Exercise 11-1.

Suppose one of your co-workers is expecting a baby and you are participating in an office pool to predict the date of birth. Assuming that bets are placed during the 30th week of pregnancy, what variables could you use to make the best prediction? You should limit yourself to variables that are known before the birth,which variables are best to predict the baby weight

In [1]:

```
# setup the data frame and import the packages

import sys
import numpy as np
import math
import pandas as pd

import first
import thinkplot
import thinkstats2
from scipy import stats
import statsmodels.formula.api as smf

#read data and build dataframe
live, firsts, others = first.MakeFrames()
live = live.dropna(subset=['agepreg', 'totalwgt_lb'])

#read data and build dataframe
live, firsts, others = first.MakeFrames()
#live = live.dropna(subset=['agepreg', 'totalwgt_lb'])
```

```
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:68: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work because the i
ntermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col:
value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operati
on inplace on the original object.


  df.birthwgt_lb.replace(na_vals, np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:69: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work because the i
ntermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col:
value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operati
on inplace on the original object.


  df.birthwgt_oz.replace(na_vals, np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:70: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work because the i
ntermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col:
value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operati
on inplace on the original object.


  df.hpagelb.replace(na_vals, np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:72: FutureWarning: A value is trying
```

```
to be set on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work because the i
ntermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col:
value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operati
on inplace on the original object.


  df.babysex.replace([7, 9], np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:73: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work because the i
ntermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col:
value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operati
on inplace on the original object.


  df.nbrnaliv.replace([9], np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:68: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work because the i
ntermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col:
value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operati
on inplace on the original object.


  df.birthwgt_lb.replace(na_vals, np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:69: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work because the i
ntermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col:
value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operati
on inplace on the original object.


  df.birthwgt_oz.replace(na_vals, np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:70: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work because the i
ntermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col:
value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operati
on inplace on the original object.


  df.hpagelb.replace(na_vals, np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:72: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work because the i
ntermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col:
value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operati
on inplace on the original object.


  df.babysex.replace([7, 9], np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:73: FutureWarning: A value is trying
```

```
to be set on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work because the i
ntermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col:
value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operati
on inplace on the original object.


  df.nbrnaliv.replace([9], np.nan, inplace=True)
```

In [2]:

```
#data mining
import nsfg

live = live[live.prglngth>30]
resp = nsfg.ReadFemResp()
resp.index = resp.caseid
join = live.join(resp, on='caseid', rsuffix='_r')
join.shape
```

Out[2]:

```
(8884, 3331)
```

In [3]:

```
#this function will try out all variables and find coresponding R2 for each variables.
# the variable for baby birth date is cmbabdob (Century Month for baby's or babies' date
of birth (delivery date))
import patsy

def GoMining(df):
    variables = []
    for name in df.columns:
        try:
            if df[name].var() < 1e-7:
                continue

            formula = 'cmbabdob ~ agepreg + ' + name
            model = smf.ols(formula, data=df)
            if model.nobs < len(df)/2:
                continue

            results = model.fit()
        except (ValueError, TypeError, patsy.PatsyError) as e:
            continue

        variables.append((results.rsquared, name))

    return variables

variables = GoMining(join)
```

In [4]:

```
# this is the function to rank and pick out the top 20 variables with largest R2 values
import re

def ReadVariables():
    """Reads Stata dictionary files for NSFG data.

    returns: DataFrame that maps variables names to descriptions
    """
    vars1 = thinkstats2.ReadStataDct('2002FemPreg.dct').variables
    vars2 = thinkstats2.ReadStataDct('2002FemResp.dct').variables

    all_vars = pd.concat([vars1, vars2])
    all_vars.index = all_vars.name
    return all_vars
```

```
def MiningReport(variables, n=20):
    """Prints variables with the highest R^2.

    t: list of (R^2, variable name) pairs
    n: number of pairs to print
    """
    all_vars = ReadVariables()

    variables.sort(reverse=True)
    for r2, name in variables[:n]:
        key = re.sub('_r$', '', name)
        try:
            desc = all_vars.loc[key].desc
            if isinstance(desc, pd.Series):
                desc = desc[0]
            print(name, r2, desc)
        except (KeyError, IndexError):
            print(name, r2)
```

In [5]:

```
MiningReport(variables)
```

```
cmprgend 1.0 CM FOR PREGNANCY END DATE (REGARDLESS OF OUTCOME)
cmbabdob 1.0 CM FOR BABY'S OR BABIES' DATE OF BIRTH (DELIVERY DATE)
maternlv_i 0.9644611042675953 MATERNLV IMPUTATION FLAG
cmintfin 0.9345564585496872 CM FOR DATE OF END OF PREGNANCY INTERVAL
datend_i 0.9143157249464409 DATEND IMPUTATION FLAG
datecon_i 0.8666324562960489 DATECON IMPUTATION FLAG
frsteatd 0.8126056954505722 AGE (IN MOS) WHEN 1ST SUPPLEMENTED - 1ST FROM THIS PREG
pncarewk_i 0.6920031959762878 PNCAREWK IMPUTATION FLAG
cmprgbeg 0.6331523928948615 CM FOR PREGNANCY START DATE
cmfstprg_r 0.5292820118493895 CM FOR R'S FIRST COMPLETED PREGNANCY
cmfstprg 0.5292820118493895 CM FOR R'S FIRST COMPLETED PREGNANCY
cmlastlb_r 0.45688384405815197 CM FOR R'S MOST RECENT LIVE BIRTH
cmlastlb 0.45688384405815197 CM FOR R'S MOST RECENT LIVE BIRTH
agepreg_i 0.4014168719339821 AGEPREG IMPUTATION FLAG
agecon_i 0.3911090825296043 AGECON IMPUTATION FLAG
bfeedwks_i 0.2702522130679593 BFEEDWKS IMPUTATION FLAG
cmhsgrad 0.2585764653626096 CENTURY MONTH OF HIGH SCHOOL GRADUATION
kidage 0.16737748339080993 CURRENT AGE (IN MOS) OF R'S CHILD(REN) FROM THIS PREGNANCY
cmlstprg_r 0.16299057868186884 CM FOR R'S MOST RECENT COMPLETED PREGNANCY
cmlstprg 0.16299057868186884 CM FOR R'S MOST RECENT COMPLETED PREGNANCY
```

```
C:\Users\gyanr\AppData\Local\Temp\ipykernel_10044\2441321277.py:30: FutureWarning: Series
.__getitem__ treating keys as positions is deprecated. In a future version, integer keys
will always be treated as labels (consistent with DataFrame behavior). To access a value
by position, use `ser.iloc[pos]`
  desc = desc[0]
```

Conclusion:

From the list above, need to keep those variables only know before brith and known to that coworkers (assume this could be first baby, the couples already know the sex of baby ahead, do not know if the pregency will finish).

Besides age of pregnency, The following variables will be picked to make date of birth prediction.

1. **agecon 0.10203149928156052 AGE AT TIME OF CONCEPTION**
2. **race_r 0.016199503586252995 RACE**
3. **race 0.016199503586252995 RACE**
4. **paydu 0.014003795578114597 IB-10 CURRENT LIVING QUARTERS OWNED/RENTED, ETC**
5. **totincr 0.011870069031173602 TOTAL INCOME OF R'S FAMILY**
6. **marcon03 0.011752599354395654 FORMAL MARITAL STATUS WHEN PREGNANCY BEGAN - 3RD**
7. **cebow 0.011437770919637158 NUMBER OF CHILDREN BORN OUT OF WEDLOCK**

Example 11-3.

If the quantity you want to predict is a count, you can use Poisson regression, which is implemented in StatsModels with a function called poisson. It works the same way as ols and logit. As an exercise, let's use it to predict how many children a woman has born; in the NSFG dataset, this variable is called numbabes.

Suppose you meet a woman who is 35 years old, black, and a college graduate whose annual household income exceeds $75,000. How many children would you predict she has born?

based on the condition, we need to select independent variables as age and square age (age_r + age2), race(race), income (totincr) and eductaion(educat) to make prediction.

In [6]:

```
join.numbabes.replace([97], np.nan, inplace=True)
join['age2'] = join.age_r**2

formula = 'numbabes ~ age_r + age2 + C(race) + totincr + educat'
model = smf.poisson(formula, data=join)
results = model.fit()
results.summary()
```

```
Optimization terminated successfully.
         Current function value: 1.677002
         Iterations 7
```

```
C:\Users\gyanr\AppData\Local\Temp\ipykernel_10044\2591200501.py:1: FutureWarning: A value
is trying to be set on a copy of a DataFrame or Series through chained assignment using a
n inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the i
ntermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col:
value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operati
on inplace on the original object.


  join.numbabes.replace([97], np.nan, inplace=True)
```

Out[6]:

Poisson Regression Results

Dep. Variable:	numbabes	**No. Observations:**	8884
Model:	Poisson	**Df Residuals:**	8877
Method:	MLE	**Df Model:**	6
Date:	Thu, 28 Nov 2024	**Pseudo R-squ.:**	0.03686
Time:	22:00:47	**Log-Likelihood:**	-14898.
converged:	True	**LL-Null:**	-15469.
Covariance Type:	nonrobust	**LLR p-value:**	3.681e-243

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-1.0324	0.169	-6.098	0.000	-1.364	-0.701
C(race)[T.2]	-0.1401	0.015	-9.479	0.000	-0.169	-0.111
C(race)[T.3]	-0.0991	0.025	-4.029	0.000	-0.147	-0.051
age_r	0.1556	0.010	15.006	0.000	0.135	0.176
age2	-0.0020	0.000	-13.102	0.000	-0.002	-0.002
totincr	-0.0187	0.002	-9.830	0.000	-0.022	-0.015
educat	-0.0471	0.003	-16.076	0.000	-0.053	-0.041

In [7]:

```
columns = ['age_r', 'age2', 'race', 'totincr', 'educat']
```

```
columns = ['age_r', 'age2', 'race', 'totincr', 'educat']
new = pd.DataFrame([[35, 35**2, 1, 14, 16]], columns=columns)
results.predict(new)
```

Out[7]:

```
0    2.496802
dtype: float64
```

Conclusion:

From the model, it predict this woman will give birth to 2-3 babies.

Example 11-4.

f the quantity you want to predict is categorical, you can use multinomial logistic regression, which is implemented in StatsModels with a function called mnlogit. As an exercise, let's use it to guess whether a woman is married, cohabitating, widowed, divorced, separated, or never married; in the NSFG dataset, marital status is encoded in a variable called rmarital.

Suppose you meet a woman who is 25 years old, white, and a high school graduate whose annual household income is about $45,000. What is the probability that she is married, cohabitating, etc?

based on the condition, we need to select independent variables as age and square age (age_r + age2), race(race), income (totincr) and eductaion(educat) to make prediction.

In [8]:

```
formula='rmarital ~ age_r + age2 + C(race) + totincr + educat'
model = smf.mnlogit(formula, data=join)
results = model.fit()
results.summary()
```

```
Optimization terminated successfully.
         Current function value: 1.084053
         Iterations 8
```

Out[8]:

MNLogit Regression Results

Dep. Variable:	rmarital	**No. Observations:**	8884
Model:	MNLogit	**Df Residuals:**	8849
Method:	MLE	**Df Model:**	30
Date:	Thu, 28 Nov 2024	**Pseudo R-squ.:**	0.1682
Time:	22:00:49	**Log-Likelihood:**	-9630.7
converged:	True	**LL-Null:**	-11579.
Covariance Type:	nonrobust	**LLR p-value:**	0.000

rmarital=2	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	9.0156	0.805	11.199	0.000	7.438	10.593
C(race)[T.2]	-0.9237	0.089	-10.418	0.000	-1.097	-0.750
C(race)[T.3]	-0.6179	0.136	-4.536	0.000	-0.885	-0.351
age_r	-0.3635	0.051	-7.150	0.000	-0.463	-0.264
age2	0.0048	0.001	6.103	0.000	0.003	0.006
totincr	-0.1310	0.012	-11.337	0.000	-0.154	-0.108
educat	-0.1953	0.019	-10.424	0.000	-0.232	-0.159
rmarital=3	**coef**	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	2.9570	3.020	0.979	0.328	-2.963	8.877

C(race)[T.2]	-0.4411	0.237	-1.863	0.062	-0.905	0.023
C(race)[T.3]	0.0591	0.336	0.176	0.860	-0.600	0.718
age_r	-0.3177	0.177	-1.798	0.072	-0.664	0.029
age2	0.0064	0.003	2.528	0.011	0.001	0.011
totincr	-0.3258	0.032	-10.175	0.000	-0.389	-0.263
educat	-0.0991	0.048	-2.050	0.040	-0.194	-0.004
rmarital=4	**coef**	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	-3.5238	1.205	-2.924	0.003	-5.886	-1.162
C(race)[T.2]	-0.3213	0.093	-3.445	0.001	-0.504	-0.139
C(race)[T.3]	-0.7706	0.171	-4.509	0.000	-1.106	-0.436
age_r	0.1155	0.071	1.626	0.104	-0.024	0.255
age2	-0.0007	0.001	-0.701	0.483	-0.003	0.001
totincr	-0.2276	0.012	-19.621	0.000	-0.250	-0.205
educat	0.0667	0.017	3.995	0.000	0.034	0.099
rmarital=5	**coef**	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	-2.8963	1.305	-2.220	0.026	-5.453	-0.339
C(race)[T.2]	-1.0407	0.104	-10.038	0.000	-1.244	-0.837
C(race)[T.3]	-0.5661	0.156	-3.635	0.000	-0.871	-0.261
age_r	0.2411	0.079	3.038	0.002	0.086	0.397
age2	-0.0035	0.001	-2.977	0.003	-0.006	-0.001
totincr	-0.2932	0.015	-20.159	0.000	-0.322	-0.265
educat	-0.0174	0.021	-0.813	0.416	-0.059	0.025
rmarital=6	**coef**	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	8.0533	0.814	9.890	0.000	6.457	9.649
C(race)[T.2]	-2.1871	0.080	-27.211	0.000	-2.345	-2.030
C(race)[T.3]	-1.9611	0.138	-14.188	0.000	-2.232	-1.690
age_r	-0.2127	0.052	-4.122	0.000	-0.314	-0.112
age2	0.0019	0.001	2.321	0.020	0.000	0.003
totincr	-0.2945	0.012	-25.320	0.000	-0.317	-0.272
educat	-0.0742	0.018	-4.169	0.000	-0.109	-0.039

In [9]:

```
#make prediction
columns = ['age_r', 'age2', 'race', 'totincr', 'educat']
new = pd.DataFrame([[25, 25**2, 2, 11, 12]], columns=columns)
results.predict(new)
```

Out[9]:

	0	1	2	3	4	5
0	0.750028	0.126397	0.001564	0.033403	0.021485	0.067122

Conclusion:

From the model, it predict this woman will have 75% chance to be married, some chance (13%) to be cohabitating. little chance to be widowed or separated)