In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split

import torch
from torch import nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
import pytorch_lightning as pl
import torchmetrics
from pytorch_lightning.callbacks import TQDMProgressBar

from tqdm import tqdm

%matplotlib inline
```

1 Feed-forward Neural Network for Binary Classification

1 - a. Data preprocessing and exploratory data analysis.

Load data

In [2]:
```
df = pd.read_csv('BRCA-prognosis-train.csv')
df
```

```
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3146: DtypeWarning: Columns (641,647,649,651,653,654,655,656,657,658,659,661) have mixed types.Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[2]:

	brca1	brca2	palb2	pten	tp53	atm	cdh1	chek2	nbn	nf1	...	ppp2cb_mut	smarcd1_mut	nras_mut	ndfip1_mut	hras_mut	prps2_mut	smarcb1_mut	stmn2_mut	sia
0	0.6522	-1.0002	0.2878	0.4661	-1.1925	1.2016	-0.7379	-0.5657	0.5885	1.0009	...	0	0	0	0	0	0	0	0	
1	-1.1329	2.7485	-0.7175	0.9446	-0.2574	-1.2881	-0.9623	-0.4673	0.2962	-0.9181	...	0	0	0	0	0	0	0	0	
2	-0.6247	0.9146	-0.0536	-0.0933	0.0326	-1.7125	-1.0493	-0.6886	-0.5311	-1.7056	...	0	0	0	0	0	0	0	0	
3	0.5877	-2.2261	2.4252	-0.6472	-1.7071	0.8207	-0.4278	3.4991	-0.5951	-1.0470	...	0	0	0	0	0	0	0	0	
4	-0.2048	-1.2539	-1.7408	0.2214	-0.0545	0.9722	0.8895	0.9785	-0.3633	-1.1611	...	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1399	-0.8468	0.4818	0.6131	0.0600	-0.6060	1.7714	0.8917	0.1010	-0.8821	-0.7944	...	0	0	0	0	0	0	0	0	
1400	-0.1844	0.0005	0.3751	-0.7222	-1.8560	0.7996	0.1025	0.7319	2.6620	-0.0314	...	0	0	0	0	0	0	0	0	
1401	1.4376	-0.3174	1.3291	0.4604	0.7625	-0.3105	-2.8114	0.1046	0.7327	0.1248	...	0	0	0	0	0	0	0	0	
1402	-1.2448	-2.2417	0.5508	0.8508	-0.6159	1.5829	0.0555	1.3599	-0.5648	-0.5968	...	0	0	0	0	0	0	0	0	
1403	-0.6506	-0.1476	-0.7637	1.1327	-0.3060	-1.4050	0.9773	-0.3467	-0.8001	-0.7799	...	0	0	0	0	0	0	0	0	

1404 rows × 663 columns

In [3]:
```
sum(list(df.isnull().sum()))
```

Out[3]:
```
0
```

- No missing values in Labeled Data.

Summary Tables

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1404 entries, 0 to 1403
Columns: 663 entries, brca1 to outcome
dtypes: float64(489), int64(2), object(172)
memory usage: 7.1+ MB
```

In [7]:
```
df.describe()
```

Out[7]:

	brca1	brca2	palb2	pten	tp53	atm	cdh1	chek2	nbn	nf1	...	srd5a3	st7	star	tnk2	
count	1404.000000	1404.000000	1404.000000	1404.000000	1404.000000	1404.000000	1404.000000	1404.000000	1404.000000	1404.000000	...	1404.000000	1404.000000	1404.000000	1404.000000	14
mean	0.000090	-0.033844	0.024035	-0.011157	0.011936	0.025500	0.014164	0.008059	0.011962	-0.006756	...	-0.004123	-0.004640	-0.006748	0.018110	
std	0.989816	0.999775	0.994332	1.008835	0.988798	0.985012	0.988141	0.981519	1.003056	0.976198	...	1.001727	0.999874	1.002540	0.995641	
min	-2.335600	-2.708900	-3.742000	-5.932800	-2.481200	-3.170500	-3.323700	-2.688700	-3.321500	-2.968600	...	-2.719400	-4.982700	-2.936400	-3.833300	
25%	-0.723200	-0.721100	-0.656775	-0.575600	-0.649175	-0.624225	-0.441425	-0.704025	-0.666425	-0.670250	...	-0.669325	-0.619375	-0.634175	-0.642400	
50%	-0.130150	-0.110750	0.004550	0.094550	-0.004450	0.030550	0.129200	-0.118950	0.001700	-0.052700	...	-0.156000	-0.053750	-0.013600	0.014300	
75%	0.549875	0.590750	0.661600	0.638850	0.655075	0.718300	0.667350	0.554775	0.642600	0.647425	...	0.495325	0.556925	0.571425	0.664675	
max	4.554200	3.737900	4.615000	2.630700	4.149200	3.824200	2.632500	3.981500	5.371300	3.816200	...	6.329000	4.571300	12.742300	3.240300	

8 rows × 491 columns

- 663-491 = 172 features contain non-numeric values (All of them are features related with mutant gene). So I will covert string values in these features into numeric values.

Clean Data - convert all string values to numeric values

In [8]:
```
# list features with string values.
col_names_mutGene = list(df.columns[df.columns.str.contains(pat = '_mut')])
col_names_with_mutGene = []
for i in col_names_mutGene:
    k = i[:-4]
    col_names_with_mutGene.append(k)

#  convert all string values in these features to numeric values
for col in col_names_mutGene:
    df[col] = df[col].apply(lambda x: 0 if (x=='0' or x==0) else 1)
```

Re-do Summary Tables for Labeled Data

In [9]:
```
df.describe()
```

Out[9]:

	brca1	brca2	palb2	pten	tp53	atm	cdh1	chek2	nbn	nf1	...	ppp2cb_mut	smarcd1_mut	nras_mut	ndfip1_mut	
count	1404.000000	1404.000000	1404.000000	1404.000000	1404.000000	1404.000000	1404.000000	1404.000000	1404.000000	1404.000000	...	1404.000000	1404.000000	1404.000000	1404.000000	1
mean	0.000090	-0.033844	0.024035	-0.011157	0.011936	0.025500	0.014164	0.008059	0.011962	-0.006756	...	0.001425	0.001425	0.001425	0.000712	
std	0.989816	0.999775	0.994332	1.008835	0.988798	0.985012	0.988141	0.981519	1.003056	0.976198	...	0.037729	0.037729	0.037729	0.026688	
min	-2.335600	-2.708900	-3.742000	-5.932800	-2.481200	-3.170500	-3.323700	-2.688700	-3.321500	-2.968600	...	0.000000	0.000000	0.000000	0.000000	
25%	-0.723200	-0.721100	-0.656775	-0.575600	-0.649175	-0.624225	-0.441425	-0.704025	-0.666425	-0.670250	...	0.000000	0.000000	0.000000	0.000000	
50%	-0.130150	-0.110750	0.004550	0.094550	-0.004450	0.030550	0.129200	-0.118950	0.001700	-0.052700	...	0.000000	0.000000	0.000000	0.000000	
75%	0.549875	0.590750	0.661600	0.638850	0.655075	0.718300	0.667350	0.554775	0.642600	0.647425	...	0.000000	0.000000	0.000000	0.000000	

	brca1	brca2	palb2	pten	tp53	atm	cdh1	chek2	nbn	nf1	...	ppp2cb_mut	smarcd1_mut	nras_mut	ndfip1_mut
max	4.554200	3.737900	4.615000	2.630700	4.149200	3.824200	2.632500	3.981500	5.371300	3.816200	...	1.000000	1.000000	1.000000	1.000000

8 rows × 663 columns

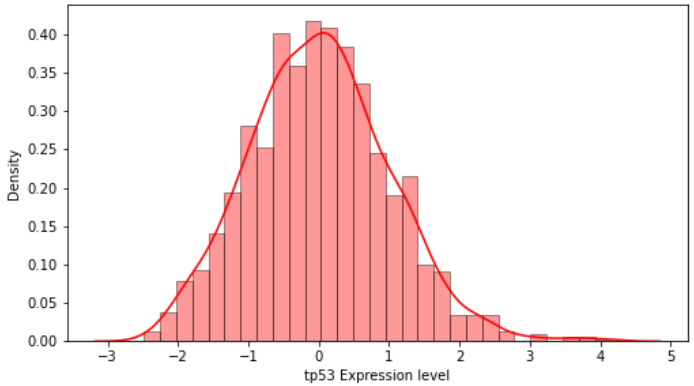
Plot histogram of the tp53 gene expression, tp53 mutant status, map3k1 gene expression, and map3k1_mut status

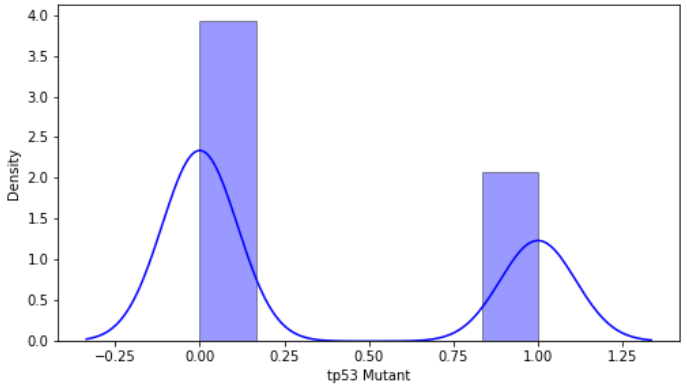
In [10]:

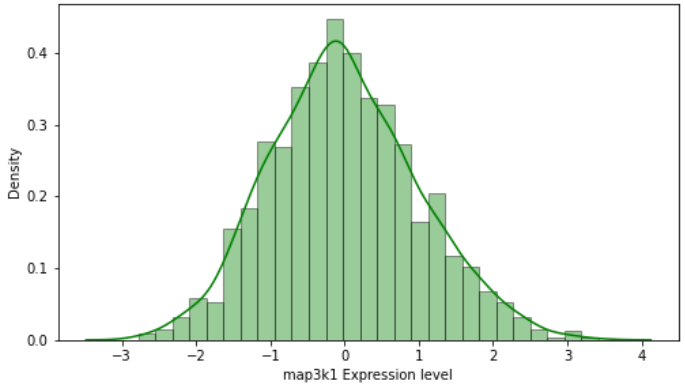
```
fig, axes = plt.subplots(2, 2, figsize=(18, 10))

tp53 = sns.distplot(df['tp53'], hist = True, hist_kws = {"edgecolor":"black"}, ax=axes[0,0], color = "red")
tp53.set( xlabel = "tp53 Expression level")

tp53_mut = sns.distplot(df['tp53_mut'], hist = True, hist_kws = {"edgecolor":"black"}, ax=axes[0,1], color = "blue")
tp53_mut.set( xlabel = "tp53 Mutant")

map3k1 = sns.distplot(df['map3k1'], hist = True, hist_kws = {"edgecolor":"black"}, ax=axes[1,0], color = "green")
map3k1.set( xlabel = "map3k1 Expression level")

map3k1_mut = sns.distplot(df['map3k1_mut'], hist = True, hist_kws = {"edgecolor":"black"}, ax=axes[1,1], color = "orange")
map3k1_mut.set( xlabel = "map3k1 Mutant")
```

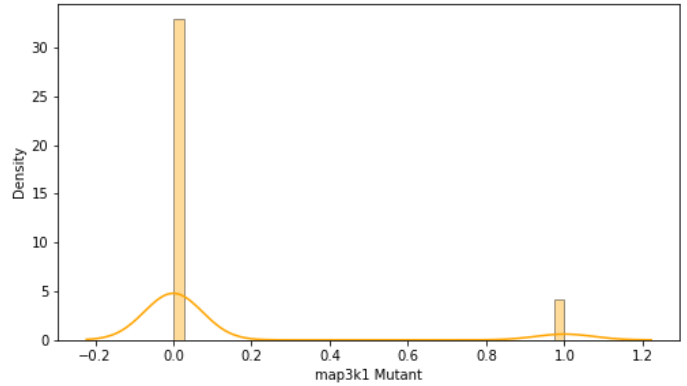
```
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed i
n a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histogram
s).
  warnings.warn(msg, FutureWarning)
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed i
n a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histogram
s).
  warnings.warn(msg, FutureWarning)
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed i
n a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histogram
s).
  warnings.warn(msg, FutureWarning)
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed i
n a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histogram
s).
  warnings.warn(msg, FutureWarning)
```

Out[10]: [Text(0.5, 0, 'map3k1 Mutant')]









Boxplot for gene expression of different group (better prognosis vs worse prognosis ; Non-Mutant vs Mutant)

In [11]:
```
df_for_box = df[['tp53','tp53_mut','map3k1','map3k1_mut', 'brca2','brca2_mut', 'erbb2', 'erbb2_mut','outcome']].copy()

df_for_box = df_for_box.assign(tp53_mut = np.where(df_for_box['tp53_mut']==0,'Non-Mutant','Mutant'),
                               map3k1_mut = np.where(df_for_box['map3k1_mut']==0,'Non-Mutant','Mutant'),
                               brca2_mut = np.where(df_for_box['brca2_mut']==0,'Non-Mutant','Mutant'),
                               erbb2_mut = np.where(df_for_box['erbb2_mut']==0,'Non-Mutant','Mutant'),
                               outcome = np.where(df_for_box['outcome']==0,'better prognosis','worse prognosis'))

df_for_box
```

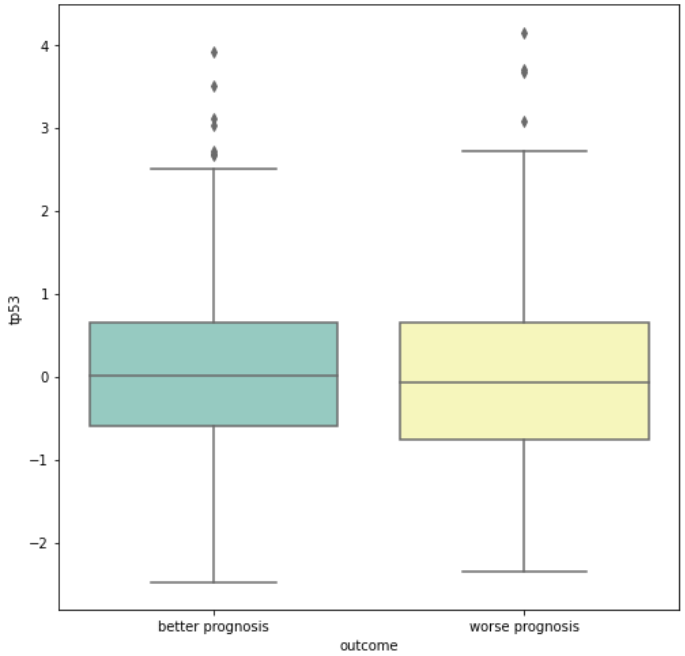
Out[11]:

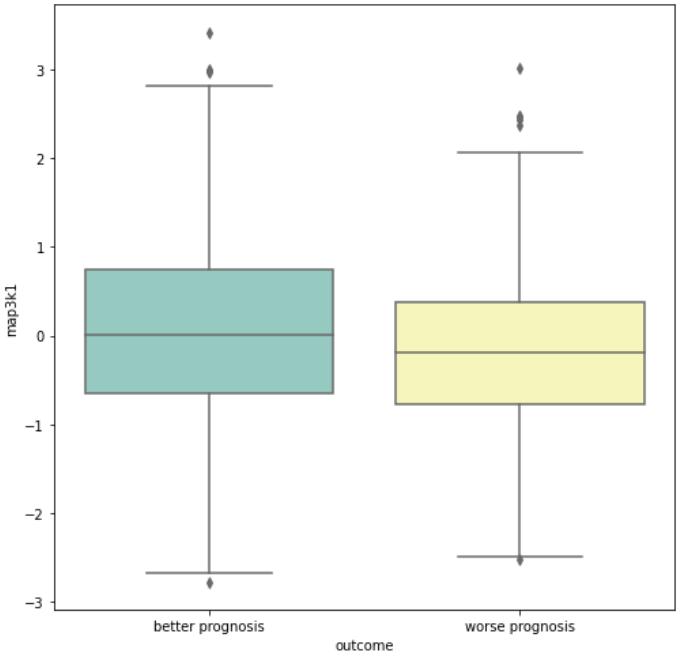
	tp53	tp53_mut	map3k1	map3k1_mut	brca2	brca2_mut	erbb2	erbb2_mut	outcome
0	-1.1925	Non-Mutant	-0.9006	Non-Mutant	-1.0002	Non-Mutant	0.0183	Non-Mutant	better prognosis
1	-0.2574	Non-Mutant	-0.4967	Non-Mutant	2.7485	Non-Mutant	-0.8339	Non-Mutant	worse prognosis
2	0.0326	Non-Mutant	-0.4701	Mutant	0.9146	Non-Mutant	0.0019	Non-Mutant	better prognosis
3	-1.7071	Non-Mutant	-1.2749	Non-Mutant	-2.2261	Non-Mutant	-0.8575	Non-Mutant	better prognosis
4	-0.0545	Non-Mutant	0.4572	Non-Mutant	-1.2539	Non-Mutant	0.1026	Non-Mutant	worse prognosis
...	...	...	...	...	...	...	...	...	...
1399	-0.6060	Mutant	-0.1359	Non-Mutant	0.4818	Non-Mutant	-0.7871	Non-Mutant	worse prognosis
1400	-1.8560	Mutant	0.0691	Non-Mutant	0.0005	Non-Mutant	1.6321	Non-Mutant	better prognosis
1401	0.7625	Non-Mutant	0.4242	Non-Mutant	-0.3174	Non-Mutant	-0.4242	Non-Mutant	better prognosis

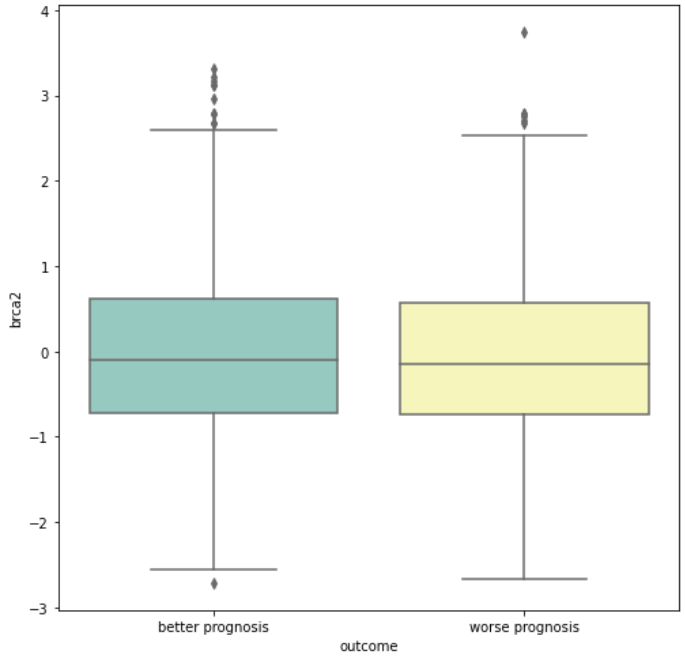
	tp53	tp53_mut	map3k1	map3k1_mut	brca2	brca2_mut	erbb2	erbb2_mut	outcome
1402	-0.6159	Mutant	0.4374	Non-Mutant	-2.2417	Non-Mutant	-1.5104	Non-Mutant	worse prognosis
1403	-0.3060	Non-Mutant	-0.0467	Non-Mutant	-0.1476	Non-Mutant	2.2038	Non-Mutant	worse prognosis

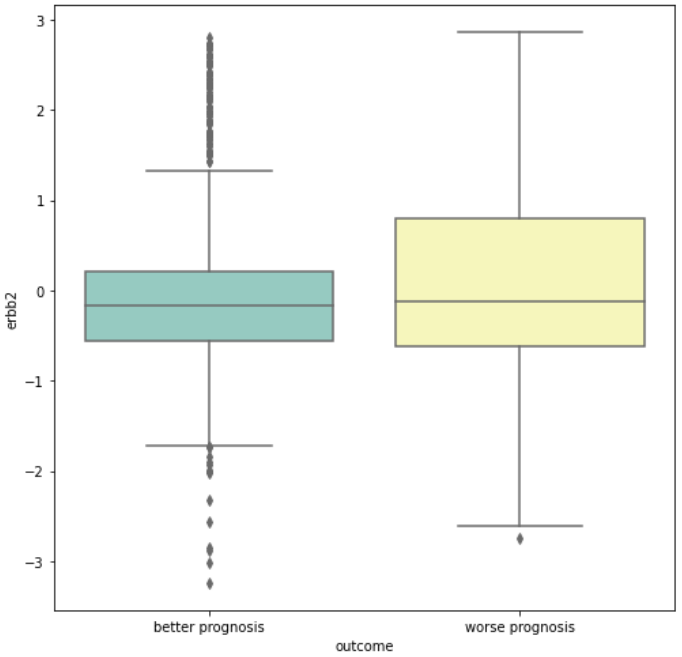
1404 rows × 9 columns

In [12]:
```
fig, axes = plt.subplots(2, 2, figsize=(18, 18))

tp53 = sns.boxplot(x='outcome', y='tp53', ax=axes[0,0], data=df_for_box, palette='Set3')
map3k1 = sns.boxplot(x='outcome', y='map3k1', ax=axes[0,1], data=df_for_box, palette='Set3')
brca2 = sns.boxplot(x='outcome', y='brca2', ax=axes[1,0], data=df_for_box, palette='Set3')
erbb2 = sns.boxplot(x='outcome', y='erbb2', ax=axes[1,1], data=df_for_box, palette='Set3')
```



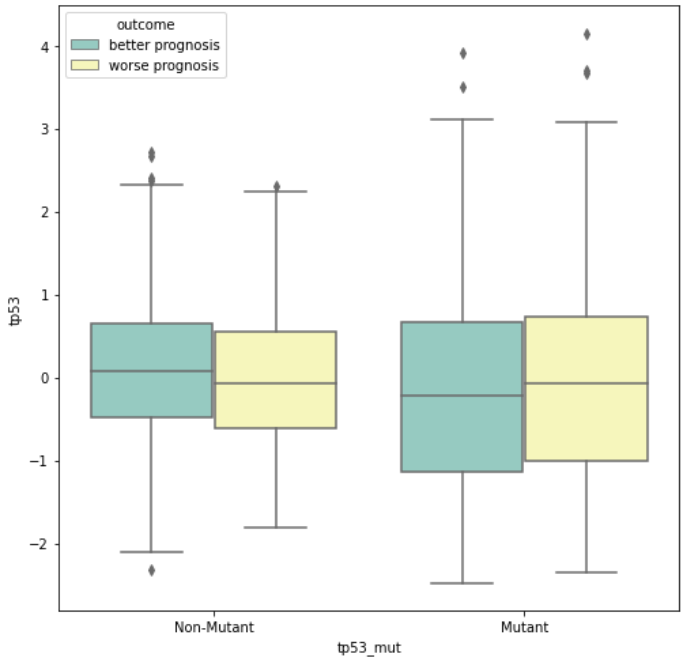


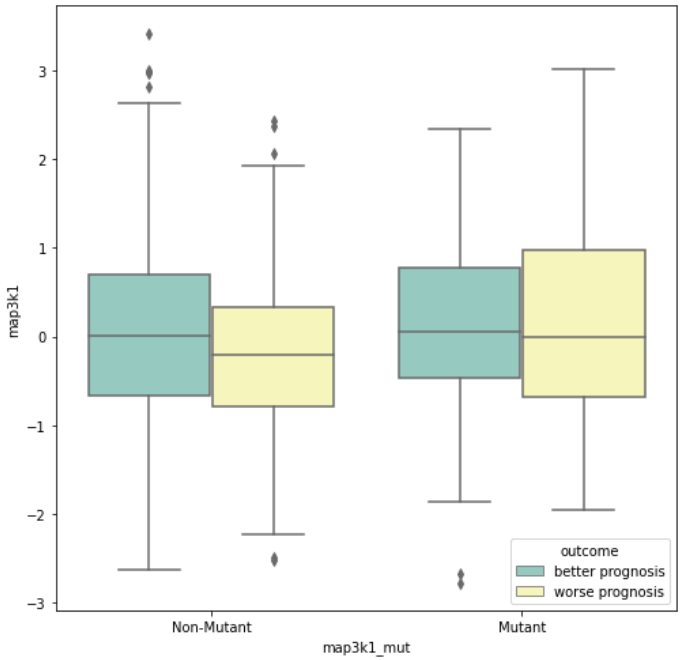


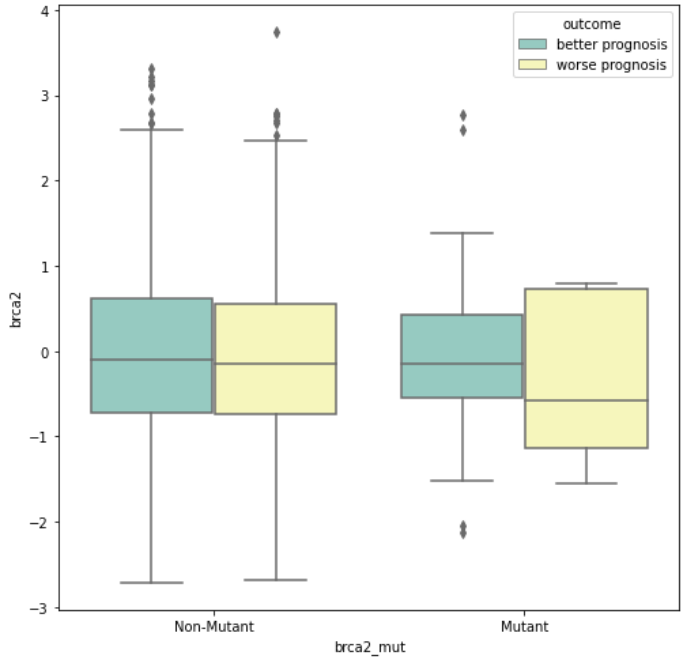


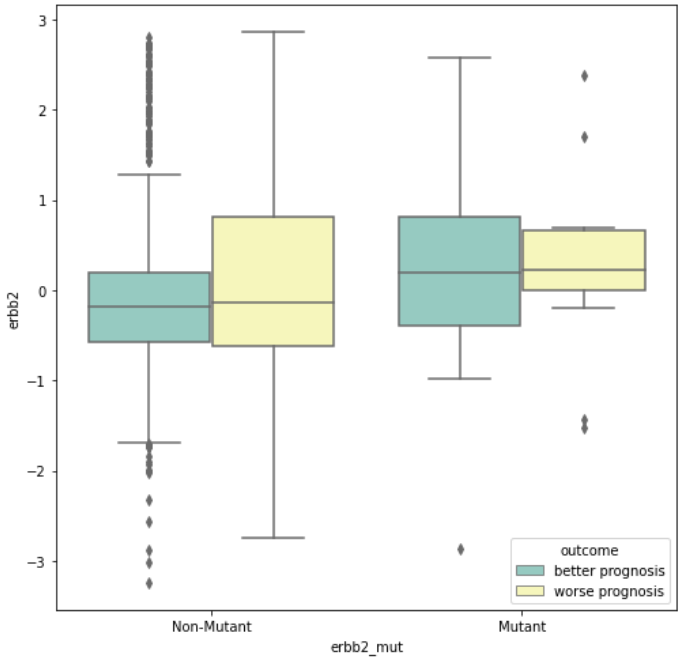
In [13]:
```
fig, axes = plt.subplots(2, 2, figsize=(18, 18))

tp53 = sns.boxplot(x='tp53_mut', y='tp53', hue='outcome', ax=axes[0,0], data=df_for_box, palette='Set3')
```

```
map3k1 = sns.boxplot(x='map3k1_mut', y='map3k1', hue='outcome', ax=axes[0,1], data=df_for_box, palette='Set3')
brca2 = sns.boxplot(x='brca2_mut', y='brca2', hue='outcome', ax=axes[1,0], data=df_for_box, palette='Set3')
erbb2 = sns.boxplot(x='erbb2_mut', y='erbb2', hue='outcome', ax=axes[1,1], data=df_for_box, palette='Set3')
```









From above summary statistics and plots:

- It seems that labeled data contains too many features in data. (1404 samples vs 662 features)
- Some gene expression doesn't have difference between patients with better prognosis and patients with worse prognosis.
- As results, I will do t-test for all features to find out which features have significant difference ($p < 0.05$) between patients with better prognosis and patients with worse prognosis.

Do t-test for all features:

In [14]:
```
from scipy import stats
import matplotlib.pyplot as plt
```

In [15]:
```
# Split data into 2 groups (patients with better prognosis and patients with worse prognosis)
outcome_0 = df[(df['outcome'] == 0)].copy()
outcome_1 = df[(df['outcome'] == 1)].copy()
```

In [16]:
```
outcome_0
```

Out[16]:

	brca1	brca2	palb2	pten	tp53	atm	cdh1	chek2	nbn	nf1	...	ppp2cb_mut	smarcd1_mut	nras_mut	ndfip1_mut	hras_mut	prps2_mut	smarcb1_mut	stmn2_mut	sial
0	0.6522	-1.0002	0.2878	0.4661	-1.1925	1.2016	-0.7379	-0.5657	0.5885	1.0009	...	0	0	0	0	0	0	0	0	
2	-0.6247	0.9146	-0.0536	-0.0933	0.0326	-1.7125	-1.0493	-0.6886	-0.5311	-1.7056	...	0	0	0	0	0	0	0	0	
3	0.5877	-2.2261	2.4252	-0.6472	-1.7071	0.8207	-0.4278	3.4991	-0.5951	-1.0470	...	0	0	0	0	0	0	0	0	
5	0.2366	-1.2391	0.8534	-0.7693	-0.4385	-0.4195	-0.7229	-0.4703	-0.9297	0.2410	...	0	0	0	0	0	0	0	0	
6	0.7570	-0.1874	1.9002	0.2603	0.7778	2.3716	1.5846	1.1316	-2.1736	0.9645	...	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1395	-0.8528	0.3132	0.4743	0.6641	-0.0636	0.2148	-0.2227	-0.2880	0.5620	0.7581	...	0	0	0	0	0	0	0	0	
1396	-0.5667	-0.5992	-0.8817	0.8963	-0.5029	-0.7141	-0.2060	-0.8106	0.0465	0.0462	...	0	0	0	0	0	0	0	0	
1398	-0.9521	-1.8101	-2.0872	1.1617	-0.5057	2.1591	-0.2501	-1.0621	-0.8328	-0.1748	...	0	0	0	0	0	0	0	0	
1400	-0.1844	0.0005	0.3751	-0.7222	-1.8560	0.7996	0.1025	0.7319	2.6620	-0.0314	...	0	0	0	0	0	0	0	0	
1401	1.4376	-0.3174	1.3291	0.4604	0.7625	-0.3105	-2.8114	0.1046	0.7327	0.1248	...	0	0	0	0	0	0	0	0	

960 rows × 663 columns

In [17]:
```
outcome_1
```

Out[17]:

	brca1	brca2	palb2	pten	tp53	atm	cdh1	chek2	nbn	nf1	...	ppp2cb_mut	smarcd1_mut	nras_mut	ndfip1_mut	hras_mut	prps2_mut	smarcb1_mut	stmn2_mut	sial
1	-1.1329	2.7485	-0.7175	0.9446	-0.2574	-1.2881	-0.9623	-0.4673	0.2962	-0.9181	...	0	0	0	0	0	0	0	0	
4	-0.2048	-1.2539	-1.7408	0.2214	-0.0545	0.9722	0.8895	0.9785	-0.3633	-1.1611	...	0	0	0	0	0	0	0	0	
15	-1.5371	-0.2021	-0.1617	-0.0149	2.7184	-0.4318	0.0497	-0.2304	0.5411	0.1981	...	0	0	0	0	0	0	0	0	
18	-0.2393	-2.4202	-0.2717	0.4880	0.3829	-2.2461	0.8632	-0.7474	0.8187	0.2631	...	0	0	0	0	0	0	0	0	
22	-0.5530	-0.4240	-1.2609	0.1169	-1.3937	1.5971	-0.2516	0.9410	-3.2153	-1.2803	...	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1394	2.0796	-0.7694	-1.1966	-0.1870	-1.5287	-0.7319	0.6596	-0.3314	0.0894	-1.1220	...	0	0	0	0	0	0	0	0	
1397	1.3367	-0.3731	-2.7215	-0.5771	0.7533	2.4777	-1.1931	-0.6719	0.1594	3.0227	...	0	0	0	0	0	0	0	0	
1399	-0.8468	0.4818	0.6131	0.0600	-0.6060	1.7714	0.8917	0.1010	-0.8821	-0.7944	...	0	0	0	0	0	0	0	0	
1402	-1.2448	-2.2417	0.5508	0.8508	-0.6159	1.5829	0.0555	1.3599	-0.5648	-0.5968	...	0	0	0	0	0	0	0	0	

	brca1	brca2	palb2	pten	tp53	atm	cdh1	chek2	nbn	nf1	...	ppp2cb_mut	smarcd1_mut	nras_mut	ndfip1_mut	hras_mut	prps2_mut	smarcb1_mut	stmn2_mut	sial
1403	-0.6506	-0.1476	-0.7637	1.1327	-0.3060	-1.4050	0.9773	-0.3467	-0.8001	-0.7799	...	0	0	0	0	0	0	0	0	

444 rows × 663 columns

In [105…
```
# Select target features which have significant difference (p < 0.05) between patients with better prognosis and patients with worse prognosis
target_col = []
for i in list(df.columns):
    t_test_res = stats.ttest_ind(outcome_0[i], outcome_1[i])
    if t_test_res.pvalue < 0.05:
        target_col.append(i)
```

In [106…
```
len(target_col)
```

Out[106… 255

- Contains 254 target features and a label (outcome).

In [20]:
```
# My final labeled data
df2= df[target_col].copy()
df2.shape
```

Out[20]: (1404, 255)

Normalize gene expression level features

In [21]:
```
# find out features (gene expression level) that need to be normalization
col_names_Gene = list(df2.columns).copy()
for i in col_names_mutGene:
    if i in col_names_Gene:
        col_names_Gene.remove(i)
col_names_Gene.remove('outcome')
print(len(col_names_Gene),' features that need to be normalization')
```

```
237  features that need to be normalization
```

In [22]:
```
# normalization function
def normalization_func(a):
    mean_ = np.mean(a)
    sd_ = np.std(a)
    return (a - mean_) / sd_

# normalize those features
for i in col_names_Gene:
    df2.loc[:,i] = normalization_func(df2.loc[:,i])
```

In [23]:
```
df2.describe()
```

Out[23]:

	pten	cdh1	chek2	nf1	bard1	mlh1	msh6	epcam	rbl1	ccna1	...	stab2_mut	rb1_mut	kdm6a_mu
count	1.404000e+03	1.404000e+03	1.404000e+03	1.404000e+03	1.404000e+03	1.404000e+03	1.404000e+03	1.404000e+03	1.404000e+03	1.404000e+03	...	1404.000000	1404.000000	1404.00000
mean	-1.296842e-17	3.036507e-17	2.530423e-17	-1.518254e-17	-2.530423e-18	2.530423e-17	-3.036507e-17	2.530423e-18	2.656944e-17	7.591269e-18	...	0.031339	0.027066	0.02065
std	1.000356e+00	1.000356e+00	1.000356e+00	1.000356e+00	1.000356e+00	1.000356e+00	1.000356e+00	1.000356e+00	1.000356e+00	1.000356e+00	...	0.174294	0.162332	0.14227
min	-5.871872e+00	-3.379128e+00	-2.748516e+00	-3.035143e+00	-3.013780e+00	-6.360105e+00	-4.028865e+00	-2.183465e+00	-2.796249e+00	-1.452642e+00	...	0.000000	0.000000	0.00000
25%	-5.596987e-01	-4.612209e-01	-7.257508e-01	-6.799140e-01	-6.976962e-01	-5.587806e-01	-6.660750e-01	-6.873950e-01	-6.794875e-01	-4.632231e-01	...	0.000000	0.000000	0.00000
50%	1.048188e-01	1.164585e-01	-1.294469e-01	-4.708103e-02	-2.682720e-02	8.149896e-02	-1.082935e-01	-1.524809e-01	-1.477494e-01	-2.042253e-01	...	0.000000	0.000000	0.00000

	pten	cdh1	chek2	nf1	bard1	mlh1	msh6	epcam	rbl1	ccna1	...	stab2_mut	rb1_mut	kdm6a_mu
75%	6.445440e-01	6.612613e-01	5.572084e-01	6.703705e-01	6.675197e-01	6.554098e-01	5.643787e-01	5.378281e-01	4.992277e-01	1.021235e-01	...	0.000000	0.000000	0.000000
max	2.619653e+00	2.650705e+00	4.049700e+00	3.917566e+00	3.377342e+00	3.258302e+00	5.357286e+00	6.472704e+00	5.087242e+00	1.067072e+01	...	1.000000	1.000000	1.000000

8 rows × 255 columns

1 - b. Propose a deep learning-based model development and evaluation plan.

1. After normalization for continuous features, I will spilt data to training and validation set (ratio: 8.5:1.5).
2. PyTorch framework will be used to train the model.
3. The numbers of neuron of 1st hidden layer to 12th hidden layer are 512-512-256-256-256-256-256-256-256-256-128-128.
4. The output layer has 2 neurons because the model is a binary classifier.
5. The activation of each hidden layer is ReLU, and I will use sofmax (class = 2) function in output layer and calculate probability.
6. Dropout regularization with probability = 0.3 will be applied to each hidden layer to prevent overfitting.
7. My optimizer will be Adam, and the loss function will be cross entropy.
8. The learning rate will start with 0.00001, and epoch value will start with 100.
9. The mini batch size will be 64.
10. All hyperparameters (number of layers, number of neurons in each hidden layer, dropout probability, learning rate, and the number of epoch) will be adjusted according to loss and the performance (AUC) in training and validation set.
11. Report precision, recall, accuracy, F1 score, and AUC in the validation set.

1 - c. Implement the feed-forward neural network model.

In [24]:
```
# split data into training set and validation set
dt_train, dt_val = train_test_split(df2, test_size=200, random_state=2, stratify=df2.outcome)

dt_train_x = dt_train.drop(['outcome'],axis=1)
dt_train_y = dt_train['outcome']
dt_val_x = dt_val.drop(['outcome'],axis=1)
dt_val_y = dt_val['outcome']
```

In [25]:
```
print(dt_train_x.shape)
print(dt_train_y.shape)
print(dt_val_x.shape)
print(dt_val_y.shape)
```

```
(1204, 254)
(1204,)
(200, 254)
(200,)
```

In [26]:
```
dt_train_x
```

Out[26]:

	pten	cdh1	chek2	nf1	bard1	mlh1	msh6	epcam	rbl1	ccna1	...	ncor1_mut	stab2_mut	rb1_mut	kdm6a_mut	pbrm1_mut	smad4_mut	prkcq_mut
737	2.175617	1.922209	-0.129498	1.357643	0.703429	0.035127	1.072567	0.708401	1.167852	-0.059458	...	0	0	1	0	0	0	0
509	-0.113778	0.879982	-0.203389	0.203265	-0.268199	0.638552	-0.795156	-0.536680	0.344665	0.245458	...	0	0	0	0	0	0	0
421	0.225645	-0.245056	0.933418	-0.394573	0.685009	-2.449957	1.105216	-0.567744	-0.702879	2.347500	...	0	0	0	0	0	0	0
296	-0.106242	-0.173381	-0.372881	1.084445	1.692470	-0.183680	0.447776	-1.397460	-1.312935	-0.230364	...	1	0	0	0	0	0	0
138	0.996808	0.872997	0.316501	0.953995	2.407627	-1.216017	1.595442	-0.356166	2.175643	0.937325	...	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

	pten	cdh1	chek2	nf1	bard1	mlh1	msh6	epcam	rbl1	ccna1	...	ncor1_mut	stab2_mut	rb1_mut	kdm6a_mut	pbrm1_mut	smad4_mut	prkcq_mut
504	1.543871	0.638635	-0.697086	-0.273345	-1.493378	1.145377	0.007069	-0.837301	0.116261	0.099485	...	0	0	0	0	0	0	0
595	0.606120	-0.396506	-0.881866	-0.206839	-0.423007	0.019011	-1.074308	-0.451584	-0.253252	-0.264746	...	0	0	0	0	0	0	0
75	0.940486	-2.805726	3.078308	2.244562	1.104641	-1.577696	-1.718152	-1.386128	2.702551	-0.235893	...	0	0	0	0	0	0	0
217	-0.061422	-0.997343	0.983358	0.676903	3.037630	1.288842	2.380499	-0.678541	2.566095	0.478192	...	0	0	1	0	0	0	0
336	-0.032567	1.171239	0.033471	0.334330	1.091858	-1.469627	-0.876034	1.040793	-0.295838	-0.881214	...	0	0	0	0	0	0	0

1204 rows × 254 columns

In [27]:
```
dt_train_y
```

Out[27]:
```
737    1
509    0
421    0
296    0
138    1
      ..
504    0
595    0
75     1
217    0
336    0
Name: outcome, Length: 1204, dtype: int64
```

Define the logger

- Reference: TA's Demo

In [28]:
```
class Logger:
    def __init__(self):
        self.log = {}
    def add(self, key, value):
        if key in self.log.keys():
            self._add_valid(key, value)
        else:
            self.log[key] = {}
            self.log[key]['history'] = []
            self.log[key]['last_epoch'] = 0 if key.split('_')[0] == 'train' else -1
            self._add_valid(key, value)

    def _add_valid(self, key, value):
        epoch = self.log[key]['last_epoch'] + 1
        self.log[key]['history'].append({'epoch':epoch, key:value})
        self.log[key]['last_epoch'] = epoch

    def get(self, key):
        if key is None:
            return self.log
        return self.log[key]['history']
```

Define the Model

- Reference: TA's Demo

In [43]:
```
class FlexFC(pl.LightningModule):
    def __init__(self):
        super().__init__()
```

```
        self.fc = nn.Sequential(
            nn.Linear(254, 512),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(256, 256),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(256, 256),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(256, 256),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(256, 256),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(256, 256),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(256, 256),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(256, 256),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.Linear(128,2))


        self.writer = Logger()
        self.train_auc = torchmetrics.AUROC(num_classes=2)
        self.validation_auc = torchmetrics.AUROC(num_classes=2)

    def forward(self, x):
        return self.fc(x)

    def training_step(self, batch, batch_idx):
        x, y = batch
        x = x.view(x.size(0), -1)
        h = self.fc(x)
        loss = F.cross_entropy(h, y.long()) # h: predictive truth; y = ground truth
        self.train_auc(h, y.int()) # h: output; y = ground truth
        self.log("train_loss", loss) # store loss
        return loss

    def training_epoch_end(self, training_step_outputs):
        res = [] # store loss terms
        for out in training_step_outputs:
            res.append(out['loss'].item())
        self.writer.add('train_auroc', self.train_auc.compute().item()) # calculate training AUROC
        self.writer.add('train_loss', sum(res)/len(res)) # average loss
        self.train_auc.reset() # reset, and during next iteration, it will start to collect the item that computed in the end of the epoch

    def validation_step(self, batch, batch_idx):
        x, y = batch
        x = x.view(x.size(0), -1)
```

```
        h = self.fc(x)
        loss = F.cross_entropy(h, y.long()) # h: predictive truth; y = ground truth
        self.validation_auc(h,y.int())
        self.log("val_loss", loss)
        return loss

    def validation_epoch_end(self, validation_step_outputs):
        res = []
        for out in validation_step_outputs:
            res.append(out.item())
        self.writer.add('val_auroc', self.validation_auc.compute().item())
        self.writer.add('val_loss', sum(res)/len(res))
        self.validation_auc.reset()

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=0.00001)
        return optimizer

# disable the validation progress bar
class LitProgressBar(TQDMProgressBar):
    def init_validation_tqdm(self):
        bar = tqdm(
            disable=True,
        )
        return bar


# define dataset loader
class PandasDataset(torch.utils.data.Dataset): # Can get dataset from the Pandas dataframe
    def __init__(self, X, y=None):
        super().__init__()
        self.X = X
        self.y = y

    def __len__(self):
        return self.X.shape[0]

    def __getitem__(self, idx):
        xi = torch.tensor(self.X.iloc[idx].to_numpy()).float()
        if self.y is None:
            return xi
        yi = torch.tensor(self.y.iloc[idx].item()).float()
        return xi, yi
```

- Reference: TA's Demo

In [60]:

```
# prepare the dataset
tt = PandasDataset(dt_train_x,dt_train_y)
train_loader = DataLoader(tt, batch_size=64, shuffle=True) # can try different batch size, num_workers affect the speed of loading dataset.

vv = PandasDataset(dt_val_x,dt_val_y)
val_loader = DataLoader(vv, batch_size=64, shuffle=False) # for val set, shuffle doesn't make difference

# initiate the model
model = FlexFC()

bar = LitProgressBar()
trainer = pl.Trainer(
    max_epochs=100,
    callbacks=[bar])
res = trainer.fit(
    model=model,
    train_dataloaders=train_loader,
    val_dataloaders=val_loader)
```

```
GPU available: False, used: False
```

```
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs

  | Name           | Type       | Params
-----------------------------------------------
0 | fc             | Sequential | 1.0 M
1 | train_auc      | AUROC      | 0
2 | validation_auc | AUROC      | 0
-----------------------------------------------
1.0 M     Trainable params
0         Non-trainable params
1.0 M     Total params
4.139     Total estimated model params size (MB)
```

In [61]:
```
history_dict = model.writer.log # look the logger
history_dict
```

Out[61]:
```
{'val_auroc': {'history': [{'epoch': 0, 'val_auroc': 0.4583693742752075},
   {'epoch': 1, 'val_auroc': 0.5165970921516418},
   {'epoch': 2, 'val_auroc': 0.5350770950317383},
   {'epoch': 3, 'val_auroc': 0.5583361983299255},
   {'epoch': 4, 'val_auroc': 0.563955545425415},
   {'epoch': 5, 'val_auroc': 0.577656090259552},
   {'epoch': 6, 'val_auroc': 0.5827250480651855},
   {'epoch': 7, 'val_auroc': 0.5899084210395813},
   {'epoch': 8, 'val_auroc': 0.6063607931137085},
   {'epoch': 9, 'val_auroc': 0.6091704368591309},
   {'epoch': 10, 'val_auroc': 0.6041305065155029},
   {'epoch': 11, 'val_auroc': 0.6105607748031616},
   {'epoch': 12, 'val_auroc': 0.6157456040382385},
   {'epoch': 13, 'val_auroc': 0.619916558265686},
   {'epoch': 14, 'val_auroc': 0.638164758682251},
   {'epoch': 15, 'val_auroc': 0.6517784595489502},
   {'epoch': 16, 'val_auroc': 0.6671880483627319},
   {'epoch': 17, 'val_auroc': 0.6897520422935486},
   {'epoch': 18, 'val_auroc': 0.7044374942779541},
   {'epoch': 19, 'val_auroc': 0.7176457643508911},
   {'epoch': 20, 'val_auroc': 0.7305063009262085},
   {'epoch': 21, 'val_auroc': 0.7401227951049805},
   {'epoch': 22, 'val_auroc': 0.7418607473373413},
   {'epoch': 23, 'val_auroc': 0.7508978843688965},
   {'epoch': 24, 'val_auroc': 0.7546054720878601},
   {'epoch': 25, 'val_auroc': 0.7564592361450195},
   {'epoch': 26, 'val_auroc': 0.7564592361450195},
   {'epoch': 27, 'val_auroc': 0.757502019405365},
   {'epoch': 28, 'val_auroc': 0.7592399716377258},
   {'epoch': 29, 'val_auroc': 0.7605143785476685},
   {'epoch': 30, 'val_auroc': 0.7614413499832153},
   {'epoch': 31, 'val_auroc': 0.7621943950653076},
   {'epoch': 32, 'val_auroc': 0.762715816497802 7},
   {'epoch': 33, 'val_auroc': 0.76329505443573},
   {'epoch': 34, 'val_auroc': 0.7638744115829468},
   {'epoch': 35, 'val_auroc': 0.7643378376960754},
   {'epoch': 36, 'val_auroc': 0.7645695209503174},
   {'epoch': 37, 'val_auroc': 0.764743447303772},
   {'epoch': 38, 'val_auroc': 0.7649171352386475},
   {'epoch': 39, 'val_auroc': 0.765728235244751},
   {'epoch': 40, 'val_auroc': 0.7664233446121216},
   {'epoch': 41, 'val_auroc': 0.7670027017593384},
   {'epoch': 42, 'val_auroc': 0.7672344446182251},
   {'epoch': 43, 'val_auroc': 0.7678136825561523},
   {'epoch': 44, 'val_auroc': 0.768335103988647 5},
   {'epoch': 45, 'val_auroc': 0.767697811126709},
   {'epoch': 46, 'val_auroc': 0.7679295539855957},
   {'epoch': 47, 'val_auroc': 0.7687405943870544},
   {'epoch': 48, 'val_auroc': 0.7696095705032349},
   {'epoch': 49, 'val_auroc': 0.7696674466133118},
   {'epoch': 50, 'val_auroc': 0.7705943584442139},
   {'epoch': 51, 'val_auroc': 0.7705943584442139},
```

```
      {'epoch': 52, 'val_auroc': 0.7718108892440796},
      {'epoch': 53, 'val_auroc': 0.7722164392471313},
      {'epoch': 54, 'val_auroc': 0.7732012271881104},
      {'epoch': 55, 'val_auroc': 0.7744178175926208},
      {'epoch': 56, 'val_auroc': 0.7745915651321411},
      {'epoch': 57, 'val_auroc': 0.7758080959320068},
      {'epoch': 58, 'val_auroc': 0.7759239673614502},
      {'epoch': 59, 'val_auroc': 0.7760398387908936},
      {'epoch': 60, 'val_auroc': 0.776329517364502},
      {'epoch': 61, 'val_auroc': 0.776329517364502},
      {'epoch': 62, 'val_auroc': 0.7770246863365173},
      {'epoch': 63, 'val_auroc': 0.7777777910232544},
      {'epoch': 64, 'val_auroc': 0.7781254053115845},
      {'epoch': 65, 'val_auroc': 0.7782412767410278},
      {'epoch': 66, 'val_auroc': 0.7778936624526978},
      {'epoch': 67, 'val_auroc': 0.7795736193656921},
      {'epoch': 68, 'val_auroc': 0.7802108526229858},
      {'epoch': 69, 'val_auroc': 0.7815432548522949},
      {'epoch': 70, 'val_auroc': 0.782296359539032},
      {'epoch': 71, 'val_auroc': 0.7828177213668823},
      {'epoch': 72, 'val_auroc': 0.7838025093078613},
      {'epoch': 73, 'val_auroc': 0.784034252166748},
      {'epoch': 74, 'val_auroc': 0.7856563329696655},
      {'epoch': 75, 'val_auroc': 0.7862356901168823},
      {'epoch': 76, 'val_auroc': 0.7870466709136963},
      {'epoch': 77, 'val_auroc': 0.7876260280609131},
      {'epoch': 78, 'val_auroc': 0.7880894541740417},
      {'epoch': 79, 'val_auroc': 0.7891322374343872},
      {'epoch': 80, 'val_auroc': 0.7894797921180725},
      {'epoch': 81, 'val_auroc': 0.7906384468078613},
      {'epoch': 82, 'val_auroc': 0.7911018133163452},
      {'epoch': 83, 'val_auroc': 0.792665958404541},
      {'epoch': 84, 'val_auroc': 0.7937666177749634},
      {'epoch': 85, 'val_auroc': 0.7951569557189941},
      {'epoch': 86, 'val_auroc': 0.7953887581825256},
      {'epoch': 87, 'val_auroc': 0.7955045700073242},
      {'epoch': 88, 'val_auroc': 0.7961996793746948},
      {'epoch': 89, 'val_auroc': 0.7959680557250977},
      {'epoch': 90, 'val_auroc': 0.7966631650924683},
      {'epoch': 91, 'val_auroc': 0.7970107197761536},
      {'epoch': 92, 'val_auroc': 0.7974741458892822},
      {'epoch': 93, 'val_auroc': 0.7966631650924683},
      {'epoch': 94, 'val_auroc': 0.7972425222396851},
      {'epoch': 95, 'val_auroc': 0.7975900769233704},
      {'epoch': 96, 'val_auroc': 0.7984011173248291},
      {'epoch': 97, 'val_auroc': 0.7986328601837158},
      {'epoch': 98, 'val_auroc': 0.798632800579071},
      {'epoch': 99, 'val_auroc': 0.7990962862968445},
      {'epoch': 100, 'val_auroc': 0.7992122173309326}],
     'last_epoch': 100},
    'val_loss': {'history': [{'epoch': 0, 'val_loss': 0.7171383500099182},
      {'epoch': 1, 'val_loss': 0.7192869633436203},
      {'epoch': 2, 'val_loss': 0.717837780714035},
      {'epoch': 3, 'val_loss': 0.716381162405014},
      {'epoch': 4, 'val_loss': 0.714884340763092},
      {'epoch': 5, 'val_loss': 0.7133704572916031},
      {'epoch': 6, 'val_loss': 0.7118236124515533},
      {'epoch': 7, 'val_loss': 0.7102627009153366},
      {'epoch': 8, 'val_loss': 0.708602711558342},
      {'epoch': 9, 'val_loss': 0.7069361954927444},
      {'epoch': 10, 'val_loss': 0.7051064372062683},
      {'epoch': 11, 'val_loss': 0.7032985091209412},
      {'epoch': 12, 'val_loss': 0.7013057321310043},
      {'epoch': 13, 'val_loss': 0.699131041765213},
      {'epoch': 14, 'val_loss': 0.6968301981687546},
      {'epoch': 15, 'val_loss': 0.6943344473838806},
      {'epoch': 16, 'val_loss': 0.6914356201887131},
      {'epoch': 17, 'val_loss': 0.6879212856292725},
      {'epoch': 18, 'val_loss': 0.6826846599578857},
      {'epoch': 19, 'val_loss': 0.6736422926187515},
      {'epoch': 20, 'val_loss': 0.6539365500211716},
      {'epoch': 21, 'val_loss': 0.6106829047203064},
```

```
{'epoch': 22, 'val_loss': 0.5584122017025948},
{'epoch': 23, 'val_loss': 0.5373690128326416},
{'epoch': 24, 'val_loss': 0.5316155180335045},
{'epoch': 25, 'val_loss': 0.5266060829162598},
{'epoch': 26, 'val_loss': 0.5243064016103745},
{'epoch': 27, 'val_loss': 0.5183638781309128},
{'epoch': 28, 'val_loss': 0.5191079452633858},
{'epoch': 29, 'val_loss': 0.5150239989161491},
{'epoch': 30, 'val_loss': 0.5098516717553139},
{'epoch': 31, 'val_loss': 0.5105930268764496},
{'epoch': 32, 'val_loss': 0.5079408958554268},
{'epoch': 33, 'val_loss': 0.5060463920235634},
{'epoch': 34, 'val_loss': 0.5026647448539734},
{'epoch': 35, 'val_loss': 0.5013997107744217},
{'epoch': 36, 'val_loss': 0.5008810237050056},
{'epoch': 37, 'val_loss': 0.4990397319197655},
{'epoch': 38, 'val_loss': 0.4968389421701431},
{'epoch': 39, 'val_loss': 0.4966994449496269},
{'epoch': 40, 'val_loss': 0.4943782314658165},
{'epoch': 41, 'val_loss': 0.4928102046251297},
{'epoch': 42, 'val_loss': 0.4911191388964653},
{'epoch': 43, 'val_loss': 0.4923556074500084},
{'epoch': 44, 'val_loss': 0.4887354373931885},
{'epoch': 45, 'val_loss': 0.4886837527155876},
{'epoch': 46, 'val_loss': 0.4859173893928528},
{'epoch': 47, 'val_loss': 0.4854200929403305},
{'epoch': 48, 'val_loss': 0.48643743991851807},
{'epoch': 49, 'val_loss': 0.4845617711544037},
{'epoch': 50, 'val_loss': 0.48412732779979706},
{'epoch': 51, 'val_loss': 0.48369279503822327},
{'epoch': 52, 'val_loss': 0.48312395066022873},
{'epoch': 53, 'val_loss': 0.4823068901896477},
{'epoch': 54, 'val_loss': 0.4818562865257263},
{'epoch': 55, 'val_loss': 0.480481892824173},
{'epoch': 56, 'val_loss': 0.4791101664304733},
{'epoch': 57, 'val_loss': 0.47769153863191605},
{'epoch': 58, 'val_loss': 0.4762540906677094},
{'epoch': 59, 'val_loss': 0.4773022457957268},
{'epoch': 60, 'val_loss': 0.47665128111839294},
{'epoch': 61, 'val_loss': 0.4770662561058998},
{'epoch': 62, 'val_loss': 0.4748305678367615},
{'epoch': 63, 'val_loss': 0.4743908792734146},
{'epoch': 64, 'val_loss': 0.4722949266433716},
{'epoch': 65, 'val_loss': 0.47262994945049286},
{'epoch': 66, 'val_loss': 0.4706062898039818},
{'epoch': 67, 'val_loss': 0.47029148042201996},
{'epoch': 68, 'val_loss': 0.4699534848332405},
{'epoch': 69, 'val_loss': 0.4694483131170273},
{'epoch': 70, 'val_loss': 0.46755217015743256},
{'epoch': 71, 'val_loss': 0.4664497897028923},
{'epoch': 72, 'val_loss': 0.46535879373550415},
{'epoch': 73, 'val_loss': 0.4651448652148247},
{'epoch': 74, 'val_loss': 0.4649663493037224},
{'epoch': 75, 'val_loss': 0.4631608834028244},
{'epoch': 76, 'val_loss': 0.46265705674886703},
{'epoch': 77, 'val_loss': 0.46288271248340607},
{'epoch': 78, 'val_loss': 0.4625531733036041},
{'epoch': 79, 'val_loss': 0.4627254232764244},
{'epoch': 80, 'val_loss': 0.46086176484823227},
{'epoch': 81, 'val_loss': 0.4583745747804642},
{'epoch': 82, 'val_loss': 0.4569164849817753},
{'epoch': 83, 'val_loss': 0.4569820649921894},
{'epoch': 84, 'val_loss': 0.45597124472260475},
{'epoch': 85, 'val_loss': 0.4550393782556057},
{'epoch': 86, 'val_loss': 0.4549345150589943},
{'epoch': 87, 'val_loss': 0.4543871507048607},
{'epoch': 88, 'val_loss': 0.45333389192819595},
{'epoch': 89, 'val_loss': 0.45218679681420326},
{'epoch': 90, 'val_loss': 0.4520377330482006},
{'epoch': 91, 'val_loss': 0.45055903121829033},
{'epoch': 92, 'val_loss': 0.4494960233569145},
{'epoch': 93, 'val_loss': 0.4489385448396206},
```

```
    {'epoch': 94, 'val_loss': 0.4479869343340397},
    {'epoch': 95, 'val_loss': 0.4480029568076134},
    {'epoch': 96, 'val_loss': 0.44726579636335373},
    {'epoch': 97, 'val_loss': 0.44644054025411606},
    {'epoch': 98, 'val_loss': 0.4467933960258961},
    {'epoch': 99, 'val_loss': 0.4461780674755573},
    {'epoch': 100, 'val_loss': 0.4458782337605953}],
   'last_epoch': 100},
  'train_auroc': {'history': [{'epoch': 1, 'train_auroc': 0.48928603529930115},
    {'epoch': 2, 'train_auroc': 0.49431854486465454},
    {'epoch': 3, 'train_auroc': 0.485060453414917},
    {'epoch': 4, 'train_auroc': 0.5114841461181641},
    {'epoch': 5, 'train_auroc': 0.49843889474868774},
    {'epoch': 6, 'train_auroc': 0.4955272078514099},
    {'epoch': 7, 'train_auroc': 0.5037631988525391},
    {'epoch': 8, 'train_auroc': 0.49670642614364624},
    {'epoch': 9, 'train_auroc': 0.5248674154281616},
    {'epoch': 10, 'train_auroc': 0.49105122685432434},
    {'epoch': 11, 'train_auroc': 0.5097747445106506},
    {'epoch': 12, 'train_auroc': 0.5035766363143921},
    {'epoch': 13, 'train_auroc': 0.5070974826812744},
    {'epoch': 14, 'train_auroc': 0.4889160990715027},
    {'epoch': 15, 'train_auroc': 0.5123420357704163},
    {'epoch': 16, 'train_auroc': 0.5159760117530823},
    {'epoch': 17, 'train_auroc': 0.47881126403808594},
    {'epoch': 18, 'train_auroc': 0.5038747787475586},
    {'epoch': 19, 'train_auroc': 0.4968554973602295},
    {'epoch': 20, 'train_auroc': 0.5376001000404358},
    {'epoch': 21, 'train_auroc': 0.5755701065063477},
    {'epoch': 22, 'train_auroc': 0.625816822052002},
    {'epoch': 23, 'train_auroc': 0.6374046206474304},
    {'epoch': 24, 'train_auroc': 0.6592279672622681},
    {'epoch': 25, 'train_auroc': 0.6828022003173828},
    {'epoch': 26, 'train_auroc': 0.6771574020385742},
    {'epoch': 27, 'train_auroc': 0.6812984943389893},
    {'epoch': 28, 'train_auroc': 0.6763967871665955},
    {'epoch': 29, 'train_auroc': 0.6887021064758301},
    {'epoch': 30, 'train_auroc': 0.6966717839241028},
    {'epoch': 31, 'train_auroc': 0.7086454629898071},
    {'epoch': 32, 'train_auroc': 0.7205792665481567},
    {'epoch': 33, 'train_auroc': 0.7031059265136719},
    {'epoch': 34, 'train_auroc': 0.7200307846069336},
    {'epoch': 35, 'train_auroc': 0.70375657081604},
    {'epoch': 36, 'train_auroc': 0.7174539566040039},
    {'epoch': 37, 'train_auroc': 0.7218947410583496},
    {'epoch': 38, 'train_auroc': 0.7206637859344482},
    {'epoch': 39, 'train_auroc': 0.7232087254524231},
    {'epoch': 40, 'train_auroc': 0.7282077074050903},
    {'epoch': 41, 'train_auroc': 0.7348300218582153},
    {'epoch': 42, 'train_auroc': 0.7332050800323486},
    {'epoch': 43, 'train_auroc': 0.7255814671516418},
    {'epoch': 44, 'train_auroc': 0.7342208623886108},
    {'epoch': 45, 'train_auroc': 0.7441072463989258},
    {'epoch': 46, 'train_auroc': 0.7358026504516602},
    {'epoch': 47, 'train_auroc': 0.7358250617980957},
    {'epoch': 48, 'train_auroc': 0.7407458424568176},
    {'epoch': 49, 'train_auroc': 0.7386418581008911},
    {'epoch': 50, 'train_auroc': 0.7403122186660767},
    {'epoch': 51, 'train_auroc': 0.7463778853416443},
    {'epoch': 52, 'train_auroc': 0.7445217967033386},
    {'epoch': 53, 'train_auroc': 0.7416771054267883},
    {'epoch': 54, 'train_auroc': 0.7401270866394043},
    {'epoch': 55, 'train_auroc': 0.7518887519836426},
    {'epoch': 56, 'train_auroc': 0.7534099817276001},
    {'epoch': 57, 'train_auroc': 0.7563774585723877},
    {'epoch': 58, 'train_auroc': 0.7563583850860596},
    {'epoch': 59, 'train_auroc': 0.7613924741744995},
    {'epoch': 60, 'train_auroc': 0.7574570178985596},
    {'epoch': 61, 'train_auroc': 0.7540749311447144},
    {'epoch': 62, 'train_auroc': 0.7604819536209106},
    {'epoch': 63, 'train_auroc': 0.7601630687713623},
    {'epoch': 64, 'train_auroc': 0.7707781195640564},
```

```
    {'epoch': 65, 'train_auroc': 0.7668203711509705},
    {'epoch': 66, 'train_auroc': 0.7710875272750854},
    {'epoch': 67, 'train_auroc': 0.762575626373291},
    {'epoch': 68, 'train_auroc': 0.762566089630127},
    {'epoch': 69, 'train_auroc': 0.7724332809448242},
    {'epoch': 70, 'train_auroc': 0.7811524271965027},
    {'epoch': 71, 'train_auroc': 0.7801956534385681},
    {'epoch': 72, 'train_auroc': 0.7784479856491089},
    {'epoch': 73, 'train_auroc': 0.7677102088928223},
    {'epoch': 74, 'train_auroc': 0.7833800911903381},
    {'epoch': 75, 'train_auroc': 0.7764612436294556},
    {'epoch': 76, 'train_auroc': 0.7899401187896729},
    {'epoch': 77, 'train_auroc': 0.7780318260192871},
    {'epoch': 78, 'train_auroc': 0.7770767211914062},
    {'epoch': 79, 'train_auroc': 0.7864894866943359},
    {'epoch': 80, 'train_auroc': 0.795838475227356},
    {'epoch': 81, 'train_auroc': 0.791689395904541},
    {'epoch': 82, 'train_auroc': 0.8018213510513306},
    {'epoch': 83, 'train_auroc': 0.7953394055366516},
    {'epoch': 84, 'train_auroc': 0.7995697259902954},
    {'epoch': 85, 'train_auroc': 0.8024272918701172},
    {'epoch': 86, 'train_auroc': 0.7962195873260498},
    {'epoch': 87, 'train_auroc': 0.8099903464317322},
    {'epoch': 88, 'train_auroc': 0.8032436370849609},
    {'epoch': 89, 'train_auroc': 0.817743182182312},
    {'epoch': 90, 'train_auroc': 0.8162714242935181},
    {'epoch': 91, 'train_auroc': 0.8158010244369507},
    {'epoch': 92, 'train_auroc': 0.8255246877670288},
    {'epoch': 93, 'train_auroc': 0.8247944116592407},
    {'epoch': 94, 'train_auroc': 0.8259855508804321},
    {'epoch': 95, 'train_auroc': 0.8261163234710693},
    {'epoch': 96, 'train_auroc': 0.8232827186584473},
    {'epoch': 97, 'train_auroc': 0.8272882699966431},
    {'epoch': 98, 'train_auroc': 0.8358559608459473},
    {'epoch': 99, 'train_auroc': 0.8315331339836121},
    {'epoch': 100, 'train_auroc': 0.8334752321243286}],
  'last_epoch': 100},
 'train_loss': {'history': [{'epoch': 1, 'train_loss': 0.7152867819133558},
    {'epoch': 2, 'train_loss': 0.71353238507321},
    {'epoch': 3, 'train_loss': 0.7128389038537678},
    {'epoch': 4, 'train_loss': 0.7111240468527141},
    {'epoch': 5, 'train_loss': 0.710525045269414},
    {'epoch': 6, 'train_loss': 0.7088797688484192},
    {'epoch': 7, 'train_loss': 0.7080360964724892},
    {'epoch': 8, 'train_loss': 0.7066190085913006},
    {'epoch': 9, 'train_loss': 0.7045974480478387},
    {'epoch': 10, 'train_loss': 0.7039455112658048},
    {'epoch': 11, 'train_loss': 0.7018181116957414},
    {'epoch': 12, 'train_loss': 0.700243542068883},
    {'epoch': 13, 'train_loss': 0.6986094838694522},
    {'epoch': 14, 'train_loss': 0.6974674620126423},
    {'epoch': 15, 'train_loss': 0.6951992009815416},
    {'epoch': 16, 'train_loss': 0.6928676460918627},
    {'epoch': 17, 'train_loss': 0.6914334673630563},
    {'epoch': 18, 'train_loss': 0.6873999363497684},
    {'epoch': 19, 'train_loss': 0.6824462664754767},
    {'epoch': 20, 'train_loss': 0.6704631447792053},
    {'epoch': 21, 'train_loss': 0.6461585101328398},
    {'epoch': 22, 'train_loss': 0.610885058578692},
    {'epoch': 23, 'train_loss': 0.5972489055834318},
    {'epoch': 24, 'train_loss': 0.5939522322855497},
    {'epoch': 25, 'train_loss': 0.5839510058101854},
    {'epoch': 26, 'train_loss': 0.5816341575823332},
    {'epoch': 27, 'train_loss': 0.583784166135286},
    {'epoch': 28, 'train_loss': 0.5845576289453005},
    {'epoch': 29, 'train_loss': 0.578747526595467},
    {'epoch': 30, 'train_loss': 0.5749979238761099},
    {'epoch': 31, 'train_loss': 0.5740945731338701},
    {'epoch': 32, 'train_loss': 0.5667779053512373},
    {'epoch': 33, 'train_loss': 0.5741828350644362},
    {'epoch': 34, 'train_loss': 0.5642629704977337},
    {'epoch': 35, 'train_loss': 0.5721232028383958},
```

```
 {'epoch': 36, 'train_loss': 0.5648054025675121},
 {'epoch': 37, 'train_loss': 0.5655530189212999},
 {'epoch': 38, 'train_loss': 0.5682565475765028},
 {'epoch': 39, 'train_loss': 0.5628894473377027},
 {'epoch': 40, 'train_loss': 0.5611890695596996},
 {'epoch': 41, 'train_loss': 0.5553736184772692},
 {'epoch': 42, 'train_loss': 0.5598209265031313},
 {'epoch': 43, 'train_loss': 0.5575100826589685},
 {'epoch': 44, 'train_loss': 0.5565778973855471},
 {'epoch': 45, 'train_loss': 0.5502275542209023},
 {'epoch': 46, 'train_loss': 0.5583960037482413},
 {'epoch': 47, 'train_loss': 0.5558561619959379},
 {'epoch': 48, 'train_loss': 0.5517204811698512},
 {'epoch': 49, 'train_loss': 0.5552825174833599},
 {'epoch': 50, 'train_loss': 0.5552572200172826},
 {'epoch': 51, 'train_loss': 0.5507061967724248},
 {'epoch': 52, 'train_loss': 0.5513375103473663},
 {'epoch': 53, 'train_loss': 0.5507910141819402},
 {'epoch': 54, 'train_loss': 0.5514770275668094},
 {'epoch': 55, 'train_loss': 0.5443169349118283},
 {'epoch': 56, 'train_loss': 0.5433380744959179},
 {'epoch': 57, 'train_loss': 0.5423819265867534},
 {'epoch': 58, 'train_loss': 0.5475351135981711},
 {'epoch': 59, 'train_loss': 0.5406884519677413},
 {'epoch': 60, 'train_loss': 0.5467219211553273},
 {'epoch': 61, 'train_loss': 0.546605530538057},
 {'epoch': 62, 'train_loss': 0.5427114163574419},
 {'epoch': 63, 'train_loss': 0.5384384754456972},
 {'epoch': 64, 'train_loss': 0.5304696591276872},
 {'epoch': 65, 'train_loss': 0.5368004754969948},
 {'epoch': 66, 'train_loss': 0.5317800343036652},
 {'epoch': 67, 'train_loss': 0.540391638090736},
 {'epoch': 68, 'train_loss': 0.5423515815483896},
 {'epoch': 69, 'train_loss': 0.5335650569514224},
 {'epoch': 70, 'train_loss': 0.5281029233807012},
 {'epoch': 71, 'train_loss': 0.5253143388974039},
 {'epoch': 72, 'train_loss': 0.5262405699805209},
 {'epoch': 73, 'train_loss': 0.5387196666315982},
 {'epoch': 74, 'train_loss': 0.5266578275906412},
 {'epoch': 75, 'train_loss': 0.5304329693317413},
 {'epoch': 76, 'train_loss': 0.5224685182696894},
 {'epoch': 77, 'train_loss': 0.530978168311872},
 {'epoch': 78, 'train_loss': 0.5292096734046936},
 {'epoch': 79, 'train_loss': 0.5269457007709303},
 {'epoch': 80, 'train_loss': 0.5214251232774634},
 {'epoch': 81, 'train_loss': 0.5201559553020879},
 {'epoch': 82, 'train_loss': 0.5146008792676424},
 {'epoch': 83, 'train_loss': 0.5182368347519323},
 {'epoch': 84, 'train_loss': 0.5134200249847612},
 {'epoch': 85, 'train_loss': 0.5124346409973345},
 {'epoch': 86, 'train_loss': 0.5186750386890612},
 {'epoch': 87, 'train_loss': 0.5068042827279944},
 {'epoch': 88, 'train_loss': 0.5130710256727118},
 {'epoch': 89, 'train_loss': 0.5002059685556512},
 {'epoch': 90, 'train_loss': 0.5021291657498008},
 {'epoch': 91, 'train_loss': 0.5013716503193504},
 {'epoch': 92, 'train_loss': 0.4912964764394258},
 {'epoch': 93, 'train_loss': 0.495334068411275},
 {'epoch': 94, 'train_loss': 0.4932216813689784},
 {'epoch': 95, 'train_loss': 0.4905162657562055},
 {'epoch': 96, 'train_loss': 0.4928084878545058},
 {'epoch': 97, 'train_loss': 0.48839133664181356},
 {'epoch': 98, 'train_loss': 0.48452161644634445},
 {'epoch': 99, 'train_loss': 0.4830731137802726},
 {'epoch': 100, 'train_loss': 0.4827893109698045}],
'last_epoch': 100}}
```


- Reference: TA's Demo

In [73]:
```
# convert it to pandas DF
dtlog = pd.concat([pd.DataFrame(x['history']).set_index('epoch') for x in history_dict.values()], axis=1)
```

```
dtlog
```

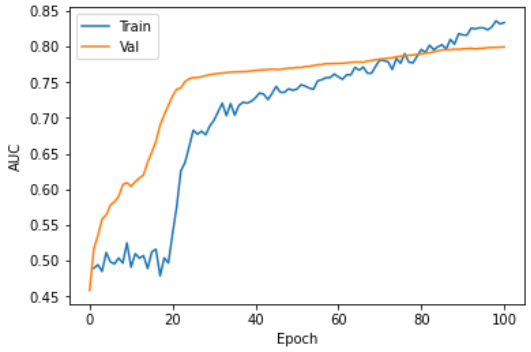
Out[73]:

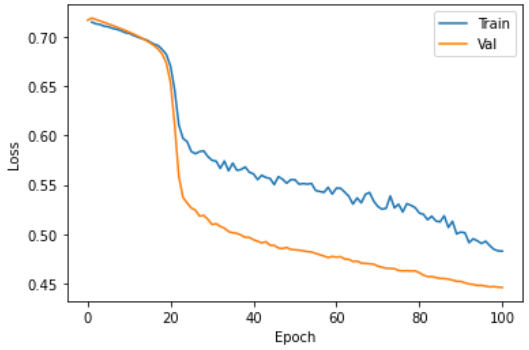
epoch	val_auroc	val_loss	train_auroc	train_loss
0	0.458369	0.717138	NaN	NaN
1	0.516597	0.719287	0.489286	0.715287
2	0.535077	0.717838	0.494319	0.713532
3	0.558336	0.716381	0.485060	0.712839
4	0.563956	0.714884	0.511484	0.711124
...	...	...	...	...
96	0.798401	0.447266	0.823283	0.492808
97	0.798633	0.446441	0.827288	0.488391
98	0.798633	0.446793	0.835856	0.484522
99	0.799096	0.446178	0.831533	0.483073
100	0.799212	0.445878	0.833475	0.482789

101 rows × 4 columns

In [74]:

```
# plot the AUC
sns.lineplot(x=dtlog.index, y=dtlog.train_auroc, label='Train')
sns.lineplot(x=dtlog.index, y=dtlog.val_auroc, label='Val')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('AUC')
plt.show()

# plot the loss curve
sns.lineplot(x=dtlog.index, y=dtlog.train_loss, label='Train')
sns.lineplot(x=dtlog.index, y=dtlog.val_loss, label='Val')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```





1 - d. Report labeled Confusion Matrix and the evaluation metrics in the validation dataset, including: precision, recall, accuracy, F1 score, and AUC.

Make predictions

- Reference: TA's Demo

In [75]:
```
vv = PandasDataset(dt_val_x) # use different dataloader because if we just want to make prediction on the un-seen data, unlabeled dataset, you do not have outcome and onl
val_loader_pred = DataLoader(vv, batch_size=64, shuffle=False)

val_pred = torch.cat(trainer.predict(model, dataloaders=val_loader_pred)) # Combine the predictions (unnormalized output) from dfferent batches
val_pred.shape
```

```
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/pytorch_lightning/trainer/connectors/data_connector.py:245: PossibleUserWarning: The dataloader, predic
t_dataloader 0, does not have many workers which may be a bottleneck. Consider increasing the value of the `num_workers` argument` (try 4 which is the number of cpus on t
his machine) in the `DataLoader` init to improve performance.
  category=PossibleUserWarning,
```

Out[75]:
```
torch.Size([200, 2])
```

In [76]:
```
# Convert to normalized predictions
val_pred_prob = torch.softmax(val_pred, dim = 1) # predictive probabilit for each class
val_pred_prob
```

Out[76]:
```
tensor([[0.9289, 0.0711],
        [0.7303, 0.2697],
        [0.9088, 0.0912],
        [0.9001, 0.0999],
        [0.8526, 0.1474],
        [0.4958, 0.5042],
        [0.5758, 0.4242],
        [0.9668, 0.0332],
        [0.4969, 0.5031],
        [0.8464, 0.1536],
        [0.6889, 0.3111],
        [0.8849, 0.1151],
        [0.9543, 0.0457],
        [0.9510, 0.0490],
        [0.9732, 0.0268],
        [0.8565, 0.1435],
        [0.6004, 0.3996],
        [0.9716, 0.0284],
        [0.8824, 0.1176],
        [0.4966, 0.5034],
        [0.5259, 0.4741],
        [0.4957, 0.5043],
        [0.4957, 0.5043],
        [0.5570, 0.4430],
```

```
       [0.4958, 0.5042],
       [0.9508, 0.0492],
       [0.7221, 0.2779],
       [0.9509, 0.0491],
       [0.6821, 0.3179],
       [0.9712, 0.0288],
       [0.9203, 0.0797],
       [0.8000, 0.2000],
       [0.5065, 0.4935],
       [0.9602, 0.0398],
       [0.9815, 0.0185],
       [0.7574, 0.2426],
       [0.8846, 0.1154],
       [0.4971, 0.5029],
       [0.8711, 0.1289],
       [0.9461, 0.0539],
       [0.5134, 0.4866],
       [0.7878, 0.2122],
       [0.6491, 0.3509],
       [0.4957, 0.5043],
       [0.5231, 0.4769],
       [0.7576, 0.2424],
       [0.4992, 0.5008],
       [0.8626, 0.1374],
       [0.9681, 0.0319],
       [0.7844, 0.2156],
       [0.9153, 0.0847],
       [0.5127, 0.4873],
       [0.8638, 0.1362],
       [0.9728, 0.0272],
       [0.9155, 0.0845],
       [0.8152, 0.1848],
       [0.6479, 0.3521],
       [0.8008, 0.1992],
       [0.8391, 0.1609],
       [0.9531, 0.0469],
       [0.5767, 0.4233],
       [0.9507, 0.0493],
       [0.9247, 0.0753],
       [0.9116, 0.0884],
       [0.6439, 0.3561],
       [0.4958, 0.5042],
       [0.8556, 0.1444],
       [0.9786, 0.0214],
       [0.5426, 0.4574],
       [0.5812, 0.4188],
       [0.4962, 0.5038],
       [0.8850, 0.1150],
       [0.4956, 0.5044],
       [0.4985, 0.5015],
       [0.9710, 0.0290],
       [0.6467, 0.3533],
       [0.8542, 0.1458],
       [0.6391, 0.3609],
       [0.9088, 0.0912],
       [0.8177, 0.1823],
       [0.8071, 0.1929],
       [0.6795, 0.3205],
       [0.9253, 0.0747],
       [0.6803, 0.3197],
       [0.9362, 0.0638],
       [0.8693, 0.1307],
       [0.7871, 0.2129],
       [0.7683, 0.2317],
       [0.8341, 0.1659],
       [0.8420, 0.1580],
       [0.9088, 0.0912],
       [0.9330, 0.0670],
       [0.7380, 0.2620],
       [0.9236, 0.0764],
       [0.8658, 0.1342],
       [0.8900, 0.1100],
```

```
       [0.9768, 0.0232],
       [0.4960, 0.5040],
       [0.8058, 0.1942],
       [0.5182, 0.4818],
       [0.5351, 0.4649],
       [0.4960, 0.5040],
       [0.4958, 0.5042],
       [0.6827, 0.3173],
       [0.5561, 0.4439],
       [0.6048, 0.3952],
       [0.8687, 0.1313],
       [0.4978, 0.5022],
       [0.6916, 0.3084],
       [0.9551, 0.0449],
       [0.8450, 0.1550],
       [0.5098, 0.4902],
       [0.4968, 0.5032],
       [0.8468, 0.1532],
       [0.9657, 0.0343],
       [0.9696, 0.0304],
       [0.4976, 0.5024],
       [0.9754, 0.0246],
       [0.7309, 0.2691],
       [0.5288, 0.4712],
       [0.9650, 0.0350],
       [0.4958, 0.5042],
       [0.9027, 0.0973],
       [0.4958, 0.5042],
       [0.9606, 0.0394],
       [0.4963, 0.5037],
       [0.7419, 0.2581],
       [0.6837, 0.3163],
       [0.8974, 0.1026],
       [0.5706, 0.4294],
       [0.7414, 0.2586],
       [0.9401, 0.0599],
       [0.4998, 0.5002],
       [0.8104, 0.1896],
       [0.9481, 0.0519],
       [0.5647, 0.4353],
       [0.5129, 0.4871],
       [0.6034, 0.3966],
       [0.8747, 0.1253],
       [0.7381, 0.2619],
       [0.7028, 0.2972],
       [0.4958, 0.5042],
       [0.4965, 0.5035],
       [0.5919, 0.4081],
       [0.7459, 0.2541],
       [0.8103, 0.1897],
       [0.9617, 0.0383],
       [0.7374, 0.2626],
       [0.6570, 0.3430],
       [0.9710, 0.0290],
       [0.5697, 0.4303],
       [0.7645, 0.2355],
       [0.7645, 0.2355],
       [0.5992, 0.4008],
       [0.6288, 0.3712],
       [0.7861, 0.2139],
       [0.8486, 0.1514],
       [0.8972, 0.1028],
       [0.4961, 0.5039],
       [0.9614, 0.0386],
       [0.7659, 0.2341],
       [0.8883, 0.1117],
       [0.9905, 0.0095],
       [0.8596, 0.1404],
       [0.4959, 0.5041],
       [0.4963, 0.5037],
       [0.8567, 0.1433],
       [0.9215, 0.0785],
```

```
        [0.8688, 0.1312],
        [0.6751, 0.3249],
        [0.8201, 0.1799],
        [0.8647, 0.1353],
        [0.6844, 0.3156],
        [0.9257, 0.0743],
        [0.8237, 0.1763],
        [0.4956, 0.5044],
        [0.4957, 0.5043],
        [0.4963, 0.5037],
        [0.6685, 0.3315],
        [0.9127, 0.0873],
        [0.7314, 0.2686],
        [0.9462, 0.0538],
        [0.6393, 0.3607],
        [0.8321, 0.1679],
        [0.9628, 0.0372],
        [0.7688, 0.2312],
        [0.4961, 0.5039],
        [0.9122, 0.0878],
        [0.8619, 0.1381],
        [0.5108, 0.4892],
        [0.9006, 0.0994],
        [0.9243, 0.0757],
        [0.8801, 0.1199],
        [0.4957, 0.5043],
        [0.8021, 0.1979],
        [0.6986, 0.3014],
        [0.9082, 0.0918],
        [0.9811, 0.0189],
        [0.9259, 0.0741],
        [0.8944, 0.1056]])
```

In [77]:
```
val_pred_class = torch.softmax(val_pred, dim = 1).argmax(1)
val_pred_class
```

Out[77]:
```
tensor([0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0,
        1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
        1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
        0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
        0, 1, 0, 0, 0, 0, 0, 0])
```

In [82]:
```
# Load Module
from sklearn.metrics import f1_score, accuracy_score, recall_score, confusion_matrix, roc_auc_score
from sklearn.metrics import plot_confusion_matrix
import matplotlib.pyplot as plt
```

Report metrics

In [94]:
```
precision_val = tp / (tp + fp)
recall_val = recall_score(dt_val_y, val_pred_class.numpy())
accuracy_val = accuracy_score(dt_val_y, val_pred_class.numpy())
F1_val = f1_score(dt_val_y, val_pred_class.numpy())
AUC_val = roc_auc_score(dt_val_y, val_pred_prob.numpy()[:, 1])
ConF_MaX = confusion_matrix(dt_val_y, val_pred_class.numpy())
ConF_MaX_plot = pd.DataFrame(ConF_MaX,
                   index=['True: better prognosis', 'True: worse prognosis'],
                   columns=['Pred: better prognosis', 'Pred: worse prognosis'])

print('Precision: ',precision_val)
print('Recall: ',recall_val)
print('Accuracy: ',accuracy_val)
print('F1 score: ',F1_val)
print('AUC: ',AUC_val)
```

```
print ('Confusion Matrix: ')
print(ConF_MaX_plot)
```

```
Precision:  0.696969696969697
Recall:  0.36507936507936506
Accuracy:  0.75
F1 score:  0.4791666666666666
AUC:  0.7992121422778358
Confusion Matrix:
                       Pred: better prognosis  Pred: worse prognosis
True: better prognosis                     127                     10
True: worse prognosis                       40                     23
```

1 - e. Apply your trained model to the Unlabeled Data

In [129…
```
df_test = pd.read_csv('BRCA-prognosis-test.csv')
df_test
```

Out[129…

	brca1	brca2	palb2	pten	tp53	atm	cdh1	chek2	nbn	nf1	...	mtap_mut	ppp2cb_mut	smarcd1_mut	nras_mut	ndfip1_mut	hras_mut	prps2_mut	smarcb1_mut	stmn
0	-0.6113	-0.4112	-0.5326	1.2881	-1.3982	-0.2587	0.3170	-0.1585	-0.9302	-0.2004	...	0	0	0	G12S	0	0	0	0	
1	-0.7632	0.6394	1.0946	-0.7267	-1.1607	1.2332	-0.6189	0.3389	3.3672	1.7262	...	0	0	0	0	0	0	0	0	
2	-0.4674	-0.4817	1.0418	0.6514	0.5839	-0.1902	-0.1550	-1.1869	-0.6927	1.4632	...	0	0	0	0	0	0	0	0	
3	1.2946	0.7721	-0.9238	0.3801	-0.3273	-1.1697	0.7206	1.8837	-1.4707	-0.6701	...	0	0	0	0	0	0	0	0	
4	-1.0671	-0.2272	0.5382	-1.4913	2.7068	-0.8004	-0.7550	1.9918	-0.2418	-0.9464	...	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
495	-0.1113	-0.9719	-1.3743	0.5981	0.0933	-0.6940	0.1871	0.1299	1.5912	-1.4806	...	0	0	0	0	0	0	0	0	
496	1.0041	-1.2991	-0.3390	-0.9913	0.9715	-1.9011	0.0309	-0.8254	0.1091	0.6994	...	0	0	0	0	0	0	0	0	
497	-1.1469	0.4460	0.2880	-0.9857	-0.6334	-0.9991	0.2086	1.3829	-0.1830	0.0712	...	0	0	0	0	0	0	0	0	
498	-0.7404	-0.6993	-1.0944	-1.2068	-0.9220	1.5662	0.8884	-0.1025	-1.6279	-0.9392	...	0	0	0	0	0	0	0	0	
499	0.3856	0.6144	0.6692	-0.0430	-0.2806	-0.7802	-0.4089	0.3448	0.5950	0.7740	...	0	0	0	0	0	0	0	0	

500 rows × 662 columns

In [130…
```
sum(list(df_test.isnull().sum()))
```

Out[130… 0

- No missing values in Unlabeled Data.

In [131…
```
#  convert all string values in these features to numeric values
for col in col_names_mutGene:
    df_test[col] = df_test[col].apply(lambda x: 0 if (x=='0' or x==0) else 1)
```

In [132…
```
# Select Target features
target_col_forTest = target_col.copy()
target_col_forTest.remove('outcome')
len(target_col_forTest)

df_test2 = df_test[target_col_forTest].copy()
df_test2.shape
```

Out[132… (500, 254)

In [133…
```
# normalize continuous features
```

```
for i in col_names_Gene:
    df_test2.loc[:,i] = normalization_func(df_test2.loc[:,i])

df_test2.describe()
```

Out[133…

	pten	cdh1	chek2	nf1	bard1	mlh1	msh6	epcam	rbl1	ccna1	...	ncor1_mut	stab2_mut	rb1_mut	k
count	5.000000e+02	5.000000e+02	5.000000e+02	5.000000e+02	5.000000e+02	5.000000e+02	5.000000e+02	5.000000e+02	5.000000e+02	5.000000e+02	...	500.000000	500.000000	500.00000	
mean	1.421085e-17	1.065814e-17	-5.329071e-18	-2.842171e-17	1.065814e-17	3.552714e-18	-1.065814e-17	-8.881784e-18	-5.329071e-18	-7.105427e-18	...	0.032000	0.062000	0.02200	
std	1.001002e+00	1.001002e+00	1.001002e+00	1.001002e+00	1.001002e+00	1.001002e+00	1.001002e+00	1.001002e+00	1.001002e+00	1.001002e+00	...	0.176176	0.241397	0.14683	
min	-5.428672e+00	-3.050453e+00	-2.100484e+00	-2.365425e+00	-2.868891e+00	-5.304116e+00	-2.663288e+00	-1.882729e+00	-2.344140e+00	-1.261457e+00	...	0.000000	0.000000	0.00000	
25%	-5.687240e-01	-4.355133e-01	-7.392572e-01	-6.892045e-01	-6.787414e-01	-5.895867e-01	-6.868272e-01	-6.829077e-01	-6.565924e-01	-4.763497e-01	...	0.000000	0.000000	0.00000	
50%	1.057546e-01	9.570715e-02	-1.504834e-01	-7.905495e-02	-4.782878e-02	1.235139e-01	-8.399092e-02	-1.444286e-01	-1.609575e-01	-2.222191e-01	...	0.000000	0.000000	0.00000	
75%	6.299958e-01	6.373637e-01	4.946314e-01	6.371382e-01	6.564663e-01	6.527362e-01	5.523078e-01	4.910615e-01	4.804253e-01	9.697794e-02	...	0.000000	0.000000	0.00000	
max	3.304990e+00	2.770672e+00	4.340576e+00	5.785557e+00	2.998663e+00	3.387827e+00	3.914279e+00	6.717527e+00	4.163356e+00	8.896280e+00	...	1.000000	1.000000	1.00000	

8 rows × 254 columns

In [134…

```
# make predictions
testtest = PandasDataset(df_test2)
test_loader_pred = DataLoader(testtest, batch_size=64, shuffle=False)

# Combine the unnormalized predictions from dfferent batches
test_pred = torch.cat(trainer.predict(model, dataloaders=test_loader_pred))

# predictive probabilit for each class
test_pred_prob = torch.softmax(test_pred, dim = 1)
```

```
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/pytorch_lightning/trainer/connectors/data_connector.py:245: PossibleUserWarning: The dataloader, predic
t_dataloader 0, does not have many workers which may be a bottleneck. Consider increasing the value of the `num_workers` argument` (try 4 which is the number of cpus on t
his machine) in the `DataLoader` init to improve performance.
  category=PossibleUserWarning,
```

In [137…

```
df_pred = pd.DataFrame(test_pred_prob[:,1])
df_pred
```

Out[137…

	0
0	0.136517
1	0.388557
2	0.260325
3	0.391593
4	0.367849
...	...
495	0.362117
496	0.503329
497	0.504274
498	0.496184
499	0.036131

500 rows × 1 columns

In [138…
```
# save the predictions
df_pred.to_csv('F0034WQ_prediction.csv', index=False, header = False)
```

2. t-SNE and UMAP for Data Visualization

In [142…
```
df = pd.read_csv('five-cancers-gene-expression.csv')
df
```

Out[142…

	Patient	Cancer	gene_13968	gene_18617	gene_20131	gene_12738	gene_5337	gene_20496	gene_3408	gene_15234	…	gene_4911	gene_4915	gene_521	gene_8588	gene_7758	gene_7221
0	sample_100	BRCA	8.591709	0.0	7.092408	9.142789	5.805636	7.033654	9.370833	5.209453	…	8.238725	9.715028	8.106783	0.341075	9.342075	8.044028
1	sample_102	BRCA	8.801388	0.0	5.891225	8.094901	3.039454	5.237969	9.180374	4.675483	…	5.235980	8.728067	7.949056	0.366140	9.433381	5.422445
2	sample_111	BRCA	8.612400	0.0	6.125393	7.892027	4.830783	5.890634	9.768285	5.335144	…	6.274644	9.391233	7.007846	0.000000	9.637172	6.409890
3	sample_114	BRCA	8.293775	0.0	5.747242	8.353469	3.768322	5.979115	9.321678	6.105724	…	8.646094	9.354020	7.111772	0.000000	9.209232	8.052677
4	sample_118	BRCA	8.776821	0.0	7.026391	7.372264	3.824371	5.639357	9.588062	5.239612	…	10.725682	9.139298	8.196587	0.000000	9.473982	4.245861
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
745	sample_9	PRAD	7.499455	0.0	4.953819	8.387251	5.960875	3.314972	9.575887	6.062892	…	3.595014	10.122841	2.635987	0.724214	9.816036	1.204141
746	sample_90	PRAD	8.203402	0.0	5.574053	8.475474	2.212320	4.580477	8.733324	7.489286	…	4.065917	9.722710	2.212320	0.000000	9.099471	6.724991
747	sample_91	PRAD	8.278584	0.0	5.918167	7.897331	4.232231	3.928039	8.348648	6.776630	…	5.962177	9.900328	5.174750	0.000000	9.326582	8.458357
748	sample_95	PRAD	8.240820	0.0	5.603068	7.973227	1.957989	3.537793	9.373994	5.225965	…	1.288418	9.842309	3.424600	0.000000	9.055936	6.034674
749	sample_98	PRAD	8.195097	0.0	5.784674	7.457011	5.051124	4.156024	9.805568	6.005330	…	9.477117	9.314243	1.382778	0.000000	9.262273	5.784674

750 rows × 77 columns

2-a. Apply t-SNE in the scikit-learn library with n_components=2

In [143…
```
df_2 = df.drop(['Patient','Cancer'],axis=1).copy()
df_2
```

Out[143…

	gene_13968	gene_18617	gene_20131	gene_12738	gene_5337	gene_20496	gene_3408	gene_15234	gene_2833	gene_6139	…	gene_4911	gene_4915	gene_521	gene_8588	gene_7758	gene_…
0	8.591709	0.0	7.092408	9.142789	5.805636	7.033654	9.370833	5.209453	2.662957	9.716420	…	8.238725	9.715028	8.106783	0.341075	9.342075	8.044
1	8.801388	0.0	5.891225	8.094901	3.039454	5.237969	9.180374	4.675483	3.039454	9.393041	…	5.235980	8.728067	7.949056	0.366140	9.433381	5.422
2	8.612400	0.0	6.125393	7.892027	4.830783	5.890634	9.768285	5.335144	4.912295	10.056543	…	6.274644	9.391233	7.007846	0.000000	9.637172	6.409
3	8.293775	0.0	5.747242	8.353469	3.768322	5.979115	9.321678	6.105724	2.519743	9.776866	…	8.646094	9.354020	7.111772	0.000000	9.209232	8.052
4	8.776821	0.0	7.026391	7.372264	3.824371	5.639357	9.588062	5.239612	4.041331	9.846992	…	10.725682	9.139298	8.196587	0.000000	9.473982	4.245
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	
745	7.499455	0.0	4.953819	8.387251	5.960875	3.314972	9.575887	6.062892	3.141433	9.556031	…	3.595014	10.122841	2.635987	0.724214	9.816036	1.20
746	8.203402	0.0	5.574053	8.475474	2.212320	4.580477	8.733324	7.489286	3.957599	9.102653	…	4.065917	9.722710	2.212320	0.000000	9.099471	6.724
747	8.278584	0.0	5.918167	7.897331	4.232231	3.928039	8.348648	6.776630	1.740453	8.867072	…	5.962177	9.900328	5.174750	0.000000	9.326582	8.458
748	8.240820	0.0	5.603068	7.973227	1.957989	3.537793	9.373994	5.225965	5.901745	10.104114	…	1.288418	9.842309	3.424600	0.000000	9.055936	6.034
749	8.195097	0.0	5.784674	7.457011	5.051124	4.156024	9.805568	6.005330	7.053470	9.985643	…	9.477117	9.314243	1.382778	0.000000	9.262273	5.784

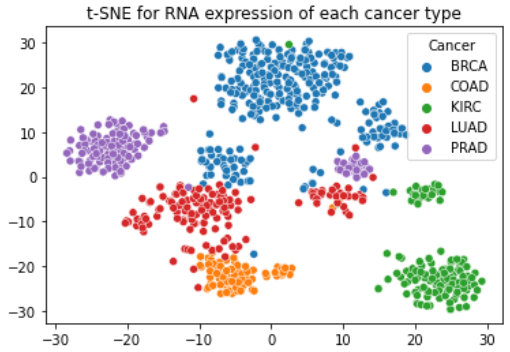
750 rows × 75 columns

In [144…
```
# Apply t-SNE
from sklearn.manifold import TSNE
df_2_tsne = TSNE(n_components=2).fit_transform(df_2)
df_2_tsne
```

Out[144…
```
array([[ 14.123409 ,  10.04349  ],
       [ -3.0648112,  16.480053 ],
       [ -1.494606 ,  17.418083 ],
       ...,
       [-26.139647 ,   8.773267 ],
       [-24.299377 ,   5.1945233],
       [-16.949133 ,   8.053046 ]], dtype=float32)
```

In [147…
```
# Scatter plot
import seaborn as sns
import matplotlib.pyplot as plt

sns.scatterplot(x= df_2_tsne[:,0], y= df_2_tsne[:,1], hue= df['Cancer'])
plt.title('t-SNE for RNA expression of each cancer type')
```

Out[147…
```
Text(0.5, 1.0, 't-SNE for RNA expression of each cancer type')
```

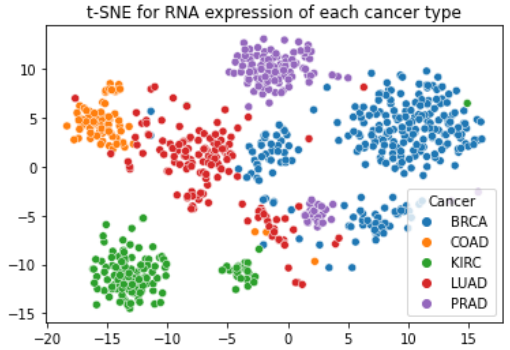


According to the plot, we can see t-SNE can map high dimensional data to 2 dimensional space and 5 cancer cell types are separated very well. However, we can see some cancer types, such as PRAD, LUAD, BRCA, and KIRC have multiple clusters. I think these small clusters might be the subtypes for those main cancer types. Also, we can see some samples of LUAD are closer to BRCA compared with LUAD. I think these samples might be outliers.

2-b. Repeat the procedures in part a, with additional hyperparameter tuning

In [160…
```
df_2_tsne_2 = TSNE(n_components=2, perplexity = 60, learning_rate = 500, n_iter = 1500).fit_transform(df_2)

sns.scatterplot(x= df_2_tsne_2[:,0], y= df_2_tsne_2[:,1], hue= df['Cancer'])
plt.title('t-SNE for RNA expression of each cancer type')
```

Out[160…
```
Text(0.5, 1.0, 't-SNE for RNA expression of each cancer type')
```



Here, I used a high learning rate (500) and high perplexity (60). The perplexity is related to the number of nearest neighbors that are used in other manifold learning algorithms. So we can see the points are close together in the cluster compared with part a. We also can significantly see that samples in COAD and LUAD are distinguishable at the middle-lower region of the plot. As a result, this large dataset is reasonable to use large perplexity.

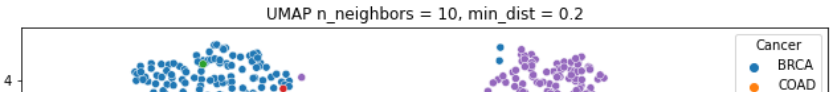
2-c. Apply UMAP on the same dataset.

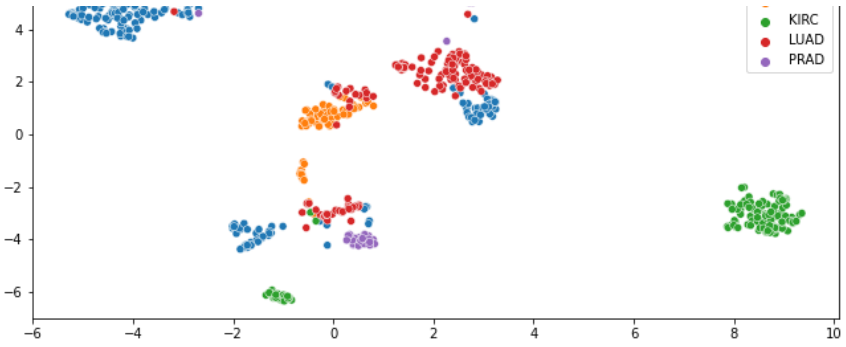
- Reference: https://umap-learn.readthedocs.io/en/latest/basic_usage.html
- Reference: https://umap-learn.readthedocs.io/en/latest/parameters.html

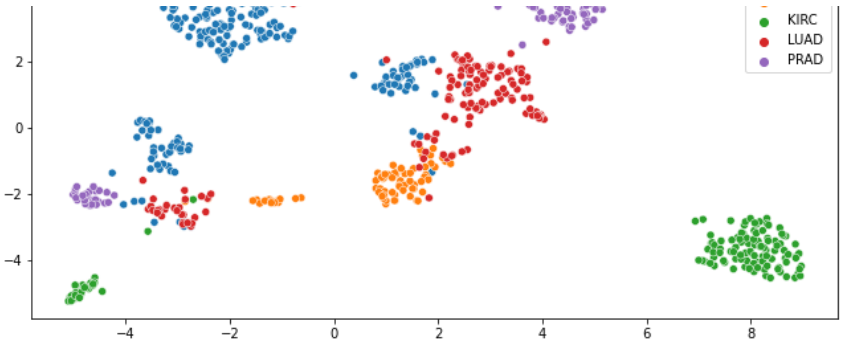
In [195…
```
# Load module
import umap
```

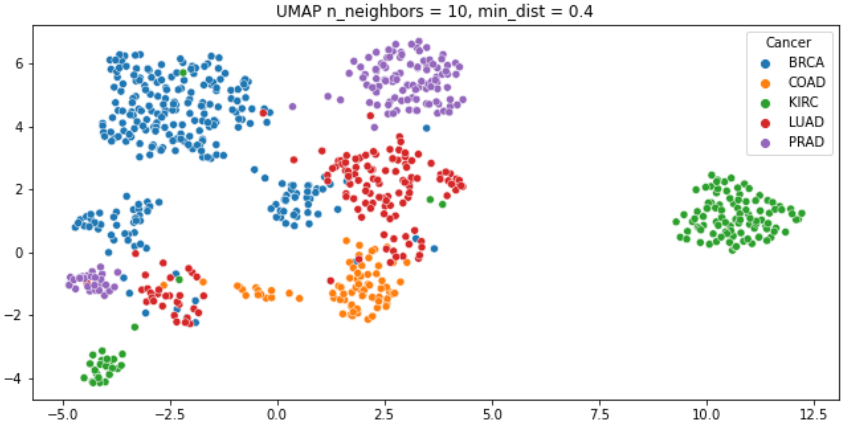
In [193…
```
fig, axes = plt.subplots(4, 2, figsize=(24, 24))

# UMAP n_neighbors = 10
df_UMAP1 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 10, min_dist = 0.1).fit_transform(df_2)
sns.scatterplot(x= df_UMAP1[:,0], y= df_UMAP1[:,1], hue= df['Cancer'], ax=axes[0,0]).set( title = "UMAP n_neighbors = 10, min_dist = 0.1")

df_UMAP2 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 10, min_dist = 0.2).fit_transform(df_2)
sns.scatterplot(x= df_UMAP2[:,0], y= df_UMAP2[:,1], hue= df['Cancer'], ax=axes[0,1]).set( title = "UMAP n_neighbors = 10, min_dist = 0.2")

df_UMAP3 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 10, min_dist = 0.4).fit_transform(df_2)
sns.scatterplot(x= df_UMAP3[:,0], y= df_UMAP3[:,1], hue= df['Cancer'], ax=axes[1,0]).set( title = "UMAP n_neighbors = 10, min_dist = 0.4")

df_UMAP4 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 10, min_dist = 0.8).fit_transform(df_2)
sns.scatterplot(x= df_UMAP4[:,0], y= df_UMAP4[:,1], hue= df['Cancer'], ax=axes[1,1]).set( title = "UMAP n_neighbors = 10, min_dist = 0.8")


# UMAP n_neighbors = 20
df_UMAP5 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 20, min_dist = 0.1).fit_transform(df_2)
sns.scatterplot(x= df_UMAP5[:,0], y= df_UMAP5[:,1], hue= df['Cancer'], ax=axes[2,0]).set( title = "UMAP n_neighbors = 20, min_dist = 0.1")

df_UMAP6 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 20, min_dist = 0.2).fit_transform(df_2)
sns.scatterplot(x= df_UMAP6[:,0], y= df_UMAP6[:,1], hue= df['Cancer'], ax=axes[2,1]).set( title = "UMAP n_neighbors = 20, min_dist = 0.2")

df_UMAP7 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 20, min_dist = 0.4).fit_transform(df_2)
sns.scatterplot(x= df_UMAP7[:,0], y= df_UMAP7[:,1], hue= df['Cancer'], ax=axes[3,0]).set( title = "UMAP n_neighbors = 20, min_dist = 0.4")

df_UMAP8 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 20, min_dist = 0.8).fit_transform(df_2)
sns.scatterplot(x= df_UMAP8[:,0], y= df_UMAP8[:,1], hue= df['Cancer'], ax=axes[3,1]).set( title = "UMAP n_neighbors = 20, min_dist = 0.8")
```

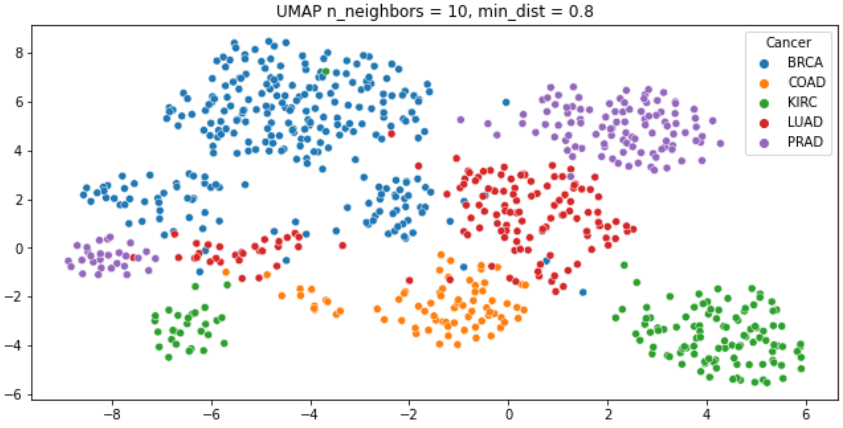
Out[193…
```
[Text(0.5, 1.0, 'UMAP n_neighbors = 20, min_dist = 0.8')]
```

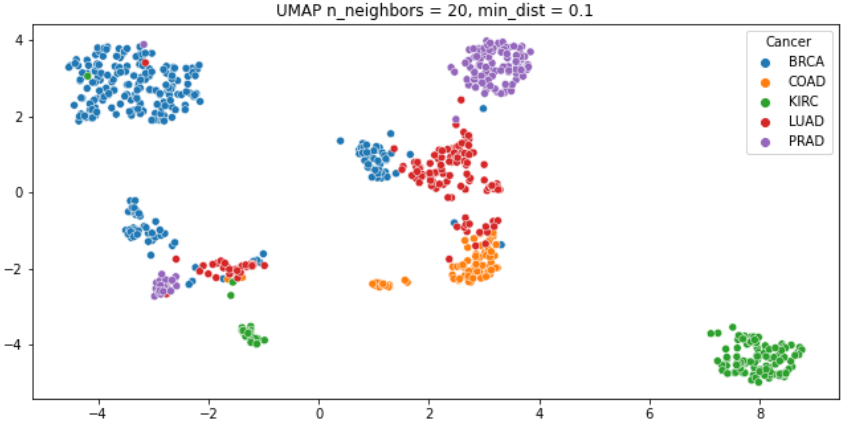


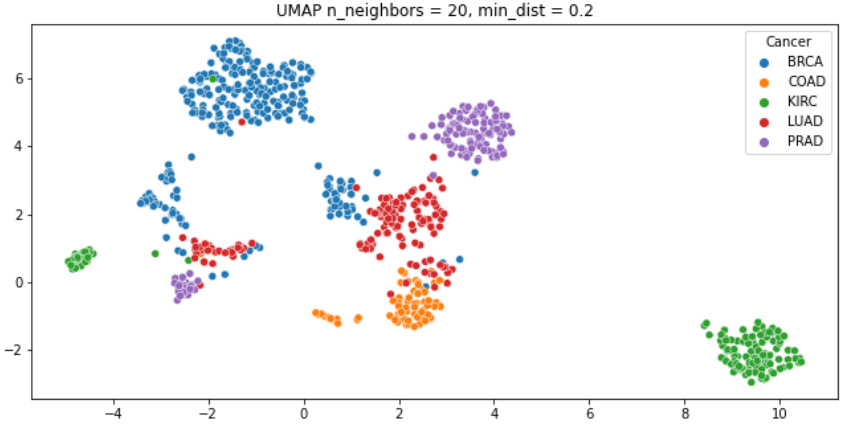

KIRC
LUAD
PRAD

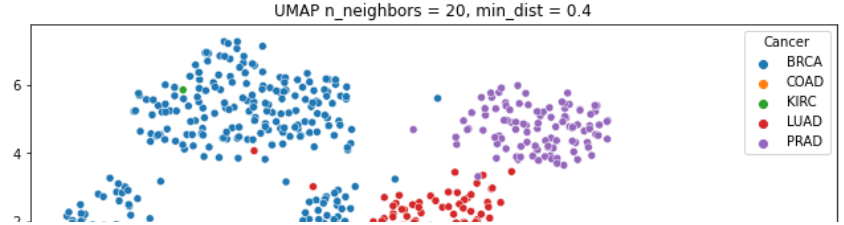

KIRC
LUAD
PRAD

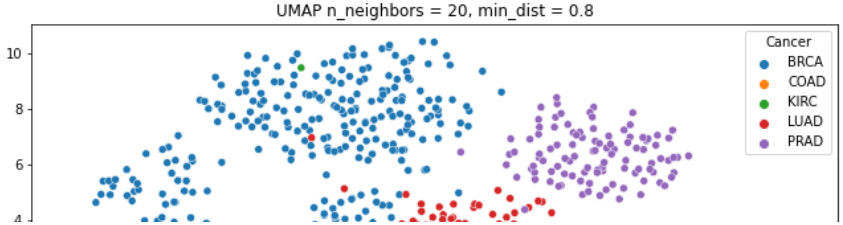

UMAP n_neighbors = 10, min_dist = 0.4
Cancer
BRCA
COAD
KIRC
LUAD
PRAD

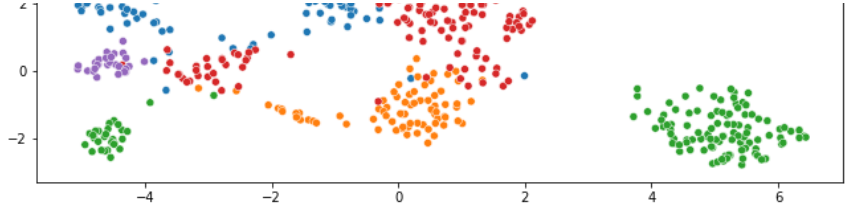

UMAP n_neighbors = 10, min_dist = 0.8
Cancer
BRCA
COAD
KIRC
LUAD
PRAD

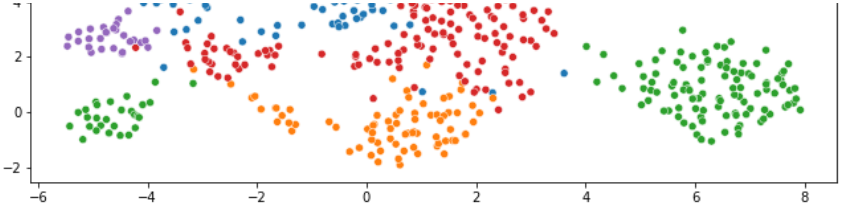

UMAP n_neighbors = 20, min_dist = 0.1
Cancer
BRCA
COAD
KIRC
LUAD
PRAD


UMAP n_neighbors = 20, min_dist = 0.2
Cancer
BRCA
COAD
KIRC
LUAD
PRAD


UMAP n_neighbors = 20, min_dist = 0.4
Cancer
BRCA
COAD
KIRC
LUAD
PRAD


UMAP n_neighbors = 20, min_dist = 0.8
Cancer
BRCA
COAD
KIRC
LUAD
PRAD



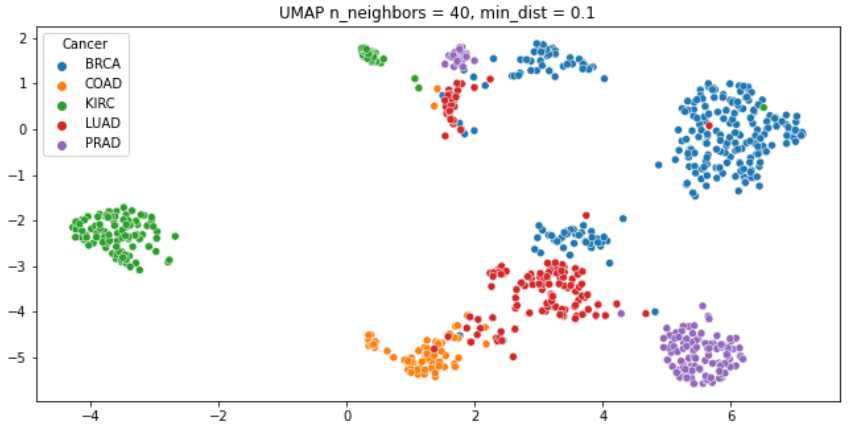


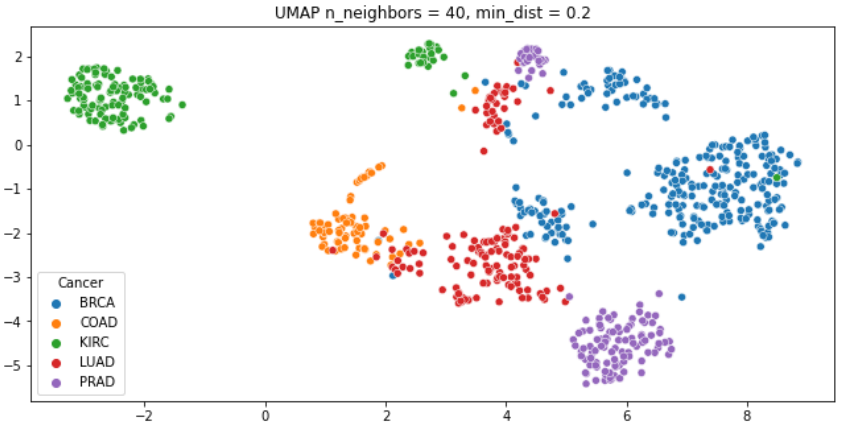
In [194…

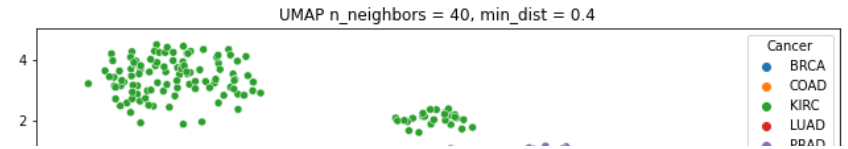
```
fig, axes = plt.subplots(4, 2, figsize=(24, 24))

# n_neighbors = 40
df_UMAP9 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 40, min_dist = 0.1).fit_transform(df_2)
sns.scatterplot(x= df_UMAP9[:,0], y= df_UMAP9[:,1], hue= df['Cancer'], ax=axes[0,0]).set( title = "UMAP n_neighbors = 40, min_dist = 0.1")

df_UMAP10 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 40, min_dist = 0.2).fit_transform(df_2)
sns.scatterplot(x= df_UMAP10[:,0], y= df_UMAP10[:,1], hue= df['Cancer'], ax=axes[0,1]).set( title = "UMAP n_neighbors = 40, min_dist = 0.2")

df_UMAP11 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 40, min_dist = 0.4).fit_transform(df_2)
sns.scatterplot(x= df_UMAP11[:,0], y= df_UMAP11[:,1], hue= df['Cancer'], ax=axes[1,0]).set( title = "UMAP n_neighbors = 40, min_dist = 0.4")

df_UMAP12 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 40, min_dist = 0.8).fit_transform(df_2)
sns.scatterplot(x= df_UMAP12[:,0], y= df_UMAP12[:,1], hue= df['Cancer'], ax=axes[1,1]).set( title = "UMAP n_neighbors = 40, min_dist = 0.8")

# UMAP n_neighbors = 80
df_UMAP13 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 80, min_dist = 0.1).fit_transform(df_2)
sns.scatterplot(x= df_UMAP13[:,0], y= df_UMAP13[:,1], hue= df['Cancer'], ax=axes[2,0]).set( title = "UMAP n_neighbors = 80, min_dist = 0.1")

df_UMAP14 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 80, min_dist = 0.2).fit_transform(df_2)
sns.scatterplot(x= df_UMAP14[:,0], y= df_UMAP14[:,1], hue= df['Cancer'], ax=axes[2,1]).set( title = "UMAP n_neighbors = 80, min_dist = 0.2")

df_UMAP15 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 80, min_dist = 0.4).fit_transform(df_2)
sns.scatterplot(x= df_UMAP15[:,0], y= df_UMAP15[:,1], hue= df['Cancer'], ax=axes[3,0]).set( title = "UMAP n_neighbors = 80, min_dist = 0.4")

df_UMAP16 = umap.UMAP(n_components=2, metric='euclidean', n_neighbors = 80, min_dist = 0.8).fit_transform(df_2)
sns.scatterplot(x= df_UMAP16[:,0], y= df_UMAP16[:,1], hue= df['Cancer'], ax=axes[3,1]).set( title = "UMAP n_neighbors = 80, min_dist = 0.8")
```

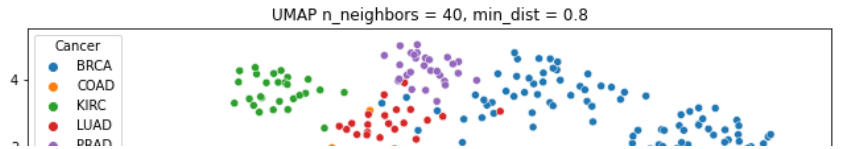
Out[194…

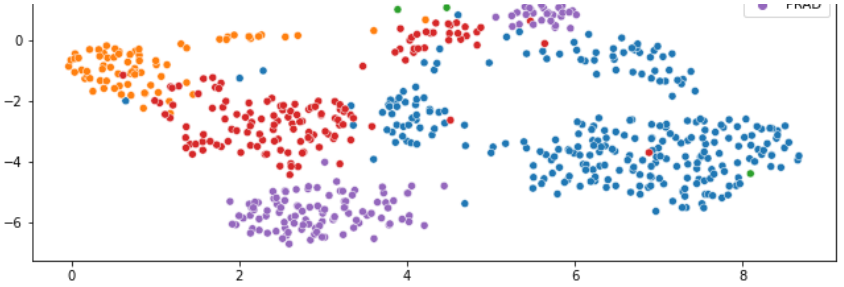
```
[Text(0.5, 1.0, 'UMAP n_neighbors = 80, min_dist = 0.8')]
```

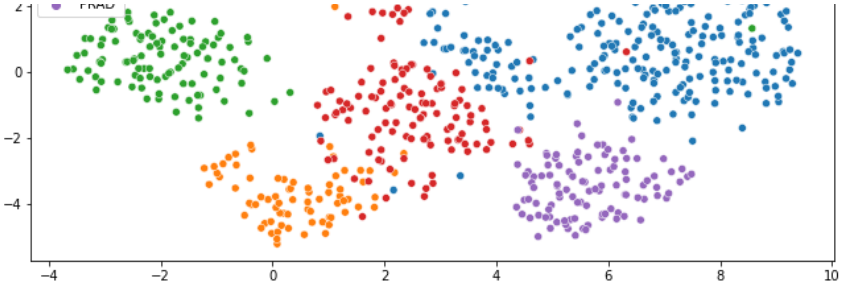


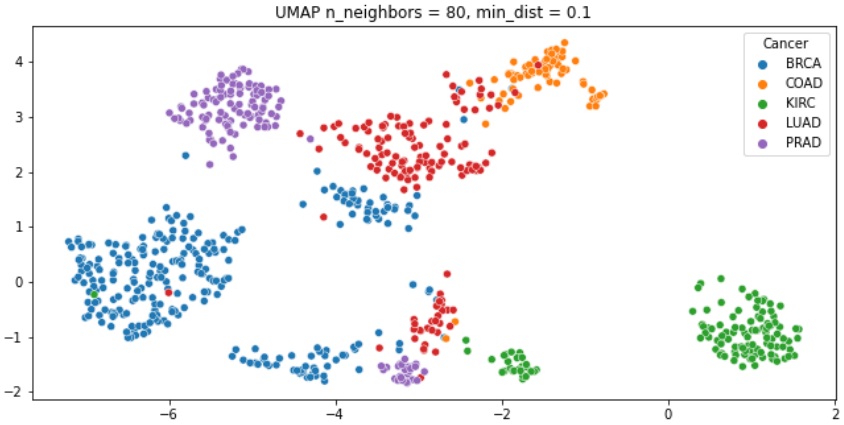


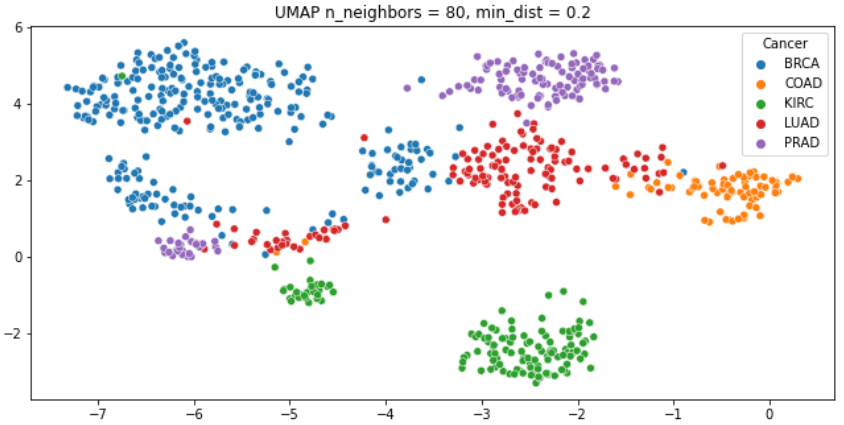


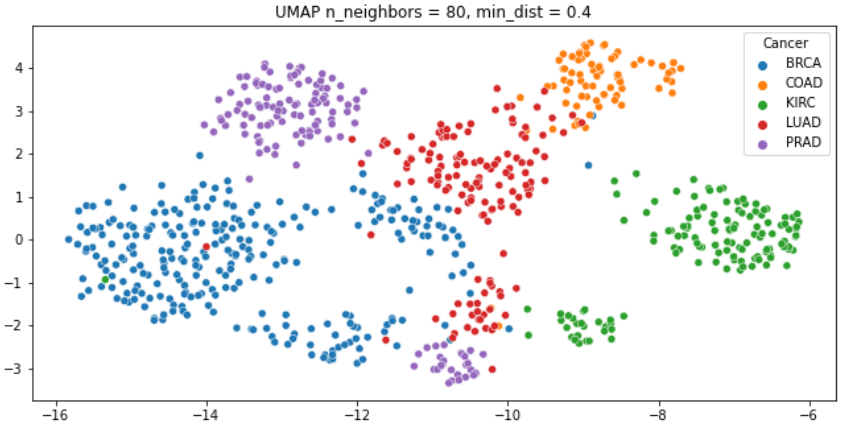


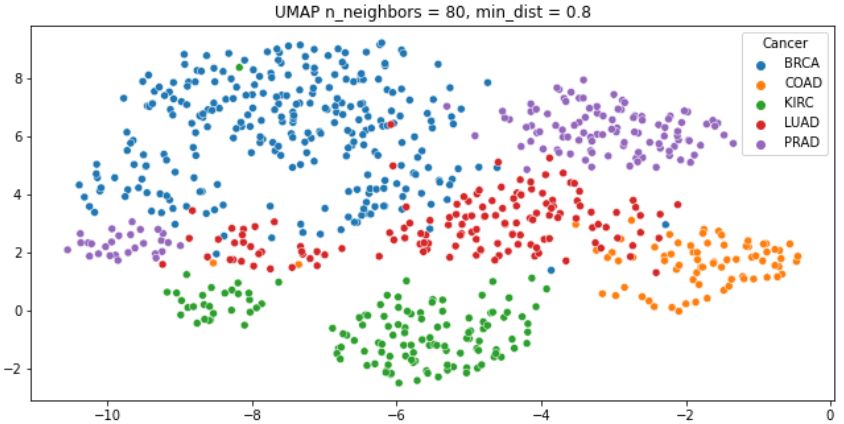












Based on https://umap-learn.readthedocs.io/en/latest/parameters.html, n_neighbors controls how UMAP balances local versus global structure in the data. Low values of n_neighbors will force UMAP to concentrate on a very local structure, and too large values lose detail structure. As a result, with n_neighbors increasing, each cancer type cluster is not compacted/dense compared with low n_neighbors values. The min_dist controls how tightly UMAP is allowed to pack points together. So in my opinion, I think the best combination of hyperparameters is n_components=2, metric='euclidean', n_neighbors = 20, and min_dist = 0.2

2-d. Estimate the time it took to fit t-SNE and UMAP on simulated datasets.

- Reference: https://umap-learn.readthedocs.io/en/latest/parameters.html

In [210…

```
import time

n_obs = [10,100,1000,10000]
n_features = [10,30,100,300]

observation_number = []
feature_number = []
t_SNE_seconds = []
UMAP_seconds = []

method_tSNE = TSNE()
method_UMAP = umap.UMAP()

for i in n_obs:
    for j in n_features:
        observation_number.append(i)
        feature_number.append(j)
        data = np.random.rand(i, j)

        start_tSNE = time.time()
        method_tSNE.fit_transform(data)
        end_tSNE = time.time()
        total_tSNE = end_tSNE - start_tSNE
        t_SNE_seconds.append(total_tSNE)

        start_UMAP = time.time()
        method_UMAP.fit_transform(data)
        end_UMAP = time.time()
        total_UMAP = end_UMAP - start_UMAP
        UMAP_seconds.append(total_UMAP)
```

```
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/umap/umap_.py:1383: UserWarning: n_neighbors is larger than the dataset size; truncating to X.shape[0]
- 1
  "n_neighbors is larger than the dataset size; truncating to "
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/umap/umap_.py:1383: UserWarning: n_neighbors is larger than the dataset size; truncating to X.shape[0]
- 1
  "n_neighbors is larger than the dataset size; truncating to "
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/umap/umap_.py:1383: UserWarning: n_neighbors is larger than the dataset size; truncating to X.shape[0]
- 1
  "n_neighbors is larger than the dataset size; truncating to "
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/umap/umap_.py:1383: UserWarning: n_neighbors is larger than the dataset size; truncating to X.shape[0]
- 1
  "n_neighbors is larger than the dataset size; truncating to "
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/numba/core/typed_passes.py:314: NumbaPerformanceWarning:
The keyword argument 'parallel=True' was specified but no transformation for parallel execution was possible.

To find out why, try turning on parallel diagnostics, see https://numba.pydata.org/numba-doc/latest/user/parallel.html#diagnostics for help.

File "../../../../../anaconda3/envs/Class/lib/python3.7/site-packages/umap/rp_tree.py", line 135:
@numba.njit(fastmath=True, nogil=True, parallel=True)
def euclidean_random_projection_split(data, indices, rng_state):
^


  state.func_ir.loc))
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/umap/nndescent.py:92: NumbaPerformanceWarning:
The keyword argument 'parallel=True' was specified but no transformation for parallel execution was possible.

To find out why, try turning on parallel diagnostics, see https://numba.pydata.org/numba-doc/latest/user/parallel.html#diagnostics for help.

File "../../../../../anaconda3/envs/Class/lib/python3.7/site-packages/umap/utils.py", line 409:
@numba.njit(parallel=True)
def build_candidates(current_graph, n_vertices, n_neighbors, max_candidates, rng_state):
^


  current_graph, n_vertices, n_neighbors, max_candidates, rng_state
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/numba/core/typed_passes.py:314: NumbaPerformanceWarning:
The keyword argument 'parallel=True' was specified but no transformation for parallel execution was possible.

To find out why, try turning on parallel diagnostics, see https://numba.pydata.org/numba-doc/latest/user/parallel.html#diagnostics for help.
```

```
File "../../../../../anaconda3/envs/Class/lib/python3.7/site-packages/umap/nndescent.py", line 47:
    @numba.njit(parallel=True)
    def nn_descent(
    ^

  state.func_ir.loc))
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/numba/core/typed_passes.py:314: NumbaPerformanceWarning:
The keyword argument 'parallel=True' was specified but no transformation for parallel execution was possible.

To find out why, try turning on parallel diagnostics, see https://numba.pydata.org/numba-doc/latest/user/parallel.html#diagnostics for help.

File "../../../../../anaconda3/envs/Class/lib/python3.7/site-packages/umap/nndescent.py", line 47:
    @numba.njit(parallel=True)
    def nn_descent(
    ^

  state.func_ir.loc))
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/numba/core/typed_passes.py:314: NumbaPerformanceWarning:
The keyword argument 'parallel=True' was specified but no transformation for parallel execution was possible.

To find out why, try turning on parallel diagnostics, see https://numba.pydata.org/numba-doc/latest/user/parallel.html#diagnostics for help.

File "../../../../../anaconda3/envs/Class/lib/python3.7/site-packages/umap/nndescent.py", line 47:
    @numba.njit(parallel=True)
    def nn_descent(
    ^

  state.func_ir.loc))
/Users/chenjiqing/anaconda3/envs/Class/lib/python3.7/site-packages/numba/core/typed_passes.py:314: NumbaPerformanceWarning:
The keyword argument 'parallel=True' was specified but no transformation for parallel execution was possible.

To find out why, try turning on parallel diagnostics, see https://numba.pydata.org/numba-doc/latest/user/parallel.html#diagnostics for help.

File "../../../../../anaconda3/envs/Class/lib/python3.7/site-packages/umap/nndescent.py", line 47:
    @numba.njit(parallel=True)
    def nn_descent(
    ^

  state.func_ir.loc))
```

In [215…

```
result = pd.DataFrame()
result['Number of Observations'] = observation_number
result['Number of Features'] = feature_number
result['t-SNE running time (seconds)'] = t_SNE_seconds
result['UMAP running time (seconds)'] = UMAP_seconds
result
```

Out[215…

	Number of Observations	Number of Features	t-SNE running time (seconds)	UMAP running time (seconds)
0	10	10	0.200663	0.021092
1	10	30	0.222042	0.018828
2	10	100	0.128698	0.018400
3	10	300	0.185330	0.018491
4	100	10	0.487030	0.176302
5	100	30	0.484668	0.187400
6	100	100	0.656883	0.181235
7	100	300	0.477384	0.182112
8	1000	10	6.442281	1.522532
9	1000	30	6.077269	1.551414
10	1000	100	8.740793	1.770407
11	1000	300	7.483002	1.772822

	Number of Observations	Number of Features	t-SNE running time (seconds)	UMAP running time (seconds)
12	10000	10	87.184475	22.233955
13	10000	30	102.072792	19.314443
14	10000	100	120.566159	21.768741
15	10000	300	159.426440	20.267445

- UMAP is faster than t-SNE!!

In [ ]: