Part 1: Bayesian Approach

Is the one who earns more (>50K) necessarily in the different age category from the one who earns less (<=50)?

X- age Y-income

1.

In [5]:

```
# Deprecation of jupyter warnings about sns functions that will be deprecated in the
import warnings
warnings.filterwarnings('ignore')

#Reading a CSV file into pandas Dataframe
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import random
from scipy.stats import f
from scipy.stats import norm
import pprint
import sys

random.seed(1)
# Removing missing values
missing_values = ["n/a", "na", "--","?"]
df_full = pd.read_csv("adult.csv",sep=",", na_values = missing_values)
df_full=df_full.dropna()

# Converting string columns to binary.
df_full['gender'] =df_full['gender'].map({'Female': 0, 'Male': 1})
df_full['income'] = df_full['income'].map({'>50K': 1, '<=50K': 0})

df_200=df_full.sample(n = 200)
df_without_sub_sample=df_full.drop(df_200.index)
df_1000=df_without_sub_sample.sample(n = 1000)
```

2.

In [6]:

```
# Define binary indicators Z_i
def get_indicator(x,threshold):
    if x>threshold:
        return 1
    return 0

threshold_of_viewed_data=df_200['age'].median()
df_200['Z']=[get_indicator(x,threshold_of_viewed_data) for x in df_200['age'].values

threshold_of_past_data=df_1000['age'].median()
df_1000['Z']=[get_indicator(x,threshold_of_past_data) for x in df_1000['age'].values
```

a.

Lets define:

$$P(Z = 1|Y = 1) = p_1$$

$$P(Z = 1|Y = 0) = p_2$$

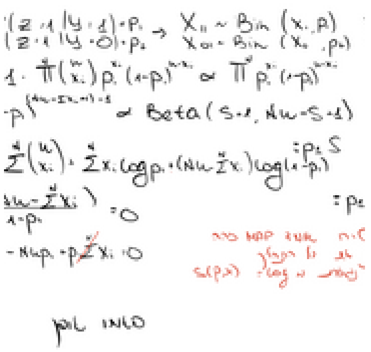
$\eta(p) = \log(p/1 - p)$

$\psi = \eta(p1) - \eta(p2) = \log((x_{00} * x_{11})/(x_{01} * x_{10}))$

In [7]:
```python
# estimation of p1,p2
def estimate_p(df,category):
    indicators=df[df['income'] ==category]['Z'].values
    total=sum(indicators)
    p=total/len(indicators)
    return p

p1=estimate_p(df_200,1)
p2=estimate_p(df_200,0)

psi_a=math.log(p1/(1-p1))-math.log(p2/(1-p2))
print('Estimation of log OR is: {}'.format(psi_a))

# esimates stansart eror of log odds ratio using bootstrap
def estimate_se_via_bootstrap(df,psi,B,n=150):
    sampled_se_list=[]
    for i in range(B):
        df=df.sample(n, replace=True)
        df=df.reset_index(drop=True)
        x00=df[df['income'] ==0][df['Z'] ==0].shape[0]
        x10=df[df['income'] ==1][df['Z'] ==0].shape[0]
        x01=df[df['income'] ==0][df['Z'] ==1].shape[0]
        x11=df[df['income'] ==1][df['Z'] ==1].shape[0]
        se=1/x00 if x00!=0 else 0
        se=se+1/x10 if x10!=0 else se
        se=se+1/x01 if x01!=0 else se
        se=se+1/x11 if x11!=0 else se
        psi_se=(np.sqrt(se))*psi
        sampled_se_list.append(psi_se)
    return sum(sampled_se_list)/B


se = estimate_se_via_bootstrap(df_200,psi_a,B=500)


z_quantile=norm.ppf(1-0.05/2)
CI=[psi_a-z_quantile*se,psi_a+z_quantile*se]

print('Confidence Interval of log OR is {}'.format(CI))
```

```
Estimation of log OR is: 1.4390997525697444
Confidence Interval of log OR is [1.08609870362007, 1.7921008015194189]
```

b.



(if you cant see the picture, please enter the link manually, to see the calculations.)

For standart uniform prior we know that MAP estimator of p_1 equal to MLE estimator.

From tutorial 6 we know MLE for p_1 is $\overline{X}_{11}/n$ and for p_2 is $\overline{X}_{01}/m$

$X_{01} \sim \text{Binomial}(X_0, p_2)$

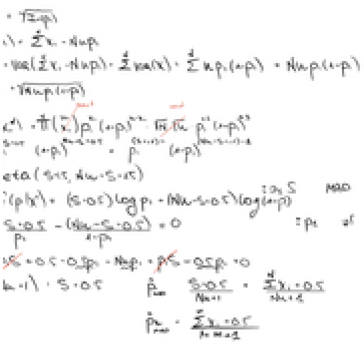
$X_{11} \sim \text{Binomial}(X_1, p_1)$

In [8]:
```
p1_b=df_200[df_200['income'] ==1][df_200['Z'] ==1]['Z'].mean()/df_200[df_200['income
p2_b=df_200[df_200['income'] ==0][df_200['Z'] ==1]['Z'].mean()/df_200[df_200['income
psi_b=math.log(p1_b/(1-p1_b))-math.log(p2_b/(1-p2_b))
print('Estimation of log OR is: {}'.format(psi_b))

def get_credible_int_b(df,sample_size,y,z):
    a=df[df['income'] ==y][df['Z'] ==z]['Z'].sum()+1
    b=df[df['income'] ==y].shape[0]*df[df['income'] ==y][df['Z'] ==z].shape[0]-df[df
    samples=np.random.beta(a, b, sample_size)
    samples.sort()
    credible_int=[samples[int(0.05*sample_size/2)],samples[int((1-0.05/2)*sample_siz
    return credible_int
ci_p1=get_credible_int_b(df_200,100000,1,1)
ci_p2=get_credible_int_b(df_200,100000,0,1)
ci_psi=[math.log(ci_p1[0]/(1-ci_p1[0]))-math.log(ci_p2[0]/(1-ci_p2[0])),math.log(ci_

print("Credible Interval of psi is {}".format(ci_psi))
```

```
Estimation of log OR is: 1.0851892683359687
Credible Interval of psi is [1.020867553133881, 1.1518594205873076]
```

c.

Inserting jeffreys prior and solving MAP we get that estimator for p_1 is $(\Sigma^N X_{11} + 0.5)/(1 + Nn)$ and for p_2 is $(\Sigma^M X_{01} + 0.5)/(1 + Mm)$



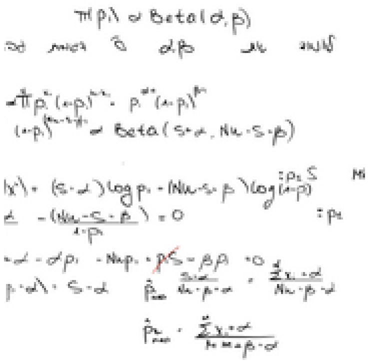
(if you cant see the picture, please enter the link manually, to see the calculations.)

In [9]:
```
p1_c=(df_200[df_200['income'] ==1][df_200['Z'] ==1]['Z'].sum()+0.5)/(1+df_200[df_200
p2_c=(df_200[df_200['income'] ==0][df_200['Z'] ==1]['Z'].sum()+0.5)/(1+df_200[df_200
psi_c=math.log(p1_c/(1-p1_c))-math.log(p2_c/(1-p2_c))
print('Estimation of log OR is: {}'.format(psi_c))

def get_credible_int_c(df,sample_size,y,z):
    a=df[df['income'] ==y][df['Z'] ==z]['Z'].sum()+1.5
    b=df[df['income'] ==y].shape[0]*df[df['income'] ==y][df['Z'] ==z].shape[0]-df[df
    samples=np.random.beta(a, b, sample_size)
    samples.sort()
    credible_int=[samples[int(0.05*sample_size/2)],samples[int((1-0.05/2)*sample_siz
    return credible_int
ci_p1=get_credible_int_c(df_200,10000,1,1)
ci_p2=get_credible_int_c(df_200,10000,0,1)
ci_psi=[math.log(ci_p1[0]/(1-ci_p1[0]))-math.log(ci_p2[0]/(1-ci_p2[0])),math.log(ci_
```

```
print("Credible Interval of psi is {}".format(ci_psi))
```

```
Estimation of log OR is: 1.0898902977078984
Credible Interval of psi is [1.027535315829402, 1.1591054335317246]
```

d.



(if you cant see the picture, please enter the link manually, to see the calculations.)

In [10]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

def get_betas_params(observations):
    a, b, loc, scale =stats.beta.fit(observations)
    return  (a,b)

a_p1,b_p1=get_betas_params(df_1000[df_1000['income'] ==1][df_1000['Z'] ==1]['Z'].val
a_p2,b_p2=get_betas_params(df_1000[df_1000['income'] ==0][df_1000['Z'] ==1]['Z'].val


p1_d=(df_200[df_200['income'] ==1][df_200['Z'] ==1]['Z'].sum()+a_p1)/(a_p1+b_p1+df_2
p2_d=(df_200[df_200['income'] ==0][df_200['Z'] ==1]['Z'].sum()+a_p2)/(a_p2+b_p2+df_2
psi_d=math.log(p1_d/(1-p1_d))-math.log(p2_d/(1-p2_d))
print('Estimation of log OR is: {}'.format(psi_d))

def get_credible_int_d(df,sample_size,y,z,alpha,beta):
    a=df[df['income'] ==y][df['Z'] ==z]['Z'].sum()+alpha
    b=df[df['income'] ==y].shape[0]*df[df['income'] ==y][df['Z'] ==z].shape[0]-df[df
    samples=np.random.beta(a, b, sample_size)
    samples.sort()
    credible_int=[samples[int(0.05*sample_size/2)],samples[int((1-0.05/2)*sample_siz
    return credible_int

ci_p1=get_credible_int_b(df_200,10000,1,1)
ci_p2=get_credible_int_b(df_200,10000,0,1)
ci_psi=[math.log(ci_p1[0]/(1-ci_p1[0]))-math.log(ci_p2[0]/(1-ci_p2[0])),math.log(ci_

print("Credible Interval of psi is {}".format(ci_psi))
```

```
Estimation of log OR is: 1.0946299487639584
Credible Interval of psi is [1.0253065129578154, 1.1643185821950337]
```

e.

All the estomators are quite simmilar.But, estimator from 2.a is less similar to the others. We tend to think that estimator from 2.d is more accurate than the others. Jaffrey's prior is better than the flat one and it is non informative. Prior calculated via past samples, assumes knowlege about the disrtibution and seems to be more precise, because in retrospect all the data came from the same data set.

In [11]:

```
print('2.a Estimation of log OR is: {}'.format(psi_a))
print('2.b Estimation of log OR is: {}'.format(psi_b))
print('2.c Estimation of log OR is: {}'.format(psi_c))
print('2.d Estimation of log OR is: {}'.format(psi_d))
```

```
2.a Estimation of log OR is: 1.4390997525697444
2.b Estimation of log OR is: 1.0851892683359687
2.c Estimation of log OR is: 1.0898902977078984
2.d Estimation of log OR is: 1.0946299487639584
```