Massive Computational Experiments, Painlessly

STATS 285
Stanford University
Vardan Papyan

Course info

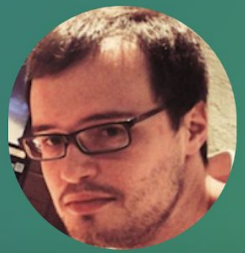
- Monday 3:00 - 4:20 PM at 380-380W
- Sept 23 - Dec 2 (10 Weeks)
- Website: http://stats285.github.io
- Twitter: @stats285
- Instructors:
  - David Donoho, email donoho@stanxxx.edu
  - Vardan Papyan, email papyan@stanxxx.edu

Massive Computational Experiments, Painlessly (STATS 285)

Stanford University, Fall 2019

Ambitious Data Science requires massive computational experimentation; the entry ticket for a solid PhD in some fields is now to conduct experiments involving 1 Million CPU hours. This course covers state-of-the-art practices for conducting massive computational experiments in the cloud in a pain-free and reproducible manner. In addition to giving students a hands-on experience with cluster computing, the course features several guest lectures by renowned data scientists.

Instructors:



List of speakers and schedule

September 30: Mark Piercy 

October 7: XY Han 

October 14: Riccardo Murri 

October 21: Percy Liang 

October 28: Orhan Firat 

November 4: Hatef Monajemi 

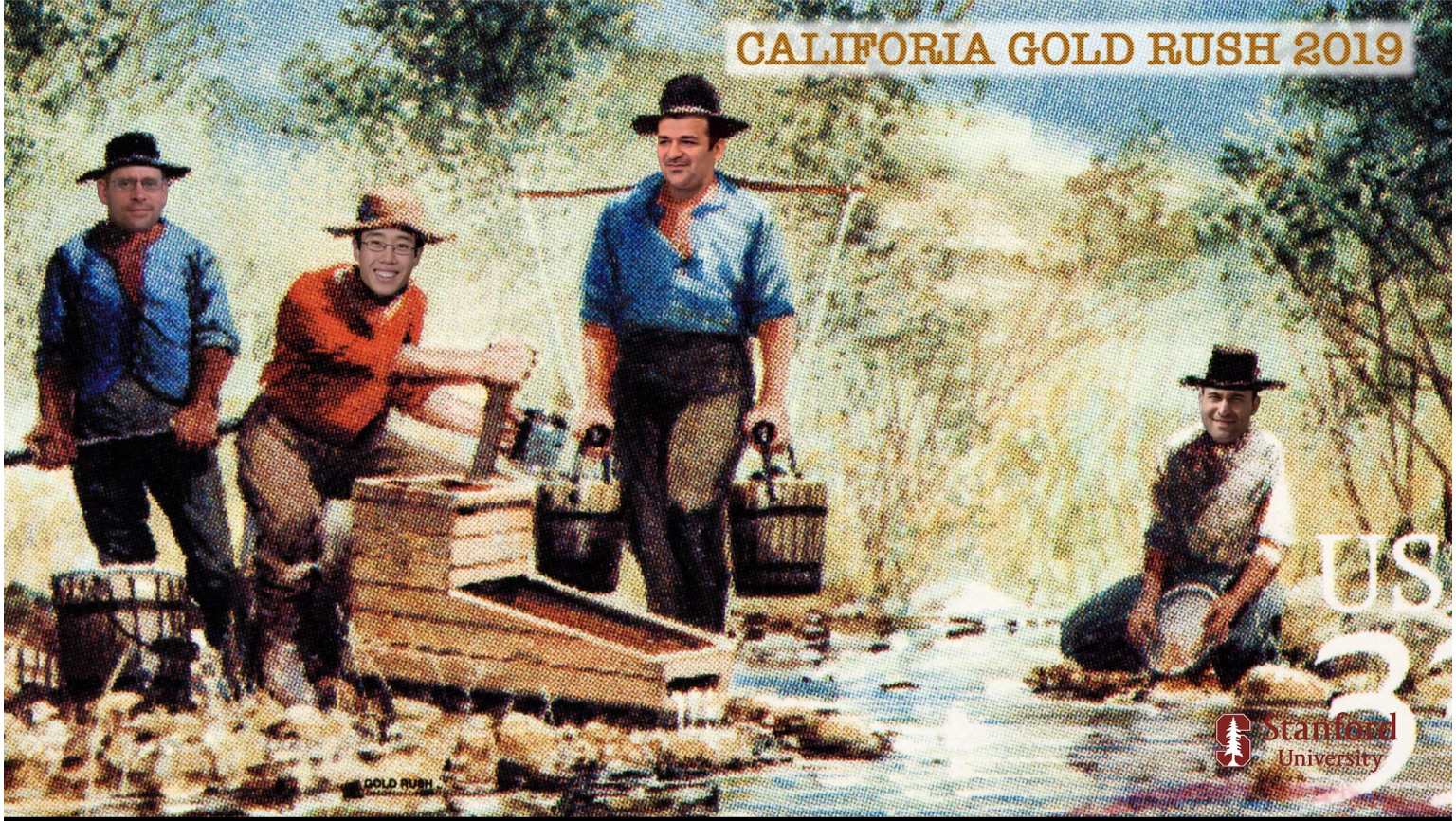
November 11: Leland Wilkinson 

November 18: Han Liu 

Massive Computational

https://stats285.github.io/

Stanford University



Massive Computational Experiments, Painlessly (STATS 285)

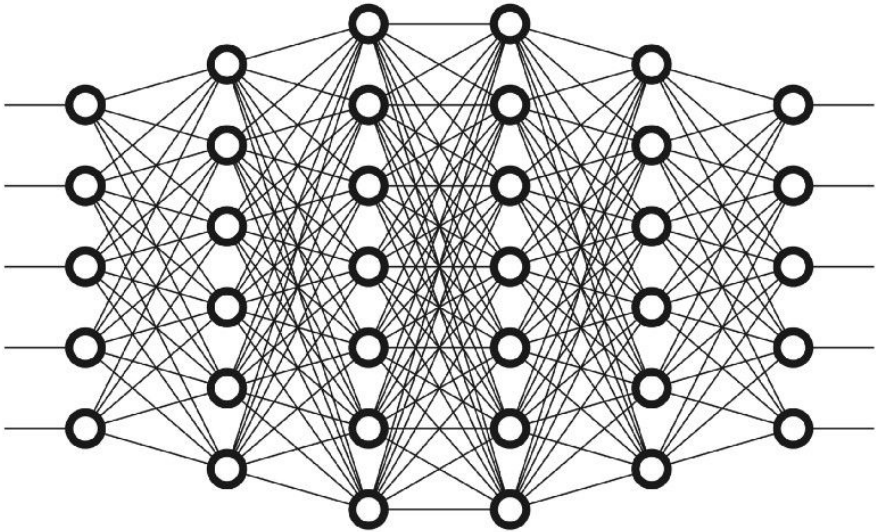
Monday 3:00 - 4:20 PM at 380 - 380W
Stanford University Fall 2019
https://stats285.github.io/

Massive Computational Experiments, Painlessly (STATS 285)

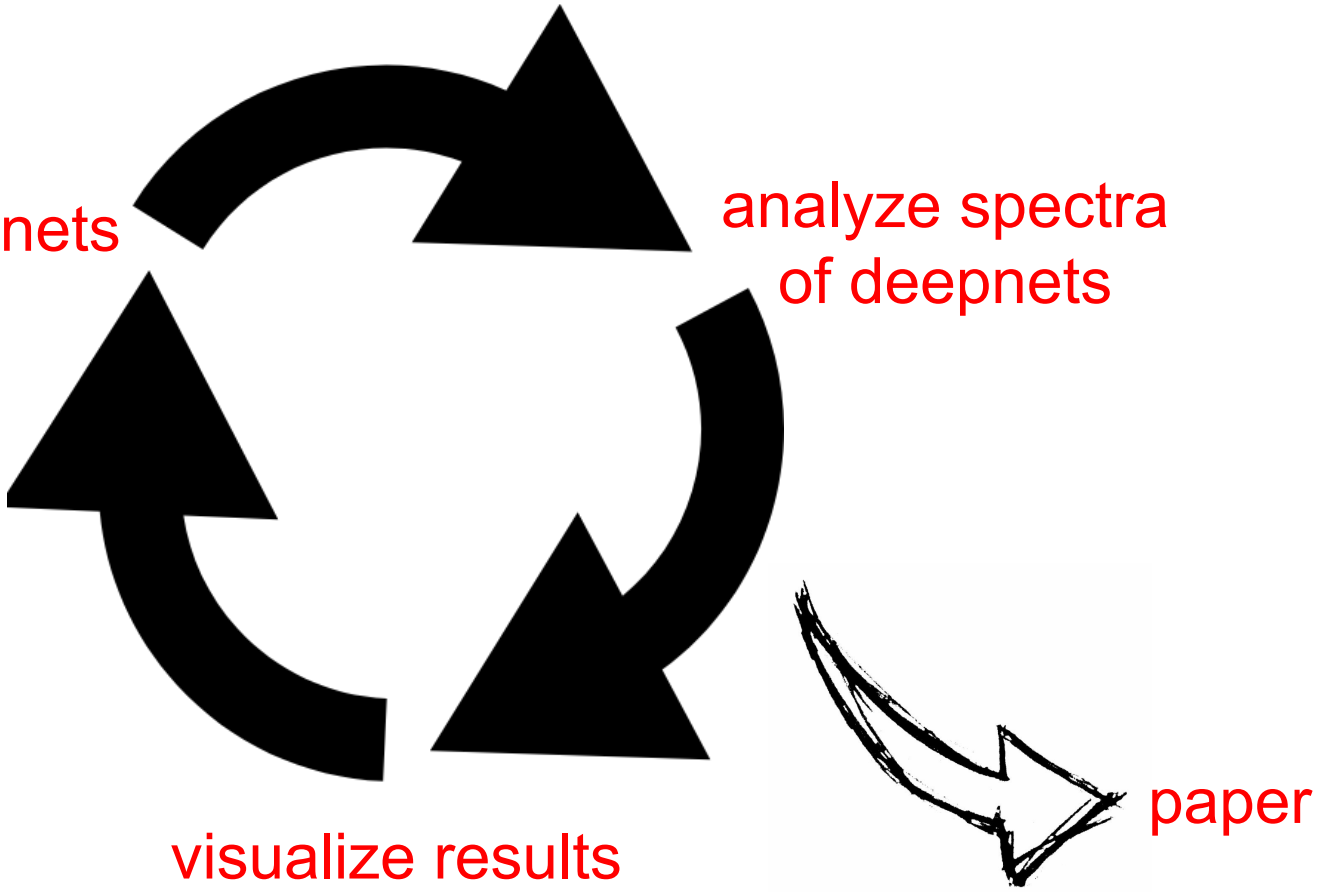
My research

Study spectra of deepnet:

- Features
- Backpropagated errors
- Gradients
- Fisher information matrix
- Hessian
- …



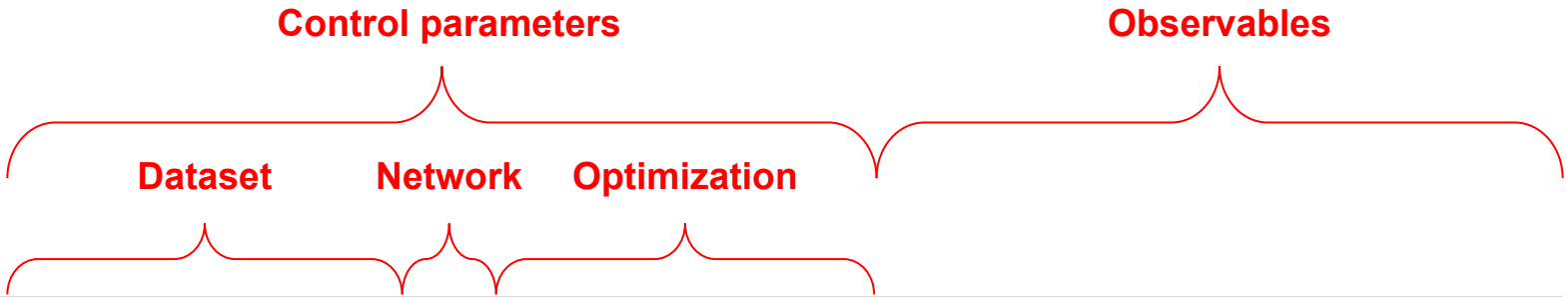
The grind


train deepnets
analyze spectra of deepnets
visualize results
paper

Training deepnets: experiment specification

- Dataset:
  - MNIST, FashionMNIST, CIFAR10, CIFAR100, ImageNet
- Network:
  - MLP, LeNet, VGG, ResNet
- Control parameters:
  - Dataset: sample size, number of classes
  - Network: width, depth
  - Optimization: algorithm, learning rate, learning rate scheduler, batch size
- Observables:
  - Top1 error, loss

Training deepnets: experiment results

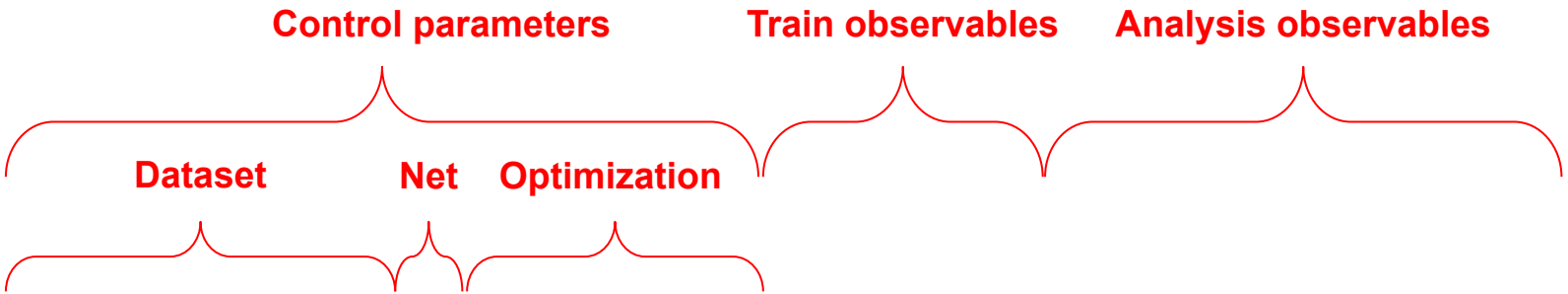


	Dataset	Network	Number of Classes	Examples per class	Depth	Optimizer	Learning rate	Batch size	Phase	Epoch	Top-1 error	Cross-entropy Loss
0006d21e9a	MNIST	VGG	10	5000	11	SGD	0.25	128	test	250	99.38322368421052	0.028507341263129524
000e518c8d	MNIST	VGG	10	5000	11	SGD	0.25	128	train	196	99.81637286324786	0.007986395810850156
01003dd1ea	MNIST	VGG	10	5000	11	SGD	0.25	128	test	30	98.41694078947368	0.057988858144534264
01421e353c	MNIST	VGG	10	5000	11	SGD	0.25	128	test	143	99.22902960526316	0.029485975980366532
0215d9cc00	MNIST	VGG	10	5000	11	SGD	0.25	128	train	129	99.8046875	0.0081629870324117
0216427c78	MNIST	VGG	10	5000	11	SGD	0.25	128	train	187	99.86812232905983	0.0058947943158957185
022684ea96	MNIST	VGG	10	5000	11	SGD	0.25	128	test	25	98.4888980263158	0.05374518193696674
027314b8b3	MNIST	VGG	10	5000	11	SGD	0.25	128	test	234	99.34210526315789	0.027728269660943432
02a513b57a	MNIST	VGG	10	5000	11	SGD	0.25	128	test	349	99.44490131578948	0.028374243527650833
031de9266f	MNIST	VGG	10	5000	11	SGD	0.25	128	test	44	98.01603618421052	0.07326431356762585
03585ce035	MNIST	VGG	10	5000	11	SGD	0.25	128	train	77	98.5844017094017	0.05233209439688641
362668092	MNIST	VGG	10	5000	11	SGD	0.25	128	train	292	100	0.001758613106277254
03672e776e	MNIST	VGG	10	5000	11	SGD	0.25	128	train	317	100	0.001763534063521104

Analyzing deepnets: analysis specification

- Dataset:
  - MNIST, FashionMNIST, CIFAR10, CIFAR100, ImageNet
- Network:
  - MLP, LeNet, VGG, ResNet
- Control parameters:
  - Dataset: sample size, number of classes
  - Network: width, depth
  - Optimization: find control parameters leading to best top-1 error
- Observables:
  - Spectra of deepnets features, backpropagated errors, gradients, Fisher information matrix, Hessian, …

Analyzing deepnets: analysis results



	Dataset	Network	Number of classes	Examples per class	Depth	Optimizer	Learning rate	Batch	Top-1 error	Path to model	Matrix type	Layer	Index of eigenvalue	Eigenvalue
00008c3007	MNIST	MLP	10	5000	8	SGD	[0.0678604404	128	98.97203947	/scratch/users/p	features	0	240	0.095884531
0003bb8fa2	MNIST	MLP	10	5000	8	SGD	[0.0678604404	128	98.97203947	/scratch/users/p	backprop errors	0	1562	6.224693205
0009cbc2ee	MNIST	MLP	10	5000	8	SGD	[0.0678604404	128	98.97203947	/scratch/users/p	backprop errors	0	228	4.193413758
000c50127e	MNIST	MLP	10	5000	8	SGD	[0.0678604404	128	98.97203947	/scratch/users/p	features	0	148	0.263989627
000d56712f	MNIST	MLP	10	5000	8	SGD	[0.0678604404	128	98.97203947	/scratch/users/p	backprop errors	0	1084	3.365795819
001039f911	MNIST	MLP	10	5000	8	SGD	[0.0678604404	128	98.97203947	/scratch/users/p	features	0	274	0.086070373
0015db9977	MNIST	MLP	10	5000	8	SGD	[0.0678604404	128	98.97203947	/scratch/users/p	backprop errors	0	699	1.015973296
001d99a2ed	MNIST	MLP	10	5000	8	SGD	[0.0678604404	128	98.97203947	/scratch/users/p	backprop errors	0	496	1.637548194
00224a8725	MNIST	MLP	10	5000	8	SGD	[0.0678604404	128	98.97203947	/scratch/users/p	backprop errors	0	975	5.148281800
00254c969b	MNIST	MLP	10	5000	8	SGD	[0.0678604404	128	98.97203947	/scratch/users/p	backprop errors	0	1986	-4.59046075
0027ad40da	MNIST	MLP	10	5000	8	SGD	[0.0678604404	128	98.97203947	/scratch/users/p	backprop errors	0	1477	7.881029162
002b584223	MNIST	MLP	10	5000	8	SGD	[0.0678604404	128	98.97203947	/scratch/users/p	backprop errors	0	1794	9.020487135
002fdcbd3a	MNIST	MLP	10	5000	8	SGD	[0.0678604404	128	98.97203947	/scratch/users/p	backprop errors	0	103	8.312590580

In practice slightly more complicated...

Phase
Dataset_path
Test_trans_only
Drop_last
Sampler
Corrupt_prob
Load_epoch
Train_batch_size
Test_batch_size
Training_results_path
Anals_results_path
Layers_func
Seed
Absorb_bn
Filter_bn
Milestones_perc
Gamma
Train_batch_size
Training_results_path
Save_middle

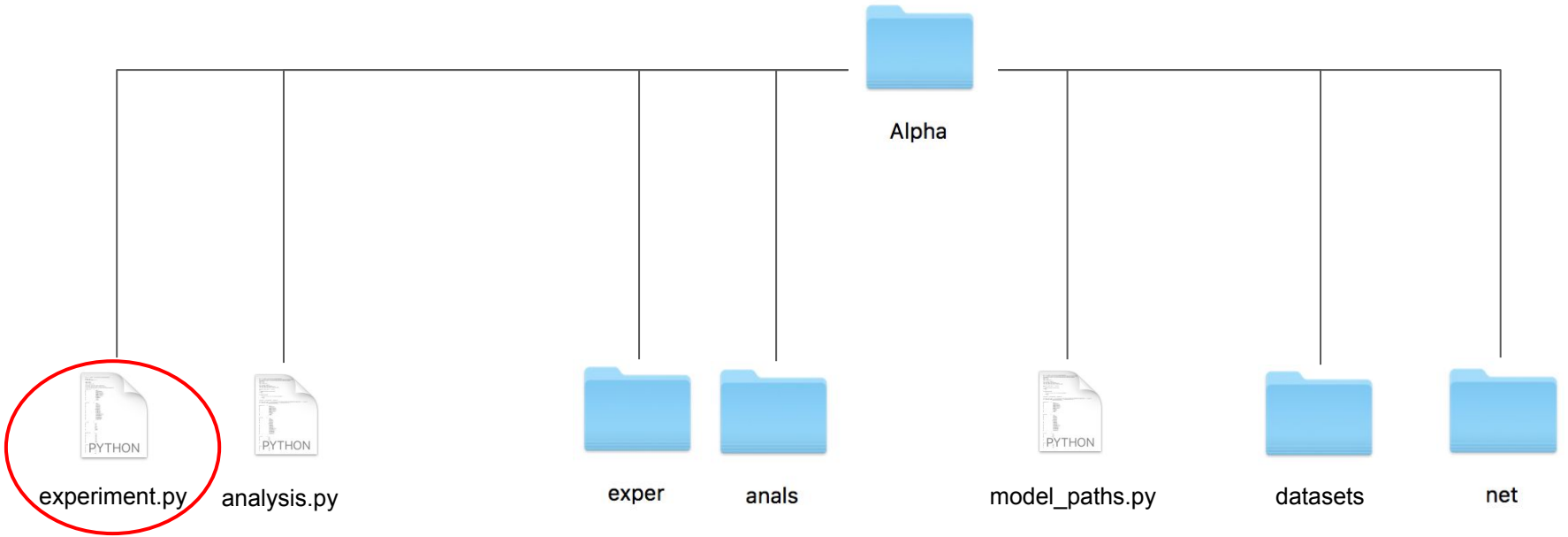
K_Normalization
Damping
Ignore_bias
save_K
Hessian_layer
All_params
Hessian_type
Init_poly_degpoly_deg
Poly_points
Spectrum_margin
Kappa
Log_hessian
Start_eig_range
Stop_eig_range
Power_method_iters
Test_batch_size
Device
Seed
Train_dump_file
Epoch_list

Repeat_idx
N_vec
Mult_num_classes
Trace_est_iters
Perplexity_list
Double
Rand_model
Bidiag
Cpu_eigvec
G_decomp_cpu
Train_dataset
Test_dataset
Loader_type
Pytorch_dataset
Dataset_path
Concat_loader
Switch_relu_pool
Scattering
Save_init_epoch
One_batch

Dataset_kwargs
Im_size
Padded_im_size
Num_classes
Input_ch
Threads
Limited_dataset
Examples_per_class
Epc_seed
Train_seed
Size_list
Pretrained
Retrain_last
Multilabel
Corrupt_prob
Reset_classifier
Resnet_type
Test_trans_only
Garbage_collect
Epochs

Double
Loader_constructor
Sampler
Pin_memory
normalized_Fashion
Momentum
Weight_decay
GAN
Forward_class
Classification
Forward_func
Critnet
Optim
Optim_kwargs
Epochs
Lr
Net_width
Num_layers

Alpha



specification of experiment and analysis

implementation of experiment and analysis

locations of trained models

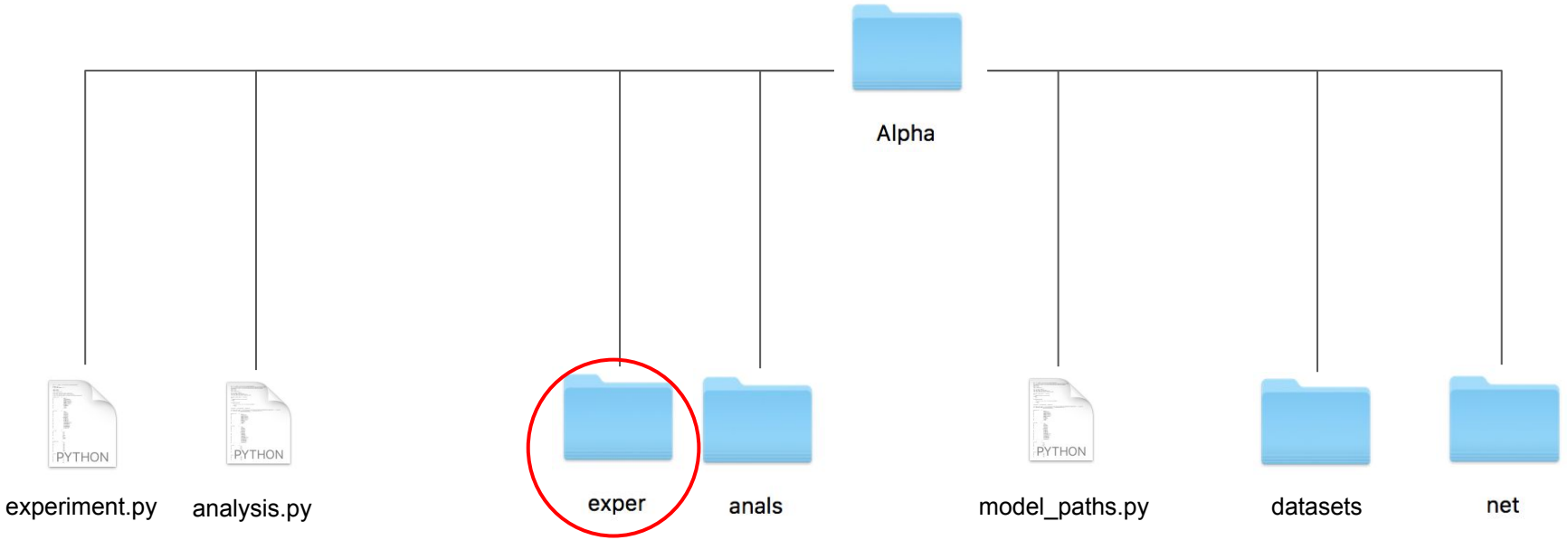
datasets

networks

experiment.py -- experiment specification

```python
from exper import Experiment

dataset_list = ['MNIST', 'FashionMNIST', 'CIFAR10']
net_list = ['VGG11_bn', 'ResNet18']
lr_list = [0.1,0.05,0.001]
size_list = [13, 26, 51, 98, 189, 365, 702, 1351, 2599, 5000]

for dataset_idx in range(3):
    for net_idx in range(2):
        for size_idx in range(10):
            for lr_idx in range(3):

                dataset_opts = {'dataset'           : dataset_list[dataset_idx]
                                'examples_per_class': size_list[size_idx],
                                }

                network_opts = {'depth'          : 8,
                                }

                optimization_opts = {'net'          : net_list[net_idx],
                                     'optim'        : 'SGD',
                                     'momentum'     : 0.9,
                                     'weight_decay': 5e-4,
                                     'epochs'       : 350,
                                     'lr'           : lr_list[lr_idx],
                                     'batch_size'   : 2**7,
                                     }

                opts = dict(dataset_opts, **optimization_opts)
                opts = dict(opts, **network_opts)

                Experiment(opts).run()
```

Alpha



specification of experiment and analysis

implementation of experiment and analysis

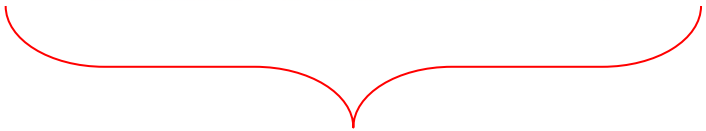
locations of trained models

datasets

networks

Experiment class -- experiment implementation

```
class Experiment:

    def __init__(self, opts):

        for key, value in opts.items():
            setattr(self, key, value)
```



Save all experiment specification in self

Experiment class -- experiment implementation

```
# criterion (loss)
import torch.nn as nn
func = getattr(nn, self.crit)
self.criterion = func()

# network
self.model = Network().construct(self.net, self)

# optimizer
import torch.optim as optim
func = getattr(optim, self.optim)
optimizer = func(self.model.parameters(), lr=lr, **self.optim_kwargs)

# datasets and loaders
constructor = LoaderConstructor().init(self)

self.train_dataset, self.train_loader = constructor.get_loader('train')
self.test_dataset, self.test_loader  = constructor.get_loader('test')
```

Use fields from experiment specification

Experiment class -- experiment implementation

```
# iterate over batches
for input,label in self.loader:
    input = input.to(self.device)
    label = label.to(self.device)

    # run model
    prediction = self.model

    # compute loss
    loss = self.crit(prediction, label)

    # backpropagate
    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()
```

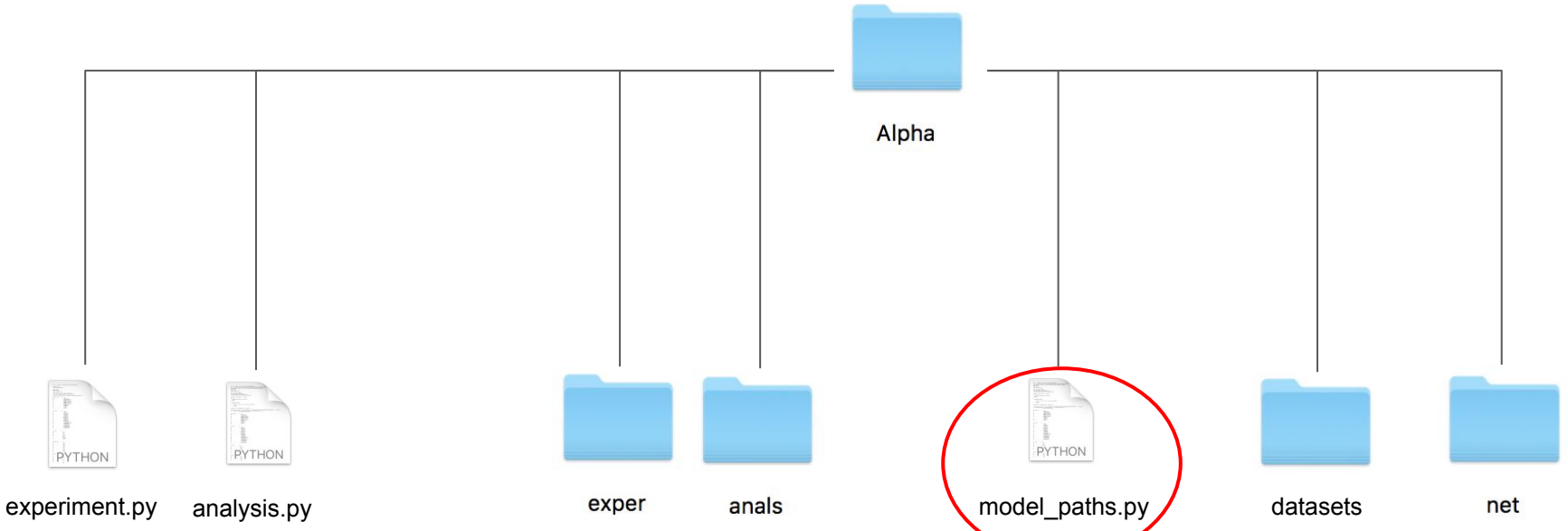
Experiment class -- experiment implementation

```
stats = {'phase'              : phase,
         'dataset'            : dataset,
         'epoch'              : epoch,
         'iter'               : iter,
         'iters'              : len(loader),
         'iter_batch_time'    : batch_time.val,
         'avg_batch_time'     : batch_time.avg,
         'iter_data_time'     : data_time.val,
         'avg_data_time'      : data_time.avg}

results.append(dict(self.__getstate__(), **stats))
```

experiment specification observables

Concatenate experiment specification to observables and as row to csv

Alpha



specification of experiment and analysis

implementation of experiment and analysis

locations of trained models

datasets

networks

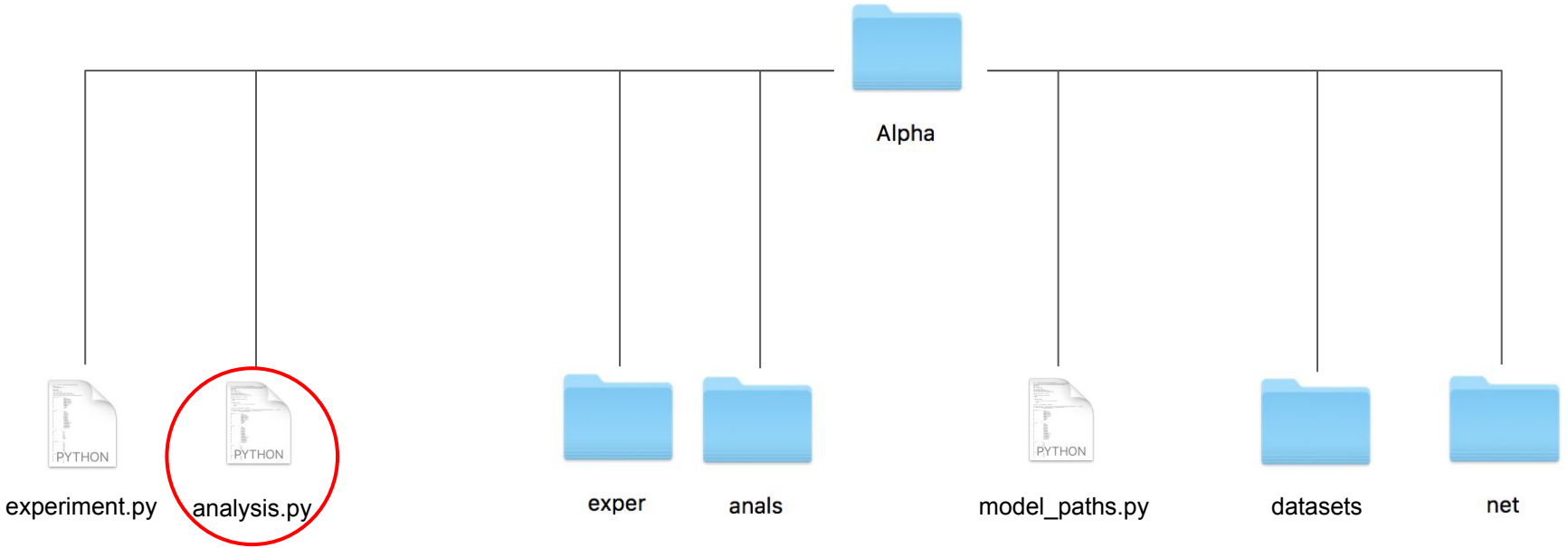
model_paths.py

dictionary of trained model paths

```python
def get_model_path(dataset, network):
    return paths[dataset+'.'+network]

paths = {'MNIST.VGG11_bn'        : '/scratch/users/papyan/trained_models/3ad186bbfe8d7f6a6dfc1b116463956e72cd8b6b/',
         'FashionMNIST.VGG11_bn' : '/scratch/users/papyan/trained_models/8dd94ceaea2fe0a49f9b2f369ae12ccb35323056/',
         'CIFAR10.VGG11_bn'      : '/scratch/users/papyan/trained_models/43a7a251e5f0377cc0d59cd0bd15b94e87cc5f15/',
         'MNIST.ResNet18'        : '/scratch/users/papyan/trained_models/eafc959c4215e81a053b9823832cee8f42db760f/',
         'FashionMNIST.ResNet18' : '/scratch/users/papyan/trained_models/5dc2d5768e4dae39739b13a0ebcafbe57bd92cae/',
         'CIFAR10.ResNet18'      : '/scratch/users/papyan/trained_models/8d8c31045a4252a1b9a7b6aa6d3e041321b063f7/',
         }
```

* Each of this paths corresponds to all the models trained for a certain dataset and a certain network

Alpha



specification of experiment and analysis

implementation of experiment and analysis

locations of trained models

datasets

networks

analysis.py -- analysis specification

```
from model_paths import get_path
from anals.analysis import Analysis
from misc import get_csv
from misc import find_best_model

dataset_list = ['MNIST', 'FashionMNIST', 'CIFAR10']
net_list = ['VGG11_bn', 'ResNet18']
size_list = [13, 26, 51, 98, 189, 365, 702, 1351, 2599, 5000]
epoch_list = [10, 100]

for dataset_idx in range(3):
    for net_idx in range(2):
        for size_idx in range(10):
            for epoch_idx in range(2):

                path = get_path(dataset_list[dataset_idx], net_list[net_idx])

                df = get_csv(path)

                best_df, _ = find_best_model(size_list[size_idx])

                analysis_opts = {'load_epoch' : epoch_list[epoch_idx],
                                 }

                opts = dict(df, **analysis_opts)

                Analysis(opts).run()
```

Sherlock (Mark Piercy, next week)

- Cluster at Stanford
- Has many computational resources
  - CPUs
  - GPUs
- Useful for storing data
  - Laptop very limited in terms of memory
  - Data can get deleted if not touched for too long
  - Cloud costs money
- Interactive IPython notebook (Sherlock on demand)

ClusterJob (Hatef Monajemi, Nov. 4th)

```python
from model_paths import get_path
from anals.analysis import Analysis
from misc import get_csv
from misc import find_best_model

dataset_list = ['MNIST', 'FashionMNIST', 'CIFAR10']
net_list = ['VGG11_bn', 'ResNet18']
size_list = [13, 26, 51, 98, 189, 365, 702, 1351, 2599, 5000]
epoch_list = [10, 100]

for dataset_idx in range(3):
    for net_idx in range(2):
        for size_idx in range(10):
            for epoch_idx in range(2):

                path = get_path(dataset_list[dataset_idx], net_list[net_idx])

                df = get_csv(path)

                best_df, _ = find_best_model(size_list[size_idx])

                analysis_opts = {'load_epoch' : epoch_list[epoch_idx],
                                 }

                opts = dict(df, **analysis_opts)

                Analysis(opts).run()
```

Easily parallelizable!

dataset_idx=0, net_idx=0, size_idx=0, epoch_idx=0

dataset_idx=0, net_idx=0, size_idx=0, epoch_idx=1

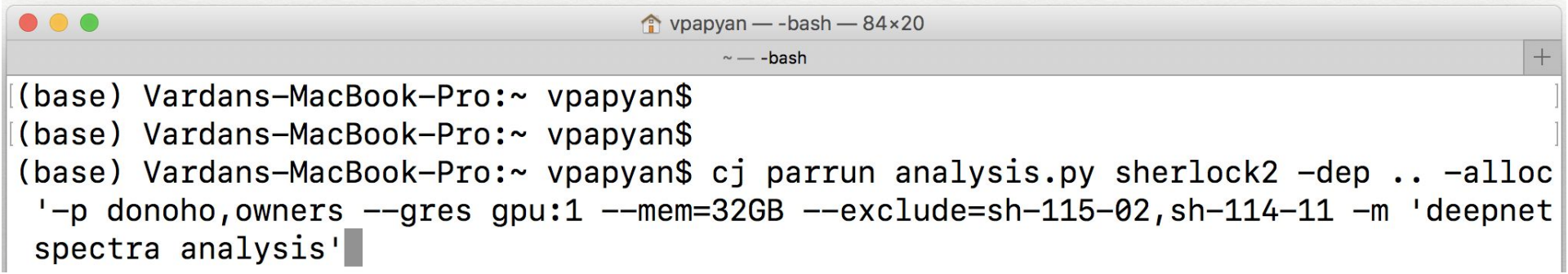
…

dataset_idx=2, net_idx=1, size_idx=3, epoch_idx=0

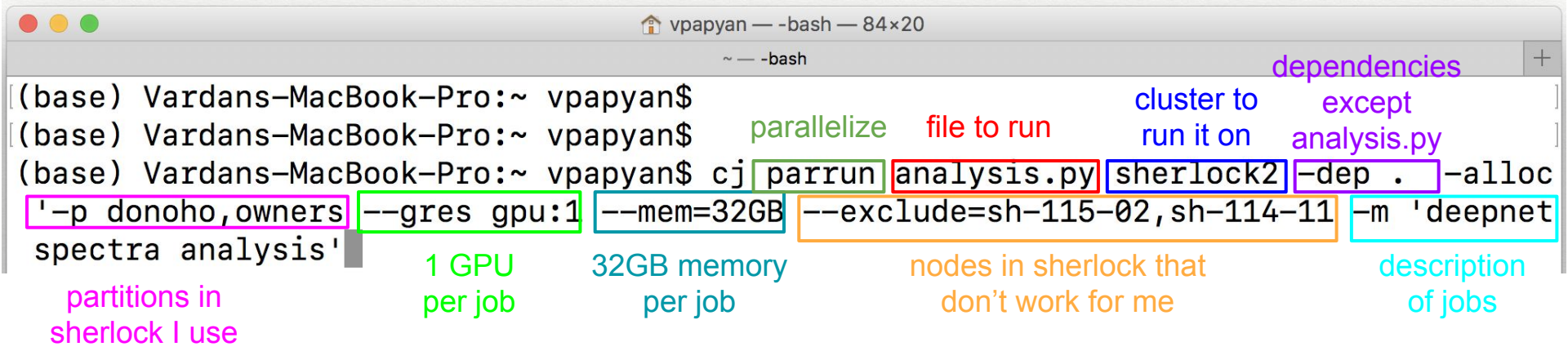
…

dataset_idx=2, net_idx=1, size_idx=9, epoch_idx=1

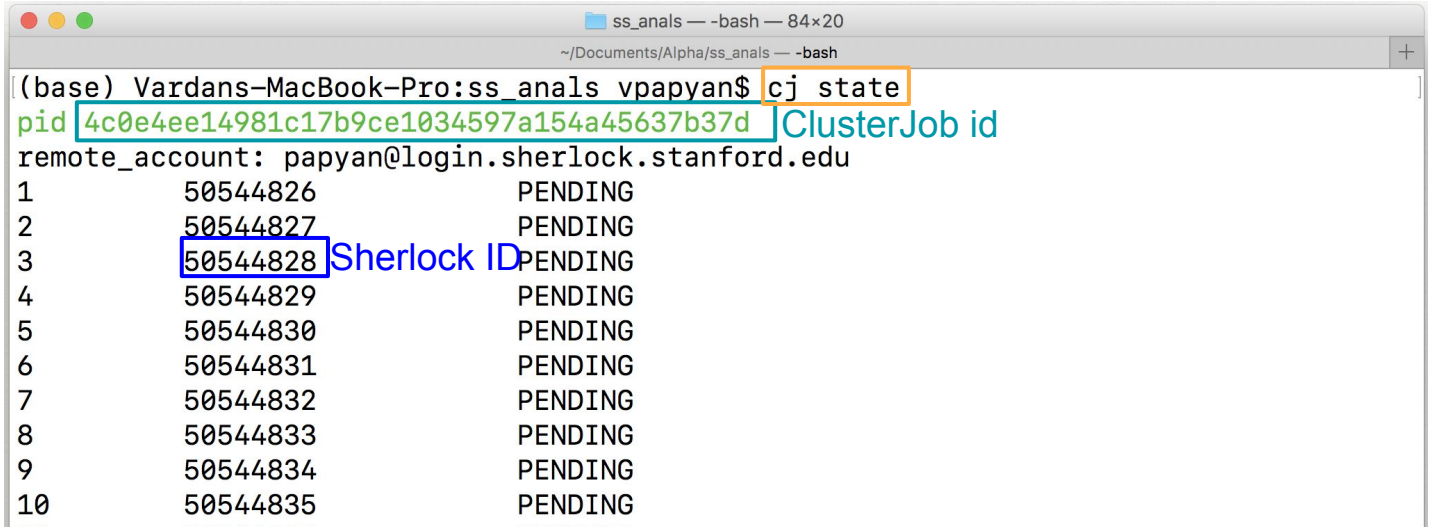
ClusterJob (Hatef Monajemi, Nov. 4th)



ClusterJob (Hatef Monajemi, Nov. 4th)



ClusterJob (Hatef Monajemi, Nov. 4th)



Sherlock ID

date on which job was submitted

* Useful command: sacct --jobs=23768102 --format=User,JobID,NodeList -S 2018-08-17

Can be used to find name of broken nodes

ClusterJob (Hatef Monajemi, Nov. 4th)

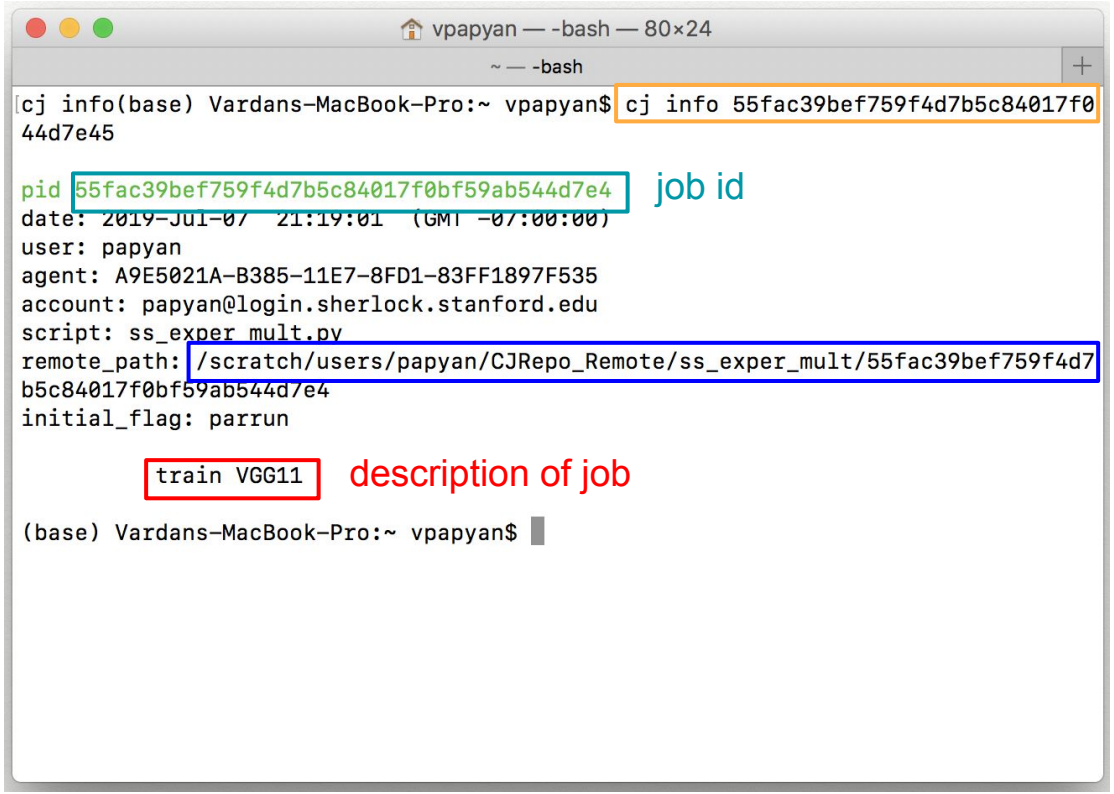
vpapyan — -bash — 167×20

~ — -bash

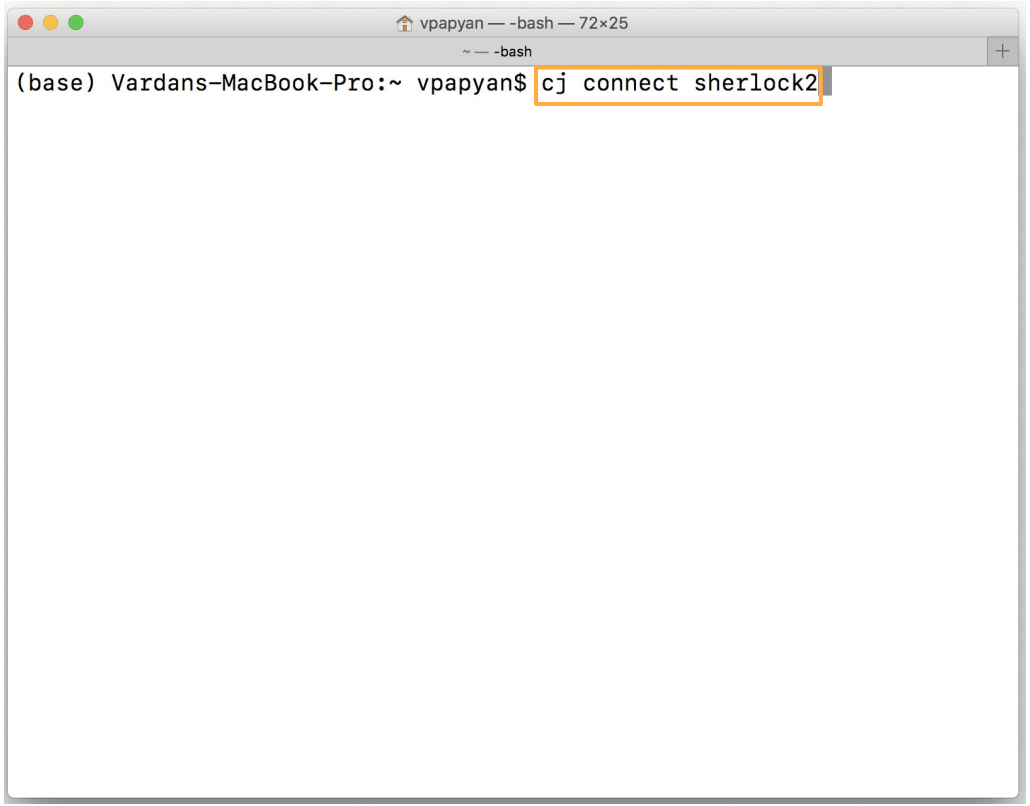
```
(base) Vardans-MacBook-Pro:~ vpapyan$ cj runlog 55fac39bef759f4d7b5c84017f0bf59ab544d7e4/1
Datasets path is /scratch/users/papyan/datasets
Creating Train Loader...
Test transform only
Loader created with  5000.0 examples per class from 10 classes.
Not Pinning Images to GPU Memory (Slower, Low memory)
Creating Test Loader...
Test transform only
Loader created with  1000.0 examples per class from 10 classes.
Not Pinning Images to GPU Memory (Slower, Low memory)
Total parameters in VGG11_bn:  28156554
Checkpoint saved to ./results/dataset=MNIST-net=VGG11_bn-lr=[0p25]-examples_per_class=5000-num_classes=10-epc_seed=0-train_seed=0-forward_class=Classification-epoch=0.
pth
/scratch/users/papyan/CJRepo_Remote/ss_exper_mult/55fac39bef759f4d7b5c84017f0bf59ab544d7e4/1
Train   Network: ['VGG11_bn']   Dataset: MNIST  Epoch: [1/350][1/468]   Time: 1.259 (1.259)    Data: 0.359 (0.359)     top1: 8.5938 (8.5938)   loss: 2.3421 (2.3421)
Train   Network: ['VGG11_bn']   Dataset: MNIST  Epoch: [1/350][2/468]   Time: 0.055 (0.657)    Data: 0.000 (0.180)     top1: 19.5312 (14.0625) loss: 3.5384 (2.9402)
Train   Network: ['VGG11_bn']   Dataset: MNIST  Epoch: [1/350][3/468]   Time: 0.040 (0.451)    Data: 0.000 (0.120)     top1: 21.8750 (16.6667) loss: 7.6854 (4.5219)
Train   Network: ['VGG11_bn']   Dataset: MNIST  Epoch: [1/350][4/468]   Time: 0.041 (0.349)    Data: 0.000 (0.090)     top1: 24.2188 (18.5547) loss: 9.7100 (5.8190)
Train   Network: ['VGG11_bn']   Dataset: MNIST  Epoch: [1/350][5/468]   Time: 0.028 (0.285)    Data: 0.000 (0.072)     top1: 15.6250 (17.9688) loss: 10.4643 (6.7480)
Train   Network: ['VGG11_bn']   Dataset: MNIST  Epoch: [1/350][6/468]   Time: 0.047 (0.245)    Data: 0.000 (0.060)     top1: 19.5312 (18.2292) loss: 8.3796 (7.0200)
```

Good for verifying jobs are running
Bad for visualizing results

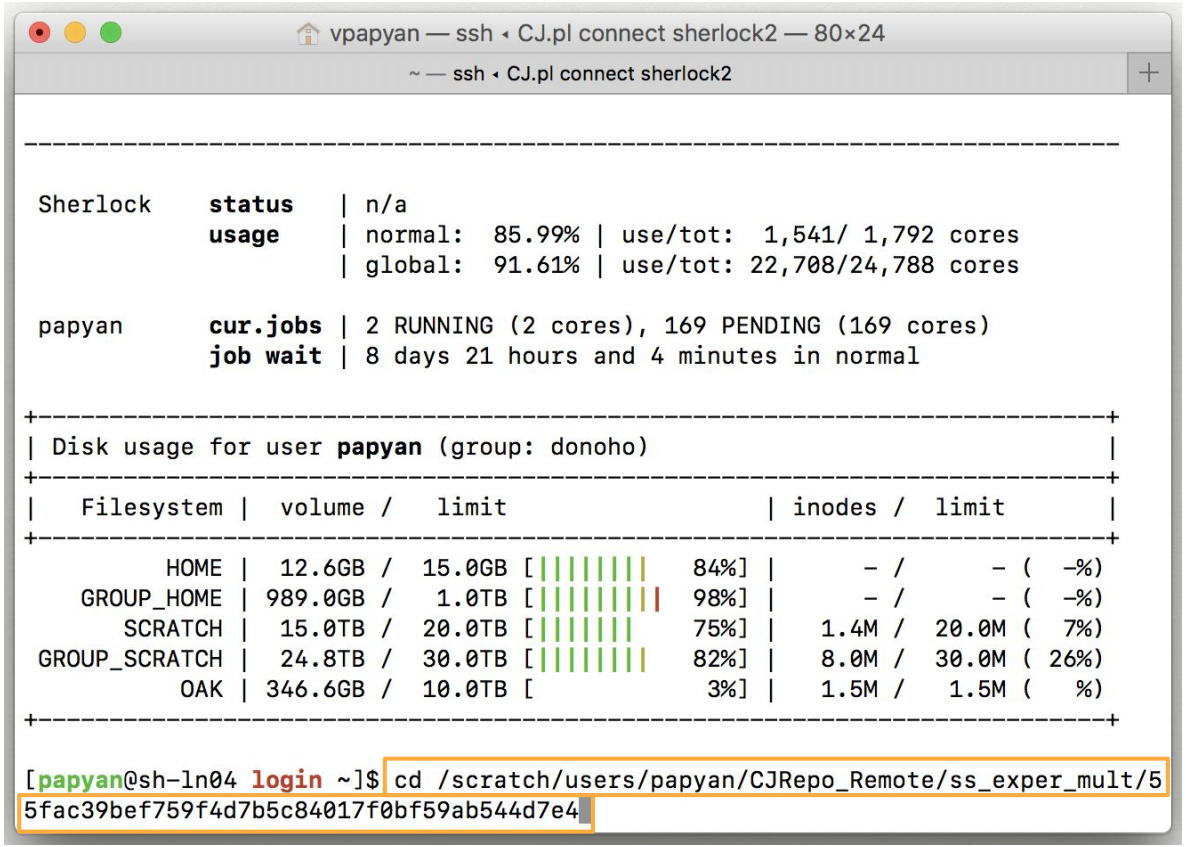
ClusterJob (Hatef Monajemi, Nov. 4th)



ClusterJob (Hatef Monajemi, Nov. 4th)

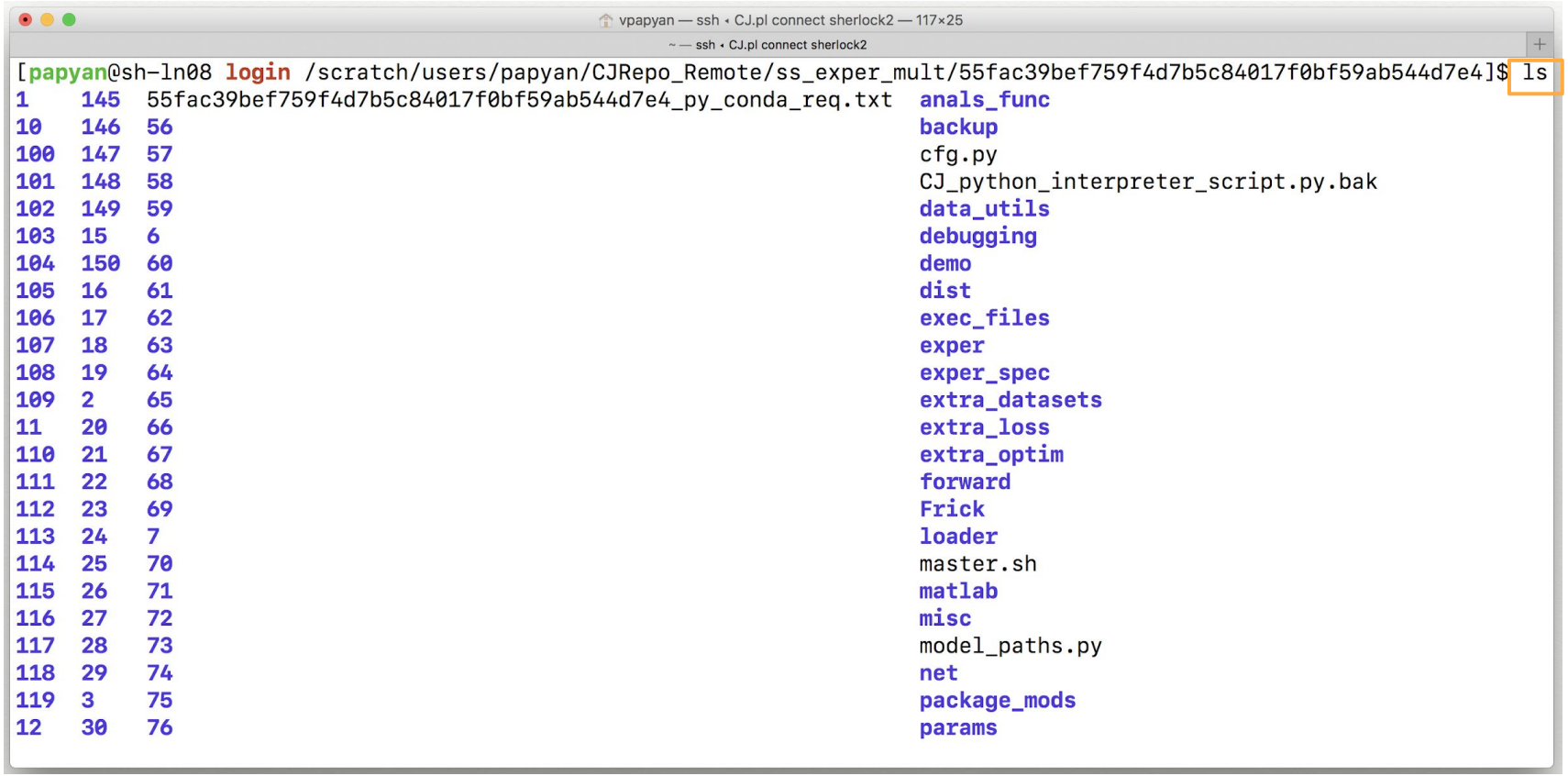


ClusterJob (Hatef Monajemi, Nov. 4th)

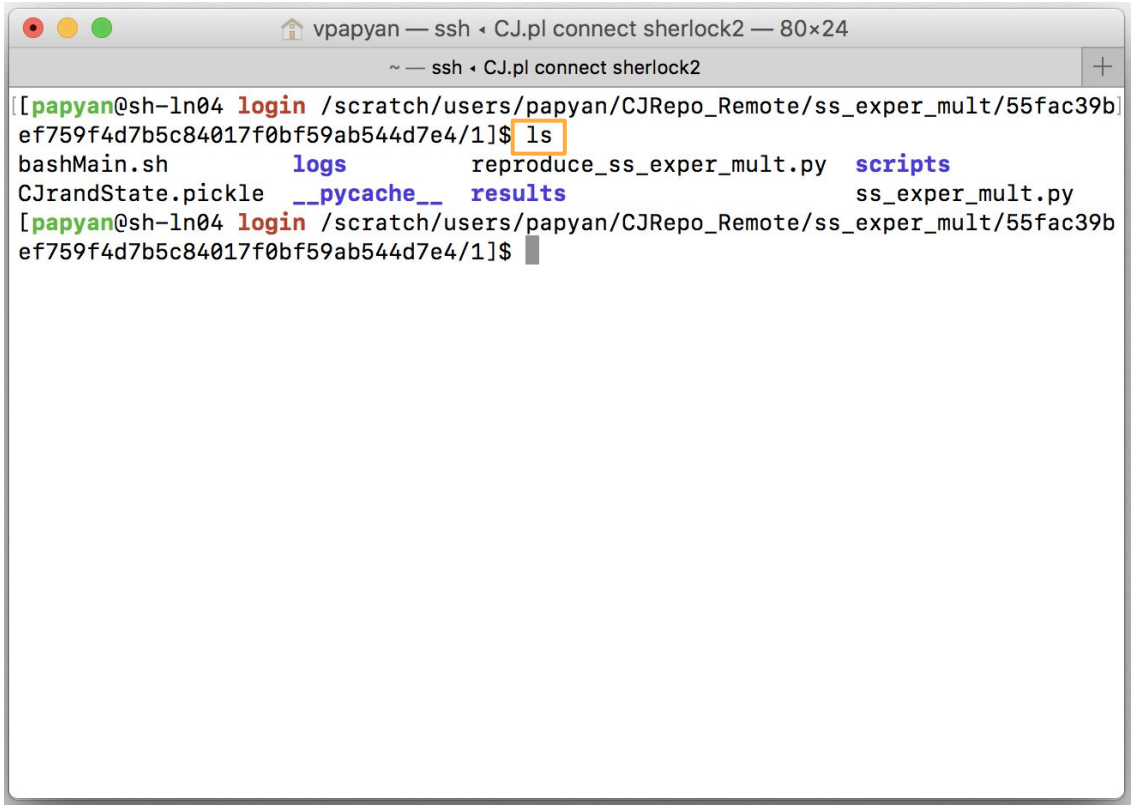


path on cluster to job

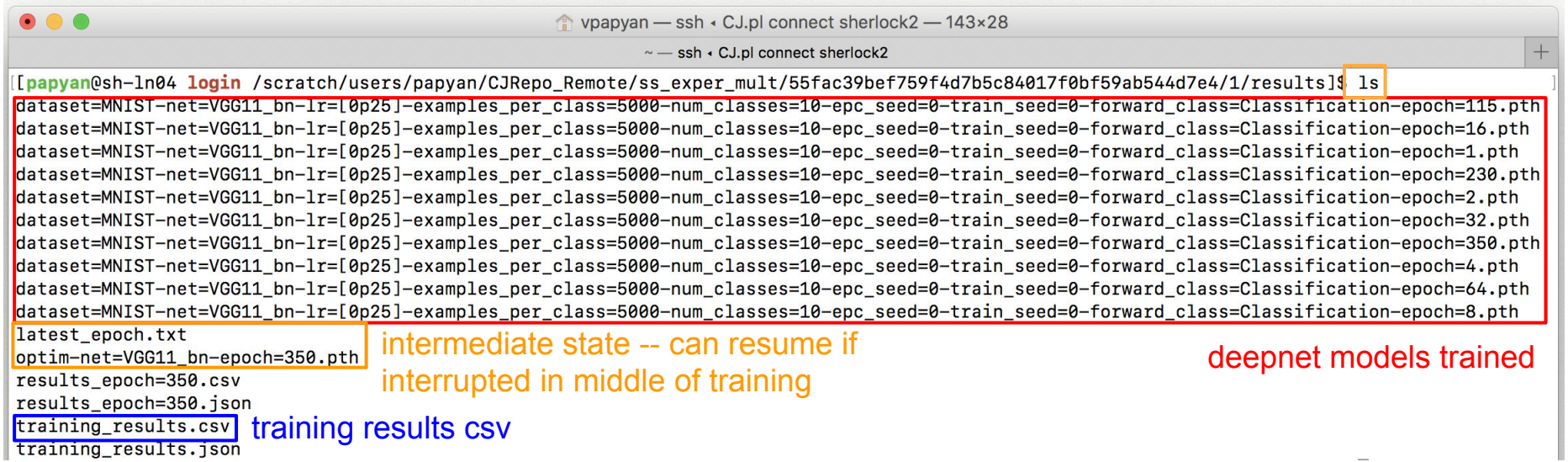
ClusterJob (Hatef Monajemi, Nov. 4th)



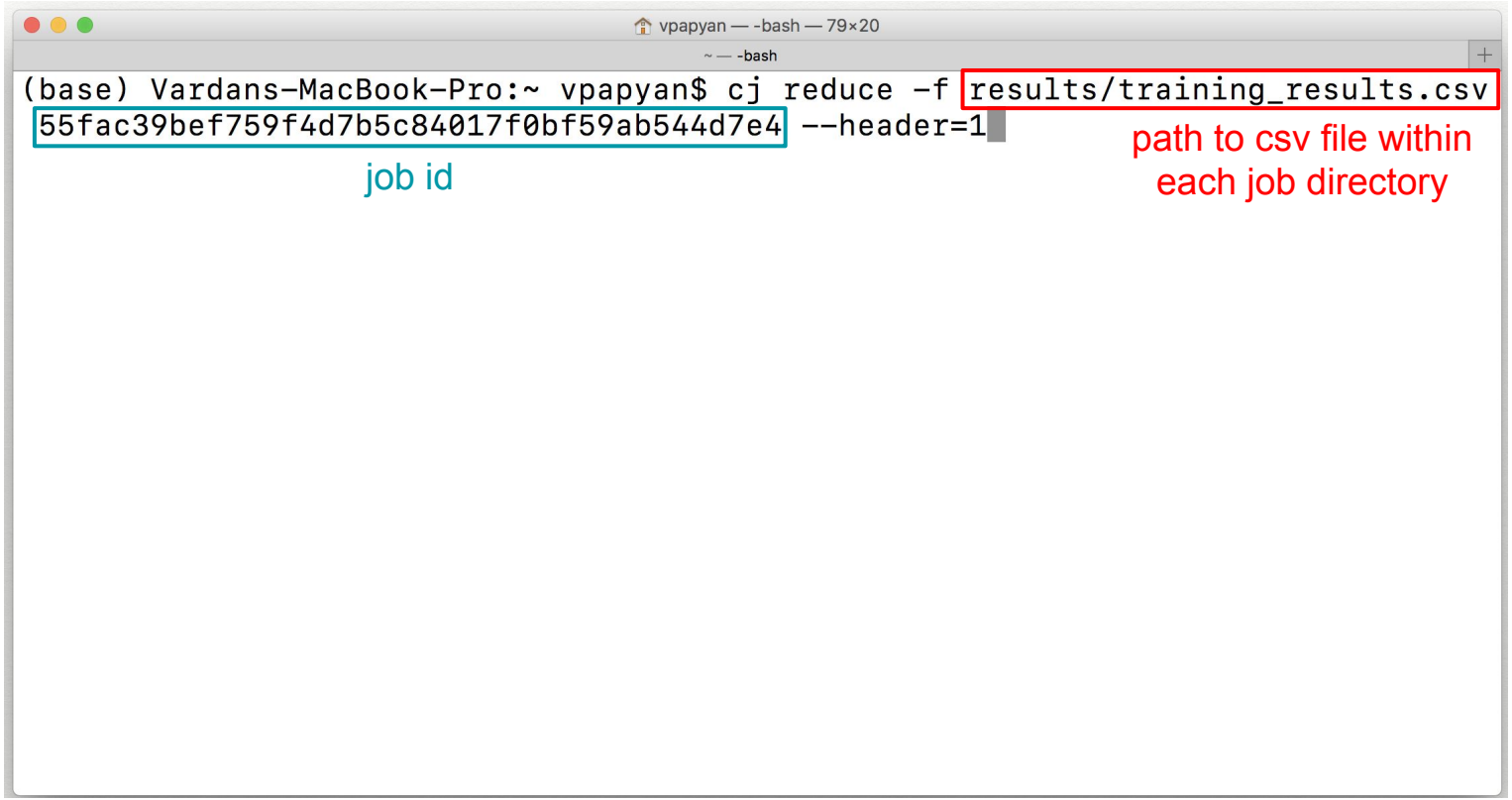
ClusterJob (Hatef Monajemi, Nov. 4th)



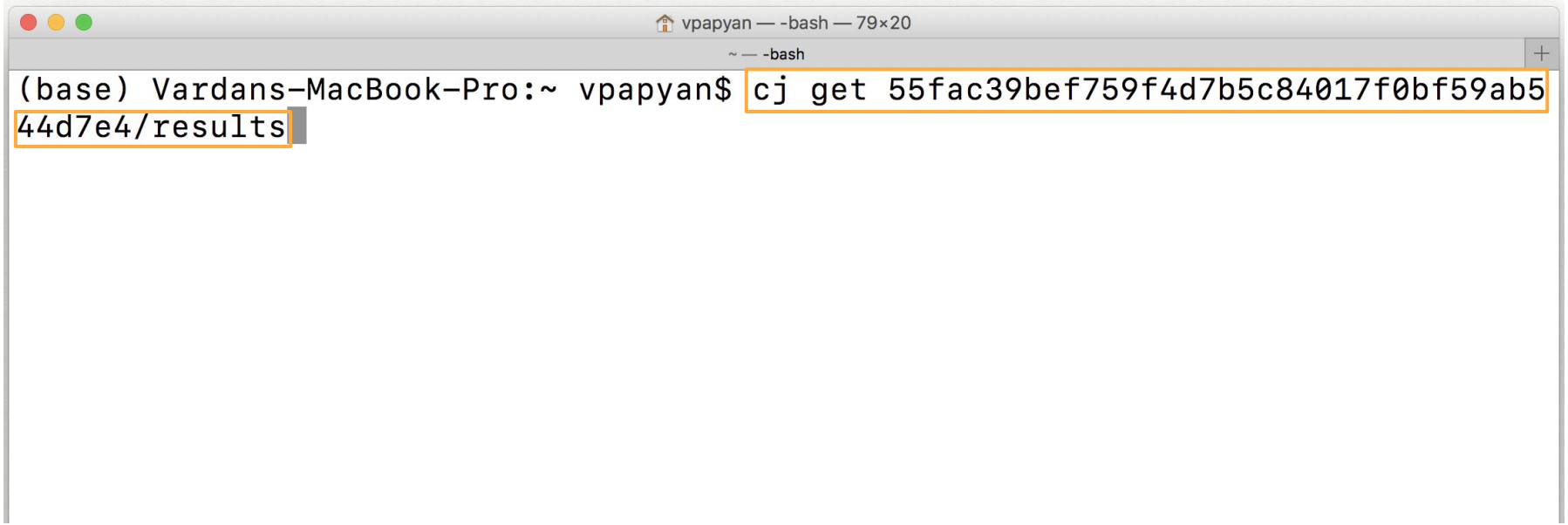
ClusterJob (Hatef Monajemi, Nov. 4th)



ClusterJob (Hatef Monajemi, Nov. 4th)



ClusterJob (Hatef Monajemi, Nov. 4th)

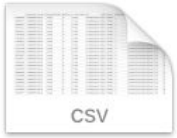


Good way of keeping track of running jobs: reduce, get, and plot locally

Elasticluster (Riccardo Murri, Oct. 14th)

- During quarter Sherlock can get busy
- Two options:
  - Work nights / weekends / holidays
  - Cloud computing
- Elasticluster allows to easily set up clusters on GCP/AWS/Azure/…
- Works seamlessly with ClusterJob

Tableau (XY Han Oct. 7th, Leland Wilkinson, Nov. 11th)



test_results.csv

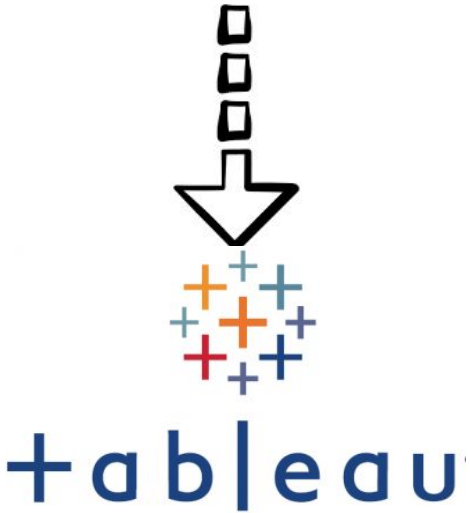


Tableau (XY Han Oct. 7th, Leland Wilkinson, Nov. 11th)

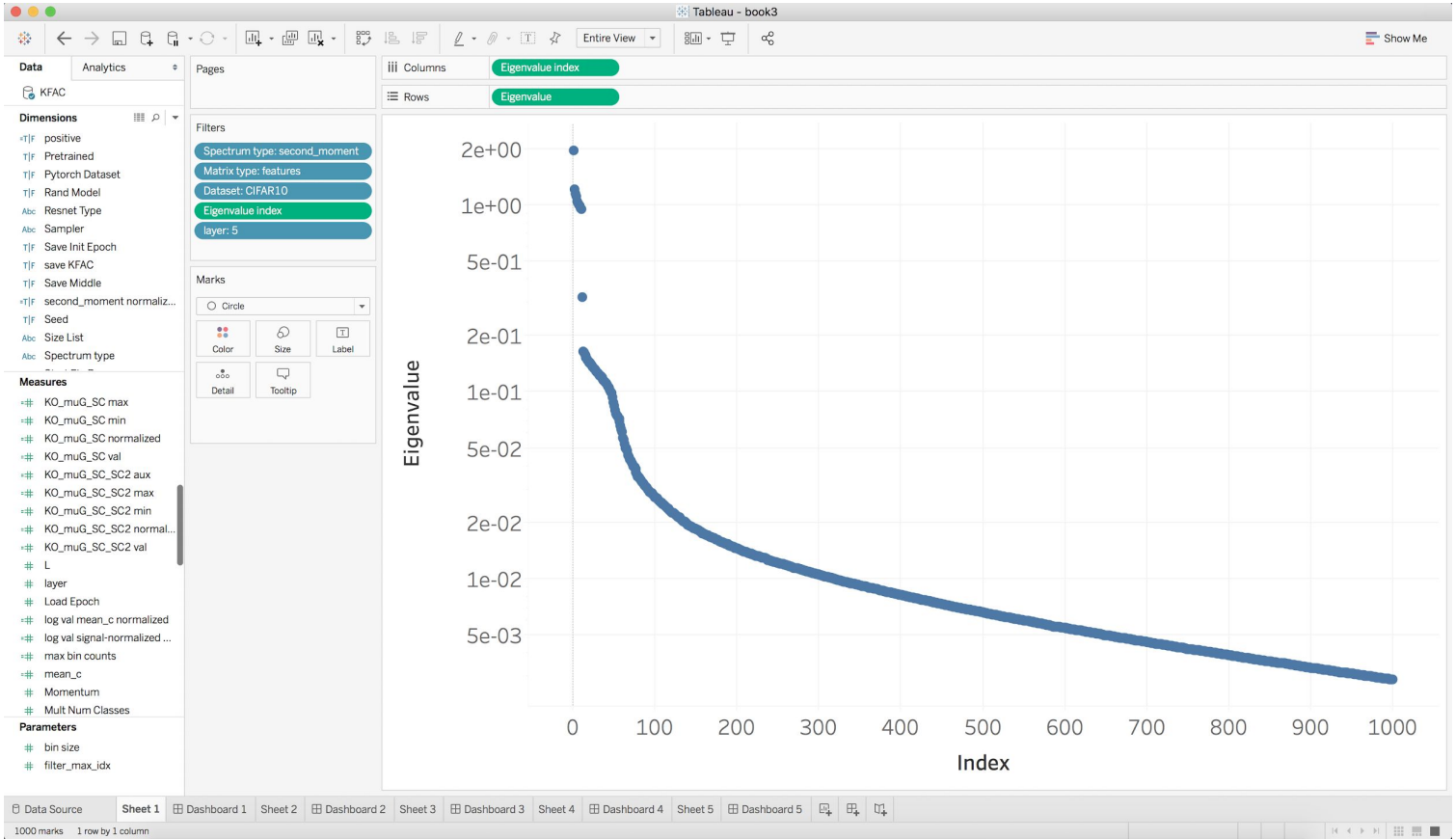
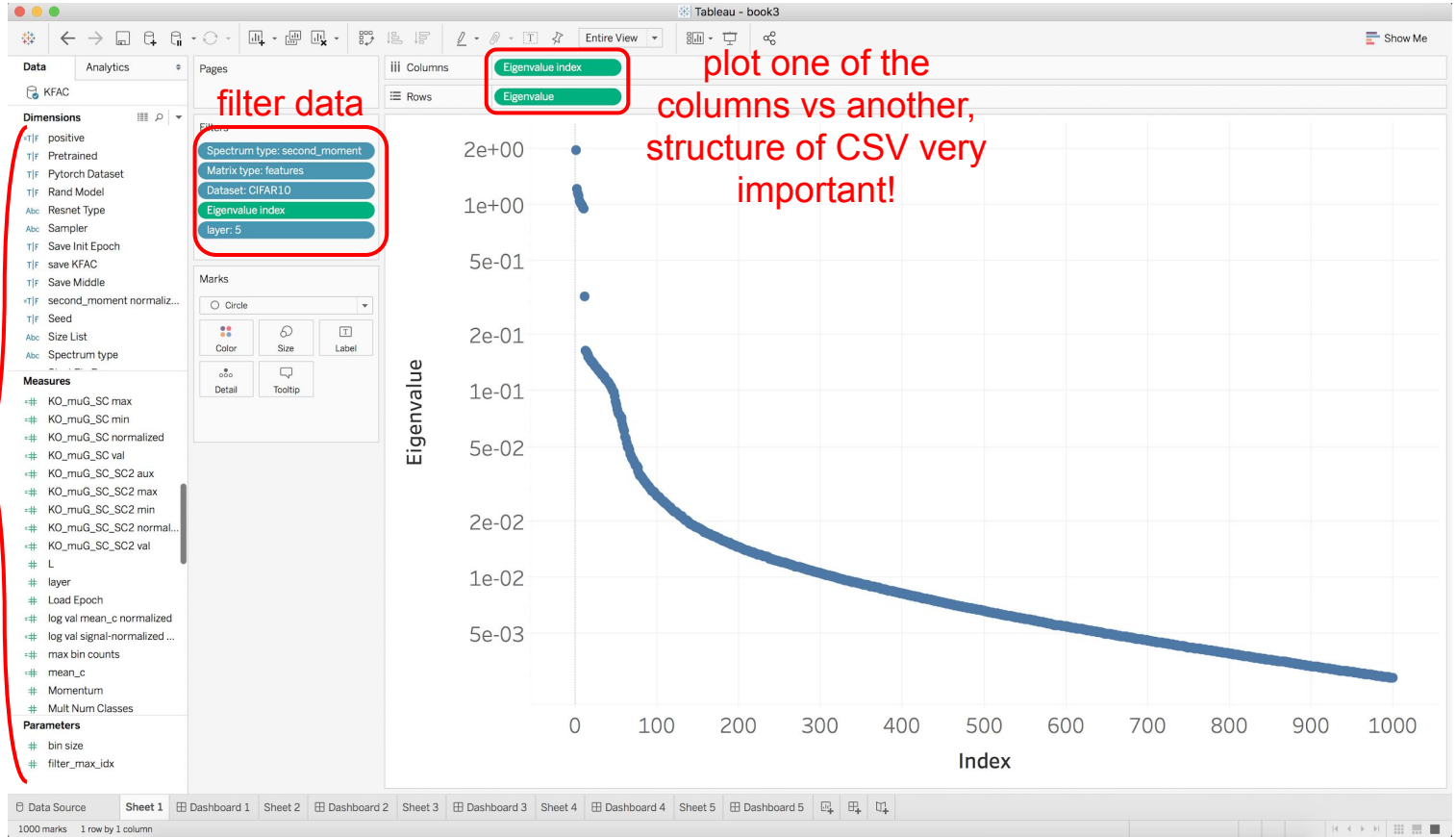


Tableau (XY Han Oct. 7th, Leland Wilkinson, Nov. 11th)



columns in csv file

Tableau (XY Han Oct. 7th, Leland Wilkinson, Nov. 11th)

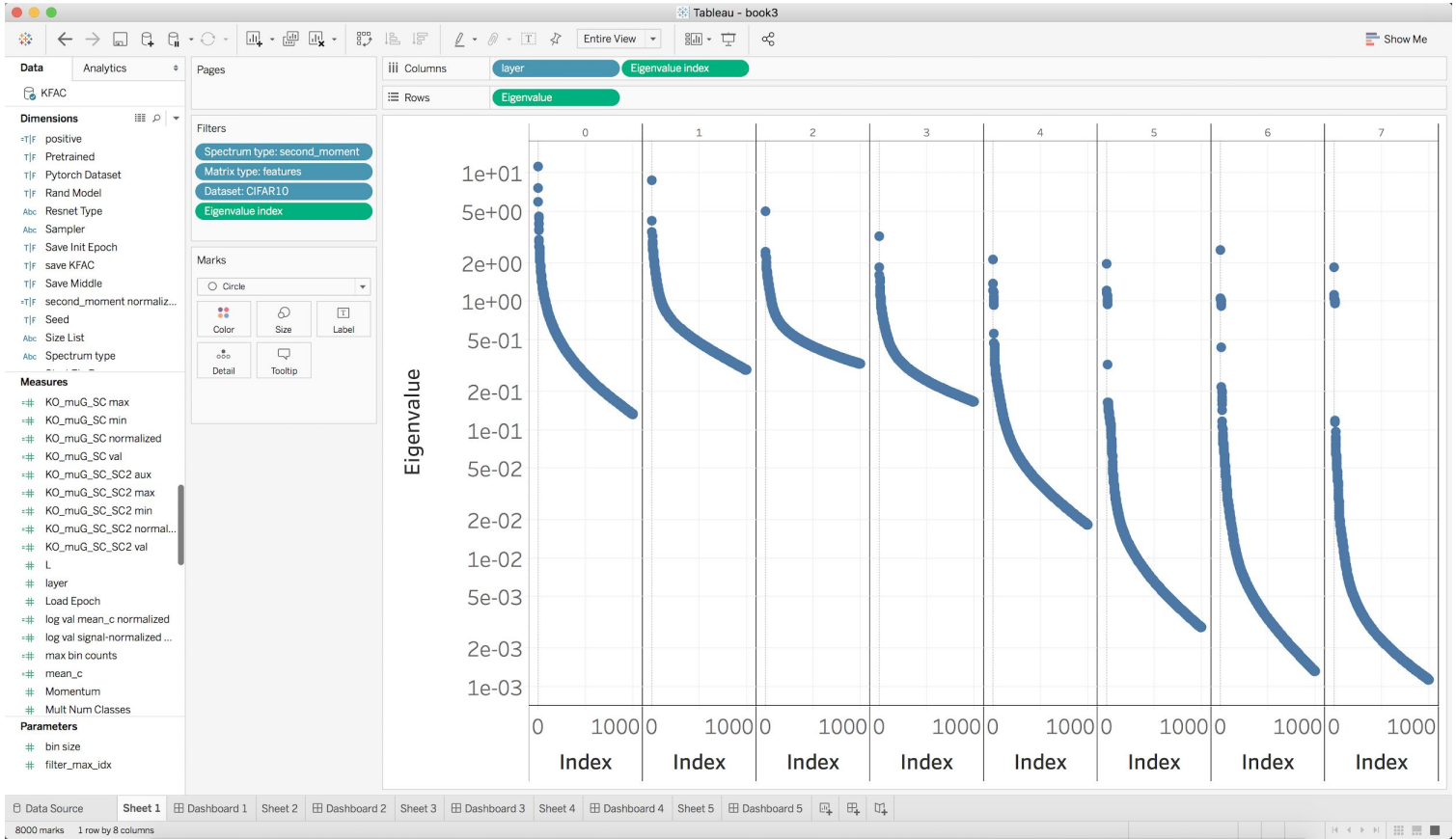


Tableau (XY Han Oct. 7th, Leland Wilkinson, Nov. 11th)

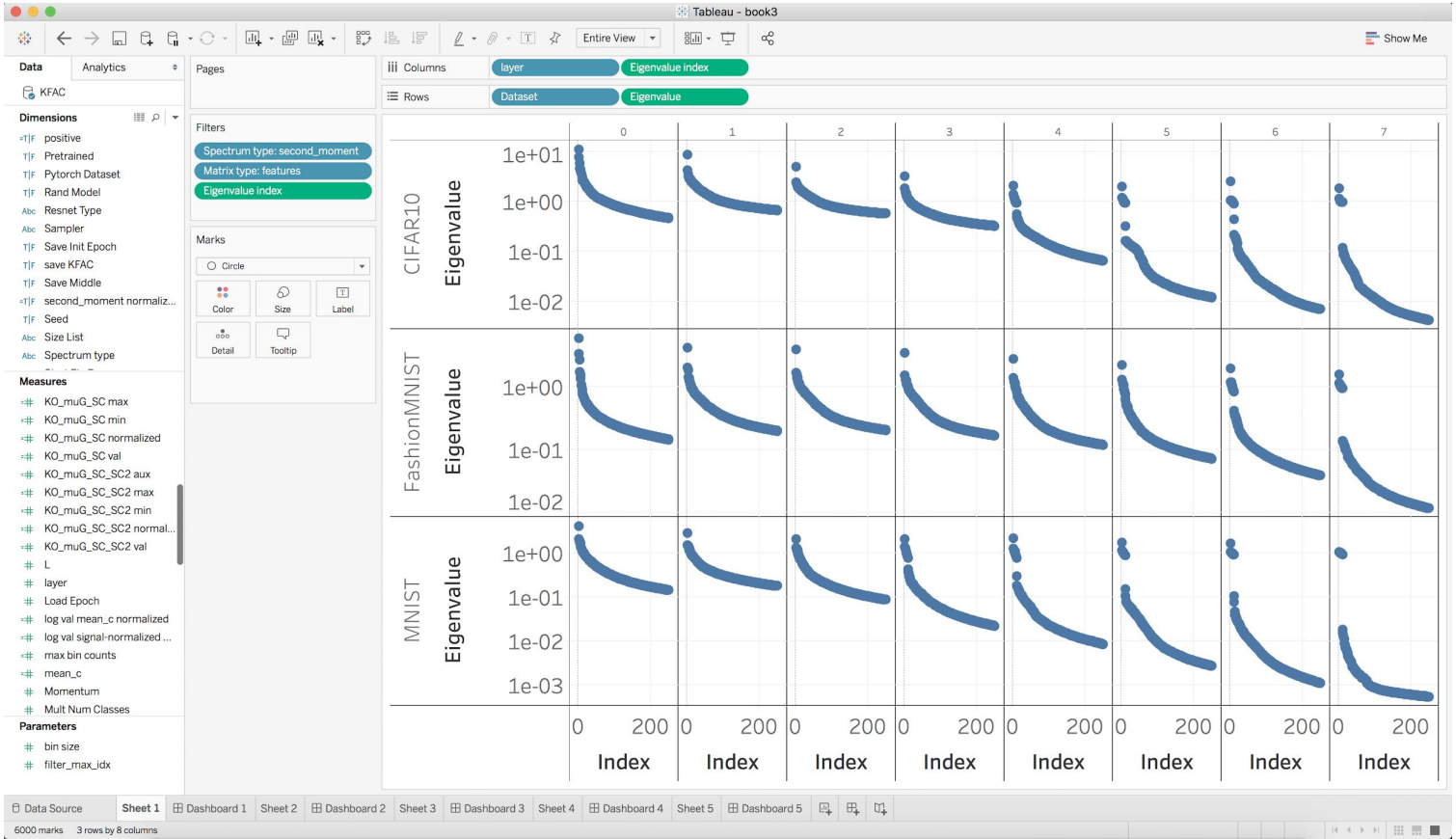


Tableau (XY Han Oct. 7th, Leland Wilkinson, Nov. 11th)

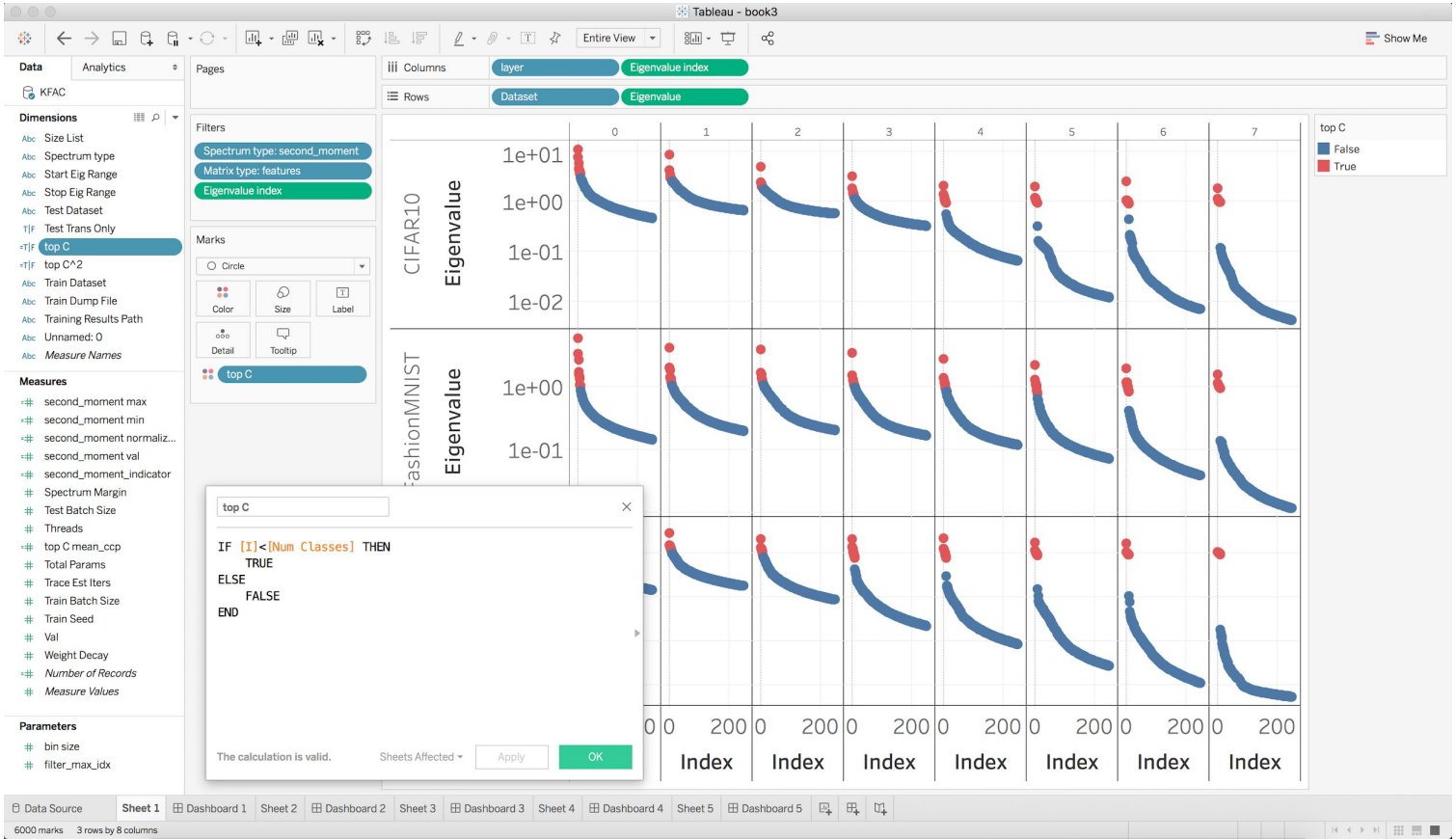


Tableau (XY Han Oct. 7th, Leland Wilkinson, Nov. 11th)

- Easy to analyze data -- drag and drop
- Easy to reproduce plots:
  - Delete results locally and keep only tableau sheet
  - Keep results on Sherlock2 / GCP
  - When need to recreate plot, download from cluster and open tableau sheet
- Easy to work with very large csv files using integration of tableau with the cloud
- Easy to calculate simple functions of existing columns

Summary

- **Alpha:** facilitates massive experiments by organizing code correctly
- **ClusterJob:** allows easy job parallelization
- **Sherlock2:** provides computational resources, storage, IPython notebooks
- **Elasticluster:** creates cluster on cloud, when sherlock is not enough
- **Tableau:** easy visualization of massive data