Report2_v2

February 22, 2021

1 Report 2

Authors: Richard Liang, Olivia majedi, Priyanka Kishore, Rhea Rakheja, Jiadong Li, Michael Strobel

[1]:
```
import pandas as pd
```

[2]:
```
df = pd.read_csv('https://raw.githubusercontent.com/jdli28/STAT440/master/
↪summer_mount_ginini.csv')
```

[3]:
```
df = df.dropna()
```

[4]:
```
df
```

[4]:
```
           Date     Location  MinTemp  MaxTemp
0     12/1/2008  MountGinini      5.2     13.0
1     12/2/2008  MountGinini      3.0     15.0
2     12/3/2008  MountGinini      6.0     15.0
3     12/4/2008  MountGinini      2.0     15.0
4     12/5/2008  MountGinini      8.0     17.8
..          …            …        …       …
743   2/24/2017  MountGinini     13.8     24.1
744   2/25/2017  MountGinini      9.5     11.2
745   2/26/2017  MountGinini      4.7     16.7
746   2/27/2017  MountGinini      5.6     16.2
747   2/28/2017  MountGinini      7.5     18.0

[745 rows x 4 columns]
```

[5]:
```
import numpy as np
```

1.1 Population size N and true parameter mu(MaxTemp)

[6]:
```
N = len(df)
N
```

[6]:
```
745
```

[7]:
```
max_temp_mean = np.mean(df.MaxTemp)
max_temp_mean
```

[7]: 19.112885906040272

1.2 Calculate sample size n for 90% and 95% confidence levels and couple different d's. Use true ^2 for these calculations

[8]:
```
sigma_sq = np.var(df.MaxTemp)
sigma_sq
```

[8]: 21.75814267825774

[9]:
```
r = [.05, .01, .1]
d = [(max_temp_mean * rval) for rval in r]
d
```

[9]: [0.9556442953020137, 0.19112885906040272, 1.9112885906040273]

[10]:
```
 from scipy import stats
z_alpha_90 = stats.norm.ppf(1-0.05)
z_alpha_95 = stats.norm.ppf(1-0.025)
```

[11]:
```
n_90 = []
n_95 = []
```

[12]:
```
for d_val in d:
  n0 = z_alpha_90**2*sigma_sq/(d_val**2)
  n_90.append(
    1/((1/n0)+(1/N))
  )

  n0 = z_alpha_95**2*sigma_sq/(d_val**2)
  n_95.append(
    1/((1/n0)+(1/N))
  )
```

[13]:
```
import math
```

[14]:
```
n_90
n_90 = [math.ceil(n) for n in n_90]
n_90
```

[14]: [60, 510, 16]

[15]:
```
n_95
n_95 = [math.ceil(n) for n in n_95]
```

```
n_95
```

[15]: [82, 563, 23]

1.3 Estimate your parameter of interest using SRS with n's which you calculated above.

1.3.1 90% CI

[16]:
```
sample_90s = []
sample_95s = []
```

[17]:
```
for n in n_90:
  sample = np.random.choice(df.MaxTemp, size=n, replace=False)
  sample_90s.append(sample)
  print(np.mean(sample))
```

```
19.415
19.396666666666665
20.03125
```

1.3.2 95% CI

[18]:
```
for n in n_95:
  sample = np.random.choice(df.MaxTemp, size=n, replace=False)
  sample_95s.append(sample)
  print(np.mean(sample))
```

```
19.640243902439025
19.127353463587923
19.778260869565216
```

1.4 Estimate variance of your estimator for these n's

[19]:
```
for sample in sample_90s:
  n = len(sample)
  s_sq = np.var(sample)
  varHat_yBar = ((N-n)/N)*(s_sq/n)
  print(varHat_yBar)
```

```
0.4107539159209545
0.013218372481071211
1.0621351687028107
```

[20]:
```
for sample in sample_95s:
  n = len(sample)
  s_sq = np.var(sample)
  varHat_yBar = ((N-n)/N)*(s_sq/n)
```

```
print(varHat_yBar)
```

```
0.22770015458317275
0.00957880666589971
1.0538009215156188
```

1.5 Calculate confidence intervals for these estimators.

1.5.1 90% CI for ybar

[21]:
```
def get_ybar_CI(ybar, z, n, s_sq):
  upper = ybar + z * np.sqrt(
    ((N-n)/N)*(s_sq/n)
  )
  lower = ybar - z * np.sqrt(
    ((N-n)/N)*(s_sq/n)
  )
  return [lower, upper]
```

[22]:
```
for sample in sample_90s:
  ybar = np.mean(sample)
  n = len(sample)
  s_sq = np.var(sample)
  ci = get_ybar_CI(ybar, z_alpha_90, n, s_sq)
  print(ci)
```

```
[18.360811891328687, 20.46918810867131]
[19.207555902457017, 19.585777430876313]
[18.3360648027285, 21.7264351972715]
```

1.5.2 95% CI for ybar

[23]:
```
for sample in sample_95s:
  ybar = np.mean(sample)
  n = len(sample)
  s_sq = np.var(sample)
  ci = get_ybar_CI(ybar, z_alpha_95, n, s_sq)
  print(ci)
```

```
[18.704989516030203, 20.575498288847847]
[18.935529087247364, 19.319177839928482]
[17.766263641631795, 21.790258097498636]
```

1.6 Choosing optimal sample sizes

[24]:
```
n = n_95[0]
n
```

[24]: 82

For convenience, we will take `n=80` instead. Since we will be studying methods for when we need to divide sampling units into groups, `n=80` will have more options for grouping

[25]:
```
n = 80
```

1.7 Guaranteeing the nominal confidence level

[26]:
```
d_val = 0.9556442953020137
```

[27]:
```
ct = 0
for i in range(100):
  sample = np.random.choice(df.MaxTemp, size=n, replace=False)
  ybar = np.mean(sample)
  d = abs(ybar - max_temp_mean)
  print(d)
  if d - d_val < 0:
    ct +=1
print("----------")
print(ct)
```

```
0.1583640939597295
0.22586409395972495
0.6241359060402729
0.27836409395972694
0.1446140939597278
0.3941359060402725
0.5896140939597281
0.30288590604027377
0.12913590604027547
0.0691359060402732
0.26836409395972893
0.8803859060402743
0.1853859060402705
0.3658640939597255
0.23961409395972666
0.29413590604027107
0.03961409395972737
0.11913590604027391
0.6003859060402732
0.27086409395972666
0.7233640939597272
```

0.3303859060402736
0.007114093959724954
0.7796140939597294
0.3246140939597275
0.9433640939597261
0.08913590604027277
0.5558640939597268
0.09788590604027192
0.4216359060402688
0.3658640939597291
0.5153859060402759
0.03586409395972723
0.2946140939597264
0.31461409395972595
0.6358640939597286
0.059135906040271635
0.38586409395972865
0.362885906040272 5
0.5933640939597282
0.08163590604027249
0.03913590604027206
0.6383640939597264
0.27836409395972694
0.15711409395972709
0.4308640939597268
0.3316359060402725
0.6603859060402719
0.6928859060402743
0.5458640939597288
0.6233640939597294
0.19586409395972382
0.33288590604027135
0.6833640939597245
0.5058640939597261
0.8166359060402755
0.3941359060402725
0.2353859060402712
0.29538590604026993
0.08336409395972666
0.06336409395972709
0.1441359060402725
0.05663590604027391
0.01663590604027121
0.036635906040274335
0.3241359060402722
0.5796140939597301
0.13288590604027561
0.27038590604027135

```
0.5041359060402719
0.5816359060402725
0.22913590604027334
0.2303859060402722
0.45836409395972666
1.7083640939597267
0.20038590604027107
0.030385906040272914
0.4933640939597268
0.06788590604027434
0.5353859060402719
0.24211409395972794
0.12163590604027164
0.25163590604027064
0.9316359060402739
0.8471140939597284
0.10163590604027561
0.9216359060402723
0.14211409395972652
0.25163590604027064
1.0896140939597245
0.9546140939597301
0.1003859060402732
0.7353859060402748
0.1846140939597305
0.03538590604027192
0.002114093959725949
0.5153859060402688
0.1491359060402715
0.2308640939597275
0.2128859060402739
----------
98
```

Almost all (98/100) of our samples have differences less than d=0.9556442953020137