Supplementary material to
"Re-analysing tobacco-industry funded research on the effect of plain packaging on minors in Australia: same data, different results"

Reverse-engineering Kaul and Wolf's figures to reconstruct the data on minors they used in their first working paper 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 paper on minors.[1] It is believed that the (original) method presented below has made it possible to reconstruct the data with 100% 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. The same procedure was used to reconstruct the data on adults in a previously published paper.

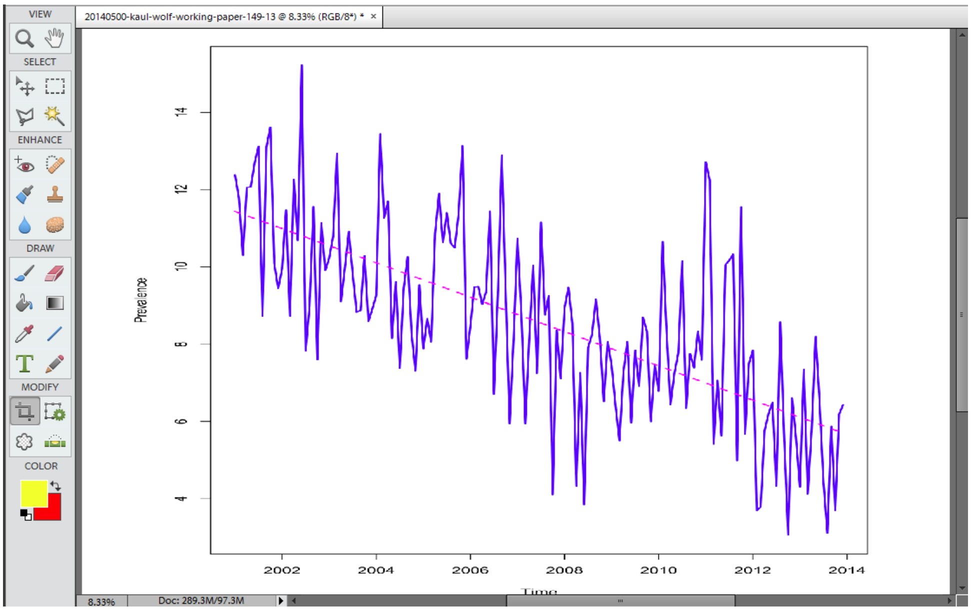
Step 1. Extracting the images from Kaul and Wolf's paper

Kaul and Wolf's working paper on minors is in PDF format and can be downloaded from the website of the University of Zürich. 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 (on page 12). 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`**.

Step 2. Pre-processing the images in Photoshop

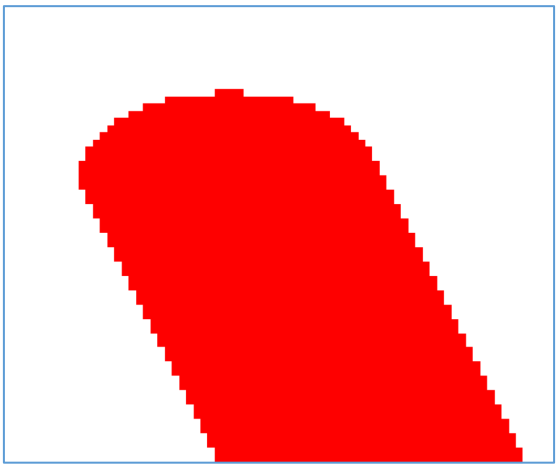
[1] Kaul A and Wolf M. The (Possible) Effect of Plain Packaging on the Smoking Prevalence of Minors in Australia: A Trend Analysis. University of Zurich Department of Economics Working Paper Series. May 2014; Available from http://www.econ.uzh.ch/static/workingpapers.php?id=828

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

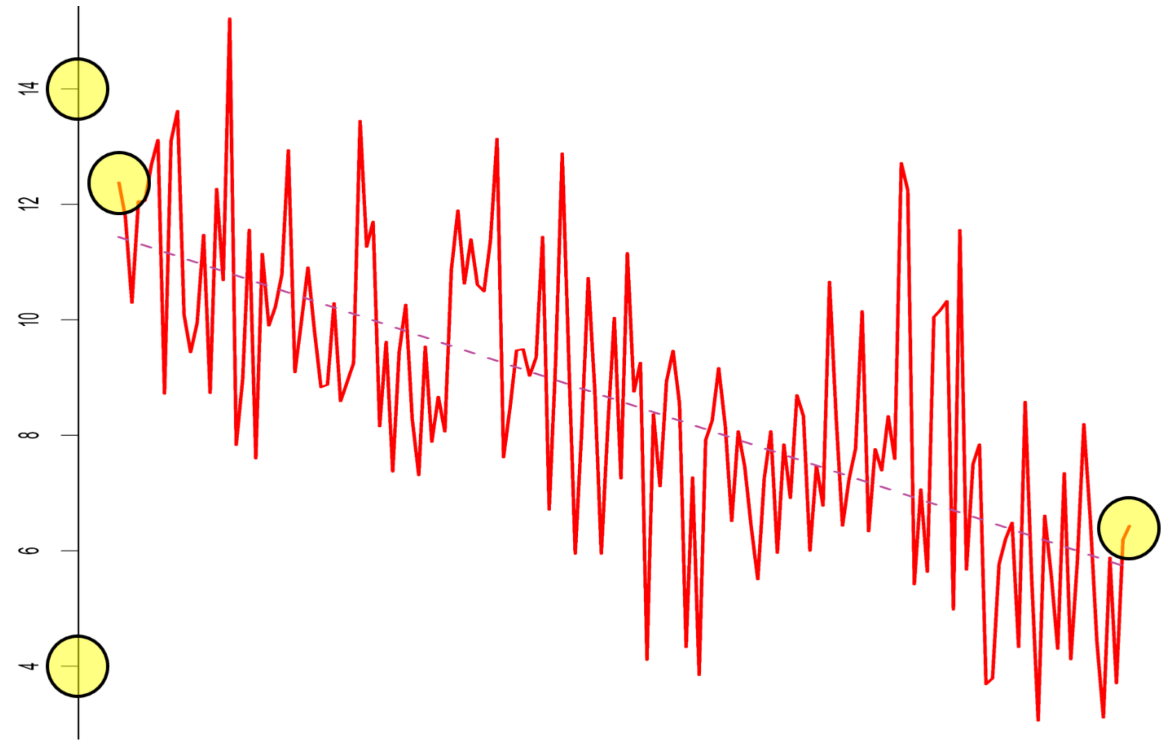


After

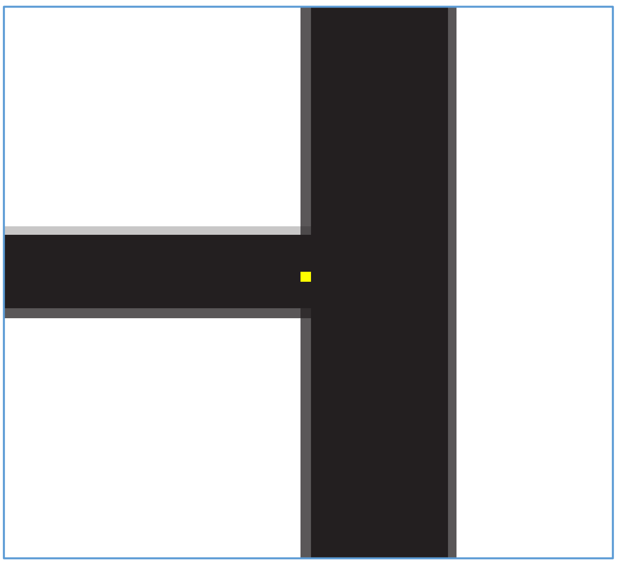
Before performing this step, we made sure pure red - colour rgb(255,0,0) - was not already used in the picture. The purpose of this step was to obtain a good contrast between the red line and its white background in order to facilitate the identification of the edge pixels of the line by the **`image2data.py`**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

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

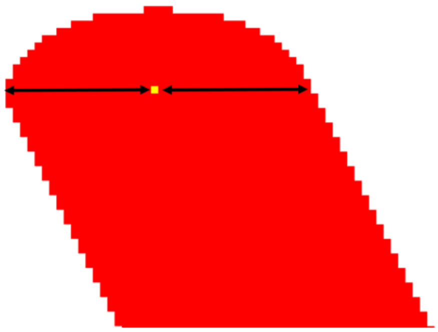
Still processing each figure in Photoshop, we also made that there was no pure yellow - rgb(255,255,0) – pixel in the image. We then painted a *single pure yellow pixel* at four particular places, as shown in the illustration below:



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`**.

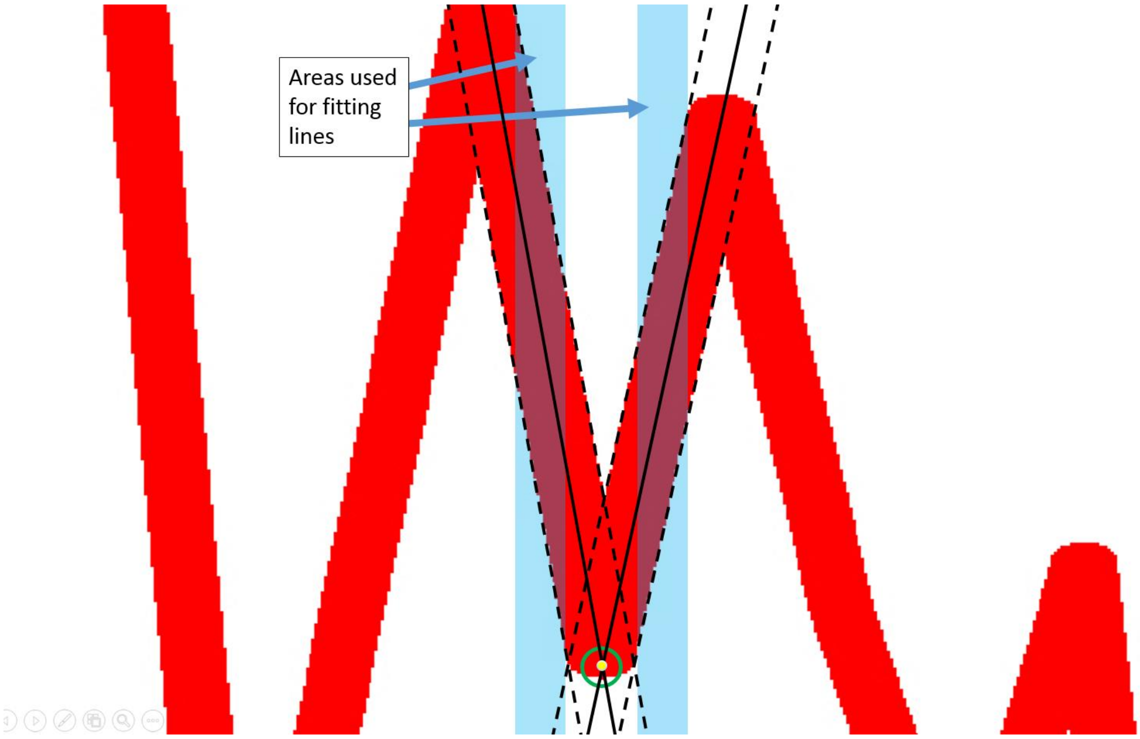
Step 4. Running Python program **`image2data.py`**

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.

Figure 1 (sample size): 200, 350, 1, 156, 156

Figure 2 (prevalence): 4, 14, 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).



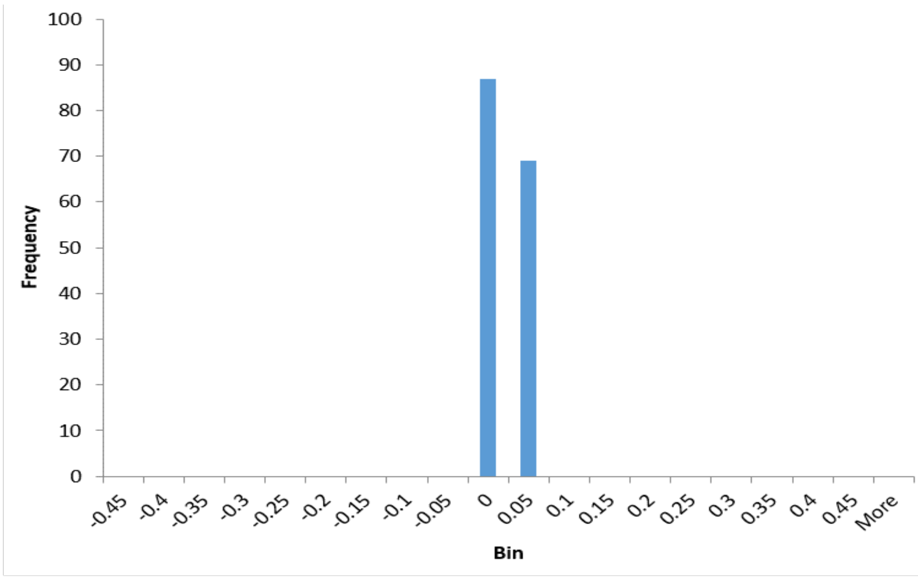
Using the Python program, we created two data files, **`sample-size.txt`** and **`prevalence.txt`** (in tab delimited text format), one containing the estimated values of sample sizes, the other containing the estimated values of observed monthly prevalence. These values were produced with high precision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

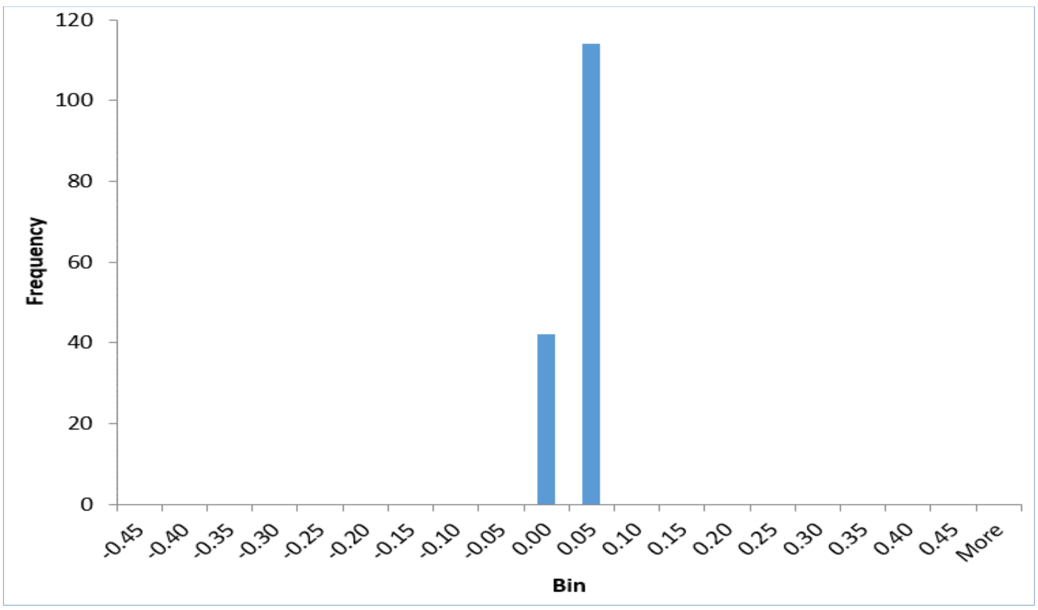
Sample size data: When working on Figure 1 (sample size), the y-coordinate of the data points obtained by the above method approximates the number of observations, which are *whole numbers*. We assumed that if our results were close to whole numbers, this indicated that the approximation was good and 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: all points are inside the range [-0.05, 0.05].

We rounded the values produced by the Python program to the nearest whole numbers, thus obtaining the reconstructed sample sizes.

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, if accurate, should be very close to a *whole number*. Therefore, 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:



One can see again that there was a concentration of this difference around zero: all points are inside the range [-0.05, 0.05]. This is indicative of a high level of accuracy of the prevalence estimates produced by the Python program.

From these prevalence estimates we obtained more accurate values by taking instead the numbers of smokers (whole numbers) which they imply divided by the sample sizes.

With this last operation, we assumed that we able to reconstruct the data with 100% accuracy. A few more checks were performed to validate this assumption.

Step 7. Further checks

We first observed that the sum of the monthly sample sizes which we reconstructed was identical to the total sample size indicated by Kaul and Wolf in their paper on minors (41,438). If there are errors in the reconstructed sample sizes, they must cancel each other out. This is very unlikely ($p < 0.01$): it would mean that an erroneous reconstructed sample size, which would be very close to a wrong whole number, would have to be cancelled by another reconstructed sample size similarly close to a wrong whole number, but with a difference in the opposite direction.

Secondly, using the reconstructed data, we were able to reproduce exactly Kaul and Wolf regression results presented in Table 1 of their paper:

(Intercept)	11.471^{***}
	(0.270)
t (Month)	-0.037^{***}
	(0.003)
R^2	0.46
Adjusted R^2	0.46
Sample size	156
Degrees of freedom	154

$^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$

Table 1: Regression output for the fitted model (3.2). The numbers in parentheses below the estimated coefficients are corresponding standard errors.

This is the output produced by R when running the same analysis using the reconstructed data:

```
Call:
lm(formula = prev ~ time, weights = size)

Weighted Residuals:
    Min      1Q  Median      3Q     Max
-64.072 -21.267   0.843  16.145  86.223

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 11.471386   0.269965   42.49   <2e-16 ***
time        -0.036978   0.003223  -11.47   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 29.45 on 154 degrees of freedom
Multiple R-squared:  0.4608,	Adjusted R-squared:  0.4573
F-statistic: 131.6 on 1 and 154 DF,  p-value: < 2.2e-16
```

This is a further indication that we were able to reconstruct the data used by Kaul and Wolf in their first working paper with 100% accuracy, or that, at any rate, performing the analysis with our reconstructed data will produce results which are undistinguishable from the results produced with the real data.

2017.08.29/pad

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

```
# -*- coding: utf-8 -*-
#
# Obtaining data by reverse engineering from published figures
#
# Author: Pascal Diethelm, 12.09.2016

PARM = [
 ['sample size', 'sample-size.png', 'sample-size.txt', 200, 350, 1, 156, 156],
 ['prevalence',  'prevalence.png',  'prevalence.txt',    4,  14, 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 first working paper (on minors) showing also indicator variables used in logistic regression

time	*prev*	*smokers*	*non.smokers*	*size*	*smoke.free*	*ghw*	*tax*	*pp*
1	12.3810	39	276	315	0	0	0	0
2	11.7825	39	292	331	0	0	0	0
3	10.2941	35	305	340	0	0	0	0
4	12.0548	44	321	365	0	0	0	0
5	12.0635	38	277	315	0	0	0	0
6	12.6935	41	282	323	0	0	0	0
7	13.1195	45	298	343	0	0	0	0
8	8.7209	30	314	344	0	0	0	0
9	13.0990	41	272	313	0	0	0	0
10	13.6201	38	241	279	0	0	0	0
11	10.0877	23	205	228	0	0	0	0
12	9.4395	32	307	339	0	0	0	0
13	9.9432	35	317	352	0	0	0	0
14	11.4754	42	324	366	0	0	0	0
15	8.7324	31	324	355	0	0	0	0
16	12.2699	40	286	326	0	0	0	0
17	10.6849	39	326	365	0	0	0	0
18	15.2231	58	323	381	0	0	0	0
19	7.8313	26	306	332	0	0	0	0
20	8.9888	32	324	356	0	0	0	0
21	11.5625	37	283	320	0	0	0	0
22	7.5988	25	304	329	0	0	0	0
23	11.1455	36	287	323	0	0	0	0
24	9.9010	30	273	303	0	0	0	0
25	10.2236	32	281	313	0	0	0	0
26	10.7843	33	273	306	0	0	0	0
27	12.9412	44	296	340	0	0	0	0
28	9.0909	29	290	319	0	0	0	0
29	9.9715	35	316	351	0	0	0	0
30	10.9091	36	294	330	0	0	0	0
31	9.8101	31	285	316	0	0	0	0
32	8.8328	28	289	317	0	0	0	0
33	8.8757	30	308	338	0	0	0	0
34	10.2894	32	279	311	0	0	0	0
35	8.5890	28	298	326	0	0	0	0
36	8.9172	28	286	314	0	0	0	0
37	9.2527	26	255	281	0	0	0	0
38	13.4483	39	251	290	0	0	0	0
39	11.2637	41	323	364	0	0	0	0

40	11.7021	33	249	282	0	0	0	0
41	8.1560	23	259	282	0	0	0	0
42	9.6210	33	310	343	0	0	0	0
43	7.3718	23	289	312	0	0	0	0
44	9.4463	29	278	307	0	0	0	0
45	10.2639	35	306	341	0	0	0	0
46	8.2781	25	277	302	0	0	0	0
47	7.3090	22	279	301	0	0	0	0
48	9.5385	31	294	325	0	0	0	0
49	7.8864	25	292	317	0	0	0	0
50	8.6687	28	295	323	0	0	0	0
51	8.0645	25	285	310	0	0	0	0
52	10.8696	35	287	322	0	0	0	0
53	11.8959	32	237	269	0	0	0	0
54	10.6250	34	286	320	0	0	0	0
55	11.3971	31	241	272	0	0	0	0
56	10.6061	28	236	264	0	0	0	0
57	10.4938	34	290	324	0	0	0	0
58	11.3793	33	257	290	0	0	0	0
59	13.1387	36	238	274	0	0	0	0
60	7.6190	24	291	315	0	0	0	0
61	8.4806	24	259	283	0.024	0	0	0
62	9.4708	34	325	359	0.024	0	0	0
63	9.4937	30	286	316	0.024	1	0	0
64	9.0301	27	272	299	0.024	1	0	0
65	9.3484	33	320	353	0.024	1	0	0
66	11.4379	35	271	306	0.024	1	0	0
67	6.7114	20	278	298	0.220	1	0	0
68	9.3023	28	273	301	0.320	1	0	0
69	12.8852	46	311	357	0.320	1	0	0
70	9.3458	30	291	321	0.320	1	0	0
71	5.9480	16	253	269	0.320	1	0	0
72	8.1081	24	272	296	0.336	1	0	0
73	10.7280	28	233	261	0.336	1	0	0
74	8.7786	23	239	262	0.336	1	0	0
75	5.9480	16	253	269	0.336	1	0	0
76	8.1633	24	270	294	0.336	1	0	0
77	10.0372	27	242	269	0.336	1	0	0
78	7.2519	19	243	262	0.336	1	0	0
79	11.1588	26	207	233	0.914	1	0	0
80	8.7558	19	198	217	0.914	1	0	0
81	9.2593	20	196	216	0.914	1	0	0
82	4.1096	9	210	219	0.914	1	0	0
83	8.3665	21	230	251	0.990	1	0	0
84	7.1146	18	235	253	0.990	1	0	0

85	8.9362	21	214	235	0.990	1	0	0
86	9.4650	23	220	243	0.990	1	0	0
87	8.5409	24	257	281	0.990	1	0	0
88	4.3290	10	221	231	0.990	1	0	0
89	7.2727	16	204	220	0.990	1	0	0
90	3.8462	8	200	208	0.990	1	0	0
91	7.9167	19	221	240	0.990	1	0	0
92	8.2524	17	189	206	0.990	1	0	0
93	9.1667	22	218	240	0.990	1	0	0
94	8.1731	17	191	208	0.990	1	0	0
95	6.5116	14	201	215	0.990	1	0	0
96	8.0717	18	205	223	0.990	1	0	0
97	7.4534	12	149	161	0.990	1	0	0
98	6.4039	13	190	203	0.990	1	0	0
99	5.5046	12	206	218	0.990	1	0	0
100	7.2650	17	217	234	0.990	1	0	0
101	8.0717	18	205	223	0.990	1	0	0
102	5.9633	13	205	218	0.990	1	0	0
103	7.8431	16	188	204	0.990	1	0	0
104	6.9124	15	202	217	0.990	1	0	0
105	8.6957	18	189	207	0.990	1	0	0
106	8.3333	17	187	204	0.990	1	0	0
107	6.0000	12	188	200	0.990	1	0	0
108	7.4766	16	198	214	0.990	1	0	0
109	6.7797	12	165	177	0.990	1	0	0
110	10.6599	21	176	197	0.990	1	0	0
111	8.3333	16	176	192	0.990	1	0	0
112	6.4327	11	160	171	0.990	1	0	0
113	7.2464	15	192	207	0.990	1	1	0
114	7.7720	15	178	193	0.990	1	1	0
115	10.1523	20	177	197	1	1	1	0
116	6.3348	14	207	221	1	1	1	0
117	7.7626	17	202	219	1	1	1	0
118	7.3913	17	213	230	1	1	1	0
119	8.3333	16	176	192	1	1	1	0
120	7.5893	17	207	224	1	1	1	0
121	12.7168	22	151	173	1	1	1	0
122	12.2449	24	172	196	1	1	1	0
123	5.4187	11	192	203	1	1	1	0
124	7.0652	13	171	184	1	1	1	0
125	5.6338	12	201	213	1	1	1	0
126	10.0478	21	188	209	1	1	1	0
127	10.1695	18	159	177	1	1	1	0
128	10.3261	19	165	184	1	1	1	0
129	4.9808	13	248	261	1	1	1	0

130	11.5578	23	176	199	1	1	1	0
131	5.6701	11	183	194	1	1	1	0
132	7.5000	15	185	200	1	1	1	0
133	7.8431	16	188	204	1	1	1	0
134	3.6885	9	235	244	1	1	1	0
135	3.7915	8	203	211	1	1	1	0
136	5.7692	12	196	208	1	1	1	0
137	6.2016	16	242	258	1	1	1	0
138	6.4885	17	245	262	1	1	1	0
139	4.3307	11	243	254	1	1	1	0
140	8.5837	20	213	233	1	1	1	0
141	5.5319	13	222	235	1	1	1	0
142	3.0568	7	222	229	1	1	1	0
143	6.6116	16	226	242	1	1	1	1
144	5.5794	13	220	233	1	1	1	1
145	4.3011	8	178	186	1	1	1	1
146	7.3469	18	227	245	1	1	1	1
147	4.1199	11	256	267	1	1	1	1
148	5.8036	13	211	224	1	1	1	1
149	8.1967	20	224	244	1	1	1	1
150	6.5385	17	243	260	1	1	1	1
151	4.4355	11	237	248	1	1	1	1
152	3.1088	6	187	193	1	1	1	1
153	5.8824	12	192	204	1	1	1	1
154	3.7037	7	182	189	1	1	1	1
155	6.1905	13	197	210	1	1	1	1
156	6.4327	11	160	171	1	1	1	1