Authors - JT Kashuba
Group - Keyboard Warriors
Last Modified - 2/10/21

Document that stores notes for code documentation standards, example function calls, necessary operation ordering and other miscellaneous project information.

# Documentation Specifications

## All numerical values passed as arguments will be floats.

## When creating a tree, root node is always set as denoise

## How Execute Tree works:

Creates a dictionary. For every single path within the tree, it executes it. Each path returns something - and whatever it returns is placed in the dictionary with the key:value pair represented as the following: key:value = leaf node:result of that pipeline

==========

## Keyboard Warriors Docstring Standard

“””
Short one-line description of functions purpose

Calls: function_A(par_1, par_2)

Called By: function_B()

Returns: _____
“””

==========

**NOTE: THE FOLLOWING EXAMPLE HAS BEEN DEPRECATED**

**Example function calls:**
Examples of how to call/use functions for the end user/sysadmin to have an idea of how to use the program. (Useful for Testing as well as essential for README and other Documentation for end-user or sysadmin to be able to easily start using or dissecting the program/code)

Instead of this format:
clip(ts, starting_date, final_date):
‘’’

‘’’

We’ll be passing all arguments other than the time series in a list like so:
“””
clip(ts, list_of_args):
‘’’
list_of_args = [starting_date, final_date]

So we can access the starting date with list_of_args[0]
and final_date with list_of_args[1]

==========

## Pandas Dataframe information

**Basic data frame creation:**

Import pandas as pd

fileName = **“**temperature.csv**”**
data = pd.read_csv(fileName)
dataframe = pd.DataFrame(data)

**Basic dataframe calls:**

*Select a row:*
dataframe.iloc[ROWINDEX]

Select last row regardless of number of columns size:
X = dataframe.iloc[:,-1,]

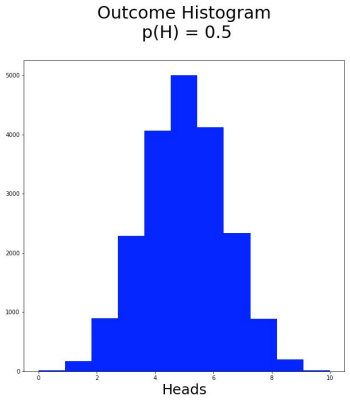
*Looping through an column selecting only the numerical data:*
```
if len(df.columns) == 1:
        for index, row in df.iterrows():
            print(row)
```

==========

## PLOTTING DATA:

```
import matplotlib.pyplot as plt


np.random.seed(15) # make sure that everyone gets the same results
# DO NOT CHANGE the seed when submitting your notebook.

N = 20000 # number of trials
n = 10    # number of flips per trial
p = 0.5   # probability of heads


result = [] # contains at each index the amount of heads that were flipped

for trial in range (N):
    trial_result = np.random.rand(n) <= p
    result.append( np.sum(trial_result) )


plt.figure(figsize = (10,10))

plt.hist( result, bins = n+1   , color = 'blue' )

plt.title( 'Outcome Histogram\n p(H) = '+str(p)+'\n', fontsize = 30 )
plt.xlabel( 'Heads', fontsize  = 25 )
plt.show()
```

**OUTPUT:**




==========

## Nosetest Example:

It needs to be inside of a file named test_something.py

```
def test_evens():
        for i in range(0, 5):
        yield check_even, i, i*3

def check_even(n, nn):
        assert n % 2 == 0 or nn % 2 == 0
```

Here, we are testing "check_even()." Our test (what we will all inevitably be writing and pushing to the corresponding test files) in this example is "test_evens()." For a test to be observed by nosetests (activated upon typing 'nosetest file_holding_test.py"), the function name must contain "test" or "Test". In the example above, the use of "yield" allows us to run multiple tests within one function.

==========

## FOR TREE ARCHITECTURE:

# The function declaration changes are:

assign_time(ts, data_start: float, increment: float)

design_matrix(ts, data_start: float, data_end: float)

split_data(ts, perc_training, perc_valid, perc_test)

ts2db(input_filename, perc_training, perc_valid, perc_test, data_start, data_end,  output_filename)

mlp_model(ts, layers)

mse(ts, input_filename: str)

mape(ts, input_filename: str)

smape(ts, input_filename: str)

==========

***ISSUE:*** At least 5 functions (split_data, mlp_model, mse, mape, smape) will not be returning a time series. This is an issue for the way Noah is structuring the way the Tree executes pipelines.

**SOLUTION:** Make these functions to have to be leaf nodes of the tree

==========

## List of which functions are dependent on others

Fit is dependent on model
Fit is dependent on design_matrix
Forecast is dependent on fit
Model -> fit -> forecast
smape is dependent on forecast
ts2db() calls read_from_file, split_data, design_matrix in that order

==========

## List of functions that were slightly renamed, omitted, or combined with another function:

Removed design_matrix_2(ts, mi, ti, mo, to).

Combined mlp.fit() into mlp_model().

Changed mlp_model() header to be mlp_model(train, layers, window_size) where train is the combination of the x_train and y_train data and window_size a scaling factor to linearly interpolate the data before passing it into the model.

Changed mlp.forecast to mlp_forecast(model_data, x_filename) where model_data also includes a scaling factor to de-linearly interpolate the data after its been output from the model.
X_filename also takes in a filename where the input matrix to forecast from can be loaded.

Added additional function db2ts() which converts forecasts back into time series.

Added optional argument fname to plot(ts, fname), histogram(ts, fname) and boxplot(ts, fname). These additional arguments