

CalSimPy:

Making the CalSim3 Data-Handling
Process a Little Easier, One Line of
Python Code at a Time

James Gilbert

Fisheries Collaborative Program, UC Santa Cruz

CWEMF Annual Meeting

April 20, 2026

Outline

- Motivation & Core Concepts
- Example CalSim functionality
- Extending to the C2VSim groundwater model
- ... and CalSimHydro, too
- Application: CalSimPy and resequencing hydrology
- Sharing and next steps

Motivation

- A single CalSim3 simulation entails a huge amount of disparate data
 - Meteorology, land use, streamflows
 - CalSimHydro, CalSim3, C2VSim
 - DSS, ascii tables, CSVs, GIS
- Excel spreadsheets are great for some of these things, but can be limiting when working with multiple types of data or many CalSim studies
- What if we had a flexible code library that made it easier to work with all of the different CalSim data?
 - My collection of *ad hoc* Python scripts → CalSimPy

CalSimPy – Core Concepts

- Each “model” or component is an object (a Python class):
 - A container to hold all the configuration and input/output data
 - Methods to interact with that data
- Access the model information the same way the numerical engines do (via *.launch, “main”, or “.in” files)
- Goal: make interacting with data across many models & many studies easier (to setup, to execute, to repeat, to adapt....)

CalSimPy: The CalSim object

- A container for CalSim study information
- Configuration
 - Start date – End Date
 - SV file path
 - DV file path
 - Init file path
 - DSS A part
 - DSS F part
 - *And all other info in the launch/config files*

```
<[1]>: cs3_study = cs3.calsim(launchFP = launch_fp1,  
calsim_version=3)
```

```
<[2]>: cs3_study
```

```
#####
```

```
CalSim object:
```

```
    Version: CalSim3
```

```
    Study Directory:
```

```
D:/02_Projects/CalSim/proj/2023_CalClimateInit/calsim/CalSim3_Sc  
enarios\s0002_DCR2023_9.3.1_danube_adj/Model_Files
```

```
    Main file: mainCS3_ReOrg_UWplusVF.wresl
```

```
    Configuration:
```

```
        Start Year-Month-Day: 1921-10-31
```

```
        End Year-Month-Day: 2021-9-30
```

```
        DV Filename:
```

```
DCR2023_DV_9.3.1_v2a_Danube_Adj_v1.8.dss
```

```
        SV Filename: DCR2023_SV_Danube_Adj_v1.8.dss
```

```
        DSS DV A Part: CALSIM
```

```
        DSS F Part: L2020A
```

```
#####
```

CalSimPy: The CalSim object

(1) Initialize a CalSim object using a launch file path

```
<[1]>: cs3_study = cs3.calsim(launchFP = launch_fp1,  
calsim_version=3)
```

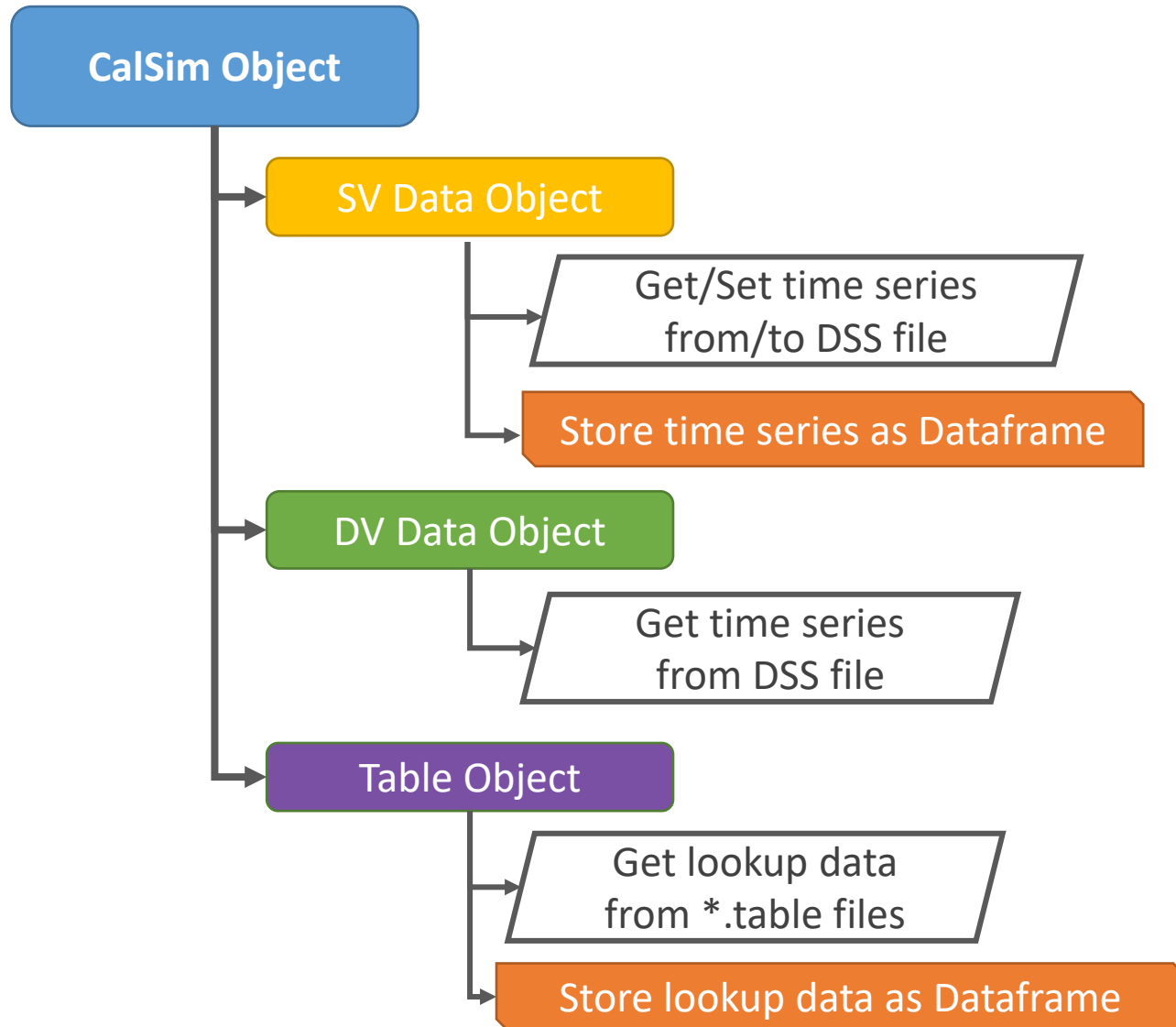
(2) Check the CalSim object

```
<[2]>: cs3_study
```

Response – subset of the study configuration info

```
#####  
CalSim object:  
    Version: CalSim3  
    Study Directory:  
D:/02_Projects/CalSim/proj/2023_CalClimateInit/calsim/CalSim3_Sc  
enarios\s0002_DCR2023_9.3.1_danube_adj/Model_Files  
    Main file: mainCS3_ReOrg_UWplusVF.wresl  
    Configuration:  
        Start Year-Month-Day: 1921-10-31  
        End Year-Month-Day: 2021-9-30  
        DV Filename:  
DCR2023_DV_9.3.1_v2a_Danube_Adj_v1.8.dss  
        SV Filename: DCR2023_SV_Danube_Adj_v1.8.dss  
        DSS DV A Part: CALSIM  
        DSS F Part: L2020A  
#####
```

CalSimPy: CalSim object – Data



- Separate “containers” for SV and DV data
- Integrated DSS interface using *yapydss* library (“yet another Python DSS” wrapper)
- Table data read/write access as well (specify by table name)

Example: Reading SV data from DSS

```

1  # get some SV data
2  bl_study.SVdata.getSVts(filter=['/I_SHSTA/']) # Let's get the Shasta inflow
   time series
3
4  # We can also filter DSS pathnames using regular expressions
5  # the `append_dataframe = True` option will add the closure term
6  # data to the new dataframe rather than create a new dataframe (the default)
7  bl_study.SVdata.getSVts(filter=["CT_\\S*_SV"], filt_method='regex',
   append_dataframe=True)
8
9  # all the retrieved timeseries data lives in
10 # be accessed using the `SVtsDF` attribute
11 svDF = bl_study.SVdata.SVtsDF
12
13 # we can confirm the data we have in the new
14 svDF.head()

```

getSVts function uses filters (a list of partial matches or regular expressions) to select DSS paths

Function stores retrieved data in an internal Pandas dataframe that retains the DSS header information

A	CALSIM						
B	I_SHSTA	CT_BENDBRIDGE_SV	CT_BUTTECITY_SV	CT_COLUSA_SV	CT_DAVIS_SV	CT_FAIROAKS_SV	CT_FREEPORT_SV
C	INFLOW	CLOSURE-TERM	CLOSURE-TERM	CLOSURE-TERM	CLOSURE-TERM	CLOSURE-TERM	CLOSURE-TERM
E	1MON	1MON	1MON	1MON	1MON	1MON	1MON
F	L2020A	L2020A	L2020A	L2020A	L2020A	L2020A	L2020A
Type	PER-AVER	PER-AVER	PER-AVER	PER-AVER	PER-AVER	PER-AVER	PER-AVER
Units	TAF	TAF	TAF	TAF	TAF	TAF	TAF
1921-1-31	231.689117	-3.560000	-37.980000	0.0	0.0	-1.63	0.000000
1-30	220.355118	-29.000000	-18.950001	0.0	0.0	0.00	21.170000
1-31	334.589569	-100.959999	-48.389999	0.0	0.0	0.00	59.660000
2-31	266.592133	-31.360001	17.520000	0.0	0.0	0.00	16.059999
1922-02-28	587.583557	9.910000	59.750000	0.0	0.0	0.00	2.690000

5 rows × 27 columns

Example: Modifying and writing SV data to DSS

- Let's say you want to change some of the SV data (e.g. reduce February Shasta inflow by 10%) and create a new DSS file with a new variable
- First – you can modify the data using operations on your dataframe

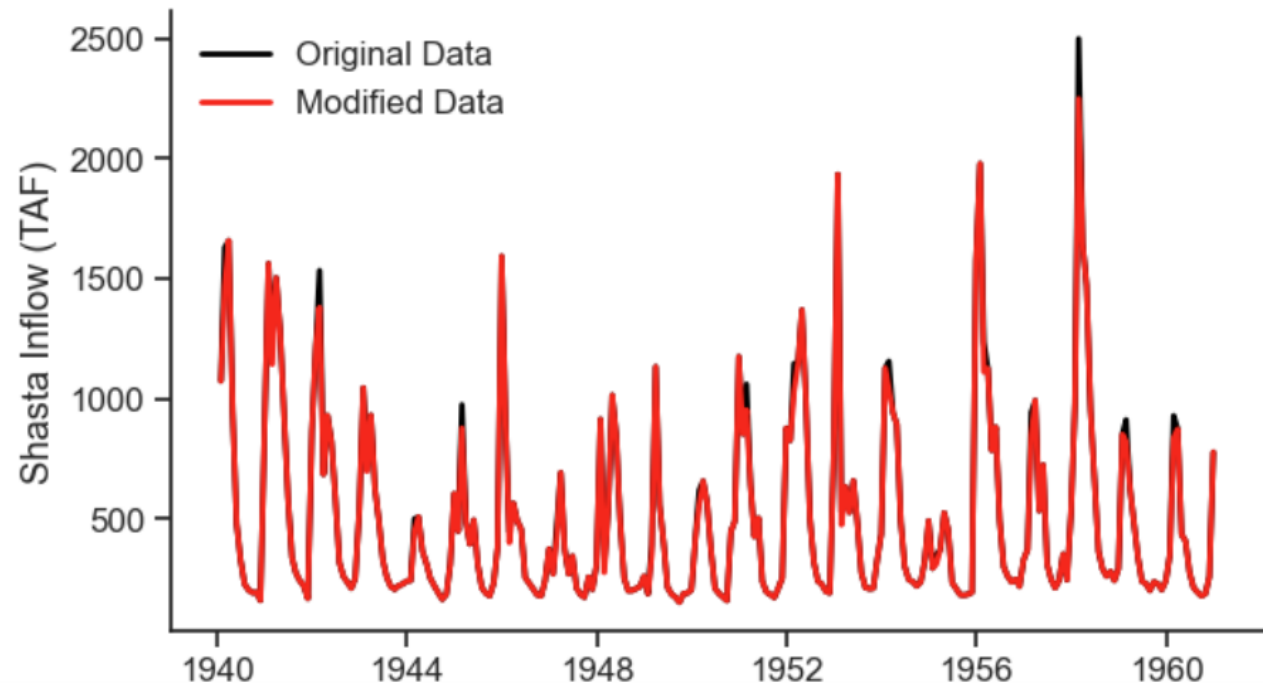
```
1  # let's reduce February inflow to Shasta by 10% as a test
2
3  # make a copy of your dataframe so we can make changes and not
4  # modify the original
5  svDF_mod = svDF.copy(deep=True)
6  |
7  # create a new variable
8  svDF_mod['CALSIM', 'I_SHSTA90pct', 'INFLOW', '1MON', 'L2020A', 'PER-AVER', 'TAF'] =
   svDF_mod.loc[:, idx[:, 'I_SHSTA' ]].values
9
10 # filter out just the Februaries
11 feb_idx = svDF_mod.index.month==2
12
13 # do the calculation
14 svDF_mod.loc[feb_idx, idx[:, 'I_SHSTA90pct']] = svDF_mod.loc[feb_idx, idx[:,
   'I_SHSTA' ]].values*0.90
```

Example: Modifying and writing SV data to DSS

- Along the way you can make a nice plot to check that you did it correctly

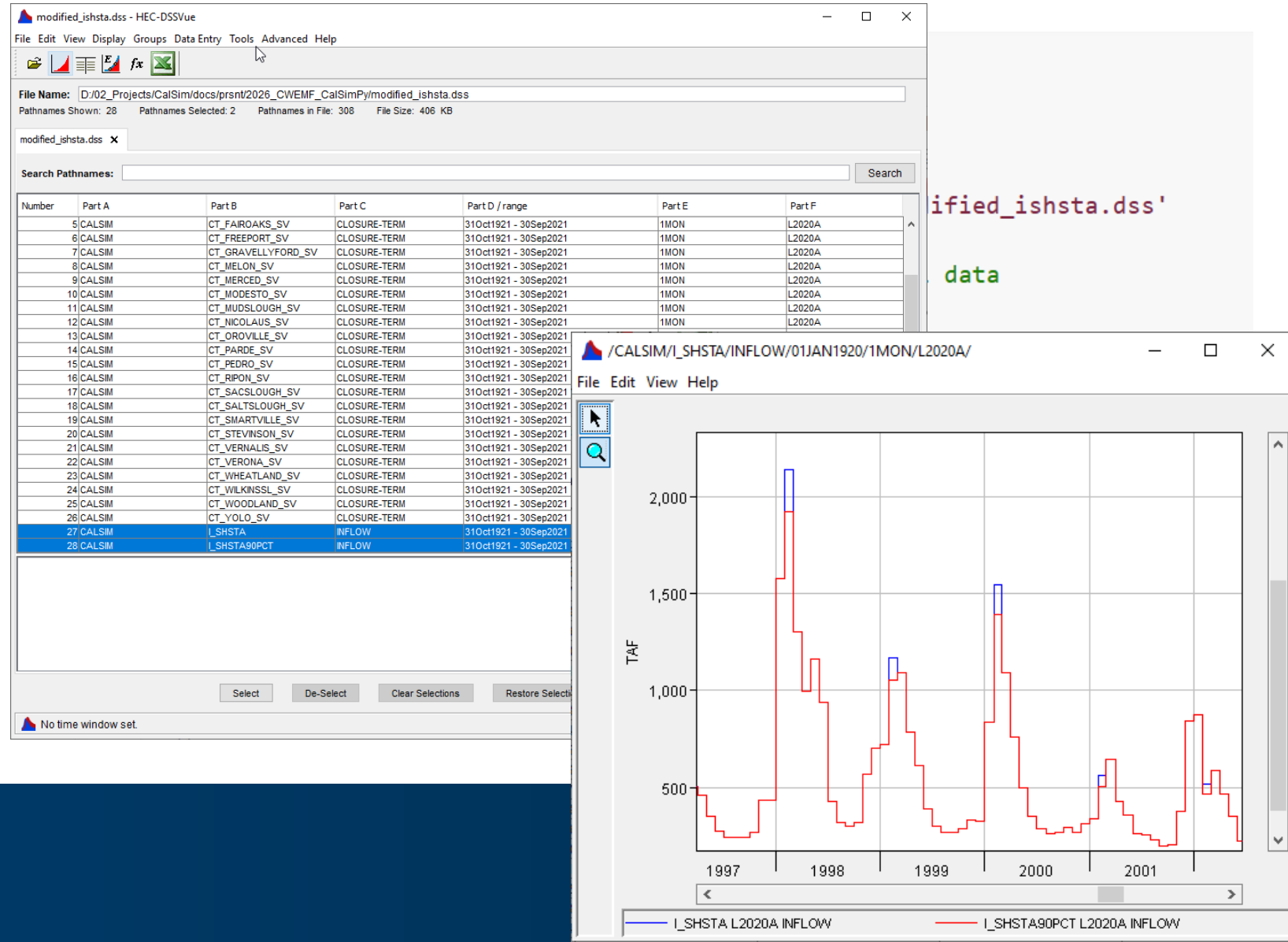
```
1 # we can plot this quick to check
2 with sns.plotting_context('notebook', font_scale=1.1):
3     fig, ax= plt.subplots(1,1, figsize=(7,4))
4
5     ax.plot(svDF_mod.loc['1940':'1960', idx[:, 'I_SHSTA']], c='k', lw=2,
6             label='Original Data')
7     ax.plot(svDF_mod.loc['1940':'1960', idx[:, 'I_SHSTA90pct']], c='red', lw=2,
8             label='Modified Data')
9     ax.set_ylabel('Shasta Inflow (TAF)')
10    plt.legend(loc='best', frameon=False)
11    sns.despine()
```

✓ 0.2s



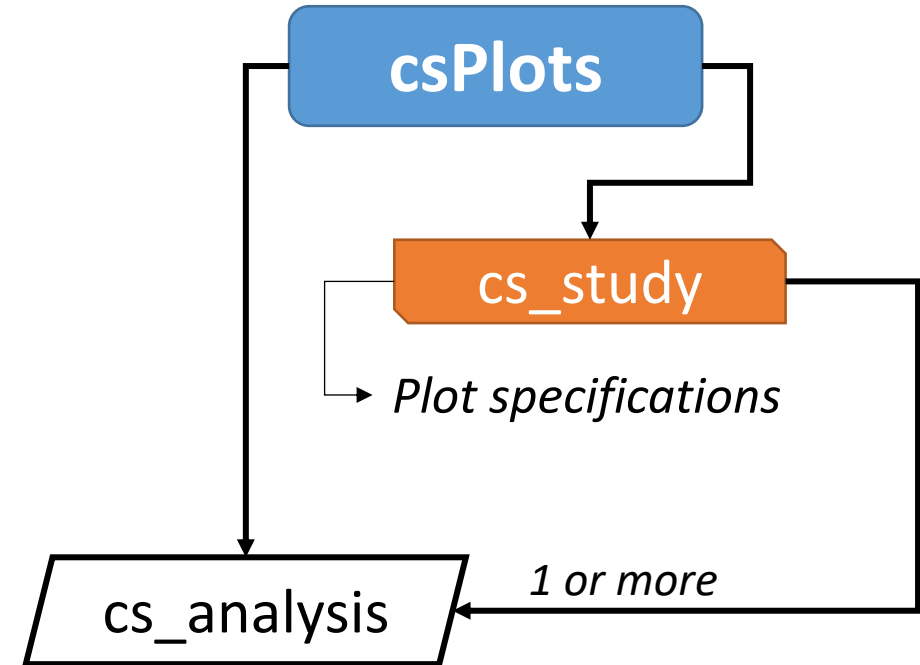
Example: Modifying and writing SV data to DSS

- Finally you can write the time series data to a new DSS file using the **setSVts** function



CalSimPy: Multi-study comparisons

- Often we want to compare data from multiple scenarios using different graphs and visualizations
- The set of functions in the **csPlots** module makes these sorts of comparisons easier
 - CalSim object → cs_study
 - cs_study → cs_analysis → data & plots!



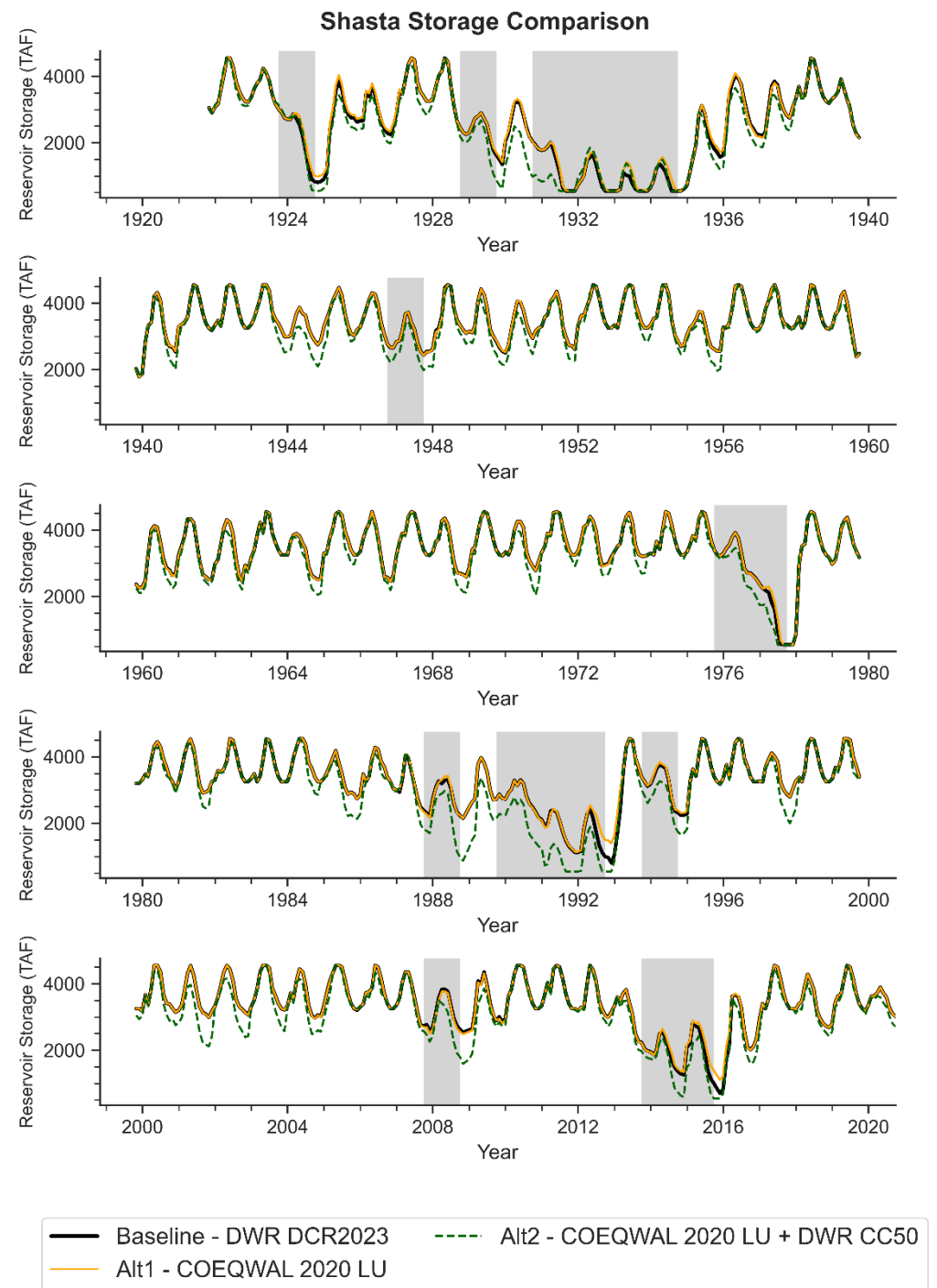
- Retrieve data from DSS files
- Time series plots
- Exceedance plots
- Pattern plots

Example: Comparing 3 Scenarios – Time series

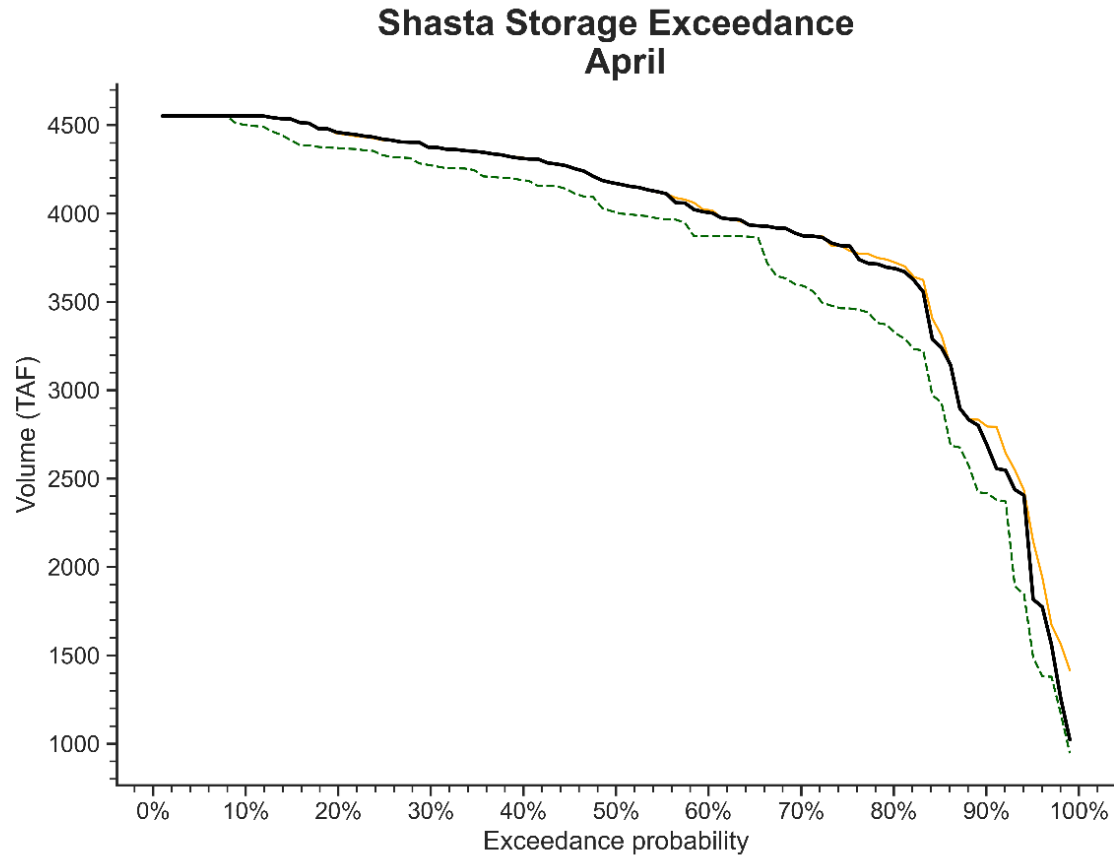
```

1 # initialize other calsim studies
2
3 alt1 = cs3.calsim(configFP = config_fp2, calsim_version=3, reorg=True)
4 alt2 = cs3.calsim(launchFP = launch_fp3, calsim_version=3, reorg=True)
5
6 # create a cs_plots study for each - set a line color, name, and description
7 # line color and labels will be used across all plot types
8 bl_study_plot = csplt.cs_study(bl_study, "Baseline - DWR DCR2023",baseline=True,
9                               color="k", desc="DWR's 2023 DCR baseline with
10                                adjusted historical hydrology")
11 alt1_study_plot = csplt.cs_study(alt1, "Alt1 - COEQWAL 2020 LU",baseline=False,
12                                color="orange", desc="COEQWAL version of 2023 DCR
13                                baseline with adjusted historical hydrology and 2020
14                                updated land use")
15
16 #make an "analysis" with the studies
17 an1 = csplt.cs_analysis([bl_study_plot, alt1_study_plot, alt2_study_plot])
18
19 # get data for all scenarios in the study
20 an1.getDV(filter=['/S_SHSTA/', '/DEL_CVP_PAG_S/'])
21
22 # set which variable you want to plot
23 an1.stageVar('S_SHSTA', 'DV')
24
25 # call the `plot_multi_monthlyTS` time series plotting function
26 fig_dict = an1.plot_multi_monthlyTS('Shasta Storage Comparison',
27                                     highlight_crit=True)

```

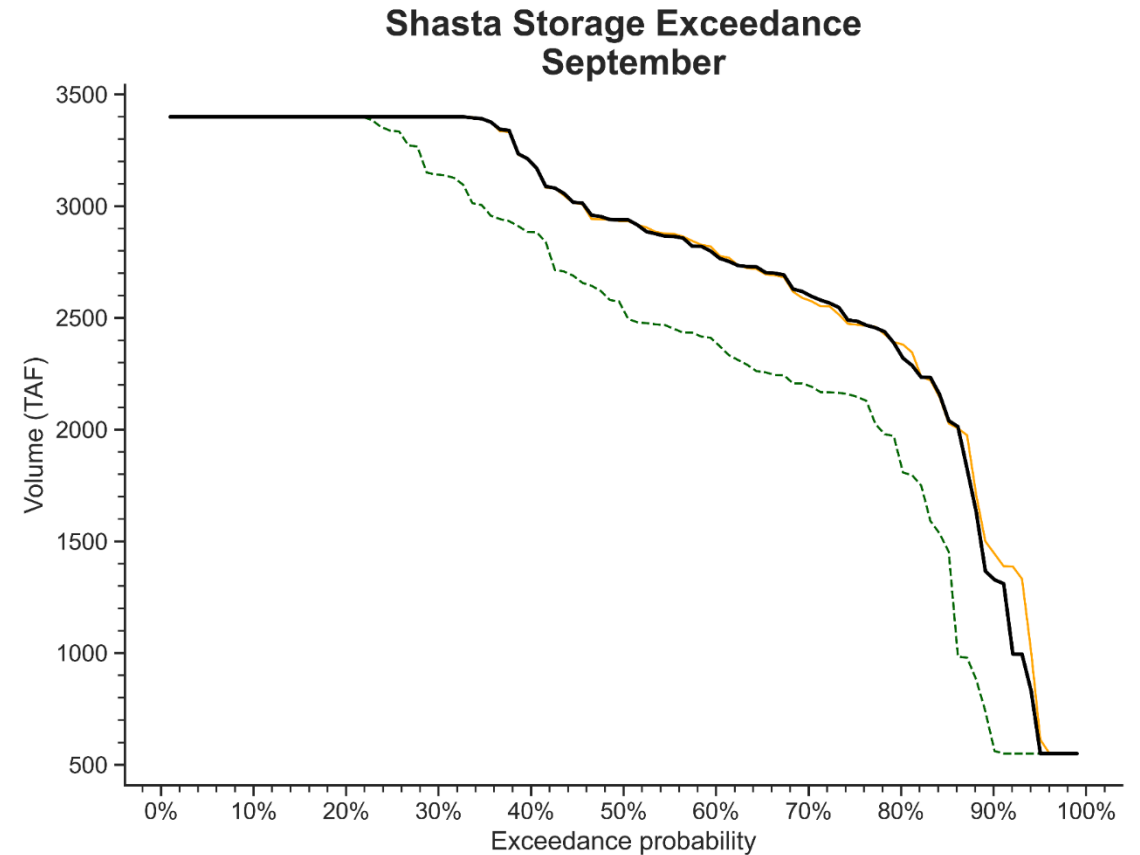


Example: Comparing 3 Scenarios – Exceedance - Select Months



— Baseline - DWR DCR2023 - - - Alt2 - COEQWAL 2020 LU + DWR CC50
— Alt1 - COEQWAL 2020 LU

```
1 fig_dict = an1.plot_annual_exceed('Shasta Storage', month_filter=[4],  
2 metric_axis=False,  
3 annualize_on='A-Sep')
```



— Baseline - DWR DCR2023 - - - Alt2 - COEQWAL 2020 LU + DWR CC50
— Alt1 - COEQWAL 2020 LU

```
1 fig_dict = an1.plot_annual_exceed('Shasta Storage', month_filter=[9],  
2 metric_axis=False,  
3 annualize_on='A-Sep')
```

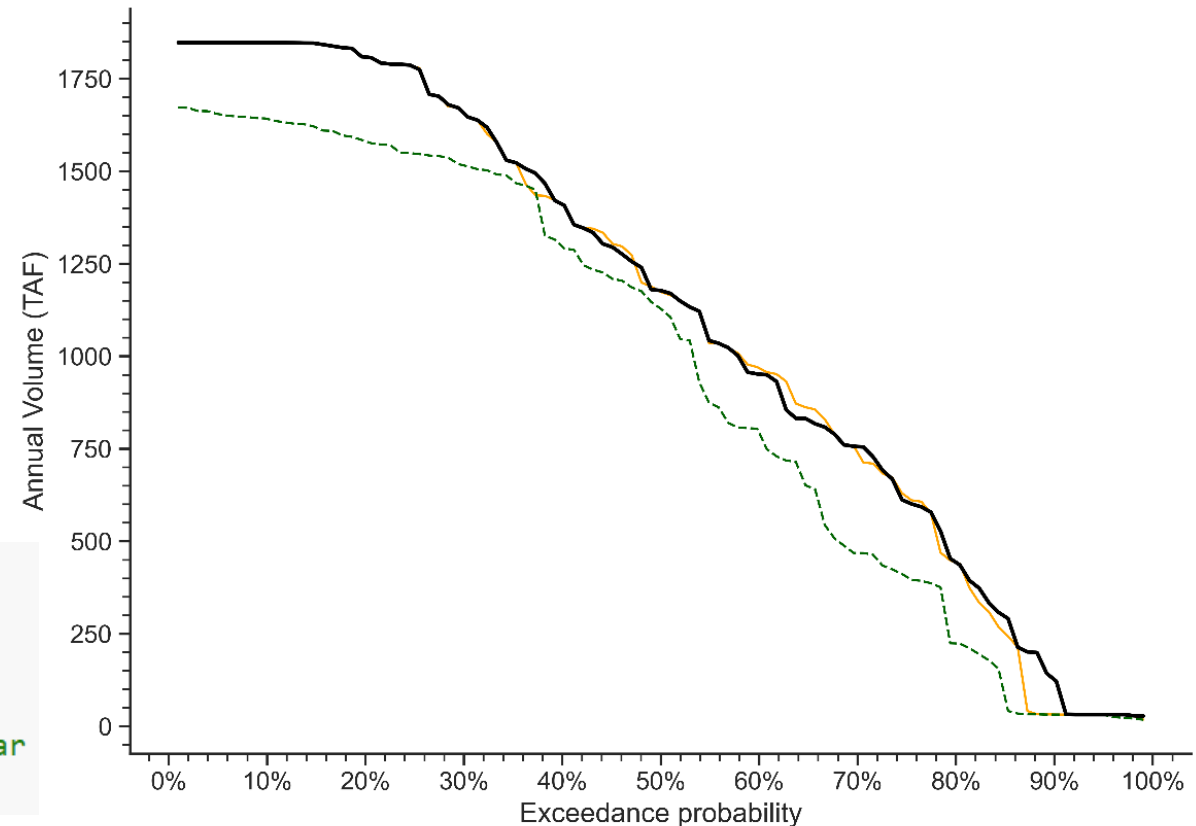
Example: Comparing 3 Scenarios – Exceedance - Annual Volumes

Automatically converts from CFS to TAF and aggregate to an annual period (e.g. Oct – Sep, Mar – Feb, Calendar Year)

```
an1.unstageVar('S_SHSTA') #<-- unload the old variable
an1.stageVar('DEL_CVP_PAG_S', 'DV') #<-- select a new variable

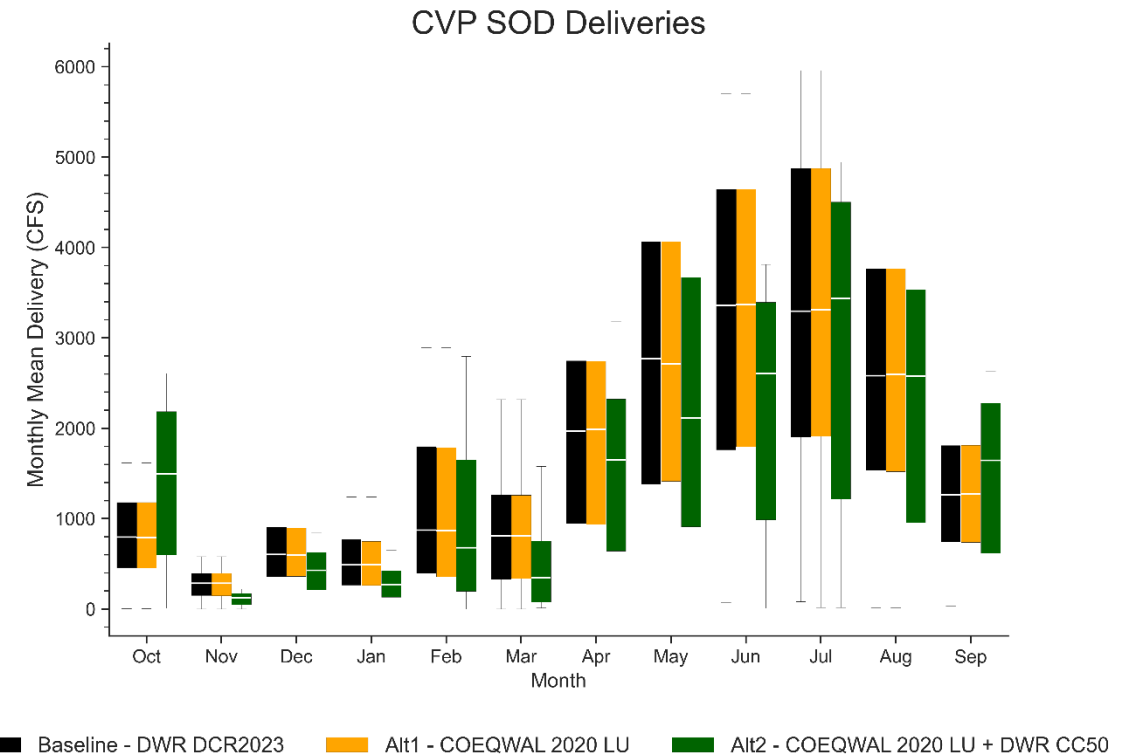
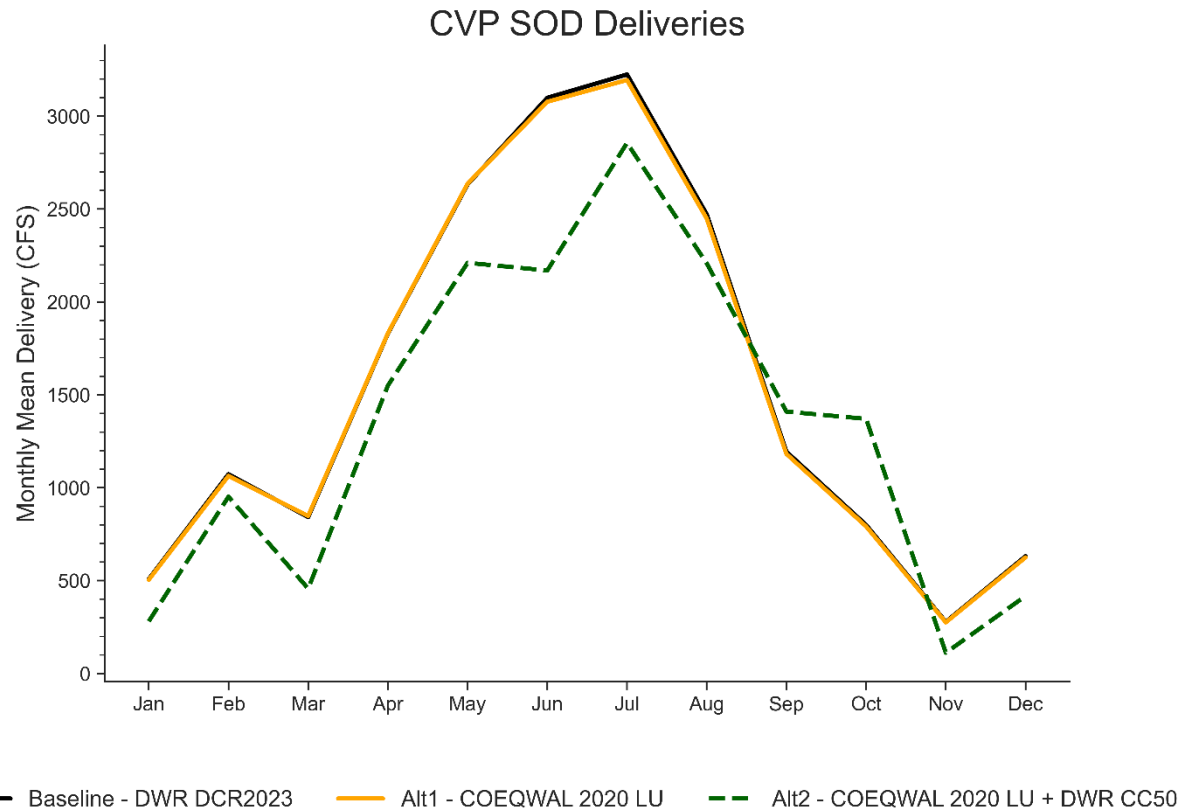
fig_dict = an1.plot_annual_exceed('CVP SOD Deliveries',
                                  annualize_on='A-Feb', #<-- Contract year
                                  metric_axis=False)
```

CVP SOD Deliveries Annual Exceedance



— Baseline - DWR DCR2023 - - - Alt2 - COEQWAL 2020 LU + DWR CC50
— Alt1 - COEQWAL 2020 LU

Example: Comparing 3 Scenarios – Monthly Patterns



```

1 # Make a monthly pattern plot - use the average values, no filtering
2 fig_dict = an1.plot_pattern('CVP SOD Deliveries', 'Monthly Mean Delivery',
3                             pattern_type='MarOct', annual_filter=[],
4                             annual_filter_type="",
5                             plotType='avg')

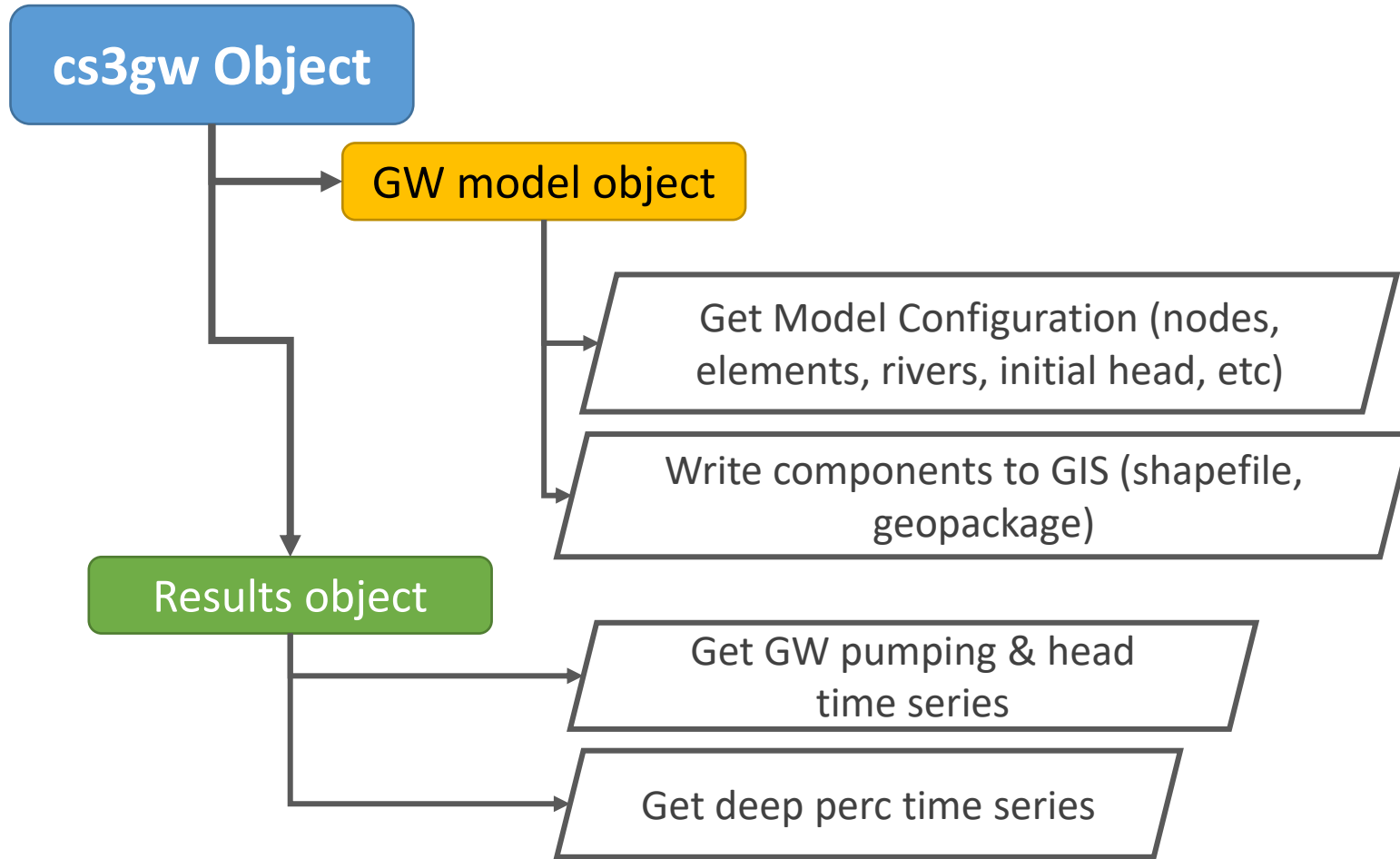
```

```

7 # Make a monthly pattern plot - show variability with box plot, no filtering
8 fig_dict = an1.plot_pattern('CVP SOD Deliveries', 'Monthly Mean Delivery',
9                             pattern_type='MarOct', annual_filter=[],
10                            annual_filter_type="",
11                            plotType='box')

```

CalSimPy: Working with the C2VSim groundwater model



- Read in the C2VSim (CS3 DLL version) data via a cs3gw object
- Can get all the model components (boundary condition, nodes, elements, etc)
- Can read in the data from DSS, write spatial components to GIS

Example: Extract and Plot Groundwater Data as a Map

```
from calsimpy.cs3gw import cs3gw
from calsimpy.cs3gw import gwResults

import datetime as dt

gw1 = cs3gw(bl_study.projDir)

# read in the nodes and layers configuration
gw1.getNodes()
gw1.getStrat()

# read in the elements configuration
gw1.getElements()

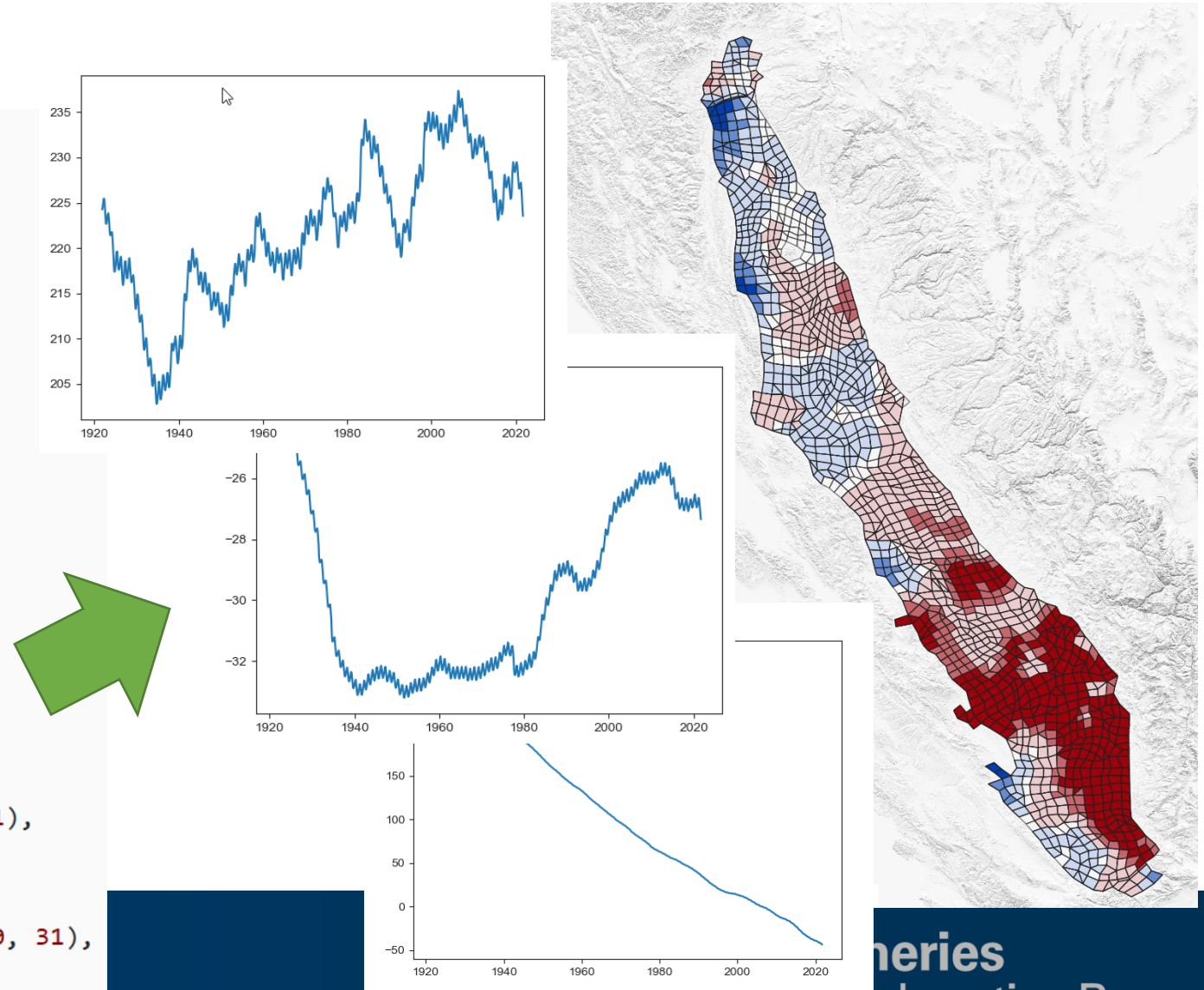
# create a geodataframe of the elements
gw1.elem2GIS()

# get the geodataframe
elemGDF = gw1.ElementsGIS

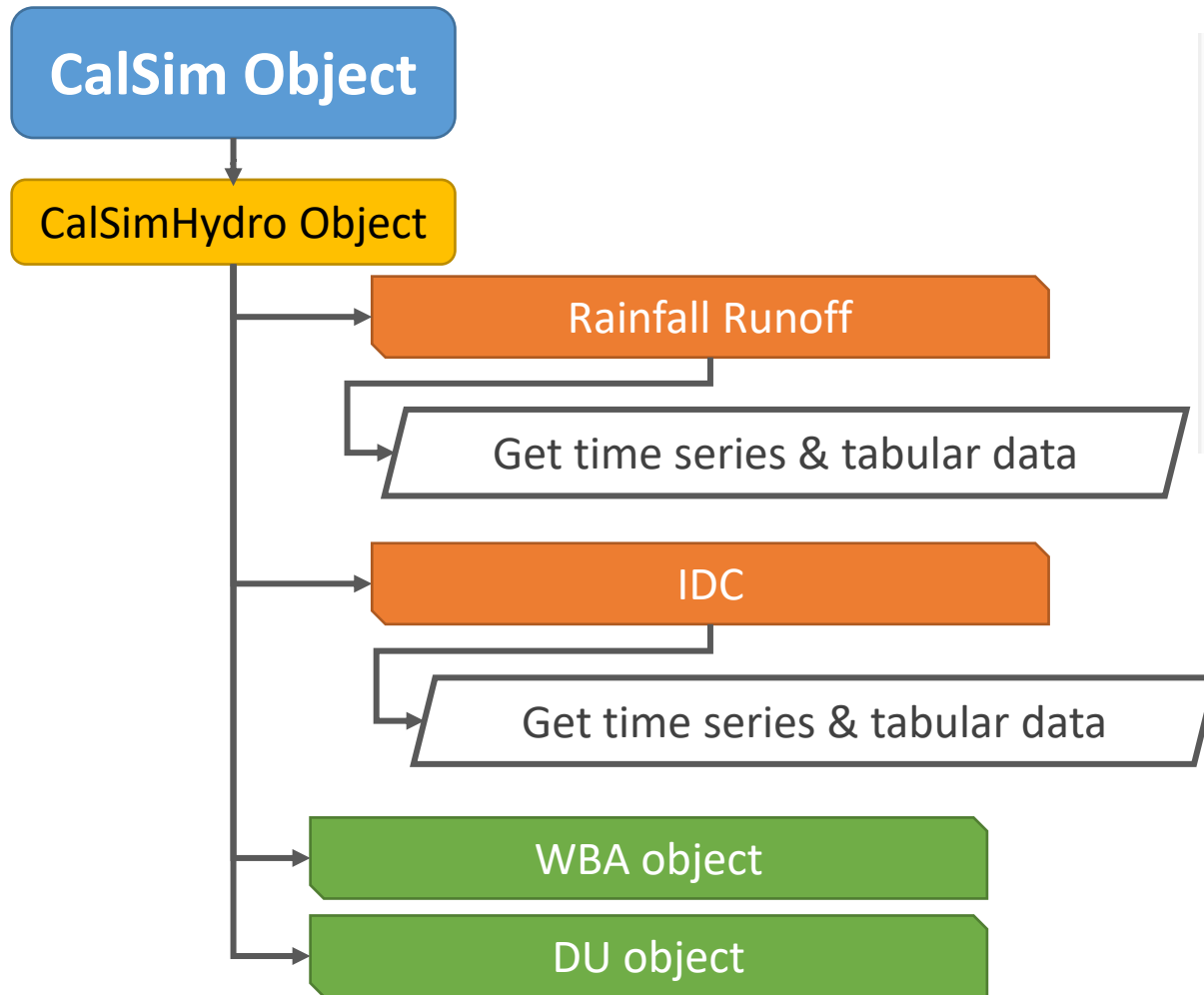
# get the results
gw1_results = gwResults(gw1)

gw1_results.getGWHead(gw_startDate=dt.datetime(1921, 10, 31),
                      gw_endDate=dt.datetime(2021, 9, 30),
                      reorg=True)

gw1_results.getGWDeepPerc(gw_startDate=dt.datetime(1921, 10, 31),
                           gw_endDate=dt.datetime(2021, 9, 30),
                           reorg=True)
```



CalSimPy: CalSimHydro too!



```
idcfp = '/CalSimHydro/2a_IDC/Main.dat'

thiscsh = cs3.calsimhydro() #<-- create a CSH object
thiscsh.IDC = cs3.idc(idcfp) #<-- initialize and IDC boject

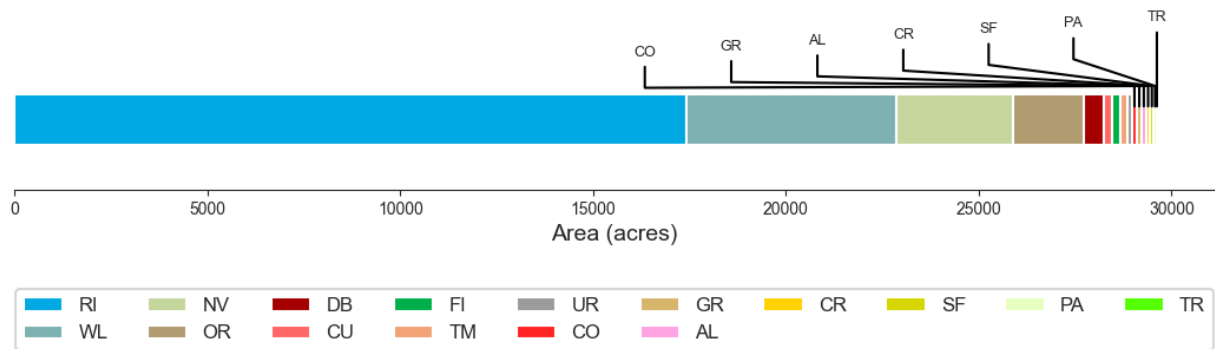
thisIDC = thiscsh.IDC
thisIDC.init_DU_list() #<-- get DU info

thisIDC.get_land_use() #<-- retrieves land use for all DUs
```

- Example use cases:
 - Easily adjust land use by DU (or whole domain)
 - View, analyze, or summarize meteorological inputs

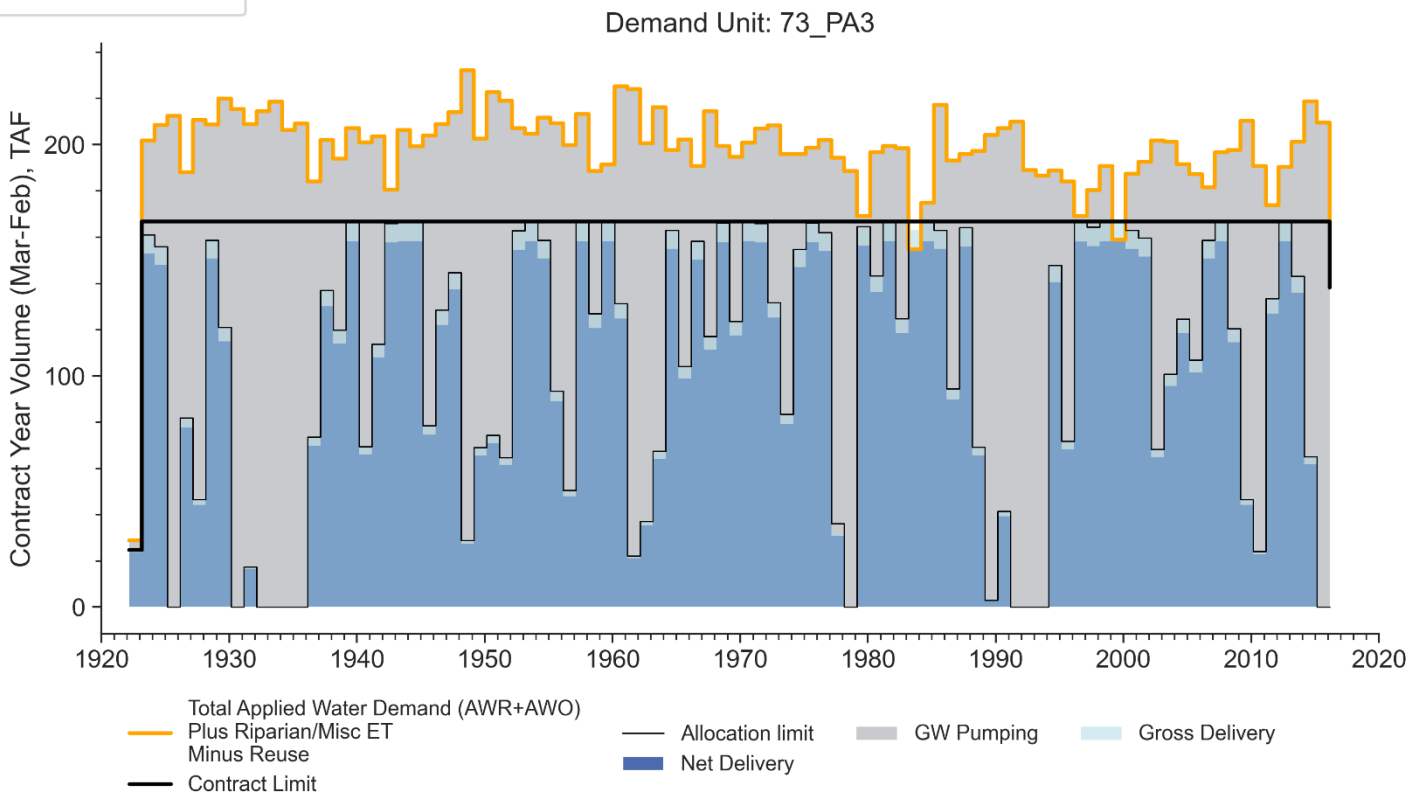
Examples: CalSimHydro

Land use in demand unit: 09_SA2



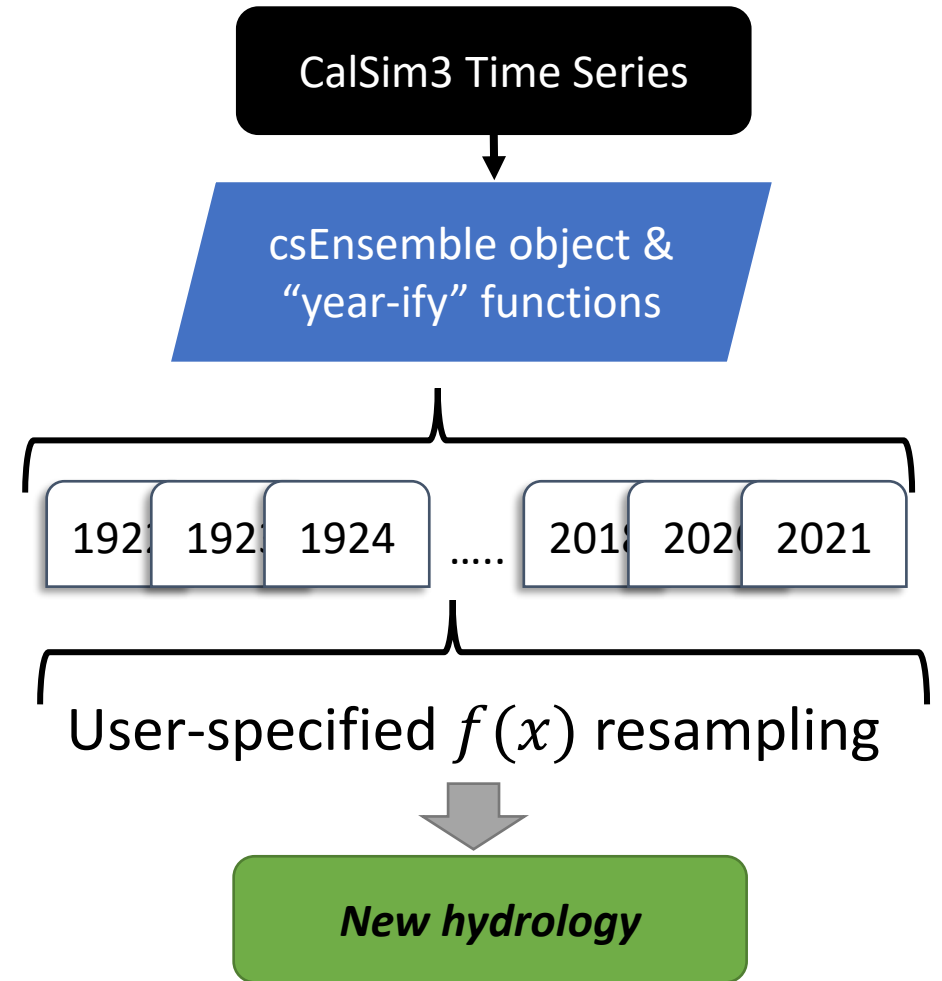
A land use summary for each Demand Unit

Combine CalSim3 and CalSimHydro data – delivery and water budget summary plots



Other applications: Custom hydrology

- Re-sequencing CalSim3 input hydrology
- Example – analysis completed for a project funded by California Department of Fish and Wildlife
- Concept – CalSim time series are split into yearly blocks
- Any custom sequence can be created



Example: Exploring novel drought sequences

- Synthetic annual sequence generated through a separate statistical method
- New selection of years becomes input to the code (e.g. [1987, 1943, 2018, 2014, 1977,...])
- **shuffle_ts** function handles the re-sequencing of CalSim yearly blocks

```
resampyr_fp = '/path/to/new/annual/hydrology/sequence.csv'

# Load a CalSim3 base model
thiscs = cs3.calsim(base_launchFP)

# create an ensemble object from
# the CalSim model
ens1 = cs3.ensembleCS(thiscs)

# Load in the new sequence of years
resamp_years_df = pnd.read_csv(resampyr_fp)
# convert it to a list
resamp_years_list = list(resamp_years_df.NewSequence)

# split the base data into yearly blocks
ens1.prepare_years()

# get a new dataset of re-ordered CalSim3 years
reordyr, reorddat, shufflyrs, seqYrs = \
    ens1.shuffle_ts(shuffle_list=resamp_years_list,
                    exclude=[1919, 1920, 1921, 2022])
```

Example: Exploring novel drought sequences

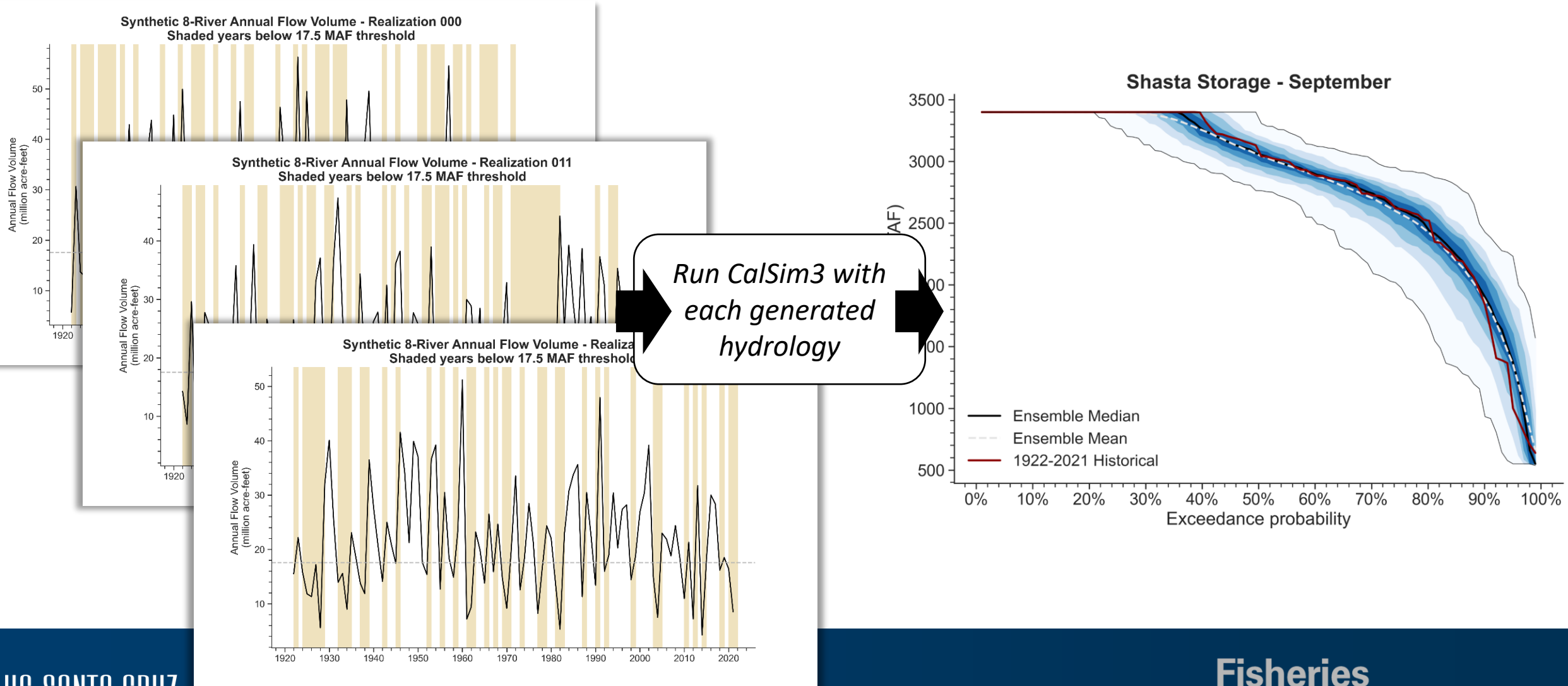
```
# combine the `reorddat` into a new dataframe
big_list = []
for yr, yd in zip(reordyr, reorddat): # Loop through the keys
    #print(yr)
    if type(yd.Other) == type(None):
        dum = seqYrs.pop(seqYrs.index(yr))
    else:
        big_list.append(yd.Other)
seqDateIndex = pnd.date_range(dt.date(seqYrs[0]-1, 10, 31),
                              dt.date(seqYrs[-1], 9, 30),
                              freq='M')

newDF = pnd.concat(big_list, ignore_index=False, axis=0)
newDF.set_index(seqDateIndex, inplace=True)

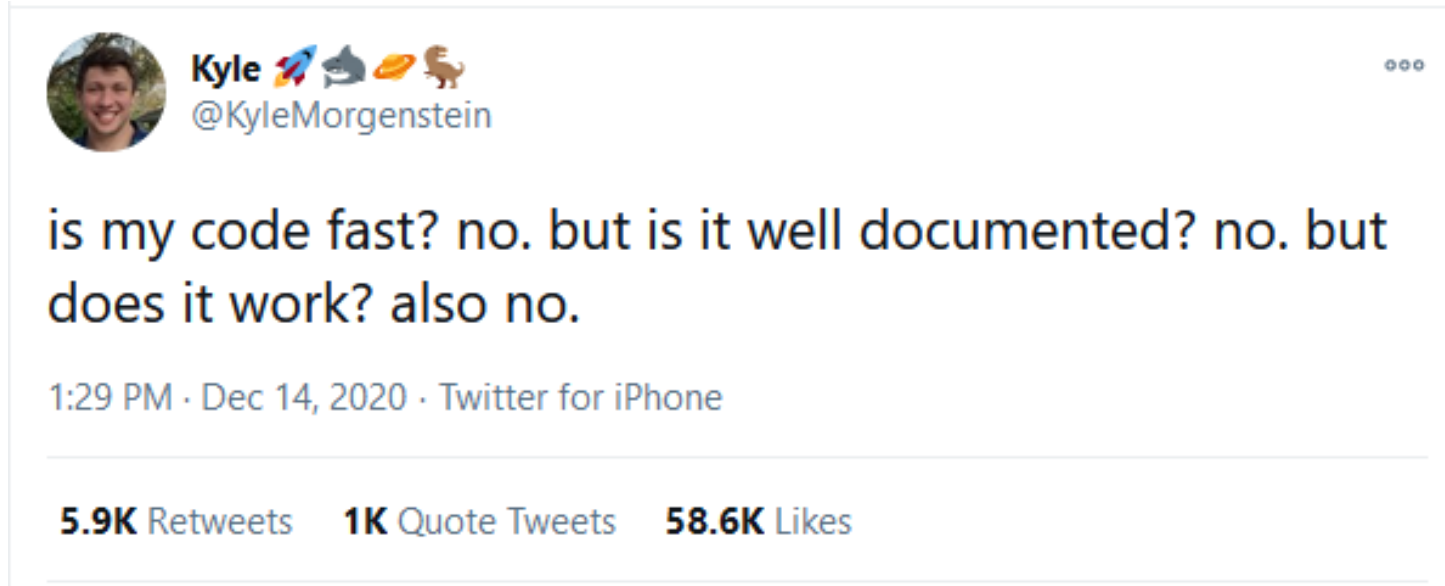
# add the new data to the CalSim object,
# then write it to DSS
shuff01 = cs3.csSVdata(thiscs.LaunchFP)
shuff01.SVtsDF = newDF
shuff01.setSVts(newdssfp, debug=False)
```

- Combine the yearly blocks into a big dataframe again
- Set that as the SV object
- Write to a new SV DSS file
- Repeat as needed!

Example: Exploring novel drought sequences



There is a lot left to do...



- Fill in missing functionality (e.g. with CalSimHydro)
- Improve consistency across components
- Documentation!

There is a lot left to do...but hopefully the code in its current form can be helpful

- CalSimPy code (warts and all!) is available on GitHub: [**github.com/james-m-gilbert/calsimpy**](https://github.com/james-m-gilbert/calsimpy)
- A new round of updates coming soon
- Documentation & examples are also in the works
- Questions and constructive critiques welcome – [**james.gilbert@ucsc.edu**](mailto:james.gilbert@ucsc.edu)



Thanks!