Will Millennials Ever Get Married?

Allen B. Downey[‡][*]

https://www.youtube.com/watch?v=XHYFNraQEEo

Abstract—Using data from the National Survey of Family Growth (NSFG), we investigate marriage patterns among women in the United States We describe and predict age at first marriage for successive generations based on decade of birth. The fraction of women married by age 22 has dropped by 11 percentage points per decade, from 69% for women born in the 1940s to 13% for women born in the 90s. The fraction of women married by age 42 fell more slowly, from 93% for women born in the 40s to 82% for women born in the 70s. Projections suggest that this fraction will be substantially lower for later generations, between 68% and 72%. Along with these results, this paper presents an introduction to survival analysis methods and an implementation in Python.

Keywords—Survival analysis, marriage patterns, Python.

Introduction

A recent study from the Pew Research Center [Wan14] reports that the fraction of adults in the U.S. who have never married is increasing. Between 1960 and 2012, the fraction of men 25 and older who had never married increased from 10% to 23%. The corresponding fraction of women increased from 8% to 17%. The Pew study focuses on the causes of these trends, but does not address this question: is the fraction of people who never marry increasing, are people marrying later, or both? That is the subject of this paper.

To answer this question, we apply tools of survival analysis to data from the National Survey of Family Growth (NSFG). Since 1973 the U.S. Centers for Disease Control and Prevention (CDC) have conducted this survey, intended to gather "information on family life, marriage and divorce, pregnancy, infertility, use of contraception, and men's and women's health." See http://cdc.gov/nchs/nsfg.htm.

NSFG data is organized in cycles; during each cycle several thousand respondents were interviewed, including women ages 14–44. Men were included starting with Cycle 6 in 2002, but for this study we use only data from female respondents.

Table 1 shows the interview dates for each cycle, the number of respondents, and the birth years of the respondents. We did not use data from Cycles 1 and 2 because they included only married women. The total sample size for this study is 52 789.

Cycle	Interview dates	Number of respondents	Birth years
3	1982–83	7 969	1937–68
4	1988–88	8 450	1943–73
5	1995	10 847	1950–80
6	2002–03	7 643	1957–88
7	2006–10	12 279	1961–95
8	2011–13	5 601	1966–98

TABLE 1: *NSFG Survey Cycles*

For each respondent we have date of birth (year and month), date of interview, and date of first marriage, if applicable. So we can compute with resolution of one month each respondent's age at interview, `age`, and age at first marriage, `agemarry`.

To study changes in marriage patterns over time, we group the respondents into cohorts by decade of birth. For each cohort, Table 2 reports the number of respondents, range of ages when they were interviewed, number who had been married at least once at time of interview, and the number of married respondents whose date of marriage was not ascertained.

Cohort 30 includes women born in the 1930s, and so on for the other cohorts. One goal of this paper is to describe and predict marriage patterns for the Millennial Generation, defined here to include women born in the 1980s and 90s.

Another goal of this paper is to present survival analysis and its implementation in Python to an audience that may not be familiar with it. We also describe the resampling methods we use to deal with the stratified sampling design of the NSFG.

The code and data for this project are available in a public Git repository at https://github.com/AllenDowney/MarriageNSFG.

Cohort	Number of respondents	Age at interview	Number married	Number with missing data
30	325	42–44	310	0
40	3 608	32–44	3275	0
50	10 631	22–44	8658	10
60	14 484	15–44	8421	27
70	12 083	14–43	5908	25
80	8 536	14–33	2203	8
90	3 122	15–23	93	0

TABLE 2: *NSFG Birth Cohorts*

* *Corresponding author: allen.downey@olin.edu*
‡ *Olin College of Engineering*

Methodology

Survival analysis

Survival analysis is a powerful set of tools with applications in many domains, but it is often considered a specialized topic.

Survival analysis is used to study and predict the time until an event: in medicine, the event might be the death of a patient, hence "survival"; but more generally we might be interested in the time until failure of a mechanical part, the lifetimes of civilizations, species, or stars; or in this study the time from birth until first marriage.

The result of survival analysis is often a **survival function**, which shows the fraction of the population that survives after t, for any time, t. If T is a random variable that represents the time until an event, the survival function, $S(t)$, is the probability that T exceeds t:

$$S(t) \equiv \Pr(T > t)$$

If the distribution of T is known, or can be estimated from a representative sample, computing $S(t)$ is simple: it is the complement of the cumulative distribution function (CDF):

$$S(t) = 1 - \mathrm{CDF}_T(t)$$

In Python we can compute the survival function like this:

```
from collections import Counter
import numpy as np

def MakeSurvivalFunction(values):
    counter = Counter(values)
    ts, fs = zip(*sorted(counter.items()))
    ts = np.asarray(ts)
    ps = np.cumsum(fs, dtype=np.float)
    ps /= ps[-1]
    ss = 1 - ps
    return SurvivalFunction(ts, ss)
```

`values` is a sequence of observed lifetimes. Counter makes a map from each unique value to the number of times it appears, which we split into a sorted sequence of times, `ts`, and their frequencies, `fs`.

We convert ts to a NumPy array [Wal11]. Then `ps` is the cumulative sum of the frequencies, normalized to go from 0 to 1, so it represents the CDF of the observed values. `ss`, which is the complement of `ps`, is the survival function.

`SurvivalFunction` is defined in `marriage.py`, a Python module we wrote for this project.

Given a survival curve, we can compute the **hazard function**, which is the instantaneous death rate at time t; that is, the fraction of people who survive until time t and then die at time t. When t is continuous, the hazard function, $\lambda(t)$, is

$$\lambda(t) = -S'(t)/S(t)$$

Where $S'(t)$ is the derivative of $S(t)$. Since the survival function decreases monotonically, its derivative is nonpositive, so the hazard function is nonnegative.

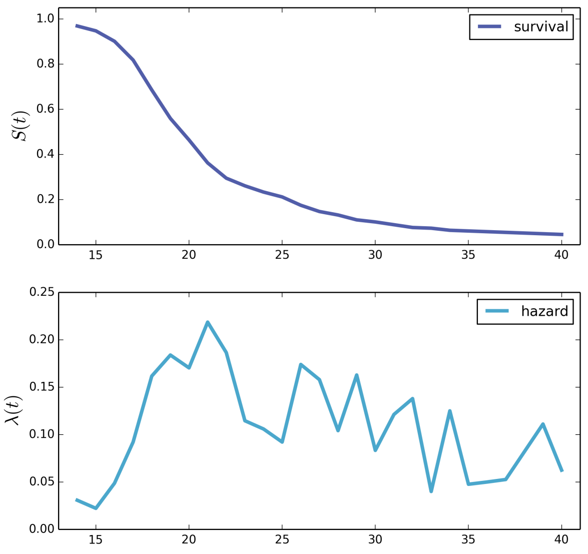

Fig. 1: *Survival and hazard functions for 1930s cohort.*

With a survival function represented by discrete `ts` and `ss`, we can compute the hazard function like this:

```
import pandas as pd

# class SurvivalFunction
def MakeHazardFunction(self):
    lams = pd.Series(index=self.ts)
    prev = 1.0
    for t, s in zip(self.ts, self.ss):
        lams[t] = (prev - s) / prev
        prev = s
    return HazardFunction(lams)
```

`MakeHazardFunction` is a method of `SurvivalFunction`, which provides attributes `ts` and `ss`. The result, `lams`, is a Pandas Series [McK10] object that maps from the same set of `ts` to the estimated hazard function, $\lambda(t)$.

Figure 1 shows the survival and hazard functions for women born in the 1930s. These women were interviewed when they were 42–44 years old. At that point more than 95% of them had been married; for the others we set age at marriage to infinity (`np.inf`). In this cohort, the hazard function is highest at ages 18–22, and lower as age increases.

This example demonstrates the simple case, where the respondents are the same age and most events are complete. But for most applications of survival analysis, the sample also includes incomplete events. For example, the 1960s cohort includes women from ages 14–44; for the ones that are not married, we don't know when they will marry, if ever. These missing data are said to be "censored".

It might be tempting to ignore unmarried women and compute the survival function for women whose ages at marriage are known. But that would discard useful information and seriously bias the results.

For women who are not married yet, their age at interview is a lower bound on their age at marriage. We can use both groups to estimate the hazard function, then compute the survival function. One common way to do that is Kaplan-Meier estimation.

The fundamental idea is that at each time, t, we know the number of events that occurred and the number of respondents who were "at risk"; that is, known to to be unmarried. The ratio of these factors estimates the hazard function.

Initially, the entire sample is considered at risk. At each time step, we subtract people who got married at age t as well as people who were interviewed at age t (and therefore no longer in the observation pool at the next time step). The following function implements this algorithm:

```python
def EstimateHazardFunction(complete, ongoing):
    hist_complete = Counter(complete)
    hist_ongoing = Counter(ongoing)

    ts = list(hist_complete | hist_ongoing)
    ts.sort()

    at_risk = len(complete) + len(ongoing)

    lams = pd.Series(index=ts)
    for t in ts:
        ended = hist_complete[t]
        censored = hist_ongoing[t]

        lams[t] = ended / at_risk
        at_risk -= ended + censored

    return HazardFunction(lams)
```

`complete` is a sequence of lifetimes for complete events, in this case age at marriage. `ongoing` is a sequence of lower bounds for incomplete observations, in this case age at interview.

`hist_complete` counts how many respondents were married at each age; `hist_ongoing` counts how many unmarried respondents were interviewed at each age.

`ts` is a sorted list of observation times, which is the union of unique values from complete and ongoing.

`at_risk` is the number of respondents at risk; initially it is the total number of respondents.

`lams` is a Pandas Series that maps from each observation time to the estimated hazard rate.

For each value of `t` we look up `ended`, which is the number of people married for the first time at `t`, and `censored`, which is the number of never married people interviewed at `t`. The estimated hazard function at `t` is the ratio of `ended` and `at_risk`.

At the end of each time step, we update `at_risk` by subtracting off `ended` and `censored`.

The result is a HazardFunction object that contains the Series `lams` and provides methods to access it.

With this estimated HazardFunction, we can compute the SurvivalFunction. The hazard function, $\lambda(t)$, is the probability of ending at time t conditioned on surviving until t. Therefore, the probability of surviving until t is the cumulative product of the complementary hazard function:

$$S(t) = \prod_{t_i < t} [1 - \lambda(t_i)]$$

Here's the Python implementation:

```python
# class HazardFunction
def MakeSurvival(self):
    series = (1 - self.series).cumprod()
    ts = series.index.values
    ss = series.values
    return SurvivalFunction(ts, ss)
```

We wrote our own implementation of these methods in order to demonstrate the methodology, and also to make them work efficiently with the resampling methods described in the next section. But Kaplan-Meier estimation and other survival analysis algorithms are also available in a Python package called Lifelines [Dav15].

Resampling

The NSFG is intended to be representative of the adult U.S. population, but it uses stratified sampling to systematically oversample certain subpopulations, including teenagers and racial minorities. Our analysis takes this design into account to generate results that are representative of the population.

As an example of stratified sampling, suppose there are 10 000 people in the population you are studying, and you sample 100. Each person in the sample represents 100 people in the population, so each respondent has the same "sampling weight".

Now suppose there are two subgroups, a minority of 1 000 people and a majority of 9 000. A sample of 100 people will have 10 members of the minority group, on average, which might not be enough for reliable statistical inference.

In a stratified sample, you might survey 40 people from the minority group and only 60 from the majority group. This design improves some statistical properties of the sample, but it changes the weight associated with each respondent. Each of the 40 minorities represents $1000/40 = 25$ people in the population, while each of the 60 others represents $9000/60 = 150$ people. In general, respondents from oversampled groups have lower weights.

The NSFG includes a computed weight for each respondent, which indicates how many people in the U.S. population she represents. Some statistical methods, like regression, can be extended to take these weights into account, but in general it is not easy.

However, bootstrapping provides a simple and effective approach. The idea behind bootstrapping is to use the actual sample as a model of the population, then simulate the results of additional experiments by drawing new samples (with replacement) from the actual sample.

With stratified sampling, we can modify the bootstrap process to take sampling weights into account. The following function performs weighted resampling on the NSFG data:

```
import thinkstats2

def ResampleRowsWeighted(df):
    weights = df.finalwgt
    cdf = thinkstats2.Cdf(dict(weights))
    indices = cdf.Sample(len(weights))
    sample = df.loc[indices]
    return sample
```

`df` is a Pandas DataFrame with one row per respondent; it includes a column that contains sampling weights, called `finalwgt`.

`weights` is a Series that maps from respondent index to sampling weight. `cdf` represents a cumulative distribution function that maps from each index to its cumulative probability. The Cdf class is provided by `thinkstats2.py`, a module that accompanies the second edition of *Think Stats* [Dow14]. We use it here because it provides an efficient implementation of random sampling from an arbitrary distribution.

`Sample` generates a random sample of indices based on the sampling weights. The return value, `sample`, is a Pandas DataFrame that contains the selected rows. Since the sample is generated with replacement, some respondents might appear more than once; others might not appear at all.

After resampling, we jitter the data by adding Gaussian noise (mean 0, standard deviation 1 year) to each respondent's age at interview and age at marriage. Jittering contributes some smoothing, which makes the figures easier to interpret, and some robustness, making the results less prone to the effect of a small number of idiosyncratic data points.

Jittering also makes sense in the context of bootstrapping. Each respondent in the sample represents several thousand people in the population; it is reasonable to assume that there is variation within each represented subgroup.

Finally, we discretize age at interview and age at marriage, rounding down to integer values.

Results

Figure 2 shows the estimated survival curve for each cohort (we omit the 1930s cohort because it only includes people born after 1936, so it is not representative of the decade). The lines show the median of 101 resampling runs; the gray regions show 90% confidence intervals.

Two trends are apparent in this figure: women are getting married later, and the fraction of women who remain unmarried is increasing.

Table 3 shows the percentage of married women in each cohort at ages 22, 32, and 42 (which are the last observed ages for cohorts 90, 80, and 70).

Two features of this data are striking:

- By age 22, only 13% of the 90s cohort have been married, contrasted with 69% of the 40s cohort. Between these cohorts, the fraction of women married by age 22 dropped more than 11 percentage points per decade.
- By age 32, only 60% of the 80s cohort is married, and their survival curve seems to have gone flat. In this cohort, 259 were at risk at age 30, and only 9 were married that year; 155 were at risk at age 31, and none were married; 63 were are risk at age 32, and again none were married. These low hazard rates are strange, but they are based on sample sizes large enough that it is hard to dismiss them.

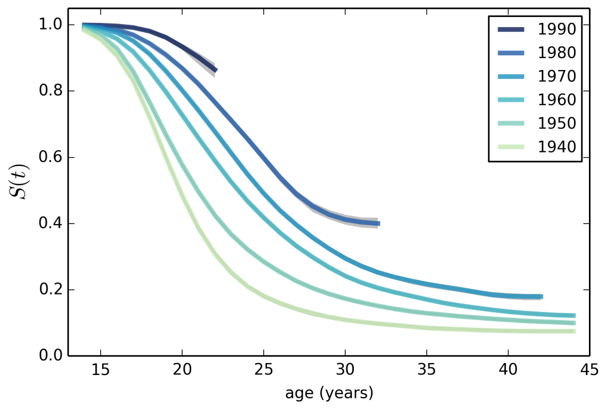

Fig. 2: *Survival functions by birth cohort.*

Cohort	% married by age		
	22	32	42
40	69	90	92
50	57	85	90
60	41	79	87
70	32	75	82
80	23	60	–
90	13	–	–

TABLE 3: *Marriage rates by birth cohort and age.*

Projection

Predicting these kinds of social trends is nearly futile. We can use current trends to generate projections, but in general there is no way to know which trends will continue and which will decrease or reverse.

As we saw in the previous section, the 80s cohort seems to be on strike, with unprecedented low marriage rates in their early thirties. Visual extrapolation of their survival curve suggests that 40% of them will remain unmarried, more than double the fraction of previous generations.

At the same time the number of women getting married at ages 35–45 has been increasing for several generations, so we might expect that trend to continue. In that case the gap between the 80s and 70s cohorts would close.

These prediction methods provide a rough upper and lower bound on what we might expect. A middle ground is to assume that the hazard function from the previous generation will apply to the next.

This method predicts higher marriage rates than extrapolating the survival curves because it takes into account the structure of the model: because fewer women married young, more are at risk at later ages, so we expect more late marriages.

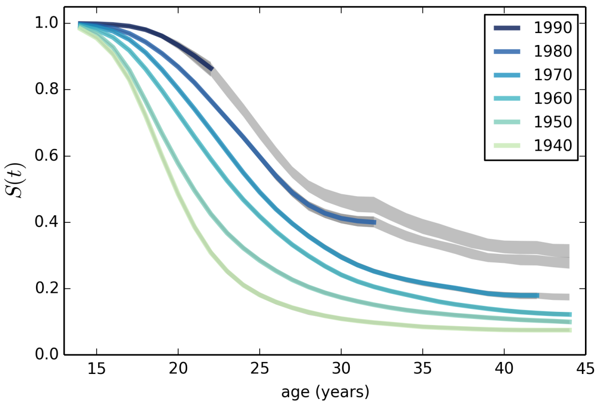

Fig. 3: *Survival functions with projections.*

To make these projections, we extend each HazardFunction using data from the previous cohort:

```
# class HazardFunction
def Extend(self, other):
    last_t = self.series.index[-1]
    other_ts = other.series.index
    hs = other.series[other_ts > last_t]
    self.series = pd.concat([self.series, hs])
```

Then we convert the extended hazard functions to survival functions using `HazardFunction.MakeSurvival`.

Figure 3 shows the results. Again, the gray regions show 90% confidence intervals. For the 80s cohort, the median projection is that 72% will marry by age 42, down from 82% in the previous cohort.

For the 90s cohort, the median projection is that only 68% will marry by age 42. This projection assumes that this cohort will also go on a “marriage strike” in their early thirties, but this event might not be repeated.

Discussion

The previous section addresses the title question of this paper, "Will Millennials Ever Get Married?" Our projections suggest that the fraction still unmarried at age 42 will be higher than in previous generations, by about 10 percentage points, unless there is a substantial increase in the hazard rate after age 30.

We also investigate how much of the change in marriage rates is driven by two factors: people getting married later, or never getting married at all. Up through the 70s cohort, people were getting married later, but the fraction who never married was increasing only slowly. Among Millennials (women born in the 80s and 90s), the fraction of people marrying young is continuing to fall, but we also see indications that the fraction of people who never marry is increasing more quickly.

Future work

This work is preliminary, and there are many avenues for future investigation:

- The NSFG includes data from male respondents, starting with Cycle 6 in 2002. We plan to repeat our analysis for these men.
- There are many subgroups in the U.S. that would be interesting to explore, including different regions, education and income levels, racial and religious groups.
- We have data from the Canadian General Social Survey, which will allow us to compare marriage patterns between countries (see http://tinyurl.com/canadagss).
- We are interested in finding similar data from other countries.

Acknowledgment

Many thanks to Lindsey Vanderlyn for help with data acquisition, preparation, and analysis. And thanks to the SciPy reviewers who made many helpful suggestions.

References

[Dow14] Allen Downey, *Think Stats: Exploratory Data Analysis*, 2nd edition, O’Reilly Media, October 2014. http://thinkstats2.com

[Dav15] Cameron Davidson-Pilon, Lifelines, (2015), Github repository, https://github.com/CamDavidsonPilon/lifelines

[McK10] Wes McKinney. "Data Structures for Statistical Computing in Python", *Proceedings of the 9th Python in Science Conference*, 51-56 (2010) http://pandas.pydata.org.

[Wal11] Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation", *Computing in Science & Engineering*, 13, 22-30 (2011) http://www.numpy.org

[Wan14] Wendy Wang and Kim Parker, “Record Share of Americans Have Never Married”, Washington D.C.: Pew Research Center’s Social and Demographic Trends project, September 2014. http://tinyurl.com/wang14pew