] ----------------- Reading Data -----------------
[INFO] Importing lgggbm_tcga_pub_processed/test/mut.csv...
[INFO] Importing lgggbm_tcga_pub_processed/test/clin.csv...
[INFO] Importing lgggbm_tcga_pub_processed/test/cna.csv...

[INFO] ----------------- Checking for problems with the input data -----------------
[INFO] Data structure is valid with no errors or warnings.

[INFO] ----------------- Processing Data (train) -----------------

[INFO] ----------------- Cleaning Up Data -----------------

[INFO] working on layer:  mut
[INFO] Number of NA values:  0
[INFO] DataFrame mut - Removed 5561 features.

[INFO] working on layer:  cna
[INFO] Number of NA values:  0
[INFO] DataFrame cna - Removed 12375 features.
[INFO] DataFrame mut - Removed 0 samples (0.00%).
[INFO] DataFrame cna - Removed 0 samples (0.00%).
[INFO] Implementing feature selection using laplacian score for layer: mut with  5503 features  and  556  sample
s
Calculating Laplacian scores: 100%|██████████| 5503/5503 [00:00<00:00, 10851.81it/s]
Filtering redundant features: 100%|██████████| 1000/1000 [00:00<00:00, 23541.29it/s]
[INFO] Implementing feature selection using laplacian score for layer: cna with  12371 features  and  556  sampl
es
Calculating Laplacian scores: 100%|██████████| 12371/12371 [00:01<00:00, 12013.74it/s]
Filtering redundant features: 100%|██████████| 1237/1237 [00:00<00:00, 404360.85it/s]
[INFO] ----------------- Processing Data (test) -----------------

[INFO] ----------------- Cleaning Up Data -----------------

[INFO] working on layer:  mut
[INFO] Number of NA values:  0
[INFO] DataFrame mut - Removed 5627 features.

[INFO] working on layer:  cna
[INFO] Number of NA values:  0
[INFO] DataFrame cna - Removed 12382 features.
[INFO] DataFrame mut - Removed 0 samples (0.00%).
[INFO] DataFrame cna - Removed 0 samples (0.00%).

[INFO] ----------------- Harmonizing Data Sets -----------------

[INFO] ----------------- Finished Harmonizing -----------------

[INFO] ----------------- Normalizing Data -----------------

[INFO] ----------------- Normalizing Data -----------------
[INFO] Training Data Stats:  {'feature_count in: cna': 1237, 'feature_count in: mut': 317, 'sample_count': 556}
[INFO] Test Data Stats:  {'feature_count in: cna': 1237, 'feature_count in: mut': 317, 'sample_count': 238}
[INFO] Merging Feature Logs...
[INFO] Data import successful.
```

1. Exploratory Data Analysis

Before building any machine learning models on the data, it is important to first familiarize yourself with the data you are working with. It is important to know the available data matrices, their sizes/shapes, available clinical variables and how they are distributed.

Below you are asked to do simple explorations of the available data.

1.1 Print the shapes of the available data matrices

- How many features and samples are available per data type in train/test datasets?

In [4]:
```
train_dataset.dat
```

```
Out[4]: {'cna': tensor([[-0.2278,  0.7062,  0.6967,  ...,  0.4075,  0.6497,  0.6497],
                 [-0.2541,  2.1161,  2.2173,  ...,  0.4871,  2.1455,  2.1455],
                 [-0.2199, -0.7857, -0.8194,  ...,  0.4404, -0.8417, -0.8417],
                 ...,
                 [ 0.3641,  1.2924,  1.3131,  ...,  0.4075,  1.2561,  1.2561],
                 [-0.2436,  0.7671,  0.7624,  ...,  0.4267,  0.7144,  0.7144],
                 [ 1.8452, -0.7121, -0.7719,  ...,  0.4322, -0.7949, -0.7949]]),
         'mut': tensor([[-0.1485, -1.0182, -0.1721,  ..., -0.0424, -0.0424, -0.0424],
                 [-0.1485, -1.0182, -0.1721,  ..., -0.0424, -0.0424, -0.0424],
                 [-0.1485,  0.9822, -0.1721,  ..., -0.0424, -0.0424, -0.0424],
                 ...,
                 [-0.1485, -1.0182, -0.1721,  ..., -0.0424, -0.0424, -0.0424],
                 [-0.1485, -1.0182, -0.1721,  ..., -0.0424, -0.0424, -0.0424],
                 [-0.1485,  0.9822, -0.1721,  ..., -0.0424, -0.0424, -0.0424]])}
```

```
In [5]: train_dataset.dat['mut'].shape, train_dataset.dat['cna'].shape
```

```
Out[5]: (torch.Size([556, 317]), torch.Size([556, 1237]))
```

```
In [6]: test_dataset.dat['mut'].shape, test_dataset.dat['cna'].shape
```

```
Out[6]: (torch.Size([238, 317]), torch.Size([238, 1237]))
```

```
In [7]: # sample names and feature names
        train_dataset.samples[:10], train_dataset.features
```

```
Out[7]: (['TCGA-DU-6405',
          'TCGA-06-2564',
          'TCGA-WH-A86K',
          'TCGA-QH-A65X',
          'TCGA-HT-7601',
          'TCGA-P5-A72W',
          'TCGA-41-2572',
          'TCGA-DH-A66F',
          'TCGA-74-6584',
          'TCGA-26-5134'],
         {'cna': Index(['SLC30A8', 'ZNF273', 'OR9A1P', 'AGL', 'KCNA5', 'MIR603', 'SNTB1',
                 'MRPL13', 'MTBP', 'SNORA72|ENSG00000252158.1',
                 ...
                 'CAV1', 'FZD1', 'BCAP29', 'MNX1', 'ADAM22', 'LRP8', 'NOM1', 'RN7SL290P',
                 'CEP41', 'snoU13|ENSG00000239044.1'],
                dtype='object', length=1237),
          'mut': Index(['IDH2', 'IDH1', 'RELN', 'ATRX', 'PIK3CA', 'EGFR', 'TP53', 'COL6A3',
                 'SVIL', 'CIC',
                 ...
                 'ZBTB34', 'OGDH', 'ZNF571', 'NYAP2', 'NEURL1', 'CAMKK1', 'SEPT12',
                 'PTPN6', 'NAGA', 'SMARCA2'],
                dtype='object', length=317)})
```

1.2 Explore sample annotations

- What are the available clinical variables? Are they available in both train and test datasets? (See .ann)

```
In [8]: train_dataset.ann.keys()
```

```
Out[8]: dict_keys(['AGE', 'OS_MONTHS', 'OS_STATUS', 'KARNOFSKY_PERFORMANCE_SCORE', 'STUDY', 'BCR_STATUS', 'HISTOLOGICA
        L_DIAGNOSIS', 'SEX'])
```

```
In [9]: test_dataset.ann.keys()
```

```
Out[9]: dict_keys(['AGE', 'OS_MONTHS', 'OS_STATUS', 'KARNOFSKY_PERFORMANCE_SCORE', 'STUDY', 'BCR_STATUS', 'HISTOLOGICA
        L_DIAGNOSIS', 'SEX'])
```

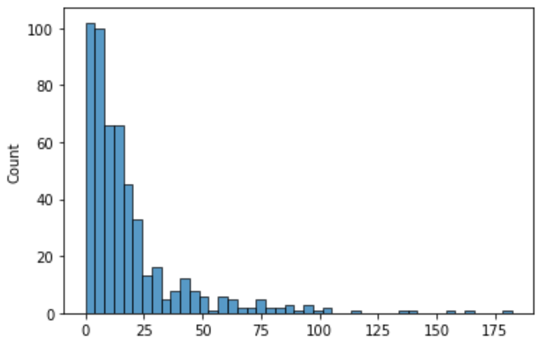
```
In [77]: test_dataset.ann
```

```
Out[77]: {'AGE': tensor([73., 38., 30., 52., 72., 70., nan, 39., 78., 47., 63., 32., 78., 44.,
                 nan, 59., nan, 58., 40., 53., nan, 43., 62., 61., 48., 35., 45., 24.,
                 20., 30., nan, 66., 59., 49., 58., 59., nan, 23., nan, 23., 21., 31.,
                 nan, 63., 50., 52., 35., nan, 35., 19., 59., 34., 38., 48., 52., 36.,
                 79., 79., 74., 53., 54., 39., 74., 73., 47., 22., 54., 63., 66., 57.,
                 50., 65., nan, 52., 57., 66., 63., 61., 73., 56., 33., 71., 52., 29.,
                 nan, 33., 36., 76., 74., 43., 36., 54., 20., 66., 37., 30., 39., 42.,
                 55., 41., 28., nan, 59., 65., 53., 62., nan, 28., 53., 44., 31., 48.,
                 nan, 83., 63., 52., 41., nan, 50., 61., 60., 24., 41., 27., 41., 40.,
                 38., 59., nan, 38., 65., 37., 58., nan, 55., 33., 36., 89., nan, 34.,
                 31., 31., 48., 87., 53., 57., 86., 54., 75., 50., 30., 61., 44., 36.,
                 58., 38., 78., 60., nan, nan, 55., 54., 34., nan, nan, 62., 56., 57.,
                 nan, 64., 62., 41., 56., 29., 44., 56., 50., 72., 24., 35., 31., 59.,
                 nan, 32., 49., 43., 64., 35., 30., 61., 58., 73., 47., 24., nan, 55.,
                 33., 52., nan, 60., 59., 33., 54., 39., 51., 31., 62., 58., 30., 63.,
                 29., 62., 32., nan, 69., nan, 39., nan, 59., 41., 62., 38., 34., 57.,
                 33., 66., 37., 68., 83., 43., 35., 76., 66., 73., 25., 63., 44., 34.],
                dtype=torch.float64),
          'OS_MONTHS': tensor([7.8000e+00, 1.0000e-01, 3.3600e+01, 1.1300e+01, 1.2000e+00, 7.6000e+00,
                        nan, 6.8000e+00, 1.9000e+00, 1.6100e+01, 1.7500e+01, 3.1800e+01,
                 2.7000e+00, 1.5000e+00,        nan, 3.1700e+01,        nan, 4.7000e+00,
                 1.7400e+01, 1.4200e+01,        nan, 2.1000e+01, 1.5800e+01, 7.8200e+01,
                 2.3800e+01, 1.2440e+02, 4.7000e+00, 1.6100e+01, 1.5400e+01, 9.4000e+00,
                        nan, 2.5000e+00, 1.5000e+01, 2.5300e+01, 9.3000e+00, 6.8000e+00,
                        nan, 2.0000e-01,        nan, 2.5500e+01, 5.0500e+01, 7.5100e+01,
                        nan, 2.6700e+01, 1.0600e+01, 2.0000e-01, 1.8200e+01,        nan,
                 7.7900e+01, 1.2000e+01, 1.1500e+01, 7.9900e+01, 8.3000e+00, 4.2000e+00,
                 3.7000e+00, 8.3600e+01, 2.9000e+00, 3.1000e+00, 1.0000e-01, 1.2600e+01,
                 1.3500e+01, 2.1600e+01, 5.2000e+00, 1.9000e+00, 7.0000e+00, 2.0000e-01,
                 3.4300e+01, 1.0000e-01, 4.8000e+00, 5.8700e+01, 2.7000e+00, 7.5000e+00,
                        nan, 1.5600e+01, 1.2900e+01, 1.5000e+01, 1.7100e+01, 1.0400e+01,
                 3.7000e+00, 8.0000e+00, 3.5000e+00, 1.1300e+01, 1.0900e+01, 3.0300e+01,
                        nan, 1.0000e-01, 7.6000e+00, 7.4000e+00, 7.2000e+00, 4.9400e+01,
                 4.6000e+00, 6.2000e+00, 1.0000e-01, 2.5000e+00, 7.4000e+00, 1.6000e+01,
                 1.2600e+01, 4.1100e+01, 1.5400e+01, 4.8400e+01, 9.9000e+00,        nan,
                 5.4200e+01, 2.3000e+00, 1.0800e+01, 3.4300e+01,        nan, 1.6500e+01,
                 2.2400e+01, 7.7000e+00, 1.8900e+01, 1.7900e+01,        nan, 1.8000e+00,
                 3.2000e+00, 1.7500e+01, 1.8400e+01,        nan, 2.2500e+01, 8.0000e-01,
                 5.7000e+00, 2.0000e-01, 5.0100e+01, 0.0000e+00, 1.5610e+02, 2.1500e+01,
                 1.3070e+02, 1.1500e+01,        nan, 1.8900e+01, 1.6000e+00, 1.8100e+01,
                 7.9000e+00,        nan, 9.7000e+00, 4.1000e+00, 1.4000e+01, 9.0000e-01,
                        nan, 7.2500e+01, 4.7000e+00, 4.0800e+01, 1.4600e+01, 1.1400e+01,
                 2.3000e+00, 5.1000e+00, 6.9000e+00, 3.2000e+00, 1.3600e+01, 1.0200e+01,
                 5.9000e+00, 6.7000e+00, 2.2600e+01, 6.4000e+00, 2.0800e+01, 3.2000e+00,
                 1.4700e+01, 5.0000e+00,        nan,        nan, 1.6200e+01, 4.1000e+00,
                 3.3000e+01,        nan,        nan, 4.2700e+01, 4.7000e+00, 3.4000e+00,
                        nan, 1.2050e+02, 2.4000e+00, 1.6000e+00, 3.0000e-01, 1.0000e-01,
                 2.5000e+00, 1.0900e+01, 1.0790e+02, 8.9000e+00, 9.4600e+01, 2.3200e+01,
                 1.9600e+01, 1.5400e+01,        nan, 1.5100e+01, 7.8000e+00, 6.6000e+00,
                 1.2700e+01, 9.4500e+01, 6.6000e+00, 5.7000e+00, 3.9000e+00, 2.1000e+00,
                 6.4600e+01, 1.3370e+02,        nan, 6.0000e-01, 1.4000e+01, 8.1000e+00,
                        nan, 3.4900e+01, 9.9000e+00, 7.9000e+00, 3.7000e+01, 7.6000e+00,
                 5.8000e+00, 1.9800e+01, 1.5100e+01, 6.0000e-01, 1.7300e+01, 4.6000e+00,
                 3.0700e+01, 1.0300e+01, 2.7000e+01,        nan, 5.4000e+00,        nan,
                 1.9200e+01,        nan, 1.6800e+01, 3.7100e+01, 5.0000e-01, 6.9000e+01,
                 1.4500e+01, 1.9600e+01, 3.0000e+01, 5.4000e+00, 1.4200e+01, 1.2300e+01,
                 4.9000e+00, 3.6500e+01, 8.0000e-01, 4.7000e+00, 8.0000e+00, 1.7500e+01,
                 7.7000e+00, 1.3900e+01, 2.7000e+01, 1.0400e+01], dtype=torch.float64),
          'OS_STATUS': tensor([0., 0., 1., 0., 1., 1., nan, 0., 1., 0., 0., 0., 1., 1., nan, 0., nan, 1.,
                 0., 0., nan, 1., 1., 0., 0., 0., 0., 0., 0., 0., nan, 1., 1., 1., 0., 0.,
                 nan, 0., nan, 1., 1., 1., nan, 1., 0., 0., 0., nan, 0., 0., 1., 1., 0., 0.,
                 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 0.,
                 nan, 0., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., nan, 0., 0., 1., 0., 0.,
                 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., nan, 0., 0., 0., 0., nan, 0.,
                 1., 0., 0., 0., nan, 1., 1., 1., 0., nan, 0., 1., 1., 0., 1., 0., 0., 0.,
                 1., 0., nan, 1., 0., 0., 1., nan, 0., 0., 1., 1., nan, 0., 0., 1., 0., 1.,
                 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 1., 0., 1., 0., nan, nan, 0., 0.,
                 0., nan, nan, 0., 0., 0., nan, 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
                 0., 1., nan, 0., 0., 0., 1., 1., 0., 0., 0., 1., 0., 1., nan, 0., 0., 0.,
                 nan, 1., 1., 1., 0., 0., 1., 1., 0., 0., 0., 0., 0., 1., 0., nan, 1., nan,
                 0., nan, 1., 0., 0., 0., 1., 0., 1., 0., 0., 1., 1., 0., 0., 1., 1., 0.,
                 0., 1., 0., 0.], dtype=torch.float64),
          'KARNOFSKY_PERFORMANCE_SCORE': tensor([ nan,  80., 100., 100.,  nan,  80.,  nan,  nan,  nan,  60.,  80.,  na
         n,
                  60.,  nan,  nan,  nan,  nan,  nan,  90.,  70.,  nan,  90.,  80.,  nan,
                  90., 100.,  80.,  90., 100., 100.,  nan,  80.,  nan,  nan,  80., 100.,
                  nan, 100.,  nan,  90.,  nan,  90.,  nan,  nan,  90.,  nan, 100.,  nan,
                  90.,  90.,  80., 100.,  nan,  90.,  nan,  nan,  40.,  nan,  nan,  nan,
                  70., 100.,  60.,  40.,  90.,  nan, 100.,  nan,  60.,  80.,  nan,  80.,
                  nan,  70.,  60., 100.,  80.,  80.,  nan, 100., 100.,  70., 100.,  90.,
                  nan,  nan,  80.,  nan,  80.,  90.,  80.,  nan,  nan,  nan,  nan,  nan,
                  80.,  90.,  nan, 100., 100.,  nan,  90.,  80.,  nan,  nan,  nan,  nan,
                  nan,  80.,  80.,  90.,  nan,  nan,  40.,  60.,  nan,  nan, 100.,  40.,
                  80.,  nan, 100.,  nan,  nan,  nan,  90.,  nan,  nan,  80.,  60.,  50.,
                 100.,  nan,  40.,  80.,  80.,  60.,  nan, 100.,  90.,  90.,  nan,  nan,
               nan,  80.,  60.,  80.,  nan,  nan,  nan,  80., 100.,  90.,  nan,  90.,
               80.,  nan,  nan,  nan,  nan,  90.,  nan,  nan,  nan,  nan, 100., 100.,
               nan,  nan,  nan,  80.,  80.,  nan,  nan,  90.,  nan,  80., 100.,  nan,
               nan,  nan,  nan,  80., 100.,  nan,  nan,  90.,  nan,  80.,  90.,  nan,
              100., 100.,  nan,  nan,  90., 100.,  nan,  80., 100.,  60., 100.,  90.,
               nan, 100., 100.,  nan,  nan,  40.,  90.,  80.,  nan,  nan,  60.,  nan,
               nan,  nan,  nan,  nan,  nan, 100.,  80., 100.,  80.,  nan,  90.,  nan,
               60.,  nan,  70.,  80.,  nan,  90.,  nan,  nan,  nan,  80.],
             dtype=torch.float64),
 'STUDY': tensor([1., 0., 1., 1., 1., 1., 0., 1., 1., 0., 0., 0., 1., 1., 0., 0., 0., 1.,
         0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1.,
         0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
         0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.,
         0., 1., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0., 1., 1., 0.,
         0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0., 0., 0., 0.,
         0., 1., 0., 0., 0., 1., 1., 1., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0.,
         0., 1., 1., 0., 1., 0., 0., 0., 1., 0., 1., 1., 0., 0., 0., 0., 0., 0.,
         1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 1., 0., 1., 1., 0., 1., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0.,
         0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
         0., 1., 1., 0., 1., 0., 1., 1., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0.,
         0., 0., 0., 0., 0., 0., 1., 0., 1., 1., 0., 1., 1., 0., 0., 1., 0., 1.,
         0., 1., 0., 1.], dtype=torch.float64),
 'BCR_STATUS': tensor([0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 1., 0., 1., 0.,
         0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
         0., 1., 1., 0., 0., 0., 1., 0., 0., 1., 1., 0., 0., 1., 0., 0., 0., 1.,
         0., 1., 0., 0., 0., 0., 1., 1., 0., 1., 0., 1., 1., 0., 0., 0., 0., 0.,
         1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1.,
         1., 0., 0., 0., 0., 0., 1., 0., 0., 1., 1., 1., 0., 0., 0., 0., 1., 0.,
         0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 1., 1., 0., 1., 0., 0., 0., 1., 0., 1., 0., 0.,
         0., 0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 1.,
         0., 0., 0., 0., 1., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 1.,
         0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1.,
         1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1.,
         0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 1., 0.,
         0., 0., 1., 0.], dtype=torch.float64),
 'HISTOLOGICAL_DIAGNOSIS': tensor([1., 2., 1., 1., 1., 1., nan, 1., 1., 3., 3., 3., 1., 1., nan, 3., nan, 1.,
         2., 1., nan, 3., 1., 2., 0., 2., 0., 3., 2., 3., nan, 1., 1., 1., 1., 1.,
         nan, 2., nan, 0., 1., 2., nan, 1., 0., 0., 2., nan, 2., 2., 1., 0., 2., 0.,
         2., 0., 1., 1., 3., 2., 3., 0., 2., 3., 2., 2., 3., 1., 1., 1., 1., 1.,
         nan, 1., 1., 1., 1., 1., 1., 3., 3., 1., 0., 0., nan, 3., 0., 1., 1., 0.,
         3., 3., 3., 1., 0., 0., 2., 0., 1., 0., 0., nan, 3., 1., 2., 3., nan, 2.,
         3., 1., 0., 0., nan, 1., 1., 1., 0., nan, 3., 1., 1., 0., 0., 3., 3., 2.,
         2., 1., nan, 0., 1., 0., 0., nan, 1., 3., 1., 1., nan, 3., 0., 2., 3., 3.,
         1., 1., 1., 1., 1., 3., 0., 2., 0., 0., 1., 2., 1., 1., nan, nan, 0., 3.,
         3., nan, nan, 2., 3., 2., nan, 1., 0., 3., 3., 2., 1., 0., 0., 1., 0., 3.,
         0., 1., nan, 0., 1., 3., 0., 3., 0., 0., 2., 1., 3., 3., nan, 0., 0., 3.,
         nan, 1., 1., 0., 1., 0., 1., 1., 3., 2., 0., 1., 0., 1., 0., nan, 1., nan,
         0., nan, 2., 3., 3., 0., 1., 0., 1., 1., 0., 1., 1., 2., 3., 1., 0., 1.,
         2., 1., 2., 1.], dtype=torch.float64),
 'SEX': tensor([0., 0., 1., 0., 1., 1., nan, 1., 0., 1., 1., 1., 1., 0., nan, 1., nan, 1.,
         0., 1., nan, 1., 1., 1., 1., 1., 0., 0., 1., 0., nan, 0., 0., 1., 1., 1.,
         nan, 0., nan, 1., 1., 0., nan, 0., 1., 0., 0., nan, 0., 1., 0., 1., 1., 0.,
         1., 1., 0., 1., 1., 0., 1., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1., 1.,
         nan, 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., nan, 0., 0., 0., 1., 0.,
         1., 1., 0., 0., 0., 0., 1., 1., 1., 0., 1., nan, 0., 0., 0., 1., nan, 1.,
         1., 0., 1., 1., nan, 1., 0., 0., 0., nan, 1., 1., 1., 0., 1., 1., 0., 0.,
         0., 0., nan, 0., 0., 1., 1., nan, 0., 1., 1., 1., nan, 1., 0., 1., 1., 1.,
         0., 1., 1., 1., 0., 1., 1., 0., 1., 0., 0., 1., 1., 1., nan, nan, 1., 1.,
         0., nan, nan, 1., 0., 0., nan, 0., 1., 0., 1., 1., 1., 0., 1., 0., 1., 0.,
         1., 1., nan, 1., 0., 1., 1., 0., 1., 0., 0., 0., 0., 0., nan, 1., 0., 0.,
         nan, 1., 0., 0., 1., 0., 1., 1., 0., 1., 0., 1., 1., 1., 1., nan, 0., nan,
         1., nan, 0., 0., 1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 0., 0., 1., 0.,
         1., 1., 1., 1.], dtype=torch.float64)}
```

- Make a histogram plot of the follow up times in months (OS_MONTHS) (use sns.histplot)

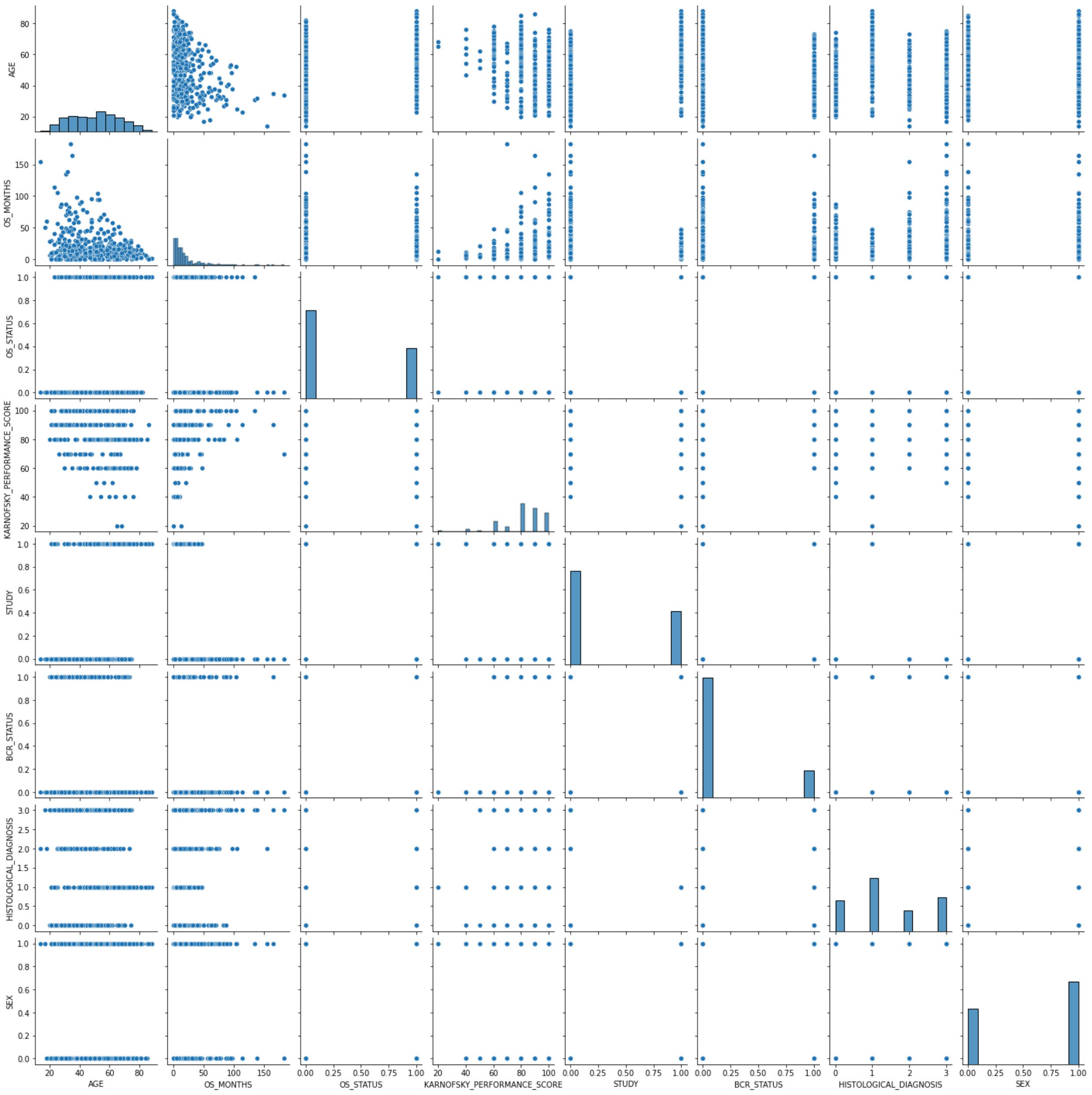
In [11]:
```python
sns.histplot(train_dataset.ann['OS_MONTHS'])
```

Out[11]:
```
<Axes: ylabel='Count'>
```



```
In [12]: sns.pairplot(pd.DataFrame(train_dataset.ann))
```

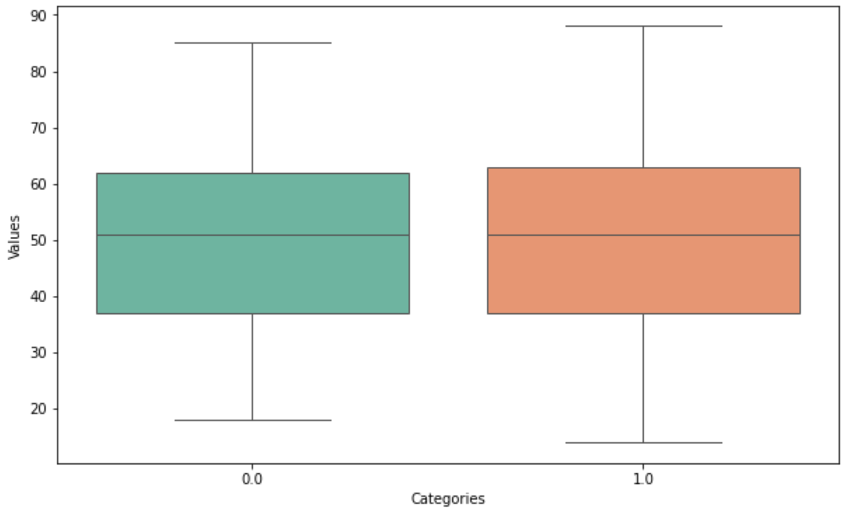
```
Out[12]: <seaborn.axisgrid.PairGrid at 0x7f4655352cb0>
```



- Make a histogram of the age distribution of the patients in the training data; facet the histogram by "SEX" variable (see flexynesis.utils.plot_boxplot)

```
In [13]: flexynesis.utils.plot_boxplot(categorical_x=train_dataset.ann['SEX'],
                                       numerical_y=train_dataset.ann['AGE'])
```

```
/gnu/store/88qdfdp9h0003w4wp1rv46khc3dpp8j6-profile/lib/python3.10/site-packages/flexynesis/utils.py:155: Future
Warning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable
to `hue` and set `legend=False` for the same effect.

/gnu/store/88qdfdp9h0003w4wp1rv46khc3dpp8j6-profile/lib/python3.10/site-packages/seaborn/_base.py:949: FutureWar
ning: When grouping with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future v
ersion of pandas. Pass `(name,)` instead of `name` to silence this warning.
/gnu/store/88qdfdp9h0003w4wp1rv46khc3dpp8j6-profile/lib/python3.10/site-packages/seaborn/categorical.py:640: Fut
ureWarning: SeriesGroupBy.grouper is deprecated and will be removed in a future version of pandas.
/gnu/store/88qdfdp9h0003w4wp1rv46khc3dpp8j6-profile/lib/python3.10/site-packages/seaborn/_base.py:949: FutureWar
ning: When grouping with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future v
ersion of pandas. Pass `(name,)` instead of `name` to silence this warning.
/gnu/store/88qdfdp9h0003w4wp1rv46khc3dpp8j6-profile/lib/python3.10/site-packages/seaborn/categorical.py:640: Fut
ureWarning: SeriesGroupBy.grouper is deprecated and will be removed in a future version of pandas.
```



- Make a summary of all available clinical variables (see flexynesis.print_summary_stats)

In [14]:
```
flexynesis.print_summary_stats(train_dataset)
```

```
Summary for variable: AGE
Numerical Variable Summary: Median = 51.0, Mean = 50.33781190019194
------
Summary for variable: OS_MONTHS
Numerical Variable Summary: Median = 11.6, Mean = 19.09097888675624
------
Summary for variable: OS_STATUS
Numerical Variable Summary: Median = 0.0, Mean = 0.36153846153846153
------
Summary for variable: KARNOFSKY_PERFORMANCE_SCORE
Numerical Variable Summary: Median = 80.0, Mean = 82.45454545454545
------
Summary for variable: STUDY
Categorical Variable Summary:
  Label: Brain Lower Grade Glioma, Count: 353
  Label: Glioblastoma multiforme, Count: 203
------
Summary for variable: BCR_STATUS
Categorical Variable Summary:
  Label: IGC, Count: 454
  Label: NCH, Count: 102
------
Summary for variable: HISTOLOGICAL_DIAGNOSIS
Categorical Variable Summary:
  Label: astrocytoma, Count: 115
  Label: glioblastoma, Count: 201
  Label: oligoastrocytoma, Count: 79
  Label: oligodendroglioma, Count: 126
  Label: nan, Count: 35
------
Summary for variable: SEX
Categorical Variable Summary:
  Label: Female, Count: 209
  Label: Male, Count: 312
  Label: nan, Count: 35
------
```

- Notice that the categorical variables such as "SEX", "STUDY", "HISTOLOGICAL_DIAGNOSIS" are encoded numerically in the "dataset.ann" objects. Use dataset.label_mappings to map the STUDY variable to their original labels. Print the top 10 values in dataset.ann['STUDY'] and the mapped label values.

```
In [15]: train_dataset.label_mappings
```

```
Out[15]: {'STUDY': {0: 'Brain Lower Grade Glioma', 1: 'Glioblastoma multiforme'},
          'BCR_STATUS': {0: 'IGC', 1: 'NCH'},
          'HISTOLOGICAL_DIAGNOSIS': {0: 'astrocytoma',
           1: 'glioblastoma',
           2: 'oligoastrocytoma',
           3: 'oligodendroglioma',
           4: nan},
          'SEX': {0: 'Female', 1: 'Male', 2: nan}}
```

```
In [16]: f = 'STUDY'
         # map the sample labels from numeric vector to initial labels
         labels = [train_dataset.label_mappings[f][x] for x in train_dataset.ann[f].numpy()]
```

```
In [17]: train_dataset.ann[f][:10].numpy(), labels[:10]
```

```
Out[17]: (array([0., 1., 0., 0., 0., 0., 1., 0., 1., 1.]),
          ['Brain Lower Grade Glioma',
           'Glioblastoma multiforme',
           'Brain Lower Grade Glioma',
           'Brain Lower Grade Glioma',
           'Brain Lower Grade Glioma',
           'Brain Lower Grade Glioma',
           'Glioblastoma multiforme',
           'Brain Lower Grade Glioma',
           'Glioblastoma multiforme',
           'Glioblastoma multiforme'])
```

- Now, let's explore the data matrices. Make a PCA plot of the mutation data matrix and color the samples by "HISTOLOGICAL_DIAGNOSIS". See flexynesis.plot_dim_reduced function

First create a pandas data frame with the data matrix of interest with feature and sample names

> df = pd.DataFrame(train_dataset.dat['cna'], index = train_dataset.samples, columns= train_dataset.features['cna'])

Check the data frame contents

> df.head()

Make a PCA plot of CNA values using the labels from the STUDY variable

Note: if you couldn't map the labels above, you can also use train_dataset.dat['STUDY'] as labels

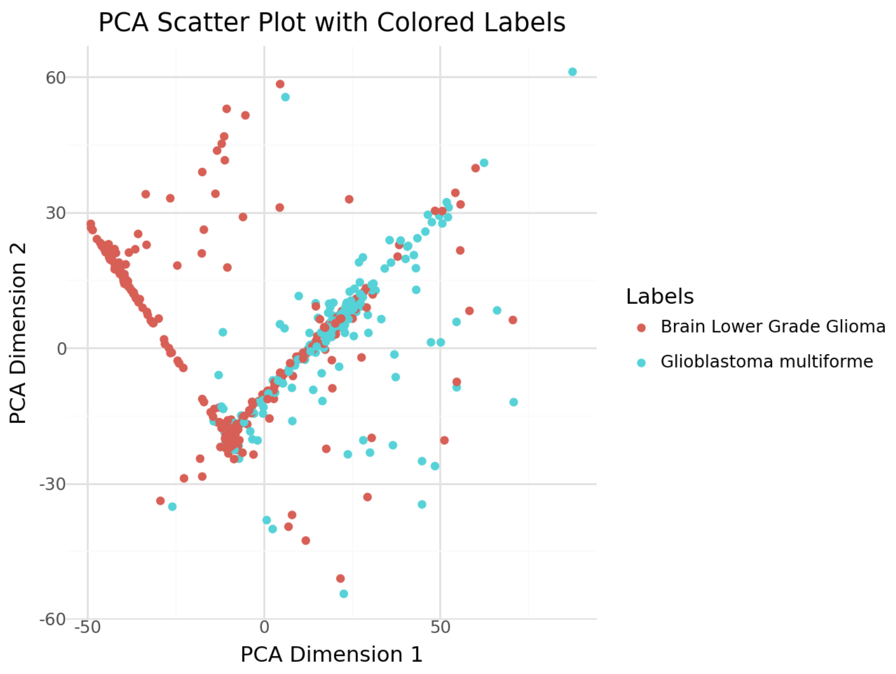
```
In [19]: df = pd.DataFrame(train_dataset.dat['cna'], index = train_dataset.samples, columns = train_dataset.features['cn
         df.head()
```

Out[19]:

	SLC30A8	ZNF273	OR9A1P	AGL	KCNA5	MIR603	SNTB1	MRPL13	MTBP	SNORA72|ENSG0
TCGA-DU-6405	-0.227827	0.706178	0.696675	0.381190	-0.207080	-1.155580	-0.236311	-0.236152	-0.236152	
TCGA-06-2564	-0.254134	2.116086	2.217327	0.373042	-0.207080	-1.137639	-0.263039	-0.262880	-0.262880	
TCGA-WH-A86K	-0.219935	-0.785676	-0.819444	0.411069	-0.180481	2.302387	-0.228293	-0.228134	-0.228134	
TCGA-QH-A65X	-0.267288	-0.749956	-0.873834	-1.794481	-0.199100	0.571161	-0.276403	-0.276243	-0.276243	
TCGA-HT-7601	-0.262026	-0.737348	-0.830775	0.389339	-0.228359	0.546493	-0.271057	-0.270898	-0.270898	

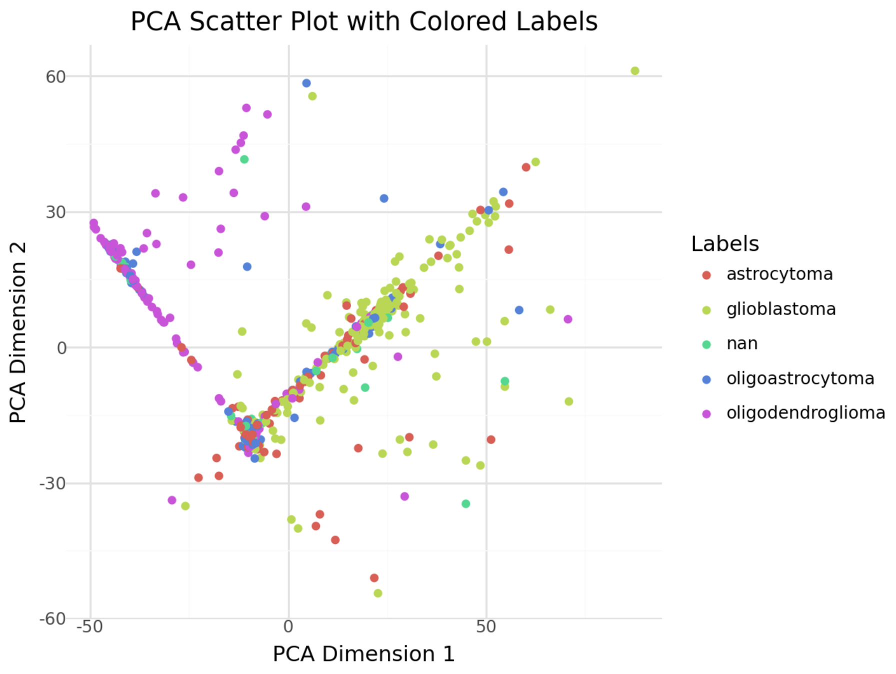
5 rows × 1237 columns

```
In [20]: flexynesis.plot_dim_reduced(matrix=df, labels=labels, method='pca')
```

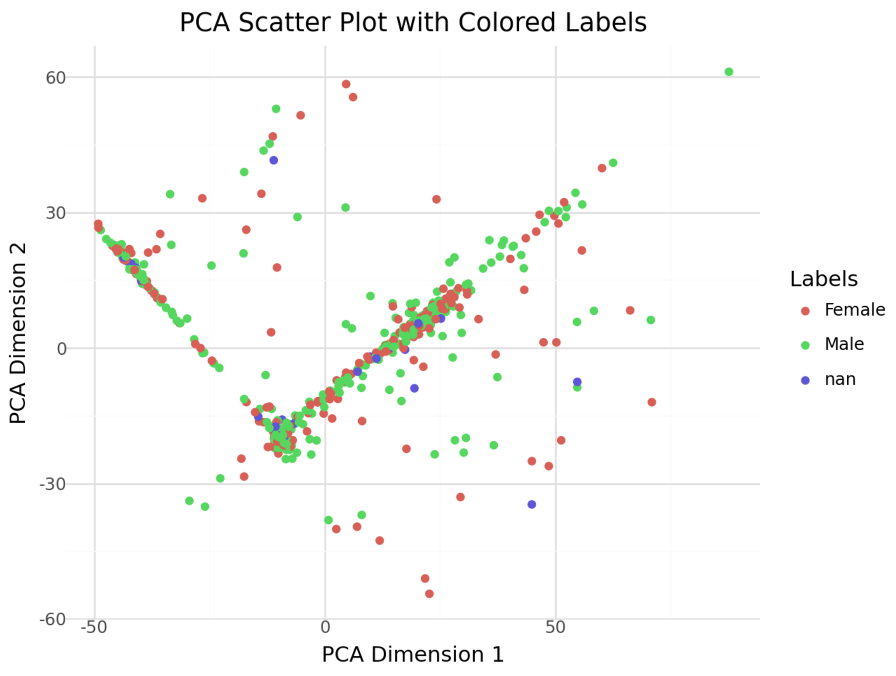


- (Optional exercise ideas):
  - Make a PCA plot coloring the samples by HISTOLOGICAL_DIAGNOSIS, GENDER, or any other clinical variable
  - Repeat the same exercise on the mutation data matrix.

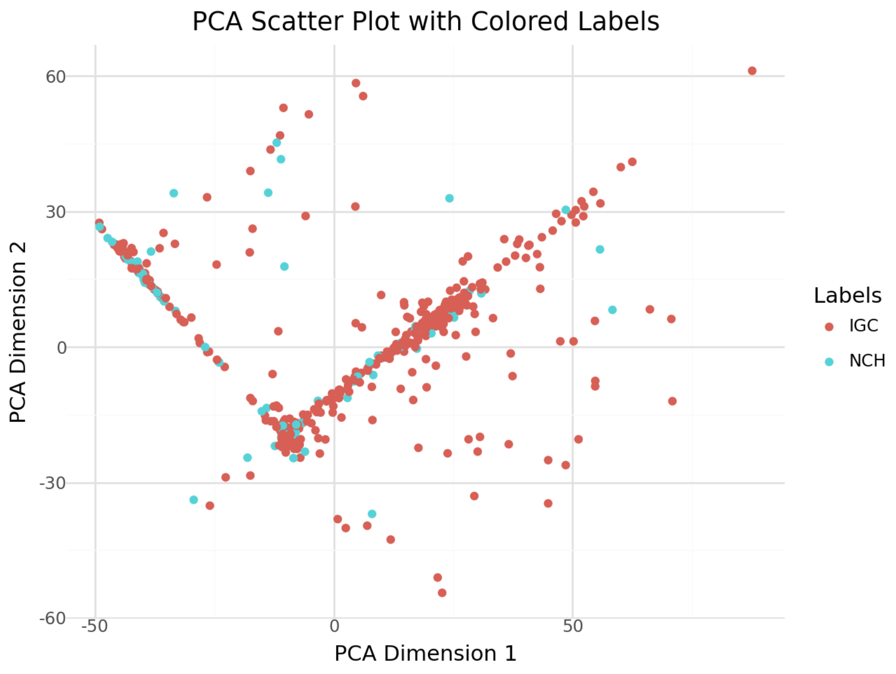
In [21]:
```
f = 'HISTOLOGICAL_DIAGNOSIS'
# map the sample labels from numeric vector to initial labels.
labels = [train_dataset.label_mappings[f][x] if not pd.isna(x) else 'nan' for x in train_dataset.ann[f].numpy()
flexynesis.plot_dim_reduced(matrix=df, labels=labels, method='pca')
```



In [22]:
```
f = 'SEX'
# map the sample labels from numeric vector to initial labels.
labels = [train_dataset.label_mappings[f][x] if not pd.isna(x) else 'nan' for x in train_dataset.ann[f].numpy()
flexynesis.plot_dim_reduced(matrix=df, labels=labels, method='pca')
```



In [23]:
```
f = 'BCR_STATUS'
# map the sample labels from numeric vector to initial labels.
labels = [train_dataset.label_mappings[f][x] if not pd.isna(x) else 'nan' for x in train_dataset.ann[f].numpy()
flexynesis.plot_dim_reduced(matrix=df, labels=labels, method='pca')
```



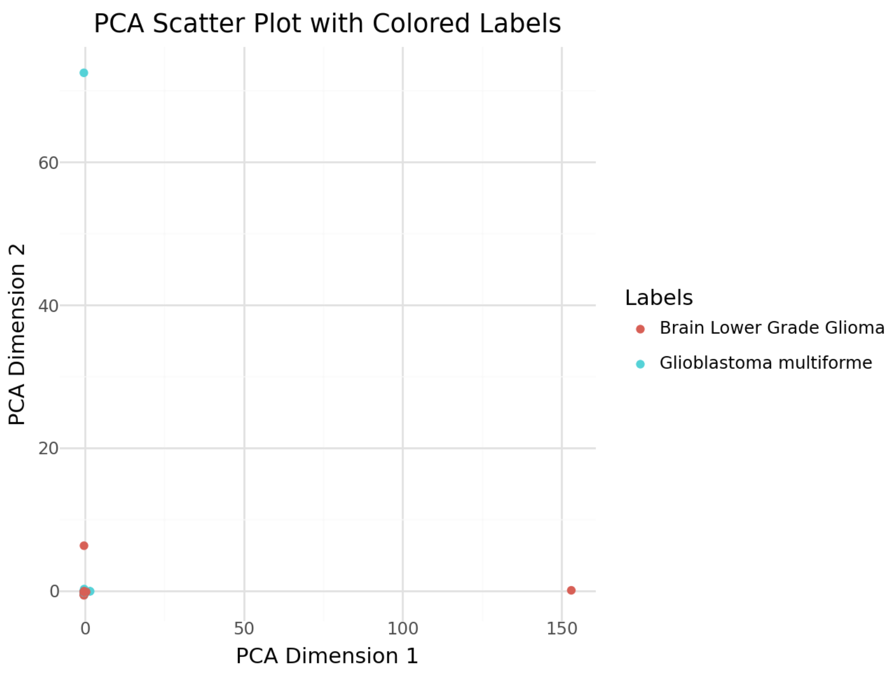
In [24]:
```
# mutation
df = pd.DataFrame(train_dataset.dat['mut'], index = train_dataset.samples, columns = train_dataset.features['mu
df.head()
```

Out[24]:

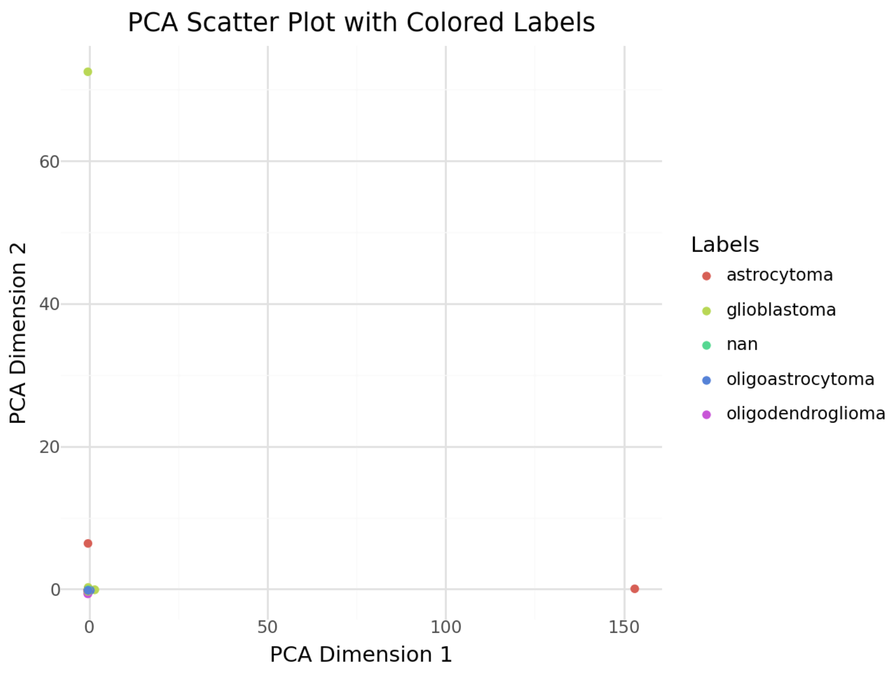
	IDH2	IDH1	RELN	ATRX	PIK3CA	EGFR	TP53	COL6A3	SVIL	CIC	...	ZB
TCGA-DU-6405	-0.148522	-1.018150	-0.172133	-0.585658	3.253204	-0.331331	-0.809174	-0.177595	-0.104447	-0.344546	...	-0.04
TCGA-06-2564	-0.148522	-1.018150	-0.172133	-0.585658	-0.307389	-0.331331	-0.809174	-0.177595	-0.104447	-0.344546	...	-0.04
TCGA-WH-A86K	-0.148522	0.982173	-0.172133	-0.585658	-0.307389	-0.331331	1.235829	-0.177595	-0.104447	-0.344546	...	-0.04
TCGA-QH-A65X	6.733003	-1.018150	-0.172133	-0.585658	-0.307389	-0.331331	-0.809174	-0.177595	-0.104447	-0.344546	...	-0.04
TCGA-HT-7601	-0.148522	0.982173	-0.172133	-0.585658	-0.307389	-0.331331	1.235829	-0.177595	-0.104447	-0.344546	...	-0.04

5 rows × 317 columns

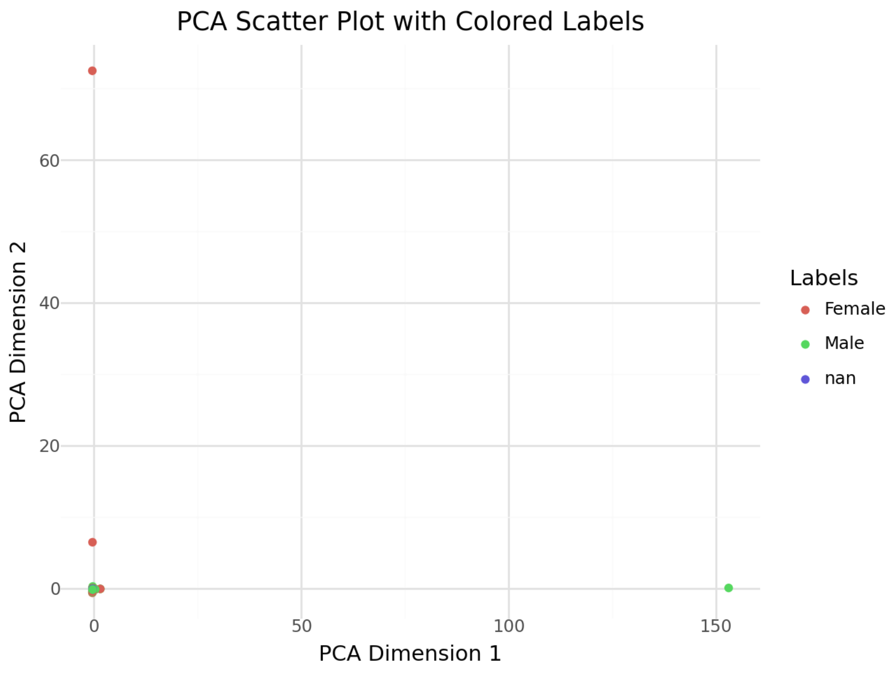
In [25]:
```python
f = 'STUDY'
# map the sample labels from numeric vector to initial labels
labels = [train_dataset.label_mappings[f][x] for x in train_dataset.ann[f].numpy()]
flexynesis.plot_dim_reduced(matrix=df, labels=labels, method='pca')
```



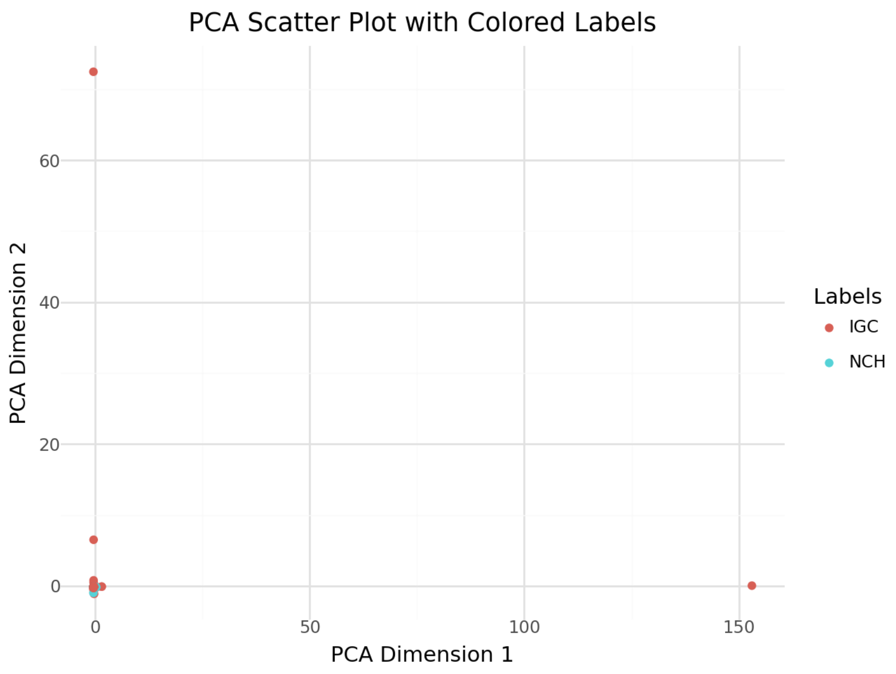
In [26]:
```python
f = 'HISTOLOGICAL_DIAGNOSIS'
# map the sample labels from numeric vector to initial labels.
labels = [train_dataset.label_mappings[f][x] if not pd.isna(x) else 'nan' for x in train_dataset.ann[f].numpy()
flexynesis.plot_dim_reduced(matrix=df, labels=labels, method='pca')
```



```
In [27]: f = 'SEX'
         # map the sample labels from numeric vector to initial labels.
         labels = [train_dataset.label_mappings[f][x] if not pd.isna(x) else 'nan' for x in train_dataset.ann[f].numpy()
         flexynesis.plot_dim_reduced(matrix=df, labels=labels, method='pca')
```



```
In [28]: f = 'BCR_STATUS'
         # map the sample labels from numeric vector to initial labels.
         labels = [train_dataset.label_mappings[f][x] if not pd.isna(x) else 'nan' for x in train_dataset.ann[f].numpy()
         flexynesis.plot_dim_reduced(matrix=df, labels=labels, method='pca')
```



2. Training a single model using manually set hyperparameters

Now that we have familiarized ourselves with the dataset at hand, we can start building models.

First we will do a single model training by manually setting hyperparameters. Based on the model performance, we will try modifying individual hyperparameters and build more and more models and see if we can improve model performance.

We will need to define the following components for starting a model training:

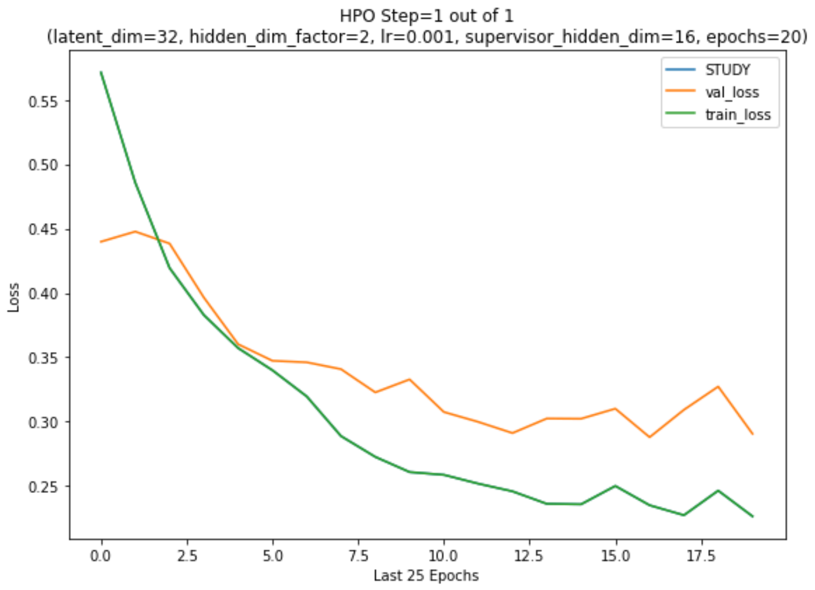
```
1. Split the train_dataset into train/validation components
2. Define data loaders for both train and validation splits
3. Define a pytorch-lightning trainer
4. Define a model with hyperparameters
5. Fit the model
```

In [29]:
```
# randomly assign 80% of samples for training, 20% for validation
train_indices = random.sample(range(0, len(train_dataset)), int(len(train_dataset) * 0.8))
val_indices = list(set(range(len(train_dataset))) - set(train_indices))
train_subset = train_dataset.subset(train_indices)
val_subset = train_dataset.subset(val_indices)

# define data loaders for train/validation splits
from torch.utils.data import DataLoader
train_loader = DataLoader(train_subset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_subset, batch_size=32, shuffle = False)
```

Now, we need to define a model with manually set hyperparameters and a lightning-trainer fit the model.

Notice: Notice the callback we are passing to the trainer which enables us to plot the loss values as the training progresses.

In [30]:
```
# Define a model with manually set hyperparameters for the DirectPred model
# 'hidden_dim_factor' multiplied by the number of features gives the number of neurons
myparams = {'latent_dim': 32, 'hidden_dim_factor': 2, 'lr': 0.001, 'supervisor_hidden_dim': 16, 'epochs': 20}
model = flexynesis.DirectPred(config = myparams, dataset = train_dataset, target_variables=['STUDY'])
trainer = pl.Trainer(max_epochs=myparams['epochs'], default_root_dir="./", logger=False, enable_checkpointing=F
                     callbacks=[flexynesis.LiveLossPlot(myparams, 1, 1)])

# flexynesis.LiveLossPlot(
#     hyperparams,
#     current_step,
#     total_steps,
#     figsize=(8, 6),
# )
# Docstring:
# A callback for visualizing training loss in real-time during hyperparameter optimization.

# This class is a PyTorch Lightning callback that plots training loss and other metrics live as the model train
# It is especially useful for tracking the progress of hyperparameter optimization (HPO) steps.

# Attributes:
#     hyperparams (dict): Hyperparameters being used in the current HPO step.
#     current_step (int): The current step number in the HPO process.
#     total_steps (int): The total number of steps in the HPO process.
#     figsize (tuple): Size of the figure used for plotting.

# Fit the model
trainer.fit(model, train_loader, val_loader)
```



```
`Trainer.fit` stopped: `max_epochs=20` reached.
```

While we can observe how well the model training went based on the "loss" values, we can also evaluate the model performance on test dataset

In [31]:
```
# evaluate the model performance on predicting the target variable
flexynesis.evaluate_wrapper("DirectPred", model.predict(test_dataset), test_dataset)
```

Out[31]:

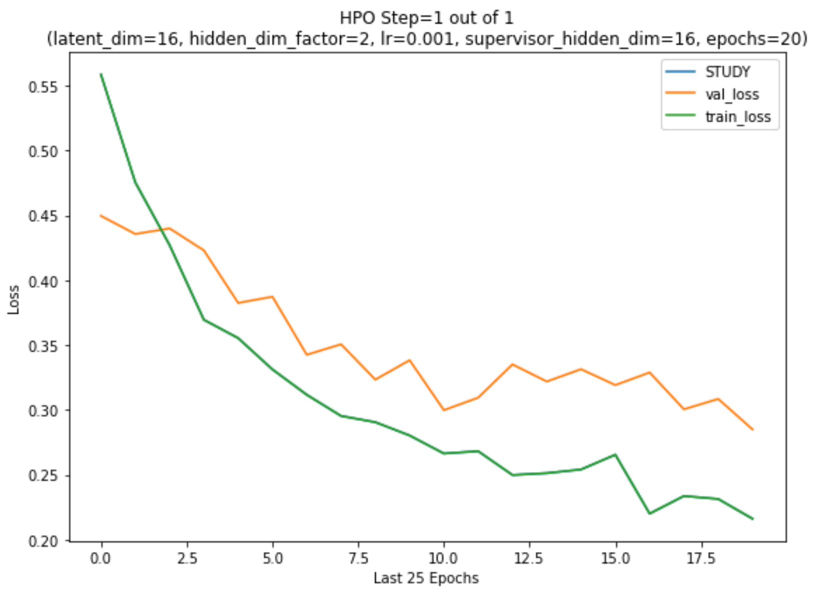
	method	var	variable_type	metric	value
0	DirectPred	STUDY	categorical	balanced_acc	0.781046
1	DirectPred	STUDY	categorical	f1_score	0.801706
2	DirectPred	STUDY	categorical	kappa	0.566535
3	DirectPred	STUDY	categorical	average_auroc	0.881815
4	DirectPred	STUDY	categorical	average_aupr	0.765778

2.1 Exercise

- Now, repeat the above model training and evaluation by manually changing the hyperparameters (Try at least 5 different combinations)
- See if you can find a better hyperparameter combination that yields a better classification performance than the initial setup we provided.
- See the default hyperparameter ranges we use for Flexynesis here: https://github.com/BIMSBbioinfo/flexynesis/blob/69b92ca9370551e9fcc82a756cb42c72bef4a4b1/flexynesis/config.py#L7, but feel free to try outside these ranges too.
- Also try to observe the impact of the changing parameters on how the train/validation loss curves change.

```
myparams = {'latent_dim': XX, 'hidden_dim_factor': XX, 'lr': XX, 'supervisor_hidden_dim': XX,
'epochs': XX}

model = flexynesis.DirectPred(config = myparams, dataset = train_dataset, target_variables=
['STUDY'])

trainer = pl.Trainer(max_epochs=myparams['epochs'], default_root_dir="./", logger=False,
enable_checkpointing=False,
                     callbacks=[flexynesis.LiveLossPlot(myparams, 1, 1)])

trainer.fit(model, train_loader, val_loader)

flexynesis.evaluate_wrapper("DirectPred", model.predict(test_dataset), test_dataset)
```

In [32]:
```
# 1. change latent dim to 16
myparams = {'latent_dim': 16, 'hidden_dim_factor': 2, 'lr': 0.001, 'supervisor_hidden_dim': 16, 'epochs': 20}
model = flexynesis.DirectPred(config = myparams, dataset = train_dataset, target_variables=['STUDY'])
trainer = pl.Trainer(max_epochs=myparams['epochs'], default_root_dir="./", logger=False, enable_checkpointing=F
                    callbacks=[flexynesis.LiveLossPlot(myparams, 1, 1)])
# Fit the model
trainer.fit(model, train_loader, val_loader)
# evaluate the model performance on predicting the target variable
flexynesis.evaluate_wrapper("DirectPred", model.predict(test_dataset), test_dataset)
```



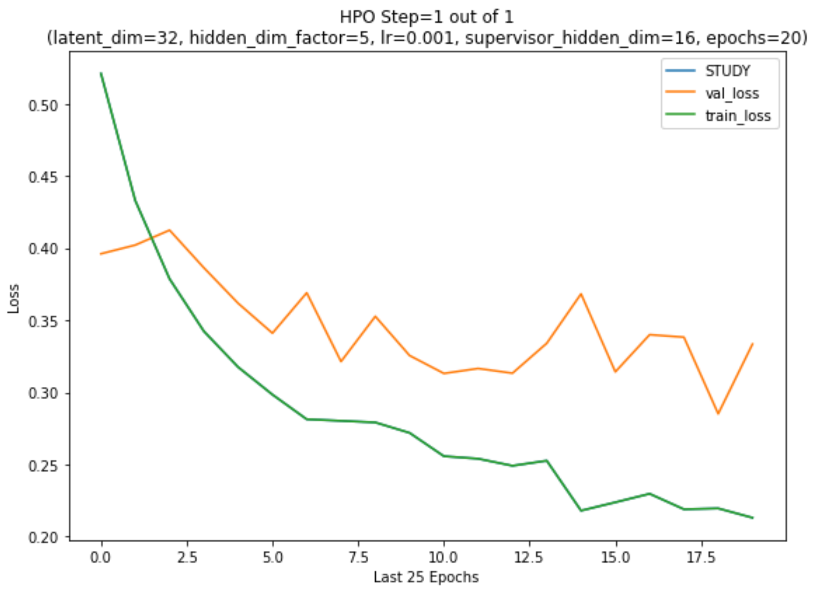
```
`Trainer.fit` stopped: `max_epochs=20` reached.
```

Out[32]:

	method	var	variable_type	metric	value
0	DirectPred	STUDY	categorical	balanced_acc	0.736601
1	DirectPred	STUDY	categorical	f1_score	0.765750
2	DirectPred	STUDY	categorical	kappa	0.484605
3	DirectPred	STUDY	categorical	average_auroc	0.842522
4	DirectPred	STUDY	categorical	average_aupr	0.757123

In [33]:

```
# 2. change hidden_dim_factor to 5
myparams = {'latent_dim': 32, 'hidden_dim_factor': 5, 'lr': 0.001, 'supervisor_hidden_dim': 16, 'epochs': 20}
model = flexynesis.DirectPred(config = myparams, dataset = train_dataset, target_variables=['STUDY'])
trainer = pl.Trainer(max_epochs=myparams['epochs'], default_root_dir="./", logger=False, enable_checkpointing=F
                    callbacks=[flexynesis.LiveLossPlot(myparams, 1, 1)])
# Fit the model
trainer.fit(model, train_loader, val_loader)
# evaluate the model performance on predicting the target variable
flexynesis.evaluate_wrapper("DirectPred", model.predict(test_dataset), test_dataset)
```

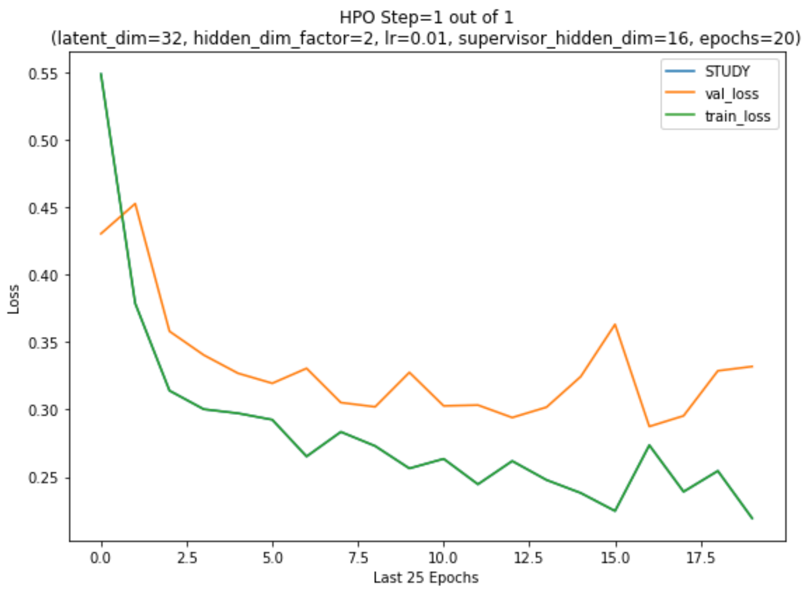


```
`Trainer.fit` stopped: `max_epochs=20` reached.
```

Out[33]:

	method	var	variable_type	metric	value
0	DirectPred	STUDY	categorical	balanced_acc	0.769935
1	DirectPred	STUDY	categorical	f1_score	0.795909
2	DirectPred	STUDY	categorical	kappa	0.551402
3	DirectPred	STUDY	categorical	average_auroc	0.860746
4	DirectPred	STUDY	categorical	average_aupr	0.747558

In [34]:
```
# 3. change learning rate to 0.01
myparams = {'latent_dim': 32, 'hidden_dim_factor': 2, 'lr': 0.01, 'supervisor_hidden_dim': 16, 'epochs': 20}
model = flexynesis.DirectPred(config = myparams, dataset = train_dataset, target_variables=['STUDY'])
trainer = pl.Trainer(max_epochs=myparams['epochs'], default_root_dir="./", logger=False, enable_checkpointing=F
                     callbacks=[flexynesis.LiveLossPlot(myparams, 1, 1)])
# Fit the model
trainer.fit(model, train_loader, val_loader)
# evaluate the model performance on predicting the target variable
flexynesis.evaluate_wrapper("DirectPred", model.predict(test_dataset), test_dataset)
```

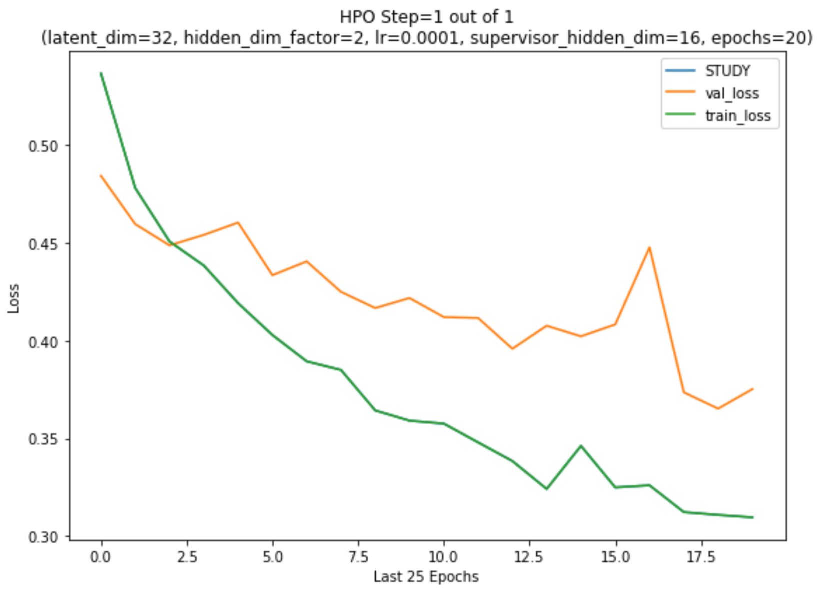


```
`Trainer.fit` stopped: `max_epochs=20` reached.
```

Out[34]:

	method	var	variable_type	metric	value
0	DirectPred	STUDY	categorical	balanced_acc	0.726144
1	DirectPred	STUDY	categorical	f1_score	0.762150
2	DirectPred	STUDY	categorical	kappa	0.473324
3	DirectPred	STUDY	categorical	average_auroc	0.859516
4	DirectPred	STUDY	categorical	average_aupr	0.767582

In [35]:
```
# 4. change learning rate to 0.0001
myparams = {'latent_dim': 32, 'hidden_dim_factor': 2, 'lr': 0.0001, 'supervisor_hidden_dim': 16, 'epochs': 20}
model = flexynesis.DirectPred(config = myparams, dataset = train_dataset, target_variables=['STUDY'])
trainer = pl.Trainer(max_epochs=myparams['epochs'], default_root_dir="./", logger=False, enable_checkpointing=F
                     callbacks=[flexynesis.LiveLossPlot(myparams, 1, 1)])
# Fit the model
trainer.fit(model, train_loader, val_loader)
# evaluate the model performance on predicting the target variable
flexynesis.evaluate_wrapper("DirectPred", model.predict(test_dataset), test_dataset)
```



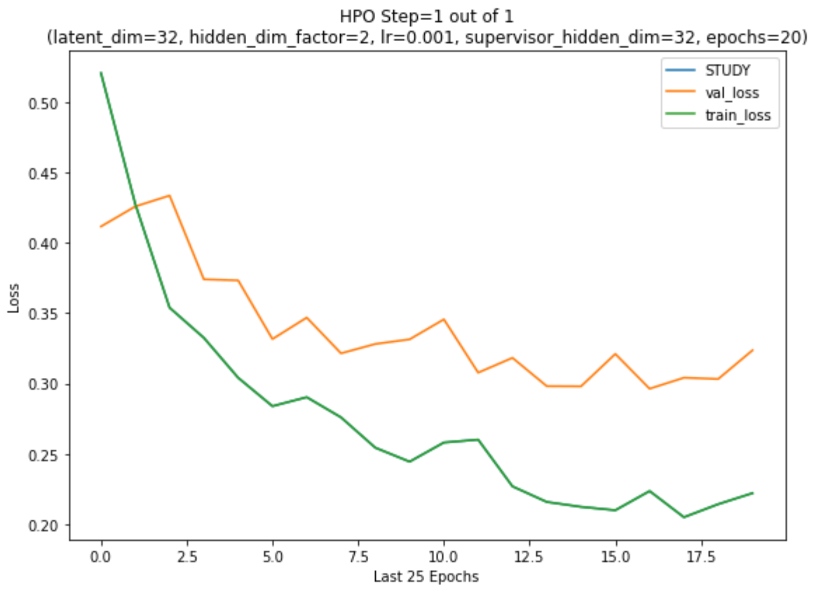
```
`Trainer.fit` stopped: `max_epochs=20` reached.
```

Out[35]:

	method	var	variable_type	metric	value
0	DirectPred	STUDY	categorical	balanced_acc	0.788235
1	DirectPred	STUDY	categorical	f1_score	0.799740
2	DirectPred	STUDY	categorical	kappa	0.567568
3	DirectPred	STUDY	categorical	average_auroc	0.883891
4	DirectPred	STUDY	categorical	average_aupr	0.783900

In [36]:

```
# 5. change supervisor_hidden_dim to 32
myparams = {'latent_dim': 32, 'hidden_dim_factor': 2, 'lr': 0.001, 'supervisor_hidden_dim': 32, 'epochs': 20}
model = flexynesis.DirectPred(config = myparams, dataset = train_dataset, target_variables=['STUDY'])
trainer = pl.Trainer(max_epochs=myparams['epochs'], default_root_dir="./", logger=False, enable_checkpointing=F
                     callbacks=[flexynesis.LiveLossPlot(myparams, 1, 1)])
# Fit the model
trainer.fit(model, train_loader, val_loader)
# evaluate the model performance on predicting the target variable
flexynesis.evaluate_wrapper("DirectPred", model.predict(test_dataset), test_dataset)
```



```
`Trainer.fit` stopped: `max_epochs=20` reached.
```

Out[36]:

	method	var	variable_type	metric	value
0	DirectPred	STUDY	categorical	balanced_acc	0.790196
1	DirectPred	STUDY	categorical	f1_score	0.810144
2	DirectPred	STUDY	categorical	kappa	0.584980
3	DirectPred	STUDY	categorical	average_auroc	0.877355
4	DirectPred	STUDY	categorical	average_aupr	0.793981

Warning!!: In reality, we don't select the best models based on performance on the test dataset.

The best model is selected based on the validation loss value, where the model parameters that yields the lowest validation loss is selected to be the best model.

The validation dataset which we use to compute the validation loss is basically a subset of the training dataset.

3. Automating the Hyperparameter Optimisation Procedure

What we did in the above section was to set random hyperparameters, build a model, evaluate the model and try different hyperparameters based on our previous model performance. However, this process can be quite time consuming and arbitrary. This process can be automated using a Bayesian approach, where the model training is sequentially done for a number of hyperparameter optimisation iterations.

Now, we are ready to do a model training using hyperparameter optimisation.

- `model_class` : We pick `DirectPred` (a fully connected network) for now.
- `config_name` : We use the default/built-in hyperparameter search space for `DirectPred` class.
- `target_variables` : 'STUDY' variable contains the type of disease
- `n_iter` : We do 5 iterations of hyperparameter optimisation. For demonstration purposes, we set it to a small number.
- `plot_losses` : We want to visualize how the training progresses.
- `early_stop_patience` : If a training does not show any signs of improving the performance on the `validation` part of the `train_dataset` for at least 10 epochs, we stop the training. This not only significantly decreases the amount spent on training by avoiding unnecessary continuation of unpromising training runs, but also helps avoid over-fitting the network on the training data.

Note 1: Notice how the hyperparameters using in different HPO steps change at each iteration.

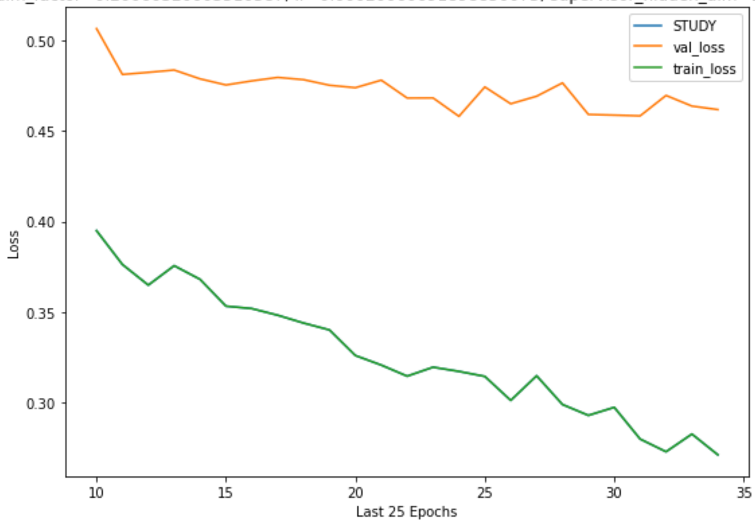
Note 2: Also notice that we are running the model for more epochs (500 by default) however, by using "early_stop_patience=10", we avoid lengthy training when validation performance is not improving.

Note 3: Try to follow the the loss curves and the used hyperparameters. See if you can spot which combination yields the lowest/best loss values.

Warning!!: In reality we need to set `n_iter` to higher values so that the optimizer can collect enough data points to learn trends in the parameter space.

In [37]:
```
# Define a tuner; See n_iter is the number of
tuner = flexynesis.HyperparameterTuning(train_dataset,
                                        model_class = flexynesis.DirectPred,
                                        config_name = "DirectPred",
                                        target_variables = ['STUDY'],
                                        n_iter=5,
                                        plot_losses=True,
                                        early_stop_patience=10)
### Perform Training
model, best_params = tuner.perform_tuning()
```

HPO Step=5 out of 5
(latent_dim=79, hidden_dim_factor=0.20060320003510387, lr=0.00020060691898850073, supervisor_hidden_dim=12, epochs=500, batch_size=64)



```
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
┃     Validate metric     ┃      DataLoader 0       ┃
│          STUDY          │   0.46175557374954224   │
│        val_loss         │   0.46175557374954224   │
```

```
Tuning Progress: 100%|██████████| 5/5 [00:43<00:00,  8.63s/it, Iteration=5, Best Loss=0.338]
[INFO] current best val loss: 0.3380390405654907; best params: {'latent_dim': 115, 'hidden_dim_factor': 0.434796
30032307554, 'lr': 0.0021607437861215903, 'supervisor_hidden_dim': 13, 'epochs': 500, 'batch_size': 64} since 2
hpo iterations
```

In [38]:
```
## See which hyperparameter combination was the best
best_params
```

Out[38]:
```
{'latent_dim': 115,
 'hidden_dim_factor': 0.43479630032307554,
 'lr': 0.0021607437861215903,
 'supervisor_hidden_dim': 13,
 'epochs': 22,
 'batch_size': 64}
```

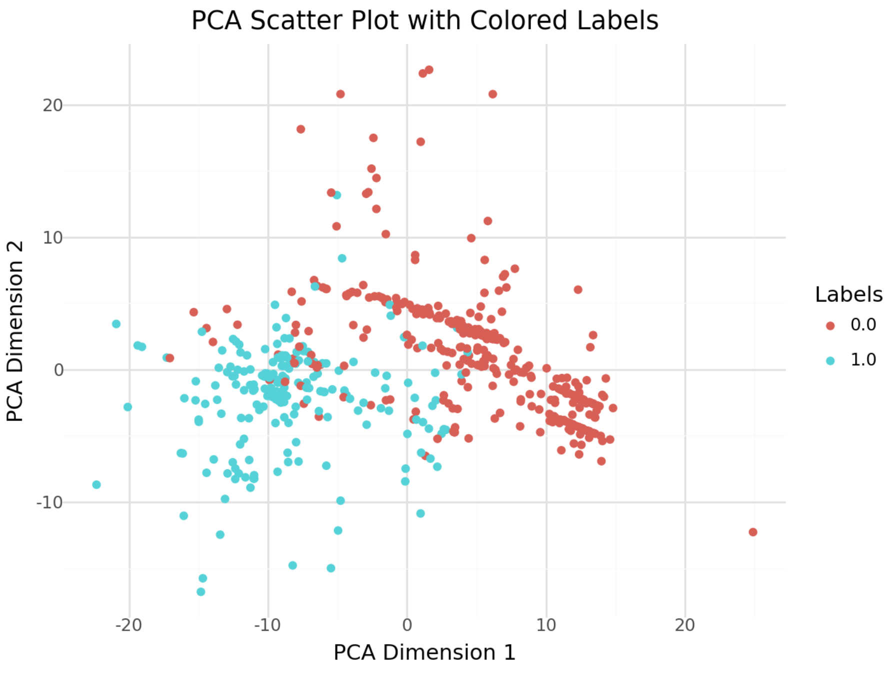
In [39]:
```
## Evaluate the model and visualising the results
flexynesis.evaluate_wrapper(method = 'DirectPred', y_pred_dict=model.predict(test_dataset), dataset = test_data
```

Out[39]:

	method	var	variable_type	metric	value
0	DirectPred	STUDY	categorical	balanced_acc	0.809150
1	DirectPred	STUDY	categorical	f1_score	0.830474
2	DirectPred	STUDY	categorical	kappa	0.628154
3	DirectPred	STUDY	categorical	average_auroc	0.870819
4	DirectPred	STUDY	categorical	average_aupr	0.731900

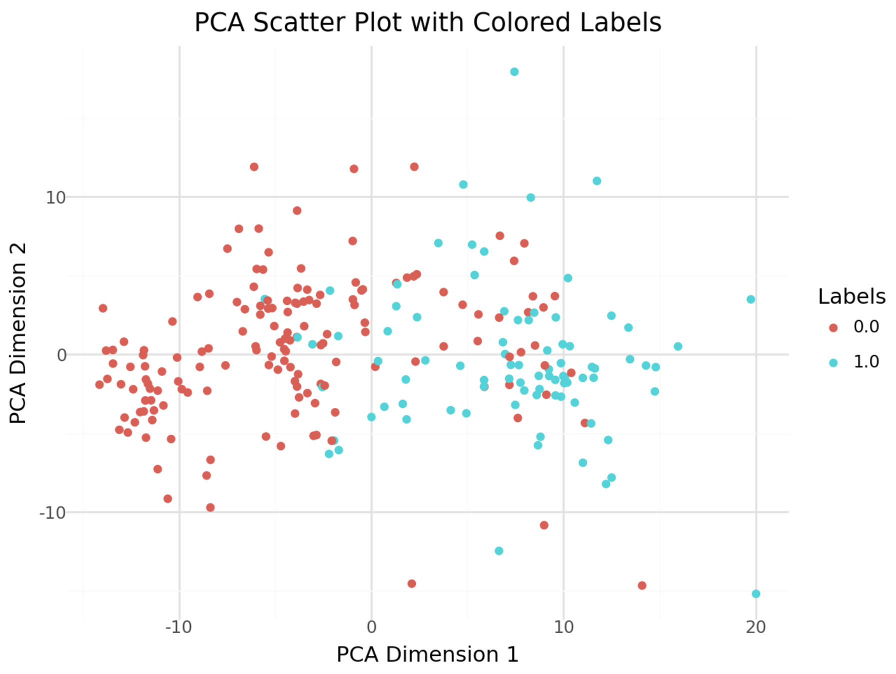
Let's extract the sample embeddings and make a PCA plot and color by the target variable

In [40]:
```
# Signature: model.transform(dataset)
# Docstring:
# Transforms the input data into a lower-dimensional representation using trained encoders.

# Args:
#     dataset: The dataset containing the input data.

# Returns:
#     pd.DataFrame: DataFrame containing the transformed data.
train_embeddings = model.transform(train_dataset)
flexynesis.plot_dim_reduced(train_embeddings, train_dataset.ann['STUDY'])
```



Repeat the same for the test dataset: extract sample embeddings for test dataset samples and make a PCA plot, colored by "STUDY" variable

In [41]:
```
test_embeddings = model.transform(test_dataset)
flexynesis.plot_dim_reduced(test_embeddings, test_dataset.ann['STUDY'])
```



3.1 Exercises

Exercise 1:

Look up what Harrell's C-index means and write down a simple description of what it measures.

In [ ]:
```
# reference in https://statisticaloddsandends.wordpress.com/2019/10/26/what-is-harrells-c-index/
# https://www.sciencedirect.com/science/article/pii/S1532046420301246
# a goodness of fit measure to evaluate risk models in survival analysis (on [0,1])
# the ability to assign higher risk scores to patients who experience the disease earlier

# each observation has a risk score eta_i (our response), and a time-to-event reponse T_i (the time until i dev
# find concordant pair of patients where eta_i > eta_j and T_i < T_j (i develops the disease earlier, hence lar
## only consider paris where: 1. both are not censored.
## 2. one is censored, say j, and we know T_j > T_i, then they are concordant pairs if eta_i > eta_j.
# C = #concordant pairs / (# concordant + # disconcordant)
```

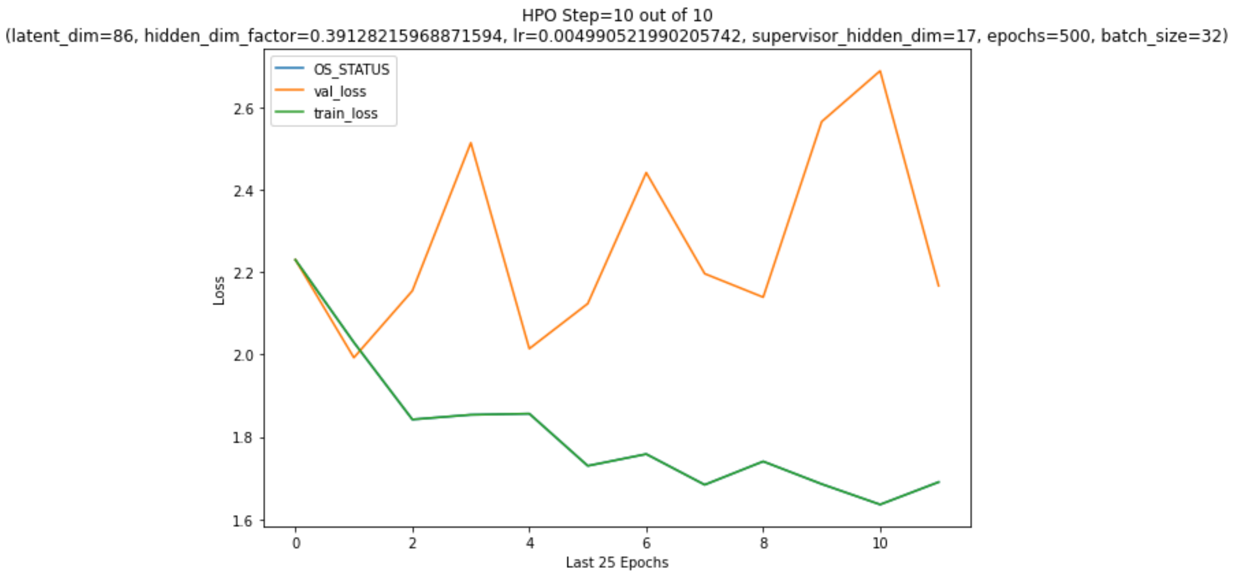
In [ ]:
```
# Values of c near 0.5 indicate that the risk score predictions are no better than a coin flip in
# determining which patient will live longer.
# Values near 1 indicate that the risk scores are good at determining which of two patients will have the disea
# Values near 0 means that the risk scores are worse than a coin flip: you might be better off
# concluding the opposite of what the risk scores tell you.
```

Exercise 2:

Now, you build a model using hyperparameter tuning (run at least 10 HPO steps) to predict the survival outcomes of patients. Evaluate the final model on test dataset, which computes the "C-index".

Feel free to cheat from the tutorial available here: https://github.com/BIMSBbioinfo/flexynesis/blob/main/examples/tutorials/survival_subtypes_LGG_GBM.ipynb See how "OS_STATUS" and "OS_MONTHS" were used.

In [43]:
```
# Define a tuner; See n_iter is the number of
tuner = flexynesis.HyperparameterTuning(train_dataset,
                                        model_class = flexynesis.DirectPred,
                                        config_name = "DirectPred",
                                        surv_event_var = 'OS_STATUS',
                                        surv_time_var = 'OS_MONTHS',
                                        target_variables = [],
                                        n_iter=10,
                                        plot_losses=True,
                                        early_stop_patience=10)
### Perform Training
model, best_params = tuner.perform_tuning()
```



```
Validation: |          | 0/? [00:00<?, ?it/s]
```

Validate metric	DataLoader 0
OS_STATUS val_loss	2.166504004004004 2.166504004004004

```
Tuning Progress: 100%|██████████| 10/10 [01:26<00:00,  8.67s/it, Iteration=10, Best Loss=1.85]
[INFO] current best val loss: 1.8504504504504504; best params: {'latent_dim': 24, 'hidden_dim_factor': 0.2503938
483744877, 'lr': 0.0003905005284080769, 'supervisor_hidden_dim': 29, 'epochs': 500, 'batch_size': 32} since 9 hp
o iterations
```

In [44]:
```
best_params
```

Out[44]:
```
{'latent_dim': 24,
 'hidden_dim_factor': 0.2503938483744877,
 'lr': 0.0003905005284080769,
 'supervisor_hidden_dim': 29,
 'epochs': 44,
 'batch_size': 32}
```

In [51]:
```
model
```

Out[51]:
```
DirectPred(
  (log_vars): ParameterDict(  (OS_STATUS): Parameter containing: [torch.FloatTensor of size 1])
  (encoders): ModuleList(
    (0): MLP(
      (layer_1): Linear(in_features=1237, out_features=309, bias=True)
      (layer_out): Linear(in_features=309, out_features=24, bias=True)
      (relu): ReLU()
      (dropout): Dropout(p=0.1, inplace=False)
      (batchnorm): BatchNorm1d(309, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): MLP(
      (layer_1): Linear(in_features=317, out_features=79, bias=True)
      (layer_out): Linear(in_features=79, out_features=24, bias=True)
      (relu): ReLU()
      (dropout): Dropout(p=0.1, inplace=False)
      (batchnorm): BatchNorm1d(79, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (MLPs): ModuleDict(
    (OS_STATUS): MLP(
      (layer_1): Linear(in_features=48, out_features=29, bias=True)
      (layer_out): Linear(in_features=29, out_features=1, bias=False)
      (relu): ReLU()
      (dropout): Dropout(p=0.1, inplace=False)
      (batchnorm): BatchNorm1d(29, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
)
```

In [52]:
```
flexynesis.evaluate_wrapper(method = 'DirectPred', y_pred_dict=model.predict(test_dataset), dataset = test_data
                           surv_event_var=model.surv_event_var, surv_time_var=model.surv_time_var)
```

Out[52]:

	method	var	variable_type	metric	value
0	DirectPred	OS_STATUS	numerical	cindex	0.69965

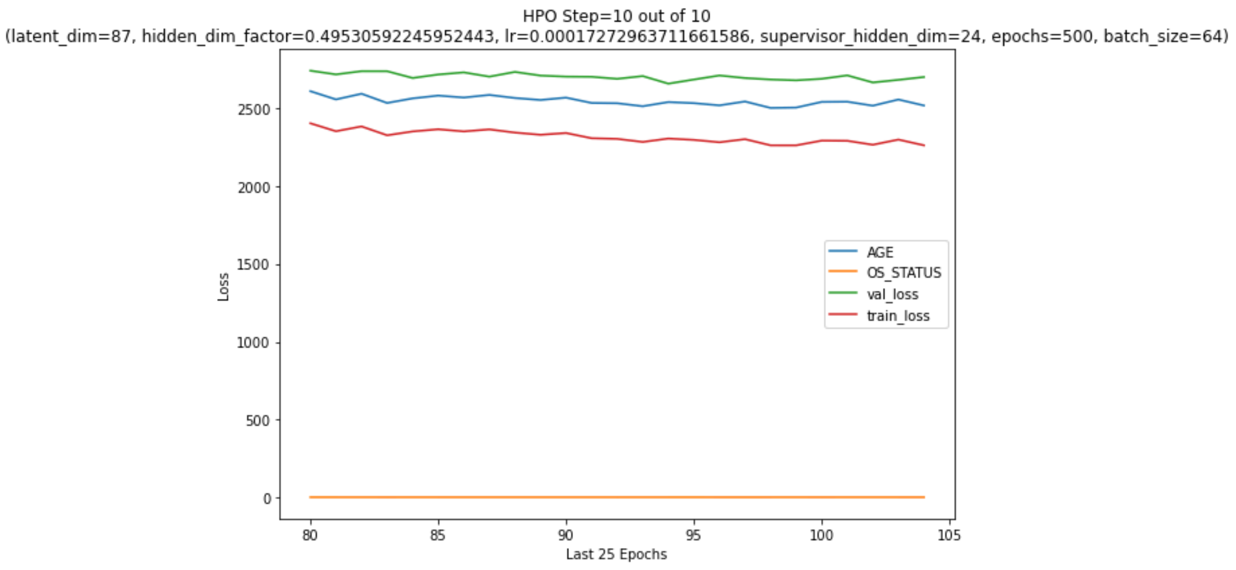
Exercise 3:

Again build a model using hyperparameter tuning to predict survival outcomes (as in Exercise 1), however, this time use additional clinical variables as targets.

```
flexynesis.HyperparameterTuning(train_dataset,
                                model_class = flexynesis.DirectPred,
                                config_name = "DirectPred",
                                surv_event_var="OS_STATUS",
                                surv_time_var="OS_MONTHS",
                                target_variables = [], => What other variables can you use
here? Try "AGE" and/or "HISTOLOGICAL_DIAGNOSIS" and see the model performance
                                ...
```

See if you can get a better C-index using additional target variables.

In [53]:
```
# Define a tuner; See n_iter is the number of
tuner = flexynesis.HyperparameterTuning(train_dataset,
                                        model_class = flexynesis.DirectPred,
                                        config_name = "DirectPred",
                                        surv_event_var = 'OS_STATUS',
                                        surv_time_var = 'OS_MONTHS',
                                        target_variables = ['AGE'],
                                        n_iter=10,
                                        plot_losses=True,
                                        early_stop_patience=10)
### Perform Training
model, best_params = tuner.perform_tuning()
```



```
Validation: |          | 0/? [00:00<?, ?it/s]
```

Validate metric	DataLoader 0
AGE OS_STATUS val_loss	2696.47412109375 2.7840283722636663 2699.25806148741

```
Tuning Progress: 100%|██████████| 10/10 [04:08<00:00, 24.87s/it, Iteration=10, Best Loss=205]
[INFO] current best val loss: 205.15267944335938; best params: {'latent_dim': 17, 'hidden_dim_factor': 0.3825214
4818823597, 'lr': 0.004912870548917598, 'supervisor_hidden_dim': 24, 'epochs': 500, 'batch_size': 128} since 1 h
po iterations
```

In [54]:
```
best_params
```

Out[54]:
```
{'latent_dim': 17,
 'hidden_dim_factor': 0.38252144818823597,
 'lr': 0.004912870548917598,
 'supervisor_hidden_dim': 24,
 'epochs': 79,
 'batch_size': 128}
```

In [55]:
```
flexynesis.evaluate_wrapper(method = 'DirectPred', y_pred_dict=model.predict(test_dataset), dataset = test_data
                           surv_event_var=model.surv_event_var, surv_time_var=model.surv_time_var)
```

Out[55]:

	method	var	variable_type	metric	value
0	DirectPred	AGE	numerical	mse	227.451242
1	DirectPred	AGE	numerical	r2	0.232289
2	DirectPred	AGE	numerical	pearson_corr	0.481964
3	DirectPred	OS_STATUS	numerical	cindex	0.726978

3.2 Survival-risk subtypes

Use the best model from the above exercises to inspect sample embeddings categorized by survival risk scores.

Let's group the samples by predicted survival risk scores into 2 groups and visualize the sample embeddings colored by risk subtypes.

Notice: You can use the code-below to get survival risk groups, however, notice that you must have built a model with "OS_STATUS" already.

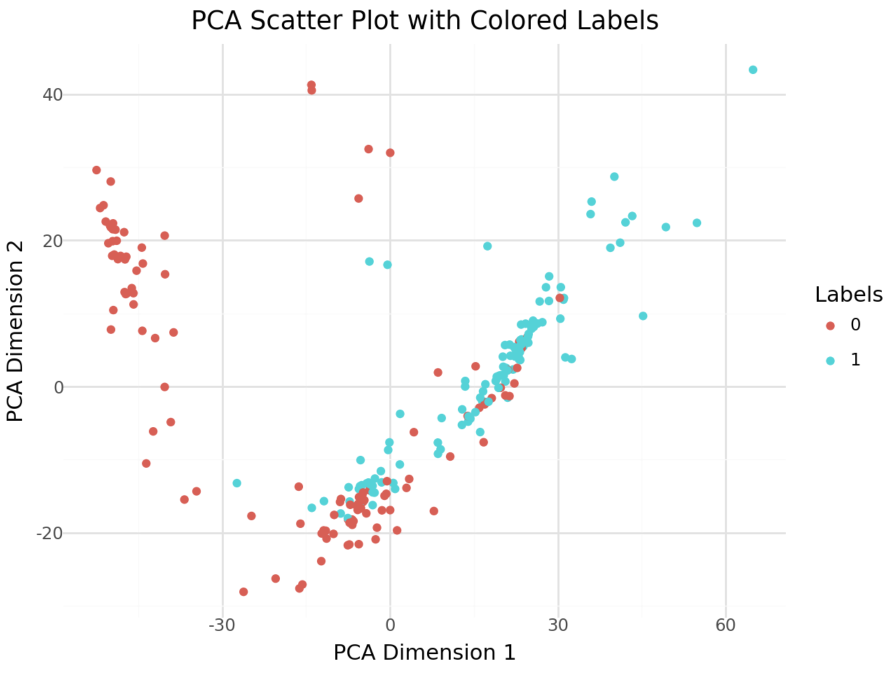
In [56]:
```python
# get model outputs for survival variable
outputs = model.predict(test_dataset)['OS_STATUS'].flatten()
risk_scores = np.exp(outputs)
# Define quantile thresholds
quantiles = np.quantile(risk_scores, [0.5])
# Assign groups based on quantiles
groups = np.digitize(risk_scores, quantiles)
```

In [60]:
```python
groups
```

Out[60]:
```
array([1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1,
       1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0])
```

In [61]:
```python
# Extract sample embeddings
E = model.transform(test_dataset)
```

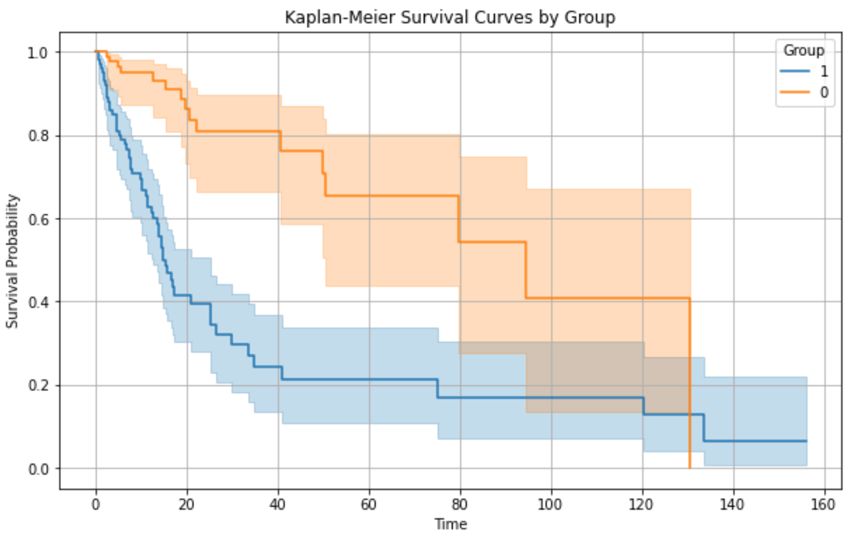
In [62]:
```python
flexynesis.plot_dim_reduced(E, groups)
```



Let's also see the Kaplan Meier Curves of the risk subtypes

In [63]:
```
# remove samples with NA values first
durations = test_dataset.ann['OS_MONTHS']
events = test_dataset.ann['OS_STATUS']
valid_indices = ~torch.isnan(durations) & ~torch.isnan(events)
```

In [64]:
```
flexynesis.plot_kaplan_meier_curves(durations[valid_indices], events[valid_indices], groups[valid_indices])
```



Finding survival-associated markers

We can also compute feature importance scores for prediction of overall survival.

In [65]:
```
model.compute_feature_importance(train_dataset, 'OS_STATUS')
```

In [66]:
```
# get top 10 features
flexynesis.get_important_features(model, var = 'OS_STATUS', top=10)
```

Out[66]:

	target_variable	target_class	target_class_label	layer	name	importance
0	OS_STATUS	0		mut	IDH1	0.480031
1	OS_STATUS	0		mut	ATRX	0.415870
2	OS_STATUS	0		mut	TP53	0.117678
3	OS_STATUS	0		mut	NF1	0.105058
4	OS_STATUS	0		mut	IDH2	0.071176
5	OS_STATUS	0		mut	MUC16	0.036255
6	OS_STATUS	0		mut	PIK3CA	0.034851
7	OS_STATUS	0		mut	CIC	0.028548
8	OS_STATUS	0		cna	MIR603	0.026571
9	OS_STATUS	0		mut	RELN	0.026166

Comparing top markers with clinical covariates

Let's build a linear Cox-PH model including the top 5 markers and other clinical variables such as histological diagnosis, disease type (STUDY), age, and sex.

In [68]:
```
# define a data.frame with clinical covariates and top markers along with survival endpoints
vars = ['AGE', 'SEX', 'HISTOLOGICAL_DIAGNOSIS', 'STUDY', 'OS_MONTHS', 'OS_STATUS']
# read clinical variables
df_clin = pd.concat(
    [pd.DataFrame({x: train_dataset.ann[x] for x in vars}, index=train_dataset.samples),
     pd.DataFrame({x: test_dataset.ann[x] for x in vars}, index=test_dataset.samples)],
    axis = 0)
# get top 5 survival markers and extract the input data for these markers for both training and test data
imp = flexynesis.get_important_features(model, var = 'OS_STATUS', top=5)
df_imp = pd.concat([train_dataset.get_feature_subset(imp), test_dataset.get_feature_subset(imp)], axis=0)
# get_feature_subset:
# Get a subset of data matrices corresponding to specified features and concatenate them into a pandas DataFram

# Args:
#     feature_df (pandas.DataFrame): A DataFrame which contains at least two columns: 'layer' and 'name'.

# Returns:
```

```
#     A pandas DataFrame that concatenates the data matrices for the specified features from all layers.

# combine markers with clinical variables
df = pd.concat([df_imp, df_clin], axis = 1)
# remove samples without survival endpoints
df = df[df['OS_STATUS'].notna()]
df.head()
```

Out[68]:

	mut_IDH1	mut_ATRX	mut_TP53	mut_NF1	mut_IDH2	AGE	SEX	HISTOLOGICAL_DIAGNOSIS	STUDY	OS_MONTHS
TCGA-DU-6405	-1.018150	-0.585658	-0.809174	-0.267004	-0.148522	51.0	0.0	0.0	0.0	19.9
TCGA-06-2564	-1.018150	-0.585658	-0.809174	-0.267004	-0.148522	50.0	1.0	1.0	1.0	5.9
TCGA-WH-A86K	0.982173	-0.585658	1.235829	-0.267004	-0.148522	65.0	1.0	0.0	0.0	5.3
TCGA-QH-A65X	-1.018150	-0.585658	-0.809174	-0.267004	6.733003	28.0	0.0	2.0	0.0	7.8
TCGA-HT-7601	0.982173	-0.585658	1.235829	-0.267004	-0.148522	30.0	0.0	0.0	0.0	5.0

In [70]:
```
imp
```

Out[70]:

	target_variable	target_class	target_class_label	layer	name	importance
0	OS_STATUS	0		mut	IDH1	0.480031
1	OS_STATUS	0		mut	ATRX	0.415870
2	OS_STATUS	0		mut	TP53	0.117678
3	OS_STATUS	0		mut	NF1	0.105058
4	OS_STATUS	0		mut	IDH2	0.071176

In [71]:
```
[pd.DataFrame({x: train_dataset.ann[x] for x in vars}, index=train_dataset.samples),
     pd.DataFrame({x: test_dataset.ann[x] for x in vars}, index=test_dataset.samples)]
```

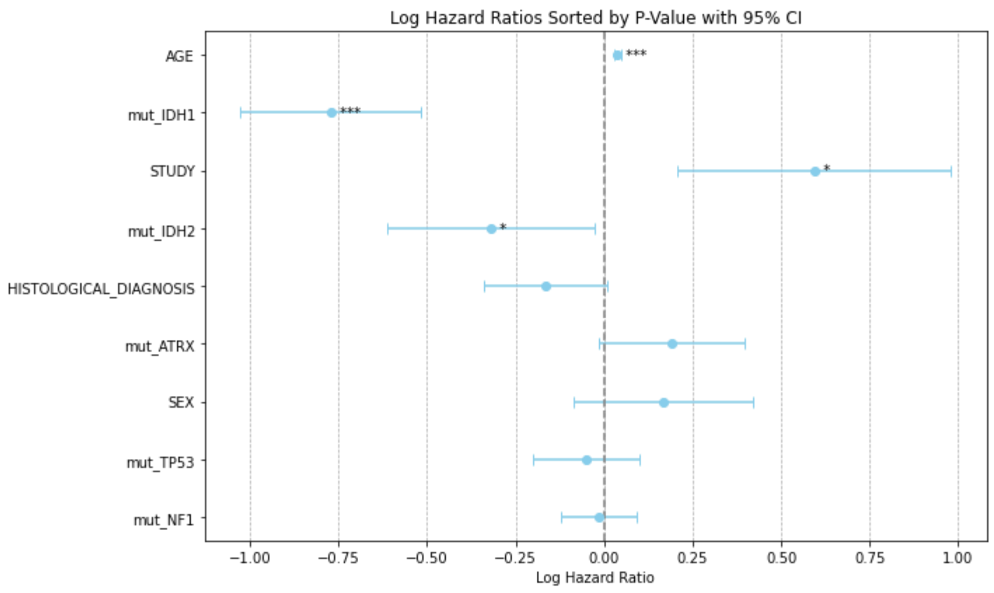
Out[71]:
```
[                AGE  SEX  HISTOLOGICAL_DIAGNOSIS  STUDY  OS_MONTHS  OS_STATUS
 TCGA-DU-6405  51.0  0.0                     0.0    0.0       19.9        1.0
 TCGA-06-2564  50.0  1.0                     1.0    1.0        5.9        0.0
 TCGA-WH-A86K  65.0  1.0                     0.0    0.0        5.3        0.0
 TCGA-QH-A65X  28.0  0.0                     2.0    0.0        7.8        0.0
 TCGA-HT-7601  30.0  0.0                     0.0    0.0        5.0        0.0
 ...            ...  ...                     ...    ...        ...        ...
 TCGA-DU-7294  53.0  0.0                     3.0    0.0       94.3        0.0
 TCGA-DU-6397  45.0  1.0                     3.0    0.0       46.0        1.0
 TCGA-06-0211  47.0  1.0                     1.0    1.0       11.8        1.0
 TCGA-E1-A7YN   NaN  NaN                     NaN    0.0        NaN        NaN
 TCGA-HT-A614  47.0  1.0                     2.0    0.0        2.7        0.0

 [556 rows x 6 columns],
                AGE  SEX  HISTOLOGICAL_DIAGNOSIS  STUDY  OS_MONTHS  OS_STATUS
 TCGA-41-6646  73.0  0.0                     1.0    1.0        7.8        0.0
 TCGA-QH-A870  38.0  0.0                     2.0    0.0        0.1        0.0
 TCGA-06-0129  30.0  1.0                     1.0    1.0       33.6        1.0
 TCGA-19-5950  52.0  0.0                     1.0    1.0       11.3        0.0
 TCGA-32-1980  72.0  1.0                     1.0    1.0        1.2        1.0
 ...            ...  ...                     ...    ...        ...        ...
 TCGA-74-6575  73.0  0.0                     1.0    1.0       17.5        0.0
 TCGA-DU-6542  25.0  1.0                     2.0    0.0        7.7        0.0
 TCGA-26-1439  63.0  1.0                     1.0    1.0       13.9        1.0
 TCGA-VV-A829  44.0  1.0                     2.0    0.0       27.0        0.0
 TCGA-27-2521  34.0  1.0                     1.0    1.0       10.4        0.0

 [238 rows x 6 columns]]
```

In [72]:
```
# build a cox model, not deep learning models
coxm = flexynesis.build_cox_model(df, 'OS_MONTHS', 'OS_STATUS')
```

```
No low variance features were removed based on event conditioning.
```

In [73]:
```
# visualize log-hazard ratios sorted by p-values
flexynesis.plot_hazard_ratios(coxm)
```

```
/gnu/store/88qdfdp9h0003w4wp1rv46khc3dpp8j6-profile/lib/python3.10/site-packages/flexynesis/utils.py:764: Future
Warning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will alw
ays be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]
`
```



3.3 Final Exercise

- Inspect the top 10 markers from section 3.2 and see if they have been characterized in the literature as important markers for Glioma disease progression.

In [75]:
```
df1=flexynesis.get_important_features(model, var = 'OS_STATUS', top=10)
df1
```

Out[75]:

	target_variable	target_class	target_class_label	layer	name	importance
0	OS_STATUS	0		mut	IDH1	0.480031
1	OS_STATUS	0		mut	ATRX	0.415870
2	OS_STATUS	0		mut	TP53	0.117678
3	OS_STATUS	0		mut	NF1	0.105058
4	OS_STATUS	0		mut	IDH2	0.071176
5	OS_STATUS	0		mut	MUC16	0.036255
6	OS_STATUS	0		mut	PIK3CA	0.034851
7	OS_STATUS	0		mut	CIC	0.028548
8	OS_STATUS	0		cna	MIR603	0.026571
9	OS_STATUS	0		mut	RELN	0.026166

In [76]:
```
df1.name.values
```

Out[76]:
```
array(['IDH1', 'ATRX', 'TP53', 'NF1', 'IDH2', 'MUC16', 'PIK3CA', 'CIC',
       'MIR603', 'RELN'], dtype=object)
```

In [ ]:
```
# IDH1/2, ATRX: https://pmc.ncbi.nlm.nih.gov/articles/PMC8508830/, https://pmc.ncbi.nlm.nih.gov/articles/PMC282
# TP53: https://pmc.ncbi.nlm.nih.gov/articles/PMC6162501/#sec2-cancers-10-00297
# NF1: https://pmc.ncbi.nlm.nih.gov/articles/PMC5739070/
# MUC16: https://www.sciencedirect.com/science/article/pii/S2210776222002812?via%3Dihub
# PIK3CA: https://pmc.ncbi.nlm.nih.gov/articles/PMC8743979/
# CIC: https://www.nature.com/articles/s41467-018-08087-9
# MIR603: https://www.sciencedirect.com/science/article/pii/S0304383515000944
# RELN: https://onlinelibrary.wiley.com/doi/10.1111/bpa.12584
```