SUMMARY OF MY SLEEP DATA

ARUN DEBRAY
SEPTEMBER 23, 2014

"I want to default on my sleep debt!"

ABSTRACT. Since mid-June 2014, I have kept track of when I have gone to sleep, when I have woken up, and when I napped. This data is useful, interesting, and sometimes sadly amusing. I've prepared some basic statistics on the data, as well as some notes on how I made everything work.

This document, the data, and the programs I used to generate them can be found at http://stanford.edu/~adebray/Haskell/sleep/. Any questions, comments, or concerns may be directed to me, at adebray@stanford.edu.

1. Basic Statistics

During this project, I have recorded my sleep for **98** days, during which I slept a grand total of **792.90** hours. Somehow it didn't feel like quite that much.

See Figure 1 for when I went to sleep and woke up each night.

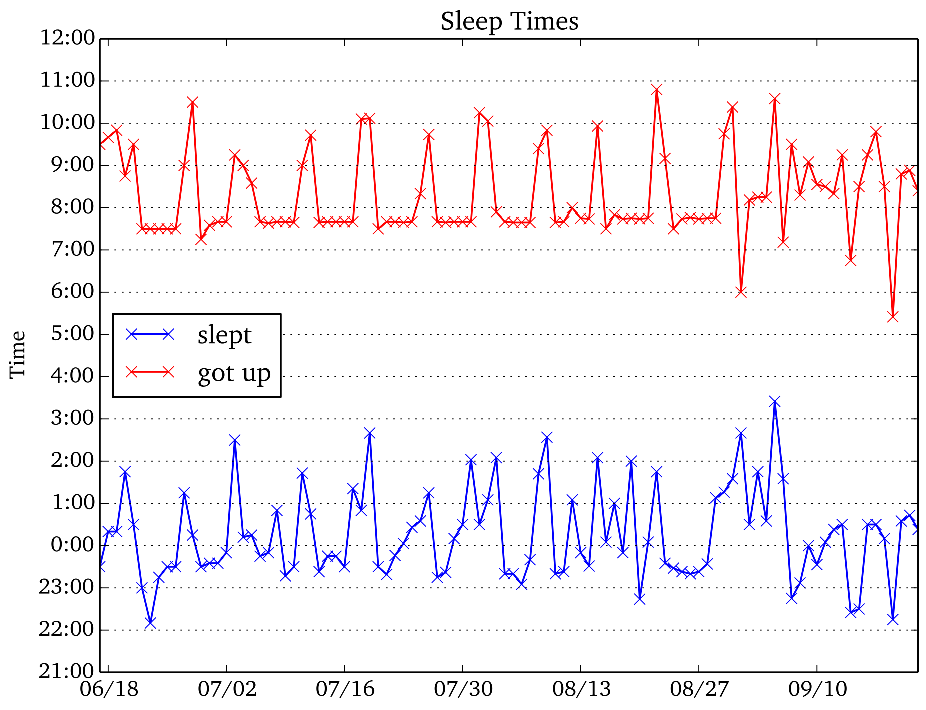

FIGURE 1. The most basic plot: when I went to sleep and when I woke up each day.

2. Averages

On average, I have gotten **8.09** hours of sleep per night. If naps are excluded, this is reduced to **8.06** hours per night.

The average has, of course, changed over time. In the last seven days, I've averaged **8.28** hours (**8.28** without naps) and in the last 30 days, I've averaged **8.16** hours (**8.09** without naps).

3. Standard Deviations

The standard deviation of my sleep has been **1.10** hours with naps and **1.06** hours without them. In the last seven days, the standard deviation was **0.61** hours (**0.61** hours without naps), and in the last 30 days, it was **1.40** hours (**1.40** hours without naps).

4. Per Day of the Week

In this section, I will analyze my sleep per day of the week (averages, standard deviations, etc). However, I have yet to do that… it's a work in progress. I do have graphs of waking and sleeping times in Figures 2 and 3, though.

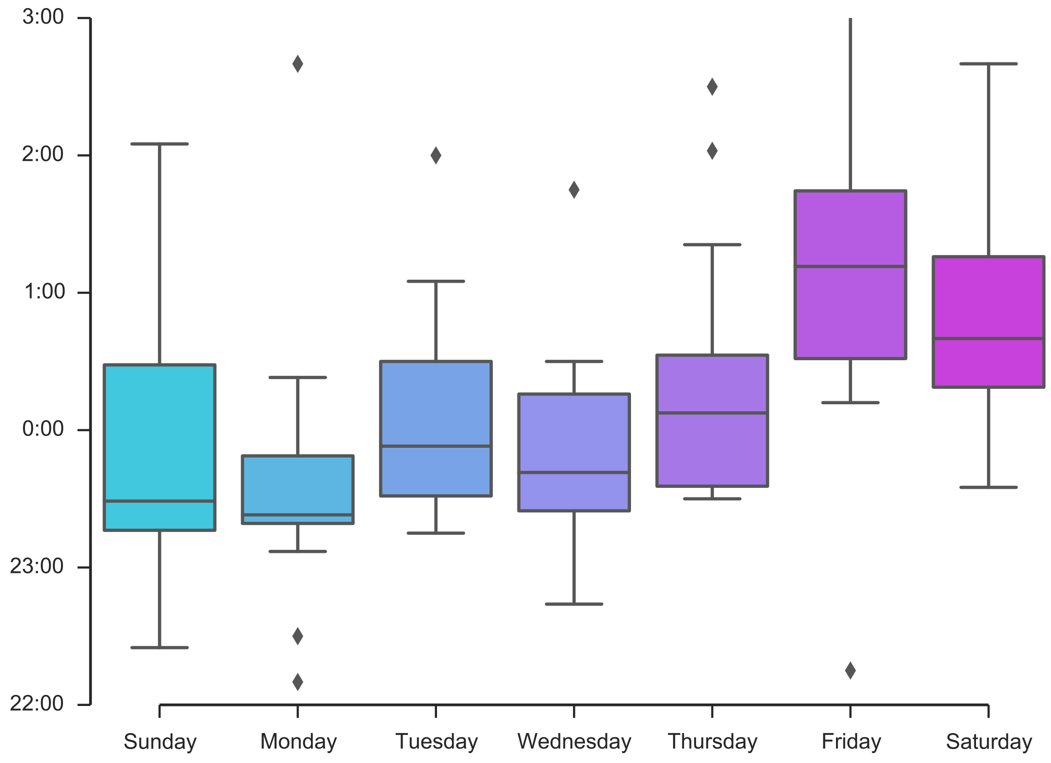

Figure 2. Box plots for when I went to sleep, broken down per day of the week. The boxes represent quartiles, so that each box contains 75% of the data of that day, and the whiskers contain the remaining 25%; the bar across the box represents the mean. Outlier values are represented by the diamonds.

5. Per Hour of the Day

Here I attempt to answer the question: how likely am I to be asleep at a given hour? See Figure 4 for the answer over the entire data collection period. A probability p means that on an arbitrary day, I am asleep at that time with probability p.

6. Some Source Code

Listing 1. Common notions for programs.

```
{-# LANGUAGE GeneralizedNewtypeDeriving #-} -- I like deriving Num

{- sleepTime.hs
   Arun Debray, June 29, 2014

   Common definitions for my sleep-data trackers, mostly data types.
 -}

module SleepTime (
    Minute (Minute),
```

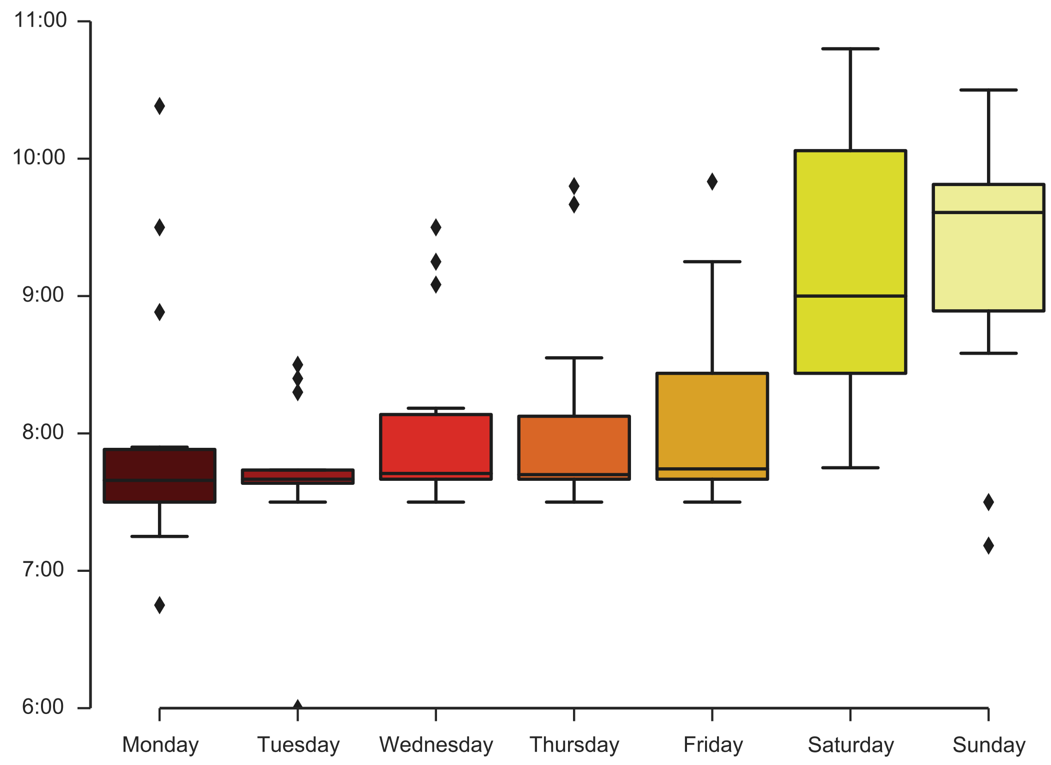

FIGURE 3. Much the same as Figure 2, this is a box plot of when I woke up over days of the week, with diamonds as outliers.

```
    Hour (Hour),
    Day (Day), -- a day is a number; a date is a DDMMYY combination
    Month (Month),
    Year (Year),
    Date (Date), day, month, year,
    Time (Time), hour, minute,
    Sleep (Sleep), rise, rest,
    Nap,
    DailyRecord (DailyRecord), today, bed, naps,
    readDataFile
) where

newtype Minute = Minute Int deriving (Read, Show, Ord, Eq, Num)
newtype Hour   = Hour Int deriving (Read, Show, Ord, Eq, Num)
newtype Day    = Day Int deriving (Read, Show, Ord, Eq, Num)
newtype Month  = Month Int deriving (Read, Show, Ord, Eq, Num)
newtype Year   = Year Int deriving (Read, Show, Ord, Eq, Num)

data Date = Date {
    day   :: Day,
    month :: Month,
    year  :: Year
} deriving (Read, Show, Eq)

-- time instance. Doesn't need to be more exact than this
data Time = Time {
    hour   :: Hour,
    minute :: Minute
} deriving (Read, Show, Eq)

instance Ord Date where
    d1 <= d2
        | year d1  < year d2  = True
        | month d1 < month d2 = True
        | day d1   <= day d2  = True
        | otherwise           = False

instance Ord Time where
    t1 <= t2
```

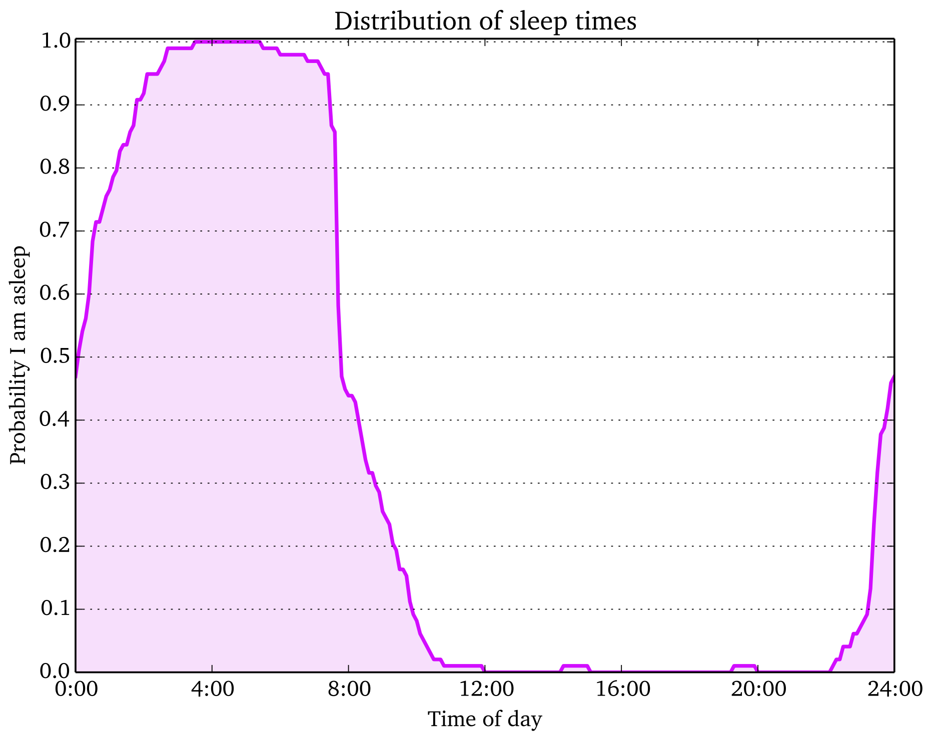

FIGURE 4. A plot of time versus how probable it is that I am asleep at a given time.

```
        | hour t1   < hour t2    = True
        | minute t1 <= minute t2 = True
        | otherwise              = False

data Sleep = Sleep { rise, rest :: Time } deriving (Read, Show)
type Nap = Sleep

data DailyRecord = DailyRecord {
    today :: Date,
    bed   :: Sleep,
    naps  :: [Nap]
} deriving (Read, Show)

-- Takes in the data file and produces everything it contains.
readDataFile :: FilePath -> IO [DailyRecord]
readDataFile filename = do
    -- will want to error handle
    contents <- readFile filename
    return $ map (\s -> (read s :: DailyRecord)) $ lines contents
```

LISTING 2. Used to record data.

```
{-  enter_data.hs
    Arun Debray, June 2014

    Command-line utility for entering sleep data into a file for later statstical analysis.
    Usage:
        ./enter_data [-f filename]
    Specify the file to place the data into; otherwise, uses the default, sleep_data.txt.
 -}

module Main where

import System.Environment (getArgs)
import System.IO (hFlush, stdout)

import SleepTime -- module for handling dates/times specified to this app

{- two small utilities for I/O.
   Might wrap them in their own module if they're useful for other programs.
 -}
putStr' :: String -> IO ()
```

```
putStr' s = do
    putStr s
    hFlush stdout

promptLine :: String -> IO String
promptLine prompt = do
    putStr' prompt
    getLine

-- actually writes to the file.
recordToFile :: FilePath -> DailyRecord -> IO ()
recordToFile filename record = appendFile filename $ (show record) ++ "\n"

queryDay :: IO Date
queryDay = do
    putStrLn "What date are you entering data for?"
    -- might be able to make this fancier
    dayStr <- promptLine "Day: "
    monthStr <- promptLine "Month: "
    yearStr <- promptLine "Year: "
    return $ Date {
        day   = Day (read dayStr :: Int),
        month = Month (read monthStr :: Int),
        year  = Year (read yearStr :: Int)
    }

{- One nice and easy fix would be for this to recognize strings of the form
   hh:mm and do something about that. Would make the program considerably
   cleaner.

   Also, until further notice, please specify all times in 24h.
 -}
queryTime :: String -> IO Time
queryTime kind = do
    putStrLn $ "When did you " ++ kind ++ "?"
    hourStr <- promptLine "Hour: "
    minuteStr <- promptLine "Minute: "
    return $ Time {
        minute = Minute (read minuteStr :: Int),
        hour   = Hour (read hourStr :: Int)
    }

-- for convenience
napMessage :: [Nap] -> String
napMessage partialList
    | null partialList  = "Did you take a nap (yes/no)? "
    | otherwise         = "Did you take another nap (yes/no)? "

getYesNo :: IO Bool
getYesNo = do
    userInput <- getLine
    case userInput of
        "yes" -> return True
        "Yes" -> return True
        "no"  -> return False
        "No"  -> return False
        _     -> do
            putStr' "Please answer 'yes' or 'no' > "
            getYesNo

-- loops to ask for naps from the user.
queryNaps :: [Nap] -> IO [Nap]
queryNaps partialList = do
    putStr' $ napMessage partialList

    nextAnswer <- getYesNo
    if nextAnswer
    then do
        start <- queryTime "sleep"
        finish <- queryTime "awake"
        let nextNap = Sleep {
            rest = start,
            rise = finish
        }
        return $ nextNap : partialList
    else return partialList

-- interactive loop
talkToUser :: IO DailyRecord
talkToUser = do
    date <- queryDay
    asleep <- queryTime "sleep"
    up <- queryTime "awake"
    napList <- queryNaps []
    return $ DailyRecord {
        today = date,
        bed = Sleep { rest = asleep, rise = up },
        naps = napList
    }

-- chooses the filename based on whether one was specified.
-- note: there is no error checking here...
getFilename :: [String] -> FilePath
getFilename args
    | length args < 2 = "sleep_data.txt"
    | otherwise       = args !! 1

main :: IO ()
main = do
    args <- getArgs
    record <- talkToUser
    recordToFile (getFilename args) record
```

LISTING 3. Used to generate statistics.

```
{- writeStatistics.hs

   Arun Debray, 22 June 2014

   This program reads the sleep data found in sleep_data.txt and generates statistics
   about them, which will be fed to the plotter and/or used directly by the final
   document.

   Ideas: maximum and minimum sleep time, and the date in question...
 -}

module Main where

import SleepTime
import System.IO

-- should factor elsewhere. (TODO)
-- is there a smarter way to write this...?
hourOf :: Time -> Int
hourOf t = case (hour t) of
    Hour h -> h

minuteOf :: Time -> Int
minuteOf t = case (minute t) of
    Minute m -> m

-- convert (hour, minute) -> number of hours, as a float
timeAsDouble :: Time -> Double
timeAsDouble t = (fromIntegral $ hourOf t) + ((fromIntegral $ minuteOf t) / 60)

-- calculates sleep time.
-- currently naively |b-a|. Perhaps this isn't ideal...
timeDifference :: Double -> Double -> Double
timeDifference awake asleep
    -- need to deal with 23 vs. 02 skewing data
    | asleep > 12 = 24 + awake - asleep
    | otherwise   = awake - asleep

-- since I generally don't nap at midnight, it's easier to have these separate functions
-- for napping.
napDifference :: Double -> Double -> Double
napDifference awake asleep = awake - asleep

napAsDouble :: Sleep -> Double
napAsDouble n = napDifference (timeAsDouble $ rise n) (timeAsDouble $ rest n)

-- convert Sleep type into its duration
sleepAsDouble :: Sleep -> Double
sleepAsDouble s = timeDifference (timeAsDouble $ rise s) (timeAsDouble $ rest s)

-- I ought to figure out how to round this or print it in rounded form.
asleepTime :: DailyRecord -> Double
asleepTime = sleepAsDouble . bed

-- guess this is a Daily Double!
asleepTimeWithNaps :: DailyRecord -> Double
asleepTimeWithNaps rec = sleepAsDouble (bed rec) + (sum $ map napAsDouble $ naps rec)

-- here's hoping this works on Doubles. Whoops
mean :: (Fractional a) => [a] -> a
mean xs = (sum xs) / (fromIntegral $ length xs)

stdDev :: (Floating a) => [a] -> a
stdDev xs = sqrt $ mean [(x - m) * (x - m) | x <- xs]
    where m = mean xs
```

```haskell
overallAverage :: [DailyRecord] -> Double
overallAverage = mean . (map asleepTimeWithNaps)

-- calculates the total average and trims to two decimal places.
overallNoNaps :: [DailyRecord] -> Double
overallNoNaps = mean . (map asleepTime)

overallStdDev :: [DailyRecord] -> Double
overallStdDev = stdDev . (map asleepTimeWithNaps)

stdDevNoNaps :: [DailyRecord] -> Double
stdDevNoNaps = stdDev . (map asleepTime)

totalHours :: [DailyRecord] -> Double
totalHours = sum . (map asleepTimeWithNaps)

recent :: Int -> [DailyRecord] -> Double
recent n = overallAverage . (take n) . reverse

recentNoNaps :: Int -> [DailyRecord] -> Double
recentNoNaps n = overallNoNaps . (take n) . reverse

recentSD :: Int -> [DailyRecord] -> Double
recentSD n = overallStdDev . (take n) . reverse

recentSDNoNaps :: Int -> [DailyRecord] -> Double
recentSDNoNaps n = stdDevNoNaps . (take n) . reverse

-- checks if the given time was between the two others.
-- I'll need to fix this if I ever sleep past noon... or get up before
-- midnight. It could happen.
timeBetween :: Double -> Sleep -> Bool
timeBetween t s
    | restTime > 12 && t > 12  = restTime <= t
    | restTime > 12 && t <= 12 = t < riseTime
    | otherwise                = restTime <= t && t < riseTime
    where restTime = timeAsDouble $ rest s
          riseTime = timeAsDouble $ rise s

-- since naps don't fall across midnight, this should be separated out.
-- I hope to make this cleaner someday, but for now this is what it shall be.
timeBetweenForNaps :: Double -> Sleep -> Bool
timeBetweenForNaps t s = (timeAsDouble $ rest s) <= t && t < (timeAsDouble $ rise s)

-- on a given night, was I asleep at the given time?
isAsleep :: Double -> DailyRecord -> Bool
isAsleep t rec = (timeBetween t $ bed rec) || any (timeBetweenForNaps t) (naps rec)

-- returns P(awake at time t), given records and time t
atTime :: Double -> [DailyRecord] -> Double
atTime t rec = (fromIntegral total) / (fromIntegral $ length rec)
    where total = length $ filter (isAsleep t) rec

-- produces list of moving quantities from a list of data
-- arguments: function to apply, window size, list
windowedStat :: ([a] -> a) -> Int -> [a] -> [a]
-- we need to build the windows
windowedStat f n xs = [f (window i) | i <- [1..length xs]]
    where window i = take n $ drop (i - 1 - n `div` 2) xs

-- calculates the moving averge. Arguments: window size, list
-- by a quick call to windowedStat
-- note that you can't pass in records to these functions, just numbers!
windowedMean :: (Fractional a) => Int -> [a] -> [a]
windowedMean = windowedStat mean

-- in the same vein, this calculates the moving standard deviation.
windowedStdDev :: (Floating a) => Int -> [a] -> [a]
windowedStdDev = windowedStat stdDev

-- Given a filename, writes to 'statistics/filename.txt'
-- In order to make things Python-readable, I don't want to write lists this way.
writeStatistic :: (Num a, Show a) => String -> a -> IO ()
writeStatistic filename stat = do
    let path = "statistics/" ++ filename ++ ".txt"
    writeFile path $ show stat

-- recenter going-to-sleep time at midnight (so 23.9 is just before 0.0)
centerFix :: Double -> Double
centerFix val
    | val <= 12 = val
    | otherwise = val - 24

-- processes awake and asleep times for a single record.
```

```haskell
putSingleTime :: Handle -> DailyRecord -> IO ()
putSingleTime h record = do
    let toSleep = timeAsDouble $ rest $ bed record
        wakeUp  = timeAsDouble $ rise $ bed record
    hPutStrLn h ((show $ centerFix toSleep) ++ "\t" ++ show wakeUp)

putSingleProb :: Handle -> [DailyRecord] -> Double -> IO ()
putSingleProb h records t = hPutStrLn h $ show $ atTime t records

-- puts raw awake/asleep data into raw form for Python to plot.
-- I think I should factor this out, which will require moving
-- some other functions into SleepTime.hs.
-- also, assumes that the file is ordered, which is true but not enforced anywhere...
putTimes :: FilePath -> [DailyRecord] -> IO ()
putTimes filename records = withFile filename WriteMode $ \h -> mapM_ (putSingleTime h) records

-- these should be refactored and/or prettified.
-- that is, I should kill putSingleTime/Prob and just map strings to lines of a file. That makes life
    simplest.
putProbs :: FilePath -> [DailyRecord] -> IO ()
putProbs filename records = withFile filename WriteMode $ \h -> mapM_ (putSingleProb h records) [x/10.0 |
     x <- [0..239]]

-- yeah, I want to refactor. Blah.
-- and then I want to include naps!
-- so many moving averages... this is weekly for now
putMovingAvgs :: FilePath -> [DailyRecord] -> IO ()
putMovingAvgs filename records = withFile filename WriteMode $ \h -> mapM_ (\x -> hPutStrLn h (show x)) $
     windowedMean 7 $ map asleepTime records

main :: IO ()
main = do
    records <- readDataFile "sleep_data.txt"
    writeStatistic "overallAverage" $ overallAverage records
    writeStatistic "overallNoNaps" $ overallNoNaps records
    writeStatistic "numDays" $ length records
    writeStatistic "totalHours" $ totalHours records
    writeStatistic "lastWeek" $ recent 7 records
    writeStatistic "weekNoNaps" $ recentNoNaps 7 records
    writeStatistic "lastMonth" $ recent 30 records
    writeStatistic "monthNoNaps" $ recentNoNaps 30 records
    writeStatistic "overallStdDev" $ overallStdDev records
    writeStatistic "stdDevNoNaps" $ stdDevNoNaps records
    writeStatistic "weekSD" $ recentSD 7 records
    writeStatistic "weekSDNoNaps" $ recentSDNoNaps 7 records
    writeStatistic "monthSD" $ recentSD 30 records
    writeStatistic "monthSDNoNaps" $ recentSD 30 records
    putTimes "raw_times.txt" records
    putProbs "raw_probs.txt" records
    putMovingAvgs "weekly_moving_avgs.txt" records
```

LISTING 4. Used to make plots.

```python
#!/usr/bin/env python3.4

# Arun Debray
# Started: 29 Jun 2014
# Updated: 10 Aug 2014

# The part of my project that makes pretty graphs.
# Uses matplotlib.

import argparse

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as dates
import matplotlib.axis as axis

# Setup: without arguments, makes all plots.
# With arguments, makes only the selected plots.
# Expand as necessary.
def handle_args():
    parser = argparse.ArgumentParser(description = 'generate plots from sleep data')
    # TODO: abstract these away into a function.
    parser.add_argument('--plot-times', dest = 'should_plot_times', action = 'store_true',
                        default = None, help = 'create the times plot in plots/raw_times.pdf')
    parser.add_argument('--plot-probs', dest = 'should_plot_probs', action = 'store_true',
                        default = None, help = 'create the probs. plot in plots/sleep_pobs.pdf')
    parser.add_argument('--plot-boxes', dest = 'should_plot_boxes', action = 'store_true',
                        default = None, help = 'create boxplots of sleep by week. plot in *_box.pdf')
    args_dict = parser.parse_args()
    return [args_dict.should_plot_times,
            args_dict.should_plot_probs,
            args_dict.should_plot_boxes]
```

```
# plot the times I slept and awoke
def plot_raw_times():
    print('Generating plot of times...')
    dbd = 735401 # offset of start date from 01-01-0001 UTC
    with open('raw_times.txt', 'r') as infile:

        time_data = [[float(s2) for s2 in s1.split('\t')] for s1 in infile]
        x = np.arange(dbd, dbd+len(time_data))

        _, ax = plt.subplots()
        fmt = dates.DateFormatter('%m/%d')
        ax.xaxis.set_major_formatter(fmt)

        ax.plot_date(x, [a[0] for a in time_data], fmt='bo', marker='x', label='slept', linestyle='-')
        ax.plot_date(x, [a[1] for a in time_data], color='r', marker='x', label='got up', linestyle='-')

        plt.legend(loc='center left')
        plt.title('Sleep Times')
        plt.ylabel('Time')

        plt.ylim(ymin = -3, ymax = 12) # may need to change this once the school year starts.
        plt.yticks(np.arange(-3,13), [str(n % 24) + ":00" for n in range(-3,13)])
        plt.grid(b='on', which='major', axis='y', linestyle=':')

        # may change to eps for file-size stuff later.
        plt.savefig('plots/raw_times.pdf', format='pdf')

# plot the probability that I am awake at a given time
# this would be interesting in the last 7 or 30 days.
def plot_raw_probs():
    print('Generating plot of probabilities...')
    with open('raw_probs.txt', 'r') as infile:
        probs_vector = [float(line) for line in infile]
        x = np.arange(0.0, 24.1, 0.1)
        probs_vector.append(probs_vector[0])

        plt.plot(x, probs_vector, color = '#D20DFF', linewidth = 2)
        plt.fill_between(x, probs_vector, alpha = 0.5, color = '#EFC0FA')

        plt.xlim(xmin = 0, xmax = 24)
        plt.xticks(np.arange(0, 24.1, 4), ['%d:00' % n for n in [0, 4, 8, 12, 16, 20, 24]])
        plt.xlabel('Time of day')

        plt.ylim(ymin = 0, ymax = 1.005) # dat font doe
        plt.yticks(np.arange(0.0, 1.01, 0.1), ['%.1f' % n for n in np.arange(0.0, 1.01, 0.1)])
        plt.ylabel('Probability I am asleep')

        plt.title('Distribution of sleep times')

        # This only comes up if plot_raw_times is suppressed
        # Still generates a warning... hopefully, I'll fix that.
        if plt.legend() is not None:
            plt.legend().set_visible(False)

        plt.savefig('plots/sleep_probs.pdf', format='pdf')


def get_sleep_times():
    with open('raw_times.txt', 'r') as f:
        return [float(line.split()[0]) for line in f]

def get_wake_times():
    with open('raw_times.txt', 'r') as f:
        return [float(line.split()[1]) for line in f]

# cycles the list so that Monday starts the awake work week, and
# Sunday the asleep work week.
# basically, restarts the cycle with arr[offset]
def rearrange(arr, offset):
    return arr[offset:] + arr[:offset]

# probably will add an optional colorscheme argument...
# and prettfiy the fonts on the y-axis.

# offset is how different the second plot is.
def boxplot_data(fn, fname, ymin, ymax, offset = 0, cmap = 'muted'):
    # this is a little hacky: I just wanted to combine the two Python programs I had, but
    # without messing with the styles. I can unify/prettify everything another time.
    import seaborn as sns

    times = fn()
    # sort by day of the week
    organized_data = [[x for j, x in enumerate(times) if j % 7 == (i + 5) % 7] for i in range(7)]
    # and then the plotting

    sns.set(style = 'ticks')
    f, ax = plt.subplots()
#   sns.offset_spines()
    sns.boxplot(organized_data, fliersize = 6, names = rearrange(['Sunday', 'Monday', 'Tuesday',
            'Wednesday', 'Thursday', 'Friday', 'Saturday'], offset), color = cmap)
    plt.ylim(ymin, ymax)
    plt.yticks(np.arange(ymin, ymax + 1), ['%d:00' % (n % 24) for n in np.arange(ymin, ymax + 1)])
    sns.despine(trim=True)

    plt.savefig('plots/' + fname)

def plot_boxes():
    print('Generating weekly boxplot breakdown...')
    boxplot_data(get_sleep_times, 'asleep_box.pdf', ymin = -2, ymax = 3, cmap = 'cool')
    boxplot_data(get_wake_times, 'awake_box.pdf', ymin = 6, ymax = 11, offset = 1, cmap = 'hot')

# This isn't yet part of the program, but is experimental testing cool stuff. hehehehe
# I promise there's no evil plotting going on here. No sir.
def window_plotting():
    print('Generating moving averags plot') # will be more general later
    with open('weekly_moving_avgs.txt') as f:
        data = [float(line) for line in f]

    dbd = 735401
    _, ax = plt.subplots()
    fmt = dates.DateFormatter('%m/%d')
    ax.xaxis.set_major_formatter(fmt)

    xs = np.arange(dbd, dbd+len(data))
    # note to self: make this look pretty someday.
    ax.plot_date(xs, data, color='r', marker='x', linestyle='-')
    plt.savefig('plots/weekly_moving_averages.pdf')

# A histogram of when I fell asleep.
# Not currently being used. I should do something with it.
def asleep_histogram():
    with open('raw_times.txt', 'r') as infile:
        asleep_data = [float(line.split('\t')[0]) for line in infile]

        plt.hist(asleep_data, color='#3FA5FF')

        plt.xlim(xmin = -2, xmax = 3) # will almost certainly need to change. q_q
        plt.xticks(np.arange(-2, 3), ['%d:00' % (n % 24) for n in np.arange(22, 28)])

        plt.savefig('plots/asleep_histogram.pdf', format='pdf')

def main():
    #window_plotting() # TODO
    #return

    flags = handle_args()
    # update as necessary
    to_plot = [plot_raw_times, plot_raw_probs, plot_boxes]
    if any(flags):
        _ = [plotfn() for flag, plotfn in zip(flags, to_plot) if flag]
    else: # no flags specified, do everything
        _ = [plotfn() for plotfn in to_plot]
#   asleep_histogram()

if __name__ == '__main__':
    main()
```