Analysis of JBead Usage

Damian Brunold

2016-04-13

Introduction

JBead is an open source program for designing crocheted bead ropes. You can find information about the program at http://www.jbead.ch/.

Every time the program starts, it checks for updates by sending a short request to the jbead.ch webserver. These requests contain some information about the equipment of the users, for example the java version used, the brand, make and version of the operating system used etc. These requests get stored in the access log of the webserver.

I extracted three full years (2013 through 2015) of these data, cleaned them up a little bit and augmented them with some additional information about e.g. the location of the user as indicated by the IP address used.

It is worth to note, that these requests do not contain any personally identifiable data except perhaps for the IP address. In addition, the user can choose to disable these update checks and thus use the program without leaving any data trail behind.

As each request corresponds to a program start, the request can serve as a proxy for usage of the program. It is not entirely accurate, since some users might leave the program running for days and weeks on end and others might open it a dozen times in a few times, e.g. when searching through a set of pattern files. Nontheless, this information is the best usage data I have and it is quite interesting as is.

Analysis

Preparation

The data is provided in a single, tidy csv file. We load it, convert the timestamp to a datetime value and take a glimpse of the data.

```
## Observations: 389,671
## Variables: 21
## $ timestamp   <dttm> 2013-01-01 00:26:25, 2013-01-01 00:27:01, 2013-01...
## $ ip          <fctr> 81.164.246.147, 81.164.246.147, 81.164.246.147, 8...
## $ version     <fctr> 1.0.23, 1.0.23, 1.0.23, 1.0.23, 1.0.23, 1.0.23, 1...
## $ locale      <fctr> nl_BE, nl_BE, nl_BE, nl_BE, nl_BE, nl_BE, nl_BE, ...
## $ javaversion <fctr> 1.6.0_17, 1.6.0_17, 1.6.0_17, 1.6.0_17, 1.6.0_17,...
## $ os          <fctr> Windows XP, Windows XP, Windows XP, Windows XP, W...
## $ osversion   <fctr> 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 6.1,...
## $ arch        <fctr> x86, x86, x86, x86, x86, x86, x86, x86, x86, x86,...
## $ osmajor     <fctr> Windows, Windows, Windows, Windows, Windows, Wind...
## $ javamajor   <int> 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 6, 7, 7, 6, 6, 7, 6,...
## $ released    <date> 2009-11-04, 2009-11-04, 2009-11-04, 2009-11-04, 2...
## $ expired     <date> 2010-01-13, 2010-01-13, 2010-01-13, 2010-01-13, 2...
## $ timezone    <fctr> Europe/Brussels, Europe/Brussels, Europe/Brussels...
## $ lat         <dbl> 50.7667, 50.7667, 50.7667, 50.7667, 50.7667, 50.76...
## $ lon         <dbl> 4.3000, 4.3000, 4.3000, 4.3000, 4.3000, 4.3000, 4....
```

```
## $ zip         <fctr> 1650, 1650, 1650, 1650, 1650, 1650, 1650, 1650, 1...
## $ country     <fctr> Belgium, Belgium, Belgium, Belgium, Belgium, Belg...
## $ countryCode <fctr> BE, BE, BE, BE, BE, BE, BE, BE, BE, TR, BE, TR, T...
## $ city        <fctr> Beersel, Beersel, Beersel, Beersel, Beersel, Beer...
## $ regionName  <fctr> Flanders, Flanders, Flanders, Flanders, Flanders,...
## $ tz.offset   <dbl> 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 3, 3, 2, 2, 3, -4...
```

We have 20 variables and 389671 observations.

Let us create some tables to get a first impression of the data.

```
##
## 1.0.21 1.0.22 1.0.23 1.0.24 1.0.25 1.0.28 1.0.29
##   2216  16209 356022  14377    816     26      5
```

```
##
##    ppc    x64    x86
##    324  62796 326551
```

```
##
##   Linux     Mac Windows
##    1699   18229  369743
```

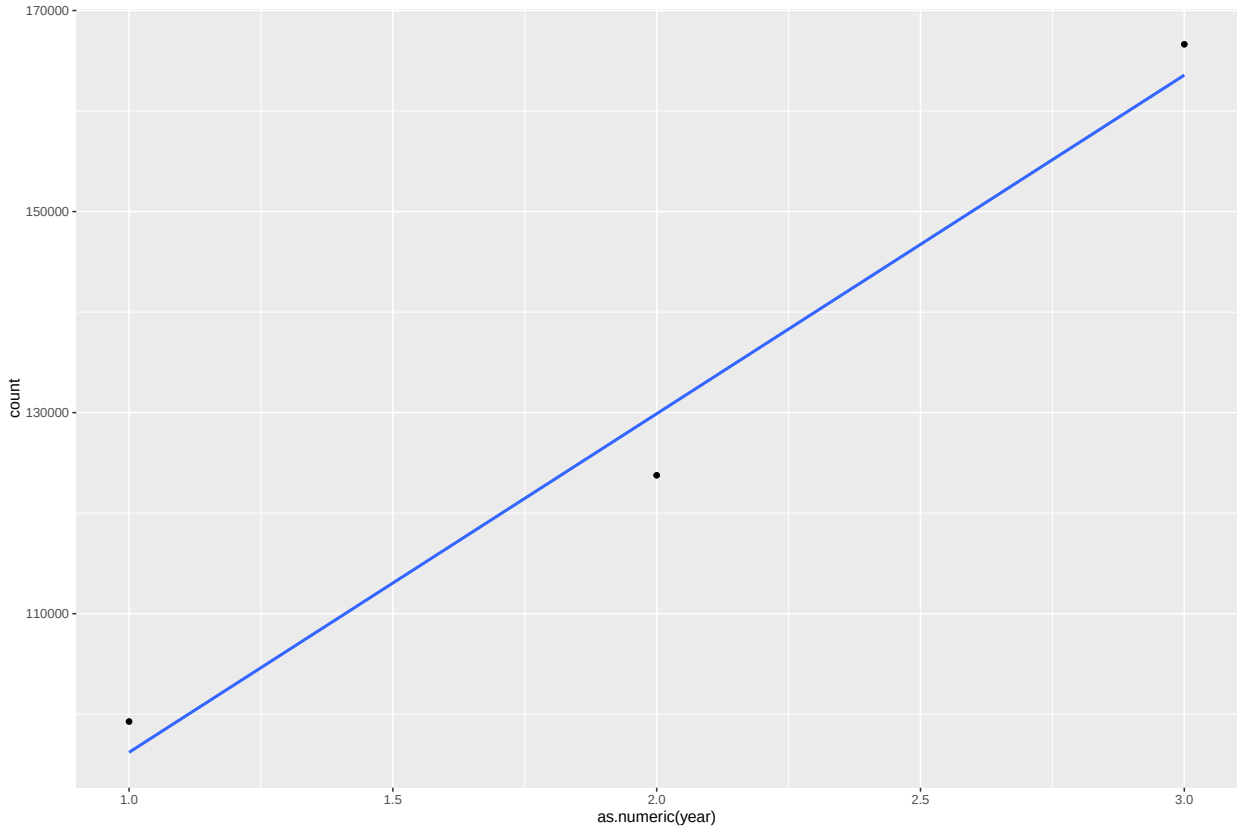
```
##
##      5      6      7      8
##    261  74925 206890 104944
```

```
##
##    AE    AG    AL    AM    AO    AR    AT    AU    AZ    BA    BB    BD
##    45     9    75   201     2   513  7650   980    35    34     9    12
##    BE    BG    BH    BM    BO    BR    BS    BW    BY    CA    CH    CI
##  6262  2213    26    12     4   889    13     4  4184  6146  2271     3
##    CL    CM    CN    CO    CR    CW    CY    CZ    DE    DK    DO    DZ
##   274     1   346   606    13    18   195 25007 51871   496   426    41
##    EC    EE    EG    ES    ET    FI    FR    GB    GE    GF    GH    GI
##    28  1410   272  6338    14   665 11623  5007   176     1     1     1
##    GL    GP    GR    GT    HK    HN    HR    HU    ID    IE    IL    IN
##     3    98  1196    83   257    66   626  2625    41   502  2546   212
##    IQ    IR    IS    IT    JM    JO    JP    KE    KG    KR    KW    KY
##    25   127    17  6350     7    49  1119     1    75    48     2     3
##    KZ    LA    LB    LC    LI    LT    LU    LV    LY    MA    MD    ME
##  1328     8   106     1     4  6802   334  2938     4   109  1183    70
##    MK    MN    MQ    MU    MX    MY    NC    NG    NI    NL    NO    NP
##    40    30    33     1  1390    48    68    57     5  2387   366    24
##    NZ    PA    PE    PF    PH    PK    PL    PR    PT    PY    QA    RE
##   218    21    93  2267   110    61 63935    38   327     2    19   252
##    RO    RS    RU    SA    SE    SG    SI    SK    SN    SV    SY    TG
##  4922   875 68029   180  2998   101   819  1061     1    24    26     2
##    TH    TJ    TM    TN    TR    TT    TW    TZ    UA    UG    US    UY
##    92    28   107    69  4515    22   160    14 39872    10 26763     5
##    UZ    VE    VN    VU    YE    YT    ZA    ZW
##   493   436    98     3    11     1   661     7
```

Application Usage Counts

In order to analyze usage counts, we add day and week columns. This allows us to group the data according to day and week. We also generate a year column in order to do facetting according to year.

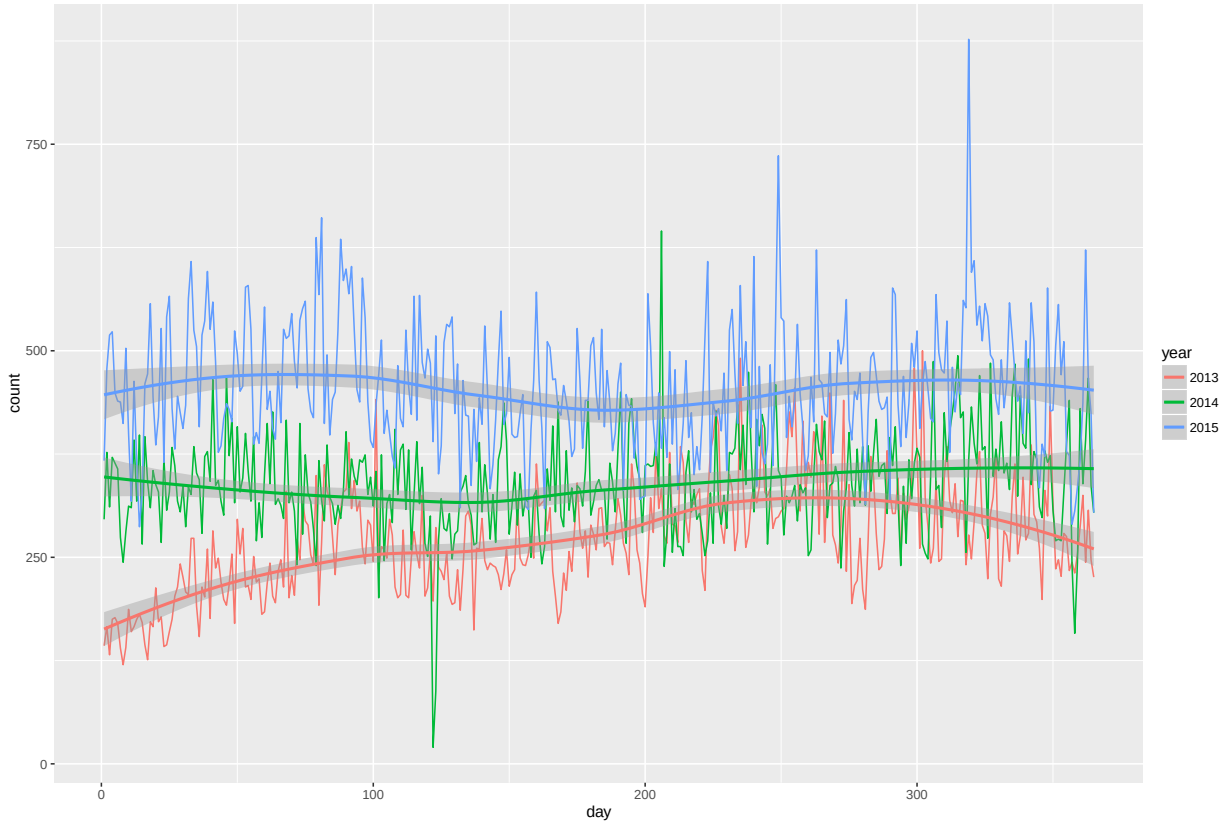
Now we generate a first overview of the three years.



We see that the usage of the application is clearly increasing across the three years.

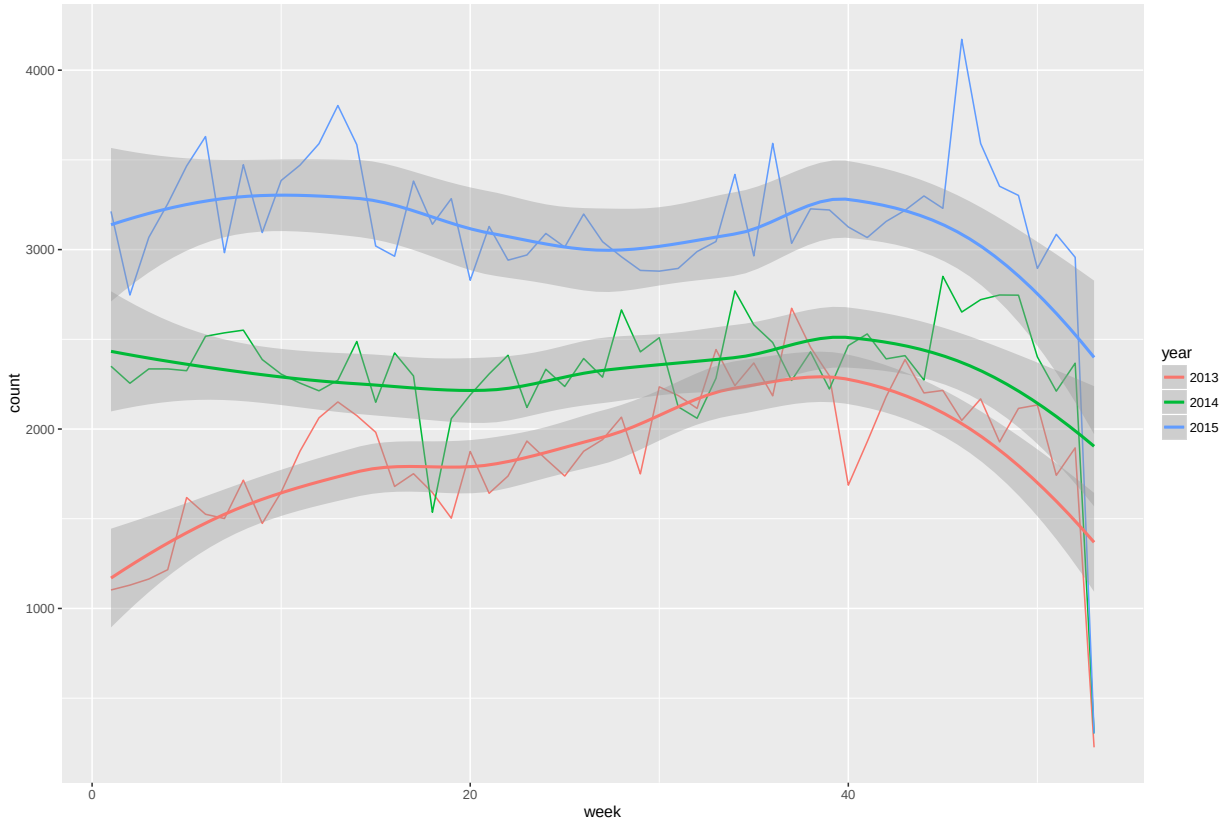
In the first year, the application was used 99268 times. In the second year it was 123765 and in the third year, it was 166638. So the year-over-year increase from 2013 to 2014 was +25% and the increase from 2014 to 2015 was +35%.

How about the time series of the three years?



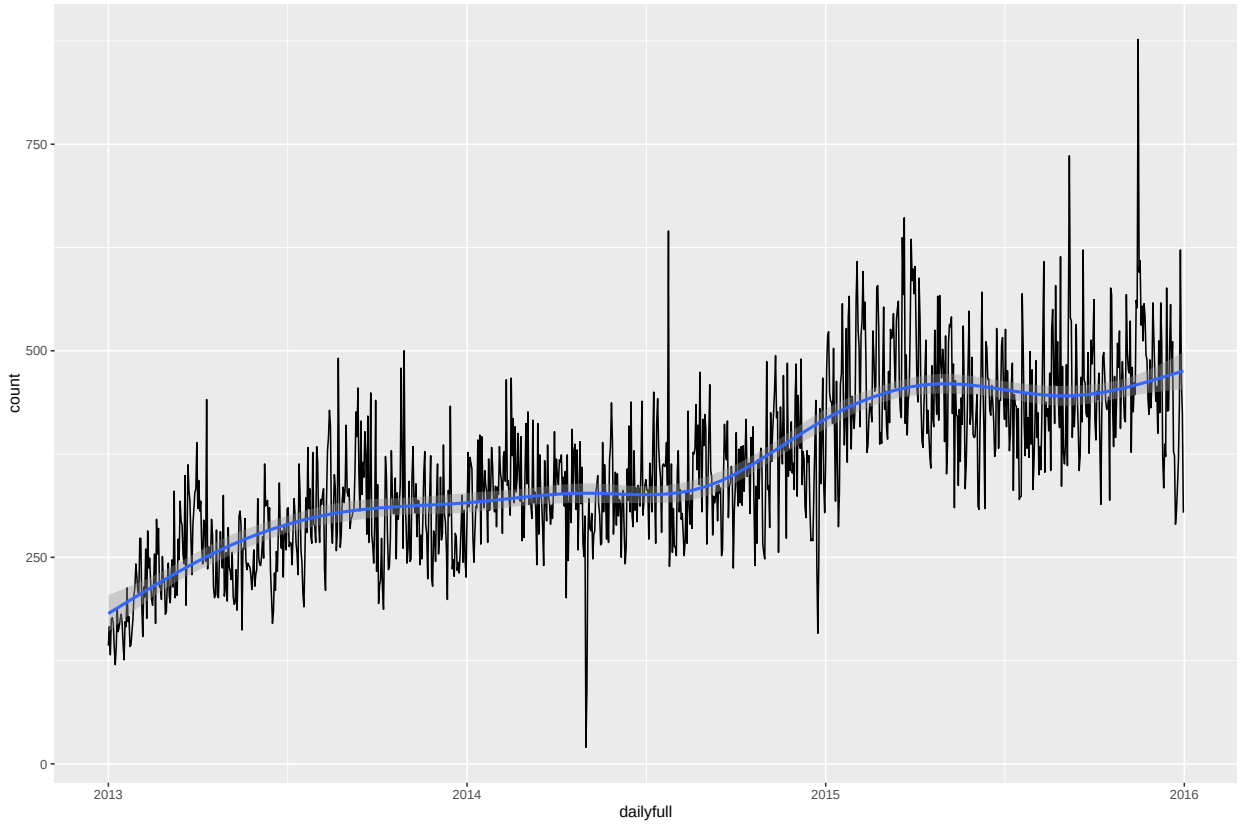
The average of daily counts for 2013 is 271.9671233, for 2014 339.0821918 and for 2015 456.5424658.

By looking at weekly groups instead of daily ones, we get a less noisy picture.



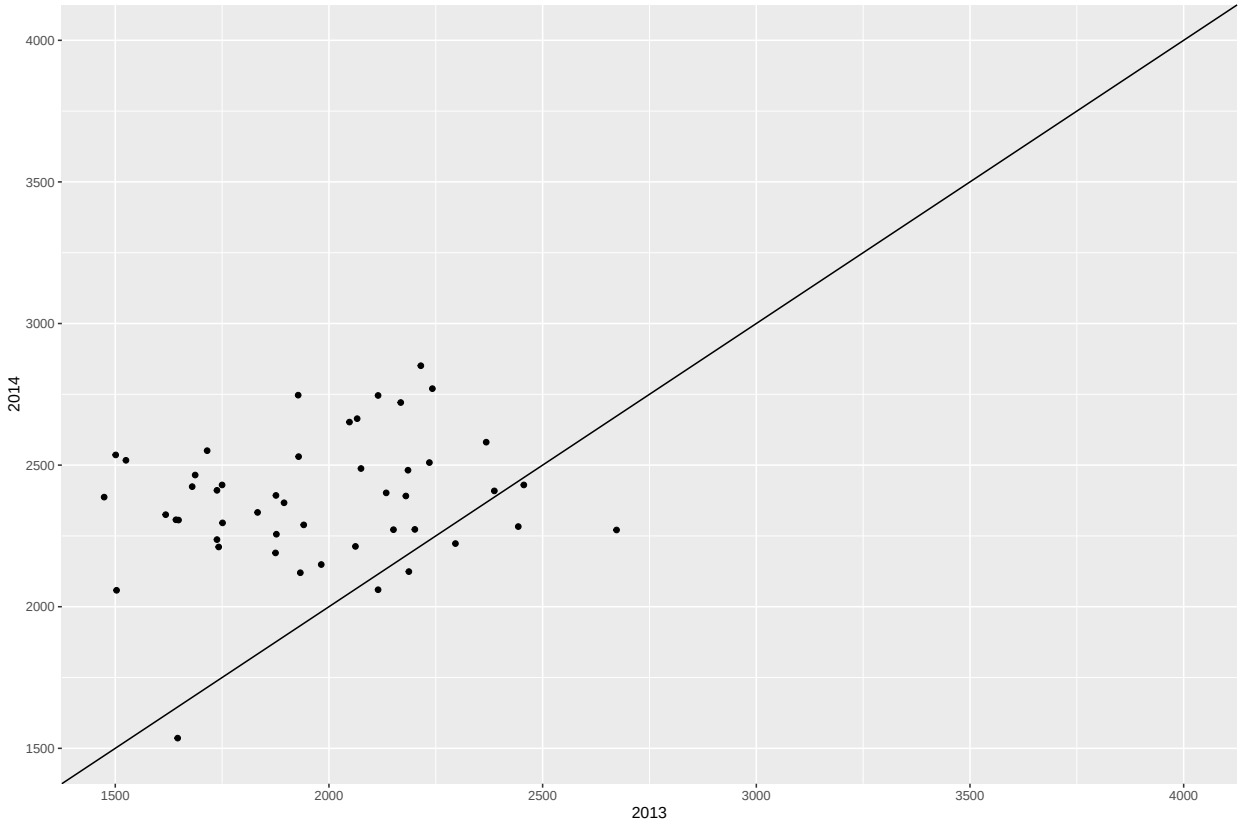
There seems to be a clear year to year increase in usage of the application.

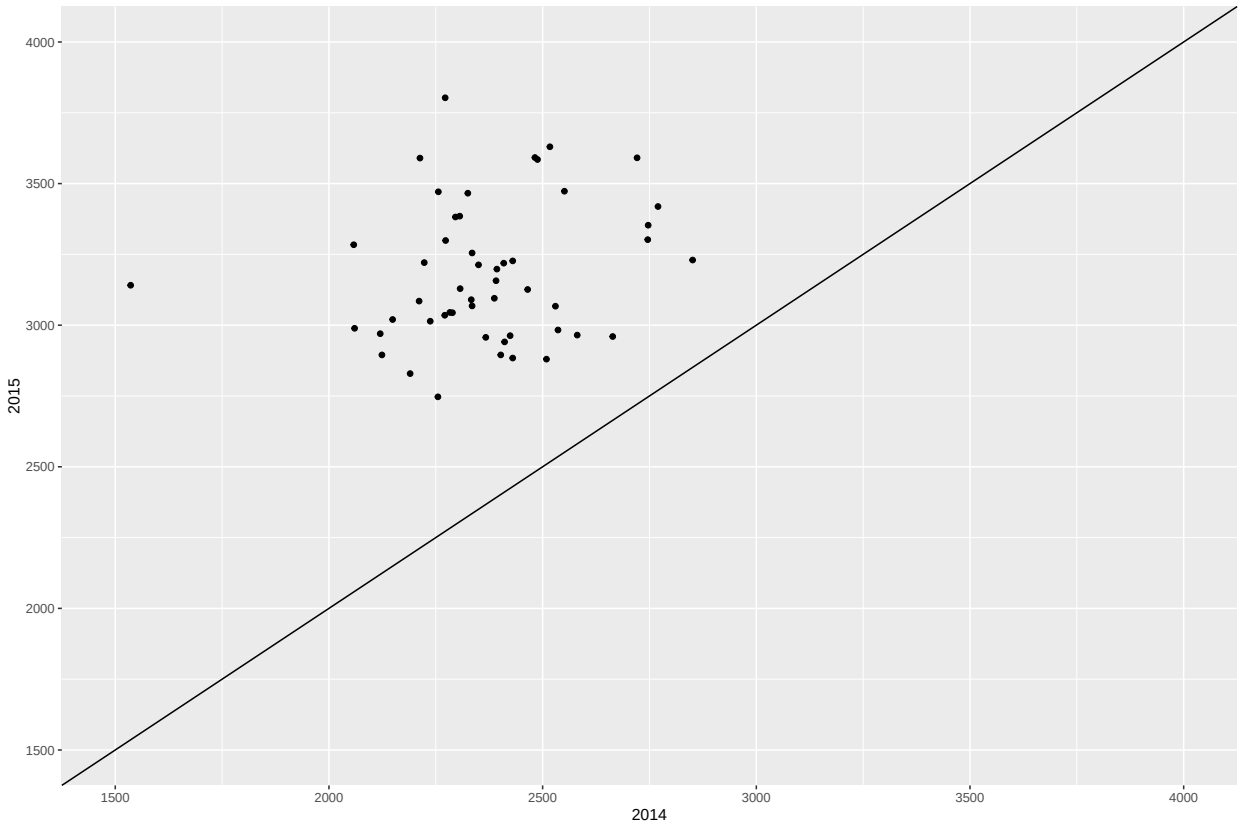
Finally, lets look at the full three year period.



The median daily count is 345.

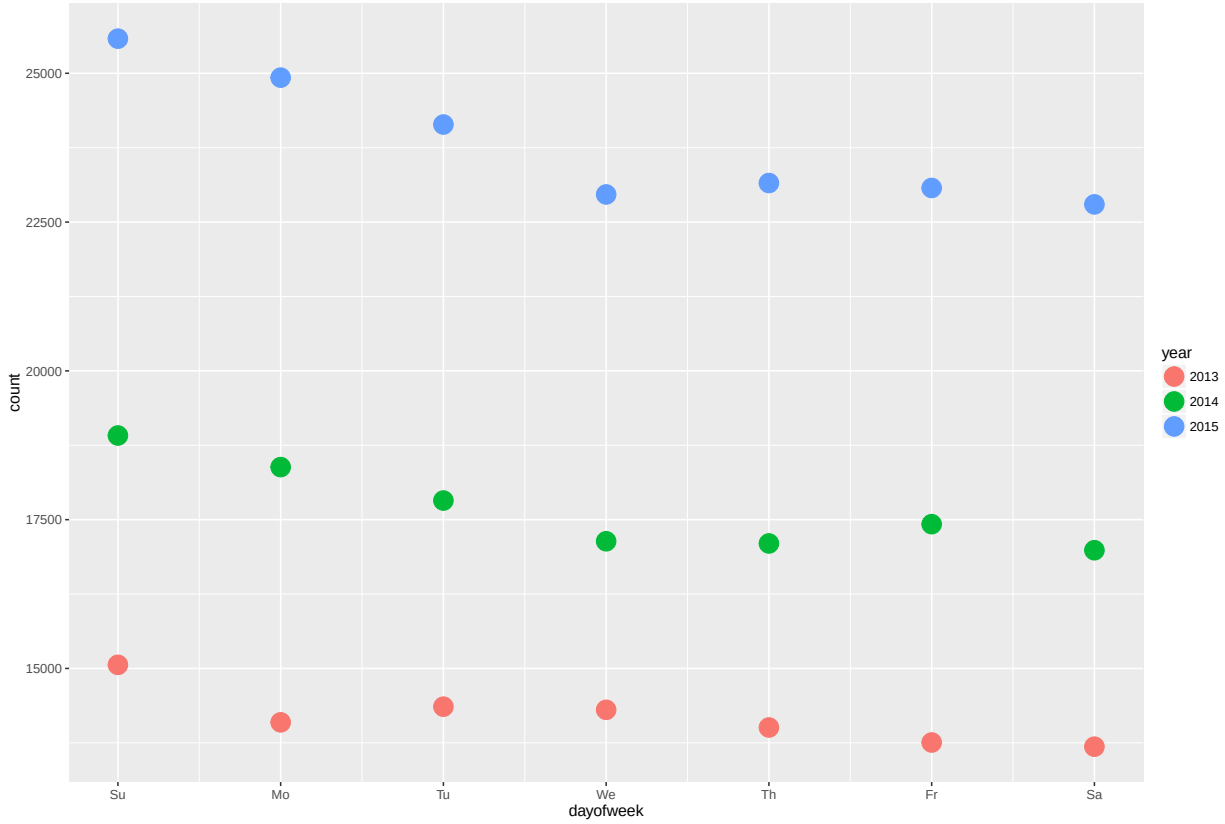
Lets do a scatter plot of weekly count from 2015 versus 2014 and so on.





Since the points are mostly above the line, we clearly see that the counts are increasing from year to year.

Can we find out how the distribution in usage is for different days of the week?



The application is most used on sundays.

```
##
##  Chi-squared test for given probabilities
##
## data:  subset(dayofweek, year == 2013, select = count)
## X-squared = 90.561, df = 6, p-value < 2.2e-16
```

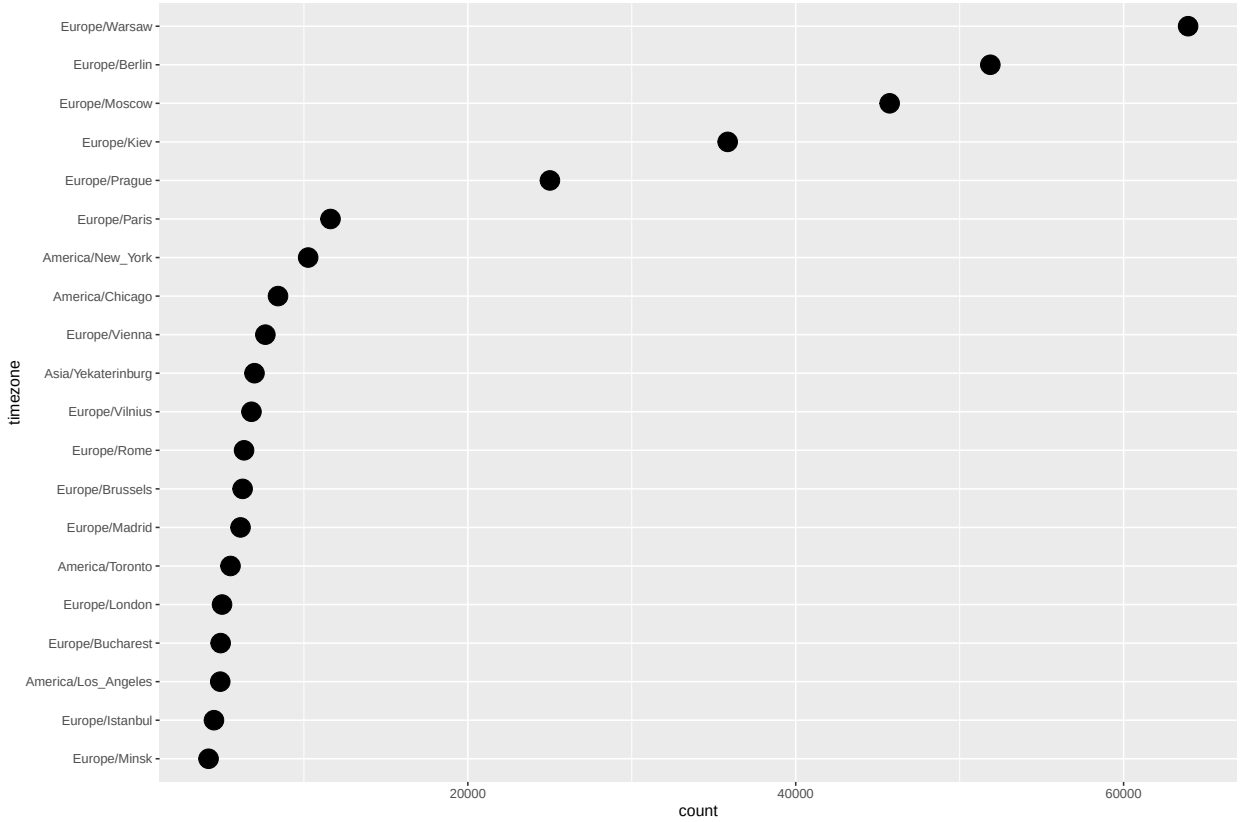
```
##
##  Chi-squared test for given probabilities
##
## data:  subset(dayofweek, year == 2014, select = count)
## X-squared = 181.85, df = 6, p-value < 2.2e-16
```

```
##
##  Chi-squared test for given probabilities
##
## data:  subset(dayofweek, year == 2015, select = count)
## X-squared = 302.81, df = 6, p-value < 2.2e-16
```

The chi-square test shows that there is a significant difference between the days of weeks.

This analysis is to be taken with a grain of salt, because the datetimes are in CET. So if a user in australia uses the application on sunday, it might be still saturday in europe. In order to account for this, it would be necessary to convert the datetimes into the local timezone.

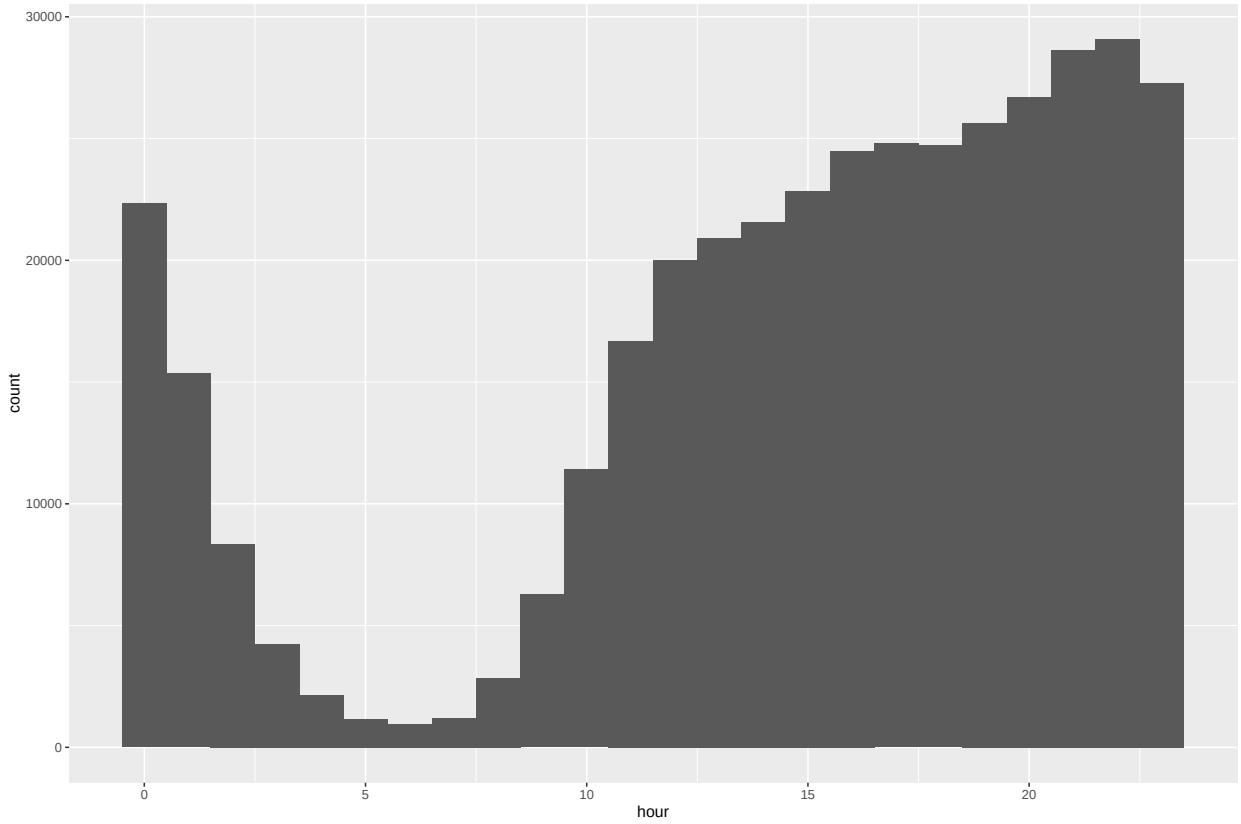
Let us have a look at the distribution accross timezones. We start with the most common 20 timezones.



The timezones vary widely: mostly european, but also american (east and west). So it clearly would be beneficial, if we adjusted the timestamp according to timezone.

As we have access to the timezone offset in the variable tz.offset, we can create a local timestamp (this does not take daylight saving into account).

Now let us see at which hours of the day users are using the application.

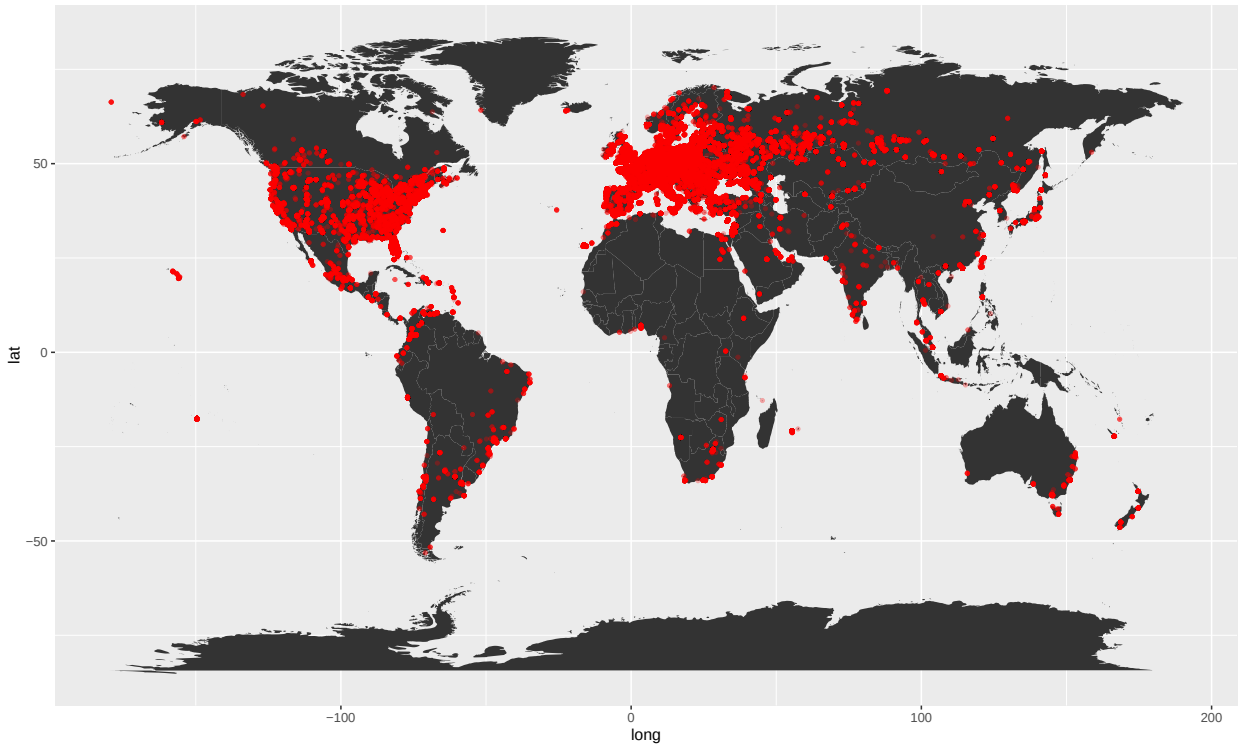


We clearly see that most people do not work with the application during the night. The peak is in the evening.

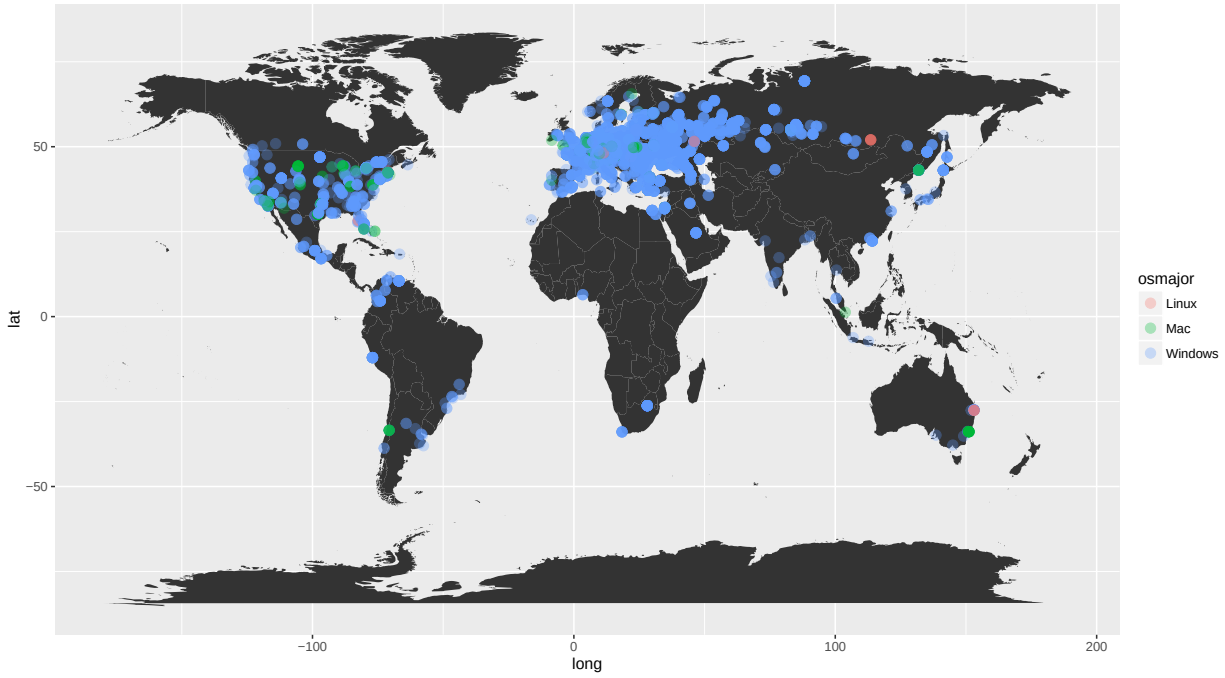
```
##
##  Chi-squared test for given probabilities
##
## data:  hourly$count
## X-squared = 147200, df = 23, p-value < 2.2e-16
```

The chi-square test does confirm that the hourly usage is not evenly distributed.

Next, I want to take a look at the distribution of the users on the world map.

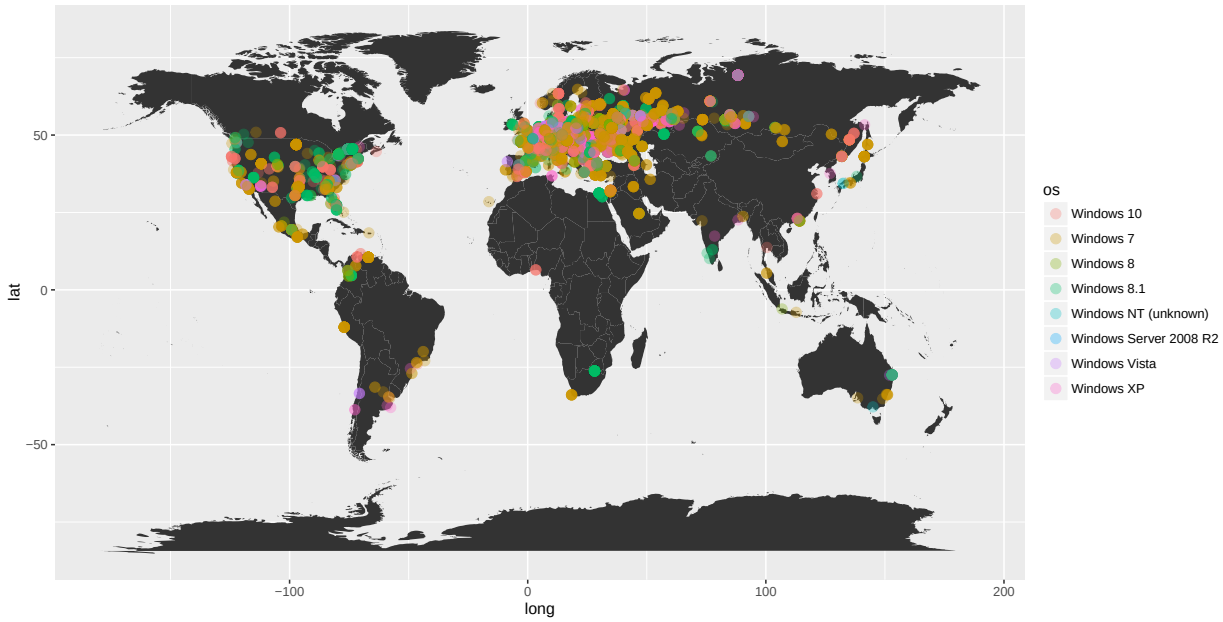


Lets differentiate the major os versions used. We take only 2015 for this.

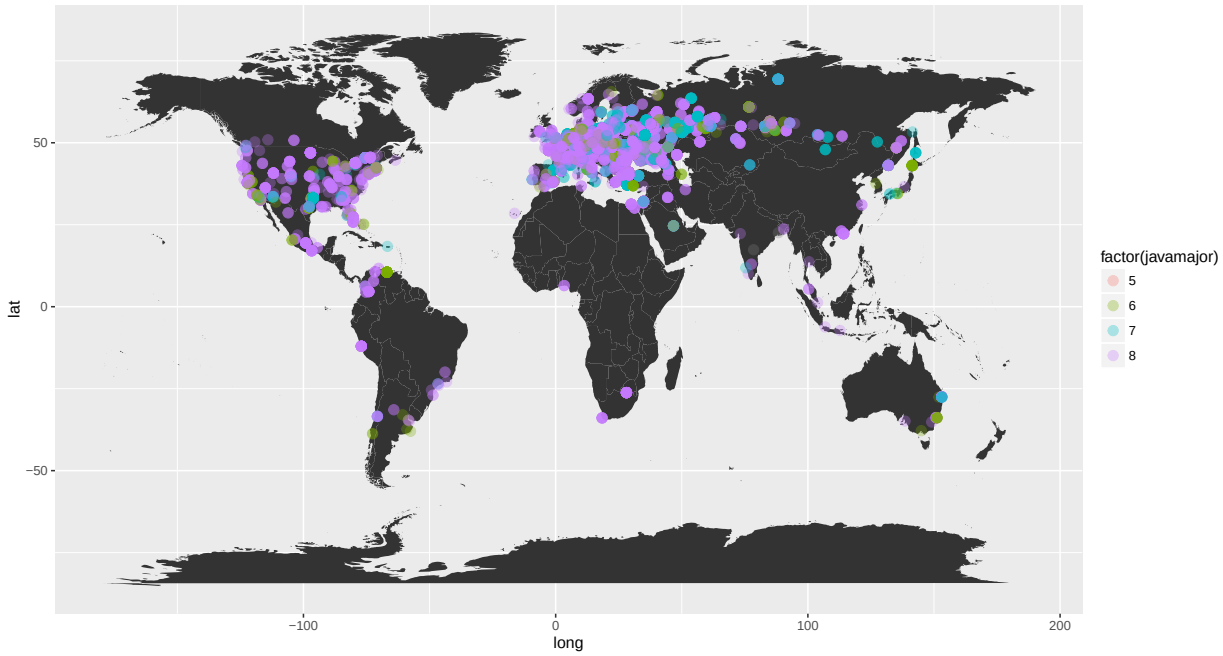


The mac seems to be mostly found in the USA. Linux is scarce but seemingly in the eastern part of europe and russia and in australia.

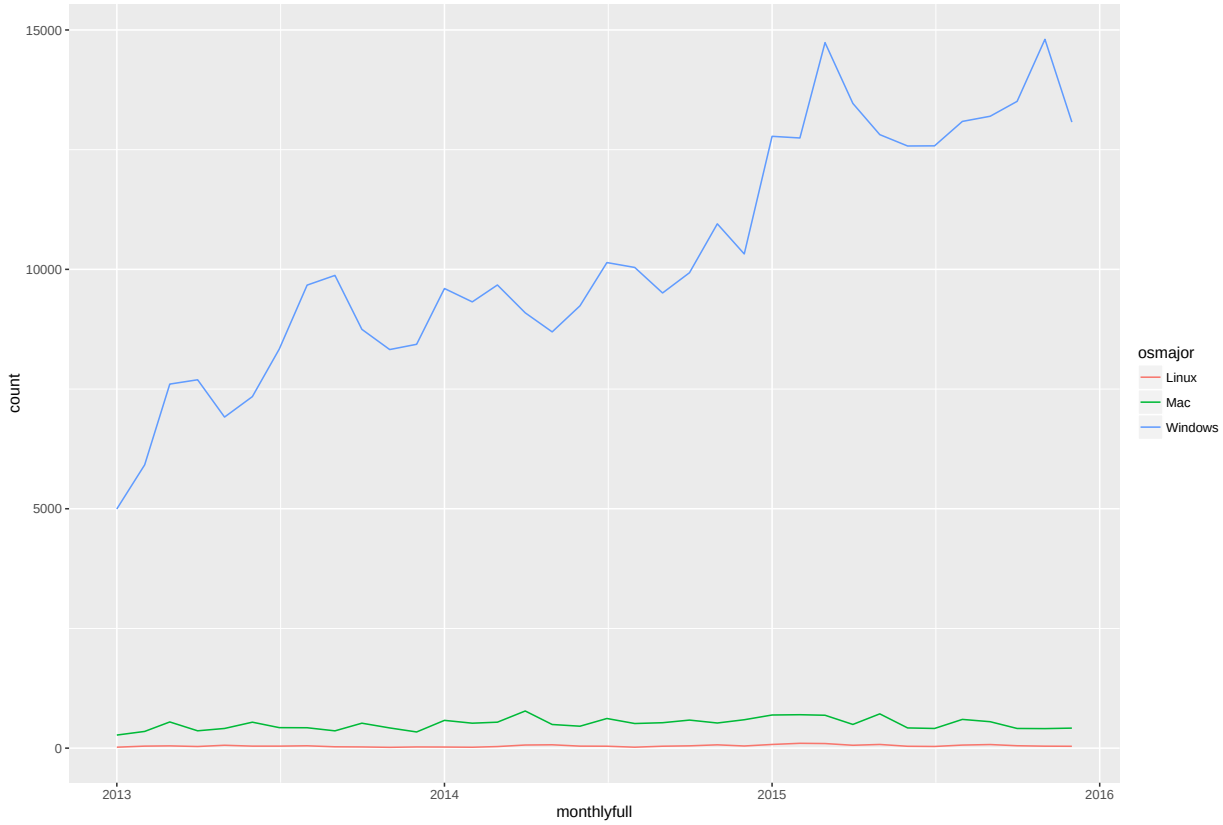
Only look at windows:



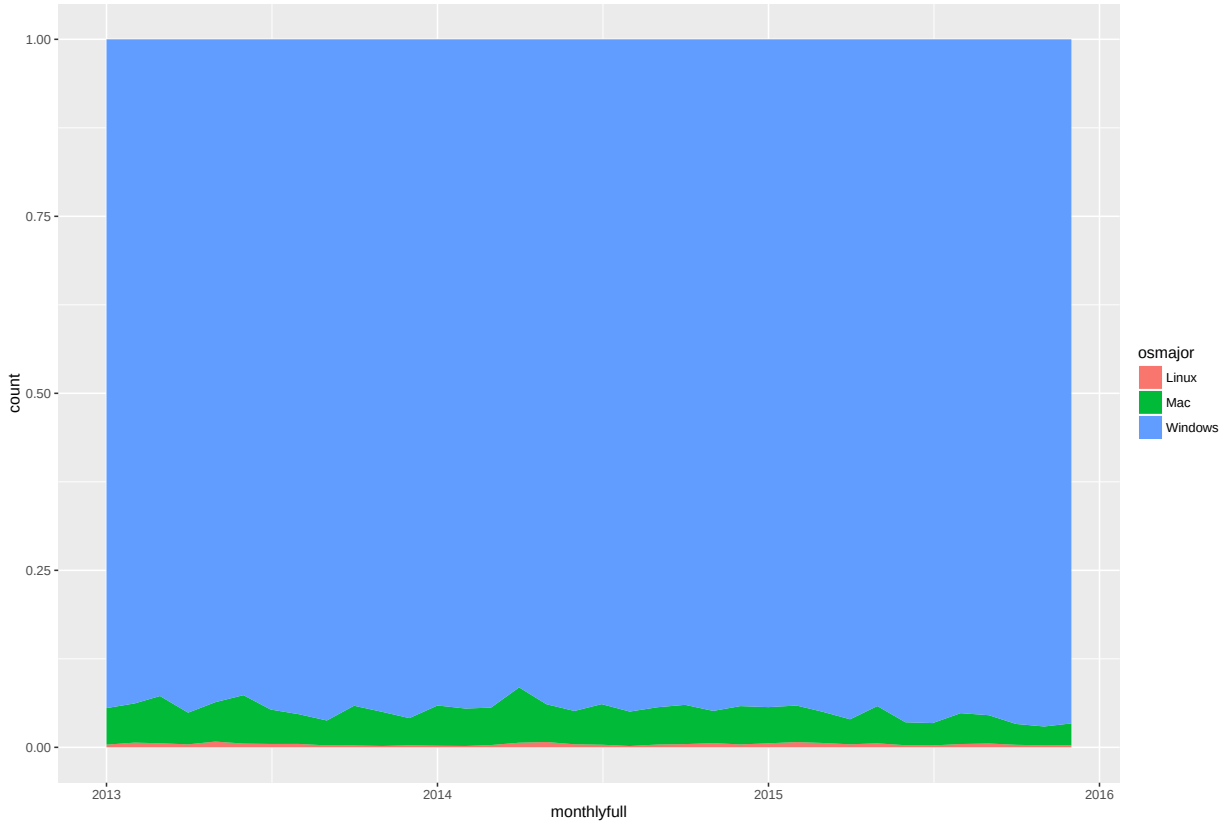
Look at the java versions:



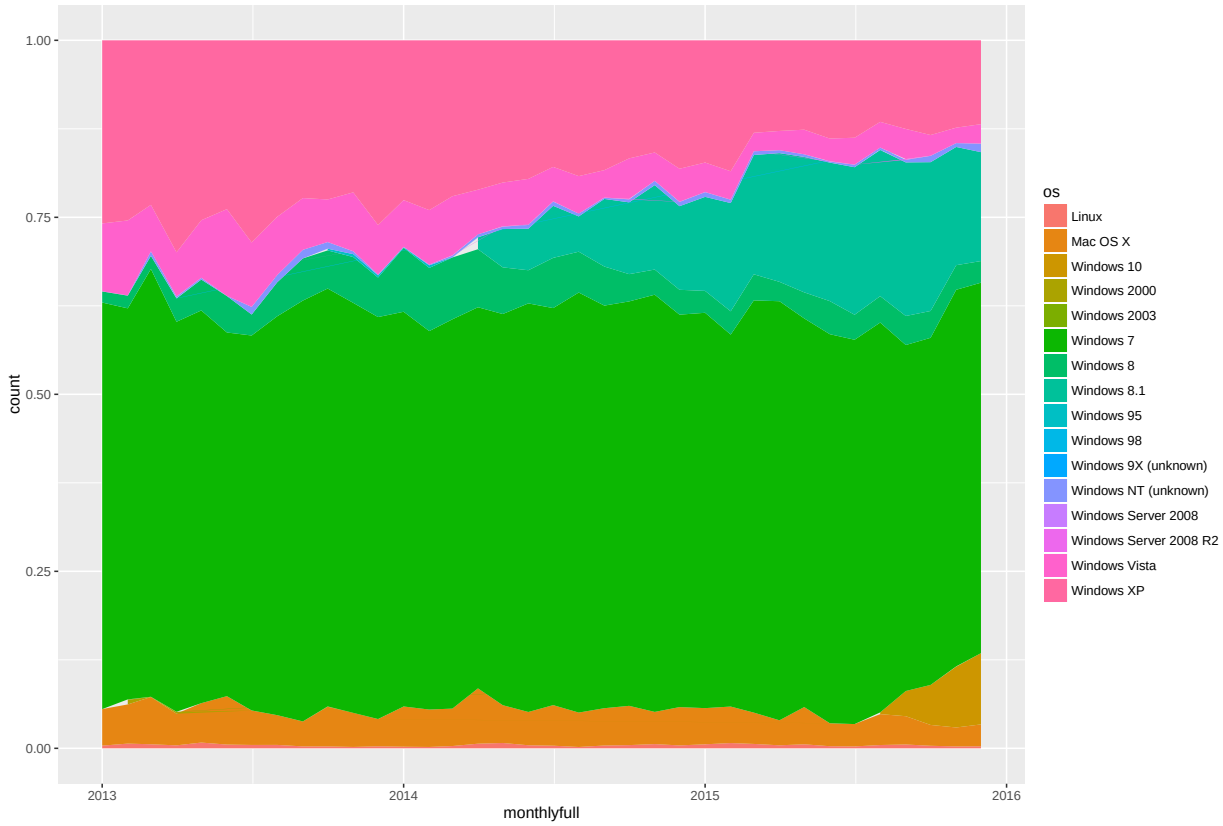
Lets delve deeper into other variables. For example the operating system used.



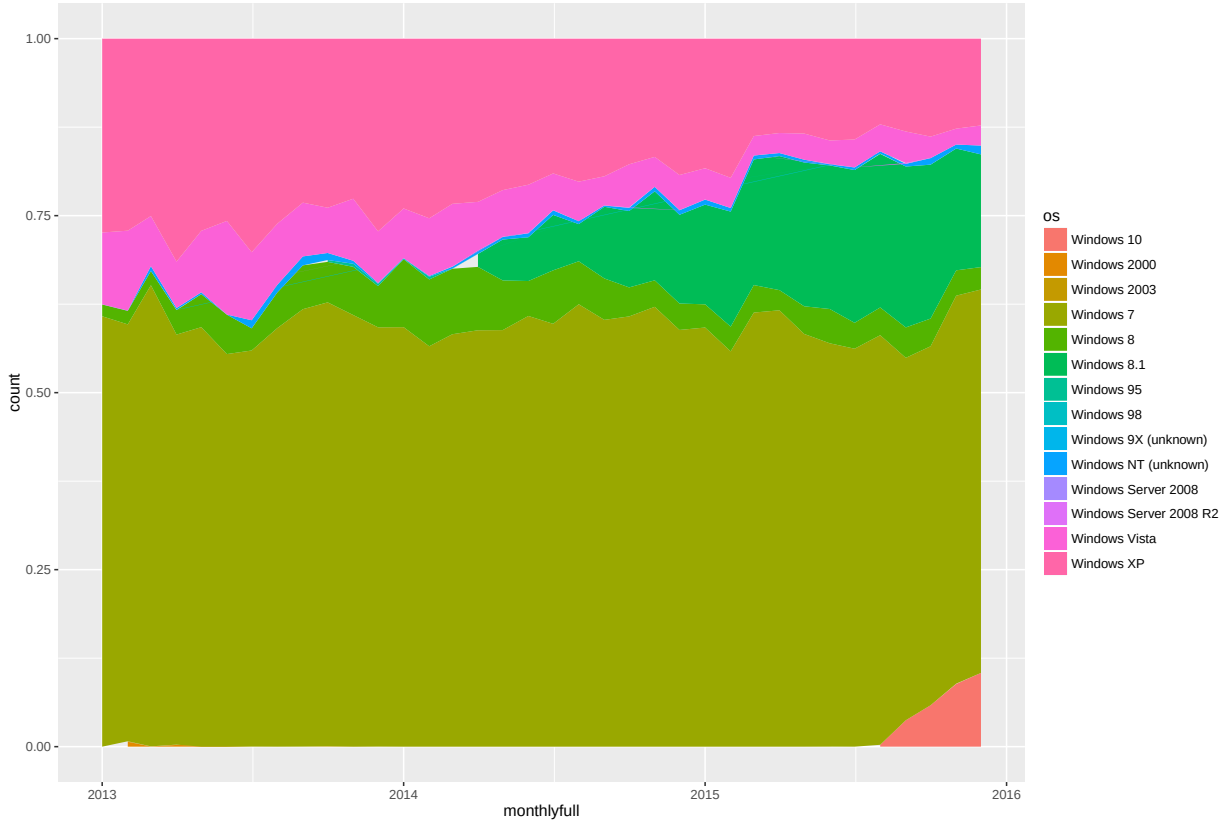
How about proportions?



Lets see the versions of windows

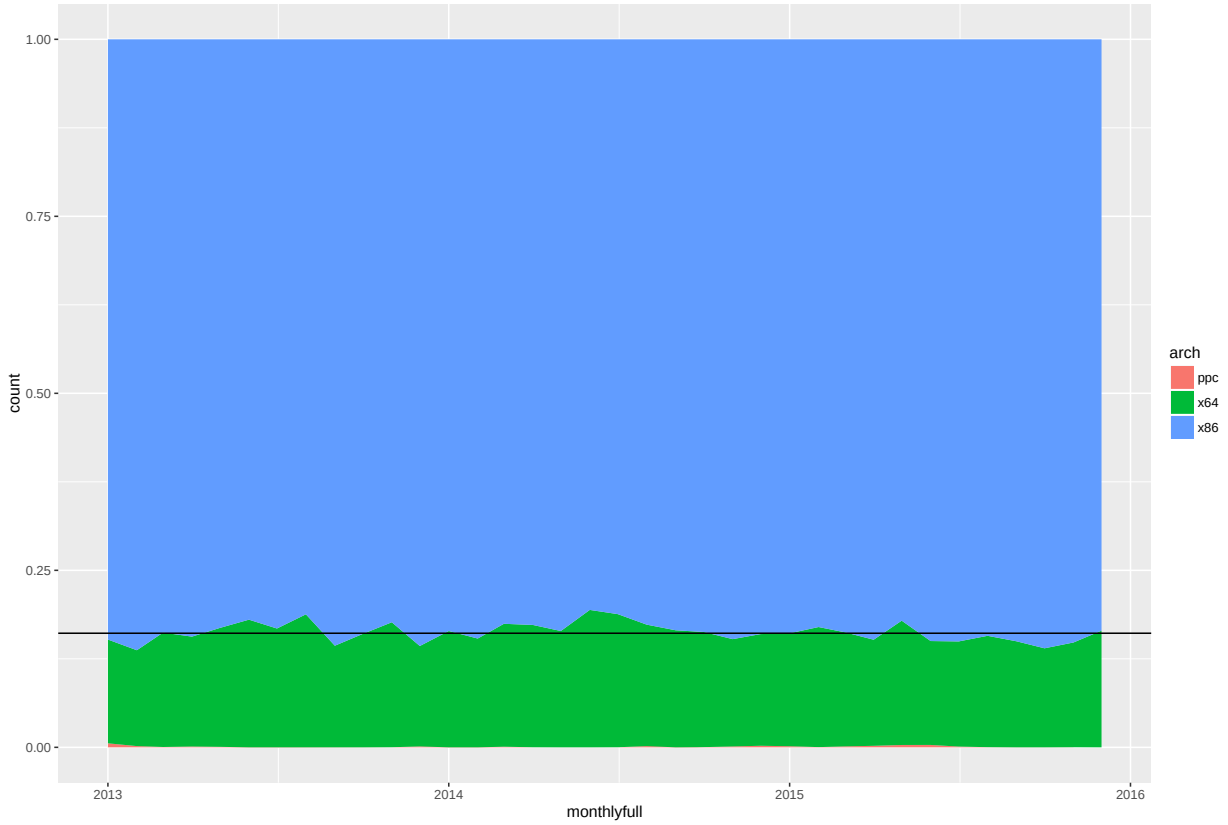


Only consider windows



Might need a better color scheme and a more logical ordering of the os factor to get a better overview.

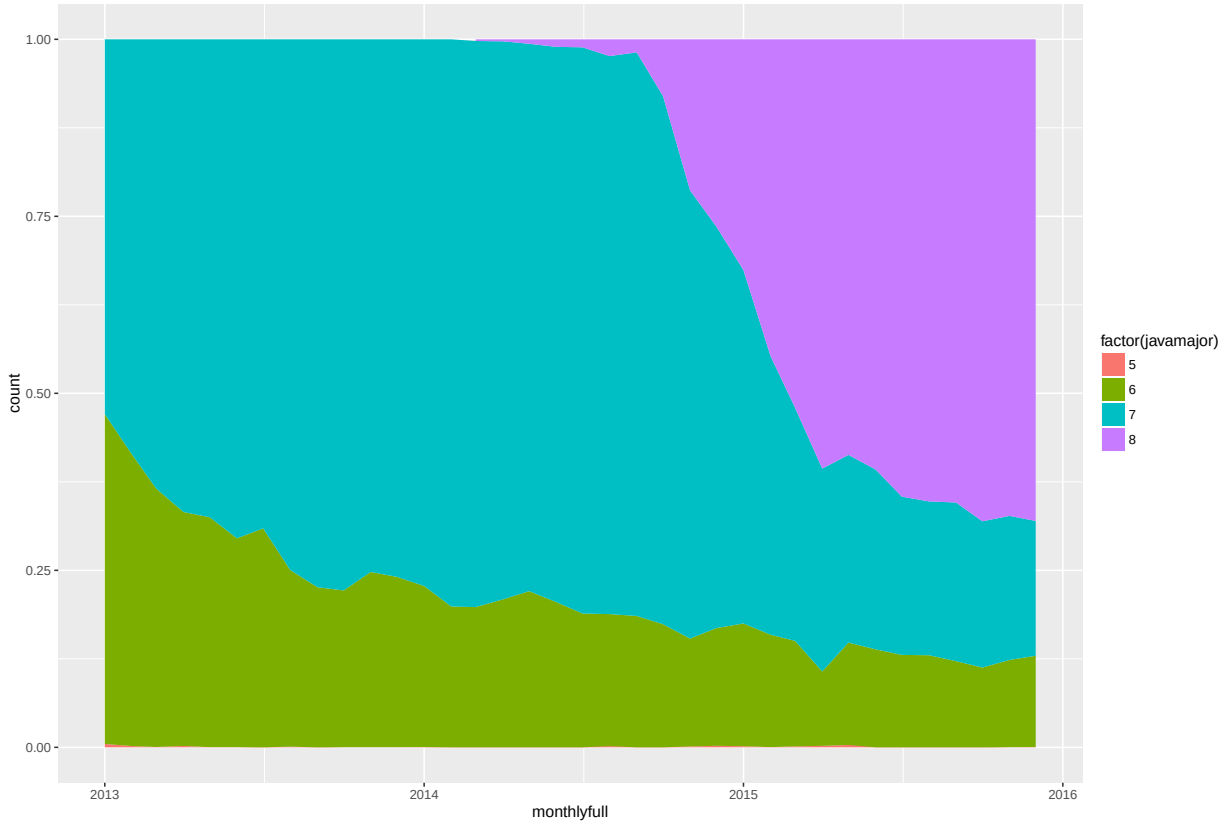
Lets do the same for computer architectures



We see that in the mean, 16% of the usage is done using an 64bit system.

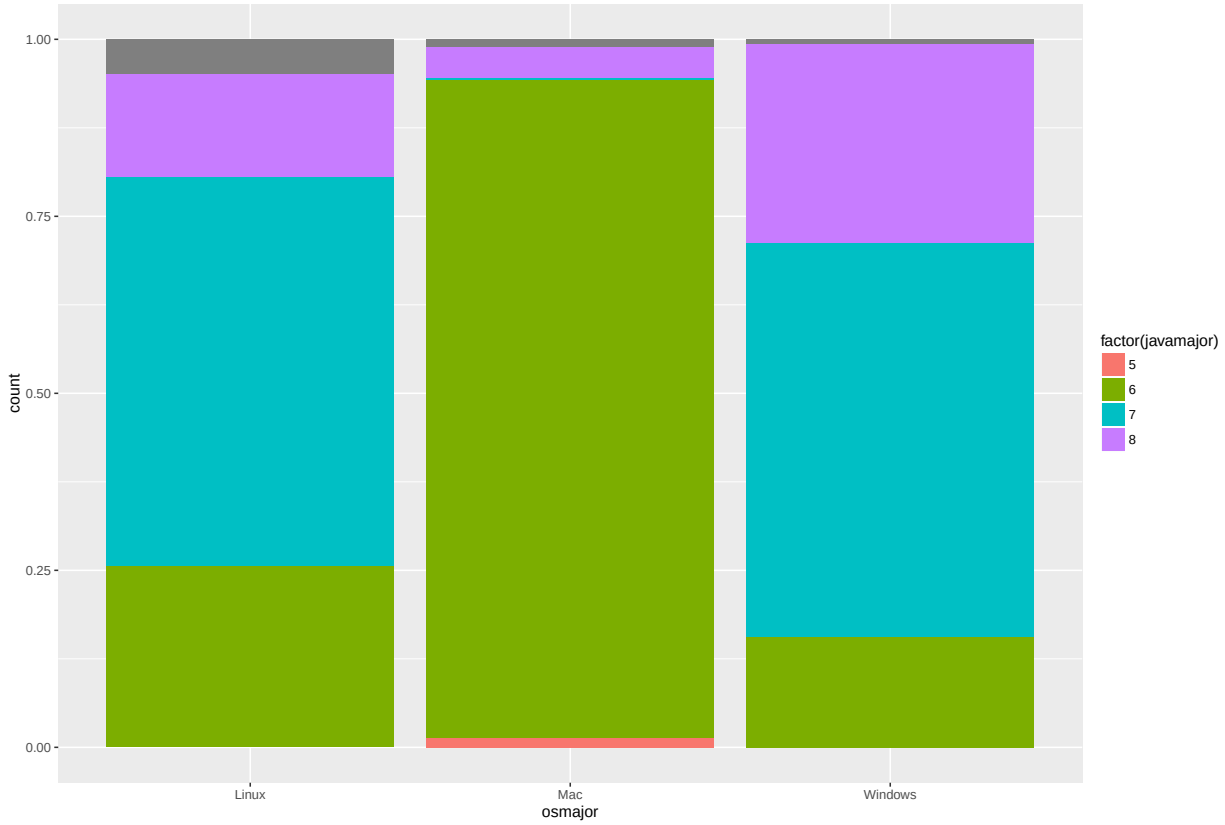
Most users seem to use 32bit architectures. This is curious, I would have expected to see an increase in 64bit architectures.

Now what about the used java versions? Lets just look at the major versions.



We see an aggressive push to java 8 starting in mid 2014. Still, a rather large proportion of users seem to use java 6, a long obsolete version.

It would be interesting to investigate the relationship of java version and operating system.



We see that the Mac has a very high proportion of java 6 users. The reason is probably, that for a long time, java 6 was the only java available for the Mac.

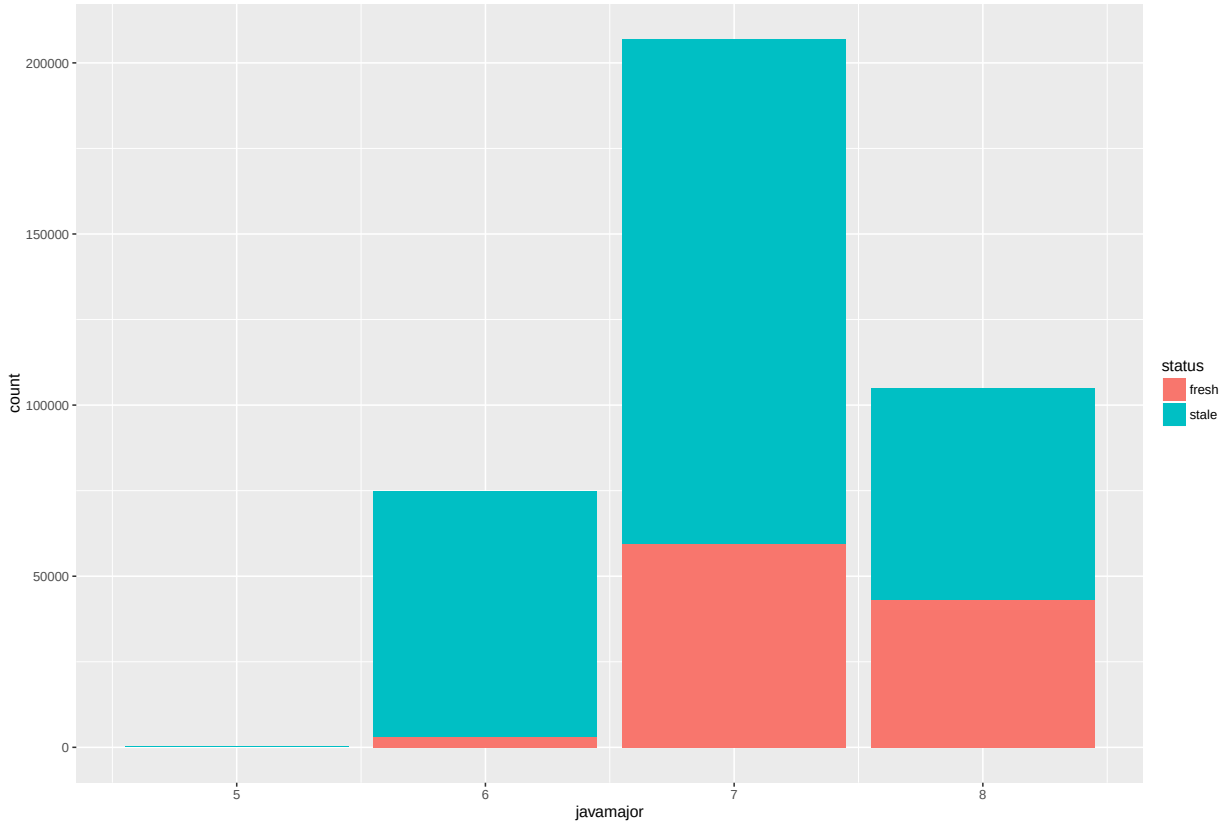
Next I am interested in the staleness of the java versions used. What do I mean by that? Due to bug fixes and fixes for security issues there are periodically new versions of java. As soon as the patch version of java is available, the previous one is considered stale, because it is not up to date.

Especially because of frequent security issues this topic is highly interesting. Earlier on, java was known for its lax upgrade policies. In recent time, the java installation contains an upgrade component which is quite aggressive in upgrading java to more recent versions.

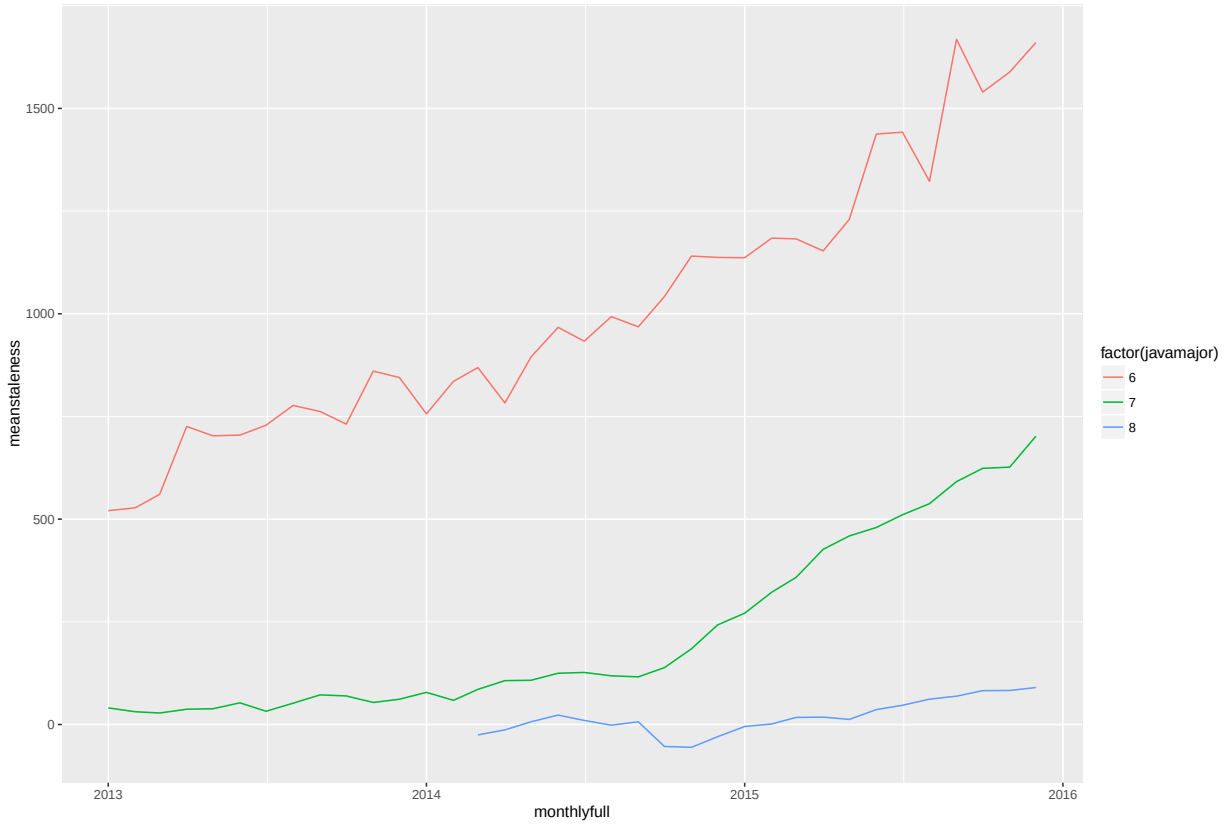
In order to analyze this topic, I already collected release dates of every java version. This also yielded the expiration dates of java versions by shifting the release dates by one entry. The data frame does contain the columns 'released' and 'expired'. Since we are interested in staleness, we use the 'expired' column to calculate the number of days the user uses a java version since it expired.

If staleness is negative then the java version is still current. Otherwise, a newer version is available.

Lets see how the distribution is between stale and fresh java installations.



We see that the proportion of stale java installations is highest for java 6 and best for java 8. This confirms the more aggressive update policy in newer versions.

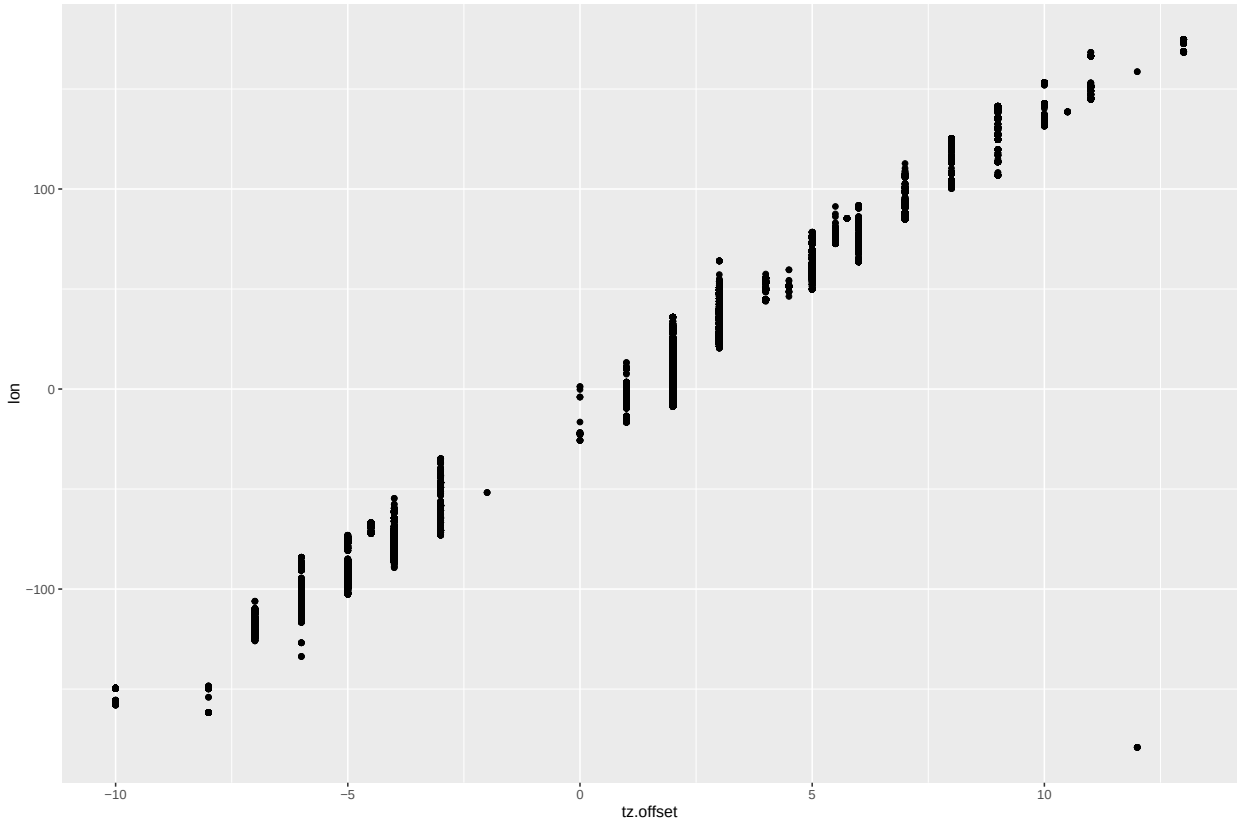


I excluded java 5 since there are only about 260 rows for it and the graph does consequently look very ragged and does not convey much information. In any case is java 5 long ago obsolete.

The mean staleness over the whole period for java 6 is 971.4414985 days and for java 7 171.2158423 days and finally for java 8 42.143492 days.

Lets explore some further relationships.

First, lets see how the timezone offsets are related to the longitudes:



As expected, there is a clear correlation (0.9859812) but we also clearly see that a bunch of longitudes are bundled into one timezone. Of course, this is the whole ideas of timezone, but we see it here nicely in this plot.

Also visible is a strange outlier in the right bottom corner.

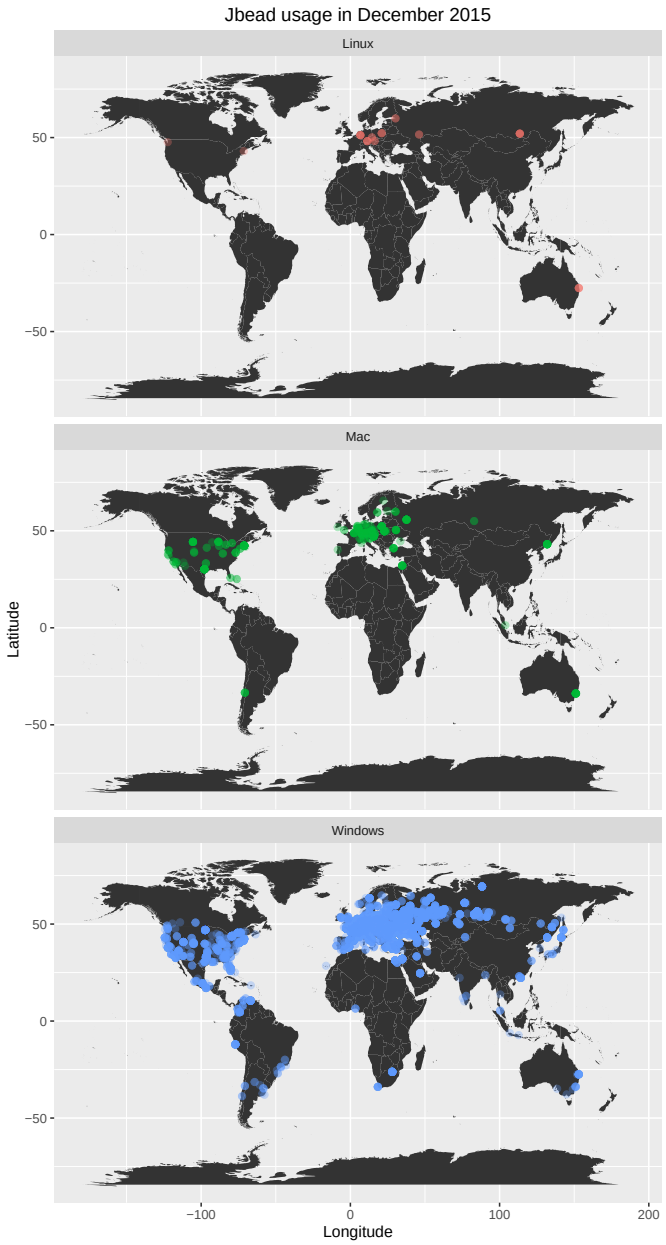
```
##                timestamp            ip version locale javaversion
## 44088 2013-07-04 00:58:09 91.106.234.66  1.0.23  ru_RU    1.6.0_24
## 44097 2013-07-04 02:12:35 91.106.234.66  1.0.23  ru_RU    1.6.0_24
## 44098 2013-07-04 02:27:42 91.106.234.66  1.0.23  ru_RU    1.6.0_24
## 44099 2013-07-04 02:44:47 91.106.234.66  1.0.23  ru_RU    1.6.0_24
## 44613 2013-07-06 03:49:59 91.106.234.66  1.0.23  ru_RU    1.6.0_24
## 46549 2013-07-13 00:49:41 91.106.234.66  1.0.23  ru_RU    1.6.0_24
##              os osversion arch osmajor javamajor   released    expired
## 44088 Windows XP       5.1  x86 Windows         6 2011-02-15 2011-03-21
## 44097 Windows XP       5.1  x86 Windows         6 2011-02-15 2011-03-21
## 44098 Windows XP       5.1  x86 Windows         6 2011-02-15 2011-03-21
## 44099 Windows XP       5.1  x86 Windows         6 2011-02-15 2011-03-21
## 44613 Windows XP       5.1  x86 Windows         6 2011-02-15 2011-03-21
## 46549 Windows XP       5.1  x86 Windows         6 2011-02-15 2011-03-21
##          timezone     lat      lon zip country countryCode      city
## 44088 Asia/Anadyr 66.3217 -179.122      Russia          RU Egvekinot
## 44097 Asia/Anadyr 66.3217 -179.122      Russia          RU Egvekinot
## 44098 Asia/Anadyr 66.3217 -179.122      Russia          RU Egvekinot
## 44099 Asia/Anadyr 66.3217 -179.122      Russia          RU Egvekinot
## 44613 Asia/Anadyr 66.3217 -179.122      Russia          RU Egvekinot
## 46549 Asia/Anadyr 66.3217 -179.122      Russia          RU Egvekinot
##                       regionName tz.offset day week year  dailyfull
```

```
## 44088 Chukotskiy Avtonomnyy Okrug        12 185   27 2013 2013-07-04
## 44097 Chukotskiy Avtonomnyy Okrug        12 185   27 2013 2013-07-04
## 44098 Chukotskiy Avtonomnyy Okrug        12 185   27 2013 2013-07-04
## 44099 Chukotskiy Avtonomnyy Okrug        12 185   27 2013 2013-07-04
## 44613 Chukotskiy Avtonomnyy Okrug        12 187   27 2013 2013-07-06
## 46549 Chukotskiy Avtonomnyy Okrug        12 194   28 2013 2013-07-13
##       dayofweek             tslocal hour monthlyfull staleness status
## 44088         5 2013-07-04 12:58:09   12  2013-07-01  836.0404  stale
## 44097         5 2013-07-04 14:12:35   14  2013-07-01  836.0921  stale
## 44098         5 2013-07-04 14:27:42   14  2013-07-01  836.1026  stale
## 44099         5 2013-07-04 14:44:47   14  2013-07-01  836.1144  stale
## 44613         7 2013-07-06 15:49:59   15  2013-07-01  838.1597  stale
## 46549         7 2013-07-13 12:49:41   12  2013-07-01  845.0345  stale
```

As it turns out, this is a place called Egvekinot and it is located to the far east of siberia in russia. In fact, it is so much to the east, that is longitude is almost wrapping around. This means in the plot, it belongs to the data in the upper right corner or the lower left corner.

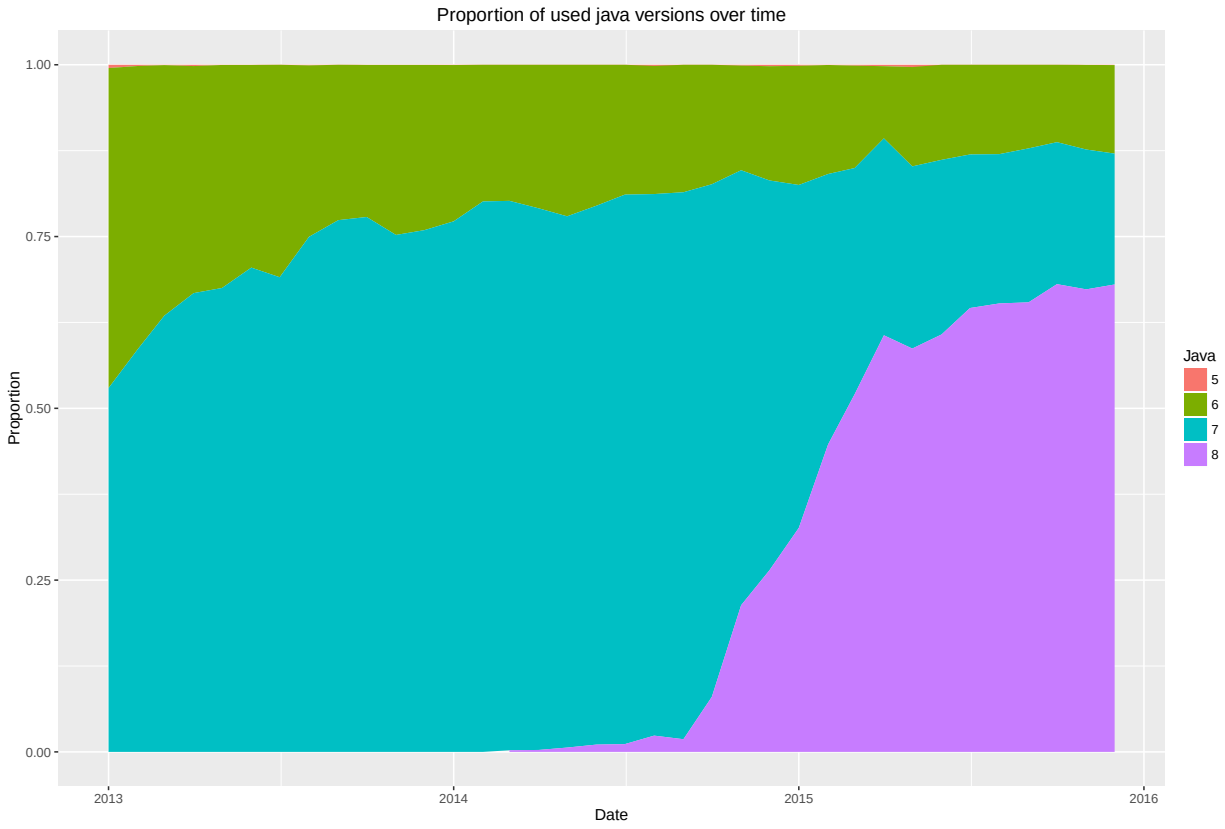
Final Plots

World wide usage distribution



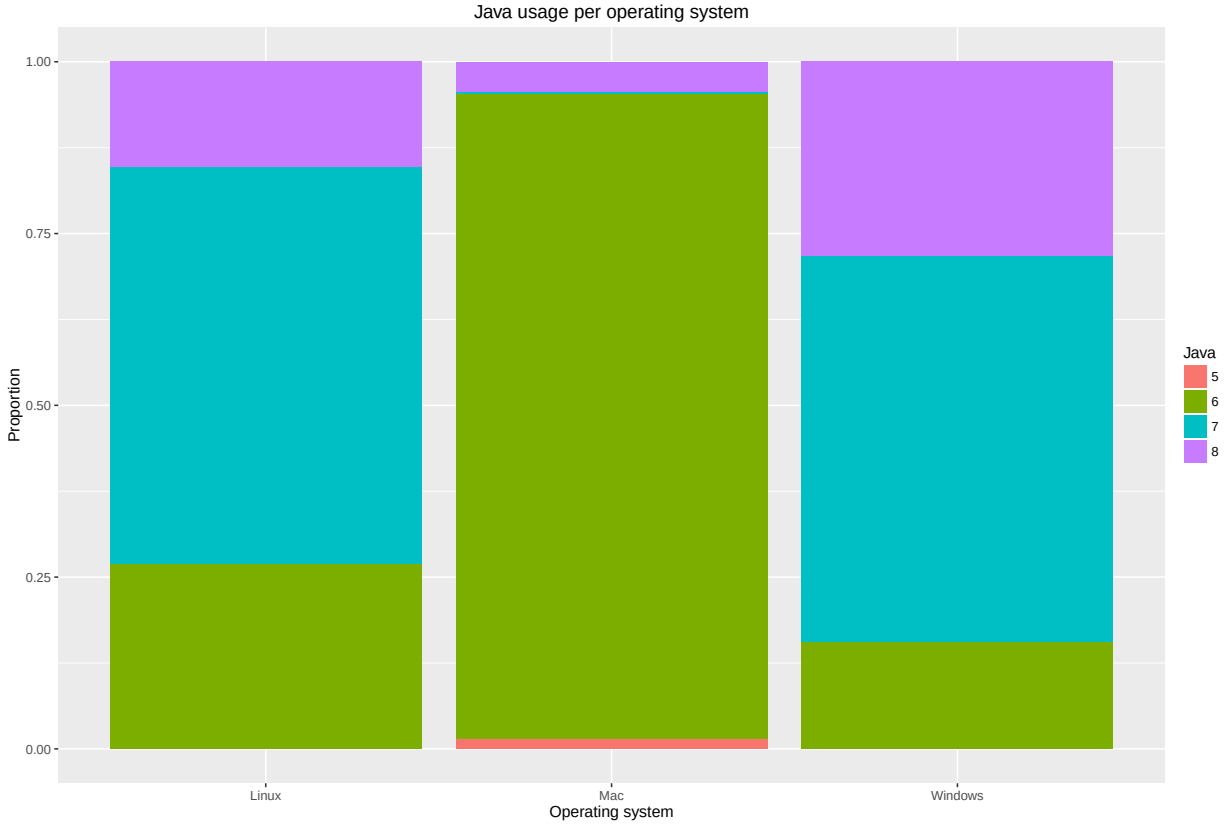
The plots show the distribution of the usage of jbead in december 2015 for the three major operating system Windows, Mac OS X and Linux. We see that linux is most heavily used in europe. The Mac is mostly used in europe and america with some clusters in the other continents. Windows is most used and spread across the globe.

Time series of major java versions proportions



The plot shows the development of proportions of java versions used over time. The java 5 version (long ago obsolete) is barely noticeable, Java 6 decreased from almost 50% to about 20%. The most dramatic change was the switch from java 7 to java 8. This started in mid-2014. First the new java 8 only gained a little, but then its usage exploded and rose rapidly to more than 50%. At the end, it stabilized at about 65%.

Proportions of java versions per operating system



The plot shows for each of the three major operating systems the proportion of java versions used. For Linux the most used version is java 7. For the mac, there is a very tiny bit of java 5 and java 7, a little of java 8 but most usage happens with java 6. On windows java 7 has the lead, followed by java 8 and finally java 6.

Reflection

I was positively surprised with how little effort it was possible to collect and to augment and clean the data.

One speed bump was the resolution of IP to geographical information. But I finally found a free service with high enough rate to convert all approx. 76'000 IPs to geo data in about ten minutes. I used python for accessing this service and for converting the output into a csv file.

For the java version information, I got data from the "java version wiki" and put them into a hand-made csv file.

For combining the base log and the ip geo information and the java version, I used R with plyr/join.

When I implemented the version check in jbead, I was quite conscious of its power. I thought long about what information to include in the request. I did not want to collect to much personal information since this would be quite intrusive to the users. But I did want to collect some usage data because this would help me decide where to invest my scarce time resources. For example, if I find that only a miniscule number of users use the linux operating system, then maybe I could stop investing time for this version.

I finally decided to collect version information about the os and java, since this was quite relevant for my development purposes.

I never collected personally identifiable information except for the IP. I never used cookies or GUIDs or something like this. Had I done this, then I would have been able to extract usage profile based on single users.

Nonetheless, the existing data is quite fascinating and allows for very interesting analyses. I was surprised by this. At the beginning of the project, I was worried, that the data would be boring.

First I did have some trouble adjusting the date and times according to local time zones. After some dead-ends, I found a simple solution: calculate the hour-offset for each timezone and simply add it to the timestamp to get local time.

Further work

One idea maybe worth investigating further would be to have a more detailed look at the geographical data grouped according to continents, i.e americas, europe, asia, ...