OPTIONAL, if processed data not available:

1. Run\_downloadData.py – downloads relevant data from FRF (and WIS) thredds server and saves them to a local directory
2. Run\_processData.py – process local data to align to time series of the same period and sampling interval

Now, do the analysis:

1. Run\_defineStorms.py – uses offshore and WIS data to define storm conditions and create time series of “storminess” (stormy = 1, not-stormy = 0) to filter ML input datasets, if desired
   1. FRF wave data used for analysis comes from waves\_FRF.pickle
   2. WIS data used for analysis comes from waves\_WISandFRF.pickle
   3. Saves output data to stormy\_times\_fullspan.pickle; obj = [time\_fullspan,stormy\_fullspan,storm\_timestart\_all,storm\_timeend\_all]
2. Run\_cleanLidarProfiles.py – does a lot of removal/smoothing of weird data in profiles
   1. Lidar comes from IO\_alignedintime.pickle
   2. X-coordinate in separate file lidar\_xFRF.pickle
   3. Briefly uses lidar statistics from IO\_lidarquality.pickle and IO\_lidarhydro\_aligned.pickle
   4. Final acceptable profiles saved as cleanLidarProfiles.pickle; obj = [lidar\_xFRF,time\_fullspan,final\_profile\_fullspan\_best]
3. Run\_blendLidarBathy.py – combines best-available lidar and available bathy surveys and fills nans; blend between lidar and bathy overlap of 0.5m
   1. Lidar comes from cleanLidarProfiles.pickle
   2. Bathy comes from tidalAveragedMetrics.pickle
   3. Combined output saved to blendedLidarBathy.pickle; obj = [lidar\_xFRF, time\_fullspan, ZbFull\_addLidar]
   4. Relies on interpgap.py located in funcs/
4. Run\_prepDatasets.py – fills small gaps in bathy and hydro data (gaps remain in both); shifts profiles in x-shore (slightly) so they all start at x = 0, z = 6m
   1. Input topobathy comes from blendedLidarBathy.pickle
   2. Hydro timeseries come from IO\_alignedintime.pickle
   3. Final output saved as preppedHydroTopobathy.pickle; obj = [lidar\_xFRF, time\_fullspan, topobathy\_fullspan\_gapfilled, xplot\_shift, topobathy\_fullspan\_gapfilled\_shift]
5. Run\_PCAdecomp.py - isolates profiles with sufficient length and reasonable (< 5m^3) hrly volume change, then performs PCA on isolated profiles
   1. Input topobathy comes from preppedHydroTopobathy.pickle
   2. Final output saved as PCAoutput.pickle; obj = [xplot, time\_fullspan, dataNorm\_fullspan, dataMean,dataStd,PCs\_fullspan,EOFs,APEV]
6. Run\_LSTMpcaprofiles – develops and tests LSTM model for profiles derived from PCA-decomposition; includes routines for filtering to remove large changes in PC amplitudes; performs scaling of parameters before creating LSTM model; includes code for testing time-series prediction based on previous prediction but seeded with data
   1. Input hydrodynamic predictors come from preppedHydroTopobathy.pickle
   2. Input/output PC amplitudes for testing/training come from PCs\_fullspan.pickle
7. Run\_LSTMbeachstats – develops and tests LSTM model for beach statistics (contour position Xc, beach width, beach volume, etc.); performs scaling of parameters before creating LSTM model; includes code for testing time-series prediction based on previous prediction but seeded with data
   1. Input hydrodynamic predictors come from preppedHydroTopobathy.pickle
   2. Input/output beach profiles used to test/train on beach stats can be either observed profiles or PC-reconstructed, both of which can be built from PCs\_fullspan.pickle
   3. Beach stats are calculated with home-built functions calculate\_beachvol.py and create\_contours.py located in funcs/