MileStone3 – More Data Analysis
METEO473 Spring 2016

Goals: In this project, we will build on previous milestones and write some statistical modeling tools. In the first, you will estimate when a change can be identified in statistical distributions generated from a statistical model generating known data with known distributions. In the second, you will examine uncertainty in estimated trends from annual mean global-mean temperature data given various sources of statistical noise generated by your code.

Step 1) Make a directory called:

/home/meteo/your_username/meteo473Sp2016/MileStones/MileStone3

and change its permissions to 700. Within this directory make a script called

MileStone3A_DataAnalysis_LastNameFirstName.py

where "LastNameFirstName" is your name. Do not use nicknames in your file names and remove apostrophes. For example, Bill O'Brian would be listed as OBrianWilliam in the file name. The instructors would have Python files named:

MileStone3A_ DataAnalysis _BalashovNikolay.py
MileStone3A_ DataAnalysis _ForestChris.py

Part A:

Step 2) In Part A, using a random number generator, you will write a simple model to mimic the roll of single six-sided die. This can then be used to generate a sample of N rolls of the die and to plot it as a histogram of the events. Once you've generated this, you will then modify the model by introducing a bias in the likelihood to roll one of the numbers. Your code should have the following options for inputs: N_rolls = number of rolls in each sample, B = bias in likelihood, X = number to be biased. Hint: One simple way to generate a six-sided die is to generate a uniform random number using the "random" module in Python and assign the roll number to the sub-divided range of outcomes.

Tasks for Part A:

1. Create a file UserFunctions.py and inside of this file make a function *generate_sample*. Turn this function into a simple statistical model that mimics rolling a 6 sided die to produce a random number from 1 to 6. The model should also include a simple modification to weight the die to increase the probability of one number. First, increase

the probability of a 3, and second, for a 6. When you run this function in the interactive ipython command line, the script should ask a series of questions to input the ***number of rolls (N_events)***, ***the bias in likelihood (B)***, and ***for the number to be biased (X)***. The function will then return a single sample for the chosen parameters.

2. In the file UserFunctions.py create a function called *plot_histogram* to produce a plot to the screen showing the histogram for a given sample with various options for {N_events, B, X}. This function should include a second option in which it displays a two panel plot. In one panel, it will display two histograms overlaid when two samples are input. In the second panel, the function must display the ratio of the same two histograms. Note: we define a "sample" as a set of draws (or rolls) for a given probability distribution with a specific set of parameters.
3. Lastly, create a third function *sample_stat* in the file UserFunctions.py to estimate the standard deviation, kurtosis, and skewness values for each sample that you generated.

Once you have your code running successfully, please answer the following:
For a given change in probability, how many roles does will it take to identify the changes?

Extra Credit: How do the results change when rolling two dice (or N dice)?

Deliverables:
UserFunctions.py with 3 functions: *generate_sample, plot_histogram*, and *sample_stat*.

The function *generate_sample* will be tested by the instructors for its ability to handle errors and also for producing the desired results.

The *plot_histogram* function should be able to create two types of plots. For one case, it will produce a plot of a single histogram of a single sample. For the second case, it will produce a two-part figure including: Part1: a plot with two histograms overlaid and Part 2: a plot with the ratio of two histograms. These are the only required plots for the function. Other options can be included if desired.

The function *sample_stat* should estimate the standard deviation, kurtosis, and skewness values for each sample and return these values as well as print them to the screen.

For the *MileStone3A_DataAnalysis_LastNameFirstName.py* code, we provide an example pseudo-code that suggests how the three functions will be used to answer the question.

Example pseudo-code:
import UserFunctions which includes the functions: CreateSample() and PlotHistogram()
run CreateSample for 1 sample
run CreateSample for a second sample
run PlotHistogram with one sample
run PlotHistogram with two samples to compare the statistics

(2) Provide an answer to the question and a short interpretation of its result.

Part B:

Step 3) For Part B, you will download a file containing the global-mean annual-mean surface temperature and estimate the trends for a 50-year subset of the data (1966-2015). Then, you will generate and add 500 individual time-series of random noise to this 50-yr subset and **re-estimate the trend and its uncertainty each time**. This will provide a distribution of trend estimates from your sample to assess the uncertainty in the trend. *Your first goal is to assess the magnitude of the random noise that is required to consider the observed trend to be insignificant.* You will then repeat this process with a lag-1 auto-correlated data and re-estimate the trends with another 500 individual time-series of random noise. *Your second goal is to assess the magnitude of the auto-correlation coefficient (while keeping the magnitude of the random noise fixed at 0.02 K) that is required to consider the observed trend to be insignificant.*

Within your MileStone3 directory make a script called:

MileStone3B_GMST_DataAnalysis_LastNameFirstName.py

where "LastNameFirstName" is your name.

Step 4) For Part B, download the full data set for global-mean annual-mean surface temperature data as found here: http://hadobs.metoffice.com/hadcrut3/diagnostics/comparison.html

1. Load the data into python and extract 50-year period from 1966-2015 and estimate trend for the data. For this estimate, assume that the temperature data are perfect.
2. Generate a 50-year time-series of random noise drawn from a normal (aka Gaussian) distribution with zero mean, *mu*, and standard deviation, *sigma*. (use: `random.normalvariate(mu,sigma)`) Add this noise to the global-mean annual-mean surface temperature time-series from step 1.
3. Generate a 50-year time-series of noise from a normal distribution (as in Step 2) but add a lag-1 auto-correlation noise (i.e., AR(1) noise) to the annual data. For this case, the noise at each time step, *AR1_noise(t)*, will be the sum of the noise at year, *t*, given by *random(t)*

plus the the noise from the previous year multiplied by a parameter *alpha*. In this case, set *sigma* to be a constant and your noise estimate will be:

*AR1_noise(t) = random(t) + alpha*random(t-1))*

where *alpha* is now the lag-1 auto-correlation and *random(t)* is the noise time series from Step 2.

4. Using results from steps 2 and 3, generate distribution of trends with the two cases for n = 500 samples of random noise added to the global-mean temperature data. In this case, you you choose *sigma* and *alpha* and create 500 realizations of the noise data. Your code will require input arguments for the values of *sigma* and *alpha*.

Answer the following questions:

- What value of *sigma* is required to consider the observed trend to be insignificant?
- What value of the auto-correlation coefficient, *alpha*, (while keeping the magnitude of the random noise, *sigma*, fixed at 0.02 K) is required to consider the observed trend to be insignificant?
- What is the impact of adding noise overall?
- What is the main impact of adding the lag-1 auto-correlated noise to the data?

Extra credit: Repeat this for 50-year periods starting in 1860, 1870, ..., 1960
Answer the following: Does the impact differ for different time-periods?

Deliverables:

(1) Produce a python function that generates a histogram plot of the trend estimates for the n=500 samples for a given choice of the input *sigma* and *alpha* values (and any other required options for plotting). The plot must be clearly labeled and include a vertical bar indicating the estimated trend for the perfect case. Values of *sigma*, *alpha*, and time-period must be displayed on the plot. The function will be tested by the instructors for its ability to handle errors and also produce the desired results. If it isn't obvious, this should make use of code from part A.

(2) Provide a short discussion to answer the questions for Part B.

Comments:

Here is a list of the modules you may want to use for this assignment:

```
numpy
scipy
pylab
random
matplotlib.pyplot
mpl_toolkits.basemap
```

Useful plots to consider:
- histograms for fair and loaded dice
- histograms for trends for uncorrelated and correlated noise
- time-series plots for dice throws and temperatures

Required UserFunctions:
plot_histogram()
statistical_functions for standard deviation, kurtosis, and skewness