Supplementary material to “Refuting tobacco-industry funded research: empirical data shows decline in smoking prevalence following introduction of plain packaging in Australia”

Reverse-engineering Kaul and Wolf’s figures to reconstruct the data they used in their working papers on plain packaging

P.A. Diethelm, OxyRomandie

Description of the method

This supplementary document describes how Diethelm and Farley reconstructed the data used by Kaul and Wolf in their working papers.[1] It is believed that the (original) method presented below has made it possible to reconstruct the data with almost perfect accuracy. The method consists in a number of steps, which will be documented in detail. The computer program used in the fourth step is shown in Annex 1 and the reconstructed data is shown in Annex 2.

Step 1. Extracting the images from Kaul and Wolf’s paper

Kaul and Wolf’s two working papers are in PDF format. They can be downloaded from the website of the University of Zürich. We consider here only the second working paper, on adults (the same procedure could be applied to the first paper). The figures showing the time series plot of the monthly sample sizes (Figure 1 in the paper) and of observed prevalence (Figure 2 in the same paper) were apparently produced using the R statistical package. We used these two figures to extract the data on sample size and prevalence. We first read the working paper into Adobe Acrobat Reader and accessed the page containing Figures 1 and 2 (page 11). We took a snapshot of each figure using Acrobat’s snapshot function and “printed” it to a PDF file, producing files **`prevalence.pdf`** and **`sample-size.pdf`**.

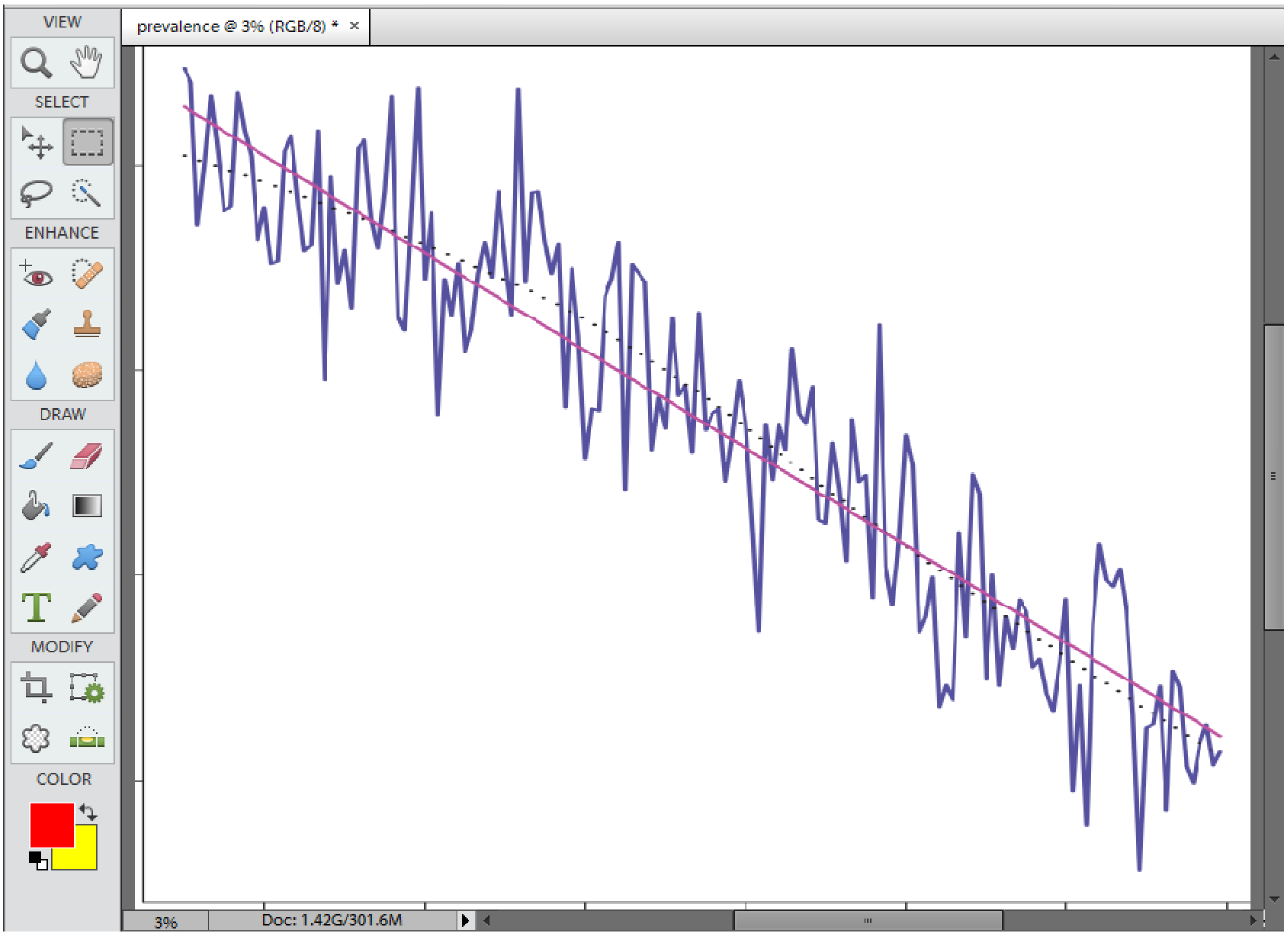
---

[1] Kaul A and Wolf M. The (Possible) Effect of Plain Packaging on the Smoking Prevalence of Minors in Australia: A Trend Analysis. University of Zurich Department of Economics Working Paper Series. May 2014; Available from http://www.econ.uzh.ch/static/workingpapers.php?id=828

Kaul A and Wolf M. The (Possible) Effect of Plain Packaging on Smoking Prevalence in Australia: A Trend Analysis. University of Zurich Department of Economics Working Paper, June 2014. Series. Available from: http://www.econ.uzh.ch/static/workingpapers.php?id=844

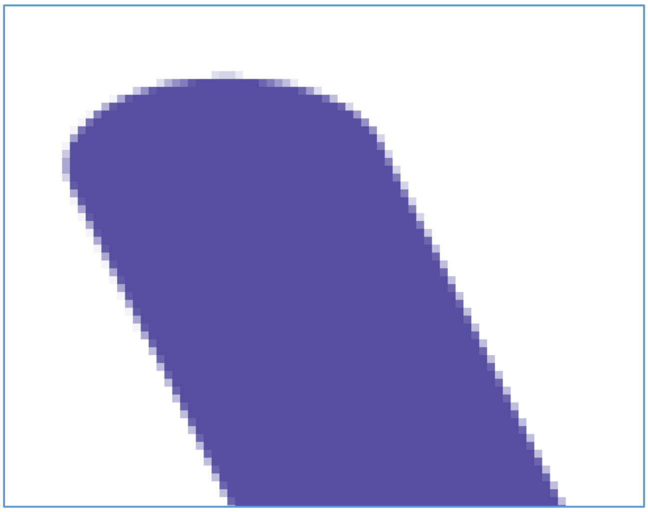
Step 2. Pre-processing the images in Photoshop

We read each PDF file produced at Step 1 into Photoshop, specifying a *very high resolution* of **2400 pixels per inch** (producing a *very large image* of about 26,000 x 20,000 pixels). The following picture shows how the image for prevalence looked like in Photoshop:

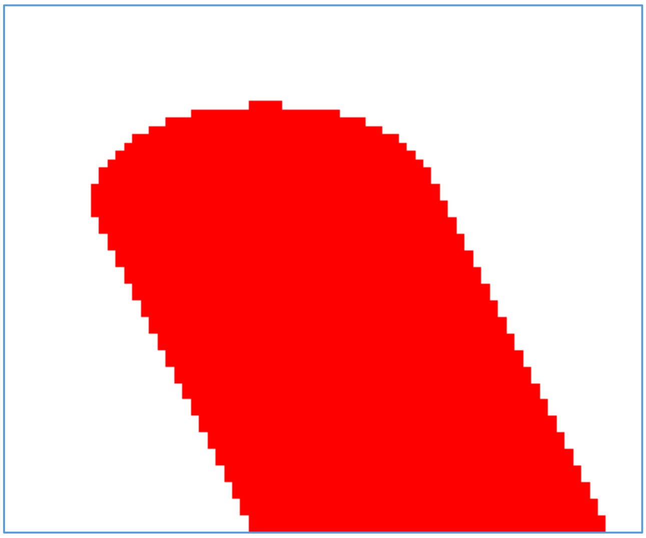


With Photoshop, we modified the colour of the prevalence (and sample size) line, made of various shades of blue (by "anti-aliasing"). We replaced all these shades of blue with a 100% pure red with no anti-aliasing. The enlarged before-and-after details below illustrate this step.

Before:



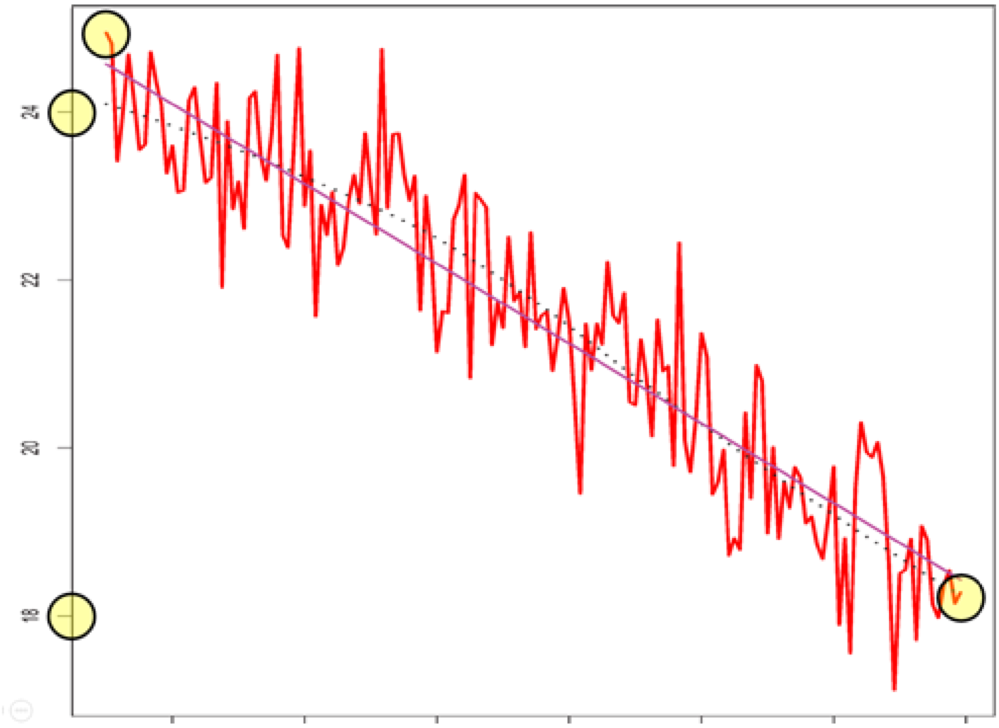
After:



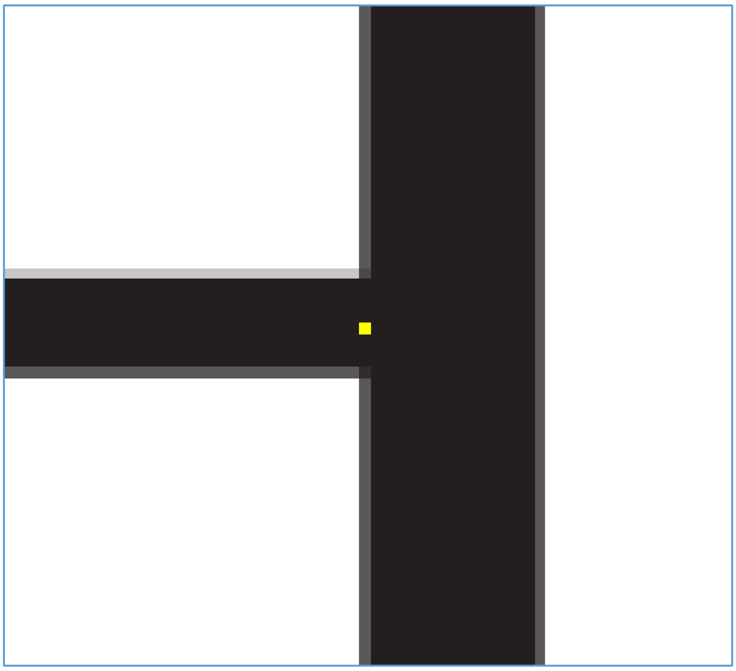
Before performing this step, we made sure pure red - colour rgb(255,0,0) - was not already used in the picture. The purpose of this step was to obtain a good contrast between the red line and its white background in order to facilitate the identification of the edge pixels of the line by the **`image2data.py`** computer program described below.

Step 3. Identifying key points on the images with yellow pixels

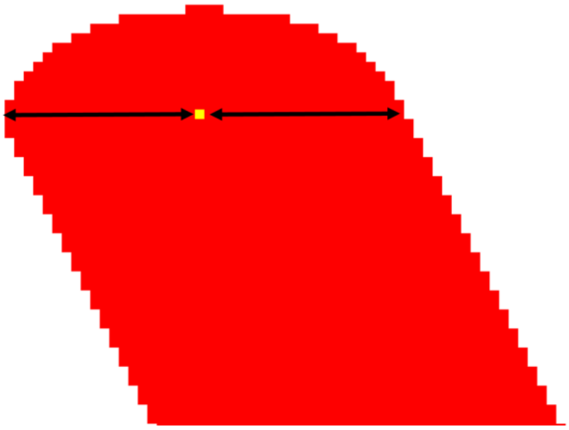
Still processing each figure in Photoshop, we also made that there was no pure yellow - rgb(255,255,0) – pixel in the image. We then painted a *single pure yellow pixel* at four particular places, as shown in the illustration below:



Two yellow pixels were painted on the vertical axis, as shown below. The pixels were be put at the middle point of the highest and lowest tick marks:



The other two yellow pixels were used to identify the starting point and the ending points of the plotted line. The pixels at the start and end of the plot line were placed as shown in the following picture, in a way to approximate as best as possible the actual starting and ending points of the underlying line.



We saved the image thus obtained for each figure in JPEG format with the highest quality (12), under file names **`prevalence.jpg`** and **`sample-size.jpg`**.

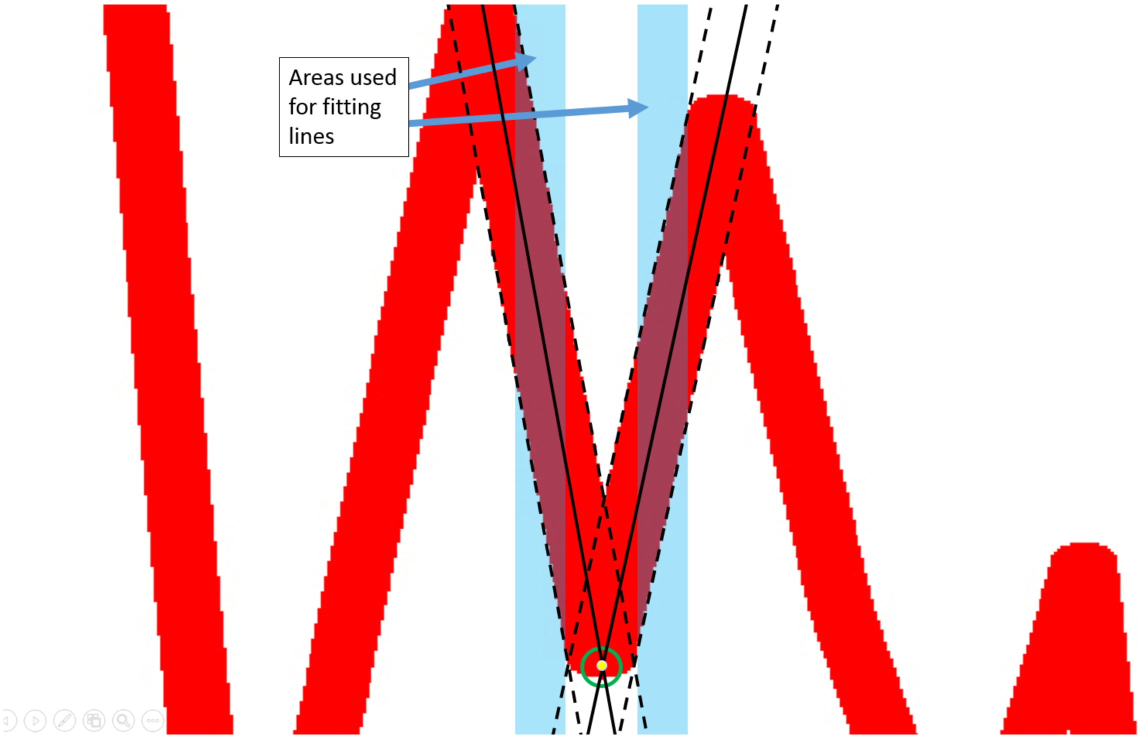
Step 4. Running Python program `image2data.py`

We ran Python program **`image2data.py`** which we wrote specifically to treat the above images (see Annex 1). For each image file, 5 parameters were specified: two for the y-values corresponding to the yellow pixels on the vertical axis (corresponding respectively to the lowest and highest tick marks), two for the x-values associated with the pixels put at the start and end of the plot line (normally 1 and 156, since the plot starts at month 1 and ends at month months 156) and one specifying the number of points (156). The parameters were as follows for Figure 1 and Figure 2 of Kaul and Wolf's June paper.

Figure 1 (sample size): 3500, 5000, 1, 156, 156

Figure 2 (prevalence): 18, 24, 1, 156, 156

The Python program calculates the data values by fitting straight lines on the edge pixels of the plot. For each line segment between two adjacent point, the program identifies the *left* edge pixels and the *right* edge pixels and fits a straight line by least square regression (if the line segment is more horizontal than vertical, the *top* and *bottom* edge pixels are used instead of the left and right edge pixels). Two lines are thus obtained – shown as dashed lines in the illustration below - , a left line and a right line (or a top line and a bottom line). The program then calculates the middle line between these two lines and assumes that this was the line representing the segment joining the two points - if the left line is $ax + b$ and the right line is $cx + d$, the middle line will be given by $\frac{1}{2}(a + c)x + \frac{1}{2}(b + d)$. The data points which we are looking for are assumed to lie at the intersection of adjacent segments, as shown in the picture (surrounded by the green circle).



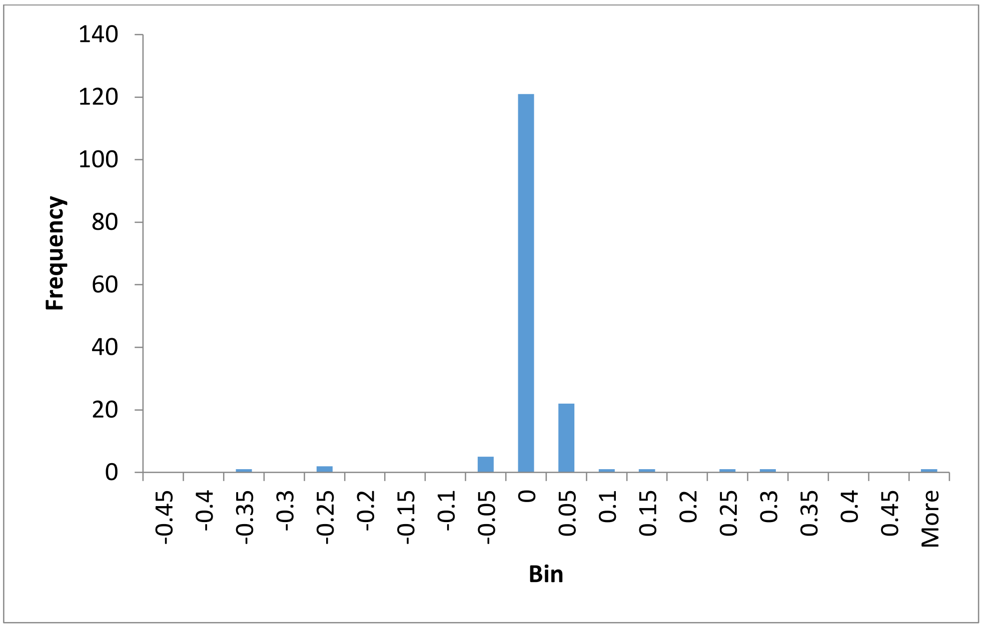
Using the Python program, we created two data files, **`sample-size.txt`** and **`prevalence.txt`** (in tab delimited text format), one containing the estimated values of sample sizes, the other containing the estimated values of observed monthly prevalence. These values were produced with high precision (10 significant digits).

Step 5. Assembling the data produced by program `image2data.py`

The two data files (**`sample-size.txt`** and **`prevalence.txt`**) produced by program **`image2data.py`** were then assembled into a single Excel file, with three columns, ***time*** (with values 1 to 156), ***prev*** and ***size***. Steps 6-7 below were performed in Excel on the joined data.

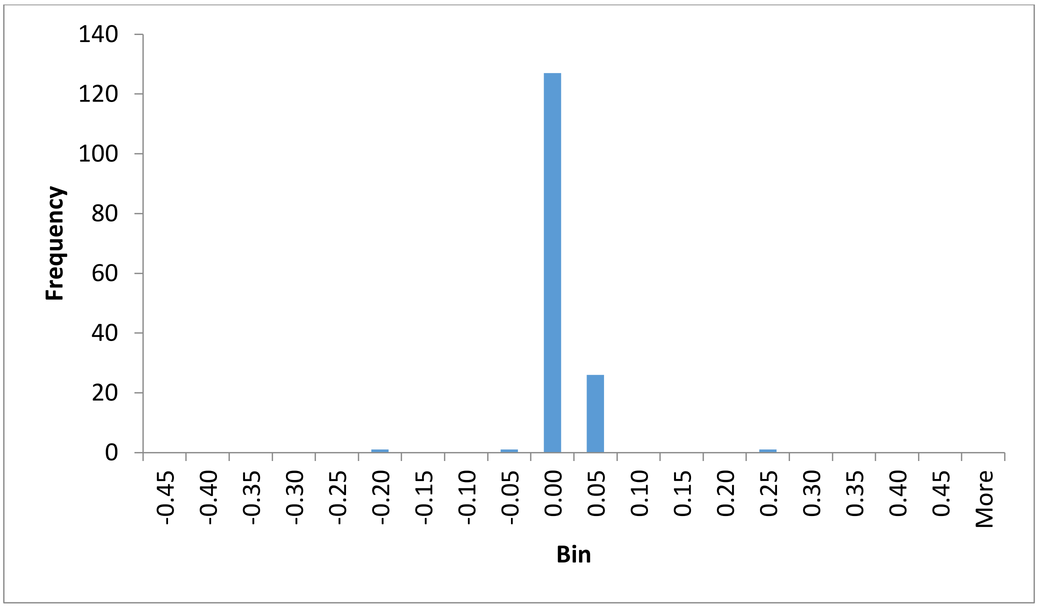
Step 6. Assessing the accuracy of the resulting approximations

Sample size data: When working on Figure 1 (sample size), the y-coordinate of the data points obtained by the above method approximates the number of observations, which are *whole numbers*. We assumed that if our results were close to whole numbers, this indicated that the approximation was good and that probably many of the actual numbers of monthly observations were reconstructed exactly. See below the histogram of the difference between our approximations of sample sizes and the nearest whole number:



One can see that indeed there was a concentration of this difference around zero: only 8 data points fell outside the range [-0.1, 0.1].

Prevalence data: We worked on Figure 2 to reconstruct the values of estimated monthly prevalence. We then assumed that the original observed prevalence data used by Kaul and Wolf were obtained by dividing the number of smokers in the monthly samples by the corresponding number of observations (sample size). We made the following reasoning: if we take the approximate prevalence values produced by our program and multiply them by the approximated sample sizes, we get an approximation of the number of smokers in the monthly sample. That is, we get a value which again approximates a *whole number*. Looking at the difference between the approximated values of the number of smokers we obtained and the nearest integer provided us with an indication of the accuracy of our approximations. See below the histogram of the differences between our approximations of the number of smokers and the nearest whole number:



Step 7. Fine tuning the sample size estimates

All data points, *except two,* fell in the interval [-0.1, 0.1]. This was excellent, but we did not stop there. We looked closely at these two particular points. Table below shows the information we considered and the correction of the sample sizes this suggested.

Month	90	110
(1) ***size*** -Estimated sample size (produced by program **`image2data.py`**)	4325.64147194	4312.26422951
(2) Sample size rounded	4326	4312
(3) ***prev*** - Estimated monthly prevalence (%, produced by program **`image2data.py`**)	21.47969507	21.07572527
(4) Estimated number of smokers – ***prev********size***/100	929.2116	908.7853
(5) Number of smokers rounded ***round(prev*size/100;0)***	929	909
(6) Deviation from nearest whole number – (4)-(5)	0.211609	-0.21473
(7) Corrected sample size	**4325**	**4313**
(8) Corrected estimate of number of smokers – (2)*(7)/100	928.9968	908.996
(9) Corrected number of smokers rounded	929	909
(10) Deviation of (8) from nearest whole number	-0.00319	-0.00397
(11) Revised estimated prevalence in % – (9)*100/(7)	21.47976879	21.07581730

When changing the estimated sample size for month 90 from 4326 to 4325 (taking the floor of the estimated size produced by program **`image2data.py`** instead of its ceiling) and applying to it the prevalence figure produced by the program, we got a number which was very close to a whole number. This suggested that 4325 was the correct sample size. We made a similar reasoning for the sample size of month 110. These were the only two manual corrections we applied to the data automatically produced by program **`image2data.py`**.

With the data set thus corrected, we calculated the number of smokers in each monthly sample (the method is illustrated in the above table) and then we *re-computed* the estimated monthly prevalence for each month as shown on row (11) of the table, to ensure that the estimated prevalence values were strictly equal to the number of smokers divided by the sample size.

The final data can be found in Appendix 2.

Step 8. Validating the data estimates by reproducing results computed with the real data

Using the final data, we were able to reproduce exactly Kaul and Wolf regression results presented in Table 1 of their June working paper (i.e. up to rounding precision). We were able to also reproduce exactly results in prof. Jann's Methodological Report,[2] as is illustrated by the following two extracts, the first one being from prof. Jann's report:

```
. blogit  smokers observations month treat

Logistic regression for grouped data              Number of obs   =     506657
                                                  LR chi2(2)      =     696.42
                                                  Prob > chi2     =     0.0000
Log likelihood = -258774.15                       Pseudo R2       =     0.0013

------------------------------------------------------------------------------
    _outcome |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       month |  -.0027001   .0001242   -21.73   0.000    -.0029435   -.0024566
       treat |  -.0158519   .0139325    -1.14   0.255    -.0431591    .0114553
       _cons |  -1.072587   .0118326   -90.65   0.000    -1.095779   -1.049396
------------------------------------------------------------------------------
```

The second is the output produced by R when running the same logit analysis using our data:

```
Call:
glm(formula = cbind(smokers, non.smokers) ~ time + pp, family = binomial("logit"))

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-3.06053  -0.85692   0.04453   0.83168   3.15736

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.0725872  0.0118326 -90.647   <2e-16 ***
time        -0.0027001  0.0001242 -21.734   <2e-16 ***
pp          -0.0158519  0.0139325  -1.138    0.255
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We also noted that the number of observations reported in prof. Jann's analysis (506,657) corresponds exactly to the number we have estimated. We have done some sensitivity analysis showing that if a single monthly sample size figure is changed by just one unit, the results are no longer totally identical.

This is a good indication that we were able to reconstruct the data used by Kaul and Wolf in their June working paper with near perfect accuracy.

2015.10.27/pad

[2] Jann B. Methodological Report on Kaul and Wolf's Working Papers on the Effect of Plain Packaging on Smoking Prevalence in Australia and the Criticism Raised by OxyRomandie. University of Bern, Institute of Sociology, Bern, 10 March 2015

Annex 1. Python code of program **image2data.py**

```
# -*- coding: utf-8 -*-
#
# Obtaining data by reverse engineering from published figures
#
# Author: Pascal Diethelm, 02.10.2015

PARM = [
 ['sample size', 'sample-size.png', 'sample-size.txt', 3500, 5000, 1, 156, 156],
 ['prevalence',  'prevalence.png',  'prevalence.txt',    18,   24, 1, 156, 156]
]

# Imports --------------------------------------------------

import os
chdir = os.chdir
import scipy
from   scipy import stats
import PIL
from   PIL import Image

# Constants ------------------------------------------------

DIR        = '.'  # Work directory (change to directory where images are stored)
MARGIN     = 0.33 # Margin around exact x-values defining vertical band considered for
fitting line

# Functions ------------------------------------------------

def is_yellow(x,y) :
   pix = PX[x,y]
   return (pix[0] >= 200 and pix[1] >= 200 and pix[2] <= 50)
def is_red(x,y) :
   pix = PX[x,y]
   return (pix[0] >= 200 and pix[1] <= 50 and pix[2] <= 50)
def is_white(x,y) :
   pix = PX[x,y]
   return (pix[0] >= 200 and pix[1] >= 200 and pix[2] >= 200)

def process_image(img_file, v0, v1, t0, t1, t_max) :

   global X0, Y0, X1, Y1, Z0, Z1, T0, T1, V0, V1, NX, NY, PX, A_T2X, A_X2T, A_Y2V

   print("Processing file "+img_file)

   X0 = -1; Y0 = -1; X1 = -1; Y1 = -1
   V0 = v0; V1 = v1; T0 = t0; T1 = t1

   img = Image.open(img_file)
   NX = img.size[0]
   NY = img.size[1]

   PX = img.load()

   print("Width = "+str(NX)+" pixels, height = "+str(NY)+" pixels")

   for x in range(NX) :
      for y in range(NY) :
        if (is_yellow(x,y)) :
           print("Yellow pixel at ("+str(x)+","+str(y)+")")
           if   (Y1 == -1) : Y1 = y
           elif (Y0 == -1) : Y0 = y
           elif (X0 == -1) : X0 = x; Z0 = y
           elif (X1 == -1) : X1 = x; Z1 = y

   if (Y1 > Y0) :
      y  = Y1
      Y1 = Y0
      Y0 = y

   A_T2X = (X1-X0)/(T1-T0)
```

```
   A_X2T = (T1-T0)/(X1-X0)
   A_Y2V = (V1-V0)/(Y1-Y0)

   segments = []
   for t in range(1,t_max) :
      x_low = t2x(t+MARGIN);
      x_hi  = t2x(t+1-MARGIN);
      segment = calc_segment(t,x_low,x_hi)
      segments.append(segment)
      print([t,segment])

   value = []

   v_est  = y2v(Z0)
   value.append([1,1,v_est,0,v_est,v_est])

   for t in range(1,t_max-1) :
      x = t2x(t+1)
      [a,b]  = segments[t-1]
      [c,d]  = segments[t]
      v_left = y2v(a*x+b)
      v_right= y2v(c*x+d)
      v_mid  = (v_left+v_right)/2
      if abs(a-c) > 0.01 :
         v_est  = y2v(a*(d-b)/(a-c) + b)
         t_est  = x2t((d-b)/(a-c))
      else :
         v_est  = v_mid
         t_est  = t
      value.append([t+1,t_est,v_est,v_left,v_right,v_mid])

   v_est  = y2v(Z1)
   value.append([t_max,t_max,v_est,v_est,0,v_est])

   return value

def t2x(t) : return int(round(X0+(t-T0)*A_T2X,0))
def y2v(y) : return V0+(y-Y0)*A_Y2V
def x2t(x) : return T0+(x-X0)*A_X2T

def calc_segment(t,x_lo,x_hi) :
   x_left  = []
   y_left  = []
   x_right = []
   y_right = []
   x_up    = []
   y_up    = []
   x_down  = []
   y_down  = []
   for x in range(x_lo+1, x_hi) :
      for y in range(1, NY-1) :
        if (is_red(x,y)) :
          left  = is_white(x-1,y-1) or is_white(x-1,y) or is_white(x-1,y+1)
          right = is_white(x+1,y-1) or is_white(x+1,y) or is_white(x+1,y+1)
          up    = is_white(x-1,y-1) or is_white(x,y-1) or is_white(x+1,y-1)
          down  = is_white(x-1,y+1) or is_white(x,y+1) or is_white(x+1,y+1)
          if left and not right :
             x_left.append(x)
             y_left.append(y)
          elif right and not left :
             x_right.append(x)
             y_right.append(y)
          if up and not down :
             x_up.append(x)
             y_up.append(y)
          elif down and not up :
             x_down.append(x)
             y_down.append(y)
   if min(len(x_left),len(x_right)) >= min(len(x_up), len(x_down)) :
      a1, b1 = linear_regress(x_left,y_left)
      a2, b2 = linear_regress(x_right,y_right)
   else :
      a1, b1 = linear_regress(x_up,y_up)
      a2, b2 = linear_regress(x_down,y_down)
   return [(a1+a2)/2, (b1+b2)/2]

def linear_regress(x, y) :
```

```
   n = len(x)
   x_bar = sum(x)/n
   y_bar = sum(y)/n
   sumxy = 0
   sumx2 = 0
   for i in range(n) :
      sumxy += (x[i]-x_bar)*(y[i]-y_bar)
      sumx2 += (x[i]-x_bar)*(x[i]-x_bar)
   a = sumxy/sumx2
   b = y_bar - a*x_bar
   return [a,b]

def write_file(file, data) :
   output("t\tt_est\tv_est\tv_left\tv_right\tv_mid")
   for [t,t_est,v_est,v_left,v_right,v_mid] in data :
      output(str(t)+"\t"+fmtP(t_est)+"\t"+fmtP(v_est)+"\t"+fmtP(v_left)+"\t"+fmtP(v_right) \
             +"\t"+fmtP(v_mid))
   write_output(file)

def output(line) :
   lines_out.append(line+"\n")
   print(line)

def write_output(file) :
   file_out = open(file, 'w');
   file_out.writelines(lines_out)
   file_out.close()

def fmtP(P) : return '{:12.8f}'.format(P)


# Main procedure -------------------------------------------

chdir(DIR)

for [label, img_file, txt_file, v0, v1, t0, t1, t_max] in PARM :
   lines_out = []
   print('Processing '+label+' graph')
   data = process_image(DIR+"/"+img_file, v0, v1, t0, t1, t_max)
   write_file(DIR+"/"+txt_file, data)

# EOF
```

Annex 2. Reconstructed data using Figure 1 and Figure 2 of Kaul and Wolf's June working paper

time	*prev*	*smokers*	*non_smokers*	*size*
1	24.94345054	1213	3650	4863
2	24.81324450	1229	3724	4953
3	23.41453594	1163	3804	4967
4	23.99176955	1166	3694	4860
5	24.68743378	1165	3554	4719
6	24.15117219	1195	3753	4948
7	23.54865086	1152	3740	4892
8	23.60905437	1116	3611	4727
9	24.71644612	1046	3186	4232
10	24.35597190	1040	3230	4270
11	24.09195402	1048	3302	4350
12	23.27227311	1118	3686	4804
13	23.59735974	1144	3704	4848
14	23.04147465	1100	3674	4774
15	23.06425041	1120	3736	4856
16	24.13793103	1141	3586	4727
17	24.29501085	1120	3490	4610
18	23.68421053	1170	3770	4940
19	23.16152685	1074	3563	4637
20	23.22264794	1091	3607	4698
21	24.34441462	1179	3664	4843
22	21.91107311	1025	3653	4678
23	23.89006342	1130	3600	4730
24	22.84420663	1110	3749	4859
25	23.17406862	1101	3650	4751
26	22.60887685	1085	3714	4799
27	24.17014510	1216	3815	5031
28	24.24677188	1183	3696	4879
29	23.48284960	1157	3770	4927
30	23.18745918	1065	3528	4593
31	23.76152691	1108	3555	4663
32	24.68268359	1089	3323	4412
33	22.52964427	1026	3528	4554
34	22.38772788	1007	3491	4498
35	23.49702709	1067	3474	4541
36	24.76149176	1142	3470	4612
37	22.88211564	1021	3441	4462
38	23.53846154	1071	3479	4550
39	21.56626506	1074	3906	4980
40	22.89288850	1043	3513	4556
41	22.53401361	1060	3644	4704
42	23.04469274	1155	3857	5012
43	22.17675941	1084	3804	4888
44	22.38805970	1020	3536	4556
45	22.94224190	1140	3829	4969
46	23.25386867	1112	3670	4782
47	22.90901673	1123	3779	4902
48	23.75049980	1188	3814	5002
49	23.11567916	1147	3815	4962
50	22.54090083	1116	3835	4951
51	24.75100657	1168	3551	4719
52	22.85775037	1075	3628	4703
53	23.73748610	1067	3428	4495

54	23.74864572	1096	3519	4615
55	23.25949367	1029	3395	4424
56	22.94450736	1013	3402	4415
57	23.24112394	1067	3524	4591
58	21.63742690	962	3484	4446
59	23.00155867	1033	3458	4491
60	22.33662534	1063	3696	4759
61	21.14260982	977	3644	4621
62	21.62595114	1080	3914	4994
63	21.60443168	1053	3821	4874
64	22.72149216	1072	3646	4718
65	22.89257936	1089	3668	4757
66	23.25399922	1192	3934	5126
67	20.83598557	982	3731	4713
68	23.03606993	1041	3478	4519
69	22.95472597	1156	3880	5036
70	22.86230430	1139	3843	4982
71	21.22186495	1056	3920	4976
72	21.73563680	1082	3896	4978
73	21.43000200	1070	3923	4993
74	22.51124584	1151	3962	5113
75	21.75214528	1090	3921	5011
76	21.85483871	1084	3876	4960
77	21.20045300	936	3479	4415
78	22.56572541	927	3181	4108
79	21.41468158	881	3233	4114
80	21.57550257	923	3355	4278
81	21.62162162	928	3364	4292
82	20.91943348	901	3406	4307
83	21.34986226	930	3426	4356
84	21.90431520	934	3330	4264
85	21.50201162	962	3512	4474
86	20.53189092	911	3526	4437
87	19.45721164	889	3680	4569
88	21.47847645	953	3484	4437
89	20.92968381	887	3351	4238
90	21.47976879	929	3396	4325
91	21.22930370	936	3473	4409
92	22.21706865	958	3354	4312
93	21.57978602	948	3445	4393
94	21.48645520	928	3391	4319
95	21.84716362	932	3334	4266
96	20.54541253	889	3438	4327
97	20.50973792	853	3306	4159
98	21.29446640	862	3186	4048
99	20.84444444	938	3562	4500
100	20.13698630	882	3498	4380
101	21.52777778	930	3390	4320
102	20.91356919	934	3532	4466
103	20.97740894	910	3428	4338
104	19.78307003	839	3402	4241
105	22.44521338	973	3362	4335
106	20.08784096	869	3457	4326
107	19.71046771	885	3605	4490
108	20.34652306	869	3402	4271
109	21.36894825	896	3297	4193
110	21.07581730	909	3404	4313
111	19.44945848	862	3570	4432
112	19.59855661	869	3565	4434

113	19.98094784	839	3360	4199
114	18.71897896	792	3439	4231
115	18.92682927	776	3324	4100
116	18.78632876	808	3493	4301
117	20.42314335	946	3686	4632
118	19.40228812	831	3452	4283
119	20.98535905	903	3400	4303
120	20.80019282	863	3286	4149
121	18.98402099	796	3397	4193
122	20.00921022	869	3474	4343
123	18.92069041	866	3711	4577
124	19.59508315	813	3336	4149
125	19.28251121	817	3420	4237
126	19.76942784	926	3758	4684
127	19.64535698	842	3444	4286
128	19.10112360	765	3240	4005
129	19.18533605	942	3968	4910
130	18.84805020	841	3621	4462
131	18.67732558	771	3357	4128
132	19.16391639	871	3674	4545
133	19.77463544	895	3631	4526
134	17.88990826	858	3938	4796
135	18.92667845	857	3671	4528
136	17.55585955	770	3616	4386
137	19.51438849	868	3580	4448
138	20.30728123	912	3579	4491
139	19.95428571	873	3502	4375
140	19.89013504	869	3500	4369
141	20.06721075	836	3330	4166
142	19.65442765	819	3348	4167
143	18.64698647	758	3307	4065
144	17.11991712	661	3200	3861
145	18.50992524	718	3161	3879
146	18.54858549	754	3311	4065
147	18.91836735	927	3973	4900
148	17.70941055	685	3183	3868
149	19.07164480	756	3208	3964
150	18.90675241	882	3783	4665
151	18.13725490	740	3340	4080
152	17.97893681	717	3271	3988
153	18.37016575	798	3546	4344
154	18.54285714	649	2851	3500
155	18.14715119	809	3649	4458
156	18.27784891	571	2553	3124
Total		**152,163**	**552,803**	**704,966**