What is the performance and transferability of LSTM prediction of streamflow across different hydrologic basins?

Robert 'Quinn' Hull
Department of Hydrology and Atmospheric Sciences
University of Arizona
Tucson, Az, USA
roberthull@email.arizona.edu

***Abstract*—This study presents a Machine Learning (ML) approach to learn streamflow dynamics in the Upper Colorado River Basin (UCRB). Long Short-Term Memory (LSTM) statistical emulators are used to predict daily streamflow across a range of hydrologic conditions for five headwater streams in the UCRB. First, we assess how well specialized LSTM models trained on only one stream perform on out-of-sample testing datasets that come from the same stream. Then, we explore how well the specialized models transfer to test datasets that come from other streams. Finally, we compare the performance of the specialized cases to a generalized LSTM model that is trained on all headwater streams together. We find that the best overall performances come from the specialized LSTM models tested on the same domains upon which they were tested. However, in some situations models trained on other domains perform best - particularly when predicting outlier (very high and very low flow) conditions. Our results are an important test case in showing how transfer learning from novel data-driven approaches, like LSTM, can be used to make hydrologically relevant predictions.)**

Keywords—streamflow prediction, LSTM, transfer learning

I. Introduction

The field of hydrology has a long history of applying machine learning approaches to the task of predicting the behavior of complex natural systems [1]. One particularly important task for hydrologists remains the prediction of streamflow. These predictions are socially and environmentally relevant because they tell us when we can recreate in our rivers, when and where they will flood and damage property, and how often there will not be enough water for human use. Due to the challenges of data-collection, a traditional approach in the field has been to collect time-series of streamflow at one point over time and use data-driven approaches (like regression) to predict streamflow time-series into the future at socially relevant time-scales (weekly, monthly, bi-annually, yearly). However, deep learning approaches to streamflow prediction are a burgeoning subfield of hydrology [2, 3]. This study uses LSTM models trained using different combinations of dynamic climate and riverine variables to predict streamflow under outlier low- and hi- flow conditions, and make a novel contribution to this new subfield.

II. Datasets

A. Study Domain

The data used in this investigation represent the Upper Colorado River Basin (UCRB). The UCRB is socially important due to the fact that it contributes water used by the entire Southwest of the United States. The UCRB is scientifically interesting because it is an especially dynamic hydrologic system in which streamflow behavior is dominated by snowpack melt-off in late April of every year.

Five watersheds that are coincident with gaged (monitored) locations were selected for further study. Their locations are shown in Figure 1. These locations were selected because they represent 'headwater' conditions at different locations throughout the UCRB. If we were to think of this as a classification problem, these basins would represent the same class - i.e. we might hypothesize these basins would respond similarly to identical climatic and streamflow conditions. This decision was made to facilitate an 'apples to apples' comparison.

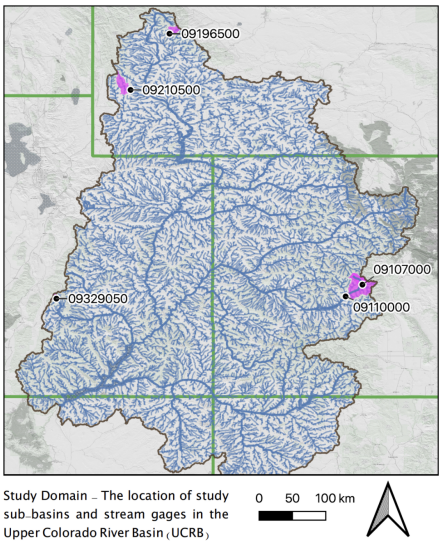


Figure 1: *The locations of USGS stream flow locations (black) and their upstream watersheds (pink) within the Upper Colorado River Basin (brown outline). Green lines are the state outlines.*

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

B. *Datasets and Source*

Two types of datasets are used in this study. The overall period of record ranged between 1983 and 2020. However, to minimize the impact of some incomplete datasets the period of analysis explored in this paper was truncated to 1987 - 2006.

The first dataset is publicly available streamflow data collected by the US Geological Survey [5]. These data are time-series data at streamflow locations, and thus can each be thought as `[1, d]` in shape, where `d` is the number of days in the dataset. The five locations are described briefly in Table 1 below.

The second data source is climate data from a spatially distributed dataset spanning the continental United States [6]. The data were resampled prior to analysis in the following way. For each pink subbasin, the original data took the shape `[n, m, h]`, where `n` and `m` are the length and width of the subbasin and `h` is the number of hours in the time series. They were resampled to the shape `[1, d]` (identical to streamflow shape, where `d` is the number of days) for ease of loading into the LSTM. These resampling procedures are not shown in the code appendix. The climate data are described in the table 2 and figure 2 below.

Table 1: a) *describes the streamflow locations and periods of record. Part* ***b)*** *shows representative streamflow statistics. Streamflow is normally reported in units of discharge - cubic feet per second. However, here discharge is reported as a scaled value between 0 and 1, where the min max scaling is determined by the overall minimum and maximum of flow data over all the datasets. The data used in LSTM analysis is all scaled in this way.*

Table 1a)

Station ID	Period of Record	Name	Description
9110000	1986-2021	TAYLOR RIVER AT ALMONT, CO.	Baseline
9329050	1986-2021	SEVEN MILE CREEK NEAR FISH LAKE, UT	Smallest Headwater
9196500	1990-2020	PINE CREEK ABOVE FREMONT LAKE, WY	Northern Headwater
9107000	1989-2020	TAYLOR RIVER AT TAYLOR PARK, CO.	Overlaps with Baseline
9210500	1990-2021	FONTENELLE C NR HERSCHLER RANCH, NR FONTENELLE, WY	Similar in size to 9107000

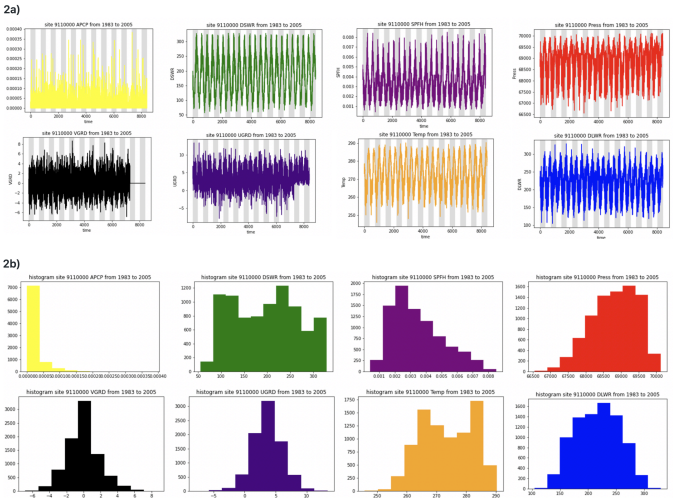
1b)

Station ID	minimum	10% exceedence	50% (median)	90% exceedence	maximum
9110000	0.02538	0.04420	0.07689	0.225832	1.0
9329050	0.0	0.00089	0.00210	0.00868	0.08416
9196500	0.00194	0.00424	0.01514	0.20121	0.92331
9107000	0.00666	0.01191	0.01998	0.10394	0.45106
9210500	0.00125	0.00626	0.01110	0.05146	0.28678

Table 2: *Description of the climate data types and units used in the analysis*

Parameter	Description
APCP	Precipitation Rate *mm/s*
Temp	Air Temperature *K*
DSWR	Downward Visible or Short Wave Radiation *W/m^2*
SPFH	Water-vapor specific humidity *kg/kg*
UGRD	West-to-East or U-component of wind *m/s*
DLWR	Downward Infrared or Long Wave Radiation *W/m^2*
Press	Atmospheric Pressure *p*
VGRD	South-to-North or V-component of wind *m/s*

Figure 2: *The __a)__ time series and __b)__ histograms of climate forcing data over the study from 1983 - 2006. Data shown only for site `09110000`, but distribution representative of other sites as well. Note that all data were scaled to between 0 and 1 prior to using machine learning methods.*



III. Statistical Approach

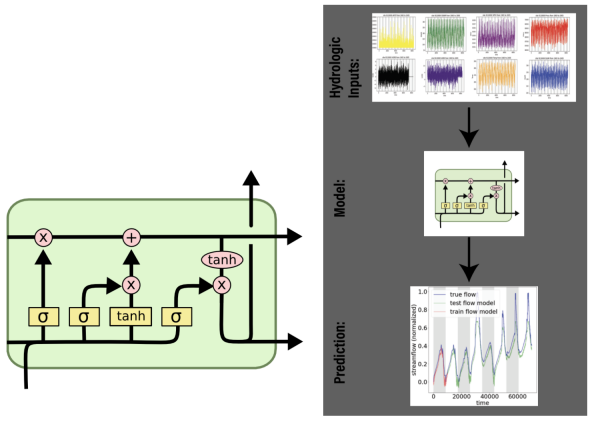
A. *LSTM Method*

The LSTM method is a deep learning approach often applied to sequential learning problems that require memory. One common application is in text prediction, however it has been used frequently in hydrology to generate time series predictions as well [3].

B. *Study Design*

This study uses a pytorch LSTM model that passes snippets of a historical climate dataset into an LSTM model to predict streamflow at a point in time in the future. A conceptual model of the study design and LSTM model are shown below in figure 3.

Figure 3: *At left, a conceptual model of study design. At right, a generalized diagram of LSTM model architecture. The LSTM takes in time-varying historical observations of streamflow and climate to make predictions of streamflow in the future.*



More specifically, this study unfolds as a comparison between three different approaches to training and testing our LSTM model on the streamflow and climate forcing datasets. These approaches are given representative names for ease of reference.

1. **Dorothy in Kansas:** Specialized LSTM models (n=5) trained on only one stream and tested on out-of-sample testing datasets that come from the same stream.
   - This approach is meant to demonstrate how well an LSTM model performs on a test dataset ostensibly representing the same population upon which it was trained (i.e. how well Dorothy navigates the challenges of being in Kansas)
2. **Dorothy in Oz**: Specialized LSTM models (n=5) transfer to test datasets that come from other streams.
   - This approach shows how well an LSTM model performs on a test dataset that doesn't represent the same population upon which it was trained (i.e. taking Dorothy out of Kansas and asking her to apply the lessons of her youth in a magical, and strange world of Oz). This could be thought of as transfer learning.
3. **Dorothy comes back to Kansas**: Generalized LSTM model (n=1) trained on all headwater streams together and tested on each stream individually.
   - This approach assesses if a generalized model trained on data from all 5 headwater streams can perform just as well as a specialized model at a stream of interest. (i.e. how well Dorothy reintegrates into life in Kansas after she's learned the lessons Oz has taught her)

C. *Design and Hyper-Parameters*

To limit the complexity of this exercise, the hyperparameters, predictive features, train-test split, date range of investigation, neural network architecture, and metrics of success are kept constant for all approaches.

HYPERPARAMETERS (ETC..)

Table 3: *Some preliminary tuning (not discussed in this report) led to the following decisions re: hyperparameters.*

(Hyper)parameter	Value	Description
`test_frac`	0.2	fraction of data to include in training set (1=all,0=none, remainder in test set)
`num_epochs`	500	number of times iterating through the model
`learning_rate`	0.001	rate of learning
`input_size`	9	nodes on the input of LSTM (should be number of features in X)
`hidden_size`	10	number of nodes in hidden layer
`num_layers`	1	number of hidden layers
`num_classes`	1	nodes in output (should be 1, i.e. the scalar value predicted)
`batch_size`	100	size of batch

LSTM Network Architecture

The pytorch (`torch`) LSTM neural network structure is shown in the code in `Appendix 2`. It is a vanilla LSTM that doesn't incorporate any form of regularization, and uses only one hidden layer.

Sample Data Range

1987 to 2006

Predictive (X) and Predicted (Y) variables

- Predictive variable features are: `['DLWR', 'DSWR', 'Press', 'APCP', 'Temp', 'SPFH', 'UGRD', 'VGRD', 'Flow_Real']`. This represents the entire suite of available climate forcings, plus streamflow.
- Predicted variable feature is: `Flow_Real'

Metrics of Success

The metrics of success (`R_2`, `RMSD`, `NSE`, and `KGE`) are evaluated along three dimensions for each test case.

1. **All Streamflow** - Goodness of match between `y_pred` and `y_real` for the entire distribution.

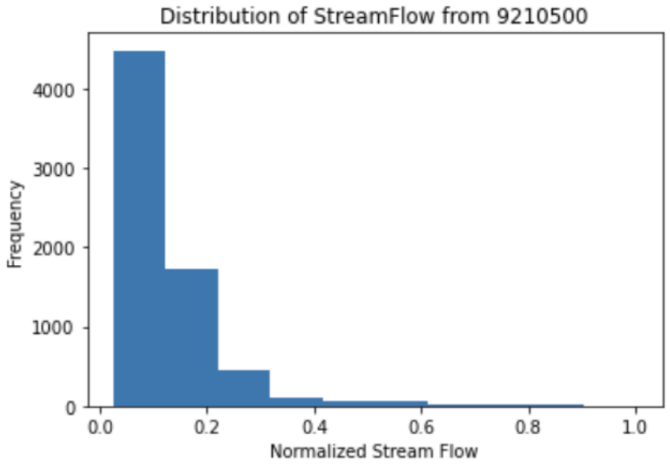
2. **10% Exceedence** - Goodness of match between `y_pred` and `y_real` for values in the bottom 10% of the distribution (low flow conditions). Threshold for each domain shown in table 1b.

3. **90% Exceedence** - Goodness of match between `y_pred` and `y_real` for values in the top 90% of the distribution (hi flow conditions). Threshold for each domain shown in table 1b

Table 4: *The study evaluates success by the match between LSTM-predicted `y_pred` and real `y_real` streamflow values during testing. This is done using the following metrics, which are further described in the literature and included in the scripts of the appendix.*

Metric	Description
`R_2`	**R Squared** - Values closer to 1 have a better match
`RMSD`	**Root Mean Squared Error** - Values closer to 0 have a better match
`NSE`	**Nash–Sutcliffe model efficiency coefficient** - Values closer to 1 have a better match (common metric in hydrology)
`KGE`	**Kling-Gupta model efficiency coefficient** - Values closer to 1 have a better match (common metric in hydrology)

***Figure 4**: Water managers are often concerned most with accurately predicting streamflow values that are in the upper and lower 10% of the distribution. This is due to that fact that flooding and drought-induced low flow tend to be the river conditions of largest human and environmental importance. Streamflow tends to be log-normally distributed (see below), and so the range of values in the bottom 10% of the distribution is much smaller than the range of values in the upper 10% of the distribution.*



D. A note on the X and Y variables

In this application, we are testing an array of predictive (X) and predicted (y) variables.

Relative to the current time `t`, the **predicted variable** is always a scalar streamflow at a future time

> *`t + fut_length`,*

where `fut_length` is user-defined design parameter.

The **predictive variables** are a matrix created by making `sliding window` of all the historical inputs of shape

> *`[d, seq_length, l]`,*

where `d` is the number of days in the dataset, `seq_length` is the number of days looking back from time `t`, and `l` is the number of predictive features.

For example, if `fut_length = 1` and `seq_length = 7` then the predicted variable is the streamflow one day in the future. And the predictive variable is matrix of shape `[d, 7, l]` that contains all 7 days previous values for each day `d` in a dataset of `l` features. The application and design of this method is shown in greater detail in the code of `appendix 2`.

E. Hydrologic Time Scales of Interest

To increase the hydrologic relevance of this discussion, the effectiveness of the LSTM at generating streamflow predictions is evaluated at different time scales of interest. As mentioned in the introduction, water managers are often interested in making predictions at different times in the future (daily, weekly, monthly, bi-annually) based on their management needs.

The different time scales are described in table 5 below, and the results of a preanalysis regarding these time scales are described further in the results section.

***Table 5**: The study set out to evaluate four hydrologic time scales of interest. The definition of those time scales and their relationship to the design parameters `fut_length` and `seq_length` are as follows.*

Time Scale Name	`fut_length` *	`seq_length` *
Daily	`7`	`1`
Weekly	`14`	`7`
Monthly	`60`	`30`
Bi-Annually	`360`	`180`

* `fut_length` is the time into the future at which streamflow is predicted from time `t`

* `seq_length` determines how many days of historical data from `t` to `t-seq_length` to incorporate in the prediction, such that the shape of the predictive variable matrix is `[d, seq_length, l]`

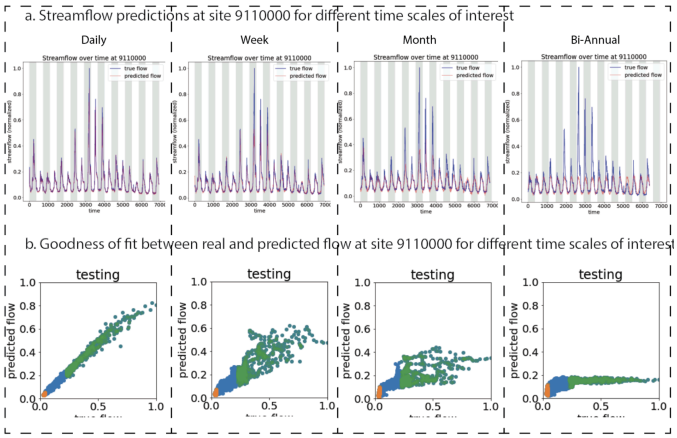
IV. Results & Interpretation

A. Hydrologic Time Scales of Interest

To identify a time scale of interest, the site `9110000` was evaluated for `Approach 1. Dorothy in Kansas` using the metrics of success as described in the methods section. Streamflow predictions for Daily, Weekly, Monthly, and Bi-Annual time scales were generated by four different LSTM models. The results are described in figure 5 below and in table 8 in `Appendix 1.`

Using the LSTM model architecture described in the methods section, we can see that short-term (Daily) predictions are quite good, while performance degrades significantly for longer-term (Monthly and Bi-Annual) predictions. We see performance degradation across all metrics, but the mismatch appears to be strongest for peak (`90% Exceedence`) flow.

***Figure 5:** __a)__ Plots of time series (time on the x axis, shown in hours; streamflow on the y axis, normalized). The blue line represents the true, or measured, streamflow. The red line represents the predicted flow by the LSTM. __b)__ Scatter plots representing the relationship between true (x-axis) and predicted (y-axis) flow at all time steps. Deviation from the 1-to-1 line represents a misfit. `Orange` points represent `10% Exceedence` and `Green` points represent `90% Exceedence`.*



Although predictive tools that can make peak flow predictions on a monthly and biannual time scale are hydrologically important, they are a subject for later investigation. The rest of this study focuses on the `weekly` time scale condition. This represents a challenge, but a sufficiently constrained one to yield useful results.

A NOTE ON METRICS

From the preliminary results shown in table 8 in the appendix, we see that `R_2` provides a convenient 'first' cut analysis when evaluating model performance, however in general fails to be a statistic of comprehensive utility due to its failure to capture relationships in the bottom part of the distribution (`10% exceedence`). For low flow, `RMSD` is the statistic of choice. For hi flow (`90% exceedence`), we find the `NSE` can provide useful insights but that by comparison `KGE` is less susceptible to bias and noise.

As such, the rest of this report will interpret model fit using `RMSD` for `10% exceedence` and `KGE` for `90% exceedence`. Based on the results we evaluate according to the following criteria:

Table 6: *Metrics used to classify model fit. __`RMSD` - used to evaluate low flow (`10% exceedence`).__ __`KGE` - used to evaluate hi flow (`90% exceedence`).__ A model often performs differently in terms of each metric.*

Property	`RMSD` Value Range (10% exceedence)	`KGE` Value Range (90% exceedence)
Very Good	`RMSD` < 0.05	`KGE` > 0.7
Good	0.05 < `RMSD` < 0.10	0.7 > `KGE` > 0.55
Satisfactory	0.10 < `RMSD` < 0.20	0.55 > `KGE` > 0.35
Unsatisfactory	`RMSD` > 0.20	`KGE` < 0.35

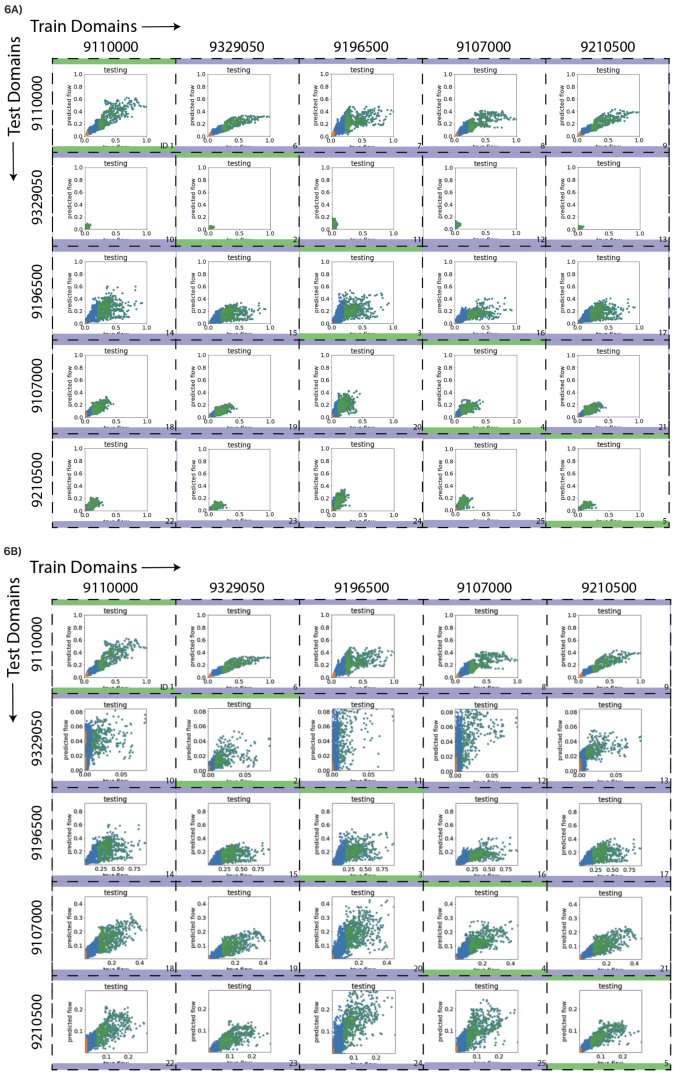
B. Results & Interpretation (1) - Dorothy in Kansas

The results of `Approach 1. Dorothy in Kansas` are shown in table 9 in the appendix and figure 6 below, IDs 1-5. In this approach, models were trained on only one stream and tested on out-of-sample reserved datasets that come from the same stream. Thus, these results represent a kind of baseline for model performance. In general, these models do a good job of predicting flow conditions. The models tend to under-predict peak (`90% exceedence`) streamflow, but still capture the general relationship.

The relationship between `y_pred` and `y_real` is captured particularly well for `10% exceedence` conditions. Using the classification system from table 6, four of the five models classify as `Very Good` (`RMSD < 0.10`), with the other model (`ID 3`) classifying as `Good`. The relationship between `y_pred` and `y_real` for `90% exeedence` is generally less robust. Three models were `satisfactory and good` (0.7 > `KGE` > 0.35), however two models were `Unsatisfactory`. For both `90% exeedence` and `10% exceedence`, the model with the greatest mismatch between `y_pred` and `y_real` is USGS station `9196500` (see `ID 3`).

The results of investigations at the weekly time scale for `Approach 1. Dorothy in Kansas` and `Approach 2. Dorothy in Oz` are shown below in figure 6 below and in table 9 in `Appendix 1`.

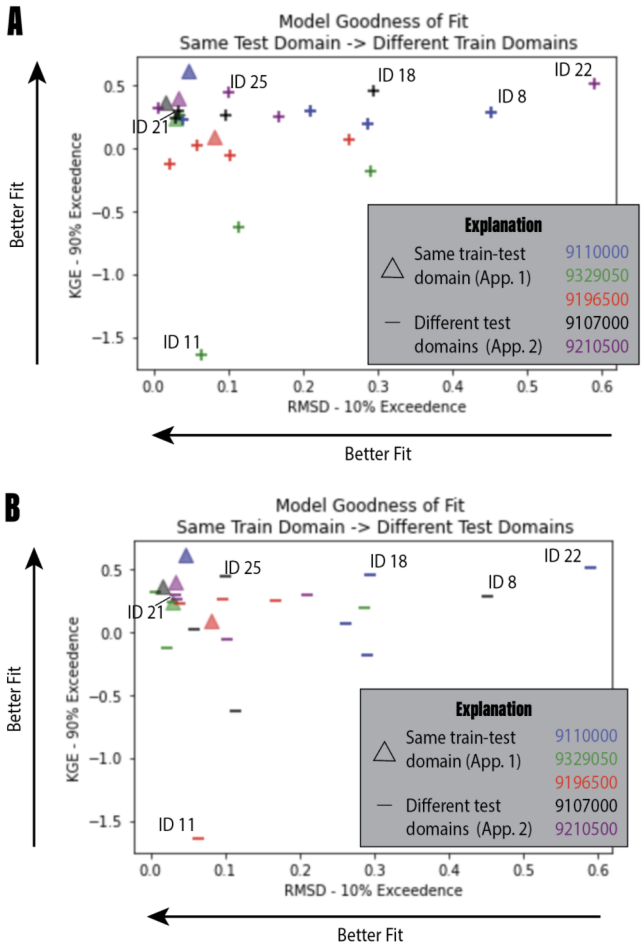
Figure 6: *Cross plots showing how well models trained in one domain perform when tested on another. Along the x-axis, different train domains. Along the y-axis, different test domains. Green boxes represent `Approach 1. Dorothy in Kansas` (Train and Test domains are the same). Blue boxes represent `Approach 2. Dorothy in Oz` (Train and Test domains differ). Within each box, the scatter plots represent the relationship between true (x-axis) and predicted (y-axis) flow at all time steps. Deviation from the 1-to-1 line represents a mismatch. `Orange` points represent `10% Exceedence` and `Green` points represent `90% Exceedence` flow at each site. Note the `ID #` located in the bottom right corner of each box. These correspond to the `ID` column in table 9. Two versions of figure shown (with same results)- `A)` results shown with normalized scale 0 to 1 for all sites; `B)` results shown with normalized scale min to max for each individual site.*



C. Results & Interpretation (2) - Dorothy in Oz

The results of `Approach 2. Dorothy in Oz` are shown in table 9 in the appendix and figure 6 above, IDs 6-25. In this approach, the models trained in approach 1 for each stream were tested on datasets from the other four streams. This approach shows how well an LSTM model performs on a test dataset that doesn't represent the same population upon which it was trained. Thus, these results represent an evaluation of the capacity for transfer learning. Evaluation and Interpretation of these result is driven by figure 7 below, which show the goodness of fit for each model and train-test combination along the dimensions of `10% Exceedence` and `90% Exceedence` based upon `RMSD` and `KGE`, respectively.

Figure 7: *Scatterplots showing the goodness of fit for each model and train-test combination along the dimensions of `10% Exceedence` (x-axis) and `90% Exceedence` (y_axis) based upon `RMSD` and `KGE` respectively. The 'best fitting' models congregate in the upper left corner. In these figures, the `(1) Dorothy in Kansas` models are triangles. The `(2) Dorothy in Oz` models are represented two ways - by the crosses and the minuses. Figure `A` shows how well models from different Train domains map onto one specific Test domain. Figure `B` shows how well a model from a specific Train domain maps onto each different Test domain. They show the same data, but symbologize it differently to give two different perspectives of the same results.*



One thing we notice immediately is that the best fitting models are (generally) triangles - meaning the highest performance in terms of predicting 90% and 10% exceedence are those models that were trained and tested on the same domain. The one exception to this trend are the results for domain `9196500`, which as we noted earlier seems to be a lower performing domain in general.

When we turn our attention to the other models (crosses and minuses), we see that they in general perform worse than the triangles. That is to say, models almost always lose their predictive power when transferred to a new test domain. However, it is interesting to note that the majority loss in performance is usually one-dimensional. I.E., a transfered model generally has low predictive power for either `10% Exceedence` or `90% Exceedence`, but not both. We know this because we don't see any points plotting in the lower right corner of these graphs. For example, the model trained on `9196500` and tested on `9329050` (`ID 11`) is very `unsatisfactory` at high flow predictions, but is `good` at high flow predictions (classification based on table 6).

It's also worth noting that some of the transferred models (crosses and minuses) perform *better* than the triangle models in one or the other dimension (`10% Exceedence` or `90% Exceedence`), but seldom both. For example, the model trained on `9110000` and tested on `9210500` (`ID 22`) makes `good` and, indeed, improved predictions for that domain for `90% Exceedence` conditions.

The two cases described above (ID `11` and `22`) are interesting because they are examples of models trained on relatively large rivers being used to make effective predictions on relatively small rivers for either `10% Exceedence` or `90% Exceedence`, but not both.

But how well do models from the same river basins transfer learning onto each other? To answer that question we look to models trained and tested on data from sites `9110000` and `9107000`, which are of different size but actually geographically overlap. To evaluate this we look to model IDs `8` (trained on `9107000`, tested on `9110000`) and `18` (trained on `9110000`, tested on `9107000`). What we find is somewhat surprising - results show that for `8`, RMSD (10% Exceedence)=0.4518 and KGE (90% Exceedence)=0.2833; for `18` RMSD (10% Exceedence)=0.2941, KGE (90% Exceedence)=0.4515. This goodness of fit is `unsatisfactory` for both models across all dimensions. And so we can conclude that geographic proximity is not a predictor for model transferability.

And what about models trained and tested on river domains of similar size and flow? From the perspective of the flow statistics described in table 1b, the most similar sites are `9107000` and `9210500`. To evaluate model transferability, we look to model IDs `21` and `25`. The goodness of fit of for `21` is represented by RMSD (10% Exceedence)=0.0337 and KGE (90% Exceedence)=0.2966; for `25`, RMSD (10% Exceedence)=0.0996 and KGE (90% Exceedence)=0.4342. All but one of these metrics yield at least `satisfactory` results. These models are perhaps the most successful results showcasing model transfer from Approach (2), and suggest that models perform best on domains that are hydrologically similar to those they were trained on.

D. *Results & Interpretation (3) - Dorothy comes back to Kansas*

We've noticed generally that models trained on one domain can be transferred to other test domains, but that there is always at least some degree of performance degradation in this transfer. In general, the most robust model for making both `10% Exceedence` and `90% Exceedence` predictions are those trained and tested on the same domain, or domains that are hydrologically similar.

But how does a generalized model trained on the data from all 5 domains perform? The answer to that question is explored in `Approach 3. Dorothy back in Kansas.` In this approach, an LSTM model (`ID 26`) is trained sequentially on train data from each of the 5 train domains. Finally, this new model is tested on reserved test data from each of the 5 domains individually. The results of this analysis are shown in table 9 in `Appendix 1`, and figures 8 and 9 below.

Figure 8: *Cross plots showing how well models perform when tested on each domain. Along the x-axis, different test domains. Along the y-axis, different model training approaches. Green boxes represent `Approach 1. Dorothy in Kansas` (Specialized model trained and tested on same domain). Light orange boxes represent `Approach 3. Dorothy back in Kansas` (Generalized model trained on all five domains but tested on each domain individually). Within each box, the scatter plots represent the relationship between true (x-axis) and predicted (y-axis) flow at all time steps. Deviation from the 1-to-1 line represents a mismatch. `Orange` points represent `10% Exceedence` and `Green` points represent `90% Exceedence` flow data at each site. Note the `ID #` located in the bottom right corner of each box. These correspond to the `ID` column in table 9. Results shown with normalized scale min to max for each individual site.*

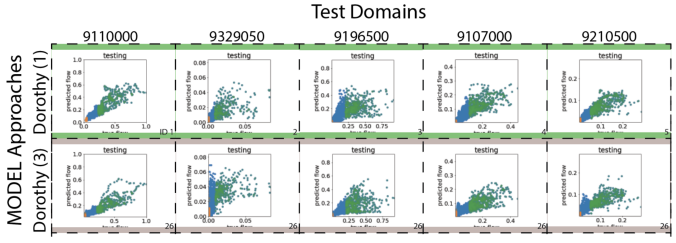
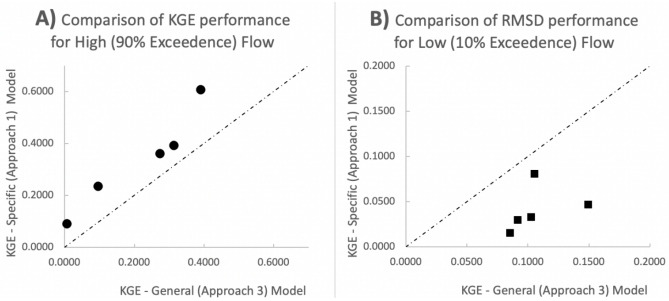


Figure 9: *Comparison of the performance of specialized models `Approach 1. Dorothy in Kansas` (green) and generalized `Approach 3. Dorothy back in Kansas` (light orange) models for each test site across the metrics of interest (`RMSE` and `KGE`). __A)__ and __B)__ compare the metrics of interest for each train domain for `90% Exceedence` and `10% Exceedence` conditions, respectively. Note that for __A)__, a train domain that plots __above__ the 1-to-1 line is _best_ represented by a specialized (Approach 1) model. For __B)__, a train domain that plots __below__ the 1-to-1 line is _best_ represented by a specialized (Approach 1) model. __C)__ shows the metrics of concern in an easy-to-view format, subsampled from table 9 in `Appendix 1`*



C) Metrics for comparing modeling approaches

General (3) Model		Specific (1) Model	
RMSD - 10%	KGE - 90%	RMSD - 10%	KGE - 90%
0.1494	0.3911	0.0466	0.6073
0.0916	0.0962	0.0296	0.2350
0.1055	0.0063	0.0806	0.0917
0.0855	0.2733	0.0155	0.3613
0.1027	0.3137	0.0331	0.3929

We can see that the generalized `Dorothy comes back to Kansas` model does well enough to capture the dynamics of streamflow at each test location in a general sense (Figure 8). However, it performs worse across every metric of success when tested against the specialized models for each streamflow location individually. This can be seen by how the points plot relative to the 1-to-1 line in the figures 9 A and B.

As such, we conclude that a generalized model may do a decent job of representing flow conditions across a range of sites, but that this capacity to generalize comes at a cost. For best predictions at a site level, models tuned to single domain seem to perform best given the architecture and assumptions used for this study.

V. Conclusion

1. Models trained in one domain may make effective predictions in other domains. However, in general performance is lower than a model trained and tested in the same domain. However, it is interesting to note that the majority loss in performance is usually one-dimensionally. I.E., it generally loses predictive power along either for `10% Exceedence` or `90% Exceedence`, but not both. This suggests that some models may be effective at making predictions in other domains during exceptionally wet and/or dry

conditions, but that this selection process should be done carefully.

2. The two cases of ID `11` and `22` are interesting because they are examples of models trained on relatively large rivers being used to make effective predictions on relatively small rivers for either `10% Exceedence` or `90% Exceedence`, but not both.

3. How well do models from the same river basins transfer learning onto each other? What we find is that these perform unsatisfactorily in at least one dimension, so we can conclude that geographic proximity is not a predictor for transferability.

4. How about models trained and tested on river domains of similar size and flow? Results suggest that models perform best on domains that are hydrologically similar to those they were trained on.

5. Finally, we can see that a generalized model does well enough to capture the dynamics of streamflow at each test location in a general sense, but performs worse across every metric of success when tested against the specialized models for each streamflow location individually. As such, we conclude that a generalized model may do a decent job of represent flow conditions across a range of sites, but that this capacity to generalize comes at a cost. For best predictions at a site level, models tuned to single domain seem to perform best given the architecture and assumptions used for this study.

VI. FURTHER STUDY

1. Find models that better predict high flow (90% exceedence) conditions. High flow conditions were generally under-predicted by the three approaches investigated in this study.

   - Given the log-normal distribution of streamflow, it could be worth investigating log transforms of data using this approach. However, I think this isn't a particularly good 'ML' approach

   - Using different activation functions, or LSTM constructions that better deal with cyclical data could help in this regard.

   - Incorporate forecasts of rainfall, which are probably an important predictive factor of streamflow that are not represented in historical climate forcing data.

2. Develop models that can predict longer term trends. As seen in the preliminary analysis of hydrologic time scales of interest, performance degrades substantially the further out in the future we look. This makes intuitive sense, as there is more uncertainty as we look to the future, and more information that the model lacks in making its prediction. However, it is these longer-term trends that often matter most to decision makers.

- Perhaps resample to monthly trends for the biannual prediction.

3. Develop a models that transfer better. In general, we saw that models trained and tested on the same domain perform best. However, practically speaking it is not always possible to build a model that is trained at every streamflow location of interest (computational, data limitations). Generalized models or models trained on other, similar watersheds show promise of making reasonable predictions but inherently come with tradeoffs. A more thorough investigation of the tradeoffs and possibilities of transfer learning is needed.

- Investigate the trade-offs between generalized models and specified models. The benefits of a generalized model is that it can be trained on one location to make meaningful predictions at other locations that might not have the same degree of information available.

4. Adding missing data that are factors in streamflow but that weren't included in the X values for this study. Adding more and different variable features may better represent the 'missing' information (such as snowpack melt off) that contribute to performance breakdown during transfer and 90% exceedence prediction.

- For example, including static variables representing basin conditions (such as basin size, shape, slope, etc...) could help us to build a better generalized model.

5. Test models that do not use streamflow as a predictive feature. One of the downsides to the approach used in this report is that we use streamflow from the past to make predictions of streamflow in the future. Naturally, this a relatively easy task because streamflow in the future is closely tied to streamflow from the past. But the downside is that we can't apply this model to locations where streamflow observations are sparse, but other observations (such as climate and basin conditions) are robust - a common occurrence in hydrology. To increase the general utility of these types of models, streamflow predictions that don't depend upon streamflow observations would a great application to this LSTM method. Preliminary investigation of this approach (not shown in this report) serve as testimony that this is a difficult task!

   - Test out some other model architectures for prediction. It would behoove us to test out different hyperparameters, add hidden layers, and use regularization.

   - Think about how LSTM was used for text prediction to generate accurate and plausible time series

REFERENCES

[1] Naghettini, Mauro, ed. Fundamentals of statistical hydrology. Switzerland: Springer International Publishing, 2017.

[2] Chaopeng Shen, 2018, 'A Transdisciplinary Review of Deep Learning Research and Its Relevance for Water Resources Scientists'.

[3] Kratzert et al. 2018. 'Rainfall–runoff modelling using Long Short-TermMemory (LSTM) networks.' Hydrology and Earth System Sciences.

[4] NSF Hydroframe-ML grant award. https://www.nsf.gov/awardsearch/showAward?AWD_ID=2040542&HistoricalAwards=false.

[5] USGS Streamflow. https://waterdata.usgs.gov/nwis/rt.

[6] Climate Forcing Dataset. https://www.wcc.nrcs.usda.gov/gis/index.html.

Hull_Appendix1.md

8. Appendix 1 - Data Tables

Table 8: *Preliminary investigation of the performance of models for different times scales (Daily, Weekly, Monthly, Biannually). Metrics and Significance described more thoroughly in methods. Interpretation in results.*

Group Model Characteristics			Individual Characteristics			Test Statistics			
Approach	Train domain	Test domain	Model Name	Time Scale	Test Statistics	R_2	RMSD	NSE	KGE
(1) Dorthy in Kansas	9110000	9110000	0506_a	Daily	All Streamflow	0.9853	0.2110	0.9853	0.9365
					10% Exceedence	0.4869	0.0215	0.4869	0.7472
					90% Exceedence	0.9450	0.4019	0.9450	0.8855
			0506_b	Weekly	All Streamflow	0.8385	0.9466	0.8385	0.7466
					10% Exceedence	-2.9883	0.0466	-2.9883	-0.3166
					90% Exceedence	0.3574	1.8808	0.3574	0.6073
			0506_c	Monthly	All Streamflow	0.5501	1.3702	0.5501	0.4697
					10% Exceedence	-27.6709	0.2206	-27.6709	-3.4126
					90% Exceedence	-0.8717	4.0027	-0.8717	0.1478
			0506_d	Bi-Annually	All Streamflow	0.3125	1.9778	0.3125	0.2778
					10% Exceedence	-42.7262	0.4478	-42.7262	-2.8634
					90% Exceedence	-1.9240	5.2494	-1.9240	-0.3267

Table 9: *Goodness of fit for `Approach 1. Dorothy in Kansas` (shown in green), `Approach 2. Dorothy in Oz` (shown in blue), and `Approach 3. Dorothy back in Kansas` (shown in light orange). Approach 1 evaluates the performance of model trained and tested on data taken from the site. Approach 2 evaluates the ability of a model to transfer to test data from a different site. Approach 3 evaluates the ability of a general model trained on data from all 5 sites and then tested on data from each site individually.*

Group Model Characteristics					Test Statistics				
Approach	ID	Train domain	Test domain	Model Name	Test Statistics	R_2	RMSD	NSE	KGE
(1) Dorthy in Kansas	1	9110000	9110000	0506_b	All Streamflow	0.8385	0.9466	0.8385	0.7466
					10% Exceedence	-2.9883	0.0466	-2.9883	-0.3166
					90% Exceedence	0.3574	1.8808	0.3574	0.6073
	2	9329050	9329050	0506_e	All Streamflow	0.4724	0.0874	0.4724	0.4933
					10% Exceedence	-142.0975	0.0296	-142.0975	-6.8059
					90% Exceedence	-0.0961	0.0919	-0.0961	0.2350
	3	9196500	9196500	0506_f	All Streamflow	0.6311	0.9533	0.6311	0.6012
					10% Exceedence	-259.2842	0.0806	-259.2842	-12.1362
					90% Exceedence	-1.2291	3.0743	-1.2291	0.0917
	4	9107000	9107000	0506_g	All Streamflow	0.7518	0.7097	0.7518	0.6557
					10% Exceedence	-9.6931	0.0155	-9.6931	-1.2631
					90% Exceedence	-0.3138	1.2450	-0.3138	0.3613
	5	9210500	9210500	0506_h	All Streamflow	0.7650	0.1530	0.7650	0.6791
					10% Exceedence	-24.7504	0.0331	-24.7504	-2.9550
					90% Exceedence	-0.0281	0.7036	-0.0281	0.3929
(2) Dorthy in Oz	6	9329050	9110000	0506_e	All Streamflow	0.4709	3.6803	0.4709	0.3347
					10% Exceedence	-8.6271	0.2869	-8.6271	0.5265
					90% Exceedence	-0.9083	4.4844	-0.9083	0.1872
	7	9196500	9110000	0506_f	All Streamflow	0.6069	1.3415	0.6069	0.5873
					10% Exceedence	-4.5671	0.0385	-4.5671	-0.7537
					90% Exceedence	-0.4740	3.0754	-0.4740	0.2254
	8	9107000	9110000	0506_g	All Streamflow	0.4514	4.1134	0.4514	0.3723
					10% Exceedence	-22.7497	0.4518	-22.7497	0.1327
					90% Exceedence	-0.7142	4.0865	-0.7142	0.2833
	9	9210500	9110000	0506_h	All Streamflow	0.5812	3.1909	0.5812	0.4165
					10% Exceedence	-6.5768	0.2106	-6.5768	-0.0405
					90% Exceedence	-0.5116	3.9498	-0.5116	0.2961
(2) Dorthy in Oz	10	9110000	9329050	0506_b	All Streamflow	-11.5019	1.3986	-11.5019	-4.4792
					10% Exceedence	-13210.4387	0.2908	-13210.4395	-79.1296
					90% Exceedence	-2.3756	0.3647	-2.3756	-0.1903
	11	9196500	9329050	0506_f	All Streamflow	-12.7705	0.3077	-12.7705	-2.3457
					10% Exceedence	-4901.9381	0.0642	-4901.9380	-65.6553
					90% Exceedence	-13.8233	0.7357	-13.8233	-1.6395
	12	9107000	9329050	0506_g	All Streamflow	-6.1357	0.7472	-6.1357	-2.2841
					10% Exceedence	-2006.2609	0.1140	-2006.2611	-29.5362
					90% Exceedence	-5.9562	0.5515	-5.9562	-0.6251
	13	9210500	9329050	0506_h	All Streamflow	-0.3368	0.3394	-0.3368	-0.3510
					10% Exceedence	-517.6834	0.0345	-517.6834	-18.9606
					90% Exceedence	-0.0173	0.1034	-0.0173	0.2527

Group Model Characteristics					Test Statistics				
Approach	ID	Train domain	Test domain	Model Name	Test Statistics	R_2	RMSD	NSE	KGE
(2) Dorthy in Oz	14	9110000	9196500	0506_b	All Streamflow	0.6474	0.2530	0.6474	0.6626
					10% Exceedence	-871.8697	0.2620	-871.8698	-9.9647
					90% Exceedence	-1.1192	2.4854	-1.1192	0.0626
	15	9329050	9196500	0506_e	All Streamflow	0.5247	1.8361	0.5247	0.3357
					10% Exceedence	-15.5245	0.0225	-15.5245	-1.4748
					90% Exceedence	-2.2495	4.3197	-2.2495	-0.1335
	16	9107000	9196500	0506_g	All Streamflow	0.5489	1.7763	0.5489	0.3361
					10% Exceedence	-56.6888	0.0587	-56.6888	-2.6420
					90% Exceedence	-2.0674	4.4198	-2.0674	0.0238
	17	9210500	9196500	0506_h	All Streamflow	0.5799	1.4247	0.5799	0.4256
					10% Exceedence	-196.8025	0.1016	-196.8025	-7.2518
					90% Exceedence	-1.8736	3.8834	-1.8736	-0.0615
(2) Dorthy in Oz	18	9110000	9107000	0506_b	All Streamflow	0.6920	0.8916	0.6920	0.6416
					10% Exceedence	-254.6997	0.2941	-254.6998	-7.6776
					90% Exceedence	-0.0199	0.7025	-0.0199	0.4515
	19	9329050	9107000	0506_e	All Streamflow	0.6375	1.0788	0.6375	0.4495
					10% Exceedence	-4.8234	0.0285	-4.8234	-0.3000
					90% Exceedence	-1.1073	1.9435	-1.1073	0.2313
	20	9196500	9107000	0506_f	All Streamflow	0.6002	0.1802	0.6002	0.7637
					10% Exceedence	-82.3343	0.0962	-82.3343	-6.4024
					90% Exceedence	-0.5784	0.2197	-0.5784	0.2613
	21	9210500	9107000	0506_h	All Streamflow	0.7104	0.7465	0.7104	0.5500
					10% Exceedence	-31.5773	0.0337	-31.5773	-3.6542
					90% Exceedence	-0.7011	1.6613	-0.7011	0.2966
(2) Dorthy in Oz	22	9110000	9210500	0506_b	All Streamflow	0.2616	1.3877	0.2616	0.1798
					10% Exceedence	-1299.6772	0.5901	-1299.6773	-19.9557
					90% Exceedence	0.1621	0.0052	0.1621	0.5105
	23	9329050	9210500	0506_e	All Streamflow	0.6878	0.4666	0.6878	0.5241
					10% Exceedence	-3.2591	0.0058	-3.2591	-0.3000
					90% Exceedence	-0.4414	0.9612	-0.4414	0.3187
	24	9196500	9210500	0506_f	All Streamflow	0.1232	0.3160	0.1232	0.3518
					10% Exceedence	-207.5994	0.1671	-207.5994	-10.2730
					90% Exceedence	-1.4072	1.0321	-1.4072	0.2513
	25	9107000	9210500	0506_g	All Streamflow	0.6561	0.1841	0.6561	0.8014
					10% Exceedence	-58.5752	0.0996	-58.5752	-4.0078
					90% Exceedence	-0.1364	0.0268	-0.1364	0.4342

Group Model Characteristics					Test Statistics				
Approach	ID	Train domain	Test domain	Model Name	Test Statistics	R_2	RMSD	NSE	KGE
(3) Dorthy back in Kansas	26	All Domains	9110000	0507_grande	All Streamflow	0.6129	2.8533	0.6129	0.4666
					10% Exceedence	-6.0120	0.1494	-6.0120	-0.4241
					90% Exceedence	-0.4478	3.8815	-0.4478	0.3911
	26	All Domains	9329050	0507_grande	All Streamflow	-1.5536	0.5286	-1.5536	-1.0993
					10% Exceedence	-957.9290	0.0916	-957.9290	-15.5344
					90% Exceedence	-0.6251	0.2000	-0.6251	0.0962
	26	All Domains	9196500	0507_grande	All Streamflow	0.5765	1.1212	0.5765	0.4812
					10% Exceedence	-207.5350	0.1055	-207.5351	-7.5229
					90% Exceedence	-1.8952	3.6895	-1.8952	0.0063
	26	All Domains	9107000	0507_grande	All Streamflow	0.6801	0.5684	0.6801	0.5356
					10% Exceedence	-52.7525	0.0855	-52.7525	-4.5270
					90% Exceedence	-0.8840	1.8107	-0.8840	0.2733
	26	All Domains	9210500	0507_grande	All Streamflow	0.7125	0.0504	0.7125	0.6529
					10% Exceedence	-35.1868	0.1027	-35.1868	-1.6595
					90% Exceedence	-0.1332	0.7451	-0.1332	0.3137

Appendix 2 - Code

In [1]:
```
# Modules:
from PIL import Image


import os
import os.path
import sys
import shutil
from pprint import pprint
from datetime import datetime

from parflowio.pyParflowio import PFData
from parflow import Run

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors

import pandas as pd

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import QuantileTransformer
from sklearn.preprocessing import PowerTransformer

import matplotlib.image as mpimg
import matplotlib.cm as cm
import matplotlib.pyplot as plt

import numpy as np
import numpy.ma as ma
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import matplotlib.cbook as cbook
from matplotlib.path import Path
from matplotlib.patches import PathPatch

from copy import copy
from copy import deepcopy

# import pfspinup
# see more: https://grapp1parflow.readthedocs.io/en/latest/python/getting_starte
from parflowio.pyParflowio import PFData # for reading / writing PFB files

# for DL
import torch
import torch.nn as nn

from torch.utils.data import TensorDataset # for refactoring x and y
from torch.utils.data import DataLoader # for batch submission

from torch.autograd import Variable
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error

from pfread_manual import pfread
```

```
from lstm_models import sliding_windows
from lstm_models import LSTM


# for metrics
from sklearn.metrics import mean_squared_error
from utils import compute_stats


# Path to the SandTank Repo
dev_path = '/home/SHARED/ML_TV/HydroGEN/modules/'
#Add Sand Tank path to the sys path
sys.path.append(dev_path)
from transform import float32_clamp_scaling
```

Post Processing Forcing Figures

Some important notes

Updated 04/24/2021

For files of name type `/UCRB_DSWR.1983.pfb`

You will see they are shape `out_data.shape = (5, 3, 366)`

Where `out_data.shape[0] = 5` -> number of stations in the dataset of this order `[9110000, 9329050, 9196500, 9107000, 9210500]`

Where `out_data.shape[1] = 3` -> scaler outputs of dynamic variables `0 = mean, 1 = std over tim, 2 = std over space`

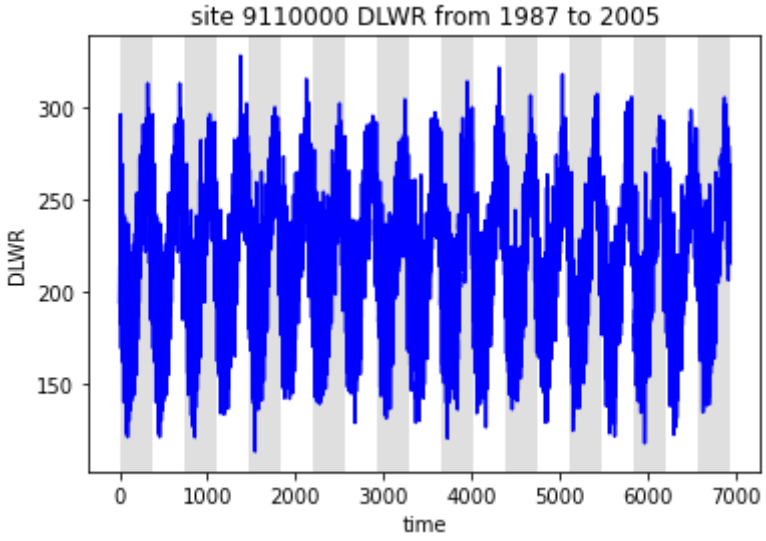
Where `out_data.shape[2] = 366` -> number of days in a year (where the last is an extra just for leap years that is normally zero)

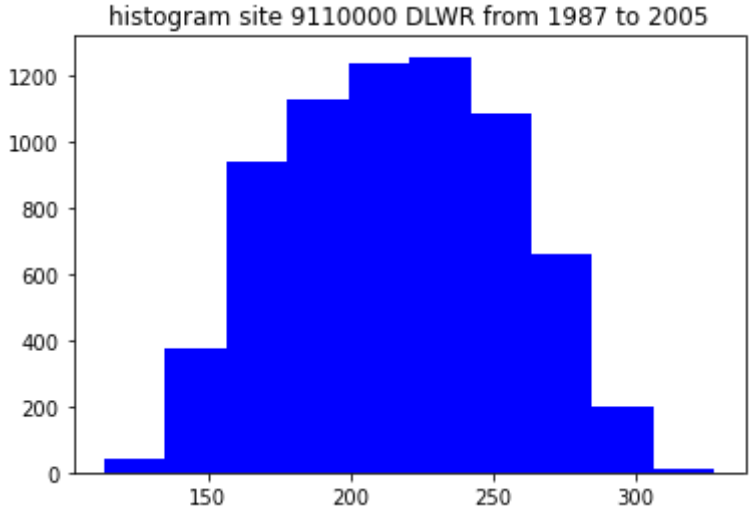
The below is taken from `scripts/UCRB_figs.ipynb`

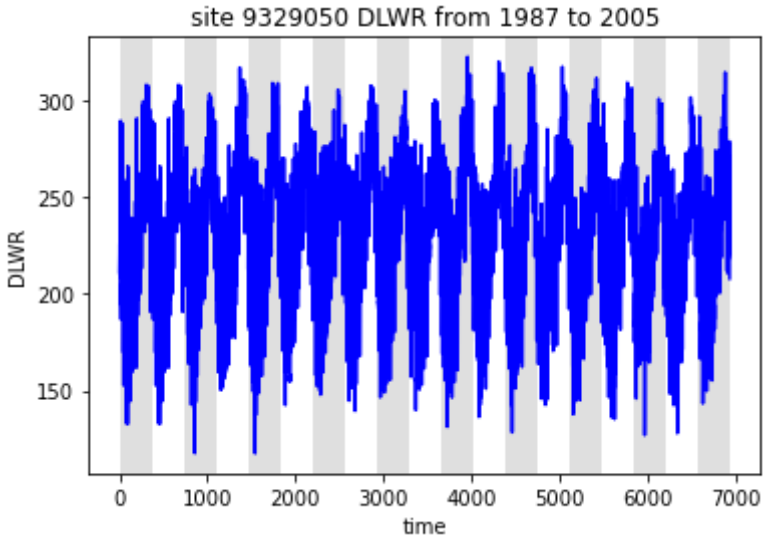
In [2]:
```
# Setup array of years
path = '../data_out'

huc_name_list = [9110000, 9329050, 9196500, 9107000, 9210500]
labelist = ['DLWR', 'DSWR', 'Press', 'APCP', 'Temp', 'SPFH', 'UGRD', 'VGRD']
collist = ['blue','green','red', 'yellow', 'orange', 'purple', 'indigo','black']
YEARS = []
for yr in range(1987,2006): # QH 1983, 2020 for all years
    YEARS.append(yr)

# #GLOBALS#
# -----
# change for exporting
exp_name = 'ML_FinalProj_1987-2007'
save = False

# keep track of params
j = 0

# for plotting
```

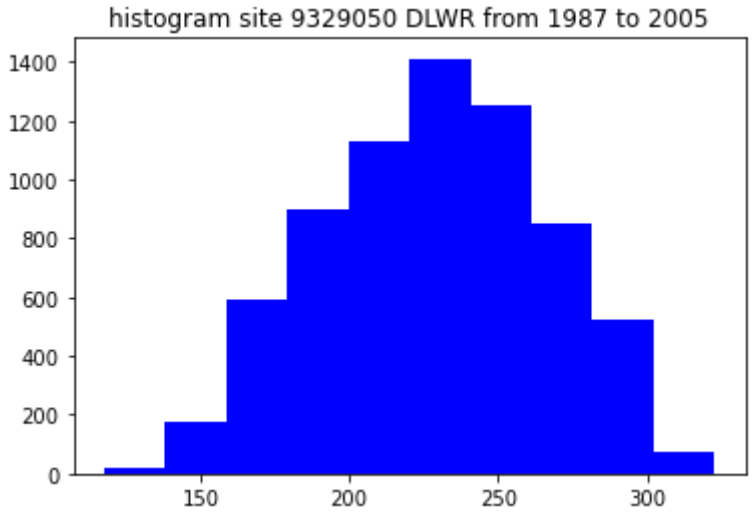
```
# add std
bracket = False
std = 0
# data, # 0 = average, 1 = std space, 2 = std time
data = 0

# Aggregate values between plots # update - assume that we use the dates by wate
agg = True
# set instance of first day (assume Oct 1, YYYY[0]-1)
y0 = YEARS[0]-1
day0 = 1
mon0 = 10
date0 = str(mon0)+'/'+str(day0)+'/'+str(y0)
# date used for resampling
date1 = '1/1/'+str(YEARS[0])
df_l = [] # list of data frames



# #LOOPING#
# -----



for label in labelist:
    # assemble data
    for yr in YEARS:
        # pull out data test
#         print(path+'/UCRB_'+label+'.'+str(yr)+'.pfb')
        out = PFData(path+'/UCRB_'+label+'.'+str(yr)+'.pfb')
        out.loadHeader()
        out.loadData()
        out_data = out.getDataAsArray()
        # get ride of random zeros in last row
        out_data = out_data[:,:,:-1]

        # create array of placeholder values
        if yr == YEARS[0]:
            label_arr = out_data
        else:
            label_arr = np.concatenate([label_arr,out_data], axis=2)
#             print(label_arr.shape)

    # Plot stuff
    # for each gage
    for ea in range(label_arr.shape[0]):
#         print(huc_name_list[ea])
        # for each parameter
        fig, ax = plt.subplots()
        # for indexing colorlist
        j = labelist.index(label)
        # time, data, color
        ax.plot(range(label_arr.shape[2]), label_arr[ea,data, :], collist[j])
        if bracket:
            ax.plot(range(label_arr.shape[2]), label_arr[ea,data, :]+label_arr[e
            ax.plot(range(label_arr.shape[2]), label_arr[ea,data, :]-label_arr[e
        ax.set_xlabel('time')
        ax.set_ylabel(label)
        plt.title(str('site '+str(huc_name_list[ea])+' '+label+' from '+str(YEAR
        for i in range(0,len(YEARS),2):
            day = i*365
```

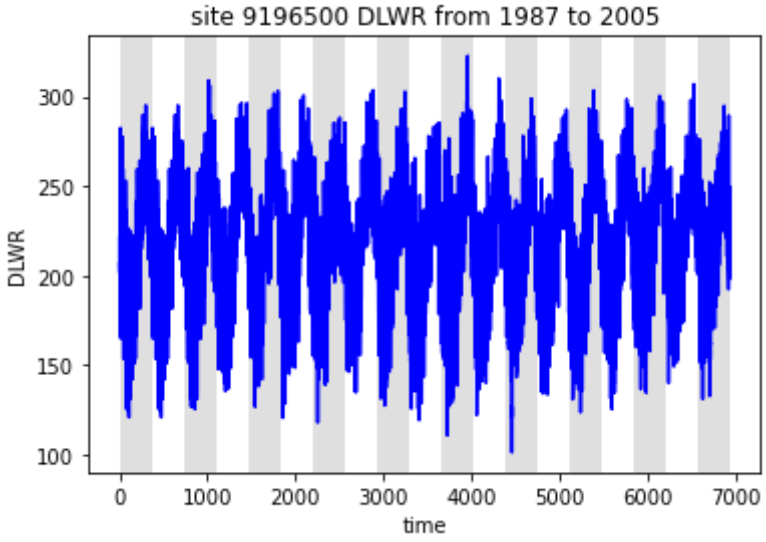
```
            plt.axvspan(day,day+365, facecolor='grey', alpha=0.25)
        if save and (huc_name_list[ea] == 9110000):
            plt.savefig(path+'/2_fig/fig'+label+exp_name+str(huc_name_list[ea])+
        plt.show()

        plt.hist(label_arr[ea,data, :], color=collist[j])
        plt.title(str('histogram site '+str(huc_name_list[ea])+' '+label+' from
        if save and (huc_name_list[ea] == 9110000):
            plt.savefig(path+'/2_fig/fig_hist'+label+exp_name+str(huc_name_list[
        plt.show()

        # if aggregate
        if agg:
            if j == 0:
                df_l.append(pd.DataFrame())
            df = df_l[ea]
            df[label] = label_arr[0,data,:]
            # set up date indexes
            df['deltatime'] = pd.to_timedelta(df.index, unit='D')
            df['timestamp'] = pd.Timestamp(date0) + df['deltatime']
            df['resampstamp'] = pd.Timestamp(date1) + df['deltatime']
```

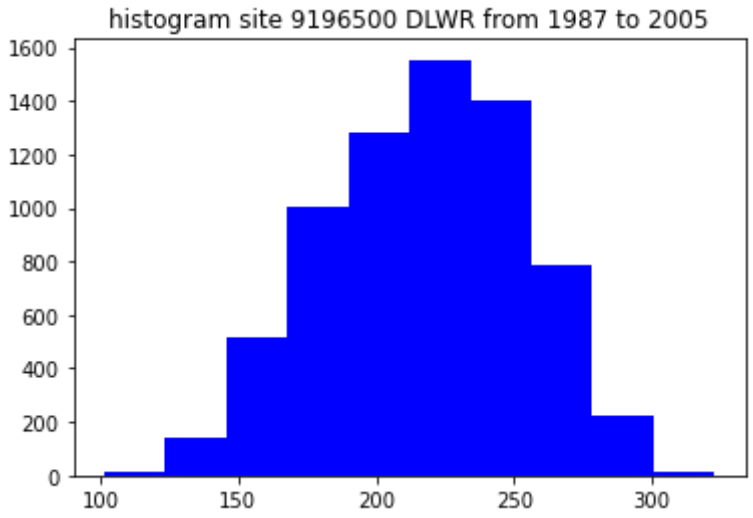


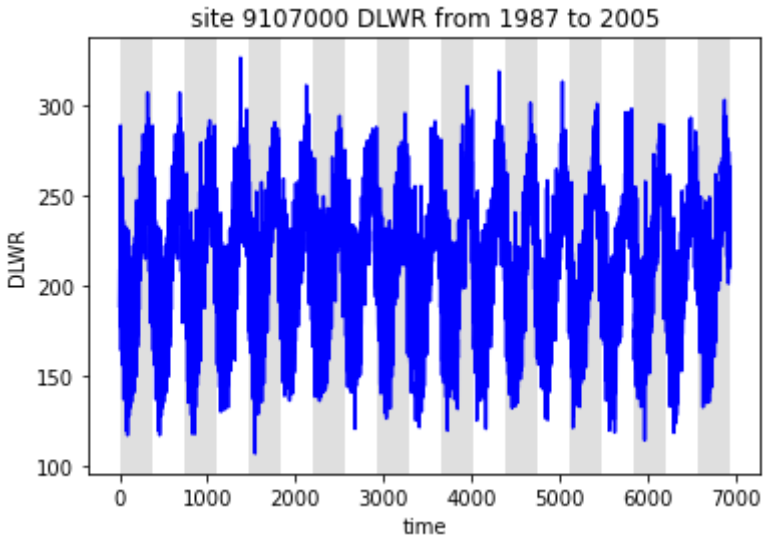


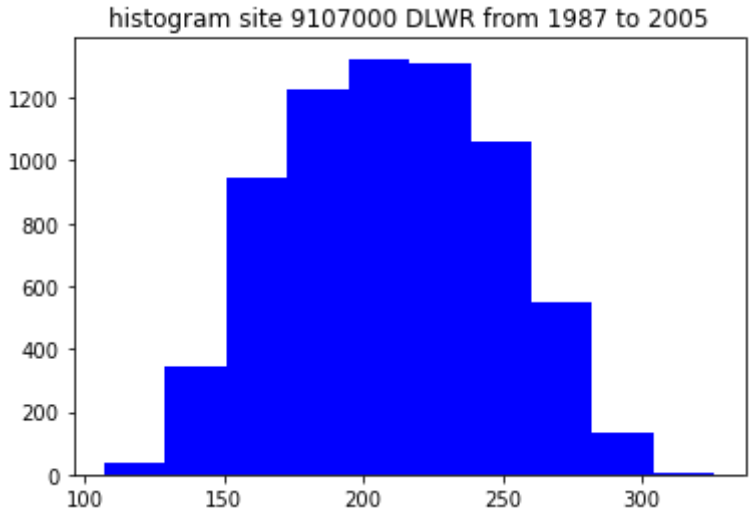

site 9329050 DLWR from 1987 to 2005
DLWR
time
300
250
200
150
0
1000
2000
3000
4000
5000
6000
7000

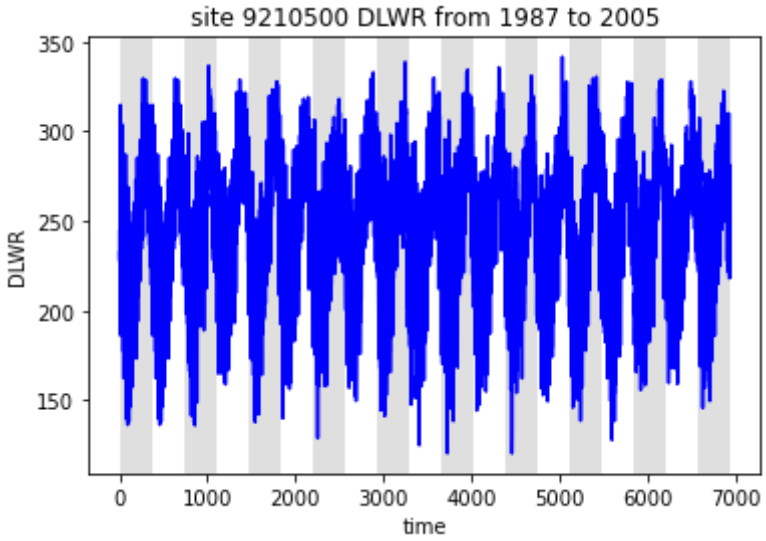

histogram site 9329050 DLWR from 1987 to 2005
1400
1200
1000
800
600
400
200
0
150
200
250
300

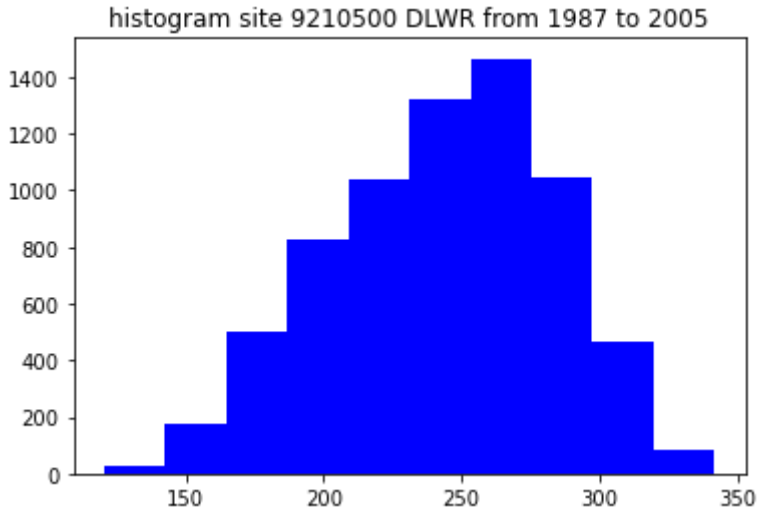

site 9196500 DLWR from 1987 to 2005
DLWR
time
300
250
200
150
100
0
1000
2000
3000
4000
5000
6000
7000

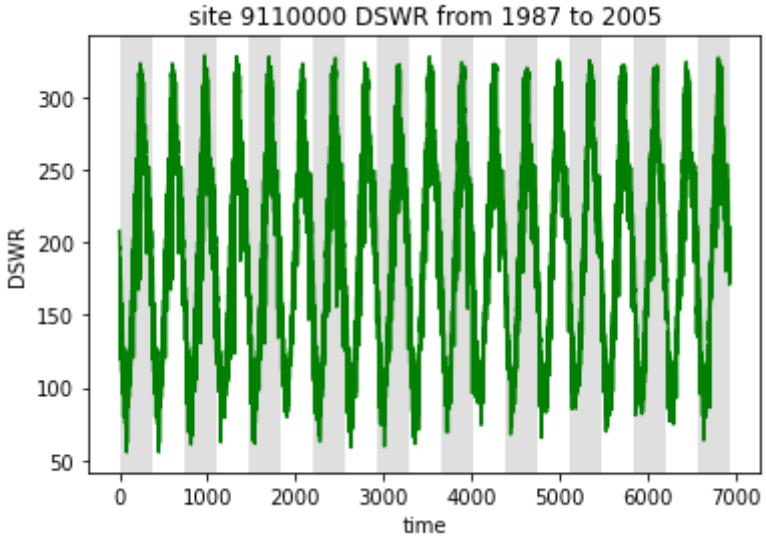

histogram site 9196500 DLWR from 1987 to 2005
1600
1400
1200
1000
800
600
400
200
0
100
150
200
250
300

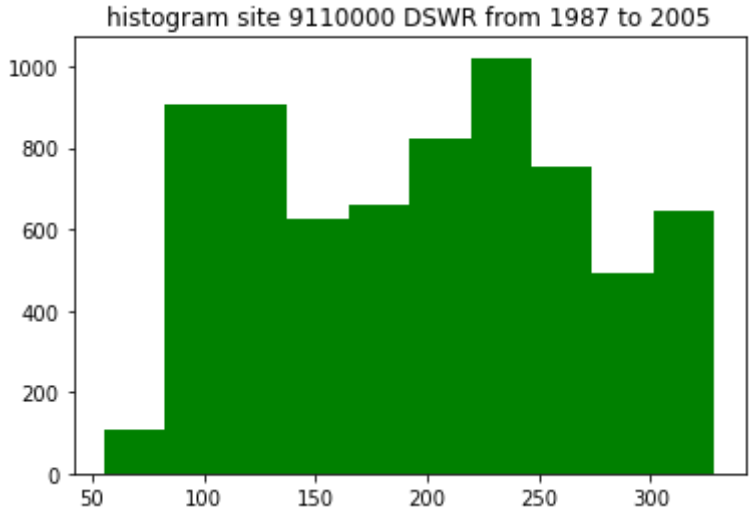

site 9107000 DLWR from 1987 to 2005
DLWR
300
250
200
150
100
0
1000
2000
3000
4000
5000
6000
7000
time

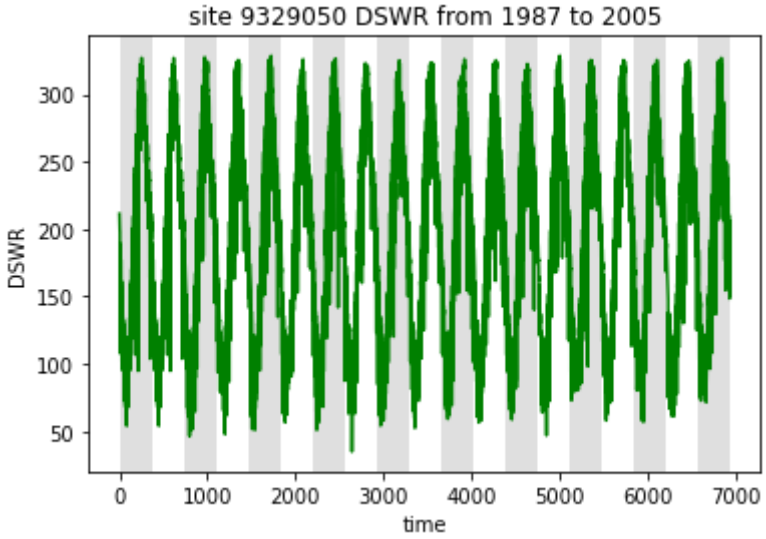

histogram site 9107000 DLWR from 1987 to 2005
1200
1000
800
600
400
200
0
100
150
200
250
300

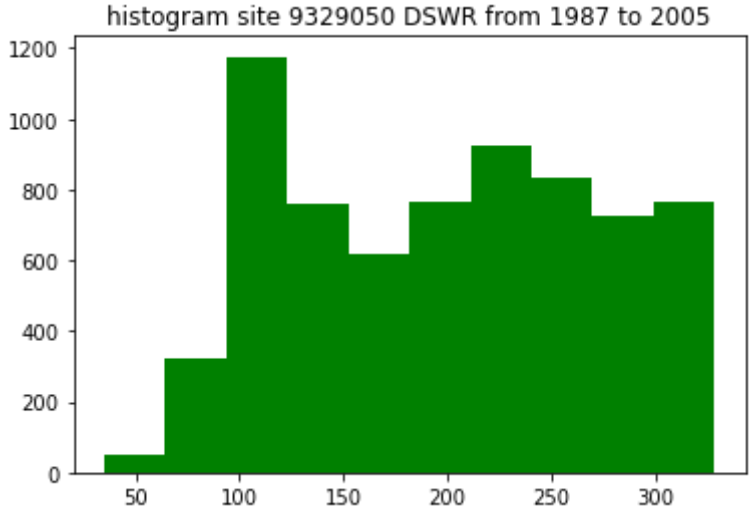

site 9210500 DLWR from 1987 to 2005
DLWR
time
350
300
250
200
150
0
1000
2000
3000
4000
5000
6000
7000

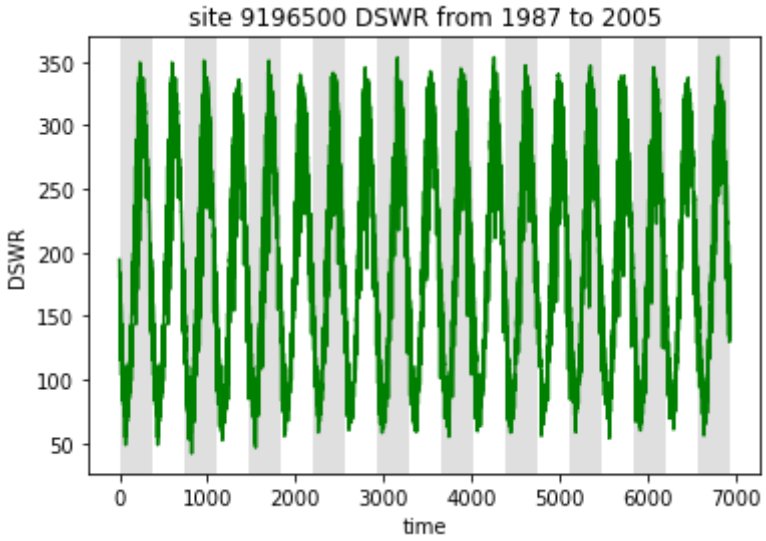

histogram site 9210500 DLWR from 1987 to 2005
1400
1200
1000
800
600
400
200
0
150
200
250
300
350

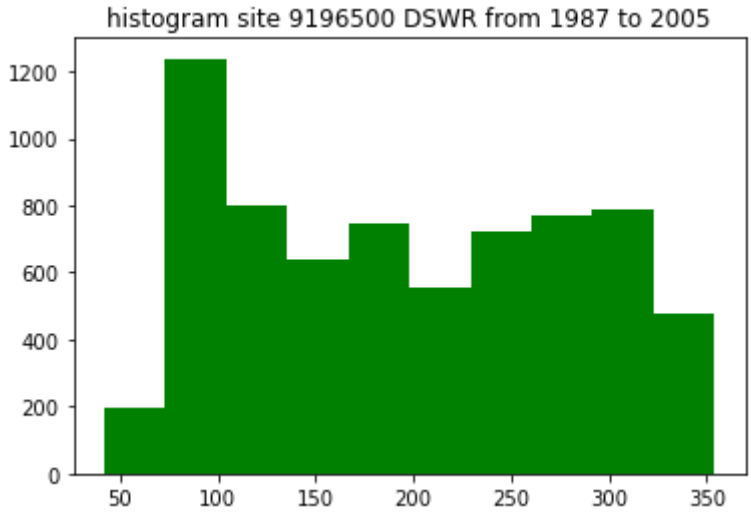

site 9110000 DSWR from 1987 to 2005
DSWR
time
300
250
200
150
100
50
0
1000
2000
3000
4000
5000
6000
7000

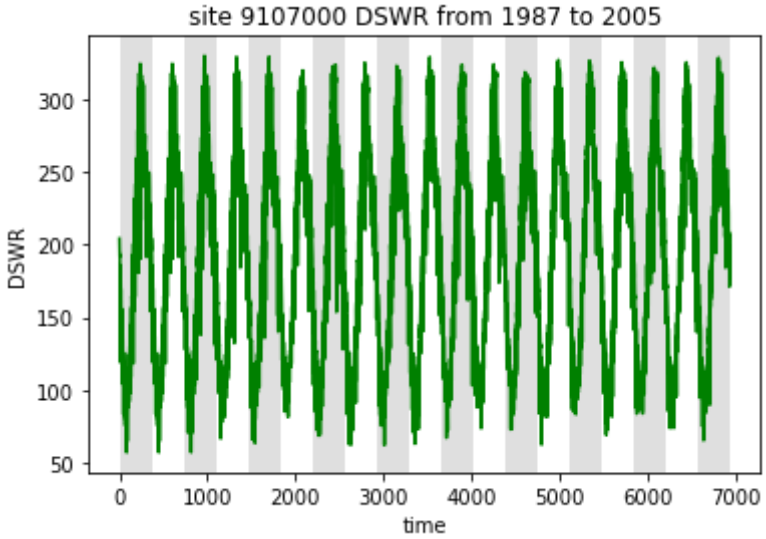

histogram site 9110000 DSWR from 1987 to 2005
1000
800
600
400
200
0
50
100
150
200
250
300

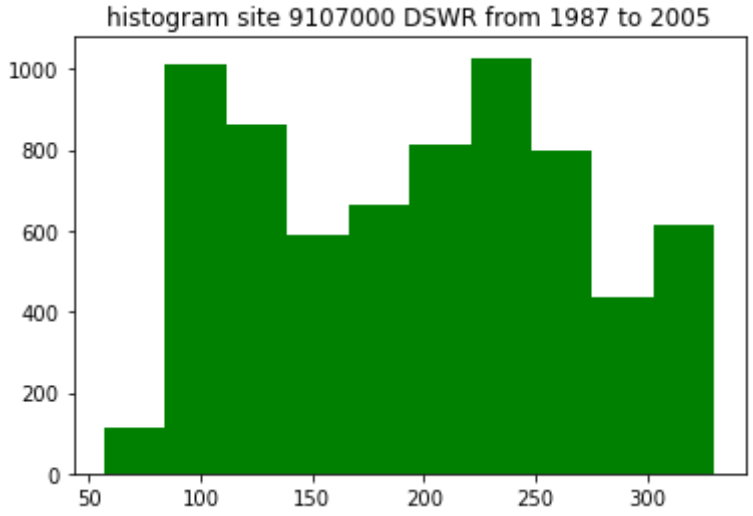

site 9329050 DSWR from 1987 to 2005
DSWR
300
250
200
150
100
50
0
1000
2000
3000
4000
5000
6000
7000
time

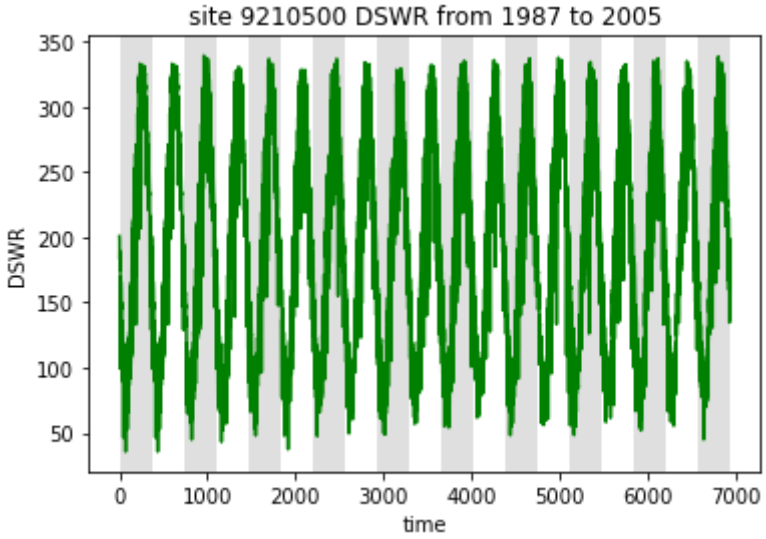

histogram site 9329050 DSWR from 1987 to 2005
1200
1000
800
600
400
200
0
50
100
150
200
250
300

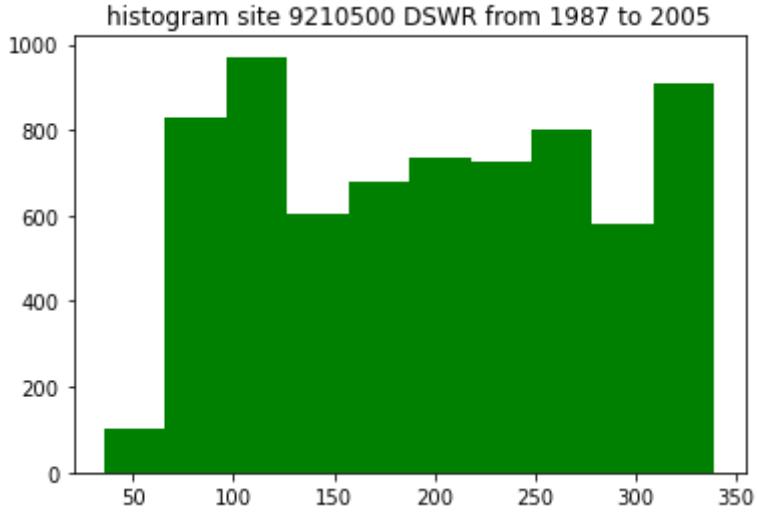

site 9196500 DSWR from 1987 to 2005
DSWR
time
0
1000
2000
3000
4000
5000
6000
7000
50
100
150
200
250
300
350

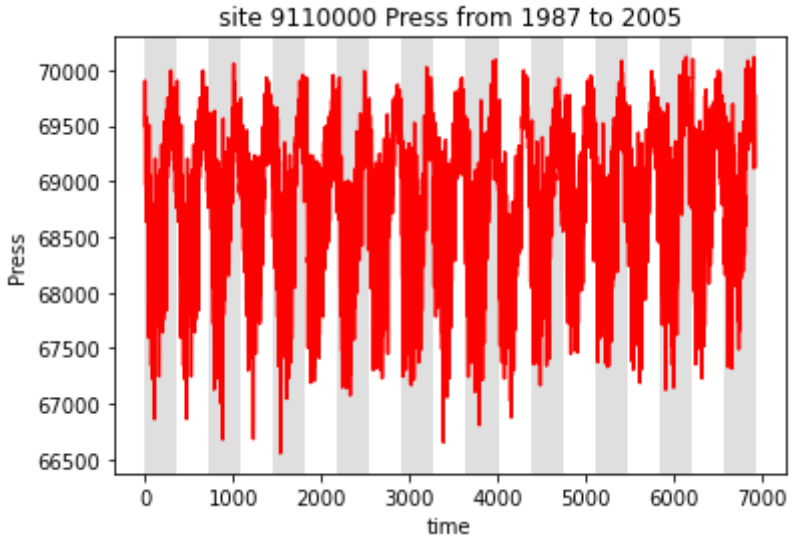

histogram site 9196500 DSWR from 1987 to 2005
0
200
400
600
800
1000
1200
50
100
150
200
250
300
350

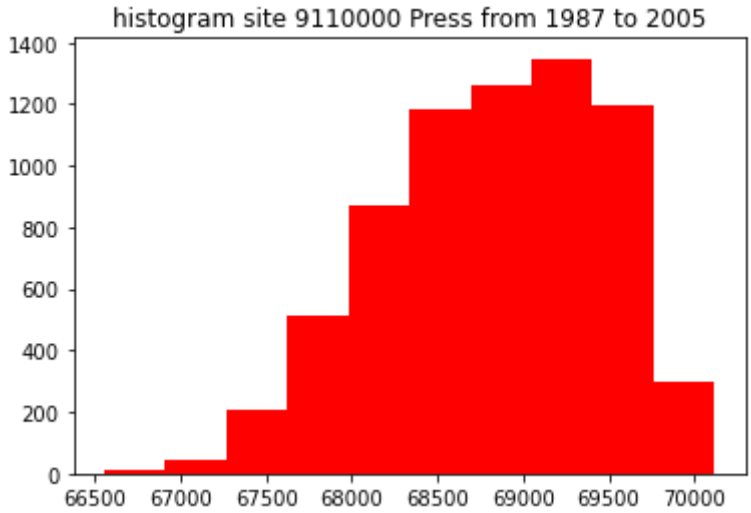

site 9107000 DSWR from 1987 to 2005
DSWR
time
0
1000
2000
3000
4000
5000
6000
7000
50
100
150
200
250
300

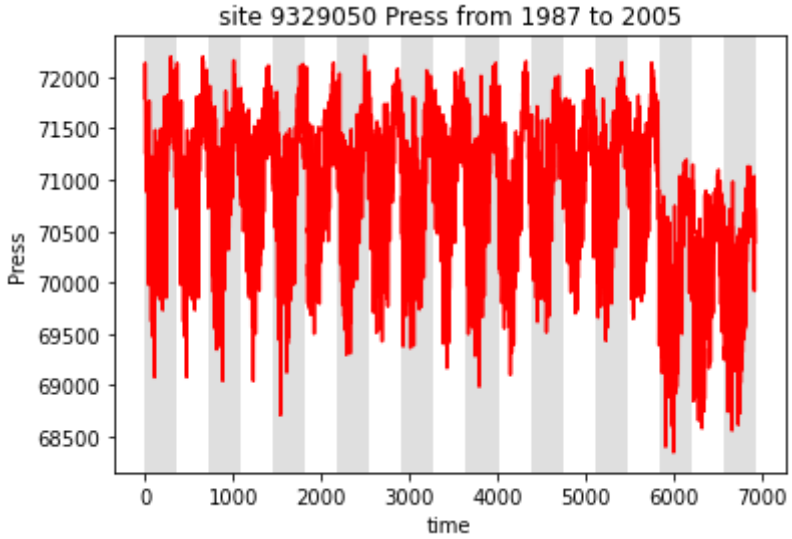

histogram site 9107000 DSWR from 1987 to 2005
1000
800
600
400
200
0
50
100
150
200
250
300

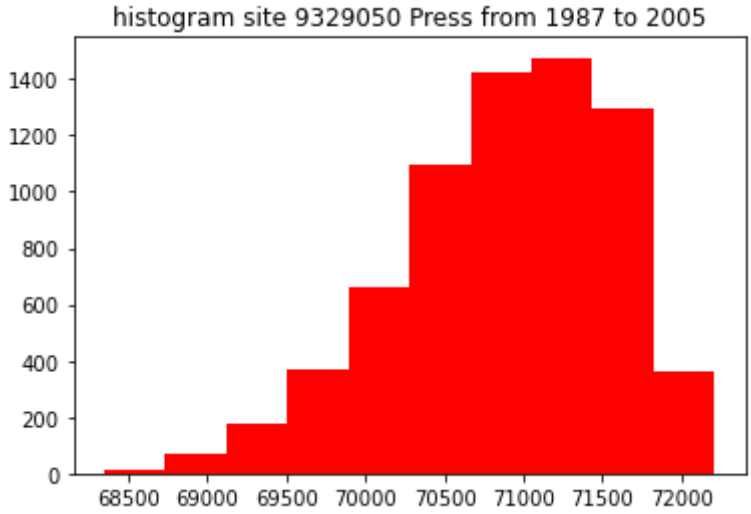

site 9210500 DSWR from 1987 to 2005
DSWR
350
300
250
200
150
100
50
0
1000
2000
3000
4000
5000
6000
7000
time

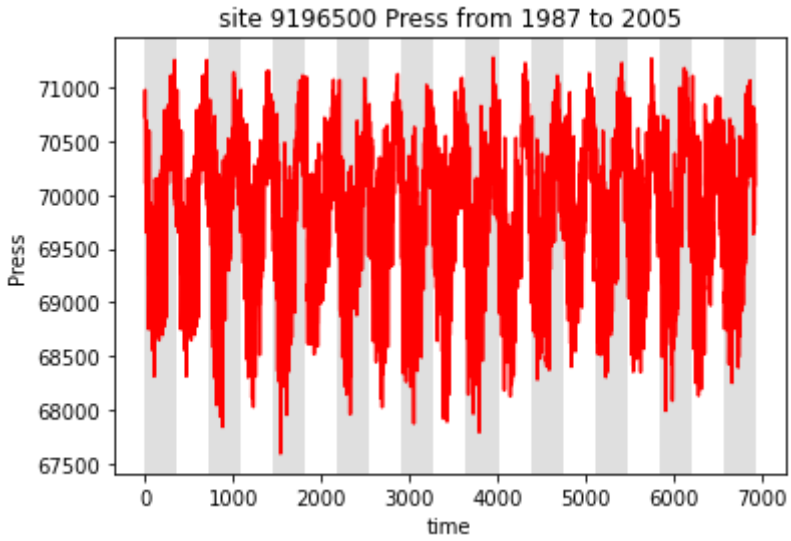

histogram site 9210500 DSWR from 1987 to 2005
1000
800
600
400
200
0
50
100
150
200
250
300
350

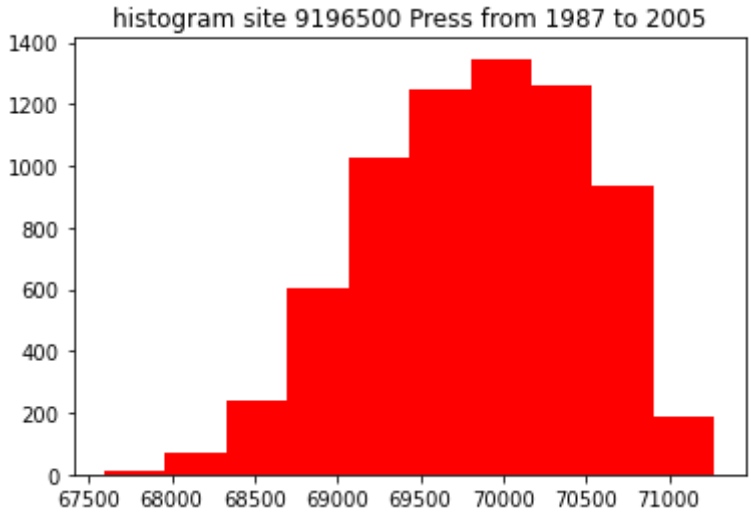

site 9110000 Press from 1987 to 2005
Press
time
70000
69500
69000
68500
68000
67500
67000
66500
0
1000
2000
3000
4000
5000
6000
7000

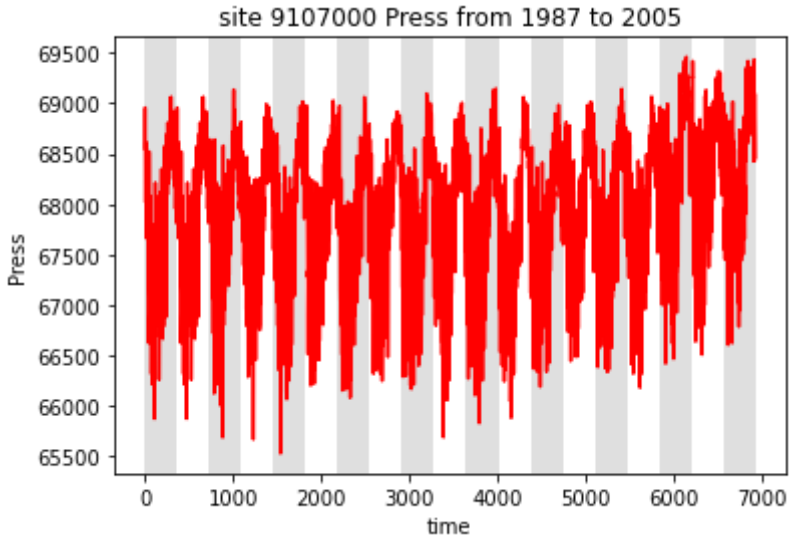

histogram site 9110000 Press from 1987 to 2005
1400
1200
1000
800
600
400
200
0
66500
67000
67500
68000
68500
69000
69500
70000

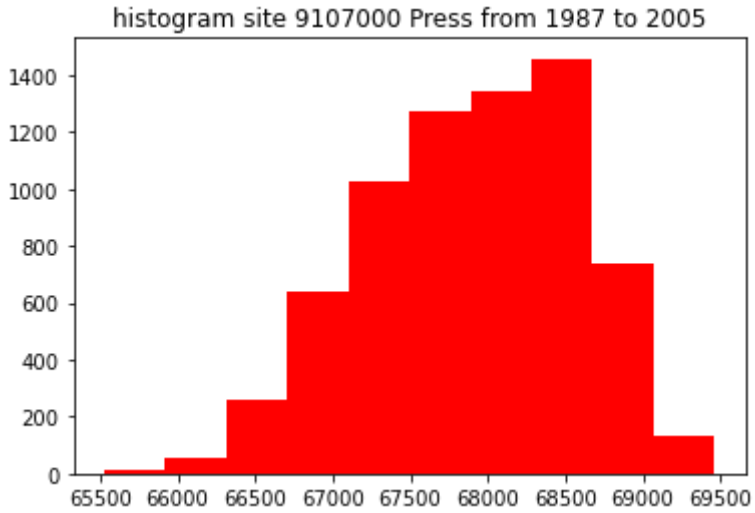

site 9329050 Press from 1987 to 2005
Press
time
72000
71500
71000
70500
70000
69500
69000
68500
0
1000
2000
3000
4000
5000
6000
7000

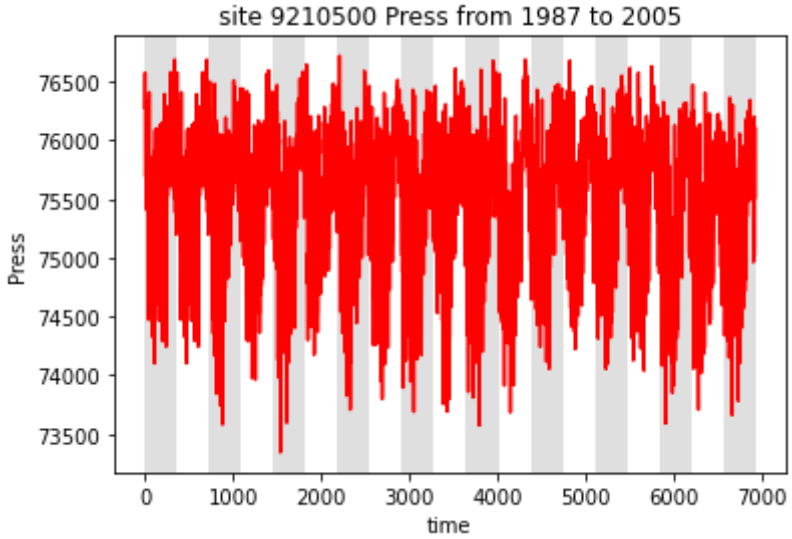

histogram site 9329050 Press from 1987 to 2005
1400
1200
1000
800
600
400
200
0
68500
69000
69500
70000
70500
71000
71500
72000

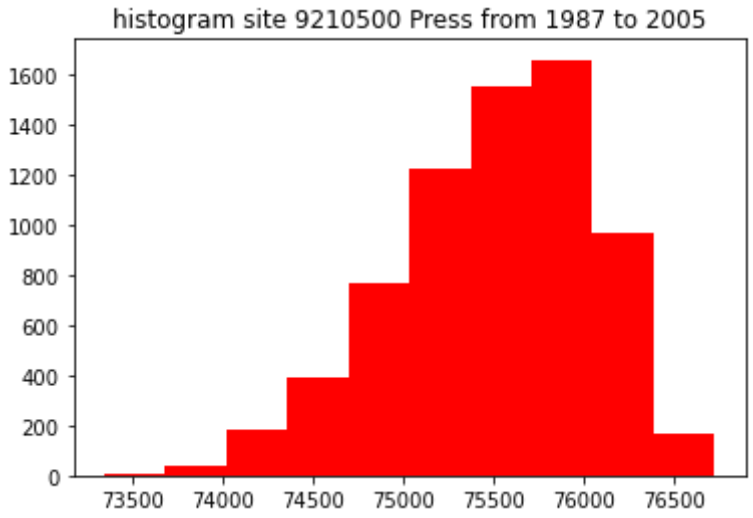

site 9196500 Press from 1987 to 2005
Press
71000
70500
70000
69500
69000
68500
68000
67500
0
1000
2000
3000
4000
5000
6000
7000
time

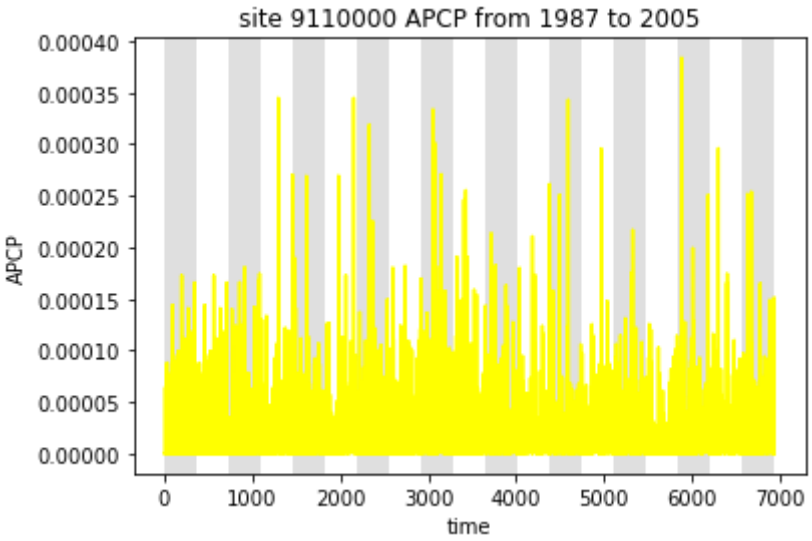

histogram site 9196500 Press from 1987 to 2005
1400
1200
1000
800
600
400
200
0
67500
68000
68500
69000
69500
70000
70500
71000

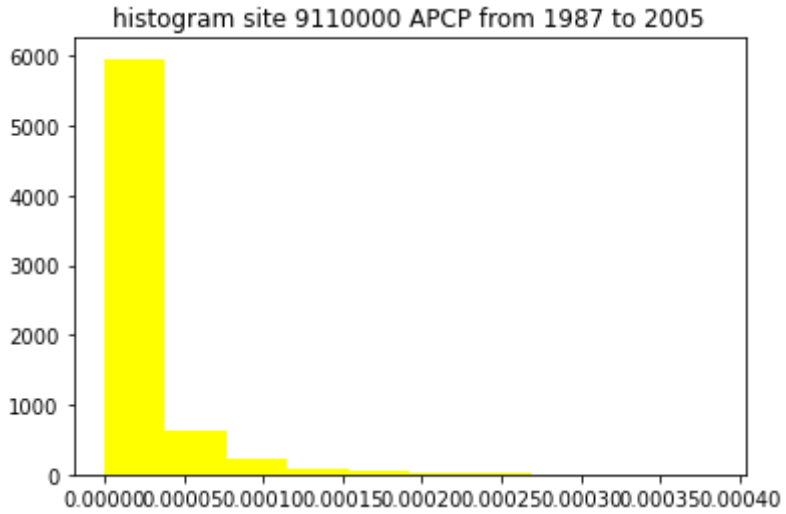

site 9107000 Press from 1987 to 2005
Press
time
69500
69000
68500
68000
67500
67000
66500
66000
65500
0
1000
2000
3000
4000
5000
6000
7000

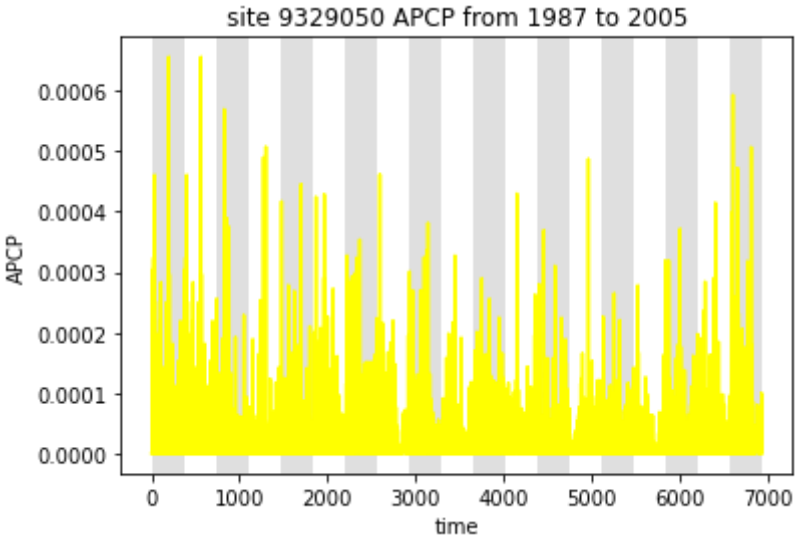

histogram site 9107000 Press from 1987 to 2005
1400
1200
1000
800
600
400
200
0
65500
66000
66500
67000
67500
68000
68500
69000
69500

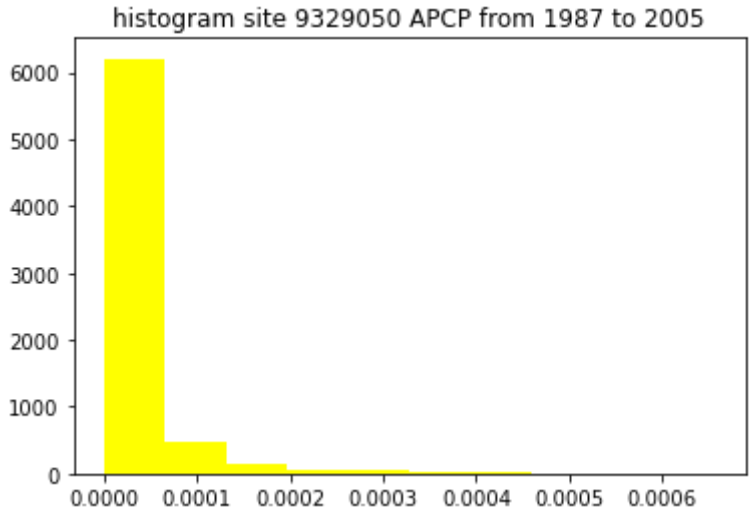

site 9210500 Press from 1987 to 2005
Press
time
76500
76000
75500
75000
74500
74000
73500
0
1000
2000
3000
4000
5000
6000
7000

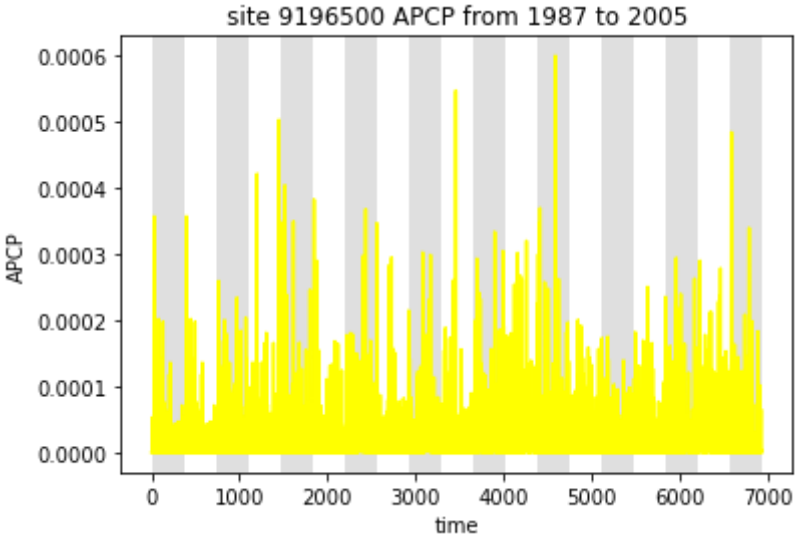

histogram site 9210500 Press from 1987 to 2005
1600
1400
1200
1000
800
600
400
200
0
73500
74000
74500
75000
75500
76000
76500

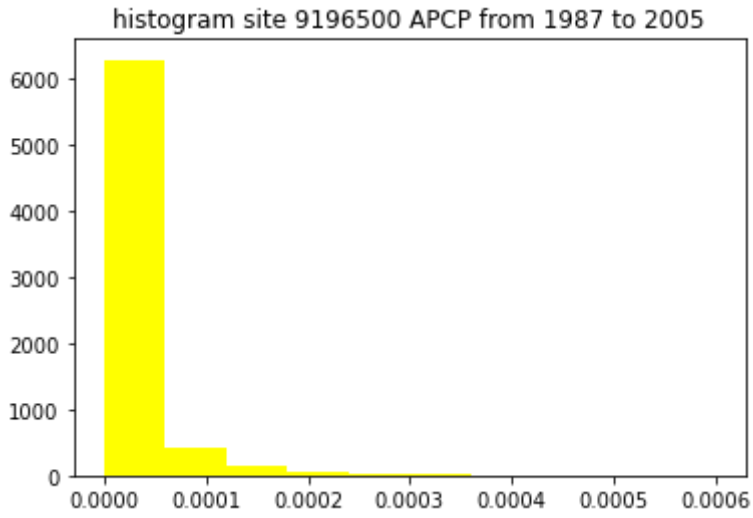

site 9110000 APCP from 1987 to 2005
APCP
0.00040
0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
0
1000
2000
3000
4000
5000
6000
7000
time

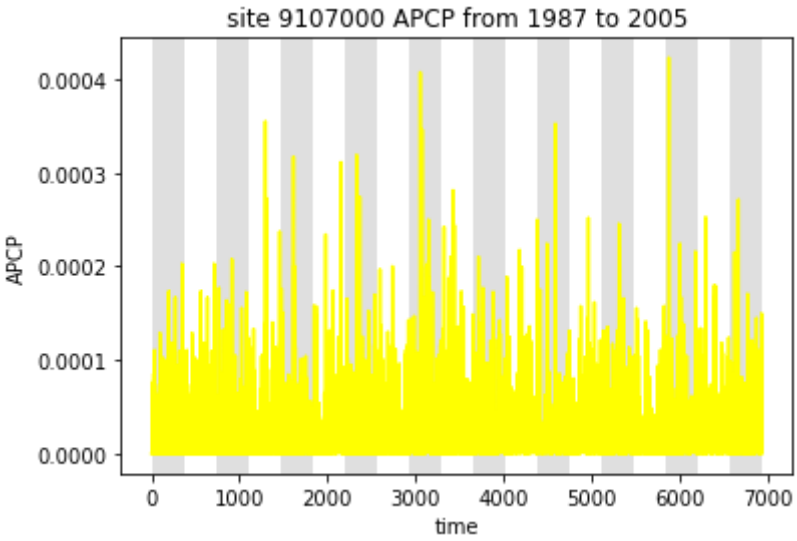

histogram site 9110000 APCP from 1987 to 2005
6000
5000
4000
3000
2000
1000
0
0.00000
0.00005
0.00010
0.00015
0.00020
0.00025
0.00030
0.00035
0.00040

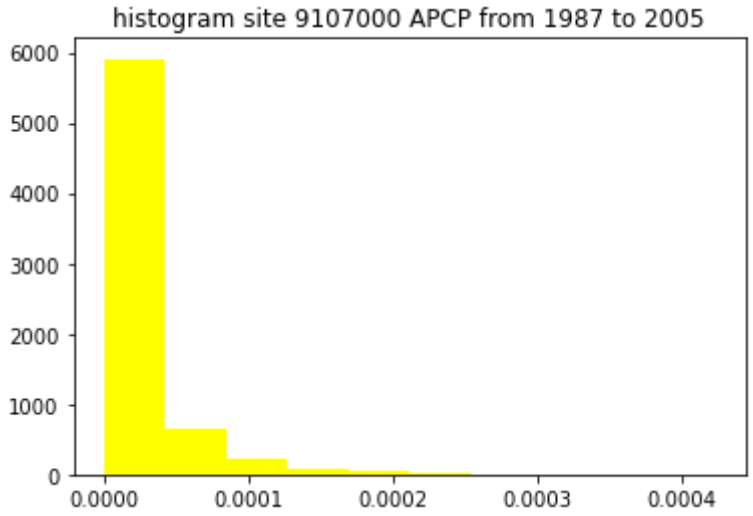

site 9329050 APCP from 1987 to 2005
APCP
time
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0
1000
2000
3000
4000
5000
6000
7000

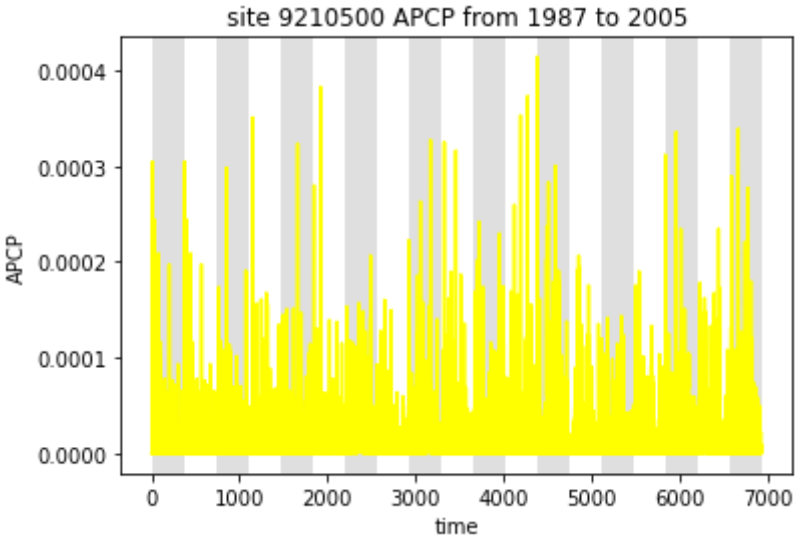

histogram site 9329050 APCP from 1987 to 2005
0
1000
2000
3000
4000
5000
6000
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006

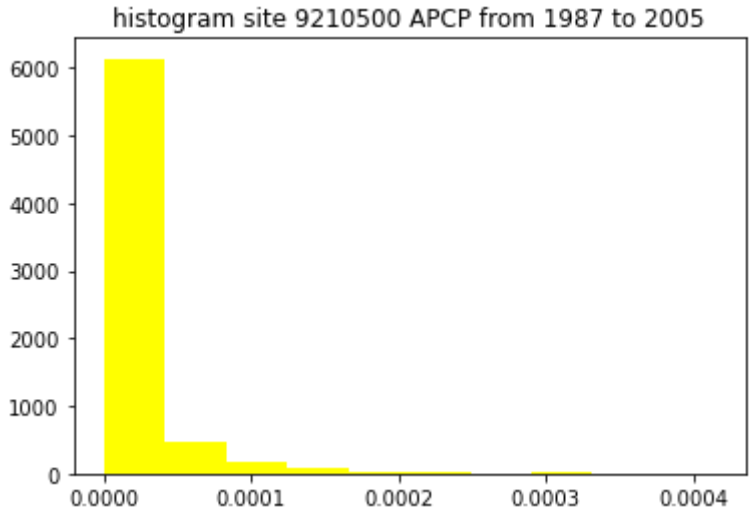

site 9196500 APCP from 1987 to 2005
APCP
time
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0
1000
2000
3000
4000
5000
6000
7000

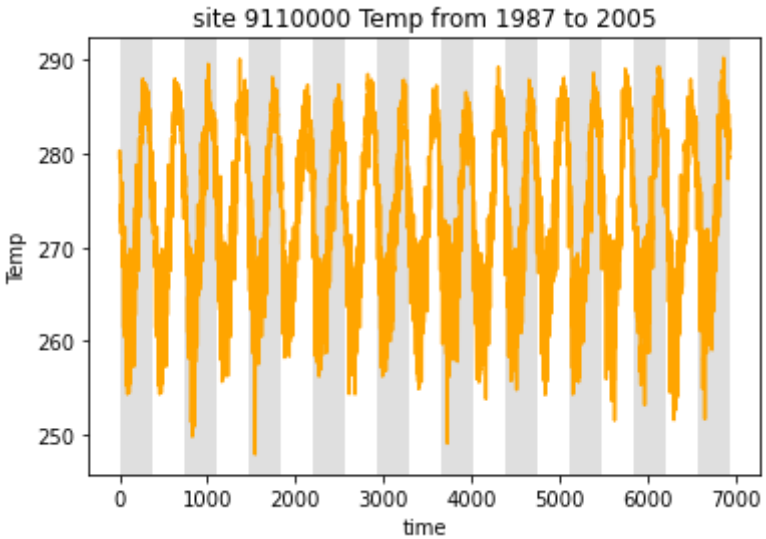

histogram site 9196500 APCP from 1987 to 2005
6000
5000
4000
3000
2000
1000
0
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006

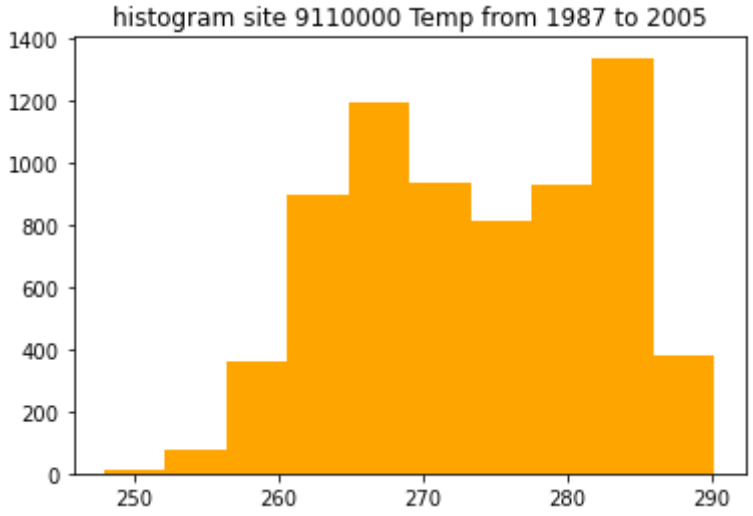

site 9107000 APCP from 1987 to 2005
APCP
0.0004
0.0003
0.0002
0.0001
0.0000
0
1000
2000
3000
4000
5000
6000
7000
time

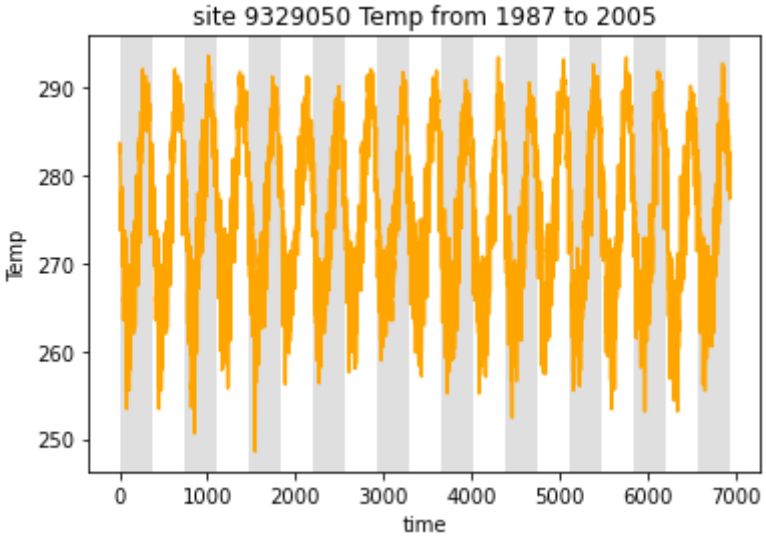

histogram site 9107000 APCP from 1987 to 2005
6000
5000
4000
3000
2000
1000
0
0.0000
0.0001
0.0002
0.0003
0.0004

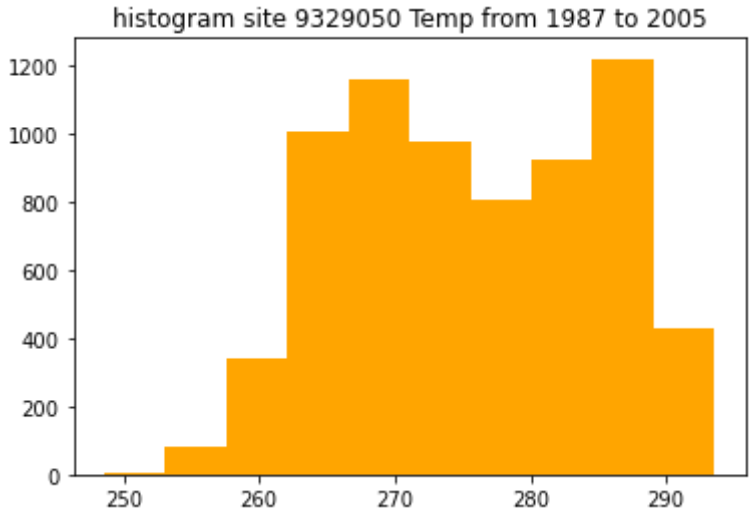

site 9210500 APCP from 1987 to 2005
APCP
0.0004
0.0003
0.0002
0.0001
0.0000
0
1000
2000
3000
4000
5000
6000
7000
time

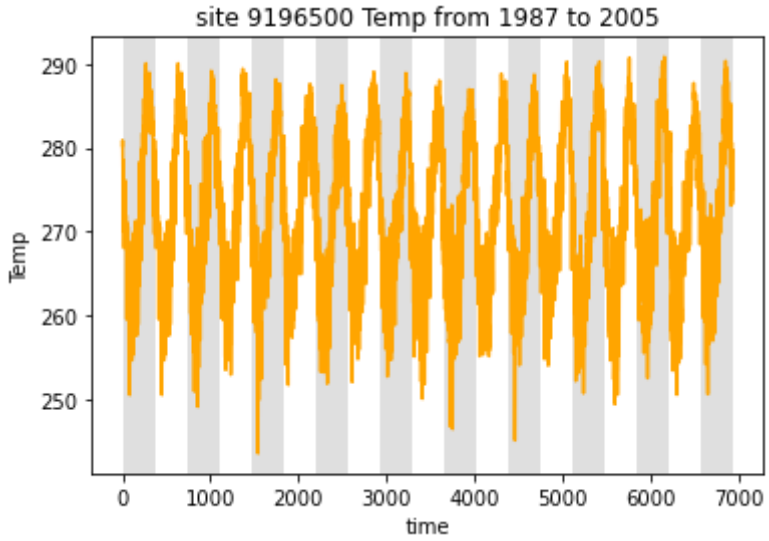

histogram site 9210500 APCP from 1987 to 2005
6000
5000
4000
3000
2000
1000
0
0.0000
0.0001
0.0002
0.0003
0.0004

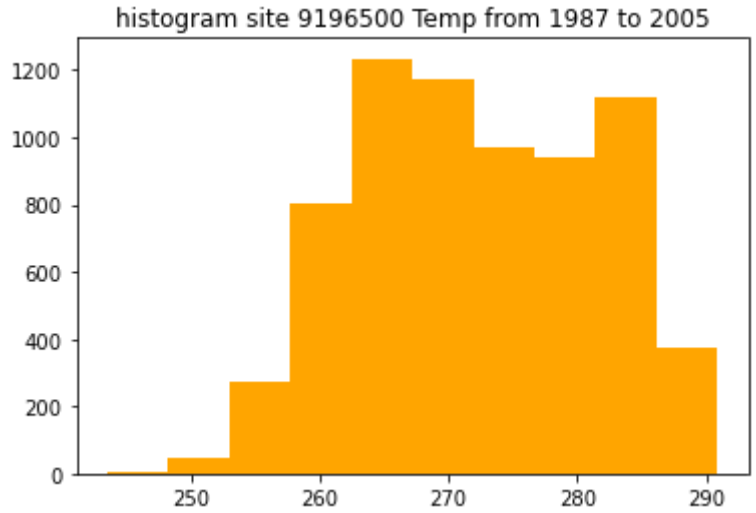

site 9110000 Temp from 1987 to 2005
Temp
290
280
270
260
250
0
1000
2000
3000
4000
5000
6000
7000
time

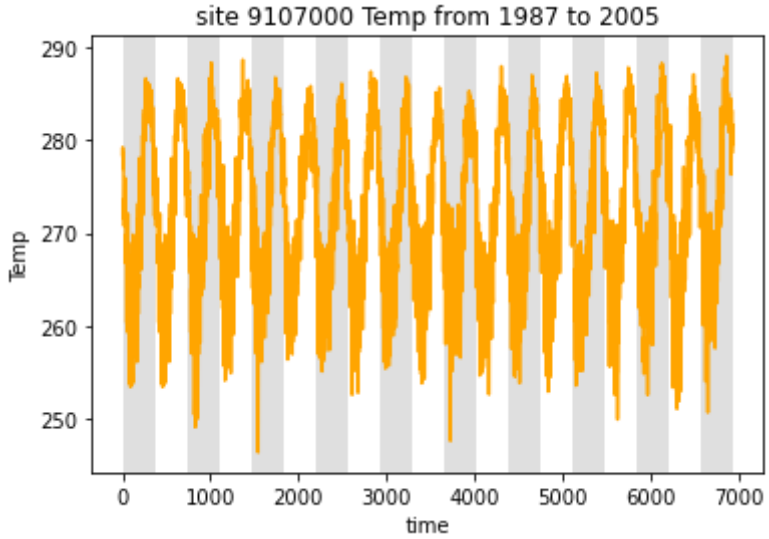

histogram site 9110000 Temp from 1987 to 2005
1400
1200
1000
800
600
400
200
0
250
260
270
280
290

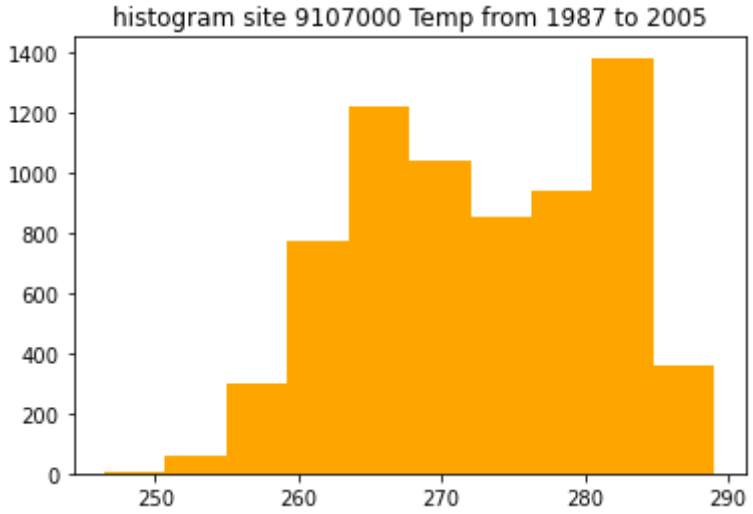

site 9329050 Temp from 1987 to 2005
Temp
290
280
270
260
250
0
1000
2000
3000
4000
5000
6000
7000
time

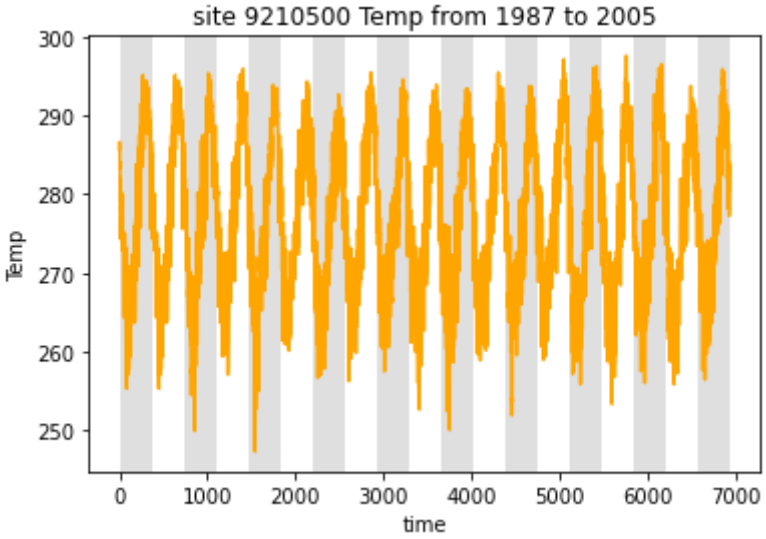

histogram site 9329050 Temp from 1987 to 2005
1200
1000
800
600
400
200
0
250
260
270
280
290

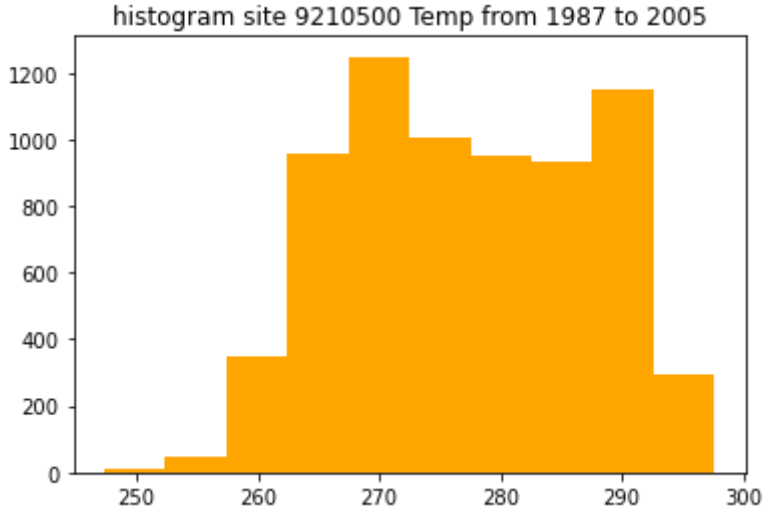

site 9196500 Temp from 1987 to 2005
Temp
time
290
280
270
260
250
0
1000
2000
3000
4000
5000
6000
7000

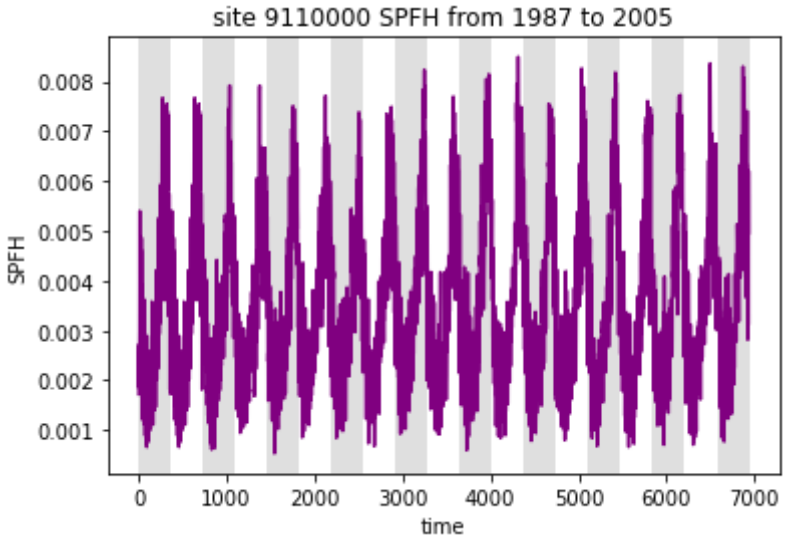

histogram site 9196500 Temp from 1987 to 2005
1200
1000
800
600
400
200
0
250
260
270
280
290

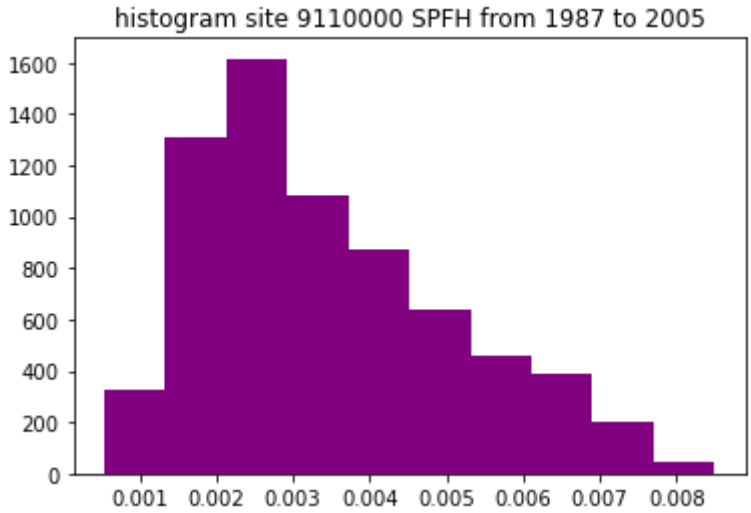

site 9107000 Temp from 1987 to 2005
Temp
time
290
280
270
260
250
0
1000
2000
3000
4000
5000
6000
7000

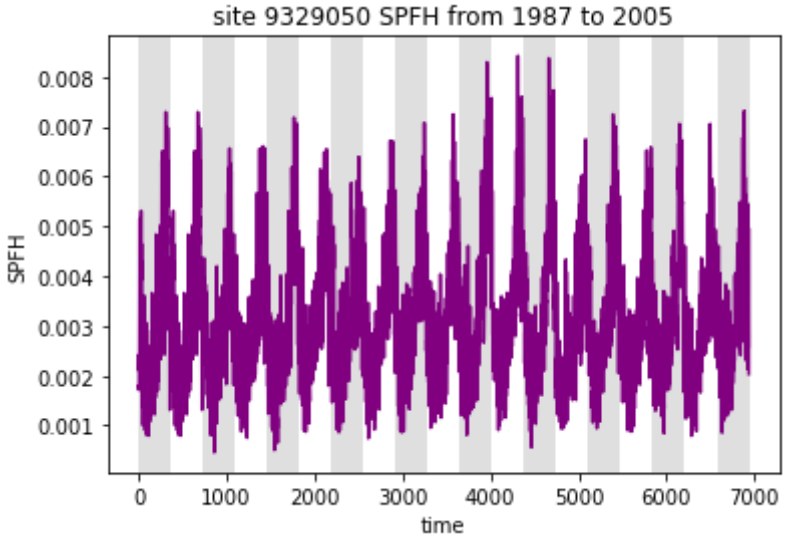

histogram site 9107000 Temp from 1987 to 2005
1400
1200
1000
800
600
400
200
0
250
260
270
280
290

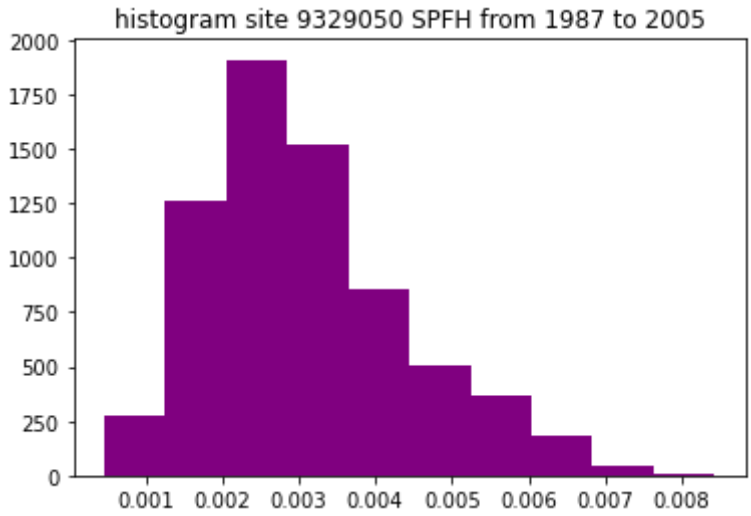

site 9210500 Temp from 1987 to 2005
Temp
300
290
280
270
260
250
0
1000
2000
3000
4000
5000
6000
7000
time

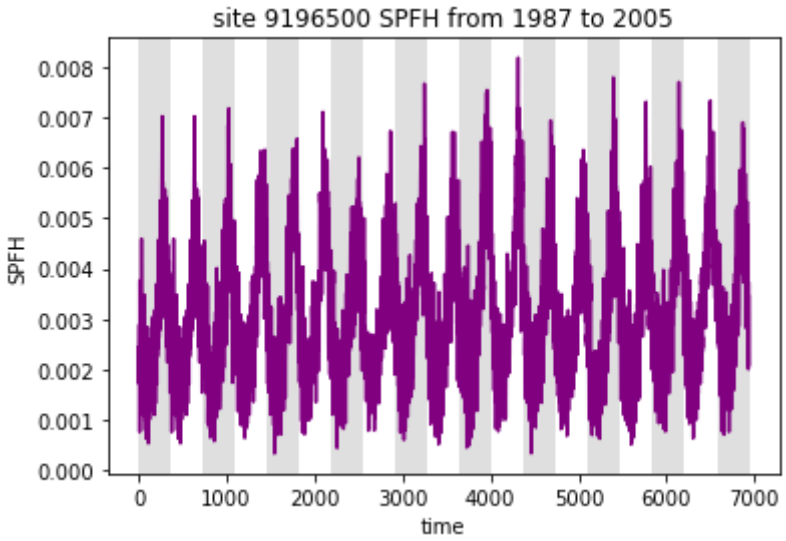

histogram site 9210500 Temp from 1987 to 2005
1200
1000
800
600
400
200
0
250
260
270
280
290
300

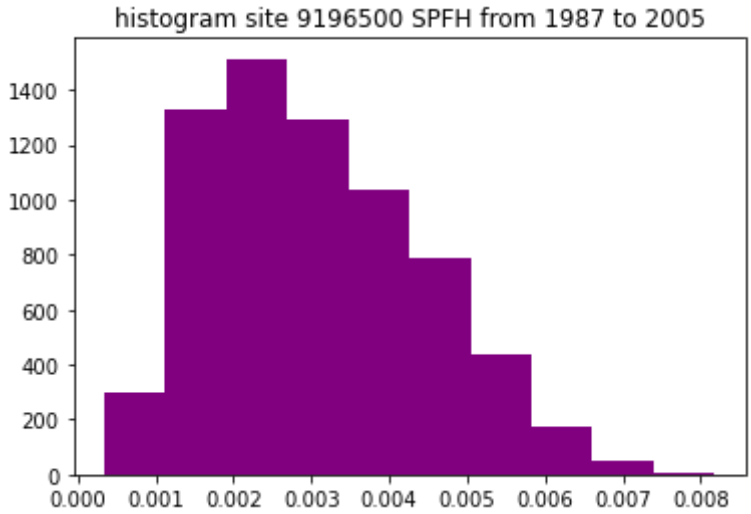

site 9110000 SPFH from 1987 to 2005
SPFH
time
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0
1000
2000
3000
4000
5000
6000
7000

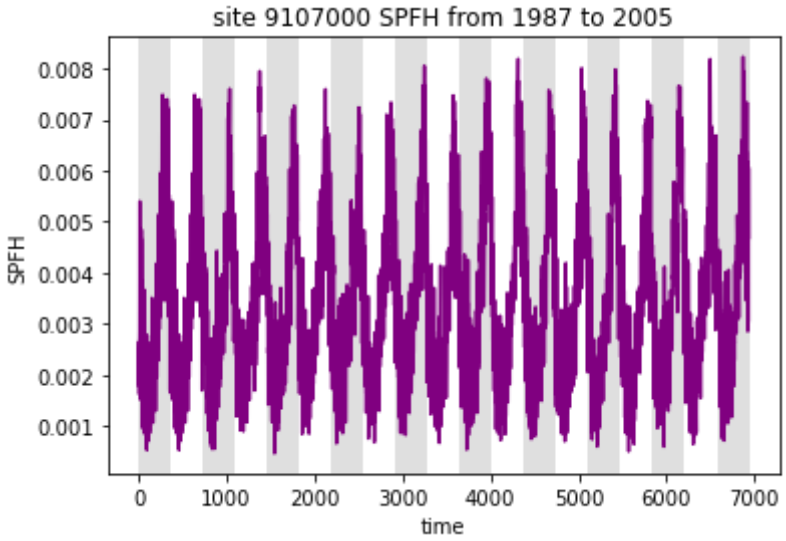

histogram site 9110000 SPFH from 1987 to 2005
0
200
400
600
800
1000
1200
1400
1600
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008

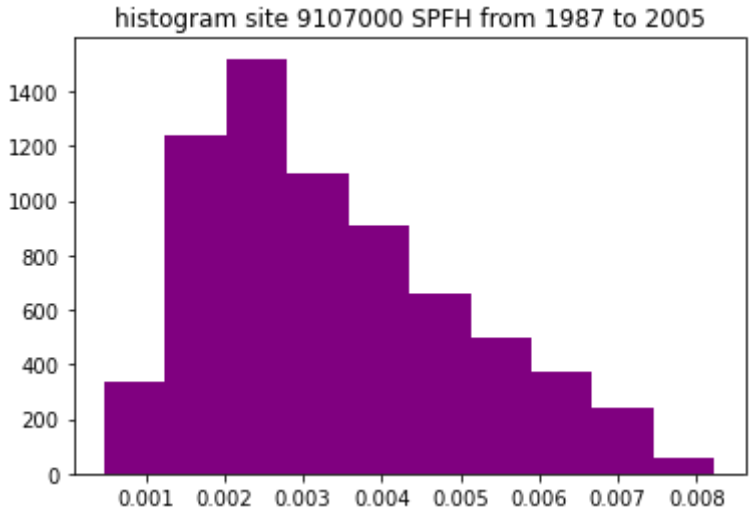

site 9329050 SPFH from 1987 to 2005
SPFH
time
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0
1000
2000
3000
4000
5000
6000
7000

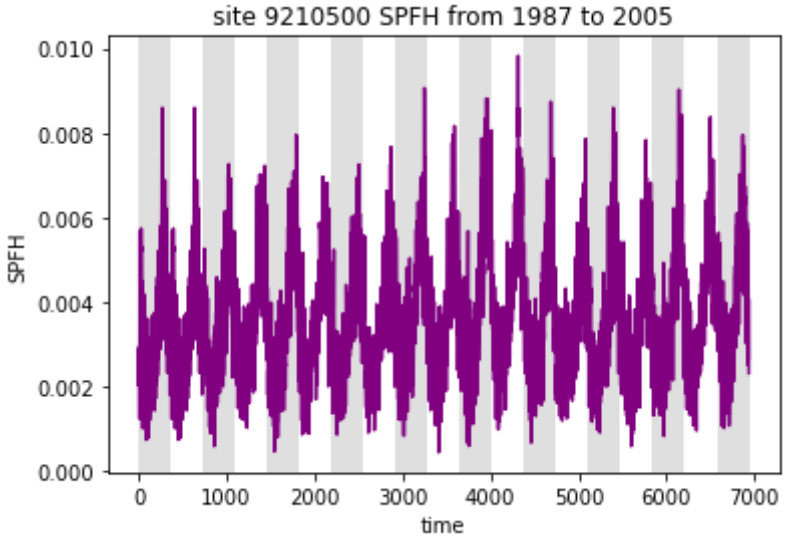

histogram site 9329050 SPFH from 1987 to 2005
2000
1750
1500
1250
1000
750
500
250
0
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008

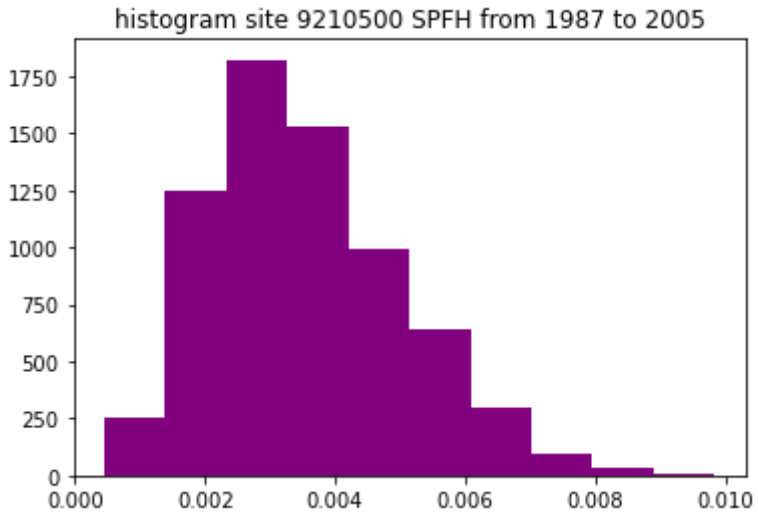

site 9196500 SPFH from 1987 to 2005
SPFH
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
1000
2000
3000
4000
5000
6000
7000
time

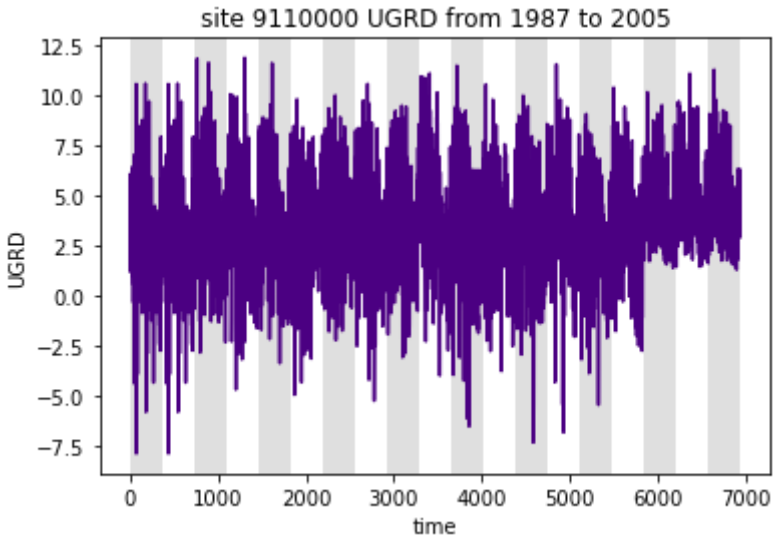

histogram site 9196500 SPFH from 1987 to 2005
1400
1200
1000
800
600
400
200
0
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008

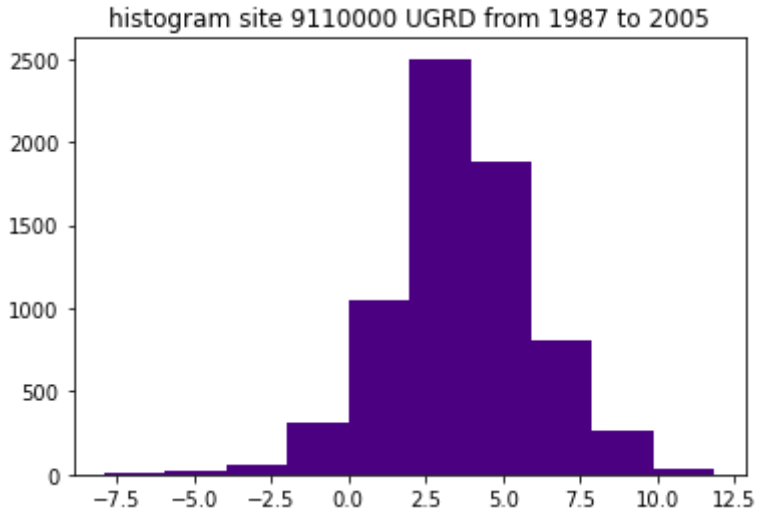

site 9107000 SPFH from 1987 to 2005
SPFH
time
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0
1000
2000
3000
4000
5000
6000
7000

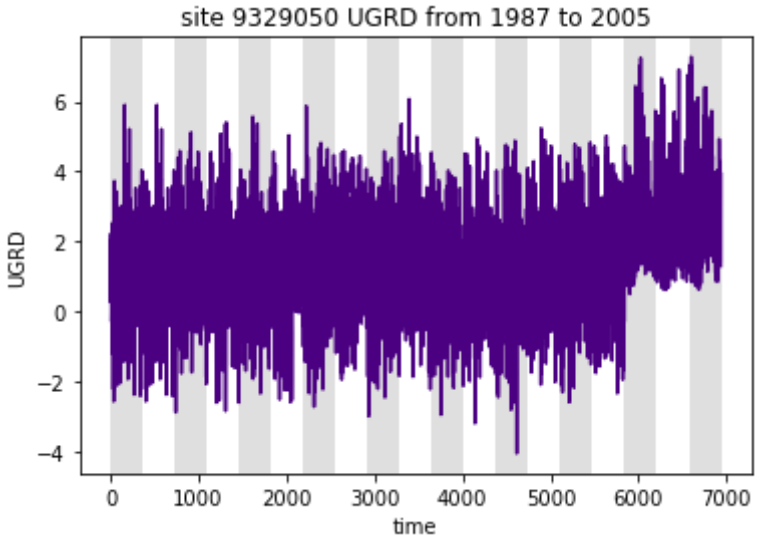

histogram site 9107000 SPFH from 1987 to 2005
0
200
400
600
800
1000
1200
1400
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008

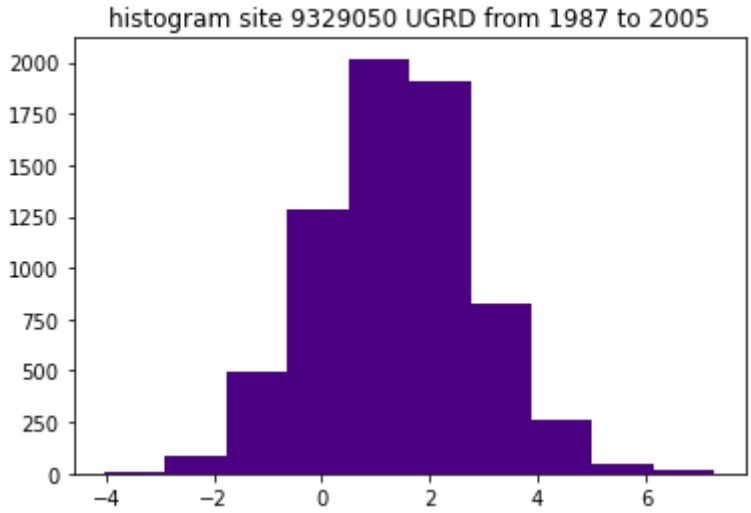

site 9210500 SPFH from 1987 to 2005
SPFH
time
0.000
0.002
0.004
0.006
0.008
0.010
0
1000
2000
3000
4000
5000
6000
7000

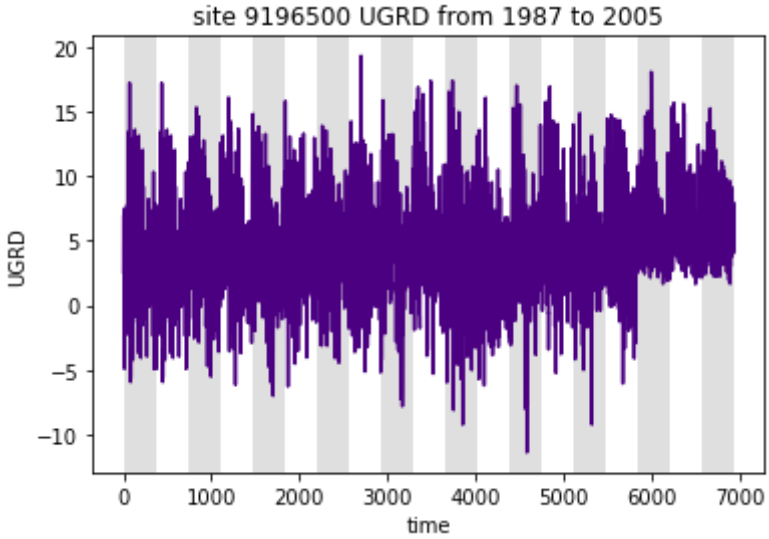

histogram site 9210500 SPFH from 1987 to 2005
1750
1500
1250
1000
750
500
250
0
0.000
0.002
0.004
0.006
0.008
0.010

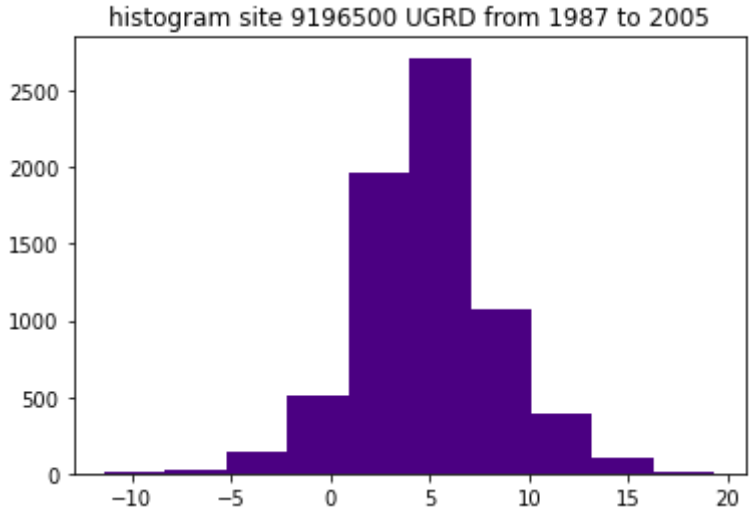

site 9110000 UGRD from 1987 to 2005
UGRD
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
0
1000
2000
3000
4000
5000
6000
7000
time

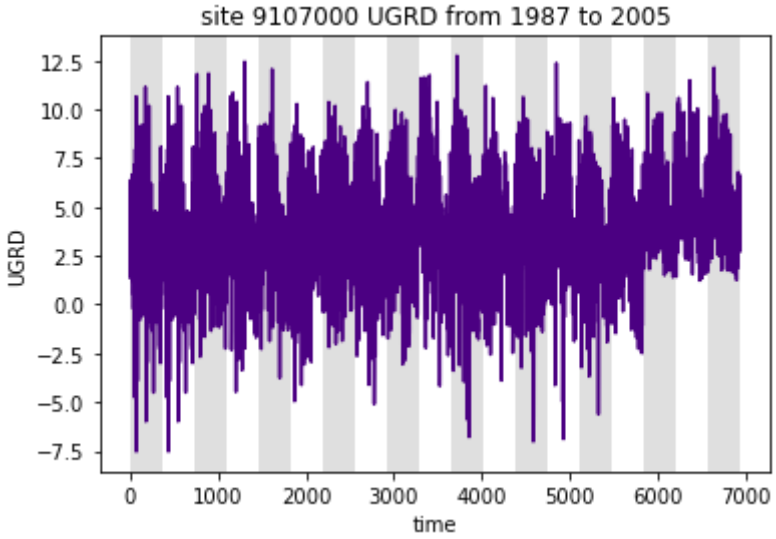

histogram site 9110000 UGRD from 1987 to 2005
2500
2000
1500
1000
500
0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5

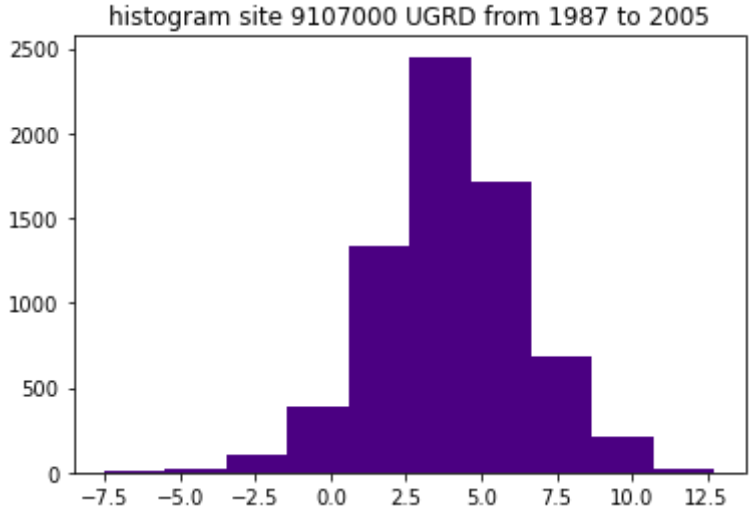

site 9329050 UGRD from 1987 to 2005
UGRD
time
6
4
2
0
−2
−4
0
1000
2000
3000
4000
5000
6000
7000

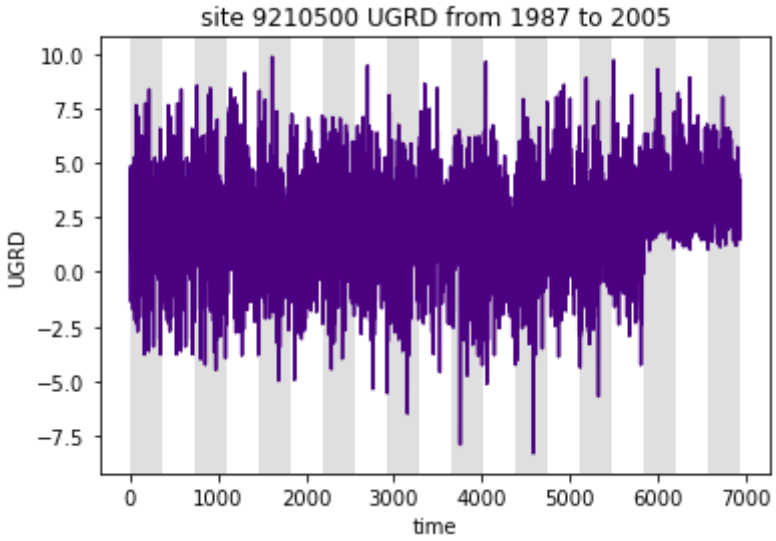

histogram site 9329050 UGRD from 1987 to 2005
2000
1750
1500
1250
1000
750
500
250
0
−4
−2
0
2
4
6

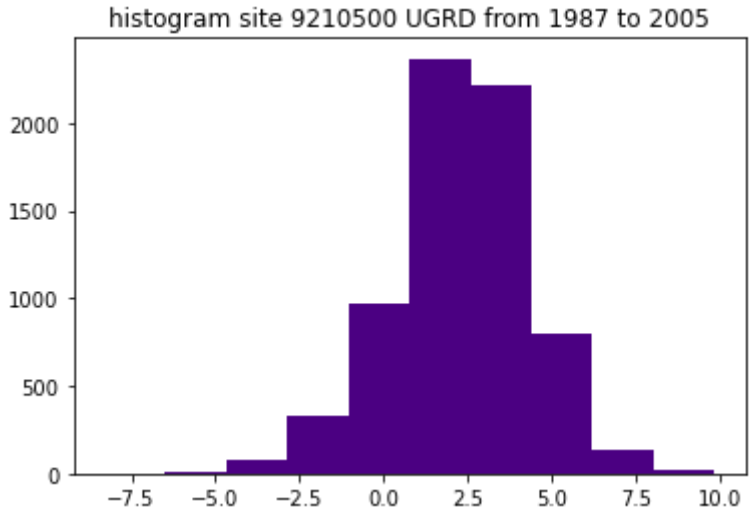

site 9196500 UGRD from 1987 to 2005
UGRD
time
20
15
10
5
0
−5
−10
0
1000
2000
3000
4000
5000
6000
7000

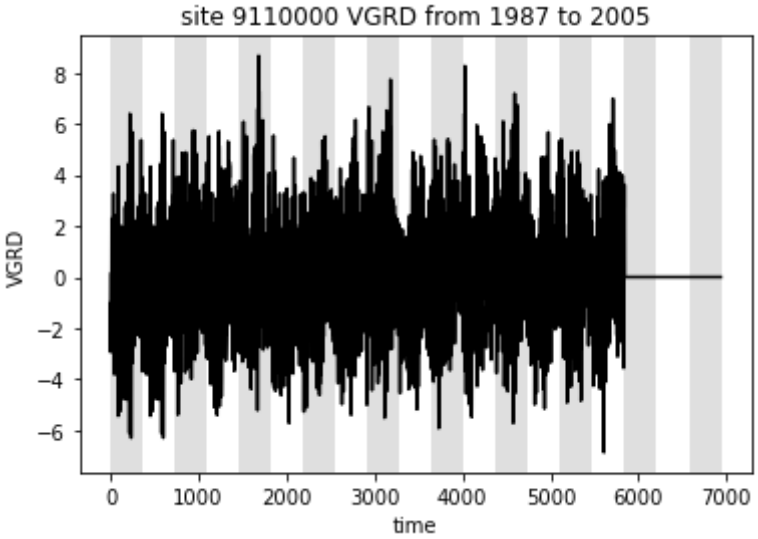

histogram site 9196500 UGRD from 1987 to 2005
2500
2000
1500
1000
500
0
−10
−5
0
5
10
15
20

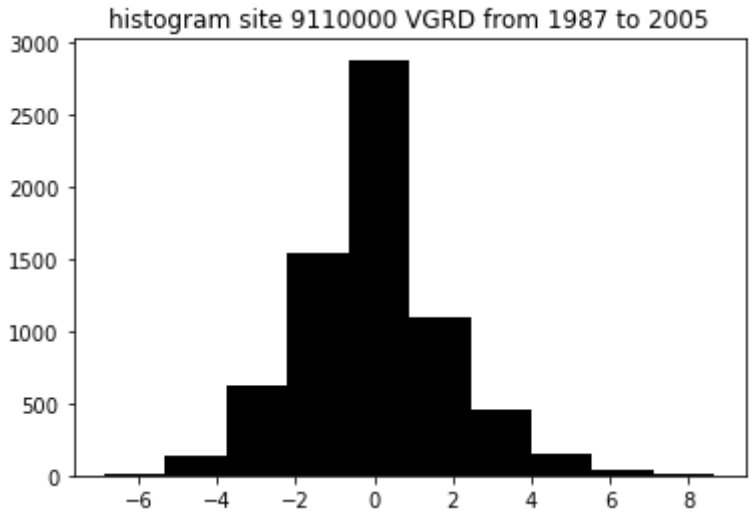

site 9107000 UGRD from 1987 to 2005
UGRD
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
0
1000
2000
3000
4000
5000
6000
7000
time

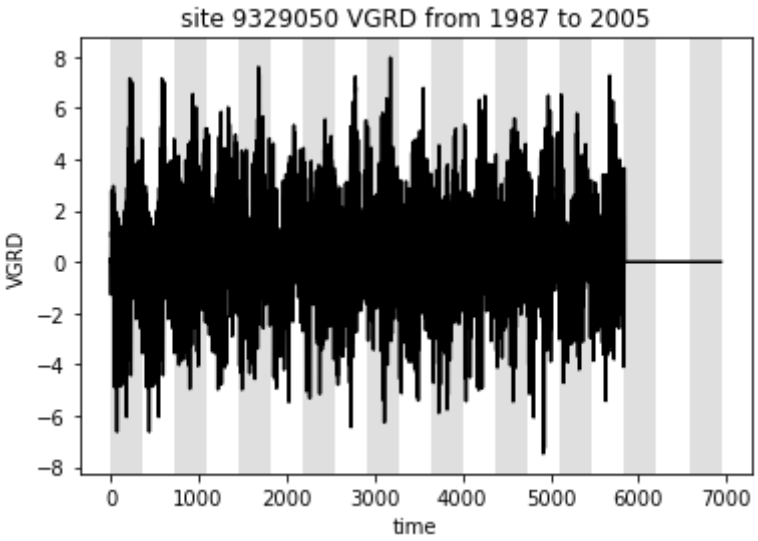

histogram site 9107000 UGRD from 1987 to 2005
2500
2000
1500
1000
500
0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5

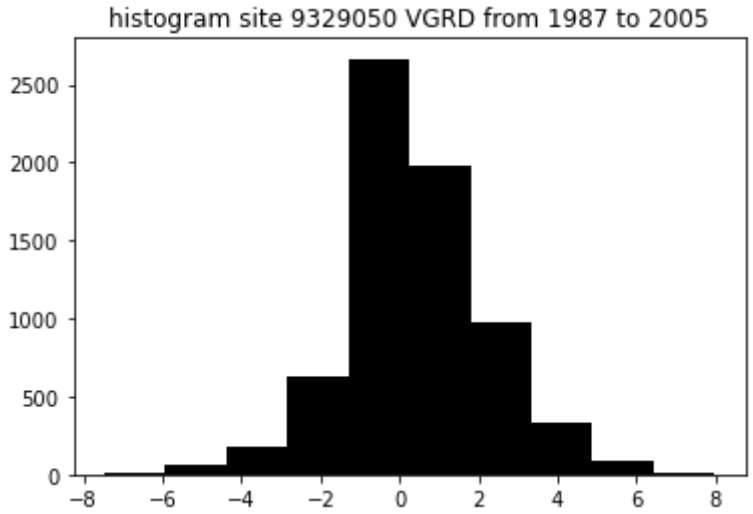

site 9210500 UGRD from 1987 to 2005
UGRD
time
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
0
1000
2000
3000
4000
5000
6000
7000

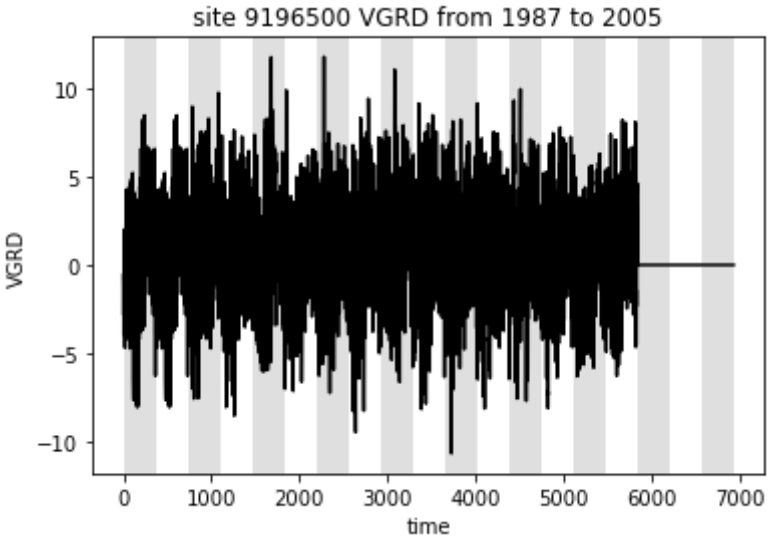

histogram site 9210500 UGRD from 1987 to 2005
2000
1500
1000
500
0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

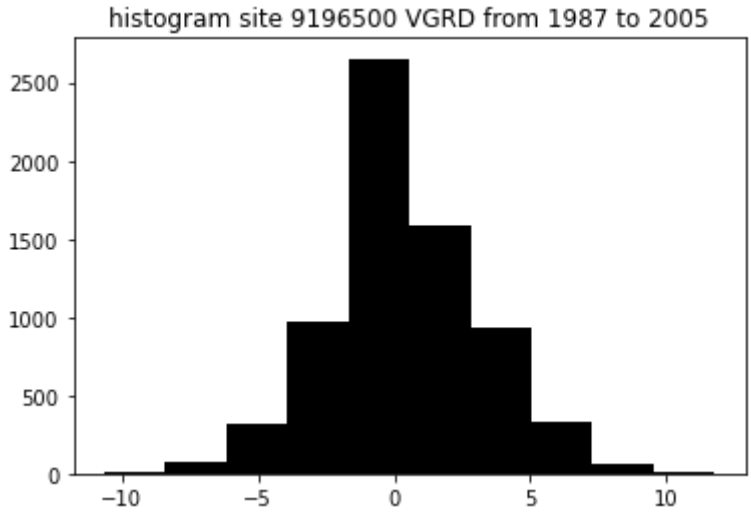

site 9110000 VGRD from 1987 to 2005
VGRD
time
8
6
4
2
0
−2
−4
−6
0
1000
2000
3000
4000
5000
6000
7000

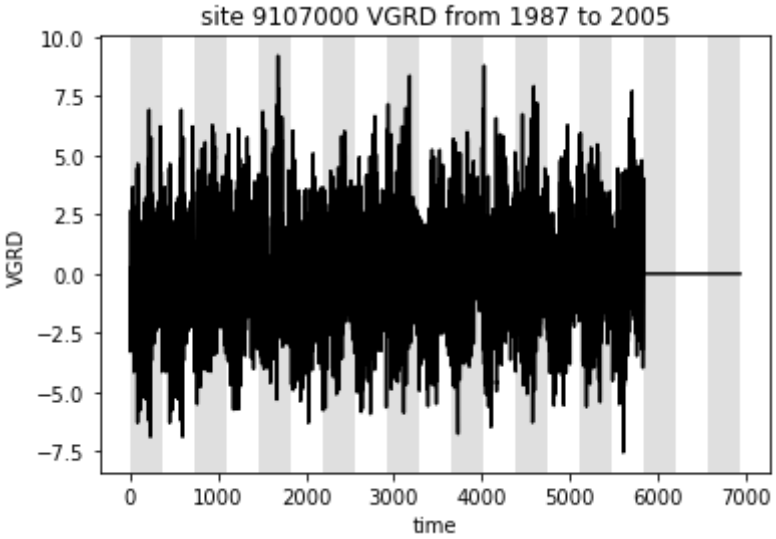

histogram site 9110000 VGRD from 1987 to 2005
3000
2500
2000
1500
1000
500
0
−6
−4
−2
0
2
4
6
8

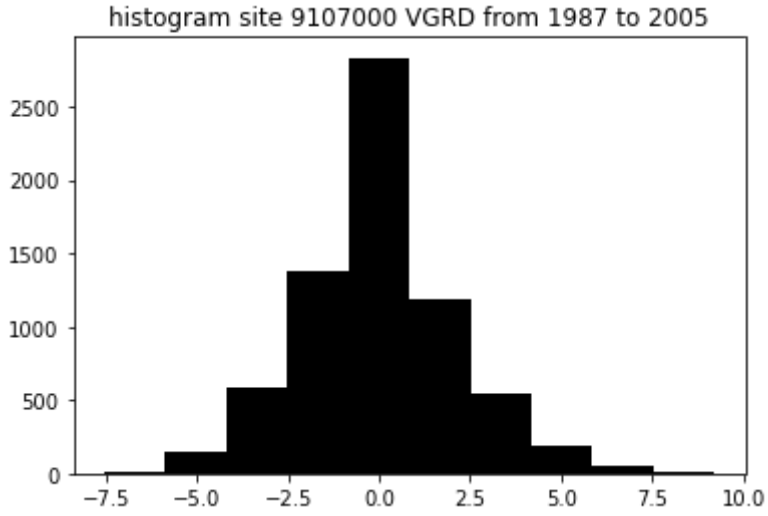

site 9329050 VGRD from 1987 to 2005
VGRD
8
6
4
2
0
−2
−4
−6
−8
0
1000
2000
3000
4000
5000
6000
7000
time

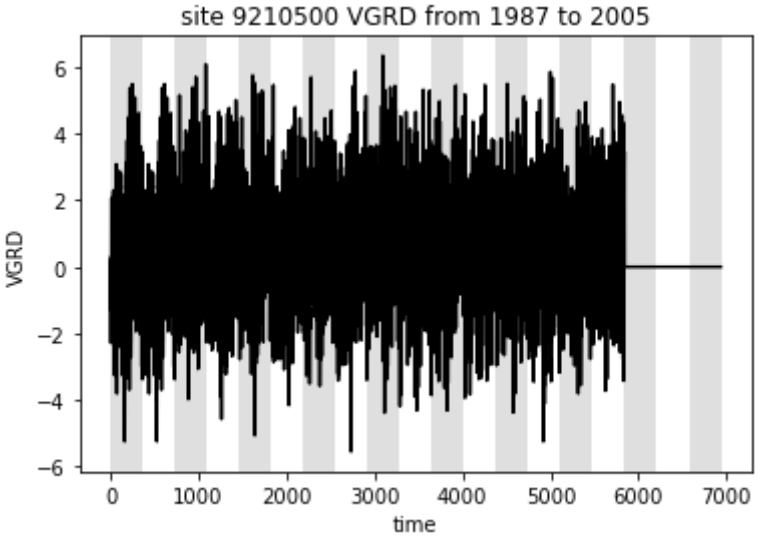

histogram site 9329050 VGRD from 1987 to 2005
2500
2000
1500
1000
500
0
−8
−6
−4
−2
0
2
4
6
8

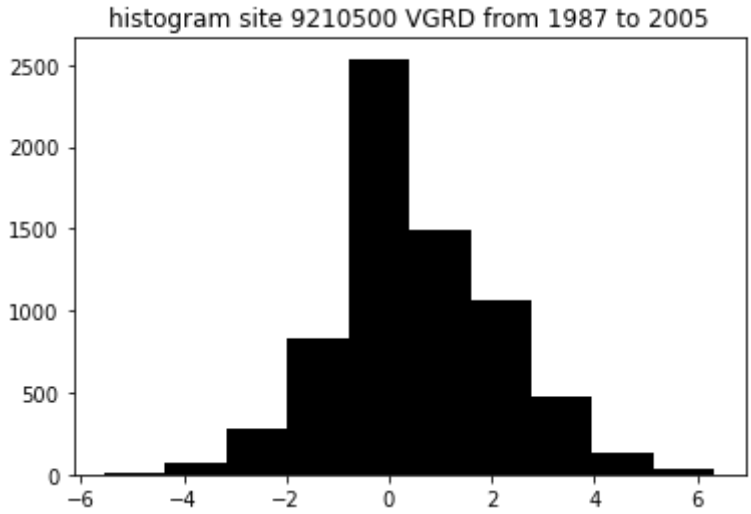

site 9196500 VGRD from 1987 to 2005
VGRD
time
10
5
0
−5
−10
0
1000
2000
3000
4000
5000
6000
7000


histogram site 9196500 VGRD from 1987 to 2005
2500
2000
1500
1000
500
0
−10
−5
0
5
10


site 9107000 VGRD from 1987 to 2005
VGRD
time
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
0
1000
2000
3000
4000
5000
6000
7000







Read in (real) StreamFlow

In [3]:
```
path_st = '../streamflow_in/0'

df_st_l = []
for idx in range(len(huc_name_list)): # through all sites
    site = huc_name_list[idx]
    df_st = pd.read_csv(path_st+str(site)+'.csv')
```

```
    df_st['date'] = pd.to_datetime(df_st['date'])
    df_st.rename(columns={'date': 'timestamp', 'flow': 'Flow_Real'}, inplace=Tru
#     df_st.info()
    df_st_l.append(df_st)
    del df_st, site
```

Read in static variable

In [4]:
```
path_static = '../data_out'

static_col_list = ['clm.drainagearea','clm.out.perm_x','clm.out.porsity','clm.ou

df_station_l = []
for site in huc_name_list: # loop through all sites
    df_static = np.zeros((2,len(static_col_list))) # for each site (indicator (x
    it = 0
    for static in static_col_list:
        out = PFData(path_static+'/'+str(site)+'_'+static+'.pfb')
#         print(path_static+'/'+str(site)+'_'+static+'.pfb')
        out.loadHeader()
        out.loadData()
        out_data = out.getDataAsArray()

        try:
#             print(out_data.shape)

            # set input average forcing
            df_static[0, it] = out_data.mean()
            # set standard dev in axis 1,2 (over time)
            df_static[1, it] = np.std(out_data)
        except:
            pass
        it = it + 1

    df_station_l.append(df_static)
```

Merge together datasets

In [5]:
```
# merge together each dataset on a left join
for idx in range(len(huc_name_list)):
    site = huc_name_list[idx]
#     print(idx)
    df_l[idx] = pd.merge(df_l[idx], df_st_l[idx], how='left', on='timestamp')
    del site
#     it = 0
#     for static in static_col_list:
#         df_l[idx][static] = df_station_l[idx][0,it]
#         df_l[idx][static+'std'] = df_station_l[idx][1,it]
```

Summarize and transform temporal data

In [6]:
```
# add flow
labelist.append('Flow_Real')
# for ea in static_col_list:
#     labelist.append(ea)
#     labelist.append(ea+'std')
```

```
min_label_list = []
max_label_list = []

i = 0
# loop through all forcings
for elem in labelist:
    min_label_list.append(1000000)
    max_label_list.append(-1000000)

    # loop through all stations in this dataset
    for df in df_l:
        # set minimum for element in station dataset
        min_local = df[elem].min()
        max_local = df[elem].max()

        # set global minimum/maximum if it is the minimum/maximum overall
        if min_local < min_label_list[i]:
            min_label_list[i] = min_local
        if max_local > max_label_list[i]:
            max_label_list[i] = max_local

        del min_local, max_local

    # iterate
    i = i+1

# review work:
print(labelist)
print(min_label_list)
print(max_label_list)


# scaled data
df_l_scaled = deepcopy(df_l)

# scale the data and save for later
scale_l = [] # this allows for un-transforming the data
j = 0 # iterate to move through the min max lists
for elem in labelist:
    # create transform and save
    transf = float32_clamp_scaling(src_range=[min_label_list[j],max_label_list[j
    scale_l.append(transf)

    # do transform in place
    for df in df_l_scaled:
        df_2_scale = df[elem].to_numpy()
        df[elem] = transf(df_2_scale)
#         print('success')
    del transf
    j = j + 1
```

```
['DLWR', 'DSWR', 'Press', 'APCP', 'Temp', 'SPFH', 'UGRD', 'VGRD', 'Flow_Real']
[113.56473291607196, 55.378321229241664, 66552.2907803778, 0.0, 247.904075929775
9, 0.0005314474740379672, -7.906057086946189, -6.868398970356629, 2.5]
[327.7075340407556, 328.68986619784863, 70118.24749962668, 0.000383967732870398
2, 290.2426188778539, 0.008497508457841231, 11.852656420342722, 8.66424730403078
4, 2480.0]
```

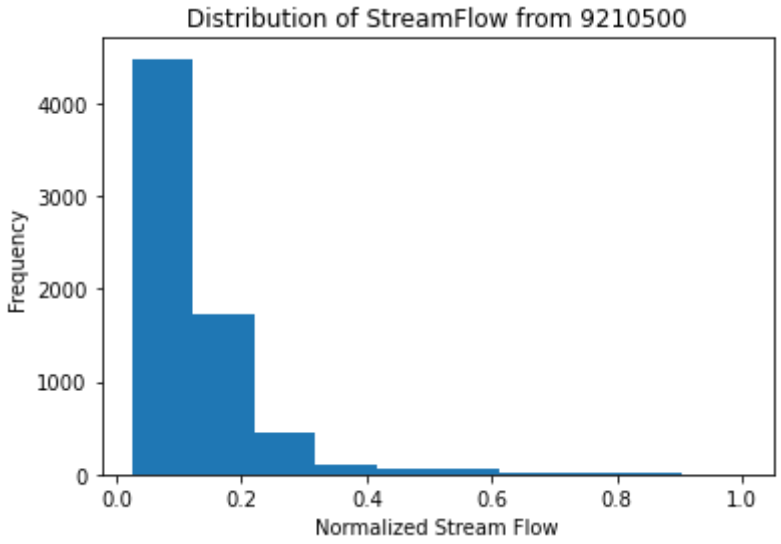
Summary Statistics (normalized from 0 to 1) for each flow site

In [11]:
```
# make an object containing the top 10 and bottom 10% flow for each site
# make an object containing the max and min of streamflow for each huc
# make an object printing out the 50th percentile
# shape -> [10%, 90%, min, max]
perc_10_arr = np.empty((4, len(huc_name_list)))
for ea in range(len(huc_name_list)):
    huc_name = huc_name_list[ea]
    print(huc_name)
    print('min ', df_l_scaled[ea]['Flow_Real'].min())
    print('10% exceedance ', df_l_scaled[ea]['Flow_Real'].quantile([0.1, 0.9]).i
    print('50% percentile ', df_l_scaled[ea]['Flow_Real'].quantile(0.5))
    print('90% exceedance ', df_l_scaled[ea]['Flow_Real'].quantile([0.1, 0.9]).i
    print('max ', df_l_scaled[ea]['Flow_Real'].max(), '\n')
    perc_10_arr[0, ea] = df_l_scaled[ea]['Flow_Real'].quantile([0.1, 0.9]).iloc[
    perc_10_arr[1, ea] = df_l_scaled[ea]['Flow_Real'].quantile([0.1, 0.9]).iloc[
    perc_10_arr[2, ea] = df_l_scaled[ea]['Flow_Real'].min()
    perc_10_arr[3, ea] = df_l_scaled[ea]['Flow_Real'].max()

print(perc_10_arr)
```

```
9110000
min  0.02538849646821393
10% exceedance  0.04419778002018163
50% percentile  0.07689202825428859
90% exceedance  0.225832492431887
max  1.0

9329050
min  0.0
10% exceedance  0.0008879919273461151
50% percentile  0.0020988900100908175
90% exceedance  0.008678102926337033
max  0.08415741675075682

9196500
min  0.0019374369323915236
10% exceedance  0.004238143289606458
50% percentile  0.015136226034308779
90% exceedance  0.2012108980827447
max  0.9233097880928355

9107000
min  0.006659939455095863
10% exceedance  0.011907164480322906
50% percentile  0.019979818365287588
90% exceedance  0.10393541876892029
max  0.4510595358224016

9210500
min  0.0012512613521695256
10% exceedance  0.0062563067608476285
50% percentile  0.011099899091826439
90% exceedance  0.05146316851664985
max  0.28678102926337035

[[4.41977800e-02 8.87991927e-04 4.23814329e-03 1.19071645e-02
  6.25630676e-03]
 [2.25832492e-01 8.67810293e-03 2.01210898e-01 1.03935419e-01
  5.14631685e-02]
 [2.53884965e-02 0.00000000e+00 1.93743693e-03 6.65993946e-03
  1.25126135e-03]
 [1.00000000e+00 8.41574168e-02 9.23309788e-01 4.51059536e-01
  2.86781029e-01]]
```

Histogram of Flow Distribution

In [8]:
```
# investigate the distribution of streamflow
plt.hist(df_l_scaled[0]['Flow_Real'])
plt.xlabel('Normalized Stream Flow')
plt.ylabel('Frequency')
plt.title('Distribution of StreamFlow from '+str(huc_name))
plt.show()
```



Globals

In [50]:
```
# data source

path = '../0507_grande'

save = False

if save:
    try:
        os.mkdir(path)
    except:
        print('warning: file exists')
        pass

huc_index = 4 # for choosing which huc to train on

data = copy(df_l_scaled[huc_index])

# note shape is important because LSTM expects three dimensions
# data = .reshape(-1, 1)

# sliding sequence length (by default, the same as windows set in globals for pr
seq_length = 14 # window
# prediction
fut_length = 7 # when prediction

# batch size
bs = 100

# fraction of data to include in training set (1=all,0=none, remainder in test s
test_frac = 0.2
```

```
# For LSTM
num_epochs = 500 # number of times iterating through the model
learning_rate = 0.001 # rate of learning

input_size = 9 # nodes on the input (should be number of features)
hidden_size = 10 # number of nodes in hidden layer
num_layers = 1 # number of hidden layers

num_classes = 1 # nodes in output (should be 1)


if save:
    print('saving')
    file = open(path+"/params.txt", "w")
    file.write(f'Run Datetime {datetime.now()}\n')
    file.write(f'Sample Date Range {YEARS[0]} to {YEARS[-1]}\n')
    file.write(f'Huc index: {huc_index}\n')
    file.write(f'Huc Name: {huc_name_list[huc_index]}\n')
    file.write(f'Sequence Length {seq_length}\n')
    file.write(f'Prediction Length {fut_length}\n')
    file.write(f'test fraction {test_frac}\n')
    file.write(f'LSTM Params: Number of epochs - {num_epochs}\n')
    file.write(f'LSTM Params: learning rate - {learning_rate}\n')
    file.write(f'LSTM Params: input_size = {input_size}\n')
    file.write(f'LSTM Params: hidden_size = {hidden_size}\n')
    file.write(f'LSTM Params: num_layers = {num_layers}\n')
    file.write(f'LSTM Params: num_classes = {num_classes}\n')
    file.close()
```

Import Data

- This is the first step of the PyTorch procedure
- Sets training data equal to normalized flow data
- Uses the function sliding_windows to define the set of training and test data
- Sets variables as Torch Tensors

In [51]:
```
def delnull(x_in, y_in):
    '''
    This sub for removing columns with null values
    in_x -> input x array from the sliding_windows procecure)
    in_y -> input y array from the sliding windows procedure)
    '''

    # keep track of dimensions
    num_x_axes = len(x_in.shape)
    if num_x_axes > 1:
        axes_tup = []
        for shp in range(1,num_x_axes,1):
            axes_tup.append(shp)
        axes_tup = tuple(axes_tup)
    else:
        axes_tup = 0
#     print(axes_tup)

    # figure out null values
    y_boolean = np.isnan(y_in)
    x_isnan = np.sum(np.isnan(x_in),(1,2))
```

```
    x_boolean = np.zeros(x_isnan.shape)
    x_boolean[x_isnan > 0] = 1
    x_boolean = np.array(x_boolean, dtype=bool)

#     print(x.shape)
#     print(x_boolean.shape)
#     print(y.shape)

    # convert all booleans
    all_boolean = np.zeros(y_boolean.shape)
    all_boolean[x_boolean | y_boolean] = 1
    all_boolean = np.array(all_boolean, dtype=bool)

    # drop y and x values
    x_del = np.delete(x_in, all_boolean, axis=(0))
    y_del = np.delete(y_in, all_boolean, axis=(0))

#     print(x_del.shape)
#     print(y_del.shape)

    return x_del, y_del, all_boolean
```

In [52]:

```
# 2. seperate train and test data, predictors and predicticands
# define sliding windows (note, these are defined in the globals) # x, y = slidi
# note - make sure target (in this case flow) is the last entry


# defaults of labelist ['DLWR', 'DSWR', 'Press', 'APCP', 'Temp', 'SPFH', 'UGRD',


# if we want to reset data to be just 1 dimension, subset the last column = 2
data = data
disc = str(labelist[:-1])+' -> '+str(labelist[-1])

it = 0
if len(labelist) > 1:
    for idx in labelist:
        x_0, y = sliding_windows(data[idx].to_numpy(), seq_length, fut_length)
        if idx == labelist[0]:
            x = np.zeros((x_0.shape[0],x_0.shape[1],len(labelist)))
#             print(x.shape)
#         print(x_0.shape)
        x[:,:,it] = x_0
        it = it+1
else:
    x, y = sliding_windows(data.to_numpy(), seq_length, fut_length)

# print(x[:,0,0])

# # reset x, if desired
# x = x[:,:,:-1]



# drop na values
x, y, t_bool = delnull(x, y)


if len(x.shape) < 3:
    x = np.expand_dims(x, axis=2)
```

```
# divide into test and train size, essentially into thirds
train_size = int(len(y) * test_frac)
test_size = len(y) - train_size

# note shape is important because LSTM expects three dimensions
# https://numpy.org/doc/stable/reference/generated/numpy.atleast_3d.html#numpy.a
# https://numpy.org/doc/stable/reference/generated/numpy.expand_dims.html
# convert to torch tensors
dataX = Variable(torch.Tensor(x))
dataY = Variable(torch.Tensor(np.expand_dims(y, axis=1)))
print(dataX.shape)

# create train vars
trainX = Variable(torch.Tensor(x[0:train_size]))
trainY = Variable(torch.Tensor(np.expand_dims(y[0:train_size], axis=1)))
print(trainX.shape)

# playing with DataLoader and TensorDataset
train_ds = TensorDataset(trainX, trainY)
train_dl = DataLoader(train_ds, batch_size = bs)

# create test vars
testX = Variable(torch.Tensor(x[train_size:len(x)]))
testY = Variable(torch.Tensor(np.expand_dims(y[train_size:len(y)],axis=1)))
print(testX.shape)

if save:
    print('saving')
    file = open(path+"/inputs.txt", "w")
    file.write(f'Number of input features {input_size}\n')
    file.write('Input features are '+disc+'\n')
    file.close()
```

```
torch.Size([6914, 14, 9])
torch.Size([1382, 14, 9])
torch.Size([5532, 14, 9])
```

Train model

- Note the global choices made below

In [29]:

```
# -------
# create model and define criterion and optimizer
# -------
module_load = False
save = True

# note specify load path
if module_load:
    load_PATH = '/home/qh8373/UCRB/0507_grande'
    with open(load_PATH+'/params.txt') as f:
        lines = f.readlines()
        print('model parameter description \n', lines, '\n')

    num_classes = int(lines[-1][-4:-1].replace('=',''))
    num_layers = int(lines[-2][-4:-1].replace('=',''))
    hidden_size = int(lines[-3][-4:-1].replace('=',''))
    input_size = int(lines[-4][-4:-1].replace('=',''))
    seq_length = int(lines[-9][-3:-1].replace('h',''))
```

```
#     print(num_classes, num_layers, hidden_size, input_size, seq_length)
    lstm = LSTM(num_classes, input_size, hidden_size, num_layers, seq_length)
    lstm.load_state_dict(torch.load(load_PATH+'/model.txt'))
else:
    lstm = LSTM(num_classes, input_size, hidden_size, num_layers, seq_length) #

criterion = torch.nn.MSELoss()    # mean-squared error for regression
optimizer = torch.optim.Adam(lstm.parameters(), lr=learning_rate) # Adam as opti

epoch_array = []
loss_array = []

# Train the model
for epoch in range(num_epochs):
    for xb, xy in train_dl:

        # make predictions
        outputs = lstm(xb)

#         print('outputs shape ', outputs.shape)
#         print('target shape ', xy.shape)

#         print(torch.isnan(xy))

        # obtain the loss function
        loss = criterion(outputs, xy)

        # back propogation
        loss.backward()

        # update the weights backward
        optimizer.step()

        # zero gradients
        optimizer.zero_grad()

#     # print results of fitting if epochal condition is met
#     if epoch % 100 == 0:
    print("Epoch: %d, loss: %1.5f" % (epoch, loss.item()))
    epoch_array.append(epoch)
    loss_array.append(loss.item())

print('')
print('the relationship between loss and epochs for given hyper parameters')
plt.plot(epoch_array, loss_array)
plt.yscale('log')
plt.show()
if save:
    plt.savefig(path+'/epochs.png')
    # save epoch file
    file = open(path+"/epochs.txt", "w")
    file.write(str(epoch_array))
    file.close()
    # save loss array file
    file = open(path+"/loss.txt", "w")
    file.write(str(loss_array))
    file.close()
    # save model
    torch.save(lstm.state_dict(), path+'/model.txt')
```

model parameter description

```
 ['Run Datetime 2021-05-07 14:14:41.798293\n', 'Sample Date Range 1987 to 2005
\n', 'Huc index: 0\n', 'Huc Name: 9110000\n', 'Sequence Length 14\n', 'Predictio
n Length 7\n', 'test fraction 0.2\n', 'LSTM Params: Number of epochs - 500\n',
'LSTM Params: learning rate - 0.001\n', 'LSTM Params: input_size = 9\n', 'LSTM P
arams: hidden_size = 10\n', 'LSTM Params: num_layers = 1\n', 'LSTM Params: num_c
lasses = 1\n']

Epoch: 0, loss: 0.00007
Epoch: 1, loss: 0.00005
Epoch: 2, loss: 0.00006
Epoch: 3, loss: 0.00007
Epoch: 4, loss: 0.00007
Epoch: 5, loss: 0.00008
Epoch: 6, loss: 0.00008
Epoch: 7, loss: 0.00008
Epoch: 8, loss: 0.00009
Epoch: 9, loss: 0.00009
Epoch: 10, loss: 0.00009
Epoch: 11, loss: 0.00010
Epoch: 12, loss: 0.00010
Epoch: 13, loss: 0.00010
Epoch: 14, loss: 0.00010
Epoch: 15, loss: 0.00010
Epoch: 16, loss: 0.00010
Epoch: 17, loss: 0.00011
Epoch: 18, loss: 0.00011
Epoch: 19, loss: 0.00011
Epoch: 20, loss: 0.00011
Epoch: 21, loss: 0.00011
Epoch: 22, loss: 0.00011
Epoch: 23, loss: 0.00011
Epoch: 24, loss: 0.00011
Epoch: 25, loss: 0.00011
Epoch: 26, loss: 0.00011
Epoch: 27, loss: 0.00011
Epoch: 28, loss: 0.00011
Epoch: 29, loss: 0.00012
Epoch: 30, loss: 0.00012
Epoch: 31, loss: 0.00012
Epoch: 32, loss: 0.00012
Epoch: 33, loss: 0.00012
Epoch: 34, loss: 0.00012
Epoch: 35, loss: 0.00012
Epoch: 36, loss: 0.00012
Epoch: 37, loss: 0.00012
Epoch: 38, loss: 0.00012
Epoch: 39, loss: 0.00012
Epoch: 40, loss: 0.00012
Epoch: 41, loss: 0.00012
Epoch: 42, loss: 0.00012
Epoch: 43, loss: 0.00012
Epoch: 44, loss: 0.00012
Epoch: 45, loss: 0.00012
Epoch: 46, loss: 0.00012
Epoch: 47, loss: 0.00012
Epoch: 48, loss: 0.00012
Epoch: 49, loss: 0.00012
Epoch: 50, loss: 0.00012
Epoch: 51, loss: 0.00012
Epoch: 52, loss: 0.00012
Epoch: 53, loss: 0.00012
Epoch: 54, loss: 0.00012
Epoch: 55, loss: 0.00012
Epoch: 56, loss: 0.00012
Epoch: 57, loss: 0.00012
```

```
Epoch: 58, loss: 0.00012
Epoch: 59, loss: 0.00012
Epoch: 60, loss: 0.00012
Epoch: 61, loss: 0.00012
Epoch: 62, loss: 0.00012
Epoch: 63, loss: 0.00012
Epoch: 64, loss: 0.00012
Epoch: 65, loss: 0.00012
Epoch: 66, loss: 0.00012
Epoch: 67, loss: 0.00012
Epoch: 68, loss: 0.00012
Epoch: 69, loss: 0.00012
Epoch: 70, loss: 0.00012
Epoch: 71, loss: 0.00012
Epoch: 72, loss: 0.00012
Epoch: 73, loss: 0.00012
Epoch: 74, loss: 0.00012
Epoch: 75, loss: 0.00012
Epoch: 76, loss: 0.00012
Epoch: 77, loss: 0.00012
Epoch: 78, loss: 0.00012
Epoch: 79, loss: 0.00012
Epoch: 80, loss: 0.00012
Epoch: 81, loss: 0.00012
Epoch: 82, loss: 0.00012
Epoch: 83, loss: 0.00012
Epoch: 84, loss: 0.00012
Epoch: 85, loss: 0.00012
Epoch: 86, loss: 0.00012
Epoch: 87, loss: 0.00012
Epoch: 88, loss: 0.00012
Epoch: 89, loss: 0.00012
Epoch: 90, loss: 0.00012
Epoch: 91, loss: 0.00012
Epoch: 92, loss: 0.00012
Epoch: 93, loss: 0.00012
Epoch: 94, loss: 0.00012
Epoch: 95, loss: 0.00012
Epoch: 96, loss: 0.00012
Epoch: 97, loss: 0.00012
Epoch: 98, loss: 0.00012
Epoch: 99, loss: 0.00012
Epoch: 100, loss: 0.00012
Epoch: 101, loss: 0.00012
Epoch: 102, loss: 0.00011
Epoch: 103, loss: 0.00011
Epoch: 104, loss: 0.00011
Epoch: 105, loss: 0.00011
Epoch: 106, loss: 0.00011
Epoch: 107, loss: 0.00011
Epoch: 108, loss: 0.00011
Epoch: 109, loss: 0.00011
Epoch: 110, loss: 0.00011
Epoch: 111, loss: 0.00011
Epoch: 112, loss: 0.00011
Epoch: 113, loss: 0.00011
Epoch: 114, loss: 0.00011
Epoch: 115, loss: 0.00011
Epoch: 116, loss: 0.00011
Epoch: 117, loss: 0.00011
Epoch: 118, loss: 0.00011
Epoch: 119, loss: 0.00011
Epoch: 120, loss: 0.00011
Epoch: 121, loss: 0.00011
Epoch: 122, loss: 0.00011
```

```
Epoch: 123, loss: 0.00011
Epoch: 124, loss: 0.00011
Epoch: 125, loss: 0.00011
Epoch: 126, loss: 0.00011
Epoch: 127, loss: 0.00011
Epoch: 128, loss: 0.00011
Epoch: 129, loss: 0.00011
Epoch: 130, loss: 0.00011
Epoch: 131, loss: 0.00011
Epoch: 132, loss: 0.00011
Epoch: 133, loss: 0.00011
Epoch: 134, loss: 0.00011
Epoch: 135, loss: 0.00011
Epoch: 136, loss: 0.00011
Epoch: 137, loss: 0.00011
Epoch: 138, loss: 0.00011
Epoch: 139, loss: 0.00011
Epoch: 140, loss: 0.00011
Epoch: 141, loss: 0.00011
Epoch: 142, loss: 0.00011
Epoch: 143, loss: 0.00011
Epoch: 144, loss: 0.00011
Epoch: 145, loss: 0.00011
Epoch: 146, loss: 0.00011
Epoch: 147, loss: 0.00011
Epoch: 148, loss: 0.00011
Epoch: 149, loss: 0.00011
Epoch: 150, loss: 0.00011
Epoch: 151, loss: 0.00011
Epoch: 152, loss: 0.00011
Epoch: 153, loss: 0.00011
Epoch: 154, loss: 0.00011
Epoch: 155, loss: 0.00011
Epoch: 156, loss: 0.00011
Epoch: 157, loss: 0.00011
Epoch: 158, loss: 0.00011
Epoch: 159, loss: 0.00011
Epoch: 160, loss: 0.00011
Epoch: 161, loss: 0.00011
Epoch: 162, loss: 0.00011
Epoch: 163, loss: 0.00011
Epoch: 164, loss: 0.00011
Epoch: 165, loss: 0.00011
Epoch: 166, loss: 0.00011
Epoch: 167, loss: 0.00011
Epoch: 168, loss: 0.00011
Epoch: 169, loss: 0.00011
Epoch: 170, loss: 0.00011
Epoch: 171, loss: 0.00011
Epoch: 172, loss: 0.00011
Epoch: 173, loss: 0.00011
Epoch: 174, loss: 0.00011
Epoch: 175, loss: 0.00011
Epoch: 176, loss: 0.00011
Epoch: 177, loss: 0.00011
Epoch: 178, loss: 0.00011
Epoch: 179, loss: 0.00011
Epoch: 180, loss: 0.00011
Epoch: 181, loss: 0.00011
Epoch: 182, loss: 0.00011
Epoch: 183, loss: 0.00011
Epoch: 184, loss: 0.00011
Epoch: 185, loss: 0.00011
Epoch: 186, loss: 0.00011
Epoch: 187, loss: 0.00011
```

```
Epoch: 188, loss: 0.00011
Epoch: 189, loss: 0.00011
Epoch: 190, loss: 0.00011
Epoch: 191, loss: 0.00011
Epoch: 192, loss: 0.00011
Epoch: 193, loss: 0.00011
Epoch: 194, loss: 0.00011
Epoch: 195, loss: 0.00011
Epoch: 196, loss: 0.00011
Epoch: 197, loss: 0.00011
Epoch: 198, loss: 0.00011
Epoch: 199, loss: 0.00011
Epoch: 200, loss: 0.00011
Epoch: 201, loss: 0.00011
Epoch: 202, loss: 0.00011
Epoch: 203, loss: 0.00011
Epoch: 204, loss: 0.00011
Epoch: 205, loss: 0.00011
Epoch: 206, loss: 0.00011
Epoch: 207, loss: 0.00011
Epoch: 208, loss: 0.00011
Epoch: 209, loss: 0.00011
Epoch: 210, loss: 0.00011
Epoch: 211, loss: 0.00011
Epoch: 212, loss: 0.00011
Epoch: 213, loss: 0.00011
Epoch: 214, loss: 0.00011
Epoch: 215, loss: 0.00011
Epoch: 216, loss: 0.00011
Epoch: 217, loss: 0.00011
Epoch: 218, loss: 0.00011
Epoch: 219, loss: 0.00011
Epoch: 220, loss: 0.00011
Epoch: 221, loss: 0.00011
Epoch: 222, loss: 0.00011
Epoch: 223, loss: 0.00011
Epoch: 224, loss: 0.00011
Epoch: 225, loss: 0.00011
Epoch: 226, loss: 0.00011
Epoch: 227, loss: 0.00011
Epoch: 228, loss: 0.00011
Epoch: 229, loss: 0.00011
Epoch: 230, loss: 0.00011
Epoch: 231, loss: 0.00011
Epoch: 232, loss: 0.00011
Epoch: 233, loss: 0.00011
Epoch: 234, loss: 0.00011
Epoch: 235, loss: 0.00011
Epoch: 236, loss: 0.00011
Epoch: 237, loss: 0.00011
Epoch: 238, loss: 0.00011
Epoch: 239, loss: 0.00011
Epoch: 240, loss: 0.00011
Epoch: 241, loss: 0.00011
Epoch: 242, loss: 0.00011
Epoch: 243, loss: 0.00011
Epoch: 244, loss: 0.00011
Epoch: 245, loss: 0.00011
Epoch: 246, loss: 0.00011
Epoch: 247, loss: 0.00011
Epoch: 248, loss: 0.00011
Epoch: 249, loss: 0.00011
Epoch: 250, loss: 0.00011
Epoch: 251, loss: 0.00011
Epoch: 252, loss: 0.00011
```

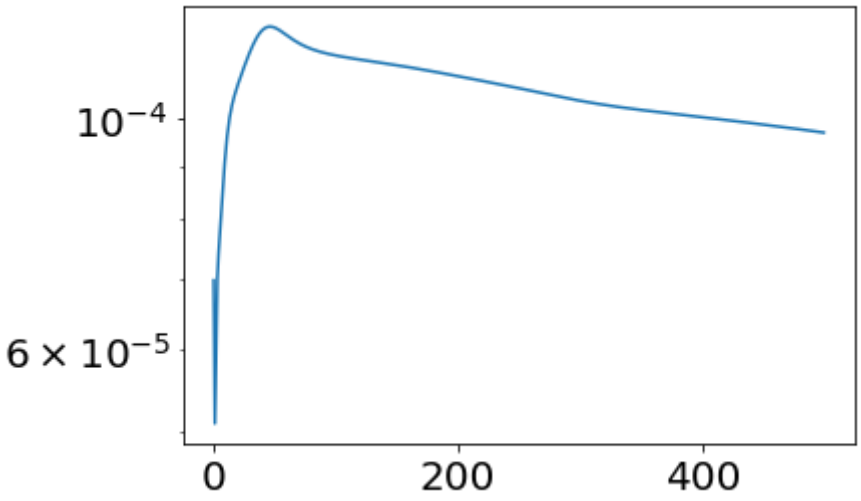
```
Epoch: 253, loss: 0.00011
Epoch: 254, loss: 0.00011
Epoch: 255, loss: 0.00011
Epoch: 256, loss: 0.00011
Epoch: 257, loss: 0.00011
Epoch: 258, loss: 0.00011
Epoch: 259, loss: 0.00011
Epoch: 260, loss: 0.00011
Epoch: 261, loss: 0.00011
Epoch: 262, loss: 0.00011
Epoch: 263, loss: 0.00011
Epoch: 264, loss: 0.00011
Epoch: 265, loss: 0.00011
Epoch: 266, loss: 0.00011
Epoch: 267, loss: 0.00011
Epoch: 268, loss: 0.00011
Epoch: 269, loss: 0.00011
Epoch: 270, loss: 0.00011
Epoch: 271, loss: 0.00011
Epoch: 272, loss: 0.00011
Epoch: 273, loss: 0.00011
Epoch: 274, loss: 0.00011
Epoch: 275, loss: 0.00011
Epoch: 276, loss: 0.00011
Epoch: 277, loss: 0.00011
Epoch: 278, loss: 0.00011
Epoch: 279, loss: 0.00011
Epoch: 280, loss: 0.00011
Epoch: 281, loss: 0.00011
Epoch: 282, loss: 0.00011
Epoch: 283, loss: 0.00011
Epoch: 284, loss: 0.00010
Epoch: 285, loss: 0.00010
Epoch: 286, loss: 0.00010
Epoch: 287, loss: 0.00010
Epoch: 288, loss: 0.00010
Epoch: 289, loss: 0.00010
Epoch: 290, loss: 0.00010
Epoch: 291, loss: 0.00010
Epoch: 292, loss: 0.00010
Epoch: 293, loss: 0.00010
Epoch: 294, loss: 0.00010
Epoch: 295, loss: 0.00010
Epoch: 296, loss: 0.00010
Epoch: 297, loss: 0.00010
Epoch: 298, loss: 0.00010
Epoch: 299, loss: 0.00010
Epoch: 300, loss: 0.00010
Epoch: 301, loss: 0.00010
Epoch: 302, loss: 0.00010
Epoch: 303, loss: 0.00010
Epoch: 304, loss: 0.00010
Epoch: 305, loss: 0.00010
Epoch: 306, loss: 0.00010
Epoch: 307, loss: 0.00010
Epoch: 308, loss: 0.00010
Epoch: 309, loss: 0.00010
Epoch: 310, loss: 0.00010
Epoch: 311, loss: 0.00010
Epoch: 312, loss: 0.00010
Epoch: 313, loss: 0.00010
Epoch: 314, loss: 0.00010
Epoch: 315, loss: 0.00010
Epoch: 316, loss: 0.00010
Epoch: 317, loss: 0.00010
```

```
Epoch: 318, loss: 0.00010
Epoch: 319, loss: 0.00010
Epoch: 320, loss: 0.00010
Epoch: 321, loss: 0.00010
Epoch: 322, loss: 0.00010
Epoch: 323, loss: 0.00010
Epoch: 324, loss: 0.00010
Epoch: 325, loss: 0.00010
Epoch: 326, loss: 0.00010
Epoch: 327, loss: 0.00010
Epoch: 328, loss: 0.00010
Epoch: 329, loss: 0.00010
Epoch: 330, loss: 0.00010
Epoch: 331, loss: 0.00010
Epoch: 332, loss: 0.00010
Epoch: 333, loss: 0.00010
Epoch: 334, loss: 0.00010
Epoch: 335, loss: 0.00010
Epoch: 336, loss: 0.00010
Epoch: 337, loss: 0.00010
Epoch: 338, loss: 0.00010
Epoch: 339, loss: 0.00010
Epoch: 340, loss: 0.00010
Epoch: 341, loss: 0.00010
Epoch: 342, loss: 0.00010
Epoch: 343, loss: 0.00010
Epoch: 344, loss: 0.00010
Epoch: 345, loss: 0.00010
Epoch: 346, loss: 0.00010
Epoch: 347, loss: 0.00010
Epoch: 348, loss: 0.00010
Epoch: 349, loss: 0.00010
Epoch: 350, loss: 0.00010
Epoch: 351, loss: 0.00010
Epoch: 352, loss: 0.00010
Epoch: 353, loss: 0.00010
Epoch: 354, loss: 0.00010
Epoch: 355, loss: 0.00010
Epoch: 356, loss: 0.00010
Epoch: 357, loss: 0.00010
Epoch: 358, loss: 0.00010
Epoch: 359, loss: 0.00010
Epoch: 360, loss: 0.00010
Epoch: 361, loss: 0.00010
Epoch: 362, loss: 0.00010
Epoch: 363, loss: 0.00010
Epoch: 364, loss: 0.00010
Epoch: 365, loss: 0.00010
Epoch: 366, loss: 0.00010
Epoch: 367, loss: 0.00010
Epoch: 368, loss: 0.00010
Epoch: 369, loss: 0.00010
Epoch: 370, loss: 0.00010
Epoch: 371, loss: 0.00010
Epoch: 372, loss: 0.00010
Epoch: 373, loss: 0.00010
Epoch: 374, loss: 0.00010
Epoch: 375, loss: 0.00010
Epoch: 376, loss: 0.00010
Epoch: 377, loss: 0.00010
Epoch: 378, loss: 0.00010
Epoch: 379, loss: 0.00010
Epoch: 380, loss: 0.00010
Epoch: 381, loss: 0.00010
Epoch: 382, loss: 0.00010
```

```
Epoch: 383, loss: 0.00010
Epoch: 384, loss: 0.00010
Epoch: 385, loss: 0.00010
Epoch: 386, loss: 0.00010
Epoch: 387, loss: 0.00010
Epoch: 388, loss: 0.00010
Epoch: 389, loss: 0.00010
Epoch: 390, loss: 0.00010
Epoch: 391, loss: 0.00010
Epoch: 392, loss: 0.00010
Epoch: 393, loss: 0.00010
Epoch: 394, loss: 0.00010
Epoch: 395, loss: 0.00010
Epoch: 396, loss: 0.00010
Epoch: 397, loss: 0.00010
Epoch: 398, loss: 0.00010
Epoch: 399, loss: 0.00010
Epoch: 400, loss: 0.00010
Epoch: 401, loss: 0.00010
Epoch: 402, loss: 0.00010
Epoch: 403, loss: 0.00010
Epoch: 404, loss: 0.00010
Epoch: 405, loss: 0.00010
Epoch: 406, loss: 0.00010
Epoch: 407, loss: 0.00010
Epoch: 408, loss: 0.00010
Epoch: 409, loss: 0.00010
Epoch: 410, loss: 0.00010
Epoch: 411, loss: 0.00010
Epoch: 412, loss: 0.00010
Epoch: 413, loss: 0.00010
Epoch: 414, loss: 0.00010
Epoch: 415, loss: 0.00010
Epoch: 416, loss: 0.00010
Epoch: 417, loss: 0.00010
Epoch: 418, loss: 0.00010
Epoch: 419, loss: 0.00010
Epoch: 420, loss: 0.00010
Epoch: 421, loss: 0.00010
Epoch: 422, loss: 0.00010
Epoch: 423, loss: 0.00010
Epoch: 424, loss: 0.00010
Epoch: 425, loss: 0.00010
Epoch: 426, loss: 0.00010
Epoch: 427, loss: 0.00010
Epoch: 428, loss: 0.00010
Epoch: 429, loss: 0.00010
Epoch: 430, loss: 0.00010
Epoch: 431, loss: 0.00010
Epoch: 432, loss: 0.00010
Epoch: 433, loss: 0.00010
Epoch: 434, loss: 0.00010
Epoch: 435, loss: 0.00010
Epoch: 436, loss: 0.00010
Epoch: 437, loss: 0.00010
Epoch: 438, loss: 0.00010
Epoch: 439, loss: 0.00010
Epoch: 440, loss: 0.00010
Epoch: 441, loss: 0.00010
Epoch: 442, loss: 0.00010
Epoch: 443, loss: 0.00010
Epoch: 444, loss: 0.00010
Epoch: 445, loss: 0.00010
Epoch: 446, loss: 0.00010
Epoch: 447, loss: 0.00010
```

```
Epoch: 448, loss: 0.00010
Epoch: 449, loss: 0.00010
Epoch: 450, loss: 0.00010
Epoch: 451, loss: 0.00010
Epoch: 452, loss: 0.00010
Epoch: 453, loss: 0.00010
Epoch: 454, loss: 0.00010
Epoch: 455, loss: 0.00010
Epoch: 456, loss: 0.00010
Epoch: 457, loss: 0.00010
Epoch: 458, loss: 0.00010
Epoch: 459, loss: 0.00010
Epoch: 460, loss: 0.00010
Epoch: 461, loss: 0.00010
Epoch: 462, loss: 0.00010
Epoch: 463, loss: 0.00010
Epoch: 464, loss: 0.00010
Epoch: 465, loss: 0.00010
Epoch: 466, loss: 0.00010
Epoch: 467, loss: 0.00010
Epoch: 468, loss: 0.00010
Epoch: 469, loss: 0.00010
Epoch: 470, loss: 0.00010
Epoch: 471, loss: 0.00010
Epoch: 472, loss: 0.00010
Epoch: 473, loss: 0.00010
Epoch: 474, loss: 0.00010
Epoch: 475, loss: 0.00010
Epoch: 476, loss: 0.00010
Epoch: 477, loss: 0.00010
Epoch: 478, loss: 0.00010
Epoch: 479, loss: 0.00010
Epoch: 480, loss: 0.00010
Epoch: 481, loss: 0.00010
Epoch: 482, loss: 0.00010
Epoch: 483, loss: 0.00010
Epoch: 484, loss: 0.00010
Epoch: 485, loss: 0.00010
Epoch: 486, loss: 0.00010
Epoch: 487, loss: 0.00010
Epoch: 488, loss: 0.00010
Epoch: 489, loss: 0.00010
Epoch: 490, loss: 0.00010
Epoch: 491, loss: 0.00010
Epoch: 492, loss: 0.00010
Epoch: 493, loss: 0.00010
Epoch: 494, loss: 0.00010
Epoch: 495, loss: 0.00010
Epoch: 496, loss: 0.00010
Epoch: 497, loss: 0.00010
Epoch: 498, loss: 0.00010
Epoch: 499, loss: 0.00010
```

the relationship between loss and epochs for given hyper parameters




```
<Figure size 432x288 with 0 Axes>
```

Score the LSTM, Compare to Reality, and save results

- This portion reflects the score and fit of the autoregression when compared to reality

In [48]:
```
def recon_Y(t_bool,y_in):
    '''
    A sub for reconstructing time series of Y
    '''

#     print(t_bool.shape)
#     print(y_in[0,0])
#     print(y_in.shape)
    y_out = np.empty((t_bool.shape[0],1))
#     print(y_out.shape)

    it = 0
    for idx in range(t_bool.shape[0]):
#         print(idx)
        if t_bool[idx]:
            y_out[idx,0] = np.nan
        else:
            y_out[idx,0] = y_in[it, 0]
            it = it + 1

#     print(y_out)

    return y_out
```

In [53]:
```
module_load = True
save = True

# note specify load path
if module_load:
    load_PATH = '/home/qh8373/UCRB/0507_grande'
    with open(load_PATH+'/params.txt') as f:
        lines = f.readlines()
        print('model parameter description \n', lines, '\n')

    num_classes = int(lines[-1][-4:-1].replace('=',''))
    num_layers = int(lines[-2][-4:-1].replace('=',''))
    hidden_size = int(lines[-3][-4:-1].replace('=',''))
```

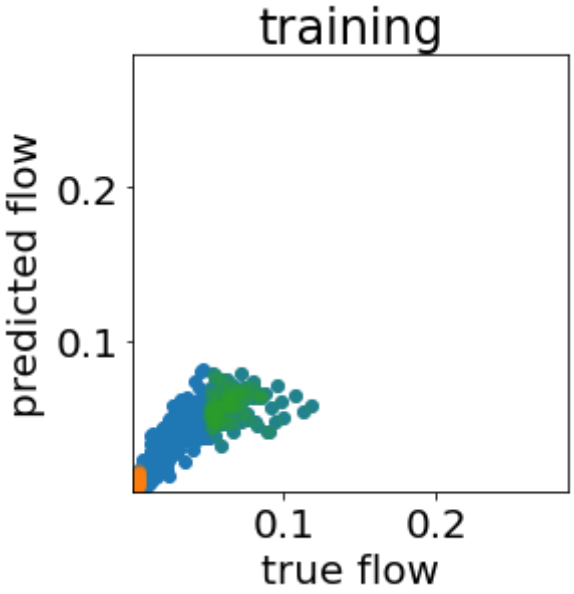
```
    input_size = int(lines[-4][-4:-1].replace('=',''))
    seq_length = int(lines[-9][-3:-1].replace('h',''))
#     print(num_classes, num_layers, hidden_size, input_size, seq_length)
    lstm = LSTM(num_classes, input_size, hidden_size, num_layers, seq_length)
    lstm.load_state_dict(torch.load(load_PATH+'/model.txt'))


# explicitly tell the model we are evaluating it now
lstm.eval()

# make a prediction using the model on all of the data
data_predict = lstm(trainX)

# convert to numpy array
data_predict = data_predict.data.numpy()
dataY_plot = trainY.data.numpy()

# create a mask just in case
data_predict = ma.masked_invalid(data_predict)
dataY_plot = ma.masked_invalid(dataY_plot)

# score the LSTM
r2_train = r2_score(dataY_plot[(dataY_plot.mask==False) & (data_predict.mask==Fa
      data_predict[(dataY_plot.mask==False) & (data_predict.mask==False)])

stats_train = compute_stats(dataY_plot[(dataY_plot.mask==False) & (data_predict.
      data_predict[(dataY_plot.mask==False) & (data_predict.mask==False)])

print('The R-squared value for LSTM is ', r2_train)

print('RMSE      |        NSE       |            KGE','\n', stats_train)


# score the LSTM on low flow

cond1 = (dataY_plot.mask==False) & (data_predict.mask==False) & (dataY_plot.data

r2_train_low = r2_score(dataY_plot[cond1], data_predict[cond1])
stats_train_low = compute_stats(dataY_plot[cond1], data_predict[cond1])

print('The R-squared value for LSTM < 10% flow is ', r2_train_low)

print('RMSE      |        NSE       |            KGE','\n', stats_train_low)


# score the LSTM on high flow

cond2 = (dataY_plot.mask==False) & (data_predict.mask==False) & (dataY_plot.data

r2_train_hi = r2_score(dataY_plot[cond2], data_predict[cond2])
stats_train_hi = compute_stats(dataY_plot[cond2], data_predict[cond2])

print('The R-squared value for LSTM > 90% flow is ', r2_train_hi)

print('RMSE      |        NSE       |            KGE','\n', stats_train_hi)


plt.rcParams.update({'font.size': 20})

# plot
fig, ax = plt.subplots()
```

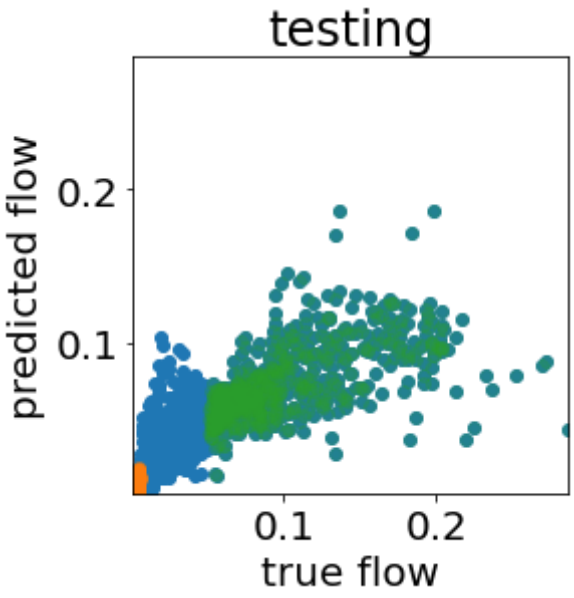
```
plt.scatter(dataY_plot, data_predict)
plt.scatter(dataY_plot[cond1], data_predict[cond1], alpha=0.25)
plt.scatter(dataY_plot[cond2], data_predict[cond2], alpha=0.25)
ax.set_xlim(perc_10_arr[2, huc_index],perc_10_arr[3, huc_index])
ax.set_ylim(perc_10_arr[2, huc_index],perc_10_arr[3, huc_index])
ax.set_xlabel('true flow')
ax.set_ylabel('predicted flow')
# ax.set_xlim(0, 1.0)
# ax.set_ylim(0, 1.0)
ax.set_aspect('equal')
plt.title('training')
fig.show()
if save:
    if module_load:
        path_out = load_PATH+'/'+str(huc_name_list[huc_index])+'_test'
        try:
            os.mkdir(path_out)
        except:
            print('warning: file exists')
            pass
    else:
        path_out = path
    fig.savefig(path_out+'/training.png')
    file = open(path_out+"/score_train.txt", "w")
    file.write(str(r2_train) + '\n' + str(stats_train) + str(r2_train_low) + '\n
    file.close()


# # make a prediction using the model on all of the data
data_predict_test = lstm(testX)

# convert to numpy array
data_predict_test = data_predict_test.data.numpy()
dataY_plot_test = testY.data.numpy()

# create a mask just in case
data_predict_test = ma.masked_invalid(data_predict_test)
dataY_plot_test = ma.masked_invalid(dataY_plot_test)

# score the LSTM
r2_test = r2_score(dataY_plot_test[(dataY_plot_test.mask==False) & (data_predict
               data_predict_test[(dataY_plot_test.mask==False) & (data_predict_t

stats_test = compute_stats(dataY_plot_test[(dataY_plot_test.mask==False) & (data
        data_predict_test[(dataY_plot_test.mask==False) & (data_predict_test.mas

print('The R-squared value for LSTM is ', r2_test)

print('RMSE      |        NSE       |            KGE','\n', stats_test)


# score the LSTM on low flow
cond1 = (dataY_plot_test.mask==False) & (data_predict_test.mask==False) & (dataY

r2_test_low = r2_score(dataY_plot_test[cond1], data_predict_test[cond1])
stats_test_low = compute_stats(dataY_plot_test[cond1], data_predict_test[cond1])

print('The R-squared value for LSTM < 10% flow is ', r2_test_low)

print('RMSE      |        NSE       |            KGE','\n', stats_test_low)
```

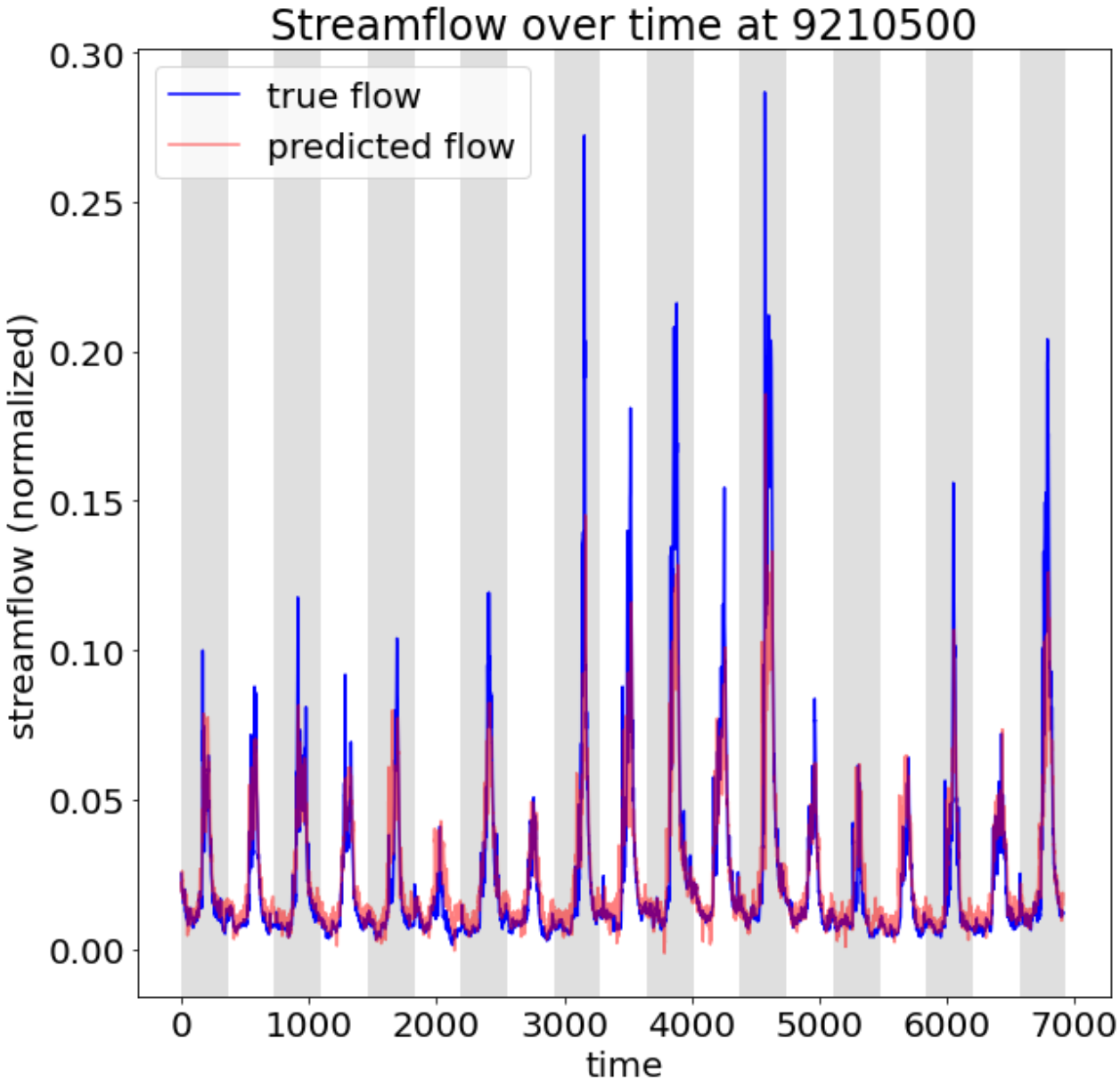
```
# score the LSTM on high flow
cond2 = (dataY_plot_test.mask==False) & (data_predict_test.mask==False) & (dataY

r2_test_hi = r2_score(dataY_plot_test[cond2],data_predict_test[cond2])
stats_test_hi = compute_stats(dataY_plot_test[cond2],data_predict_test[cond2])

print('The R-squared value for LSTM > 90% flow is ', r2_test_hi)

print('RMSE      |        NSE       |           KGE','\n', stats_test_hi)

# plot
fig, ax = plt.subplots()
plt.scatter(dataY_plot_test, data_predict_test)
plt.scatter(dataY_plot_test[cond1], data_predict_test[cond1], alpha=0.25)
plt.scatter(dataY_plot_test[cond2], data_predict_test[cond2], alpha=0.25)
ax.set_xlim(perc_10_arr[2, huc_index],perc_10_arr[3, huc_index])
ax.set_ylim(perc_10_arr[2, huc_index],perc_10_arr[3, huc_index])
ax.set_xlabel('true flow')
ax.set_ylabel('predicted flow')
# ax.set_xlim(0, 1.0)
# ax.set_ylim(0, 1.0)
ax.set_aspect('equal')
plt.title('testing')
fig.show()
if save:
    if module_load:
        path_out = load_PATH+'/'+str(huc_name_list[huc_index])+'_test'
    else:
        path_out = path
    fig.savefig(path_out+'/testing.png')
    file = open(path_out+"/score_test.txt", "w")
    file.write(str(r2_test) + ' ' + str(stats_test) + '\n' + str(r2_test_low) +
    file.close()

# data for plotting
dataY_plot = dataY.data.numpy()
print(dataY_plot.shape)
dataY_plot = recon_Y(t_bool,dataY_plot)
dataY_plot = ma.masked_invalid(dataY_plot)

predY_plot = lstm(dataX).data.numpy()
predY_plot = recon_Y(t_bool,predY_plot)
predY_plot = ma.masked_invalid(predY_plot)

# defaults of labelist ['DLWR', 'DSWR', 'Press', 'APCP', 'Temp', 'SPFH', 'UGRD',

label_dex = 4
inputX_1 = np.expand_dims(np.mean(dataX.data.numpy()[:,:,label_dex], axis = 1) ,
print(inputX_1.shape)
inputX_1 = recon_Y(t_bool,inputX_1)
inputX_1 = ma.masked_invalid(inputX_1)


# general plots comparing predicted streamflow over prediction interval
fig, ax = plt.subplots(figsize=(10,10))
ax.plot(range(dataY_plot.shape[0]), dataY_plot, color='blue', label='true flow')
# ax.plot(range(data_predict.shape[0],data_predict.shape[0]+data_predict_test.sh
# ax.plot(range(data_predict.shape[0]), data_predict, color='red', label='train
ax.plot(range(predY_plot.shape[0]), predY_plot, color='red', label='predicted fl
# ax.plot(range(inputX_1.shape[0]), inputX_1, color='green', label=labelist[labe
ax.set_xlabel('time')
```

```
ax.set_ylabel('streamflow (normalized)')
ax.set_title('Streamflow over time at '+str(huc_name_list[huc_index]))
plt.legend(loc=0)
for i in range(0,len(YEARS),2):
    day = i*365
    plt.axvspan(day,day+365, facecolor='grey', alpha=0.25)
fig.show()
if save:
    if module_load:
        path_out = load_PATH+'/'+str(huc_name_list[huc_index])+'_test'
    else:
        path_out = path
    fig.savefig(path_out+'/flow_comp.png')
```

```
model parameter description
 ['Run Datetime 2021-05-07 14:14:41.798293\n', 'Sample Date Range 1987 to 2005
\n', 'Huc index: 0\n', 'Huc Name: 9110000\n', 'Sequence Length 14\n', 'Predictio
n Length 7\n', 'test fraction 0.2\n', 'LSTM Params: Number of epochs - 500\n',
'LSTM Params: learning rate - 0.001\n', 'LSTM Params: input_size = 9\n', 'LSTM P
arams: hidden_size = 10\n', 'LSTM Params: num_layers = 1\n', 'LSTM Params: num_c
lasses = 1\n']

The R-squared value for LSTM is  0.8145465052194059
RMSE      |        NSE       |            KGE
 [0.10271243 0.8145465  0.81494547]
The R-squared value for LSTM < 10% flow is  -32.80976770005311
RMSE      |        NSE       |            KGE
 [  0.03757775 -32.80976105  -1.1199542 ]
The R-squared value for LSTM > 90% flow is  -0.7409377982907885
RMSE      |        NSE       |            KGE
 [ 0.09780248 -0.74093747  0.04591805]
The R-squared value for LSTM is  0.7125183426929422
RMSE      |        NSE       |            KGE
 [0.05040575 0.71251833 0.65293844]
The R-squared value for LSTM < 10% flow is  -35.186758671333884
RMSE      |        NSE       |            KGE
 [  0.10270151 -35.18675995  -1.65945959]
The R-squared value for LSTM > 90% flow is  -0.13315421365821467
RMSE      |        NSE       |            KGE
 [ 0.74512818 -0.13315427  0.31367454]
(6914, 1)
(6914, 1)
```







Generate Statistics to Evaluate Model Transferability and

Performance

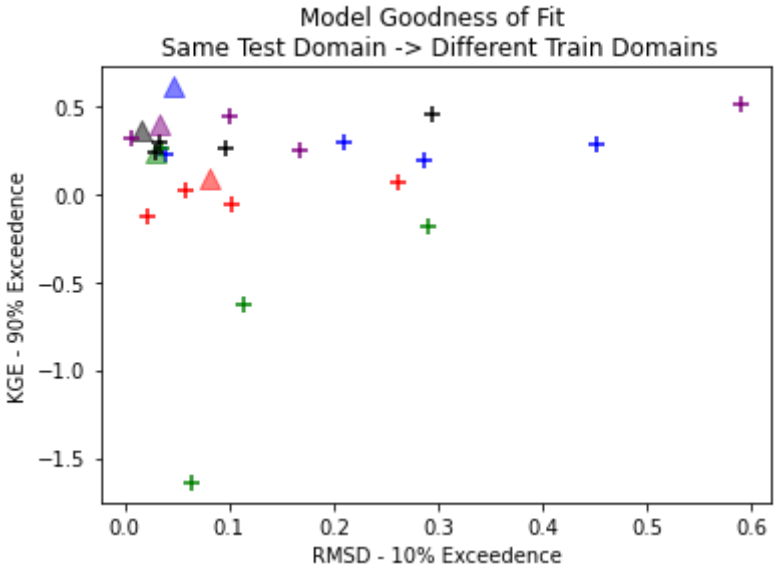
In [47]:
```
# Load Data
MP_simp = pd.read_csv('Mainplot_simple_statistics.csv')
print(MP_simp)

# set upconditions
cond_10 = (MP_simp['Test Statistics'] == '10% Exceedence')
cond_90 = (MP_simp['Test Statistics'] == '90% Exceedence')
cond_1 = (MP_simp['Approach'] == 1)
cond_2 = (MP_simp['Approach'] == 2)

collist = ['blue','green','red', 'black', 'purple'] #  'purple', 'indigo','black
marker_list = ['^', '+']

print(huc_name_list)

# set up colors
i = 0
for huc in huc_name_list:
    # plot approach 1
    cond_huc = (MP_simp['Test domain'] == huc)
    plt.scatter(MP_simp['RMSD'][cond_10 & cond_1 & cond_huc], MP_simp['KGE'][con
    plt.scatter(MP_simp['RMSD'][cond_10 & cond_2 & cond_huc], MP_simp['KGE'][con
    #     plt.scatter(MP_simp['RMSD'](cond_10 & cond_1 & (MP_simp['Test domain']
#     plt.scatter(MP_simp['KGE'](cond_90 & cond_2 & (MP_simp['Test domain'] == h
    i = i + 1

plt.title('Model Goodness of Fit \n Same Test Domain -> Different Train Domains'
plt.xlabel('RMSD - 10% Exceedence')
plt.ylabel('KGE - 90% Exceedence')
plt.show()

collist = ['blue','green','red', 'black', 'purple'] #  'purple', 'indigo','black
marker_list = ['^', '_']
# set up colors
i = 0
for huc in huc_name_list:
    # plot approach 1
    cond_huc = (MP_simp['Train domain'] == huc)
    plt.scatter(MP_simp['RMSD'][cond_10 & cond_1 & cond_huc], MP_simp['KGE'][con
    plt.scatter(MP_simp['RMSD'][cond_10 & cond_2 & cond_huc], MP_simp['KGE'][con
    #     plt.scatter(MP_simp['RMSD'](cond_10 & cond_1 & (MP_simp['Test domain']
#     plt.scatter(MP_simp['KGE'](cond_90 & cond_2 & (MP_simp['Test domain'] == h
    i = i + 1

plt.title('Model Goodness of Fit \n Same Train Domain -> Different Test Domains'
plt.xlabel('RMSD - 10% Exceedence')
plt.ylabel('KGE - 90% Exceedence')
plt.show()
```

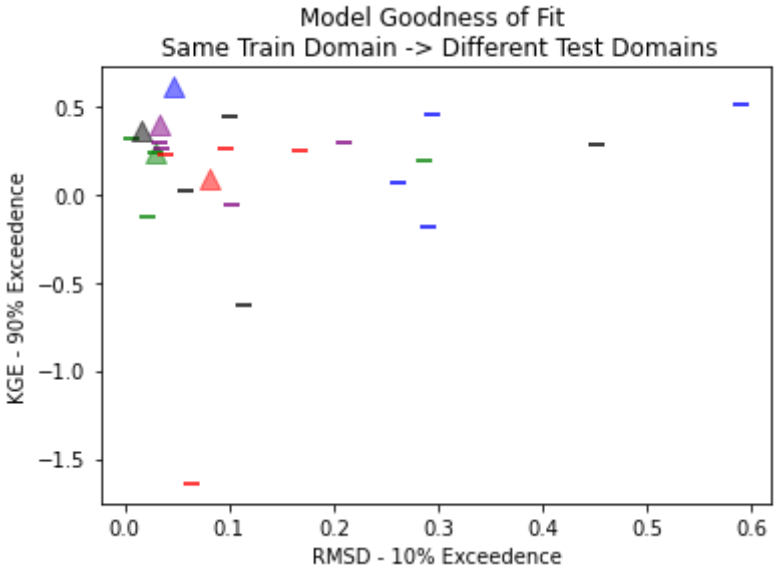
```
    Approach  ID  Train domain  Test domain Model Name Test Statistics  \
0          1   1       9110000      9110000     0506_b  All Streamflow
1          1   1       9110000      9110000     0506_b  10% Exceedence
2          1   1       9110000      9110000     0506_b  90% Exceedence
3          1   2       9329050      9329050     0506_e  All Streamflow
4          1   2       9329050      9329050     0506_e  10% Exceedence
..       ...  ..           ...          ...        ...             ...
```

```
70          2   24       9196500       9210500      0506_f   10% Exceedence
71          2   24       9196500       9210500      0506_f   90% Exceedence
72          2   25       9107000       9210500      0506_g   All Streamflow
73          2   25       9107000       9210500      0506_g   10% Exceedence
74          2   25       9107000       9210500      0506_g   90% Exceedence
```

	R_2	RMSD	NSE	KGE	Unnamed: 10	Unnamed: 11
0	0.8385	0.9466	0.8385	0.7466	NaN	NaN
1	-2.9883	0.0466	-2.9883	-0.3166	NaN	NaN
2	0.3574	1.8808	0.3574	0.6073	NaN	NaN
3	0.4724	0.0874	0.4724	0.4933	NaN	NaN
4	-142.0975	0.0296	-142.0975	-6.8059	NaN	NaN
..	...	...	...	...	...	...
70	-207.5994	0.1671	-207.5994	-10.2730	NaN	NaN
71	-1.4072	1.0321	-1.4072	0.2513	NaN	NaN
72	0.6561	0.1841	0.6561	0.8014	NaN	NaN
73	-58.5752	0.0996	-58.5752	-4.0078	NaN	NaN
74	-0.1364	0.0268	-0.1364	0.4342	NaN	NaN

```
[75 rows x 12 columns]
[9110000, 9329050, 9196500, 9107000, 9210500]
```





lstm_models.py

```python
# for DL
import numpy as np
import torch
import torch.nn as nn

from torch.utils.data import TensorDataset # for refactoring x and y
from torch.utils.data import DataLoader # for batch submission

from torch.autograd import Variable
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
```

```python
def sliding_windows(data, seq_length, future=1):
    """This takes in a dataset and sequence length, and uses this
    to dynamically create (multiple) sliding windows of data to fit
    """
    x = []
    y = []

    for i in range(len(data)-seq_length-future):
        tx = data[i:(i+seq_length)]
        ty = data[i+seq_length+future-1]
        x.append(tx)
        y.append(ty)

    return np.array(x),np.array(y)
```

```
class LSTM(nn.Module):
    # input_size = the number of features in the input

    # num_classes = dimensions of the export layer

    # hidden_size = dimensions of the hidden layer

    # num_layers = number of layers to use
    def __init__(self, num_classes, input_size, hidden_size, num_layers, seq_length):
        # Super
        super(LSTM, self).__init__()

        # attributes
        self.num_classes = num_classes
        self.num_layers = num_layers
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.seq_length = seq_length

        # the LSTM takes inputs and exports hidden states
        # note we are only passing in dimensions here
        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
                            num_layers=num_layers, batch_first=True)

        # the LSTM takes hidden states space to tag space
        self.fc = nn.Linear(hidden_size, num_classes)

      # this moves weights forward through network
      # I'm not sure how this works!!
    def forward(self, x):
        h_0 = Variable(torch.zeros(
            self.num_layers, x.size(0), self.hidden_size))

        c_0 = Variable(torch.zeros(
            self.num_layers, x.size(0), self.hidden_size))

        # Propagate input through LSTM
        ula, (h_out, _) = self.lstm(x, (h_0, c_0))

        h_out = h_out.view(-1, self.hidden_size)

        out = self.fc(h_out)

        return out
```