In [ ]:

```
!pip install scanpy
!pip install igraph
!pip install louvain
!pip install lifelines
```

```
Requirement already satisfied: scanpy in /usr/local/lib/pytho
n3.7/dist-packages (1.8.2)
Requirement already satisfied: patsy in /usr/local/lib/python
3.7/dist-packages (from scanpy) (0.5.2)
Requirement already satisfied: scikit-learn>=0.22 in /usr/loc
al/lib/python3.7/dist-packages (from scanpy) (1.0.1)
Requirement already satisfied: importlib_metadata>=0.7 in /us
r/local/lib/python3.7/dist-packages (from scanpy) (4.8.2)
Requirement already satisfied: tqdm in /usr/local/lib/python
3.7/dist-packages (from scanpy) (4.62.3)
Requirement already satisfied: sinfo in /usr/local/lib/python
3.7/dist-packages (from scanpy) (0.3.4)
Requirement already satisfied: seaborn in /usr/local/lib/pyth
on3.7/dist-packages (from scanpy) (0.11.2)
Requirement already satisfied: matplotlib>=3.1.2 in /usr/loca
l/lib/python3.7/dist-packages (from scanpy) (3.2.2)
Requirement already satisfied: h5py>=2.10.0 in /usr/local/li
b/python3.7/dist-packages (from scanpy) (3.1.0)
Requirement already satisfied: networkx>=2.3 in /usr/local/li
b/python3.7/dist-packages (from scanpy) (2.6.3)
Requirement already satisfied: umap-learn>=0.3.10 in /usr/loc
al/lib/python3.7/dist-packages (from scanpy) (0.5.2)
Requirement already satisfied: numba>=0.41.0 in /usr/local/li
b/python3.7/dist-packages (from scanpy) (0.51.2)
Requirement already satisfied: anndata>=0.7.4 in /usr/local/l
ib/python3.7/dist-packages (from scanpy) (0.7.8)
Requirement already satisfied: natsort in /usr/local/lib/pyth
on3.7/dist-packages (from scanpy) (5.5.0)
Requirement already satisfied: statsmodels>=0.10.0rc2 in /us
r/local/lib/python3.7/dist-packages (from scanpy) (0.10.2)
Requirement already satisfied: scipy>=1.4 in /usr/local/lib/p
ython3.7/dist-packages (from scanpy) (1.4.1)
Requirement already satisfied: pandas>=0.21 in /usr/local/li
b/python3.7/dist-packages (from scanpy) (1.1.5)
Requirement already satisfied: tables in /usr/local/lib/pytho
n3.7/dist-packages (from scanpy) (3.4.4)
Requirement already satisfied: numpy>=1.17.0 in /usr/local/li
b/python3.7/dist-packages (from scanpy) (1.19.5)
Requirement already satisfied: packaging in /usr/local/lib/py
thon3.7/dist-packages (from scanpy) (21.3)
Requirement already satisfied: joblib in /usr/local/lib/pytho
n3.7/dist-packages (from scanpy) (1.1.0)
Requirement already satisfied: xlrd<2.0 in /usr/local/lib/pyt
hon3.7/dist-packages (from anndata>=0.7.4->scanpy) (1.1.0)
Requirement already satisfied: cached-property in /usr/local/
lib/python3.7/dist-packages (from h5py>=2.10.0->scanpy) (1.5.
2)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/py
thon3.7/dist-packages (from importlib_metadata>=0.7->scanpy)
(3.6.0)
Requirement already satisfied: typing-extensions>=3.6.4 in /u
Requirement already satisfied: typing-extensions>=3.6.4 in /u
sr/local/lib/python3.7/dist-packages (from importlib_metadata
>=0.7->scanpy) (3.10.0.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/loca
l/lib/python3.7/dist-packages (from matplotlib>=3.1.2->scanp
y) (1.3.2)
Requirement already satisfied: python-dateutil>=2.1 in /usr/l
ocal/lib/python3.7/dist-packages (from matplotlib>=3.1.2->sca
npy) (2.8.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/li
b/python3.7/dist-packages (from matplotlib>=3.1.2->scanpy)
(0.11.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.
1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from m
atplotlib>=3.1.2->scanpy) (3.0.6)
Requirement already satisfied: setuptools in /usr/local/lib/p
ython3.7/dist-packages (from numba>=0.41.0->scanpy) (57.4.0)
Requirement already satisfied: llvmlite<0.35,>=0.34.0.dev0 in
/usr/local/lib/python3.7/dist-packages (from numba>=0.41.0->s
canpy) (0.34.0)
Requirement already satisfied: pytz>=2017.2 in /usr/local/li
b/python3.7/dist-packages (from pandas>=0.21->scanpy) (2018.
9)
Requirement already satisfied: six>=1.5 in /usr/local/lib/pyt
hon3.7/dist-packages (from python-dateutil>=2.1->matplotlib>=
3.1.2->scanpy) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/l
ocal/lib/python3.7/dist-packages (from scikit-learn>=0.22->sc
anpy) (3.0.0)
Requirement already satisfied: pynndescent>=0.5 in /usr/loca
l/lib/python3.7/dist-packages (from umap-learn>=0.3.10->scanp
y) (0.5.5)
Requirement already satisfied: stdlib-list in /usr/local/lib/
python3.7/dist-packages (from sinfo->scanpy) (0.8.0)
Requirement already satisfied: numexpr>=2.5.2 in /usr/local/l
ib/python3.7/dist-packages (from tables->scanpy) (2.7.3)
Requirement already satisfied: igraph in /usr/local/lib/pytho
n3.7/dist-packages (0.9.8)
Requirement already satisfied: texttable>=1.6.2 in /usr/loca
l/lib/python3.7/dist-packages (from igraph) (1.6.4)
Requirement already satisfied: louvain in /usr/local/lib/pyth
on3.7/dist-packages (0.7.0)
Requirement already satisfied: python-igraph>=0.8.0 in /usr/l
ocal/lib/python3.7/dist-packages (from louvain) (0.9.8)
Requirement already satisfied: igraph==0.9.8 in /usr/local/li
b/python3.7/dist-packages (from python-igraph>=0.8.0->louvai
n) (0.9.8)
Requirement already satisfied: texttable>=1.6.2 in /usr/loca
l/lib/python3.7/dist-packages (from igraph==0.9.8->python-igr
aph>=0.8.0->louvain) (1.6.4)
Requirement already satisfied: lifelines in /usr/local/lib/py
thon3.7/dist-packages (0.26.3)
Requirement already satisfied: autograd-gamma>=0.3 in /usr/lo
cal/lib/python3.7/dist-packages (from lifelines) (0.5.0)
Requirement already satisfied: matplotlib>=3.0 in /usr/local/
lib/python3.7/dist-packages (from lifelines) (3.2.2)
Requirement already satisfied: scipy>=1.2.0 in /usr/local/li
b/python3.7/dist-packages (from lifelines) (1.4.1)
Requirement already satisfied: formulaic<0.3,>=0.2.2 in /usr/
local/lib/python3.7/dist-packages (from lifelines) (0.2.4)
Requirement already satisfied: pandas>=0.23.0 in /usr/local/l
Requirement already satisfied: pandas>=0.23.0 in /usr/local/l
ib/python3.7/dist-packages (from lifelines) (1.1.5)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/li
b/python3.7/dist-packages (from lifelines) (1.19.5)
Requirement already satisfied: autograd>=1.3 in /usr/local/li
b/python3.7/dist-packages (from lifelines) (1.3)
Requirement already satisfied: future>=0.15.2 in /usr/local/l
ib/python3.7/dist-packages (from autograd>=1.3->lifelines)
(0.16.0)
Requirement already satisfied: wrapt in /usr/local/lib/python
3.7/dist-packages (from formulaic<0.3,>=0.2.2->lifelines) (1.
13.3)
Requirement already satisfied: astor in /usr/local/lib/python
3.7/dist-packages (from formulaic<0.3,>=0.2.2->lifelines) (0.
8.1)
Requirement already satisfied: interface-meta>=1.2 in /usr/lo
cal/lib/python3.7/dist-packages (from formulaic<0.3,>=0.2.2->
lifelines) (1.2.4)
Requirement already satisfied: python-dateutil>=2.1 in /usr/l
ocal/lib/python3.7/dist-packages (from matplotlib>=3.0->lifel
ines) (2.8.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/li
b/python3.7/dist-packages (from matplotlib>=3.0->lifelines)
(0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/loca
l/lib/python3.7/dist-packages (from matplotlib>=3.0->lifeline
s) (1.3.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.
1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from m
atplotlib>=3.0->lifelines) (3.0.6)
Requirement already satisfied: pytz>=2017.2 in /usr/local/li
b/python3.7/dist-packages (from pandas>=0.23.0->lifelines) (2
018.9)
Requirement already satisfied: six>=1.5 in /usr/local/lib/pyt
hon3.7/dist-packages (from python-dateutil>=2.1->matplotlib>=
3.0->lifelines) (1.15.0)
```

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as sps
import seaborn as sns
import scanpy as sc
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
from sklearn.neighbors import kneighbors_graph
from sklearn.cluster import SpectralClustering
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import statsmodels.api as sm
from sklearn.metrics import accuracy_score, r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from scipy import stats
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from lifelines import KaplanMeierFitter
```

In [ ]:

```
sample = [0]*6
for i in range(4):
  sample[i] = pd.read_table(f'/content/drive/MyDrive/sample_{i+1}.txt', se
p = '\t', index_col = 'Gene').transpose()

data =pd.concat([sample[0],sample[1], sample[2], sample[3]],axis=0)
```

In [ ]:

```
adata_1 = sc.AnnData(X = data)
adata_1.var["mt"] = adata_1.var_names.str.startswith("MT-")
adata_1
```

Out[ ]:

```
AnnData object with n_obs × n_vars = 4677 × 27899
    var: 'mt'
```

Контроль качества

Фильтрация клеток

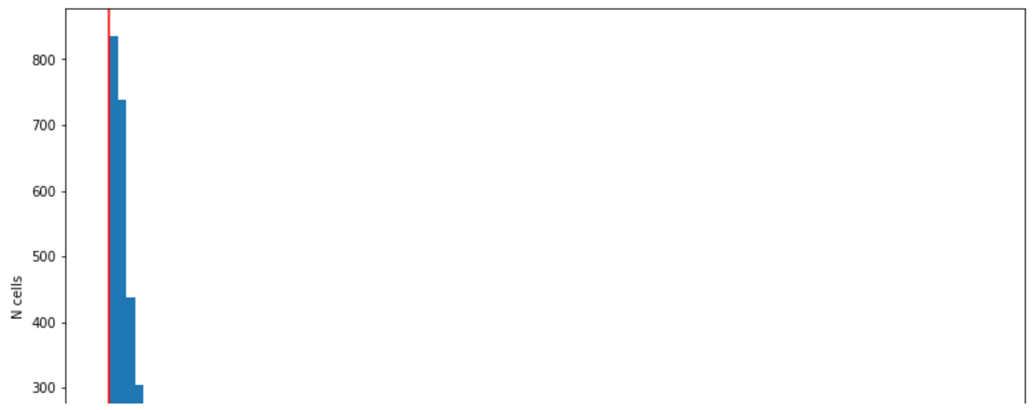
In [ ]:

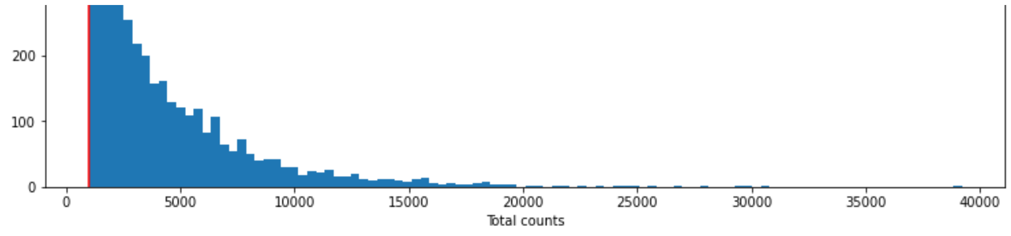
```
qc = sc.pp.calculate_qc_metrics(adata_1, qc_vars = ['mt'])

cell_qc_dataframe = qc[0]
gene_qc_dataframe = qc[1]
```

Убераем клетки, в которых меньше 1000 прочтений

In [ ]:

```
plt.figure(figsize = (13,8))
plt.hist(cell_qc_dataframe['total_counts'], bins=100)
plt.xlabel('Total counts')
plt.ylabel('N cells')
plt.axvline(1000, color='red');
```





In [ ]:

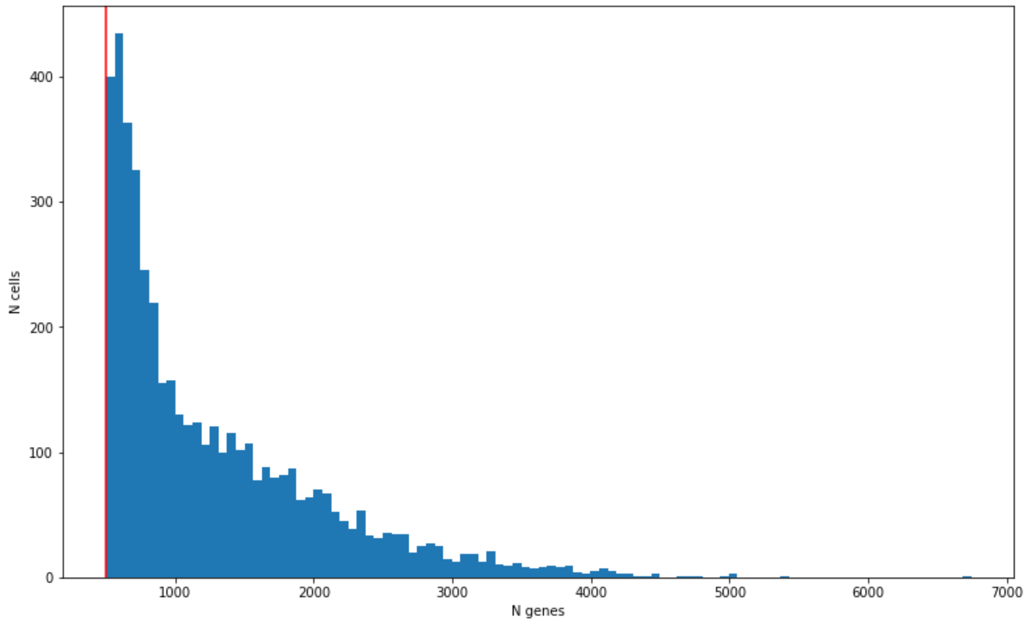
```
print('Started with: \n', adata_1)
sc.pp.filter_cells(adata_1, min_counts = 1000)
print('Finished with: \n', adata_1)
```

```
Started with:
 AnnData object with n_obs × n_vars = 4677 × 27899
    var: 'mt'
Finished with:
 AnnData object with n_obs × n_vars = 4677 × 27899
    obs: 'n_counts'
    var: 'mt'
```

Убираем клетки с маленьким количеством генов

In [ ]:

```
plt.figure(figsize = (13,8))
plt.hist(cell_qc_dataframe['n_genes_by_counts'], bins=100)
plt.xlabel('N genes')
plt.ylabel('N cells')
plt.axvline(500, color='red');
```



In [ ]:

```
print('Started with: \n', adata_1)
sc.pp.filter_cells(adata_1, min_genes = 500)
print('Finished with: \n', adata_1)
```

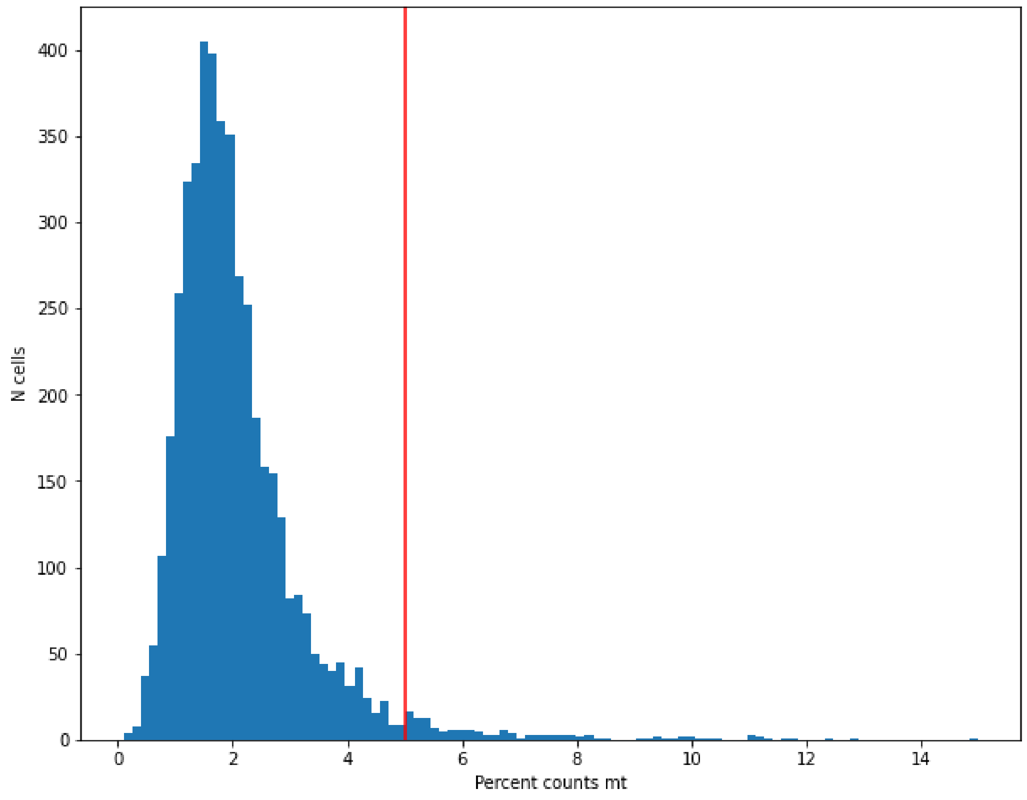
```
Started with:
 AnnData object with n_obs × n_vars = 4677 × 27899
```

```
    obs: 'n_counts'
    var: 'mt'
Finished with:
 AnnData object with n_obs × n_vars = 4677 × 27899


    obs: 'n_counts', 'n_genes'
    var: 'mt'
```

Убираем клетки с большим количеством mt генов

In [ ]:

```
plt.figure(figsize = (10,8))
plt.hist(cell_qc_dataframe['pct_counts_mt'], bins=100)
plt.xlabel('Percent counts mt')
plt.ylabel('N cells')
plt.axvline(5, color='red');
```



In [ ]:

```
print('Started with: \n', adata_1)
high_mt_mask = (cell_qc_dataframe['pct_counts_mt'] < 5)
adata_1 = adata_1[high_mt_mask]
print('Finished with: \n', adata_1)
```

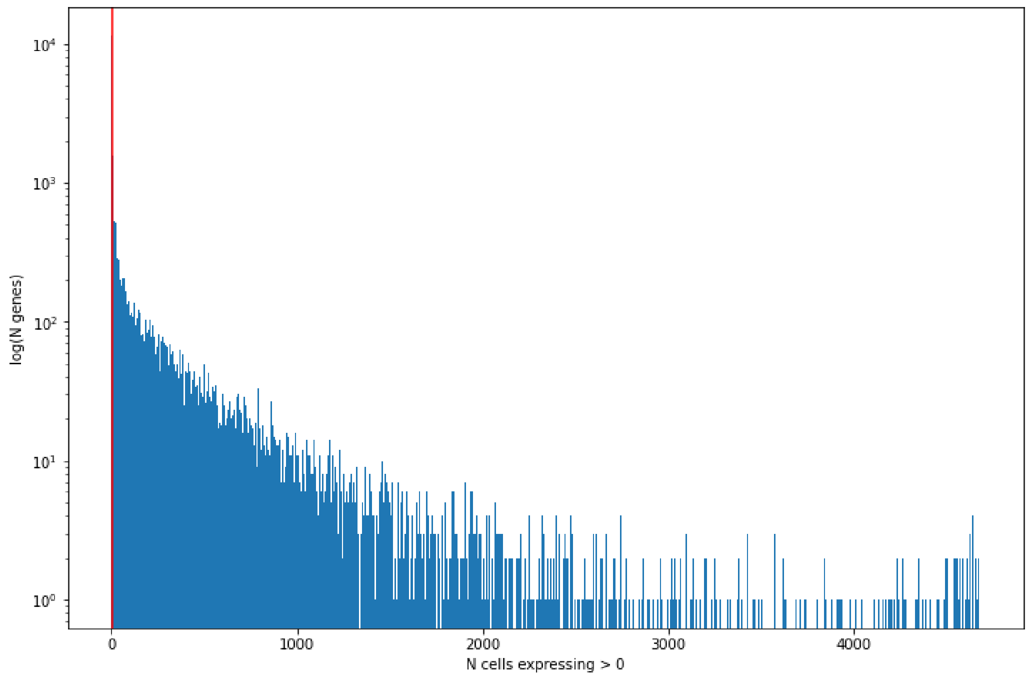
```
Started with:
 AnnData object with n_obs × n_vars = 4677 × 27899
    obs: 'n_counts', 'n_genes'
    var: 'mt'
Finished with:
 View of AnnData object with n_obs × n_vars = 4534 × 27899
    obs: 'n_counts', 'n_genes'
    var: 'mt'
```

Фильтрация генов

Убираем гены, которые экспрессируются мало и в малых количествах клеток

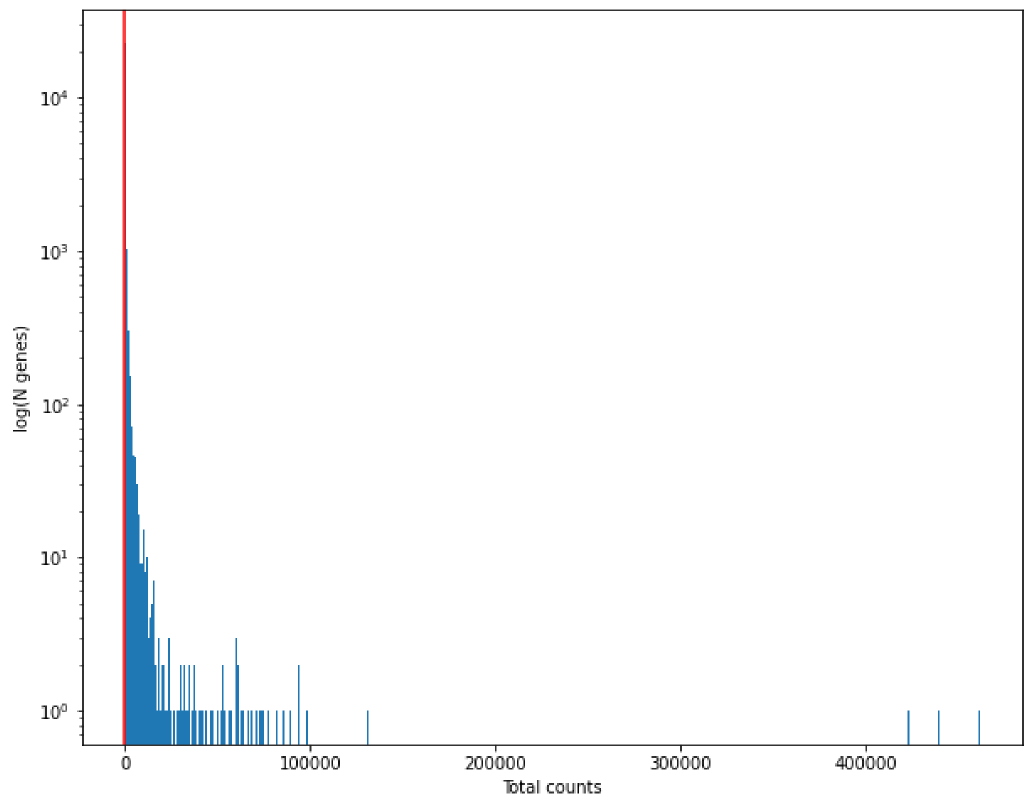
In [ ]:

```
plt.figure(figsize = (12,8))
plt.hist(gene_qc_dataframe['n_cells_by_counts'], bins=1000)
plt.xlabel('N cells expressing > 0')
plt.ylabel('log(N genes)')
plt.axvline(2, color='red')
plt.yscale('log');
```



In [ ]:

```
plt.figure(figsize = (10,8))
plt.hist(gene_qc_dataframe['total_counts'], bins=1000)
plt.xlabel('Total counts')
plt.ylabel('log(N genes)')
plt.yscale('log')
plt.axvline(10, color='red');
```



In [ ]:

```
print('Started with: \n', adata_1)
sc.pp.filter_genes(adata_1, min_cells = 2)
sc.pp.filter_genes(adata_1, min_counts = 10)
print('Finished with: \n', adata_1)
```

```
Started with: 
 View of AnnData object with n_obs × n_vars = 4534 × 27899
    obs: 'n_counts', 'n_genes'
    var: 'mt'
```

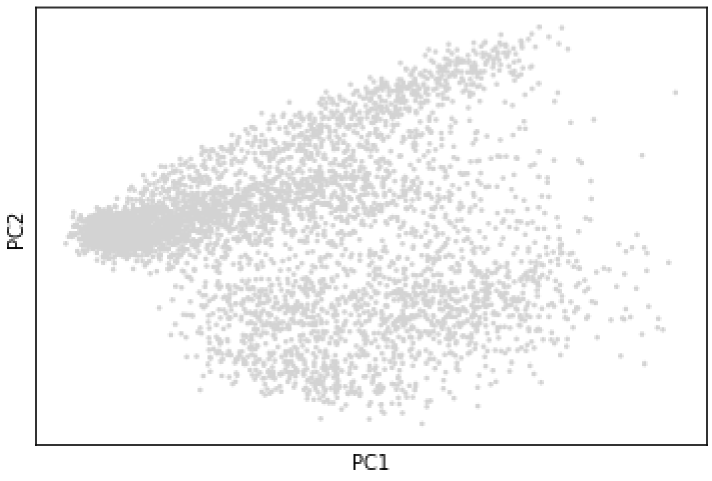
```
Trying to set attribute `.var` of view, copying.
```

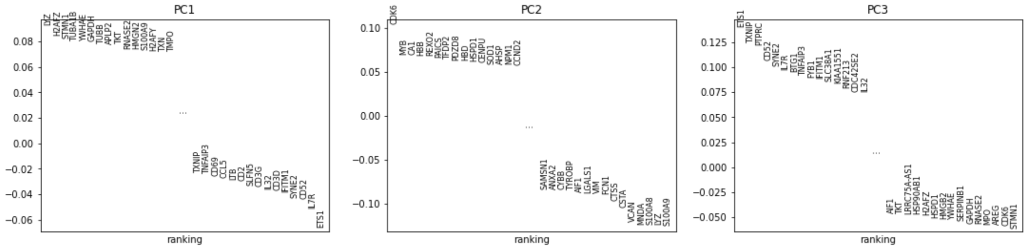
```
Finished with: 
 AnnData object with n_obs × n_vars = 4534 × 15029
    obs: 'n_counts', 'n_genes'
    var: 'mt', 'n_cells', 'n_counts'
```

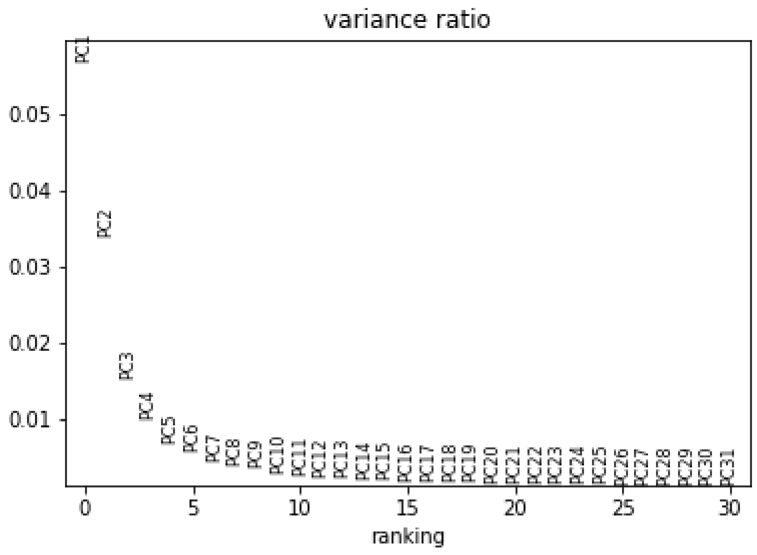
Нормализация и метод главных компонент

In [ ]:

```
adata_1.raw = adata_1
sc.pp.normalize_total(adata_1, target_sum = 1e6, exclude_highly_expressed=
True)
sc.pp.log1p(adata_1)
sc.pp.filter_genes_dispersion(adata_1, n_top_genes = 5000)
sc.pp.pca(adata_1)
sc.pl.pca_overview(adata_1)
```



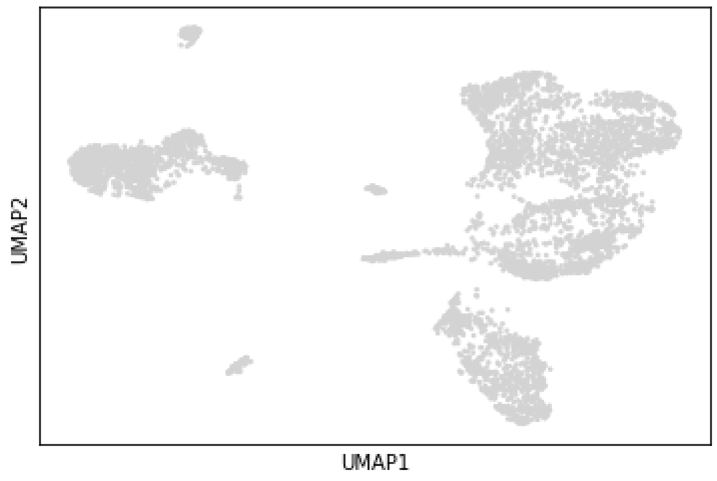




Кластеризация

In [ ]:

```
sc.pp.neighbors(adata_1)
sc.tl.umap(adata_1, min_dist = 0.3, spread = 3,random_state=1, n_component
s=2)
sc.pl.umap(adata_1)
```

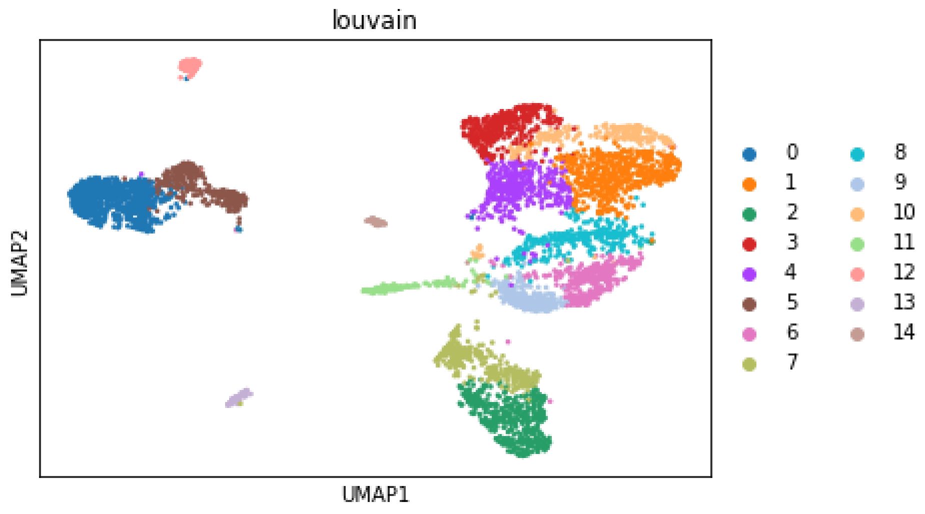


In [ ]:

```
sc.tl.louvain(adata_1)
```

In [ ]:

```
sc.pl.umap(adata_1, color='louvain');
```

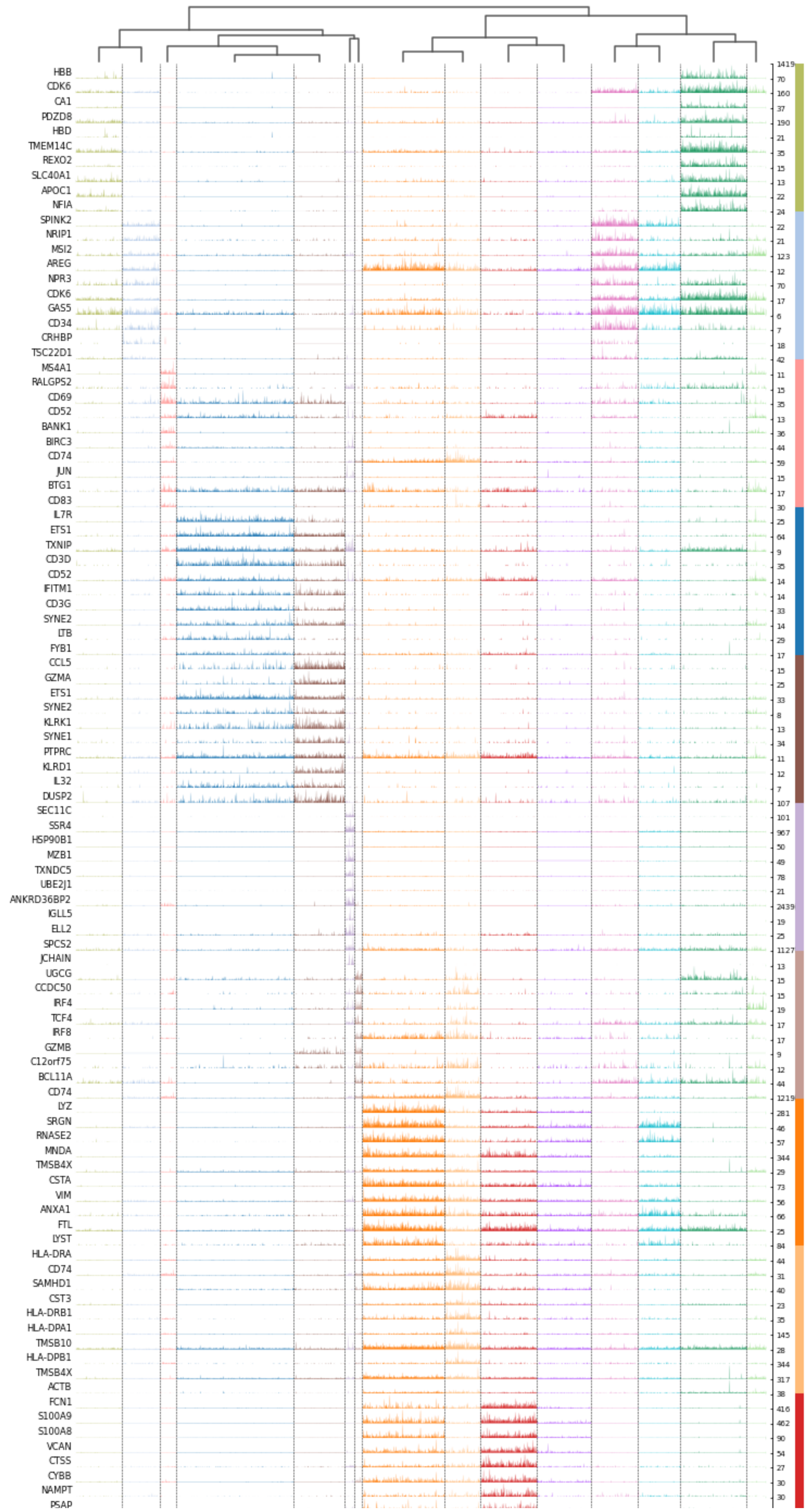


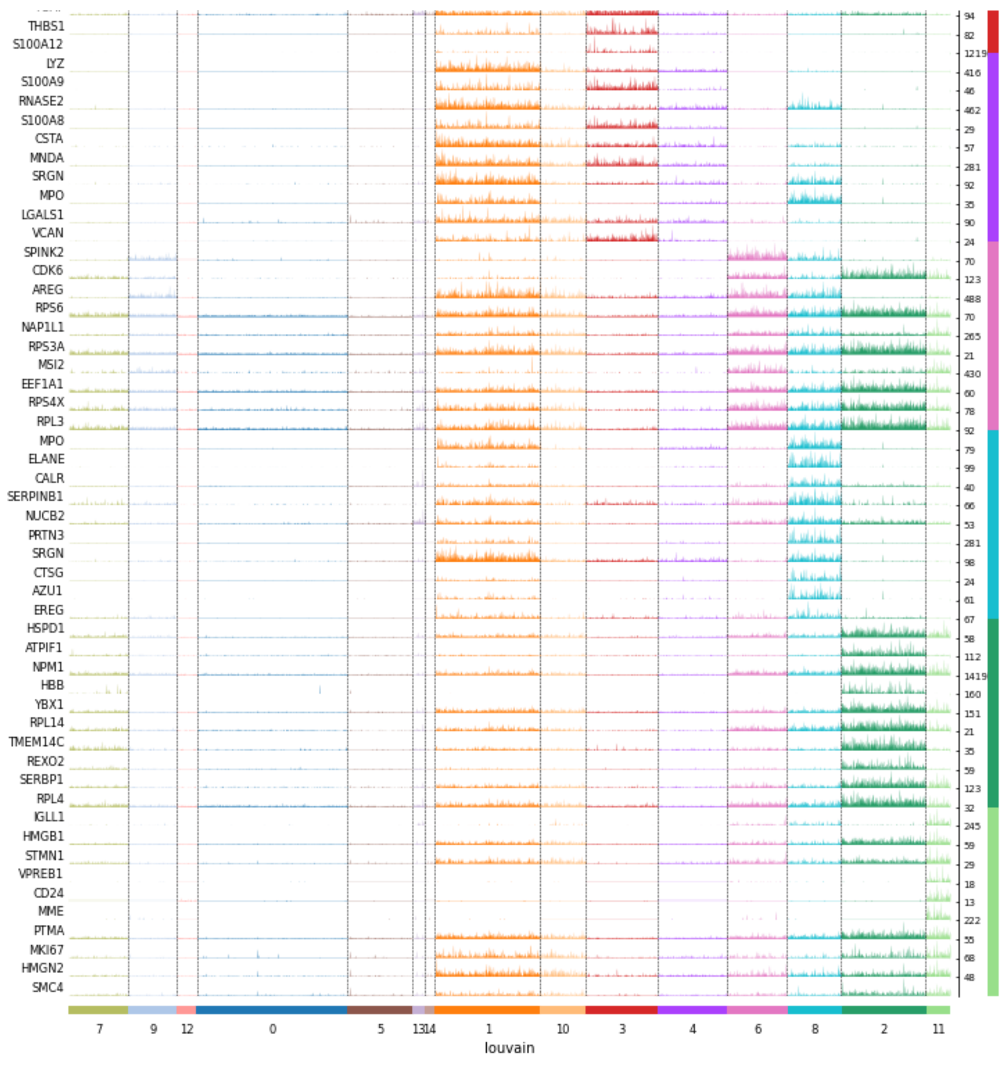
Дифференциальная экспрессия

In [ ]:

```
sc.tl.rank_genes_groups(adata_1, groupby='louvain', use_raw=True, method=
'wilcoxon', n_genes=10)
sc.tl.dendrogram(adata_1, groupby='louvain')
sc.pl.rank_genes_groups_tracksplot(adata_1, groupby='louvain')
```

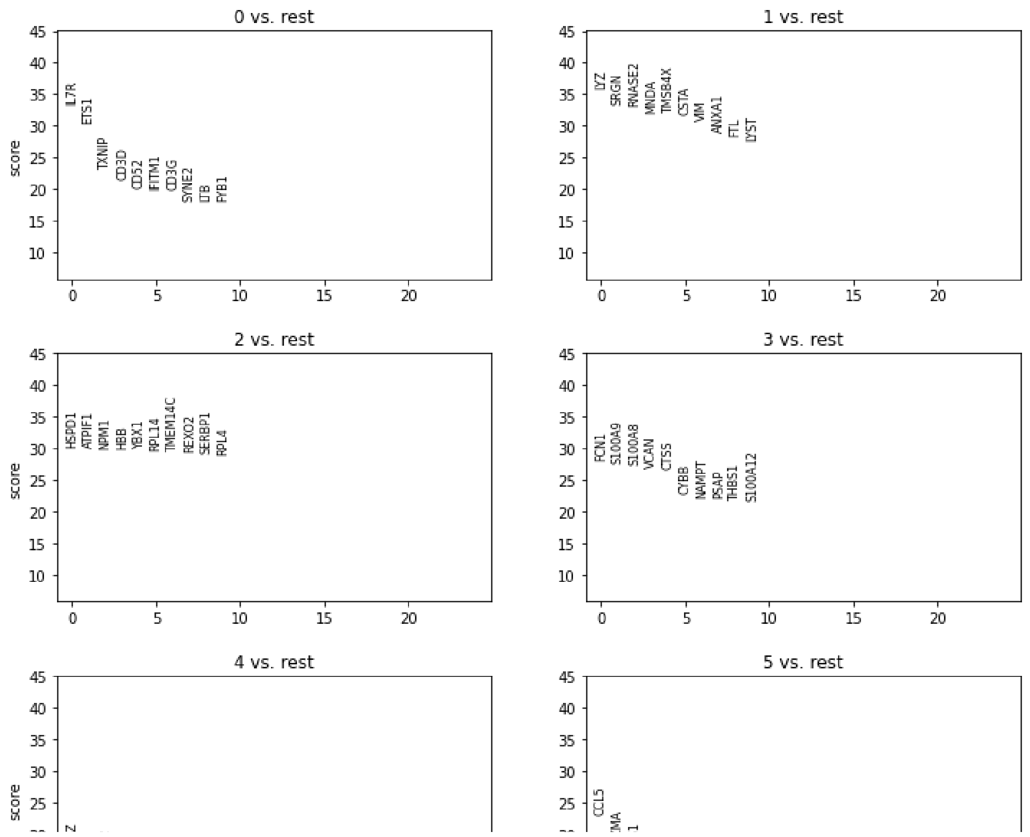
```
WARNING: It seems you use rank_genes_groups on the raw count
data. Please logarithmize your data before calling rank_genes
_groups.
```

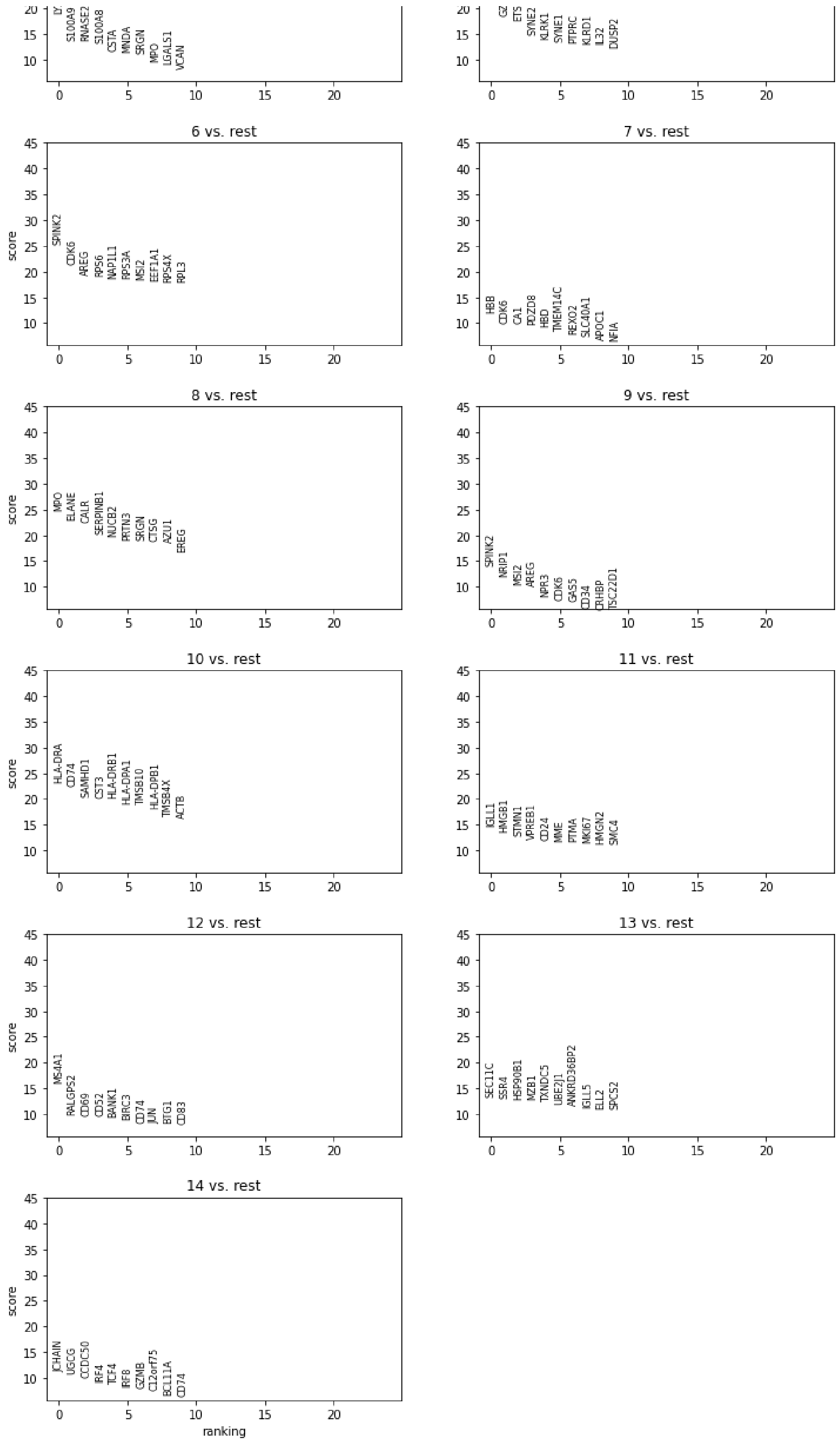




In [ ]:

```
sc.pl.rank_genes_groups(adata_1, n_genes = 25, frontsize = 30, ncols = 2)
```





In [ ]:

```
targets = pd.DataFrame(adata_1.uns['rank_genes_groups']['names'])
for i in targets.columns:
    print(*targets[i].to_numpy(), sep=', ')
```

```
IL7R, ETS1, TXNIP, CD3D, CD52, IFITM1, CD3G, SYNE2, LTB, FYB1
LYZ, SRGN, RNASE2, MNDA, TMSB4X, CSTA, VIM, ANXA1, FTL, LYST
```

```
HSPD1, ATPIF1, NPM1, HBB, YBX1, RPL14, TMEM14C, REXO2, SERBP
1, RPL4
FCN1, S100A9, S100A8, VCAN, CTSS, CYBB, NAMPT, PSAP, THBS1, S
100A12
LYZ, S100A9, RNASE2, S100A8, CSTA, MNDA, SRGN, MPO, LGALS1, V

CAN
CCL5, GZMA, ETS1, SYNE2, KLRK1, SYNE1, PTPRC, KLRD1, IL32, DU
SP2
SPINK2, CDK6, AREG, RPS6, NAP1L1, RPS3A, MSI2, EEF1A1, RPS4X,
RPL3
HBB, CDK6, CA1, PDZD8, HBD, TMEM14C, REXO2, SLC40A1, APOC1, N
FIA
MPO, ELANE, CALR, SERPINB1, NUCB2, PRTN3, SRGN, CTSG, AZU1, E
REG
SPINK2, NRIP1, MSI2, AREG, NPR3, CDK6, GAS5, CD34, CRHBP, TSC
22D1
HLA-DRA, CD74, SAMHD1, CST3, HLA-DRB1, HLA-DPA1, TMSB10, HLA-
DPB1, TMSB4X, ACTB
IGLL1, HMGB1, STMN1, VPREB1, CD24, MME, PTMA, MKI67, HMGN2, S
MC4
MS4A1, RALGPS2, CD69, CD52, BANK1, BIRC3, CD74, JUN, BTG1, CD
83
SEC11C, SSR4, HSP90B1, MZB1, TXNDC5, UBE2J1, ANKRD36BP2, IGLL
5, ELL2, SPCS2
JCHAIN, UGCG, CCDC50, IRF4, TCF4, IRF8, GZMB, C12orf75, BCL11
A, CD74
```

In [ ]:

```
clusters = {
    '0' :'T',
    '1': 'proMono', '2': 'Ery', '3' : 'Mono', '4': 'GMP','5' : 'NK', '6' :
'proGen','7' :'LateEr','8':'GMP','9':'Undiff','10':'Unknown','11':'pro B',
'12':'B','13':'Plazma','14':'pDC'
}
```

In [ ]:

```
adata_1.obs['louvain'].to_csv('louvain.csv')
adata_1.obs['cluster_name'] = adata_1.obs['louvain'].map(clusters)
sc.tl.umap(adata_1, min_dist = 0.3, spread = 3,random_state=1, n_component
s=2)
with rc_context({'figure.figsize': (10, 8)}):
  sc.pl.umap(adata_1, color='cluster_name')
```

```
... storing 'cluster_name' as categorical
```

Анализ

Берем ненормализованную таблицу:

In [ ]:

```
adata_1 = sc.AnnData(X = data)
adata_1.var["mt"] = adata_1.var_names.str.startswith("MT-")
sc.pp.filter_cells(adata_1, min_counts = 1000)
sc.pp.filter_cells(adata_1, min_genes = 500)
high_mt_mask = (cell_qc_dataframe['pct_counts_mt'] < 5)
adata_1 = adata_1[high_mt_mask]
sc.pp.filter_genes(adata_1, min_cells = 2)
sc.pp.filter_genes(adata_1, min_counts = 10)
```

```
Trying to set attribute `.var` of view, copying.
```

Бутстреп

In [ ]:

```
data = pd.DataFrame(adata_1.X, index= adata_1.obs.T.columns)
data.columns = adata_1.var.T.columns
data['target'] = pd.read_csv('louvain.csv', index_col = 'Unnamed: 0')
data
```

Out[ ]:

Gene	A1BG	A1BG-AS1	A2M	A2M-AS1	A4GALT	AAAS	AACS	AADA
BM1_AAAGTCTCAAAC	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
BM1_AAATTTCCATTG	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.
BM1_AAGGTTCCATAA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
BM1_ACACCGATAATG	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
BM1_ACACGTGCGCAA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	.
BM4_GCCCAAATCGCT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
BM4_GTCTCTGTTGTN	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
BM4_TAAACGGTGCCC	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
BM4_TTCGGCAACCAC	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.

BM4_TTCTGCTTGCCT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.(

4534 rows × 15030 columns

```
In [ ]:
```

```python
data = data.T
for i in data.columns:
  data[i] = data[i] / sum(data[i]) * 1000
data = data.T
print(data.sum(axis = 1))
data.head()
```

```
BM1_AAAGTCTCAAAC    1000.0
BM1_AAATTTCCATTG    1000.0
BM1_AAGGTTCCATAA    1000.0
BM1_ACACCGATAATG    1000.0
BM1_ACACGTGCGCAA    1000.0
                     ...
BM4_GCCCAAATCGCT    1000.0
BM4_GTCTCTGTTGTN    1000.0
BM4_TAAACGGTGCCC    1000.0
BM4_TTCGGCAACCAC    1000.0
BM4_TTCTGCTTGCCT    1000.0
Length: 4534, dtype: float64
```

```
Out[ ]:
```

Gene	A1BG	A1BG-AS1	A2M	A2M-AS1	A4GALT	AAAS	AACS	AAI
BM1_AAAGTCTCAAAC	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	
BM1_AAATTTCCATTG	0.0	0.0	0.0	0.0	0.0	0.0	0.156006	
BM1_AAGGTTCCATAA	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	
BM1_ACACCGATAATG	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	
BM1_ACACGTGCGCAA	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	

5 rows × 15030 columns

Теперь каждая клетка экспрессирует тысячу генов. Теперь оставим только таргетные гены.

```
In [ ]:
```

```python
df = pd.DataFrame(index = data.T.columns)
for i in data.columns:
    if i in targets.to_numpy().reshape(75):
        df[i] = data[i]
print(df.shape)
df.head()
```

```
(4534, 65)
```

```
Out[ ]:
```

	AREG	ATPIF1	BANK1	CA1	CALR	CCDC50
BM1_AAAGTCTCAAAC	1.744440	1.308330	0.0	0.000000	0.000000	0.0

BM1_AAATTTCCATTG	0.156006	0.468019	0.0	0.156006	3.588144	0.0
BM1_AAGGTTCCATAA	0.000000	0.829876	0.0	0.000000	0.000000	0.0
BM1_ACACCGATAATG	0.000000	0.000000	0.0	0.000000	0.850340	0.0
BM1_ACACGTGCGCAA	2.259036	0.000000	0.0	0.000000	0.000000	0.0

In [ ]:

```
df['target'] = pd.read_csv('louvain.csv', index_col = 'Unnamed: 0')
df.head() #Добавили таргетную колонку
```

Out[ ]:

	AREG	ATPIF1	BANK1	CA1	CALR	CCDC50
BM1_AAAGTCTCAAAC	1.744440	1.308330	0.0	0.000000	0.000000	0.0
BM1_AAATTTCCATTG	0.156006	0.468019	0.0	0.156006	3.588144	0.0
BM1_AAGGTTCCATAA	0.000000	0.829876	0.0	0.000000	0.000000	0.0
BM1_ACACCGATAATG	0.000000	0.000000	0.0	0.000000	0.850340	0.0
BM1_ACACGTGCGCAA	2.259036	0.000000	0.0	0.000000	0.000000	0.0

In [ ]:

```
df.to_csv('df_with_target_and_genes.csv')
```

In [ ]:

```
df.shape
```

Out[ ]:

```
(4534, 66)
```

In [ ]:

```
bootstrep_boolk = pd.DataFrame(index=np.hstack( (df.columns.to_numpy(),[i
for i in range(15)]) ))

for i in range(300):
  indexes = np.random.randint(0,3636, size = 100) #Генерируем индексы клет
ок, которые создадут 1 бутстрепный элемент

  clusters = [] #Массив кластеров этих 100 клеток
  for j in range(100):
    clusters.append(df.iloc[indexes[j],df.shape[1] - 1])
  clusters = np.array(clusters)

  pc_clusters = [] #Процент кластеров этих 100 клеток
  for k in range(15):
    pc_clusters.append( (clusters == k).sum() / 100)
  pc_clusters = np.array(pc_clusters)


  A = np.zeros(81)
  for l in range(100):
        A += np.hstack((df.T.iloc[:,indexes[l]].to_numpy(),pc_clusters))
    bootstrep_boolk[str(i)] = A
```

Обучим линейную модель

In [ ]:

```
X = bootstrep_boolk.iloc[:65,:].T
y = bootstrep_boolk.iloc[66:,:].T
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
model = Lasso(fit_intercept = True, alpha = 0.5).fit(X_train, y_train)
r2_score(y_test, model.predict(X_test)), r2_score(y_train, model.predict(X
_train))
```

Out[ ]:

```
(0.38558021183075597, 0.5864827334482992)
```

In [ ]:

```
print(max((model.predict(X_test) - y_test).to_numpy().reshape(y_test.shape
[0]*y_test.shape[1])))
```

```
7.1256416192734395
```

In [ ]:

```
model.predict(X_test) - y_test
```

Out[ ]:

	66	67	68	69	70	71	72	
14	-0.007824	-0.085738	2.089140	1.111088	0.578140	-0.699517	-2.675833	-3.
46	2.268810	-4.085256	0.880681	2.002456	1.965519	-0.729529	-1.147983	3.
18	-3.874587	0.748947	2.333868	0.207166	0.923624	0.607173	2.959617	-3.
50	2.958585	-3.989642	-3.896341	1.023341	-1.810861	-2.984264	-1.167095	2.
29	-1.685279	5.432874	-0.960545	-0.123251	0.308773	-0.480248	-2.724697	-0.
17	1.630000	-0.502448	-1.907629	-2.848661	1.075185	0.072296	3.053029	-1.
7	1.149408	-3.545121	3.015269	2.359595	-4.031819	1.497006	2.563355	-3.
136	2.877198	-4.048883	-5.000808	-0.466804	3.276908	-2.754908	-0.669822	5.
170	0.698653	-4.042186	3.204809	2.397990	2.965798	-0.099096	-1.313012	1.
175	-0.647954	4.714391	2.933727	-0.059761	-3.743128	-0.636056	0.322525	-1.
52	0.500930	0.735421	-0.209776	-0.224820	0.663338	2.669524	1.841341	0.
193	3.161746	1.493457	-1.607192	0.922989	-3.540507	2.298066	0.480120	3.
1	-1.382581	3.208964	-1.857082	-1.616445	3.103899	2.070572	2.705725	-3.
161	1.501047	1.769222	-3.087574	3.686078	-4.829787	1.387871	1.988267	3.
129	-0.371192	-4.465625	2.261068	0.687231	1.980078	2.164901	-2.780470	-3.
63	0.105690	-0.003733	-3.472727	-0.826531	-0.031513	-0.725666	-0.416874	3.
2	-2.068931	1.586264	2.600213	0.445102	-0.153828	0.921231	-2.424841	-5.
66	-0.172336	0.148891	-1.269002	-1.438292	4.704403	-1.046229	3.757671	-4.

54	5.422417	2.275363	0.669386	-3.302510	3.858779	-2.874625	-1.610551	-0.
15	3.637624	-0.682817	8.050863	0.326750	3.361358	-2.113984	-2.811756	-4.
42	1.326351	-0.696960	1.064103	0.655181	0.412632	-1.442269	-1.416044	0.
98	-0.239484	5.705065	2.460546	1.072933	-3.914256	1.589618	-2.974576	2.
48	-0.682929	-0.263262	-3.033959	0.820031	-1.791888	2.275251	-2.341734	0.
5	-1.114120	-1.483174	-2.157073	0.577914	2.307325	0.534120	2.321065	-1.
45	-0.022679	-2.338004	0.622786	1.099191	2.533839	-0.205427	3.708245	1.
65	-3.001542	-2.369148	-1.876419	-1.709985	0.894763	-0.660004	0.514835	1.
27	1.496177	0.747612	-1.617967	1.574874	-1.200092	2.301994	-0.431826	0.
96	2.966579	-8.026823	-6.323358	4.557189	-0.631314	0.895513	2.274188	3.
43	-1.144521	3.006824	2.157245	0.728207	-2.235612	0.681893	-3.446386	2.
80	3.274928	0.307563	-5.343194	-1.892587	0.011669	0.424535	2.764873	-2.
196	0.467962	4.018186	4.234526	0.551651	-1.728470	-0.688175	-3.648253	-2.
191	-0.810895	0.500668	0.939290	-0.483106	-0.446692	-3.351155	0.328643	-0.
121	-0.944083	1.556595	-1.109336	-0.657634	1.076721	0.751368	-1.448668	-1.
79	3.068070	-8.604724	3.981920	-0.138420	2.162861	-0.170125	0.228514	-3.
49	-0.373869	1.179742	-0.712617	1.528334	-0.103227	1.458595	-0.241204	-3.
173	-3.911741	1.543459	-0.140365	-0.481092	0.263503	-0.820200	-0.209567	-3.
199	4.323838	0.969204	3.472957	1.740596	0.394223	-1.887970	-1.637031	-0.
73	0.584503	0.981135	-0.784481	1.406494	-3.279363	-2.402460	0.469777	-2
56	5.347074	-1.029697	0.624771	0.118992	3.550794	-2.174094	-1.708966	3.
105	3.523371	-1.465492	-0.948715	-0.347248	0.911284	0.636899	2.456296	-0.

Посмотрим предсказания на настоящем bulk анализе:

In [ ]:

```
bulk = pd.read_table('/content/drive/MyDrive/expressions_leukemia.tsv', se
p = '\t', index_col = 'Gene')
print(bulk.shape)
bulk.head()
```

```
(20062, 137)
```

Out[ ]:

	ohsu_4310	ohsu_4303	ohsu_4299	ohsu_4291	ohsu_4260	ohsu_4252
Gene						
A1BG	0.000000	0.084965	0.000000	0.000000	0.000000	0.000000
A1CF	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
A2M	3.098876	0.035833	0.011140	0.279445	0.133723	0.269452
A2ML1	0.703550	0.638242	0.676287	0.851540	0.691908	0.987448
A3GALT2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

5 rows × 137 columns

Нормализуем и оставим только таргетные гены:

In [ ]:

```
for i in bulk.columns:
    bulk[i] = bulk[i] / sum(bulk[i]) *  100000
bulk.head()
```

Out[ ]:

	ohsu_4310	ohsu_4303	ohsu_4299	ohsu_4291	ohsu_4260	ohsu_4252
Gene						
A1BG	0.000000	0.008496	0.000000	0.000000	0.000000	0.000000
A1CF	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
A2M	0.309888	0.003583	0.001114	0.027944	0.013372	0.026945
A2ML1	0.070355	0.063824	0.067629	0.085154	0.069191	0.098745
A3GALT2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

5 rows × 137 columns

Ого, уже нормализованы! Оставляем только таргетные гены

In [ ]:

```
bulk = bulk.T
for i in bulk.columns:
    if i not in targets.to_numpy().reshape(75):
        bulk = bulk.drop(i, 1)
bulk = bulk.T
print(bulk.shape)
```

```
(65, 137)
```

In [ ]:

```
bulk.to_csv('bulk_target.csv')
```

Видим, что осталось количество таргетных генов присутствующих в датасете, не совпадает с тем, что было у нас при обучении модели. Обучим модель заного, оставим только общие гены, и сделаем предсказание

In [ ]:

```
df = pd.read_csv('df_with_target_and_genes.csv', index_col = 'Unnamed: 0')
df
```

Out[ ]:

	AREG	ATPIF1	BANK1	CA1	CALR	CCDC50
BM1_AAAGTCTCAAAC	1.744440	1.308330	0.0	0.000000	0.000000	0.0

BM1_AAAGTCTCAAAC	1.744440	1.308330	0.0	0.000000	0.000000	0.0
BM1_AAATTTCCATTG	0.156006	0.468019	0.0	0.156006	3.588144	0.0
BM1_AAGGTTCCATAA	0.000000	0.829876	0.0	0.000000	0.000000	0.0
BM1_ACACCGATAATG	0.000000	0.000000	0.0	0.000000	0.850340	0.0
BM1_ACACGTGCGCAA	2.259036	0.000000	0.0	0.000000	0.000000	0.0
...	...	...	...	...	...	...
BM4_GCCCAAATCGCT	0.000000	0.000000	0.0	0.000000	0.000000	0.0
BM4_GTCTCTGTTGTN	4.467610	0.744602	0.0	0.000000	0.000000	0.0
BM4_TAAACGGTGCCC	4.743083	0.000000	0.0	0.000000	0.000000	0.0
BM4_TTCGGCAACCAC	0.000000	0.000000	0.0	0.000000	0.000000	0.0
BM4_TTCTGCTTGCCT	0.000000	0.000000	0.0	0.000000	0.976562	0.0

4534 rows × 66 columns

In [ ]:

```
for i in df.columns:
    if i not in bulk.T.columns:
        df = df.drop(i, 1)
df['target'] = pd.read_csv('louvain.csv', index_col = 'Unnamed: 0')
print(df.shape)
df
```

```
(4534, 66)
```

Out[ ]:

	AREG	ATPIF1	BANK1	CA1	CALR	CCDC50
BM1_AAAGTCTCAAAC	1.744440	1.308330	0.0	0.000000	0.000000	0.0
BM1_AAATTTCCATTG	0.156006	0.468019	0.0	0.156006	3.588144	0.0
BM1_AAGGTTCCATAA	0.000000	0.829876	0.0	0.000000	0.000000	0.0
BM1_ACACCGATAATG	0.000000	0.000000	0.0	0.000000	0.850340	0.0
BM1_ACACGTGCGCAA	2.259036	0.000000	0.0	0.000000	0.000000	0.0
...	...	...	...	...	...	...
BM4_GCCCAAATCGCT	0.000000	0.000000	0.0	0.000000	0.000000	0.0
BM4_GTCTCTGTTGTN	4.467610	0.744602	0.0	0.000000	0.000000	0.0
BM4_TAAACGGTGCCC	4.743083	0.000000	0.0	0.000000	0.000000	0.0
BM4_TTCGGCAACCAC	0.000000	0.000000	0.0	0.000000	0.000000	0.0
BM4_TTCTGCTTGCCT	0.000000	0.000000	0.0	0.000000	0.976562	0.0

4534 rows × 66 columns

In [ ]:

```
bootstrep_boolk = pd.DataFrame()

for i in range(200):
  indexes = np.random.randint(0,3636, size = 100) #Генерируем индексы клет
ок, которые создадут 1 бутстрепный элемент

  clusters = [] #Массив кластеров этих 100 клеток
  for j in range(100):
    clusters.append(df.iloc[indexes[j],65])
  clusters = np.array(clusters)

  pc_clusters = [] #Процент кластеров этих 100 клеток
  for k in range(15):
    pc_clusters.append( (clusters == k).sum() / 100)
  pc_clusters = np.array(pc_clusters)


  A = np.zeros(81)
  for l in range(100):
     A += np.hstack((df.T.iloc[:,indexes[l]].to_numpy(),pc_clusters))

  bootstrep_boolk[i] = A
```

In [ ]:

```
X = bootstrep_boolk.iloc[:65,:].T
y = bootstrep_boolk.iloc[66:,:].T
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
model = LinearRegression(fit_intercept=False, positive=True)
model.fit(X_train, y_train)
r2_score(y_test, model.predict(X_test))
```

Out[ ]:

```
0.4080717686808072
```

In [ ]:

```
model_predict = model.predict(bulk.T);
model_predict.shape
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:439: U
serWarning: X has feature names, but LinearRegression was fit
ted without feature names
  f"X has feature names, but {self.__class__.__name__} was fi
tted without"
```

Out[ ]:

```
(137, 15)
```

In [ ]:

```
for i in range(137): #нормализуем
  model_predict[i] = model_predict[i] / model_predict[i].sum()
```

Теперь добавим эти данные к аннотации, чтобы потом пытаться предсказать время смерти:

In [ ]:

```
death = pd.read_table('annotation_leukemia.tsv', sep = '\t', index_col =
'ID')
death.shape
```

```
for i in range(15):
  death[i] = model_predict[:,i]
```

In [ ]:

```
death.to_csv('death.csv')
death.head()
```

Out[ ]:

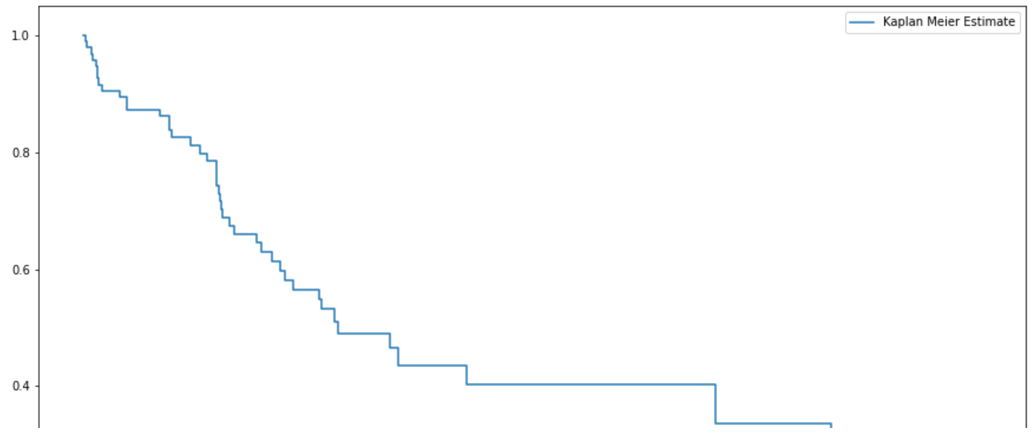
	Tissue	Sample Timepoint	OS	OS_FLAG	Cause of death	Age	Gender	Ethnicity
ID								
ohsu_635	BM	Denovo	697.8	1.0	Dead-Disease	55.0	F	White
ohsu_764	BM	Denovo	1377.6	0.0	Alive	50.0	F	White
ohsu_923	BM	Denovo	NaN	NaN	NaN	61.0	M	White
ohsu_50	BM	Relapse	1633.2	1.0	Dead-Disease	11.0	M	White
ohsu_29	BM	NaN	587.1	1.0	Dead-Unknown	34.0	M	White

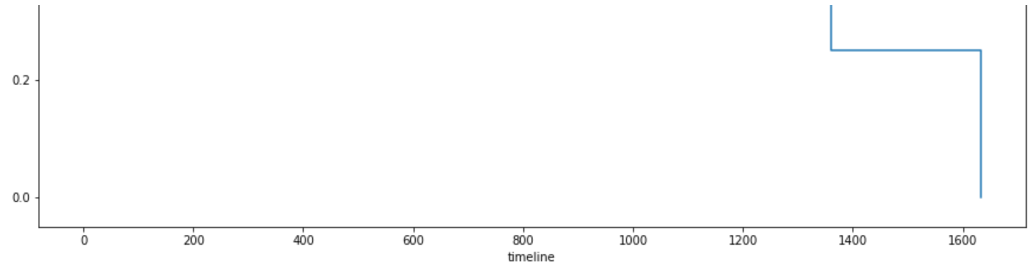
Предсказываем смерть

Заполним пропуски линейной моделью и обучим KaplanMeierFitter:

In [ ]:

```
death.fillna(np.nan)
death = death.dropna(axis='index', how='any')

kmf = KaplanMeierFitter()

kmf.fit(death['OS'], death['OS_FLAG'],label='Kaplan Meier Estimate')

plt.figure(figsize = (15,10))
kmf.plot(ci_show=False);
```



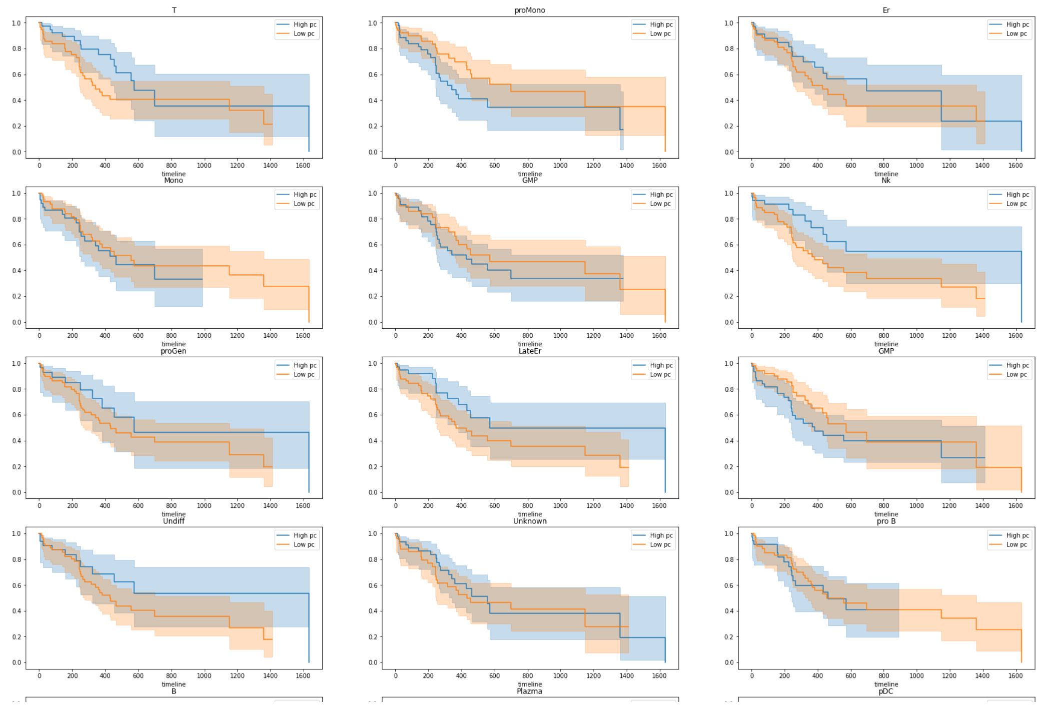


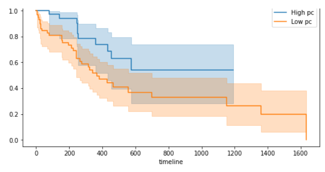
In [ ]:

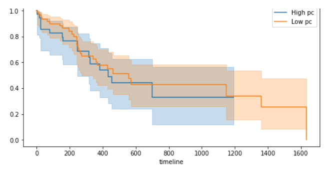
```
clusters = {
    '0' :'T',
    '1': 'proMono', '2': 'Ery', '3' : 'Mono', '4': 'GMP','5' : 'NK', '6' : 'proGen','7' :'LateEr','8':'GMP','9':'Undiff','10':'Unknown','11':'pro B','12':'B','13':'Plazma','14':'pDC'
}
```

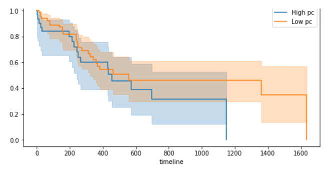
In [ ]:

```
means = [np.mean(death[i]) for i in range(15)]
plt.figure(figsize = (30,25))

for i in range(15):

  data_high = death[death[i] > means[i]]
  data_low = death[death[i] < means[i]]
  kmf_high = KaplanMeierFitter()
  kmf_high.fit(data_high['OS'], data_high['OS_FLAG'],label='Kaplan Meier Estimate')
  kmf_low = KaplanMeierFitter()
  kmf_low.fit(data_low['OS'], data_low['OS_FLAG'],label='Kaplan Meier Estimate')


  plt.subplot(5,3,i+1)
  plt.title(clusters[i])
  kmf_high.plot(label = 'High pc')
  kmf_low.plot(label = 'Low pc')
```




High pc
Low pc
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000
1200
1400
1600
timeline


High pc
Low pc
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000
1200
1400
1600
timeline


High pc
Low pc
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000
1200
1400
1600
timeline