A MACHINE LEARNING APPROACH TO STREAMFLOW ESTIMATION

AND FORECASTING FOR HYSTERETIC STREAMS

**Abstract**

Streamflow hysteresis is a phenomenon that occurs during flood events or due to seasonal vegetation growth in which the stage-discharge relationship is different between the rising and falling limbs of a hydrograph. This phenomenon poses serious challenges, as the current practice for estimating streamflow is based on linear rating curve methods. Due to the characteristic loop of the stage-discharge relationship during the streamflow hysteresis cycle as seen in Figure 1, flows during the rising limb of the hydrograph are underestimated and flows during the falling limb are overestimated by a linear approximation. Machine learning time series analyses may be applied using directly measured streamflow velocity, water level, and water surface slope to characterize streamflow variables in a hysteretic stream and use these relationships in flood forecasting. In this study, a Long Short-Term Memory (LSTM) model is developed for streamflow estimation and forecasting. Optimal input variables, hyperparameters, time lags, and record lengths are determined by evaluating a series of model setup trials. The study uses numerical model-simulated data from a USACE HEC-RAS model of the Illinois River from USGS. One outcome of this study is recommendations to include water level, velocity, and water surface slope input features for estimating and forecasting streamflow. A second recommendation for monitoring and hydroinformatics applications is that with a 1-hour to 1-day lag and approximately 2 years of training data, streamflow may be accurately forecasted in a hysteretic stream with sufficient warning time for a hypothetical operational system. Understanding the temporal relationships of flow variables in cyclical processes will help to close the gap between knowledge of open-channel cyclical processes and the current approaches for collecting continuous streamflow data to support model and data-driven predictions.

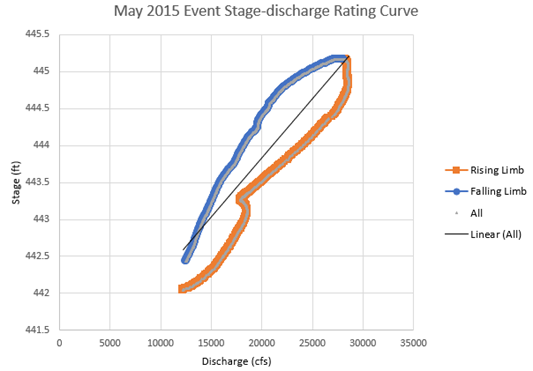
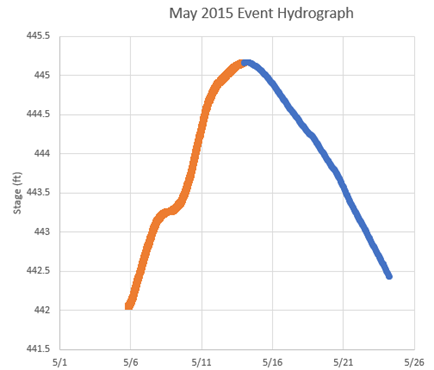


Figure 1: An example of a model-simulated a) hydrograph and b) stage-discharge rating curve, revealing the hysteretic loop compared to a linear relationship commonly used for discharge estimations.

**Introduction**

The problem at the focus of this analysis is flood peak prediction in hysteretic streams based on model-generated time series of flow variables, which represents directly measured data. Properly considering hysteresis in streamflow monitoring and forecasting is an important problem because hysteresis effects have up to a 40% measurement error with the current streamflow measurement techniques (Muste et al., 2020). Additionally, 67% of streams monitored by USGS are affected by hysteresis (Holmes, 2016). More accurate monitoring and predictions provide the opportunity for better warning systems, reducing losses of property and life.

The goal of this analysis is to parameterize relationships between flow variables during the cyclical process using historical data to aid in the development of more accurate streamflow forecasting models. As seen in Figure 1, during a hysteretic event, there is evidence of a sequential phenomenon in which the water free-surface slope peaks before the index velocity, which peaks before the streamflow, which peaks before the water level.

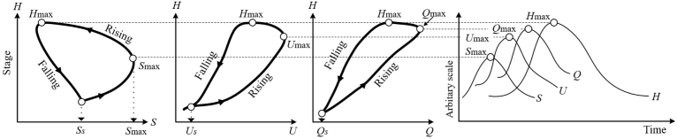


Figure 2: Hysteresis effects on flow variables during cyclical processes: a) stage vs. Water free-surface slope; b) stage vs. index velocity; c) stage vs. Discharge; d) sequencing of the hydrograph variables (Muste et al., 2022).

By utilizing time series analysis models such as LSTM, dynamic streamflow variables may be used to predict the streamflow peak time and magnitude. This could be extremely useful in practice if monitoring equipment which is capable of continuously measuring all these variables can be established at a monitoring station. By applying a more inductive, machine learning, approach to modeling hysteretic streams, we aim to create a model that is significantly more robust at predicting these variables than the current methods. Thus, a streamflow forecast based on these variables will capture more of the dynamics and be much more physically comprehensive than that of the current protocol, estimations based on stage-discharge rating curves. This approach is novel because most flood forecasts do not accurately account for hysteresis, and furthermore, the use of machine learning techniques can provide information on the relationships between these variables to guide future monitoring and forecasting protocols.

The result of these experiments is the characterization of flow variable relationships for the use in streamflow forecasting, including an analysis on feature importance. Although the data used for this analysis is model generated, the design of the tested variables can be replicated in an *in-situ* monitoring system. We provide a framework and general recommendations as a starting point, but more work can be done on improving this approach to be more accurate by further tuning the model to each specific location or use case, and robust with the inclusion of additional streams of variable vectors for training.

**Background & Proposed Methods**

The preceding work that is related to this subject generally does not consider hysteresis in streamflow estimation and forecasting. Rather, the effects of hysteresis on streamflow are considered epistemic uncertainties. A significant number of recent papers indicate a growing interest in using machine learning and deep learning approaches to hydrologic predictions (Basu et al., 2022; Zhang et al., 2021; Atashi et al., 2022; Muste et al., 2022; Xiang et al., 2020; Arab et al., 2021). The comprehensive review, Sit et al (2020), specifically indicated ~60 papers utilizing an LSTM framework for hydrologic research.

The machine learning model that we have selected as the most applicable for our data is the LSTM (Long short-term memory). LSTMs (Long Short-Term Memory), as the name implies, increase the short-term memory over the input that the model uses for predictions (Sit et al., 2020; Sher, 2022). The LSTM model has been proven in many fields and situations to generate predictions effectively and robustly for time-series and time-series-like data (Gawehn et al., 2016; Malhotra et al, 2015; Sutskever & Vinyals, 2014; Kratzert et al., 2018). Instead of generating a single output from each neuron and passing it along to the next, as is the way of a “vanilla” or basic RNN (recurrent neural networks), the LSTM improves upon the base by carrying a secondary matrix of weights. This secondary matrix is responsible for assigning importance or weight for each value in the input matrix (Hochreiter & Schmidhuber, 1997). This importance matrix is what gives the LSTM its ability to discern longer term relationships in time series data. Over time, the LSTM network determines which values are more important given x input value. This is a key improvement over the vanilla RNNs of the past.

Past the normal LSTMs, we aim to explore other models and model configurations to better understand and capture the complex relationships in hysteretic streams. We plan to explore the use of GRUs (gated recurrent units) and more complex LSTMs: attention-based LSTMs, stacked LSTMs, and conditional LSTMs. Each provides their own benefit which we will explore to determine if the additional complexities are justified by the hypothetical improvements.

One challenge to this approach is the currently deployed streamflow monitoring instrumentation. The USGS currently monitors water levels and discharge at many monitoring stations, but characterizing hysteretic streams requires the measurement of index velocity, water surface slope, as well as other variables, as this analysis and several others indicate (Cheng et al., 2019; Muste et al., 2022; Thaisiam et al., 2022). Current technology can and does achieve this but has not been implemented at a large enough scale to transform the monitoring and streamflow estimation protocols.

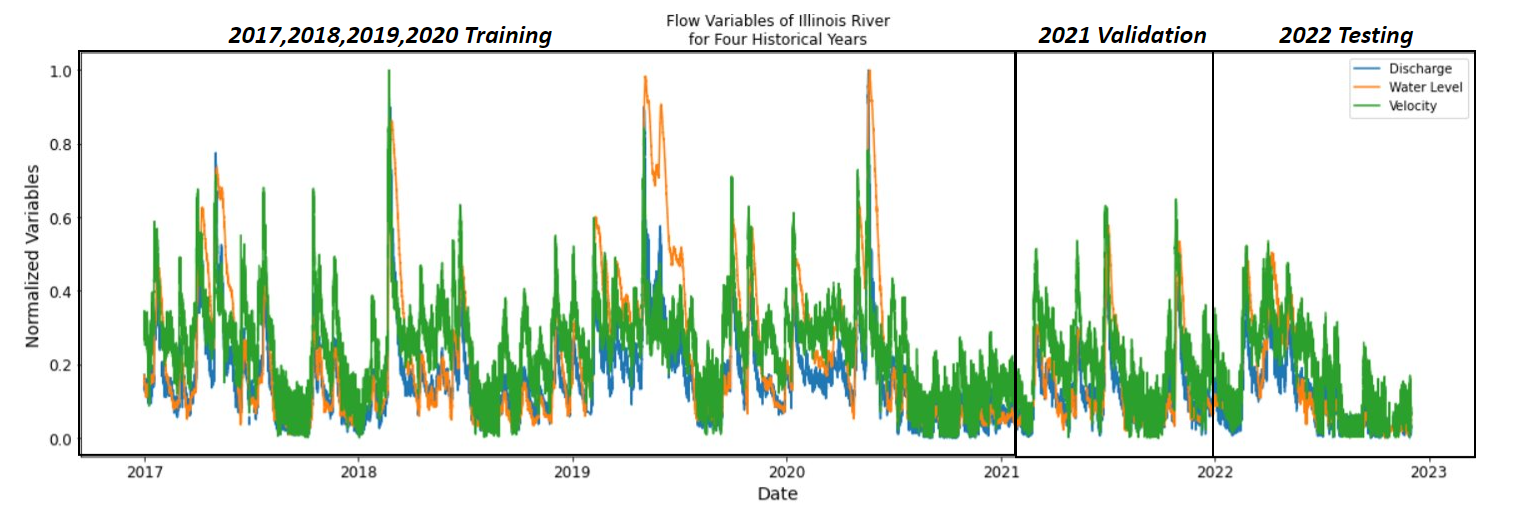
Current approaches to estimating streamflow are reliant on the rating curve method. These are empirical rating curves developed for each stream based on historical stage and discharge data, but these are linear and fail to capture the hysteretic behavior that is characteristic of most streams (Cheng et al., 2019). Remarkedly, there is no method that is currently used independently to forecast streamflow. Index velocity, water surface slope, and other parameters are not commonly used for estimating and forecasting streamflow, and current models rely on rainfall-runoff relationships which are often empirical (Thaisiam et al., 2022). The inclusion of the additional variables of index velocity and water surface slope in an improved streamflow estimation and forecasting machine learning model is the scientific contribution of this study.

The methods are continued in code in the attached appendix.

**Preliminary Figures and Frameworks**

Table 1: A sample of the data used for training including some variables.



  
Figure 3: Normalized streamflow variables for the USGS station in the Illinois River near Henry, IL for the years 2017-2020, used for training, validation, and testing of the machine learning models.

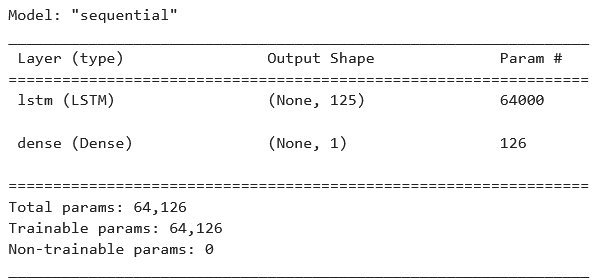


Figure 4: A preliminary, minimally-tuned LSTM used for proof-of-concept modeling.

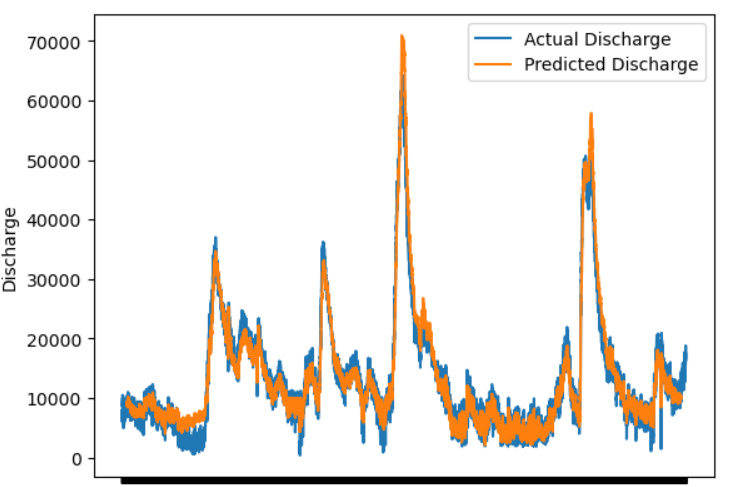


Figure 5: A prediction over a training set of the Henry, IL monitoring station on the Illinois River using the LSTM outlined in figure 3.

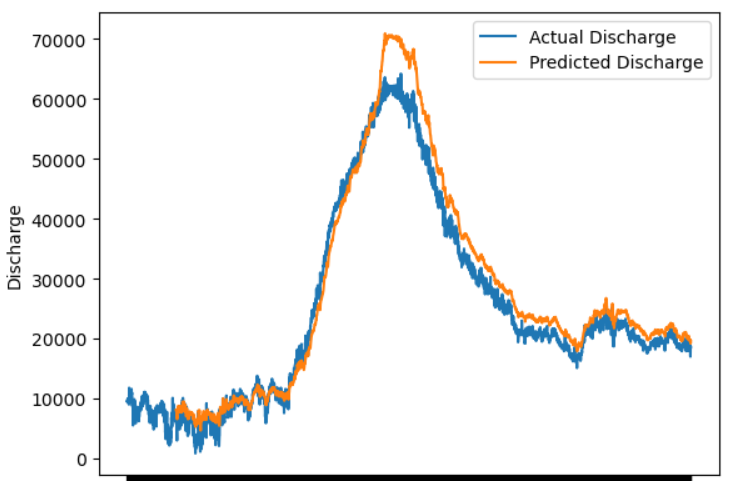


Figure 6: A close-up prediction of a specific event produced by the same LSTM as in figure 3.

**Conclusion**

Finally, this project seeks to illuminate and quantify the role hysteresis plays in effective river forecasting and management. We aim to accomplish this through the utilization of machine learning models with specific focus on LSTMs of a few varieties. Through extensive background research and preliminary testing, we found that this model performs best and, conceptually and theoretically, is the most rigorously reviewed in this field. It is our hope that the results of this project will provide a better understanding and forecasting capabilities for risk management engineers and policy makers.

**References**

Arab, S. T., Noguchi, R., Matsushita, S., & Ahamed, T. (2021). Prediction of grape yields from time-series vegetation indices using satellite remote sensing and a machine-learning approach. Remote Sensing Applications: Society and Environment, 22, 100485.

Atashi, V., Gorji, H. T., Shahabi, S. M., Kardan, R., & Lim, Y. H. (2022). Water Level Forecasting Using Deep Learning Time-Series Analysis: A Case Study of Red River of the North. Water, 14(12), 1971.

Basu, B., Morrissey, P., & Gill, L. W. (2022). Application of nonlinear time series and machine learning algorithms for forecasting groundwater flooding in a lowland karst area. Water Resources Research, 58(2), e2021WR029576.

Cheng Z, Lee K, Kim D, Muste M, Vidmar P, Hulme J (2019) Experimental evidence on the performance of rating curves for continuous discharge estimation in complex flow situations. Journal of Hydrology, 568:959–971. <https://doi.org/10.1016/j.jhydrol.2018.11.021>

Gawehn E, Hiss JA, Schneider G. Deep Learning in Drug Discovery. Mol Inform. 2016 Jan;35(1):3-14. doi: 10.1002/minf.201501008. Epub 2015 Dec 30. PMID: 27491648.

Kratzert, F., Klotz, D., Brenner, C., Schulz, K., & Herrnegger, M. (2018). Rainfall–runoff modelling using long short-term memory (LSTM) networks. *Hydrology and Earth System Sciences*, *22*(11), 6005–6022. <https://doi.org/10.5194/hess-22-6005-2018>

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

Holmes, R. R. (2016). River rating complexity. In Proceedings of the International Conference on Fluvial Hydraulics (River Flow 2016), St. Louis, USA.

Malhotra, P., Vig, L., Shroff, G., & Agarwal, P. (2015). Long short term memory networks for anomaly detection in time series. Proceedings. Presses universitaires de Louvain, 89.

Muste, M., Lee, K., Kim, D., Bacotiu, C., Oliveros, M. R., Cheng, Z., & Quintero, F. (2020). Revisiting hysteresis of flow variables in monitoring unsteady streamflows. Journal of hydraulic research, 58(6), 867-887.

Muste, M., Kim, D., & Kim, K. (2022). A flood-crest forecast prototype for river floods using only in-stream measurements. Communications Earth & Environment, 3(1), 1-10.

Sher, D. V. (2022, October 1). *Time series modeling using Scikit, Pandas, and Numpy*. Medium. Retrieved December 15, 2022, from <https://towardsdatascience.com/time-series-modeling-using-scikit-pandas-and-numpy-682e3b8db8d1>

Sit, M., Demiray, B. Z., Xiang, Z., Ewing, G. J., Sermet, Y., & Demir, I. (2020). A comprehensive review of deep learning applications in hydrology and water resources. Water Science and Technology, 82(12), 2635-2670.

Sklearn.neural\_network.Mlpregressor. (n.d.). Retrieved December 16, 2022, from <https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html>

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Advances in neural information processing systems (pp. 3104-3112).

Thaisiam, W., Saelo, W., & Wongchaisuwat, P. (2022). Enhancing a Multi-Step Discharge Prediction with Deep Learning and a Response Time Parameter. Water, 14(18), 2898.

USGS 05558300 Illinois River at Henry, IL. (n.d.). Retrieved December 14, 2022, from <https://waterdata.usgs.gov/nwis/uv?site_no=05558300>

USGS-python. (n.d.). USGS-python/dataretrieval: Tools for downloading hydrologic and climate data. Retrieved December 16, 2022, from <https://github.com/USGS-python/dataretrieval>

Xiang, Z., Yan, J., & Demir, I. (2020). A rainfall‐runoff model with LSTM‐based sequence‐to‐sequence learning. Water resources research, 56(1), e2019WR025326.

Zhang, Y., Ren, J., Wang, R., Fang, F., & Zheng, W. (2021). Multi-step sequence flood forecasting based on MSBP model. Water, 13(15), 2095.

**Appendix**

LSTM Preliminary Methods

# Imports

In [1]:

**import** numpy **as** np

**import** tensorflow **as** tf

**from** keras.models **import** Sequential

**from** keras.layers **import** LSTM

**from** keras.layers **import** Dense, Dropout

**import** pandas **as** pd

**from** matplotlib **import** pyplot **as** plt

**from** sklearn.preprocessing **import** StandardScaler

**import** seaborn **as** sns

# Read and EDA

In [6]:

*#Read the csv file*

df **=** pd**.**read\_csv('henry\_csv\_17-23.csv')*#, infer\_datetime\_format= True)* renames **=** {'00065': 'Gage Height', '00060': 'Discharge', '72254': 'Velocity'} df **=** df**.**rename(columns **=** renames)

df **=** df[['datetime', 'Discharge', 'Gage Height', 'Velocity']] df **=** df**.**set\_index('datetime')

df**.**dropna(axis **=** 0, inplace **= True**)

print(df**.**head()) *#3 columns*

C:\Users\emmac\AppData\Local\Temp\ipykernel\_19692\80301310.py:2: DtypeWarning: Colu mns (2,5,11,13,15,17,19,21,23,25) have mixed types. Specify dtype option on import or set low\_memory=False.

df = pd.read\_csv('henry\_csv\_17-23.csv')#, infer\_datetime\_format= True)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | datetime | Discharge | Gage | Height | Velocity |
| 2017-01-01 | 00:00:00-06:00 20400.0 |  | 16.95 | 2.45 |
| 2017-01-01 | 00:15:00-06:00 19800.0 |  | 16.96 | 2.37 |
| 2017-01-01 | 00:30:00-06:00 20900.0 |  | 16.98 | 2.51 |
| 2017-01-01 | 00:45:00-06:00 19300.0 |  | 16.95 | 2.32 |
| 2017-01-01 | 01:00:00-06:00 20700.0 |  | 16.98 | 2.49 |
| In [7]: | *# Separate* | *dates for future plotting* |  |  |  |
|  | train\_dates **=** df**.**index**.**to\_series() print(train\_dates**.**tail(15)) *#Check last few dates.* | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| datetime |  | | |
| 2022-12-01 | 20:15:00-06:00 | 2022-12-01 | 20:15:00-06:00 |
| 2022-12-01 | 20:30:00-06:00 | 2022-12-01 | 20:30:00-06:00 |
| 2022-12-01 | 20:45:00-06:00 | 2022-12-01 | 20:45:00-06:00 |
| 2022-12-01 | 21:00:00-06:00 | 2022-12-01 | 21:00:00-06:00 |
| 2022-12-01 | 21:15:00-06:00 | 2022-12-01 | 21:15:00-06:00 |
| 2022-12-01 | 21:30:00-06:00 | 2022-12-01 | 21:30:00-06:00 |
| 2022-12-01 | 21:45:00-06:00 | 2022-12-01 | 21:45:00-06:00 |
| 2022-12-01 | 22:00:00-06:00 | 2022-12-01 | 22:00:00-06:00 |
| 2022-12-01 | 22:15:00-06:00 | 2022-12-01 | 22:15:00-06:00 |
| 2022-12-01 | 22:30:00-06:00 | 2022-12-01 | 22:30:00-06:00 |
| 2022-12-01 | 22:45:00-06:00 | 2022-12-01 | 22:45:00-06:00 |
| 2022-12-01 | 23:00:00-06:00 | 2022-12-01 | 23:00:00-06:00 |
| 2022-12-01 | 23:15:00-06:00 | 2022-12-01 | 23:15:00-06:00 |
| 2022-12-01 | 23:30:00-06:00 | 2022-12-01 | 23:30:00-06:00 |
| 2022-12-01 | 23:45:00-06:00 | 2022-12-01 | 23:45:00-06:00 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Name: datetime, dtype: object |  | |
| In [25]: | *# Check out the dataset*  df |  |  |
| Out[25]: | **Discharge** | **Gage Height** | **Velocity** |
|  | **datetime** |  |  |
|  | **2017-01-01 00:00:00-06:00** 20400.0 | 16.95 | 2.45 |
|  | **2017-01-01 00:15:00-06:00** 19800.0 | 16.96 | 2.37 |
|  | **2017-01-01 00:30:00-06:00** 20900.0 | 16.98 | 2.51 |
|  | **2017-01-01 00:45:00-06:00** 19300.0 | 16.95 | 2.32 |
|  | **2017-01-01 01:00:00-06:00** 20700.0 | 16.98 | 2.49 |
|  | **...** ... | ... | ... |
|  | **2022-12-01 22:45:00-06:00** 4510.0 | 14.83 | 0.60 |
|  | **2022-12-01 23:00:00-06:00** 3800.0 | 14.87 | 0.50 |
|  | **2022-12-01 23:15:00-06:00** 3440.0 | 14.88 | 0.45 |
|  | **2022-12-01 23:30:00-06:00** 3510.0 | 14.87 | 0.46 |
|  | **2022-12-01 23:45:00-06:00** 3720.0 | 14.86 | 0.49 |
|  | 199343 rows × 3 columns |  |  |

# Data manipulation

In [26]:

*# LSTM uses sigmoid and tanh that are sensitive to magnitude so values need to be n # Scale before splitting to scale the range of the whole dataset uniformally*

*# Normalize the dataset*

scaler **=** StandardScaler() scaler **=** scaler**.**fit(df)

df\_scaled **=** scaler**.**transform(df)

In [27]:

df\_scaled

Out[27]:

|  |  |  |  |
| --- | --- | --- | --- |
| array([[ | 0.20959378, | -0.13987508, | 0.78355675], |
| [ | 0.16634792, | -0.13691045, | 0.69871099], |
| [ | 0.245632 , | -0.13098117, | 0.84719107], |
| [-1.01282265, | | -0.75355472, | -1.33758719], |
| [-1.0077773 , | | -0.75651935, | -1.32698147], |
| [-0.99264125, | | -0.75948399, | -1.29516431]]) |

In [28]:

*# Convert from an array to dataframe*

df\_scaled **=** pd**.**DataFrame(df\_scaled, columns **=** ['Discharge','Gage Height','Velocity'

print(df\_scaled) print(type(df\_scaled))

|  |  |  |  |
| --- | --- | --- | --- |
| datetime | Discharge | Gage Height | Velocity |
| 2017-01-01 00:00:00-06:00 | 0.209594 | -0.139875 | 0.783557 |
| 2017-01-01 00:15:00-06:00 | 0.166348 | -0.136910 | 0.698711 |
| 2017-01-01 00:30:00-06:00 | 0.245632 | -0.130981 | 0.847191 |
| 2017-01-01 00:45:00-06:00 | 0.130310 | -0.139875 | 0.645682 |
| 2017-01-01 01:00:00-06:00 | 0.231217 | -0.130981 | 0.825980 |
| ... | ... | ... | ... |
| 2022-12-01 22:45:00-06:00 | -0.935701 | -0.768378 | -1.178501 |
| 2022-12-01 23:00:00-06:00 | -0.986875 | -0.756519 | -1.284559 |
| 2022-12-01 23:15:00-06:00 | -1.012823 | -0.753555 | -1.337587 |
| 2022-12-01 23:30:00-06:00 | -1.007777 | -0.756519 | -1.326981 |
| 2022-12-01 23:45:00-06:00 | -0.992641 | -0.759484 | -1.295164 |

[199343 rows x 3 columns]

<class 'pandas.core.frame.DataFrame'>

In [34]:

*# Subset the train/testing data # Train: 2017-2018, 2020*

*# Test: 2019, 2021*

train\_scaled **=** df\_scaled['2017':'2020'] *# Years 2017, 2018 and 2019 (this includes*

test\_scaled **=** df\_scaled['2020':'2021'] *# Year of 2020*

In [35]:

*# We do not want to be training with the discharge we are attempting to replace*

dfx **=** train\_scaled**.**drop("Discharge", axis**=** 1) dfx

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Out[35]: | **datetime** | **Gage Height** | **Velocity** |  |
|  | **2017-01-01 00:00:00-06:00** | -0.139875 | 0.783557 |
|  | **2017-01-01 00:15:00-06:00** | -0.136910 | 0.698711 |
|  | **2017-01-01 00:30:00-06:00** | -0.130981 | 0.847191 |
|  | **2017-01-01 00:45:00-06:00** | -0.139875 | 0.645682 |
|  | **2017-01-01 01:00:00-06:00**  **... 2019-12-31 22:45:00-06:00** | -0.130981  ...  -0.003502 | 0.825980  ... 1.154757 |
|  | **2019-12-31 23:00:00-06:00** | -0.000537 | 1.144151 |
|  | **2019-12-31 23:15:00-06:00** | -0.000537 | 1.133546 |
|  | **2019-12-31 23:30:00-06:00** | 0.002427 | 1.186574 |
|  | **2019-12-31 23:45:00-06:00** | 0.005392 | 1.197180 |
|  | 100908 rows × 2 columns |  |  |
| In [36]: | train\_scaled |  |  |  |
| Out[36]: | **datetime** | **Discharge** | **Gage Height** | **Velocity** |
|  | **2017-01-01 00:00:00-06:00** | 0.209594 | -0.139875 | 0.783557 |
|  | **2017-01-01 00:15:00-06:00** | 0.166348 | -0.136910 | 0.698711 |
|  | **2017-01-01 00:30:00-06:00** | 0.245632 | -0.130981 | 0.847191 |
|  | **2017-01-01 00:45:00-06:00** | 0.130310 | -0.139875 | 0.645682 |
|  | **2017-01-01 01:00:00-06:00**  **... 2019-12-31 22:45:00-06:00** | 0.231217  ... 0.447446 | -0.130981  ...  -0.003502 | 0.825980  ... 1.154757 |
|  | **2019-12-31 23:00:00-06:00** | 0.440238 | -0.000537 | 1.144151 |
|  | **2019-12-31 23:15:00-06:00** | 0.433031 | -0.000537 | 1.133546 |
|  | **2019-12-31 23:30:00-06:00** | 0.461861 | 0.002427 | 1.186574 |
|  | **2019-12-31 23:45:00-06:00** | 0.469069 | 0.005392 | 1.197180 |
|  | 100908 rows × 3 columns |  |  |  |

In [37]:

*# Convert*

train\_scaled **=** train\_scaled**.**to\_numpy() test\_scaled **=** test\_scaled**.**to\_numpy() dfx **=** dfx**.**to\_numpy()

# LSTM

|  |  |  |
| --- | --- | --- |
| In [38]: | | *#As required for LSTM networks, we require to reshape an input data into n\_samples #In this example, the n\_features is 3. We will make timesteps = 672 (past 7 days da*  *#Empty lists to be populated using formatted training data*  trainX **=** [] trainY **=** []  time\_to\_hr **=** 4 *# 4 timesteps per hour*  time\_to\_day **=** time\_to\_hr **\*** 24 *# 24hrs in a day*  n\_future **=** 12 *# Number of timesteps we want to look into the future based on the pa*  n\_past **=** 3 **\*** time\_to\_day *# Number of past timesteps we want to use to predict the f #Reformat input data into a shape: (n\_samples x timesteps x n\_features)*  **for** i **in** range(n\_past, len(train\_scaled) **-** n\_future **+**1): trainX**.**append(dfx[i **-** n\_past : i, 0:train\_scaled**.**shape[1]])  trainY**.**append(train\_scaled[i **+** n\_future **-** 1:i **+** n\_future, 0]) *#0 = Discharge*  trainX, trainY **=** np**.**array(trainX), np**.**array(trainY) |
|  |  | print('trainX shape == {}.'**.**format(trainX**.**shape)) print('trainY shape == {}.'**.**format(trainY**.**shape)) |
|  |  | trainX shape == (100609, 288, 2).  trainY shape == (100609, 1). |
|  |  | Model Dev |
| In | [40]: | **import** os |
|  |  | os**.**getcwd() |

Out[40]:

In [41]:

*# Don't retrain unless necessary*

*# Verify the shape aligns with the training data shape above! # If not, retrain!*

model\_lstm **=** tf**.**keras**.**models**.**load\_model('./saved\_model/LSTM\_Saved\_Henry\_2020\_2021') model\_lstm**.**summary()

'C:\\Users\\emmac\\Documents\\GitHub\\Hysterisis-ML-Modeling'

Total params: 64,126

Trainable params: 64,126

Non-trainable params: 0

In [42]:

*#In my case, trainX has a shape (100301, 672, 3).*

*#100301 because we are looking back 672 timesteps ##(12823 - 14 = 12809).*

*#Remember that we cannot look back 12 timesteps until we get to the 13th timesteps.*

*#Also, trainY has a shape (100301, 1). Our model only predicts a single value, but #it needs multiple variables (5 in my example) to make this prediction.*

*#This is why we can only predict a single day after our training, the day after whe #To predict more days in future, we need all the 5 variables which we do not have. #We need to predict all variables if we want to do that.*

*# The LSTM architecture -- Basic RNN | one layer then dense*

model\_lstm **=** Sequential()

model\_lstm**.**add(LSTM(units**=**125, activation**=**"tanh", input\_shape**=**(trainX**.**shape[1], tra model\_lstm**.**add(Dense(units**=**1))

*# Compiling the model*

model\_lstm**.**compile(optimizer**=**"RMSprop", loss**=**"mse") model\_lstm**.**summary()

batch\_size **=** 16

model\_lstm**.**fit(trainX, trainY, epochs**=**10, batch\_size**=**batch\_size, verbose**=**1)

*#'''*

Total params: 64,126

Trainable params: 64,126

Non-trainable params: 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Epoch 1/10 |  | | | | |
| 6289/6289 | [==============================] | - 949s | 150ms/step | - loss: | 0.0187 |
| Epoch 2/10 |  |  |  |  |  |
| 6289/6289 | [==============================] | - 822s | 131ms/step | - loss: | 0.0141 |
| Epoch 3/10 |  |  |  |  |  |
| 6289/6289 | [==============================] | - 831s | 132ms/step | - loss: | 0.0133 |
| Epoch 4/10 |  |  |  |  |  |
| 6289/6289 | [==============================] | - 817s | 130ms/step | - loss: | 0.0129 |
| Epoch 5/10 |  |  |  |  |  |
| 6289/6289 | [==============================] | - 824s | 131ms/step | - loss: | 0.0120 |
| Epoch 6/10 |  |  |  |  |  |
| 6289/6289 | [==============================] | - 816s | 130ms/step | - loss: | 0.0119 |
| Epoch 7/10 |  |  |  |  |  |
| 6289/6289 | [==============================] | - 820s | 130ms/step | - loss: | 0.0111 |
| Epoch 8/10 |  |  |  |  |  |
| 6289/6289 | [==============================] | - 815s | 130ms/step | - loss: | 0.0105 |
| Epoch 9/10 |  |  |  |  |  |
| 6289/6289 | [==============================] | - 814s | 129ms/step | - loss: | 0.0102 |

Out[42]:

In [43]:

*## Save the entire model as a SavedModel*

**import** os

**if not** "saved\_model" **in** os**.**listdir(): *#If the saved model directory doesn't exist,*

**!**mkdir -p saved\_model

model\_lstm**.**save('saved\_model/LSTM\_Saved\_Henry\_2017\_2019')

Epoch 10/10

6289/6289 [==============================] - 1295s 206ms/step - loss: 0.0099

<keras.callbacks.History at 0x206d8219480>

WARNING:absl:Found untraced functions such as lstm\_cell\_1\_layer\_call\_fn, lstm\_cell\_ 1\_layer\_call\_and\_return\_conditional\_losses while saving (showing 2 of 2). These fun ctions will not be directly callable after loading.

INFO:tensorflow:Assets written to: saved\_model/LSTM\_Saved\_Henry\_2017\_2019\assets INFO:tensorflow:Assets written to: saved\_model/LSTM\_Saved\_Henry\_2017\_2019\assets

## Transform the test data

In [28]:

model\_lstm**.**summary()

Model: "sequential"

Layer (type) Output Shape Param #

lstm (LSTM) (None, 125) 64000

dense (Dense) (None, 1) 126

Total params: 64,126

Trainable params: 64,126

Non-trainable params: 0

In [29]:

testX **=** [] testY **=** []

n\_future **=** 12 *# Number of timesteps we want to look into the future based on the pa*

n\_past **=** 3 **\*** time\_to\_day

scaler2 **=** StandardScaler() scaler2 **=** scaler2**.**fit(test)

test\_scaled **=** scaler2**.**transform(test)

In [30]:

**for** i **in** range(n\_past, len(test\_scaled) **-** n\_future **+**1):

testX**.**append(test\_scaled[i **-** n\_past:i, 1 : test\_scaled**.**shape[1]]) *# Start at [1*

testY**.**append(test\_scaled[i **+** n\_future **-** 1 : i **+** n\_future, 0]) *# 0 = Discharge*

testX, testY **=** np**.**array(testX), np**.**array(testY)

In [31]:

testX**.**shape *# (Only velocity and WL)*

Out[31]:

In [32]:

*#Make prediction*

*# Will only use the last 7 days of the training data to predict X number of days in*

prediction **=** model\_lstm**.**predict(testX, verbose **=** 1)

(33726, 288, 2)

1054/1054 [==============================] - 101s 95ms/step

In [33]:

*#Perform inverse transformation to rescale back to original range*

*#Since we used 5 variables for transform, the inverse expects same dimensions #Therefore, let us copy our values 5 times and discard them after inverse transform*

prediction\_copies **=** np**.**repeat(prediction, test**.**shape[1], axis**=-**1)

y\_pred **=** scaler2**.**inverse\_transform(prediction\_copies)[:, 0] *# Location of Discharg*

test\_predicts **=** test[:**-**n\_past **-**11]**.**copy() test\_predicts["Predicted Discharge"] **=** y\_pred test\_predicts**.**rename(columns**=** {**-**1 : "Predicted Discharge"}) *#test\_predicts = test\_predicts.drop("Dis", axis=1)*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Out[33]: | **datetime** | **Discharge** | **Gage Height** | **Velocity** | **Predicted Discharge** |
|  | **2021-01-01 00:00:00-06:00** | 6430.0 | 14.79 | 0.87 | 8734.588867 |
|  | **2021-01-01 00:15:00-06:00** | 6350.0 | 14.78 | 0.86 | 8841.984375 |
|  | **2021-01-01 00:30:00-06:00** | 6350.0 | 14.78 | 0.86 | 8872.700195 |
|  | **2021-01-01 00:45:00-06:00** | 6150.0 | 14.80 | 0.83 | 8909.343750 |
|  | **2021-01-01 01:00:00-06:00** | 6360.0 | 14.79 | 0.86 | 8925.500000 |
|  | **...** | ... | ... | ... | ... |
|  | **2021-12-28 16:00:00-06:00** | 11600.0 | 15.15 | 1.55 | 15124.246094 |
|  | **2021-12-28 16:15:00-06:00** | 9610.0 | 15.20 | 1.27 | 14987.659180 |
|  | **2021-12-28 16:30:00-06:00** | 10000.0 | 15.18 | 1.33 | 14979.979492 |
|  | **2021-12-28 16:45:00-06:00** | 9890.0 | 15.18 | 1.31 | 14758.552734 |
|  | **2021-12-28 17:00:00-06:00** | 10900.0 | 15.20 | 1.45 | 14742.546875 |

33726 rows × 4 columns

In [34]:

*# Plot one event to examine model performance*

**import** seaborn **as** sns tstart **=** '2021-06-15'

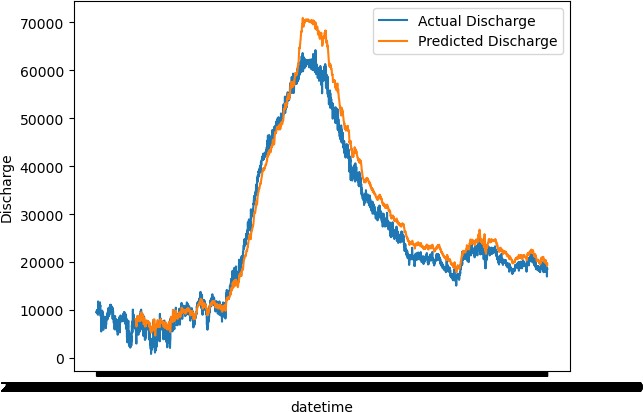
tend **=** '2021-07-20'

sns**.**lineplot(test['Discharge'][tstart : tend], errorbar**= None**) sns**.**lineplot(test\_predicts["Predicted Discharge"][tstart : tend]**.**shift(n\_past), err *# shifting 7 to overlay due to inherent shifting from original formatting of the da*

plt**.**legend(labels**=**["Actual Discharge","Predicted Discharge"])

Out[34]:

<matplotlib.legend.Legend at 0x1b0239f2ec0>



In [35]:

*# Plot whole test year to examine model performance (this takes a little while to r*

tstart **=** '2021-01-01'

tend **=** '2021-12-31'

sns**.**lineplot(test['Discharge'], errorbar**= None**) sns**.**lineplot(test\_predicts["Predicted Discharge"]**.**shift(n\_past), errorbar**= None**)

*# shifting 7 to overlay due to inherent shifting from original formatting of the da*

plt**.**legend(labels**=**["Actual Discharge","Predicted Discharge"])

A graph showing a couple of blue and orange lines

Description automatically generated

*# Evaluate model using statistics # First load them in*

**from** sklearn.metrics **import** mean\_squared\_error, r2\_score, mean\_squared\_log\_error

In [37]:

*# Report mean squared log error*

mean\_squared\_log\_error(test['Discharge'][:**-**n\_past**-**11], test\_predicts["Predicted Dis

In [38]:

Out[38]:

*# Report mean squared error*

mean\_squared\_error(test['Discharge'][:**-**n\_past**-**11], test\_predicts["Predicted Dischar

In [39]:

0.18970896423009212

Out[39]:

*# Report R2*

r2\_score(test['Discharge'][:**-**n\_past**-**11], test\_predicts["Predicted Discharge"])

In [40]:

40063875.04360296

Out[40]:

0.622771435658164

# Future Predictions

In [110…

*# Run test*

*# Prediction*

*#pred\_WL = model\_lstm.predict(testX)* n\_future\_pred **=** 3 **\*** time\_to\_hr overlap **=** 1

pred\_dis **=** model\_lstm**.**predict(testX[**-**n\_future\_pred **-** overlap:])

pred\_dis **=** np**.**repeat(pred\_dis, test**.**shape[1], axis**=-**1)

future\_dis **=** scaler2**.**inverse\_transform(pred\_dis)[:, 0] *# Location of Discharge in*

1/1 [==============================] - 0s 56ms/step

In [111…

predict\_period\_dates **=** pd**.**date\_range(list(test**.**index)[**-**(overlap)],periods**=**n\_future\_ predict\_period\_dates

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Out[111]: | array([Timestamp('2021-12-31  T'), | 23:45:00-0600', | tz='pytz.FixedOffset(-360)', | freq='15 |
|  | Timestamp('2022-01-01 | 00:00:00-0600', | tz='pytz.FixedOffset(-360)', | freq='15 |
|  | T'), |  |  |  |
|  | Timestamp('2022-01-01 | 00:15:00-0600', | tz='pytz.FixedOffset(-360)', | freq='15 |
|  | T'), |  |  |  |
|  | Timestamp('2022-01-01 | 00:30:00-0600', | tz='pytz.FixedOffset(-360)', | freq='15 |
|  | T'), |  |  |  |
|  | Timestamp('2022-01-01 | 00:45:00-0600', | tz='pytz.FixedOffset(-360)', | freq='15 |
|  | T'), |  |  |  |
|  | Timestamp('2022-01-01 | 01:00:00-0600', | tz='pytz.FixedOffset(-360)', | freq='15 |
|  | T'), |  |  |  |
|  | Timestamp('2022-01-01 | 01:15:00-0600', | tz='pytz.FixedOffset(-360)', | freq='15 |
|  | T'), |  |  |  |
|  | Timestamp('2022-01-01 | 01:30:00-0600', | tz='pytz.FixedOffset(-360)', | freq='15 |
|  | T'), |  |  |  |
|  | Timestamp('2022-01-01 | 01:45:00-0600', | tz='pytz.FixedOffset(-360)', | freq='15 |
|  | T'), |  |  |  |
|  | Timestamp('2022-01-01 | 02:00:00-0600', | tz='pytz.FixedOffset(-360)', | freq='15 |
|  | T'), |  |  |  |
|  | Timestamp('2022-01-01 | 02:15:00-0600', | tz='pytz.FixedOffset(-360)', | freq='15 |
|  | T'), |  |  |  |
|  | Timestamp('2022-01-01 | 02:30:00-0600', | tz='pytz.FixedOffset(-360)', | freq='15 |
|  | T'), |  |  |  |
|  | Timestamp('2022-01-01 | 02:45:00-0600', | tz='pytz.FixedOffset(-360)', | freq='15 |
|  | T')], |  |  |  |
|  | dtype=object) |  |  |  |

In [112… Out[112]:

future\_dis **=** np**.**reshape(future\_dis, (len(future\_dis),))

predict\_period\_dates**.**shape

In [113…

(13,)

In [115…

'''

# Convert timestamp to date forecast\_dates = []

for time\_i in predict\_period\_dates: forecast\_dates.append(time\_i.date())

len(y\_pred\_future) '''

df\_forecast **=** pd**.**DataFrame({'Date': pd**.**to\_datetime(predict\_period\_dates),

'Discharge':future\_dis})

In [116…

df\_forecast**.**set\_index('Date', inplace**= True**)

In [117…

original **=** df[['Discharge']]

original **=** original**.**loc['2021-12-25': '2022']

In [118…

joint\_df **=** pd**.**concat([original, df\_forecast])

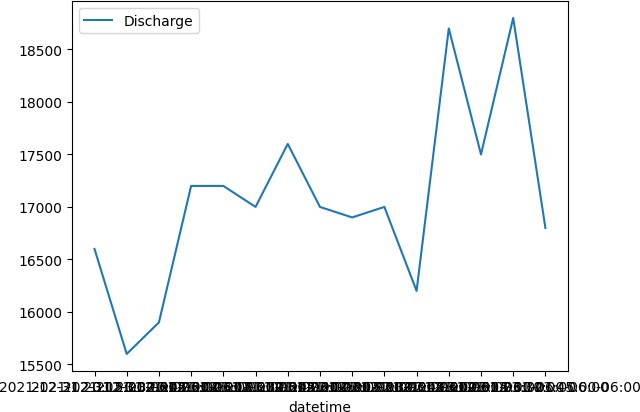
In [ ]: In [136…

sns**.**lineplot(original[**-**15:], errorbar**= None**) print(original[**-**1:])

Discharge

datetime

2021-12-31 23:45:00-06:00 16800.0



In [139…

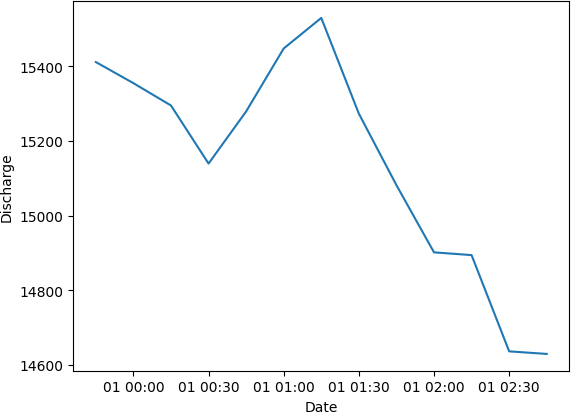
sns**.**lineplot(df\_forecast, x**=** "Date", y**=** "Discharge", errorbar**= None**) df\_forecast[:1]

Out[139]:

**Date**

**Discharge**

**2021-12-31 23:45:00-06:00** 15411.088867



|  |  |  |
| --- | --- | --- |
| Out[93]: | **Date** | **Gage Height** |
|  | **2022-12-01 00:00:00-06:00** | 0.959061 |
|  | **2022-12-01 00:15:00-06:00** | 0.948979 |
|  | **2022-12-01 00:30:00-06:00** | 0.939894 |
|  | **2022-12-01 00:45:00-06:00** | 0.954180 |
|  | **2022-12-01 01:00:00-06:00** | 0.956698 |
|  | **2022-12-01 01:15:00-06:00** | 0.951986 |
|  | **2022-12-01 01:30:00-06:00** | 0.951351 |
|  | **2022-12-01 01:45:00-06:00** | 0.944818 |
|  | **2022-12-01 02:00:00-06:00** | 0.943667 |
|  | **2022-12-01 02:15:00-06:00** | 0.950522 |
|  | **2022-12-01 02:30:00-06:00** | 0.939742 |
|  | **2022-12-01 02:45:00-06:00** | 0.935389 |
|  | **2022-12-01 03:00:00-06:00** | 0.938183 |

*# Convert timestamp to date*

forecast\_dates **=** []

**for** time\_i **in** predict\_period\_dates: forecast\_dates**.**append(time\_i**.**date())

df\_forecast **=** pd**.**DataFrame({'Date':np**.**array(forecast\_dates), 'Open':y\_pred\_future}) df\_forecast['Date']**=**pd**.**to\_datetime(df\_forecast['Date'])

original **=** df[['Date', 'Open']] original['Date']**=**pd**.**to\_datetime(original['Date']) original **=** original**.**loc[original['Date'] **>=** '2020-5-1']

sns**.**lineplot(original['Date'], original['Open']) sns**.**lineplot(df\_forecast['Date'], df\_forecast['Open'])

In [ ]: