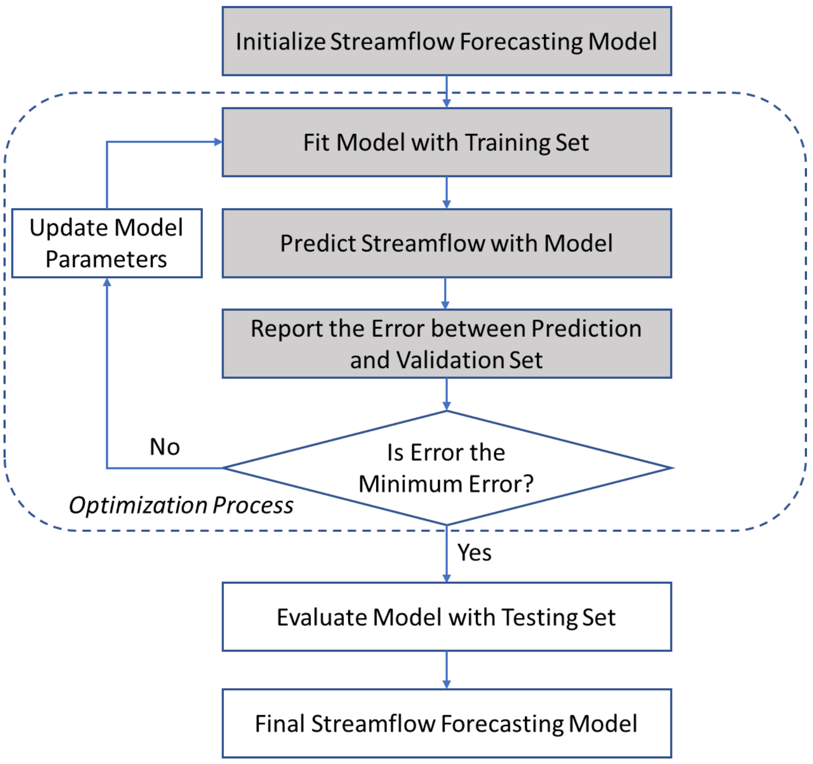
FAIR CLIMATE AND WATER SCIENCE
Module 2: Data Processing (DP) for Water Science
Daily Streamflow Forecasting by First Order Exponential Smoothing Method

Prepared by Pin-Ching Li, Sayan Dey and Venkatesh Merwade
Lyles School of Civil Engineering, Purdue University
vmerwade@purdue.edu

1. INTRODUCTION

Long-term prediction of streamflow is useful for the planning and operation of water resources. Similarly, short-term prediction of streamflow is important for managing extreme events and reservoirs. Several approaches exist to predict streamflow time series, such as exponential smoothing method, Auto Regressive Integrated Moving Average (ARIMA) model and machine learning tools. The objective of this exercise is to develop an exponential smoothing model for streamflow forecasting using past streamflow timeseries.

The flowchart of optimizing a streamflow forecasting model is shown below. This whole process is divided into three tutorials. This tutorial is the start of the tutorial series. Students would learn how to build-up their forecasting model and try to predict the streamflow by their model without optimization, which are the first four steps in the flow chart. The instruction of building-up a first order exponential smoothing model for prediction of streamflow of a USGS gage is given.


Initialize Streamflow Forecasting Model
Fit Model with Training Set
Update Model Parameters
Predict Streamflow with Model
Report the Error between Prediction and Validation Set
No
Is Error the Minimum Error?
Optimization Process
Yes
Evaluate Model with Testing Set
Final Streamflow Forecasting Model

2. COMPUTER AND DATA REQUIREMENTS

A web browser is required for this tutorial with good internet connection and login credentials for an account on www.mygeohub.org. For this exercise, a Jupyter notebook file, streamflow_forecasting_exercise.ipynb, is provided to you. It has code for downloading USGS streamflow dataset using hydrofunctions library, and the definition of class for a first order exponential smoothing model.

3. INSTRUCTIONS

3.1 Downloading USGS streamflow data (code provided)

Until now you have learned how to write code to download USGS streamflow data by building the NWIS url. Here you going to use an existing library, called hydrofunctions to get the data by feeding the NWIS object (gage number, 'iv' for instantaneous observation, start date, end date) and applying get_data() function to download the dataset. Here we download instantaneous data for USGS 03335500 WABASH RIVER AT LAFAYETTE, IN for a period of 6 months from 1st January 2019 to 30th June 2019. After the dataframe of streamflow dataset is obtained, resample the dataframe into daily mean streamflow series.

```
import numpy as np                                # vectors and matrices
import pandas as pd                               # tables and data manipulations
import warnings                                   # There would be no warnings
warnings.filterwarnings('ignore')
import hydrofunctions as hf
# create NWIS by the information of streamflow station
observation = hf.NWIS('03335500', 'iv', start_date='2019-01-01',end_date='2019-06-30')
# apply function to get streamflow dataset
observation.get_data()
# store dataset as a dataframe
Timeseries = observation.df()
Timeseries.columns=["discharge","flag"]
# show the first 5 result of dataframe
Timeseries.head()
# save the dataframe into .csv file
Timeseries.to_csv("Timeseries.csv",sep = ',')
# resample the time series dataset from instantaneous data into daily data
Daily = Timeseries.resample('D').mean()
```

3.2 Plotting Streamflow Data

Create a cell to plot the daily streamflow time series for preliminary visualization and checking if there is any missing value. Name the plot as: "2019 USGS 03335500 WABASH RIVER AT LAFAYETTE, IN". The x axis title should have date (yyyy/mm) and y-axis title should be "Discharge (cfs)".

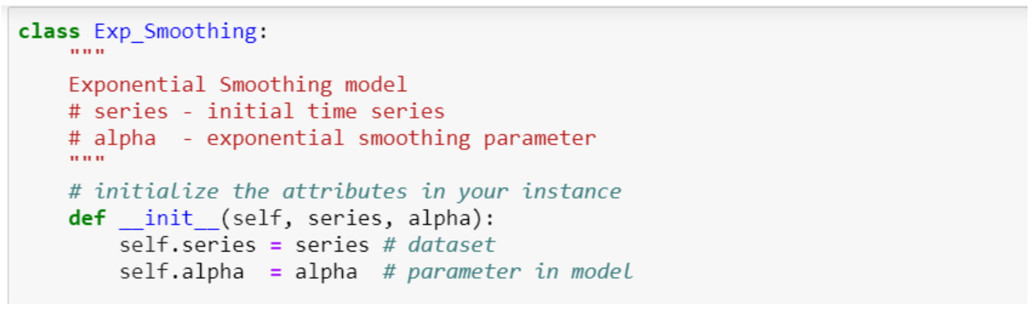
```
import matplotlib.pyplot as plt
%matplotlib inline
# transfer the date time from string type to datetime type
Time = pd.to_datetime(Daily.index)
```

3.3 Building a first order exponential smoothing model

First order exponential smoothing model is a single smoothing technique for time series. The forecasting formula is shown below:

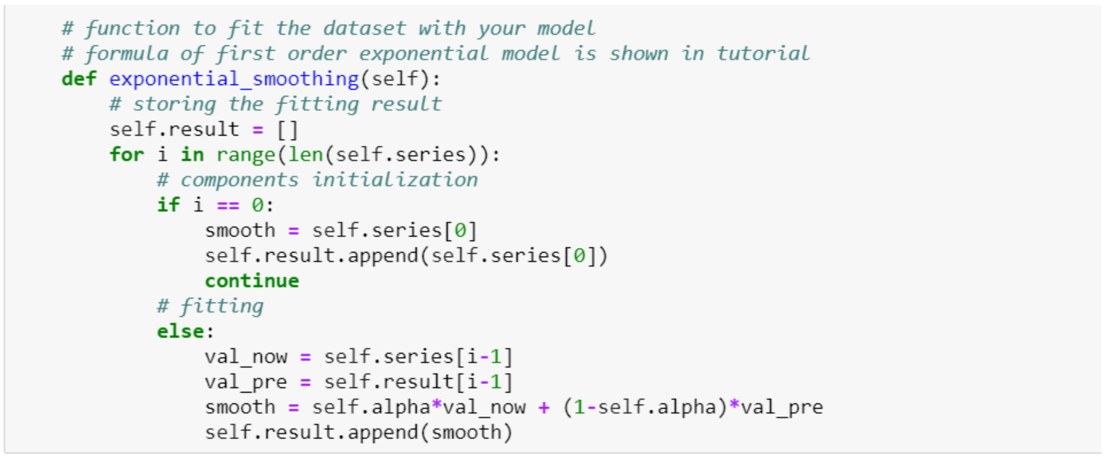
$$\hat{y}_t = \alpha y_{t-1} + (1-\alpha)\hat{y}_{t-1}$$

A class to implement the above equation for the exponential smoothing model is created in an object-oriented programming (OOP) style. Python Class serves as a user-defined blueprint for bundling attributes of object and functions together. The method __init__(self, arguments) in a class is a constructor used to initialize an instance (“self”) in Exp_Smoothing class. The attributes in “self” are streamflow dataset (series), and the parameter of exponential smoothing model (alpha).



```python
class Exp_Smoothing:
    """
    Exponential Smoothing model
    # series - initial time series
    # alpha  - exponential smoothing parameter
    """
    # initialize the attributes in your instance
    def __init__(self, series, alpha):
        self.series = series # dataset
        self.alpha  = alpha  # parameter in model
```

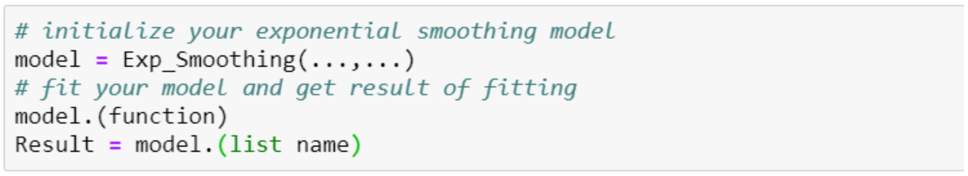
The smoothing process of model is done within exponential_smoothing() function in Exp_Smoothing class. In the first step (i = 0), the result of smoothing equals to the observation (initial condition). When i > 0, the function would calculate the result based on the forecasting formula.



```python
    # function to fit the dataset with your model
    # formula of first order exponential model is shown in tutorial
    def exponential_smoothing(self):
        # storing the fitting result
        self.result = []
        for i in range(len(self.series)):
            # components initialization
            if i == 0:
                smooth = self.series[0]
                self.result.append(self.series[0])
                continue
            # fitting
            else:
                val_now = self.series[i-1]
                val_pre = self.result[i-1]
                smooth = self.alpha*val_now + (1-self.alpha)*val_pre
                self.result.append(smooth)
```

3.4 Plotting Output from first order model

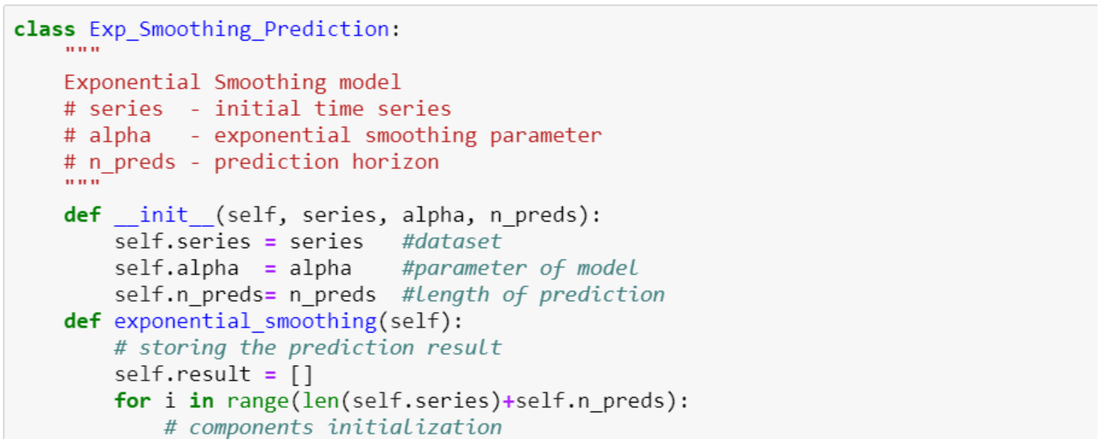
Write code to call and plot the model output (use first order smoothing with $\alpha = 1$) along with observation dataset. The template of how to call the model is shown below:



```
# initialize your exponential smoothing model
model = Exp_Smoothing(...,...)
# fit your model and get result of fitting
model.(function)
Result = model.(list name)
```

3.5 Making Predictions from your first order model

Write code similar to code in 3.3 to run the model for more days than original dataset to make predictions for the extra days. Call this class as Exp_Smoothing_Prediction. The initial part of your code will look like below. You will be running your for-loop for the extra days (n_preds).

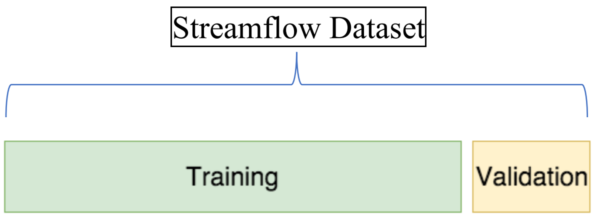


```
class Exp_Smoothing_Prediction:
    """
    Exponential Smoothing model
    # series   - initial time series
    # alpha    - exponential smoothing parameter
    # n_preds - prediction horizon
    """
    def __init__(self, series, alpha, n_preds):
        self.series = series   #dataset
        self.alpha  = alpha    #parameter of model
        self.n_preds= n_preds  #length of prediction
    def exponential_smoothing(self):
        # storing the prediction result
        self.result = []
        for i in range(len(self.series)+self.n_preds):
            # components initialization
```

Call this function to make prediction for 10 more days to see how your prediction looks like using alpha = 1. You can play with alpha values to see what value gives you the best result. It will not be perfect or good, but just choose one that looks reasonable to you.

Homework: Report the error of prediction made by your exponential model

Split your streamflow dataset into training set and validation set. The prediction made by your model is compared to the validation set split in the beginning. This procedure is known as the training process, which is popular in optimization problem. Create a function to report the mean square error of your model. Fit your model with training set and change alpha value manually. Find the best model which returns the minimum MSE value. Turn in the alpha value and the MSE value you get.



```
from sklearn.metrics import mean_squared_error
def Train_Score(param, series, validsize, loss_function=mean_squared_error):
    """
        Returns error
        param    - parameter for optimization
        series    - timeseries dataset
        validsize- size of validation dataset
    """
    values = series.values
    alpha  = param

    return error
```