Using the Pandas Python Data Toolkit

Today we will highlight some very useful and cool features of the Pandas library in Python while playing with some nematode worm behaviour data collected from the multi-worm-tracker (Swierczek et al., 2011).

Specifically, we will explore:

1. Loading data
2. Dataframe data structures
3. Element-wise mathematics
4. Working with time series data
5. Quick and easy visualization

Some initial setup

In [1]:

```
## load libraries
%matplotlib inline
import pandas as pd
import numpy as np

from pandas import set_option
set_option("display.max_rows", 4)

## magic to time cells in ipython notebook
%install_ext https://raw.github.com/cpcloud/ipython-autotime/maste
r/autotime.py
%load_ext autotime
```

```
Installed autotime.py. To use it, type:
  %load_ext autotime
```

1. Loading data from a local text file

More details, see http://pandas.pydata.org/pandas-docs/stable/io.html (http://pandas.pydata.org/pandas-docs/stable/io.html)

Let's first load some behaviour data from a collection of wild-type worms.

In [2]:
```
filename = 'data/behav.dat'
behav = pd.read_table(filename, sep = '\s+')
behav
```

Out[2]:

	plate	time	strain	frame	area	speed	angular_speed
0	20141118_131037	5.065	N2	126	0.094770	0.3600	0.8706
1	20141118_131037	5.109	N2	127	0.094770	0.3600	0.8630
...	...	...	...	...	...	...	...
249997	20141118_132717	249.048	N2	6158	0.108621	0.0792	0.5000
249998	20141118_132717	249.093	N2	6159	0.107892	0.0693	0.6000

249999 rows × 13 columns

```
time: 632 ms
```

2. Dataframe data structures

For more details, see http://pandas.pydata.org/pandas-docs/stable/dsintro.html (http://pandas.pydata.org/pandas-docs/stable/dsintro.html)

Pandas provides access to data frame data structures. These tabular data objects allow you to mix and match arrays of different data types in one "table".

In [3]:
```
print behav.dtypes
```

```
plate           object
time           float64
                 ...
bias           float64
pathlength     float64
dtype: object
time: 5.22 ms
```

3. Element-wise mathematics

Suppose we want to add a new column that is a combination of two columns in our dataset. Similar to `numpy`, `Pandas` lets us do this easily and deals with doing math between columns on an element by element basis. For example, We are interested in the ratio of the midline length divided by the morphwidth to look at whether worms are crawling in a straight line or curling back on themselves (*e.g.,* during a turn).

In [4]:
```
## vectorization takes 49.3 ms
behav['mid_width_ratio'] = behav['morphwidth']/behav['midline']
behav[['morphwidth', 'midline', 'mid_width_ratio']].head()
```

Out[4]:

	morphwidth	midline	mid_width_ratio
0	1	12.1	0.082645
1	1	5.9	0.169492
...	...	...	...
3	1	14.9	0.067114
4	1	6.3	0.158730

5 rows × 3 columns

```
time: 54.9 ms
```

In [ ]:
```
## looping takes 1 min 44s
mid_width_ratio = np.empty(len(behav['morphwidth']), dtype='float6
4')

for i in range(1,len(behav['morphwidth'])):
    mid_width_ratio[i] =+ behav.loc[i,'morphwidth']/behav.loc[i,'mi
dline']

behav['mid_width_ratio'] = mid_width_ratio
behav[['morphwidth', 'midline', 'mid_width_ratio']].head()
```

`apply()`

For more details, see: [http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.apply.html (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.apply.html)](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.apply.html)

Another bonus about using Pandas is the `apply` function - this allows you to apply any function to a select column(s) or row(s) of a dataframe, or accross the entire dataframe.

In [5]:
```
## custom function to center data
def center(data):
    return data - data.mean()
```

time: 1.77 ms

In [8]:
```
## center all data on a column basis
behav.iloc[:,4:].apply(center).head()
```

Out[8]:

	area	speed	angular_speed	aspect	midline	morphwidth	kink
0	-0.002280	0.249039	-6.313001	-0.219804	11.004384	0.904059	-43.962
1	-0.002280	0.249039	-6.320601	-0.220104	4.804384	0.904059	-43.955
...	...	...	...	...	...	...	...
3	-0.000093	0.229039	-6.279701	-0.220304	13.804384	0.904059	-43.942
4	0.000636	0.221039	-6.257501	-0.217504	5.204384	0.904059	-43.935

5 rows × 10 columns

time: 66.4 ms

4. Working with time series data

Indices

For more details, see http://pandas.pydata.org/pandas-docs/stable/indexing.html (http://pandas.pydata.org/pandas-docs/stable/indexing.html)

Given that this is time series data we will want to set the index to time, we can do this while we read in the data.

In [9]:
```
behav = pd.read_table(filename, sep = '\s+', index_col='time')
behav
```

Out[9]:

	plate	strain	frame	area	speed	angular_speed	aspect
time							
5.065	20141118_131037	N2	126	0.094770	0.3600	0.8706	0.0822
5.109	20141118_131037	N2	127	0.094770	0.3600	0.8630	0.0819
...	...	...	...	...	...	...	...
249.048	20141118_132717	N2	6158	0.108621	0.0792	0.5000	0.1470
249.093	20141118_132717	N2	6159	0.107892	0.0693	0.6000	0.1520

249999 rows × 12 columns

```
time: 620 ms
```

To utilize functions built into Pandas to deal with time series data, let's convert our time to a date time object using the `to_datetime()` function.

In [10]:
```
behav.index.dtype
```

Out[10]:
```
dtype('float64')
```

```
time: 2.56 ms
```

In [11]:
```
behav.index = pd.to_datetime(behav.index, unit='s')
print behav.index.dtype
behav
```

```
datetime64[ns]
```

Out[11]:

	plate	strain	frame	area	speed	angular_speed	asp
1970-01-01 00:00:05.065	20141118_131037	N2	126	0.094770	0.3600	0.8706	0.0
1970-01-01 00:00:05.109	20141118_131037	N2	127	0.094770	0.3600	0.8630	0.0
...	...	...	...	...	...	...	...
1970-01-01 00:04:09.048	20141118_132717	N2	6158	0.108621	0.0792	0.5000	0.1
1970-01-01 00:04:09.093	20141118_132717	N2	6159	0.107892	0.0693	0.6000	0.1

249999 rows × 12 columns

```
time: 401 ms
```

Now that our index is of datetime object, we can use the resample function to get time intervals. With this function you can choose the time interval as well as how to downsample (mean, sum, *etc.*)

In [12]:
```
behav_resampled = behav.resample('10s', how=('mean'))
behav_resampled
```

Out[12]:

	frame	area	speed	angular_speed	aspect	midline	m
1970-01-01 00:00:00	158.970096	0.099870	0.172162	9.021929	0.271491	3.385725	0.
1970-01-01 00:00:10	362.347271	0.098067	0.166863	11.942732	0.319444	1.880583	0.
...	...	...	...	...	...	...	...
1970-01-01 00:04:00	5924.536608	0.097678	0.127150	5.646088	0.242850	1.785435	0.
1970-01-01 00:04:10	6041.902439	0.098643	0.255963	0.910815	0.088282	34.607500	0.

26 rows × 10 columns

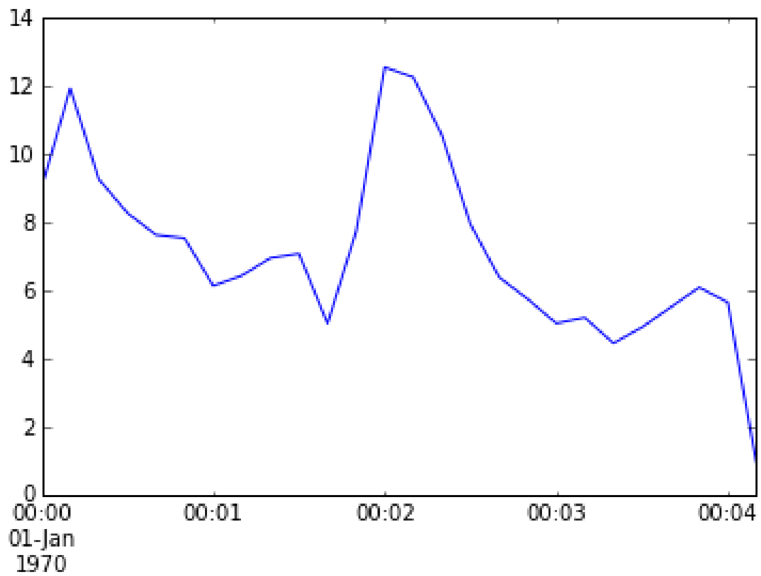
```
time: 101 ms
```

5. Quick and easy visualization

For more details, see: http://pandas.pydata.org/pandas-docs/version/0.15.0/visualization.html (http://pandas.pydata.org/pandas-docs/version/0.15.0/visualization.html)

In [13]:
```
behav_resampled['angular_speed'].plot()
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1075f5810>

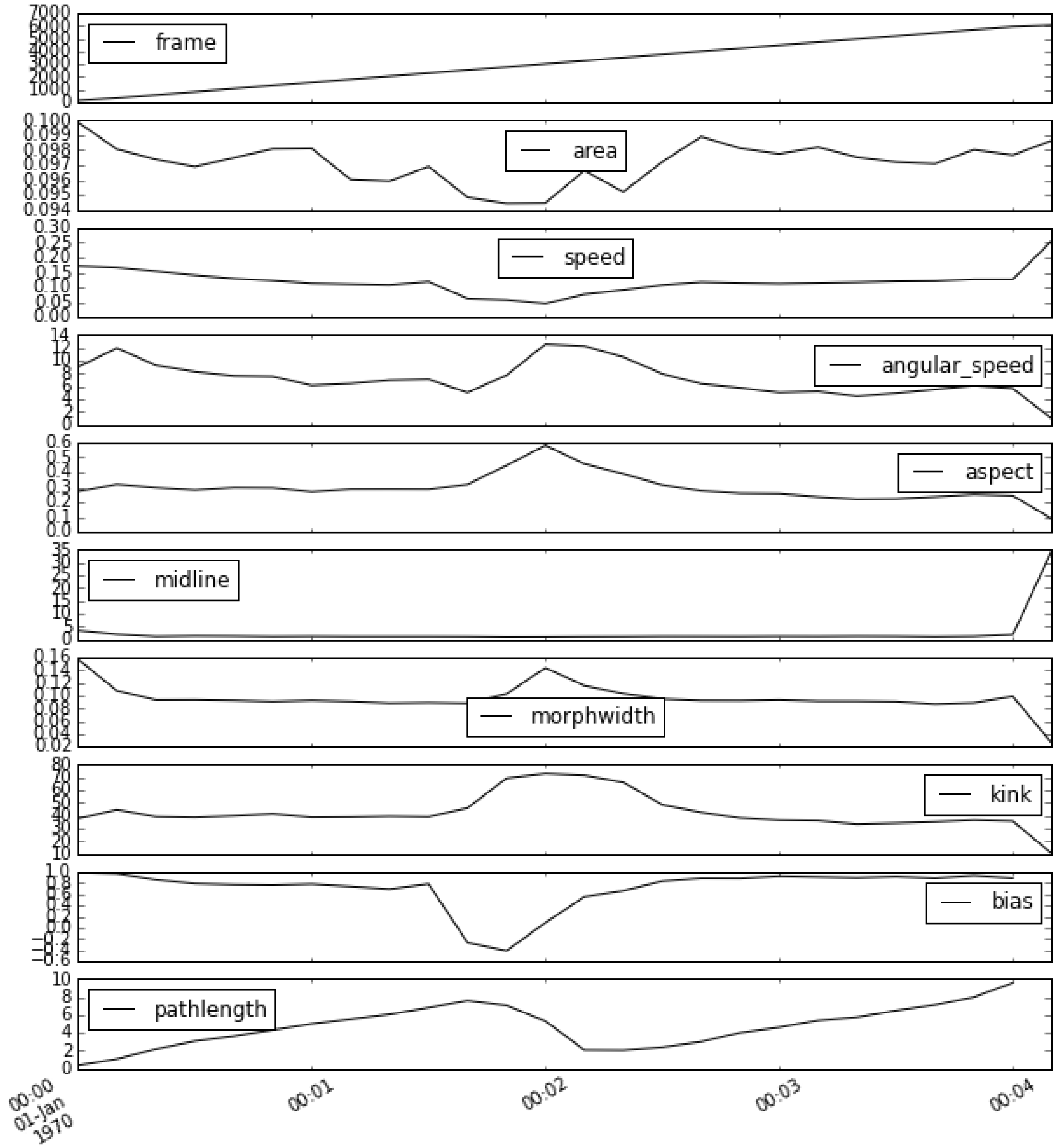


time: 183 ms

In [14]:

```
behav_resampled.plot(subplots=True, figsize = (10, 12))
```

```
Out[14]: array([<matplotlib.axes._subplots.AxesSubplot object at 0x114b3849
         0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x10769291
         0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x10771741
         0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x1080038d
         0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x109e0685
         0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x10803b75
         0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x10a2aa69
         0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x10a33021
         0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x10a389e1
         0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x10a40ccd
         0>], dtype=object)
```

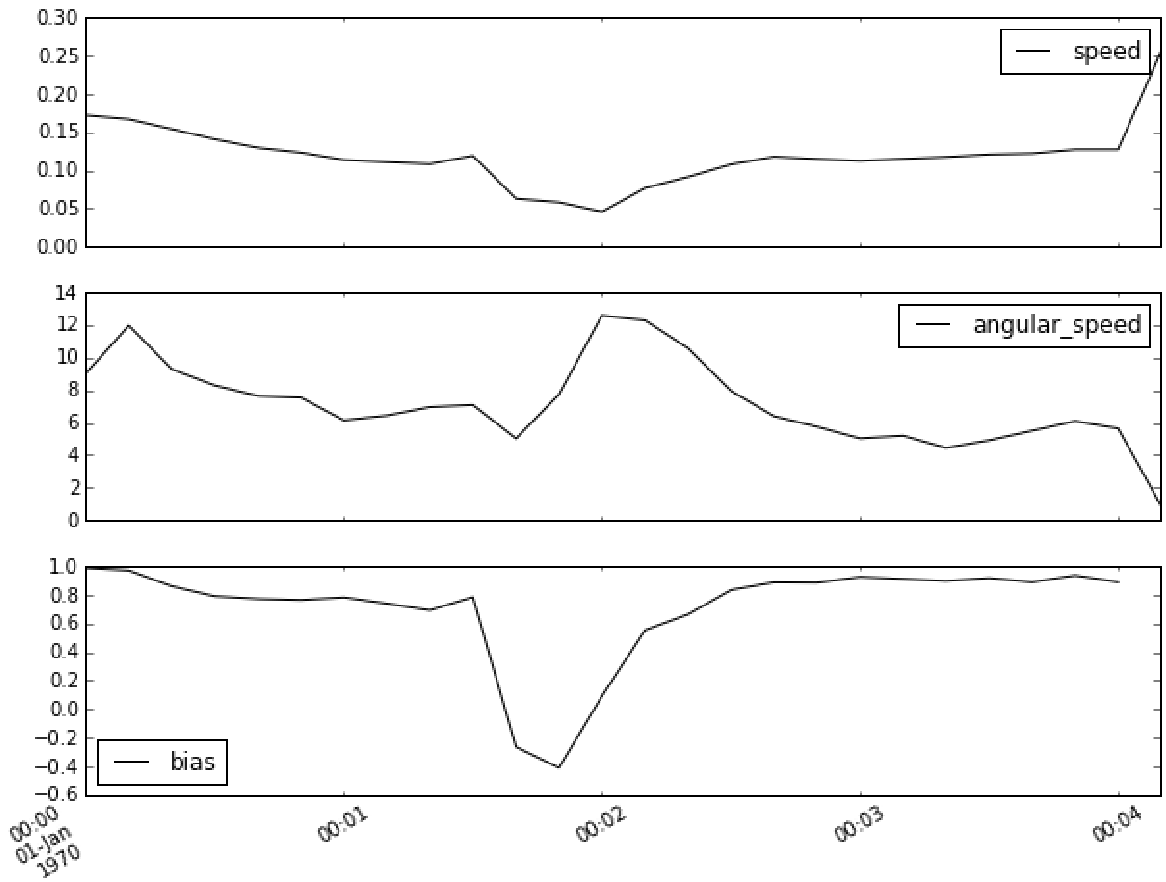


frame
area
speed
angular_speed
aspect
midline
morphwidth
kink
bias
pathlength
00:00
01-Jan
1970
00:01
00:02
00:03
00:04

time: 1.52 s

In [15]:
```
behav_resampled[['speed', 'angular_speed', 'bias']].plot(subplots =
True, figsize = (10,8))
```

Out[15]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x10aea81d
0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x10c0c7bd
0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x10ce3505
0>], dtype=object)
```




```
time: 533 ms
```

Summary

Pandas is a extremely useful and efficient tool for scientists, or anyone who needs to wrangle, analyze and visualize data!

Pandas is particularly attractive to scientists with minimal programming experience because:

- Strong, welcoming and growing community
- It is readable
- Idiom matches intuition

To learn more about Pandas see:

- Pandas Documentation (http://pandas.pydata.org/)
- ipython notebook tutorial (http://nsoontie.github.io/2015-03-05-ubc/novice/python/Pandas-Lesson.html) by Nancy Soontiens (Software Carpentry)
- Video tutorial (https://www.youtube.com/watch?v=0CFFTJUZ2dc&list=PLYx7XA2nY5Gcpabmu61kKcToLz0FapmHu&index=12) from SciPy 2015 by Jonathan Rocher
- History of Pandas (https://www.youtube.com/watch?v=kHdkFyGCxiY) by Wes McKinney