date; find . | grep csv | while read full_path; do cat $full_path | awk -F,'$5 == "1" {print

0}' >> all_failures_all_smart__column_count_varies.log; done; date

count the number of columns in the file by counting commas per line

while read p; do echo $p | tr -cd , | wc -c; done <

all_failures_all_smart__column_count_varies.log Because the columns are inconsistent, check which columns are aligned date; find . | grep csv | while read full_path; do head -n1 $full_path >> $all_{column_headers}.log$; done; $date Now we can determine which columns are consistent for i in 1..130; do echo column$i; cat all_column_headers.log | cut -d',' -f$i | sort | uniq -c; done answer: columns 1 to 31 are present in all CSVs. Reduce failure list to first 31 columns cat all_failures_all_smart__column_count_varies.log | cut -d',' -f1-31 > all_failures_smart_columns1to31.log

Smart_9_raw is power-on hours

https://en.wikipedia.org/wiki/S.M.A.R.T.#Known_ATA_S.M.A.R.T._attributes
(https://en.wikipedia.org/wiki/S.M.A.R.T.#Known_ATA_S.M.A.R.T._attributes)

```
In [1]: import pandas
        print('pandas',pandas.__version__)
        import glob
        import pickle
        import numpy
        import time
        import random
        import matplotlib.pyplot as plt

        pandas 0.23.4
```

data from Backblaze

```
In [2]: df_header_only=pandas.read_csv('zipped_data/data_Q2_2018.zip_folder/2018-04-01.csv',
        nrows=3)
        #df_header_only.head()
```

```
In [3]: df = pandas.read_csv('data_synthesized_from_csvs/all_failures_smart_columns1to31.lo
        g',header=None)
        df.columns=df_header_only.columns[0:31]
        df.shape

Out[3]: (8743, 31)
```

Extrapolating failure rates from Backblaze has a few assumptions:

- drive technology
- drive vendor/model/batch
- does not account for time drive exists prior to production use

Given those caveats, we'll use the Backblaze failures as a sampling distribution

In [4]:
```python
max_hours=50000
max_days=max_hours//24
number_drives_failed_that_day, day_index, patches = plt.hist(df[df['smart_9_raw'] <
max_hours]['smart_9_raw'].dropna(how='any')/24,
                               bins=max_days,
                               facecolor='blue',
                               alpha=0.5)
plt.xlabel('power on days',fontsize=12)
plt.ylabel('count',fontsize=12);
plt.title('all drives',fontsize=14);
```
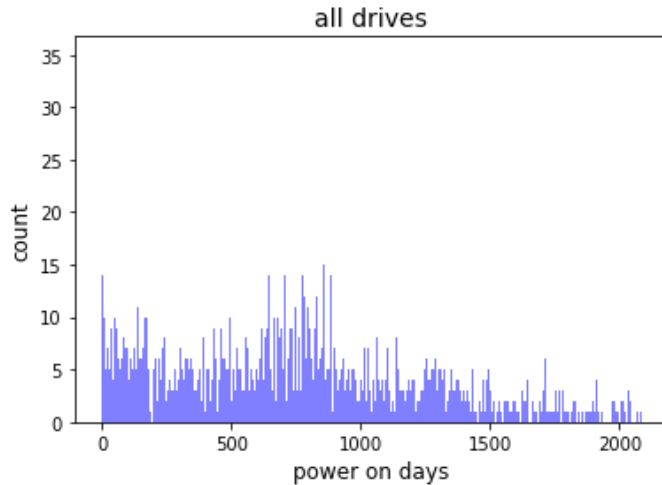

In [5]:
```python
def create_ary(number_of_drives,lifespan_in_days):
    ary = numpy.empty([number_of_drives,lifespan_in_days])
    ary[:] = -1

    start_time=time.time()
    # initial lifespan
    for drive_indx in range(number_of_drives):
        ary[drive_indx,0]=1 # every drive exists on day 1
        # https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.random.c
hoice.html
        drive_lifespan_in_days = int(numpy.random.choice(day_index[:-1],p=number_dri
ves_failed_that_day/sum(number_drives_failed_that_day)))
#         print(drive_lifespan_in_days)
#         print('drive lifespan:',drive_lifespan_in_days)
        ary[drive_indx,0:drive_lifespan_in_days]=1
#         print(ary[drive_indx,:])
    for drive_indx in range(number_of_drives):
        for day_indx in range(lifespan_in_days):
            while ary[drive_indx,day_indx]<0:
                drive_lifespan_in_days = int(numpy.random.choice(day_index[:-1],p=nu
mber_drives_failed_that_day/sum(number_drives_failed_that_day)))
                #print(drive_lifespan_in_days)
                #print('drive:',drive_indx)
                #print('day indx:',day_indx)
                #print('end = ',min([day_indx+drive_lifespan_in_days,lifespan_in_day
s]))

                #print(ary[drive_indx,:])
                ary[drive_indx,day_indx:min([day_indx+drive_lifespan_in_days,lifespa
n_in_days])] = ary[drive_indx,day_indx-1]+1
                #print(ary[drive_indx,:])
        #print("\n")
    print('elapsed:',time.time()-start_time,'seconds')
    return ary
```

```
In [6]: lifespan_in_days=365*10
        number_of_drives=1500

        ary = create_ary(number_of_drives,lifespan_in_days)

        print('initial:',sum(ary[:,0]))
        # by summing right column, we can get the cumulative number of drives
        print('final sum:',sum(ary[:,-1]))
```
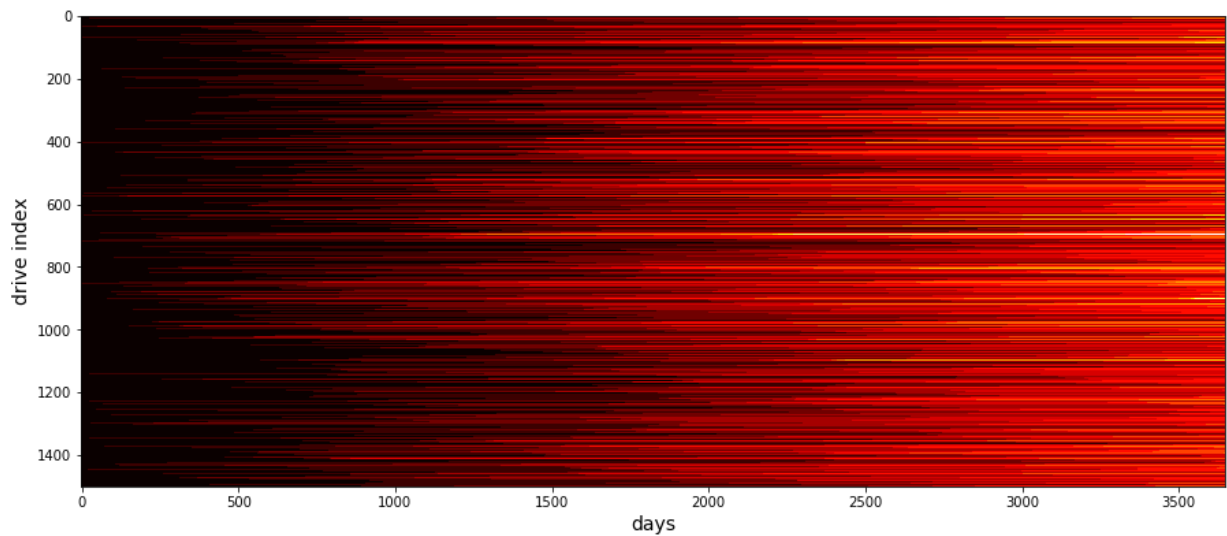
```
elapsed: 4.969458818435669 seconds
initial: 1500.0
final sum: 8396.0
```

heatmap image shows drive replacement over time for a single simulation

Time goes left-to-right: Left side is "initial status" and right side is "final status."

```
In [7]: plt.figure(figsize=(15,8))
        plt.imshow(ary, cmap='hot', interpolation='nearest');
        plt.xlabel('days',fontsize=14)
        plt.ylabel('drive index',fontsize=14);
```


Re-run this simulation to get a distribution of outcomes

```
In [8]: list_of_drive_counts = []
        for indx in range(100):
            ary = create_ary(number_of_drives,lifespan_in_days)
            list_of_drive_counts.append(sum(ary[:,-1]))
```

```
elapsed: 5.04577374458313 seconds
elapsed: 5.050820589065552 seconds
elapsed: 5.154571533203125 seconds
elapsed: 5.261159181594849 seconds
elapsed: 5.318852424621582 seconds
elapsed: 5.2335264682769775 seconds
elapsed: 5.222853183746338 seconds
elapsed: 5.116400718688965 seconds
elapsed: 5.156287431716919 seconds
elapsed: 5.237210750579834 seconds
elapsed: 5.252328634262085 seconds
elapsed: 5.211637735366821 seconds
elapsed: 5.237936496734619 seconds
elapsed: 5.267507791519165 seconds
elapsed: 5.27951717376709 seconds
elapsed: 5.208266258239746 seconds
elapsed: 5.252132177352905 seconds
elapsed: 5.230208396911621 seconds
elapsed: 5.198857069015503 seconds
elapsed: 5.138091802597046 seconds
elapsed: 5.146061420440674 seconds
elapsed: 5.130908250808716 seconds
elapsed: 5.12840461730957 seconds
elapsed: 5.163543701171875 seconds
elapsed: 5.162506580352783 seconds
elapsed: 5.08320426940918 seconds
elapsed: 5.118050813674927 seconds
elapsed: 5.143949508666992 seconds
elapsed: 5.119701147079468 seconds
elapsed: 5.107584476470947 seconds
elapsed: 5.150885820388794 seconds
elapsed: 5.041784763336182 seconds
elapsed: 5.116059064865112 seconds
elapsed: 5.119296073913574 seconds
elapsed: 5.151350259780884 seconds
elapsed: 5.139135360717773 seconds
elapsed: 5.0662925243377686 seconds
elapsed: 5.104346036911011 seconds
elapsed: 5.092221736907959 seconds
elapsed: 5.098390102386475 seconds
elapsed: 5.203756332397461 seconds
elapsed: 5.3006203174591064 seconds
elapsed: 5.233245372772217 seconds
elapsed: 5.104794502258301 seconds
elapsed: 5.0915868282318115 seconds
elapsed: 5.100254774093628 seconds
elapsed: 5.079478740692139 seconds
elapsed: 5.089666843414307 seconds
elapsed: 5.104024648666382 seconds
elapsed: 5.086277484893799 seconds
elapsed: 5.100026369094849 seconds
elapsed: 5.084925651550293 seconds
elapsed: 5.0372841358184814 seconds
elapsed: 5.128356695175171 seconds
elapsed: 5.106295824050903 seconds
elapsed: 5.085468292236328 seconds
elapsed: 5.137332916259766 seconds
elapsed: 5.1073901653289795 seconds
elapsed: 5.096286058425903 seconds
elapsed: 5.137896299362183 seconds
elapsed: 5.077430009841919 seconds
elapsed: 5.101404428482056 seconds
elapsed: 5.047858476638794 seconds
elapsed: 5.092929840087891 seconds
elapsed: 5.105214595794678 seconds
```

```
elapsed: 5.071438312530518 seconds
elapsed: 5.1497790813446045 seconds
elapsed: 5.096721172332764 seconds
elapsed: 5.086073637008667 seconds
elapsed: 5.103805780410767 seconds
elapsed: 5.103344917297363 seconds
elapsed: 5.227600336074829 seconds
elapsed: 5.125780344009399 seconds
elapsed: 5.157757759094238 seconds
elapsed: 5.122168064117432 seconds
elapsed: 5.088284492492676 seconds
elapsed: 5.098348379135132 seconds
elapsed: 5.1348631381988525 seconds
elapsed: 5.107399940490723 seconds
elapsed: 5.116827726364136 seconds
elapsed: 5.081140995025635 seconds
elapsed: 5.082913875579834 seconds
elapsed: 5.071341037750244 seconds
elapsed: 5.078421115875244 seconds
elapsed: 5.095585823059082 seconds
elapsed: 5.113112926483154 seconds
elapsed: 5.087881088256836 seconds
elapsed: 5.113469839096069 seconds
elapsed: 5.139415502548218 seconds
elapsed: 5.102813482284546 seconds
elapsed: 5.1270751953125 seconds
elapsed: 5.078030586242676 seconds
elapsed: 5.104695558547974 seconds
elapsed: 5.092835187911987 seconds
elapsed: 5.1148927211761475 seconds
elapsed: 5.126449346542358 seconds
elapsed: 5.089248895645142 seconds
elapsed: 5.007911920547485 seconds
elapsed: 5.1159257888793945 seconds
elapsed: 5.097278833389282 seconds
```

show the spread on number of drives

```
In [9]:  plt.hist(list_of_drive_counts,bins=10)
         plt.xlabel('total drive count')
         plt.ylabel('count');
```