In [18]:

```python
# Environment setup; succesful loading here should mean all dependencies correctly installed #
import numpy as np
import scanpy as sc
import anndata as anndata
from scipy import  stats
import seaborn as sbn
import matplotlib.pyplot as plt
```

Load in the dataset

In [19]:

```python
pbmc = sc.read_h5ad('pbmc.h5ad')
print(f'The number of cells are: {pbmc.shape[0]}')
print(f'The number of genes are: {pbmc.shape[1]}')
```

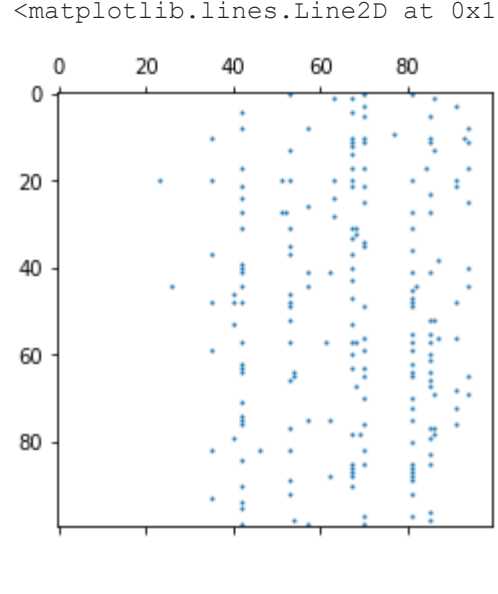
```
The number of cells are: 22475
The number of genes are: 32738
```

Check coverage

In [20]:

```python
plt.spy(pbmc.X[:100, :100], markersize=1)
```

Out[20]: `<matplotlib.lines.Line2D at 0x1f321e48488>`



QC Metrics

In [21]:

```python
#inplace saves all of the metrics to pbmc.obs
sc.pp.calculate_qc_metrics(pbmc, inplace=True)
```

In [22]:

```python
pbmc.obs[0:5]
```

Out[22]:

	cell-types	n_genes_by_counts	log1p_n_genes_by_counts	total_counts	log1p_total_counts	pct_counts_in_top_50_genes	pct_cc
AAACATACACCCAA-1	CD8+ Cytotoxic T	498	6.212606	1216.0	7.104144	44.572368	
AAACATACTTCTAC-1	CD8+ Cytotoxic T	925	6.830874	2683.0	7.895063	39.619829	
AAACATTGCTTCGC-1	CD8+ Cytotoxic T	554	6.318968	1310.0	7.178545	43.358779	
AAACCGTGCGATAC-1	CD8+ Cytotoxic T	687	6.533789	1854.0	7.525640	44.983819	
AAACCGTGTCACCC-1	CD8+ Cytotoxic T	471	6.156979	1274.0	7.150702	49.450549	

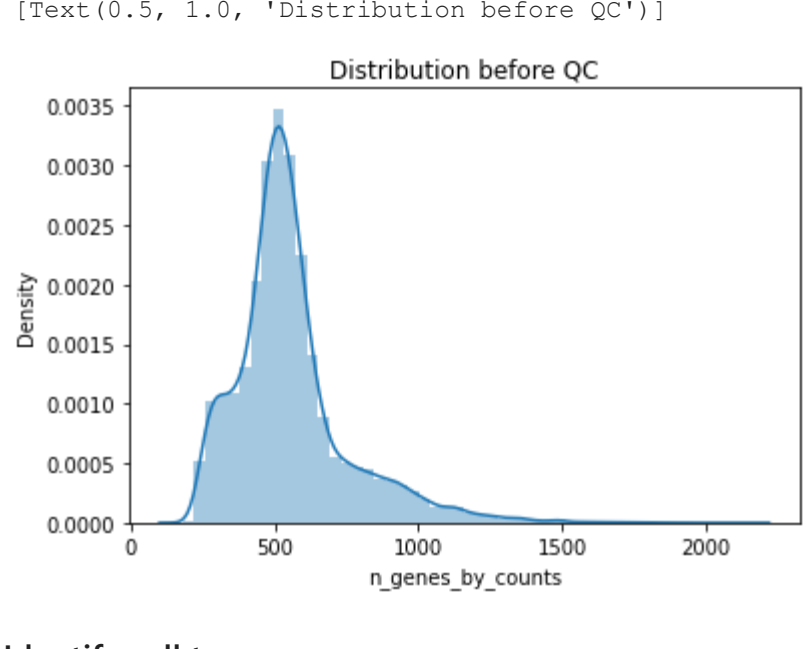
Plot the distribution before QC

In [23]:

```python
sbn.distplot(pbmc.obs['n_genes_by_counts']).set(title='Distribution before QC')
```

```
c:\Python37\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in 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 histograms).
  warnings.warn(msg, FutureWarning)
```

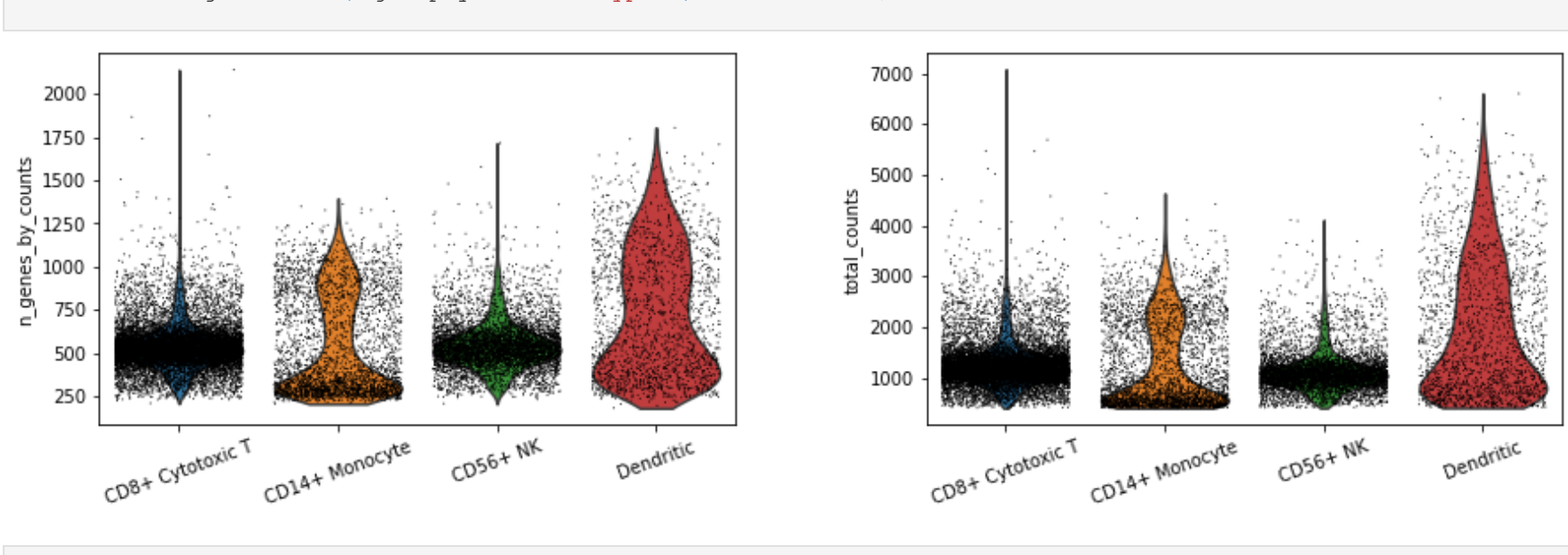
Out[23]: `[Text(0.5, 1.0, 'Distribution before QC')]`



Identify cell types

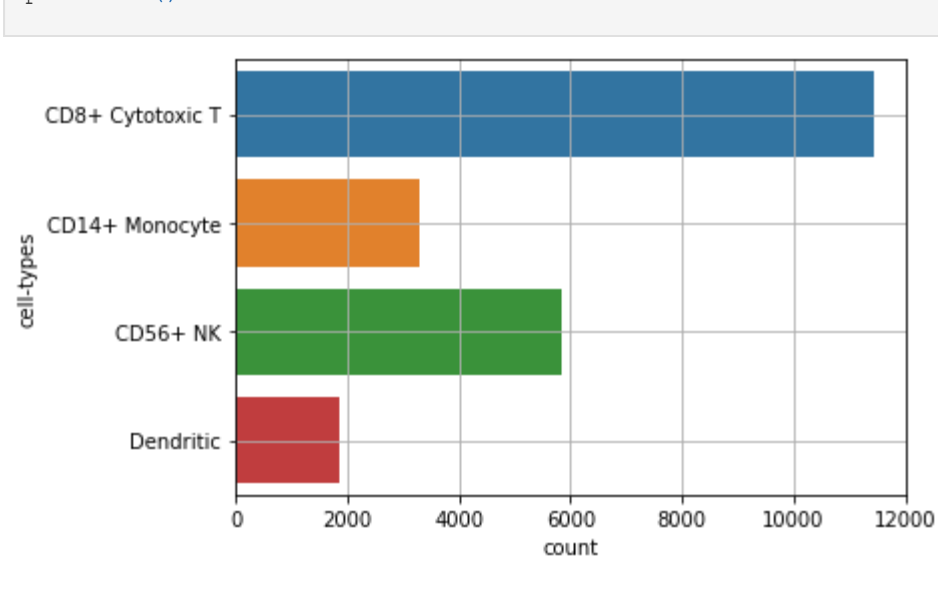
In [24]:

```python
sc.pl.violin(pbmc, ['n_genes_by_counts', 'total_counts'],
             jitter=0.4, groupby = 'cell-types', rotation= 20)
```



In [25]:

```python
p =sbn.countplot(data=pbmc.obs, y='cell-types')
plt.grid()
plt.show()
```



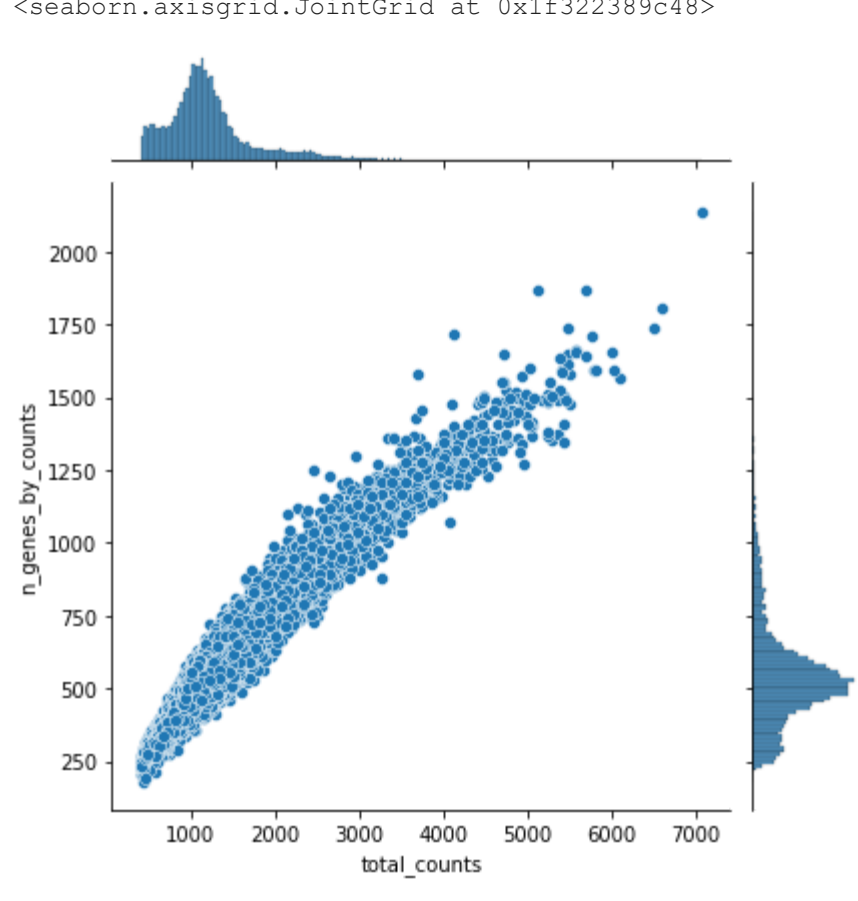
Gene coverage as read counts increase

This plot gives us an indication of how we'll we're sampling genes clearly the number of genes is underrepresented but the data is beggining to platau as total read counts increases which is a good sign

In [26]:

```python
sbn.jointplot(data=pbmc.obs, x='total_counts', y='n_genes_by_counts')
```

Out[26]: `<seaborn.axisgrid.JointGrid at 0x1f322389c48>`



Removing cells with too many or too few genes

Using a metric like Median absolute deviation we can remove outliers that are expressing too many or too few genes per cell

In [27]:

```python
total_genes_per_cell = pbmc.obs['n_genes_by_counts']

median = np.median(total_genes_per_cell)


abs_deviation = abs(total_genes_per_cell - median)

MAD = np.median(abs_deviation)

lower_bound = median - MAD*3
upper_bound = median + MAD*3

outliers = len([x for x in total_genes_per_cell \
                if x < lower_bound or x > upper_bound])

print('Original number of cells: ' + str(len(total_genes_per_cell)))
print('Median #genes : ' + str(median))
print('MAD: ' + str(MAD))
print('Upper bound: ' + str(upper_bound))
print('Lower bound: ' + str(lower_bound))
print('Number of outliers: ' + str(outliers))
```

```
Original number of cells: 22475
Median #genes : 524.0
MAD: 86.0
Upper bound: 782.0
Lower bound: 266.0
Number of outliers: 3277
```

In [28]:

```python
#Filter out these outliers with scanpy

sc.pp.filter_cells(pbmc, max_genes = upper_bound)
sc.pp.filter_cells(pbmc, min_genes = lower_bound)

print(f'Number of cells after filtering: {pbmc.shape[0]}')
```

```
Number of cells after filtering: 19198
```

Remove genes that are not present in enough cells

We don't want to inflate our data with genes that either don't express at all or aren't in at least 3 cells

In [29]:

```python
print("Genes before filtering: ", pbmc.shape[1])

sc.pp.filter_genes(pbmc, min_cells=3)

print("Genes after filtering: ", pbmc.shape[1])
```

```
Genes before filtering:  32738
Genes after filtering:  14945
```

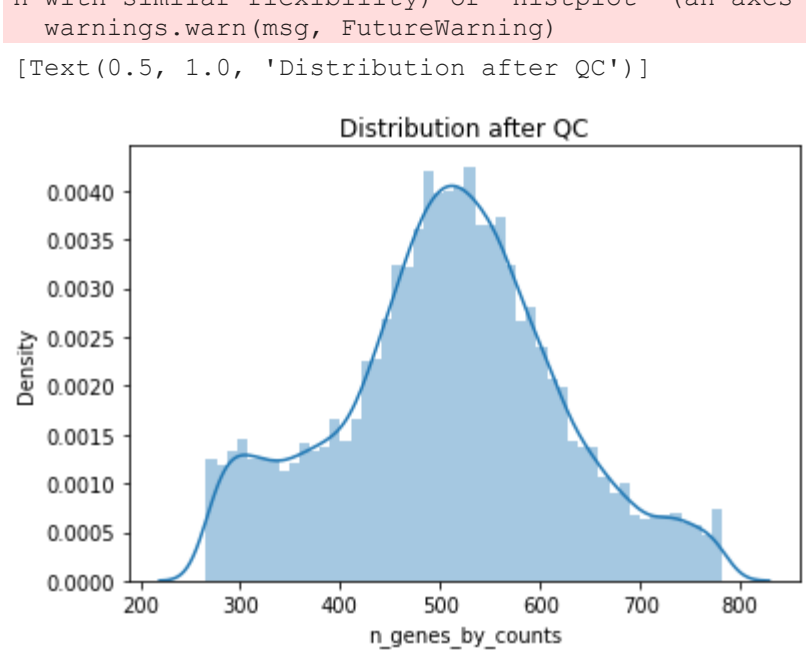
Visualise the changes

In [30]:

```python
sbn.distplot(pbmc.obs['n_genes_by_counts']).set(title='Distribution after QC')
```

```
c:\Python37\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in 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 histograms).
  warnings.warn(msg, FutureWarning)
```

Out[30]: `[Text(0.5, 1.0, 'Distribution after QC')]`



In [31]:

```python
pbmc.layers['counts'] = pbmc.X.copy()
```

There are several ways to normalise scRNAseq - lets take a look at the differences based on their distributions

Perform normalisation using calculated library size

In [32]:

```python
og_data = pbmc.X.toarray()

#now we now how many reads each cell has in total
reads_per_cell = og_data.sum(axis=1)

total_cells = pbmc.shape[0]

#total reads across all ~19000 cells
total_library_size = sum(pbmc.obs['total_counts'])

#Calculate the size factor for library size normalisation
size_factors = (reads_per_cell * total_cells) / total_library_size

norm_reads = np.apply_along_axis(np.divide, 0, og_data, size_factors)

#####THIS LINE IS WHAT MAKES OUR DATA FOLLOW THIS NORMALISATION
pbmc.X = norm_reads
```

Example of 10,000 counts per cell normalisation

In [33]:

```python
X = pbmc.layers['counts'].toarray()


reads_per_cells = X.sum(axis=1)
after = 1e4
scale_factors = reads_per_cells / after   # scale factor for each cell

# Calculating 'counts-per-ten-thousand' normalisation between cells.
x_counts_norm = np.apply_along_axis(np.divide, 0, X, scale_factors)

reads_per_cells_norm = x_counts_norm.sum(axis=1)

#for constant library size
pbmc.layers['constant_ls'] = x_counts_norm

stats.describe(reads_per_cells_norm) # confirm that most cells have total reads ~1e4
```

Out[33]:

```
DescribeResult(nobs=19198, minmax=(9999.913, 10000.092), mean=10000.0, variance=0.0005933718, skewness=-0.06664492934942245, kurtosis=-0.23289474552878398)
```

Normalisation by median

In [34]:

```python
X = pbmc.layers['counts'].toarray()

reads_per_cells = X.sum(axis=1)
median = np.median(reads_per_cell)
print(median)
scale_factors = reads_per_cells / median   # scale factor for each cell

# Calculating 'counts-per-ten-thousand' normalisation between cells.
x_counts_norm = np.apply_along_axis(np.divide, 0, X, scale_factors)

reads_per_cells_norm = x_counts_norm.sum(axis=1)

#for constant library size
pbmc.layers['median_norm'] = x_counts_norm

stats.describe(reads_per_cells_norm) # confirm that most cells have total reads ~1e4
```

```
1105.0
```

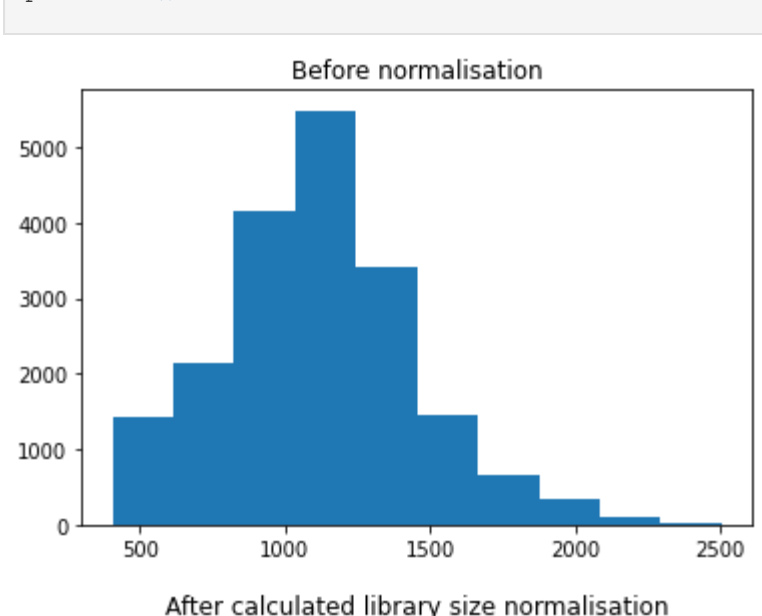
Out[34]:

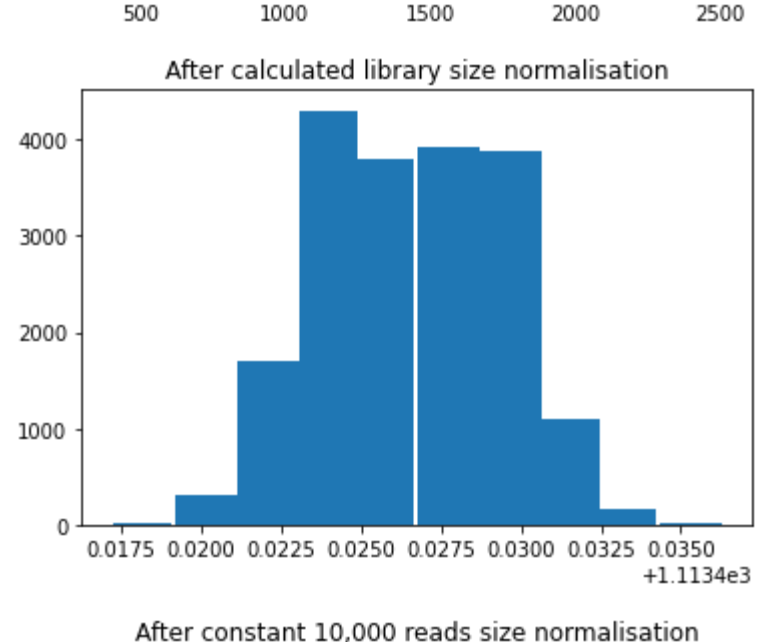
```
DescribeResult(nobs=19198, minmax=(1104.9912, 1105.0095), mean=1105.0, variance=7.908801e-06, skewness=0.06783406436443329, kurtosis=-0.5836937236907622)
```

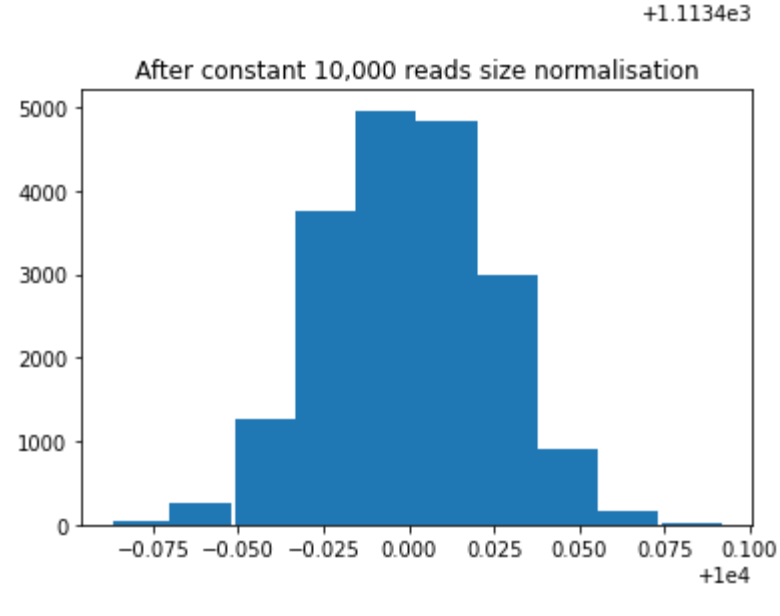
Now we can compare normalisation methods

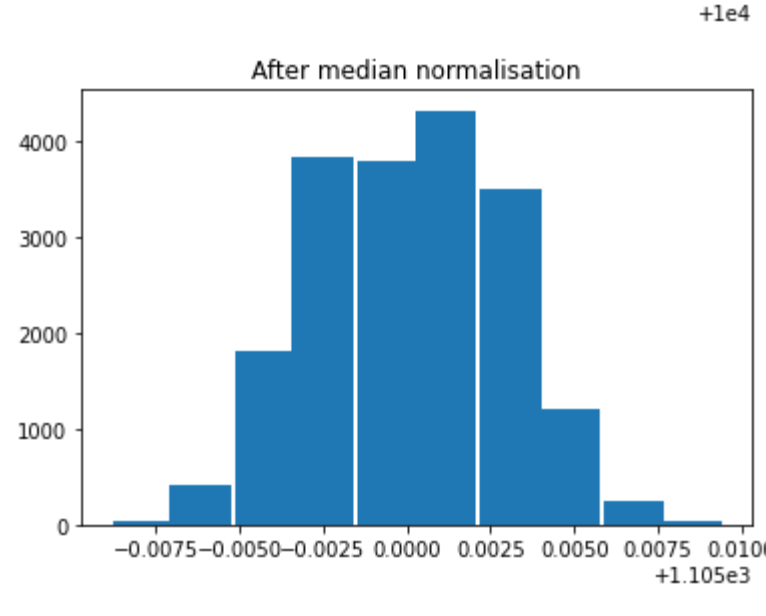
In [35]:

```python
plt.hist(pbmc.layers['counts'].sum(axis=1))
plt.title('Before normalisation')
plt.show()

plt.hist(pbmc.X.sum(axis=1))
plt.title('After calculated library size normalisation')
plt.show()

plt.hist(pbmc.layers['constant_ls'].sum(axis=1))
plt.title('After constant 10,000 reads size normalisation')
plt.show()

plt.hist(pbmc.layers['median_norm'].sum(axis=1))
plt.title('After median normalisation')
plt.show()
```









Log-transformation

We do this to reduce difference between genes that are expressing at very different levels so that we can do our feature selection.

In [36]:

```python
sc.pp.log1p(pbmc)
```

Feature selection

In [37]:

```python
#all of the variables I selected are just the
#default values from the docs

#Applys an annotation of variability so that we can plot it
sc.pp.highly_variable_genes(pbmc, min_mean=0.0125, max_mean=3, min_disp=0.5)
```

In [38]:

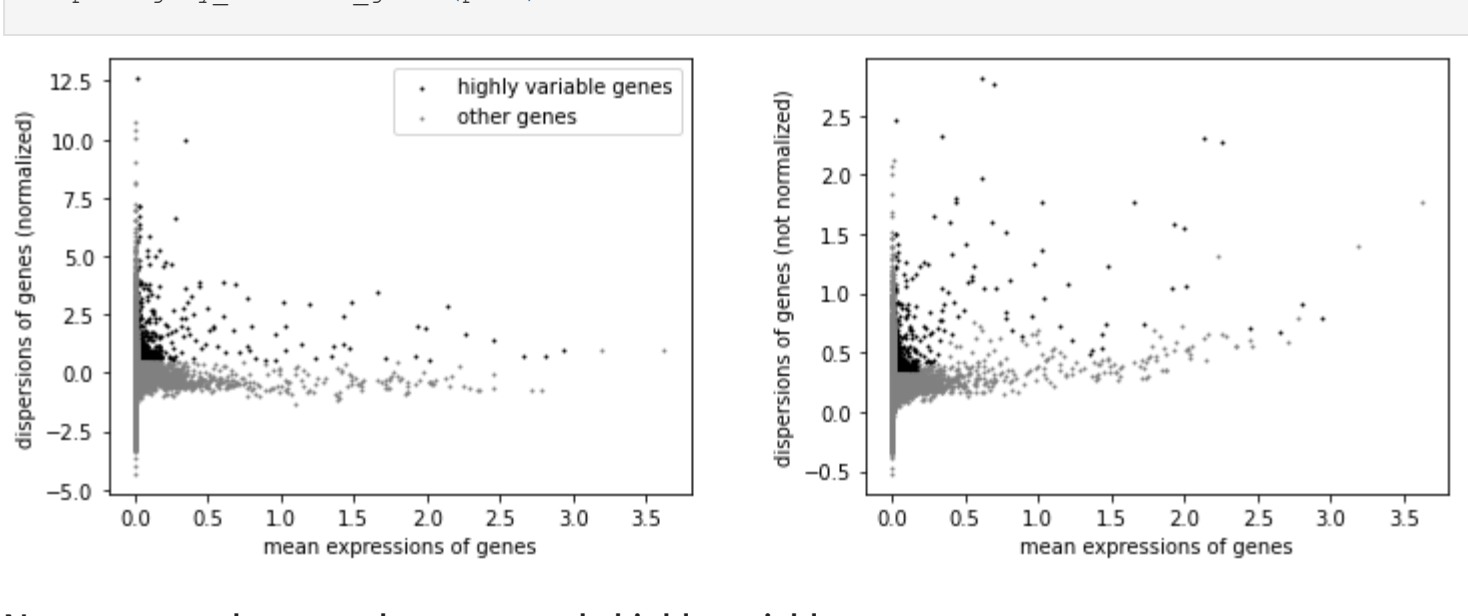
```python
genes = pbmc.var['highly_variable']

variable_genes = genes[genes == True]

print(f'The number of variable genes are {len(variable_genes)}')
```

```
The number of variable genes are 903
```

Some cool plots showing the mean expression and variability of genes

In [39]:

```python
sc.pl.highly_variable_genes(pbmc)
```



Now we can subset our data set to only highly variable genes

In [40]:

```python
sc.pp.highly_variable_genes(pbmc, min_mean=0.0125, max_mean=3, min_disp=0.5, subset=True)
```

In [41]:

```python
print(f'The number of cells are: {pbmc.shape[0]}')
print(f'The number of genes are: {pbmc.shape[1]}')
```

```
The number of cells are: 19198
The number of genes are: 903
```

Visualisation

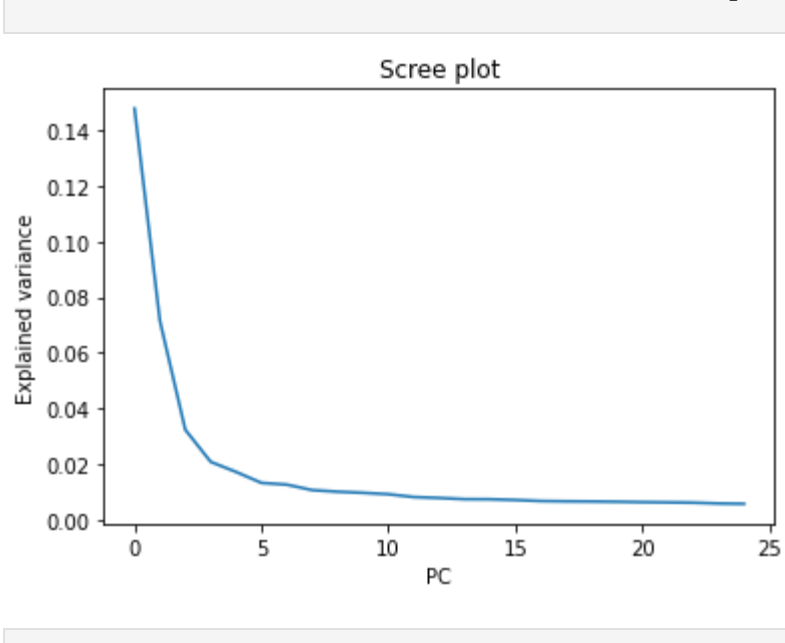
PCA analysis

In [42]:

```python
sc.tl.pca(pbmc, svd_solver='arpack')
```

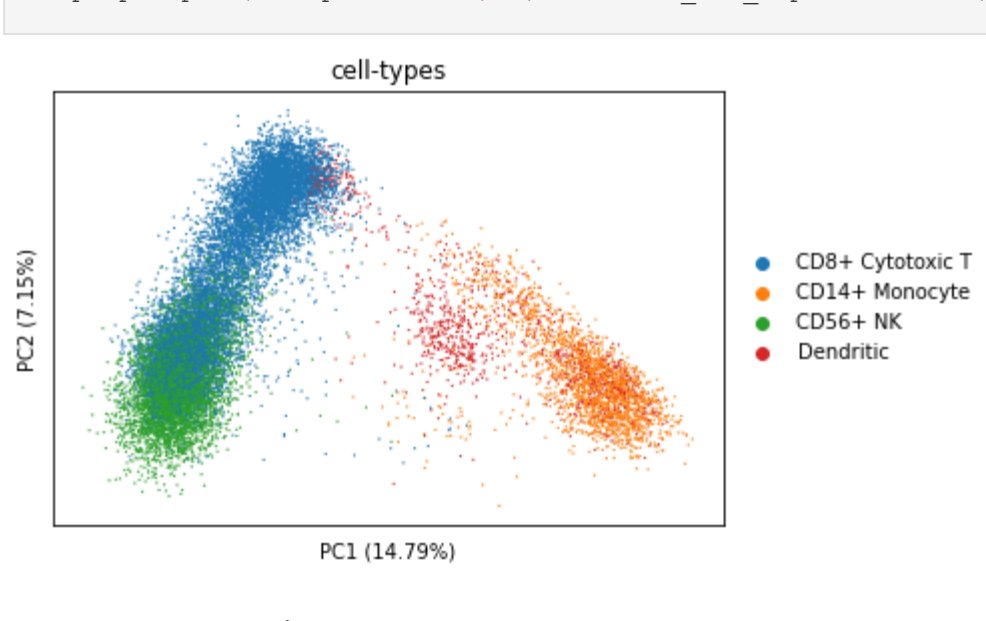
In [43]:

```python
explained_variance = pbmc.uns['pca']['variance_ratio'][:25]

p = sbn.lineplot(data=explained_variance).set(title='Scree plot', \
                                               xlabel='PC',
                                               ylabel='Explained variance')
```



In [44]:

```python
sc.pl.pca(pbmc, components='1,2', annotate_var_explained=True, color='cell-types')
```



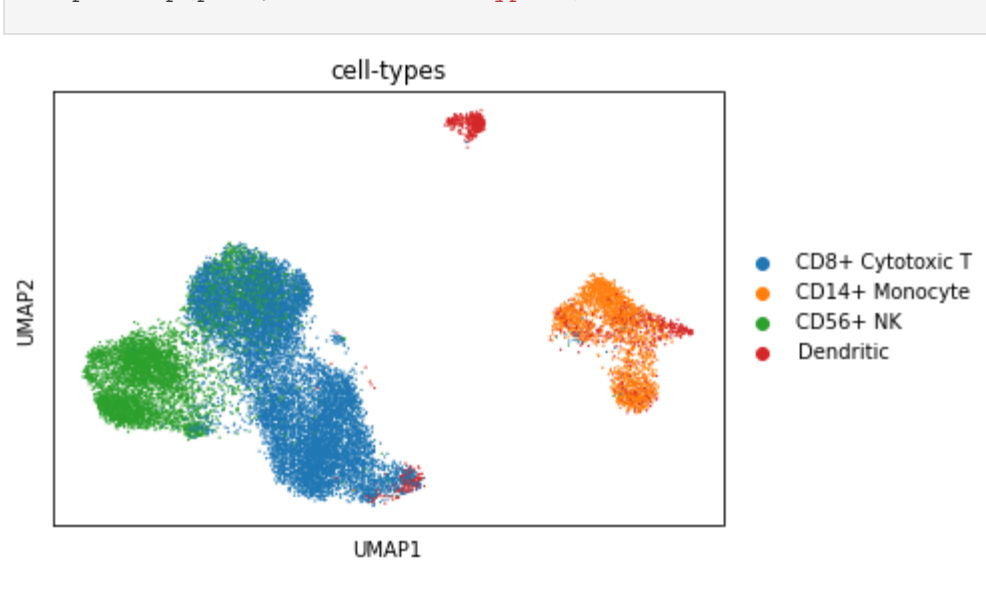
UMAP representation

Just experimenting with different cluster sizes here, can check the appropriate number of cluster neighbors

30 neighbors

In [45]:

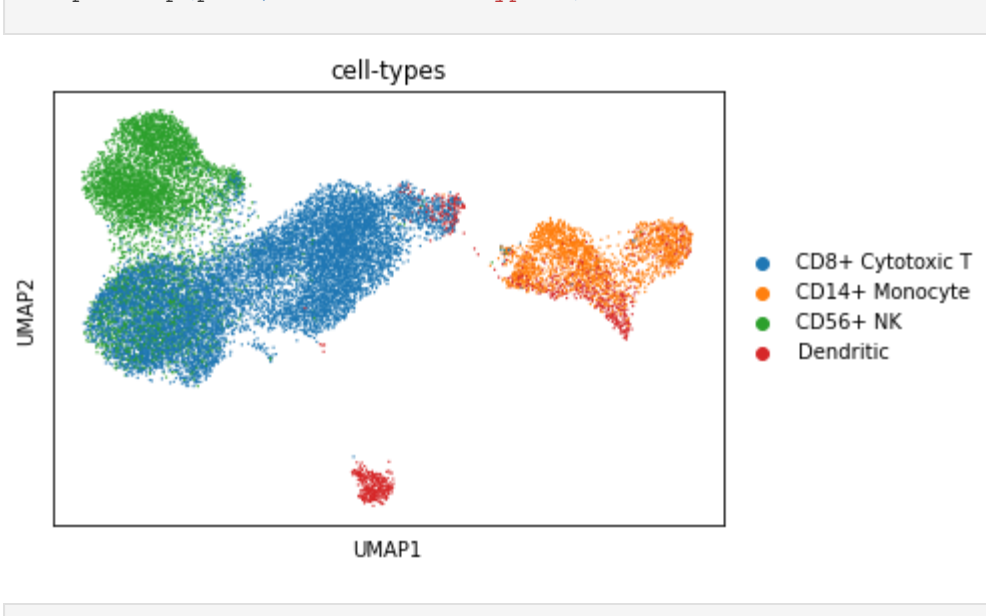
```python
#neighbours is just a function that computes
#pairwise distances between points for UMAP
sc.pp.neighbors(pbmc, n_neighbors=30, n_pcs=40)

#now with our distances, we can create the UMAP plot
sc.tl.umap(pbmc)

sc.pl.umap(pbmc, color='cell-types')
```



15 Neighbours

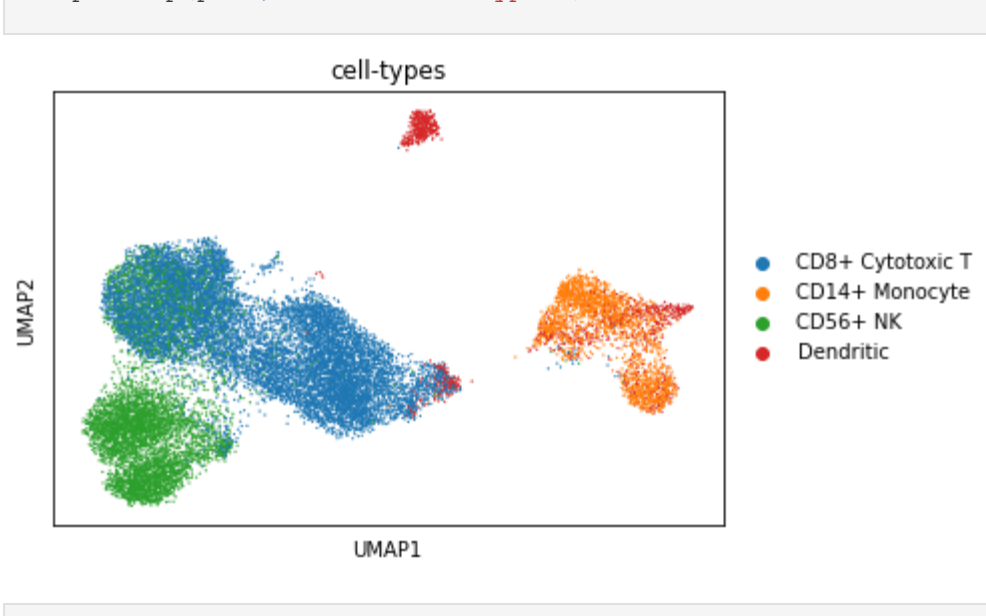
In [46]:

```python
sc.pp.neighbors(pbmc, n_neighbors=15, n_pcs=13)

#now with our distances, we can create the UMAP plot
sc.tl.umap(pbmc)

sc.pl.umap(pbmc, color='cell-types')
```



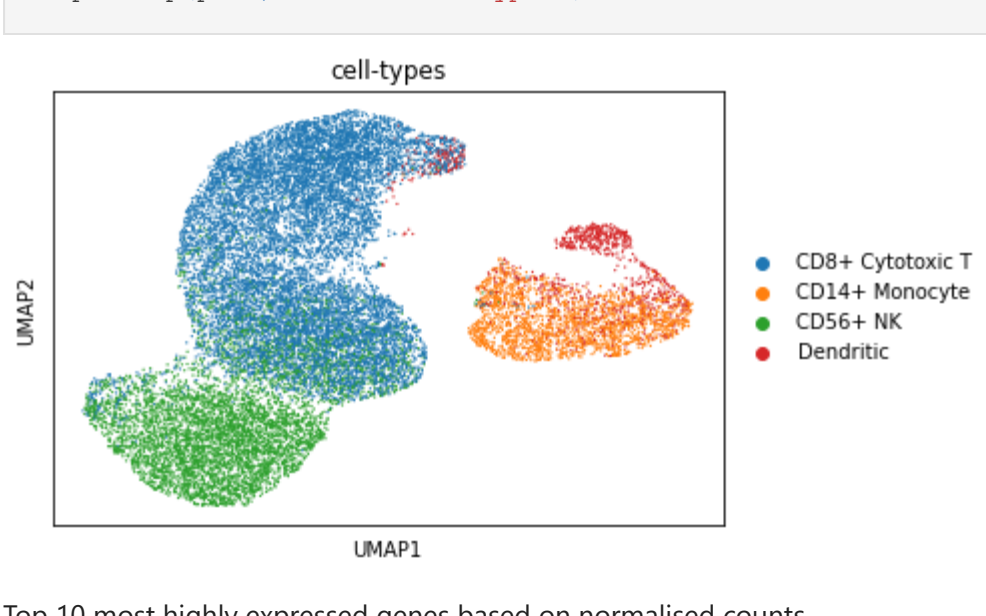
In [47]:

```python
sc.pp.neighbors(pbmc, n_neighbors=15, n_pcs=20)

#now with our distances, we can create the UMAP plot
sc.tl.umap(pbmc)

sc.pl.umap(pbmc, color='cell-types')
```



In [77]:

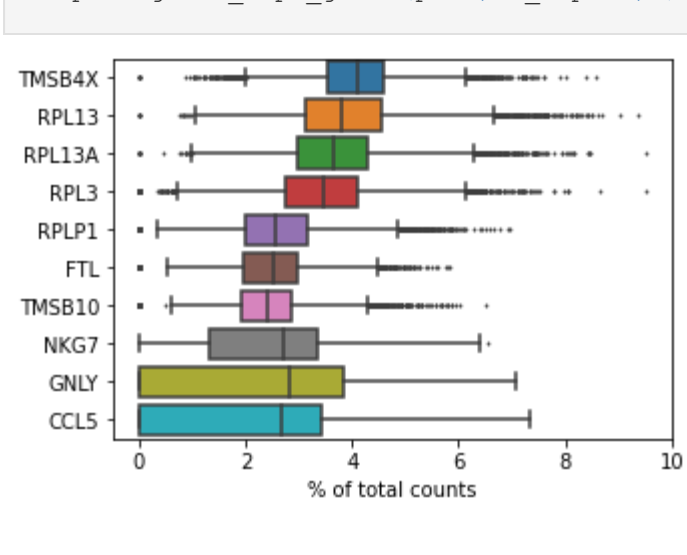
```python
#now lets checkk the UMAP based on the elbow of the scree plot, which showed around PCAs = 4

sc.pp.neighbors(pbmc, n_neighbors=15, n_pcs=4)


sc.tl.umap(pbmc)

sc.pl.umap(pbmc, color='cell-types')
```



Top 10 most highly expressed genes based on normalised counts

In [65]:

```python
sc.pl.highest_expr_genes(pbmc, n_top=10, )
```



Machine Learning

Using an autoencoder architecture with Stochastic gradient descent optimiser, KNN and random forest classification

In [48]:

```python
#ML imports
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
```

```
Using TensorFlow backend.
```

In [49]:

```python
filt_df = pd.DataFrame(pbmc.X, index = pbmc.obs_names, columns = pbmc.var_names)
filt_df
```

Out[49]:

	HES4	ISG15	TNFRSF18	TNFRSF4	ACAP3	MRPL20	RBP7	PGD	FBXO44	AGTRAP	...	MRPL39	USP16	C21orf
AAACATACACCCAA-1	0.0	0.000000	0.0	0.0	0.0	0.650055	0.0	0.000000	0.000000	0.0	...	0.0	0.0	
AAACATTGCTTCGC-1	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.000000	0.000000	0.0	...	0.0	0.0	
AAACCGTGCGATAC-1	0.0	0.000000	0.0	0.0	0.0	0.470350	0.0	0.000000	0.000000	0.0	...	0.0	0.0	
AAACCGTGTCACCC-1	0.0	1.011345	0.0	0.0	0.0	0.000000	0.0	0.000000	0.628421	0.0	...	0.0	0.0	
AAACGCACGGGTGA-1	0.0	0.000000	0.0	0.0	0.0	0.821195	0.0	0.000000	0.000000	0.0	...	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
TTTCGAACTACTGG-8	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.000000	0.000000	0.0	...	0.0	0.0	
TTTCTACTGGGACA-8	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.000000	0.000000	0.0	...	0.0	0.0	
TTTGACTGAAGTAG-8	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.538942	0.000000	0.0	...	0.0	0.0	
TTTGACTGCCCAAA-8	0.0	0.000000	0.0	0.0	0.0	0.604478	0.0	0.000000	0.000000	0.0	...	0.0	0.0	
TTTGCATGCTGCAA-8	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.000000	0.000000	0.0	...	0.0	0.0	

19198 rows × 903 columns

In [50]:

```python
label_df = pd.DataFrame(pbmc.obs['cell-types'])
label_df
```

Out[50]:

	cell-types
AAACATACACCCAA-1	CD8+ Cytotoxic T
AAACATTGCTTCGC-1	CD8+ Cytotoxic T
AAACCGTGCGATAC-1	CD8+ Cytotoxic T
AAACCGTGTCACCC-1	CD8+ Cytotoxic T
AAACGCACGGGTGA-1	Dendritic
...	...
TTTCGAACTACTGG-8	CD8+ Cytotoxic T
TTTCTACTGGGACA-8	CD8+ Cytotoxic T
TTTGACTGAAGTAG-8	CD8+ Cytotoxic T
TTTGACTGCCCAAA-8	CD8+ Cytotoxic T
TTTGCATGCTGCAA-8	CD8+ Cytotoxic T

19198 rows × 1 columns

One Hot Encoding

In [51]:

```python
labels = label_df['cell-types'].values
encoder = LabelEncoder()
encoder.fit(labels)

encoded_Y = encoder.transform(labels)
encoded_Y
```

Out[51]: `array([2, 2, 2, ..., 2, 2, 2])`

In [52]:

```python
#turns the encodings into vectors
dummy_y = np_utils.to_categorical(encoded_Y)
# Let's say we want to split the data in 75 /12.5 /12.5 for train / valid / test dataset
X_train_n, X_remain_n, y_train_n, y_remain_n = train_test_split(filt_df, dummy_y, train_size=0.75, shuffle=True

# Now since we want the valid and test size to be equal (10% each of overall data).
# we have to define valid_size=0.5 (that is 50% of remaining data)
X_val_n, X_test_n, y_val_n, y_test_n = train_test_split(X_remain_n,y_remain_n, test_size=0.3, shuffle=True, ran
print(X_train_n.shape), print(y_train_n.shape)
print(X_remain_n.shape), print(y_remain_n.shape)
print(X_test_n.shape), print(y_test_n.shape)
```

```
(14398, 903)
(14398, 4)
(4800, 903)
(4800, 4)
(1440, 903)
(1440, 4)
```

Out[52]: `(None, None)`

Running the models

In [53]:

```python
model = Sequential()

model.add(Dense(903, input_dim = 903, activation='relu'))
model.add(Dense(600, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(600, activation='relu'))
model.add(Dense(4, activation='sigmoid'))
#tf.keras.metrics.Precision(),
model.compile(loss='binary_crossentropy', optimizer='SGD', metrics=['accuracy', tf.keras.metrics.Recall()])

history = model.fit(X_train_n, y_train_n, validation_data = (X_val_n, y_val_n), epochs = 30, batch_size = 24)
model.summary()
```