(a) Sensitivity : $TP = P(D=1 \mid R=1) = 99\% \Rightarrow FN = 1\% = P(D=0 \mid R=1)$

Specificity : $TN = P(D=0 \mid R=0) = 98\% \Rightarrow FP = 2\% = P(D=1 \mid R=0)$

(i) $P(D=1 \mid H_0) = P(D=1 \mid R=0) = FP = 2\%$

(ii) $$P(R=0 \mid D=1) = \frac{P(R=0 \cap D=1)}{P(D=1)} = \frac{P(D=1 \mid R=0) \cdot P(R=0)}{P(D=1 \mid R=0) \cdot P(R=0) + P(D=1 \mid R=1)P(R=1)}$$

$$= \frac{FP \cdot \pi}{FP \cdot \pi + TP \cdot (1-\pi)}$$

$$= \frac{0.02 \cdot 0.995}{0.02 \cdot 0.995 + 0.005 \cdot 0.99} = \frac{0.0199}{0.02485} = 0.801$$

(iii)

WANT $P(D_2 = 1 \mid D_1 = 1)$

Independence :

$$P(D_1 \cap D_2 \mid R) = P(D_1 \mid R) P(D_2 \mid R)$$

$$P(D_2 = 1 \mid D_1 = 1) = \frac{P(D_1 = 1 \cap D_2 = 1)}{P(D_1 = 1)}$$

$$= \frac{P(D_1 = 1 \cap D_2 = 1 \mid R=0)\,(R=0) + P(D_1 = 1 \cap D_2 = 1 \mid R=1) P(R=1)}{P(D_1 = 1)}$$

$$= \frac{\left[\left[P(D_1 = 1 \mid R=0) P(D_2 = 1 \mid R=0)\right] P(R=0) + \left[P(D_1 = 1 \mid R=1) P(D_1 = 1 \mid R=1)\right] P(R=1) \right]}{P(D=1 \mid R=0) \cdot P(R=0) + P(D=1 \mid R=1)P(R=1)}$$

$$= \frac{FP^2 \cdot \pi + TP^2 \cdot (1-\pi)}{FP \cdot \pi + TP \cdot (1-\pi)}$$

$$= \frac{(0.02)^2 \cdot 0.995 + 0.99^2 \times (0.005)}{0.002 \cdot 0.995 + 0.99 \times 0.005}$$

$$= 0.213$$

b) (i) $$LR(T) = \frac{f_1(T)}{f_0(T)} = \frac{2c\ e^{-2ct}}{c\ e^{-ct}} = 2\ e^{-ct}$$

(ii) $$2e^{-ct} < \gamma \qquad T < -\frac{1}{c}\log\left(\frac{\gamma}{2}\right)$$

$$FPR = P(D=1 \mid R=0) = P(D=1 \mid H_0) = \alpha$$

$$= P\left(T < -\frac{1}{c}\log\left(\frac{\gamma}{2}\right) \mid H_0\right)$$

$$= \int_0^{-\frac{1}{c}\log\left(\frac{\gamma}{2}\right)} c\,e^{-ct}\,dt \qquad = 1 - \frac{\gamma}{2}$$

$$1 - \frac{\gamma}{2} = \alpha \qquad \gamma = 2 - 2\alpha$$

(iii) $$P\left(T < -\frac{1}{c}\log\left(\frac{\gamma}{2}\right) \mid H_1\right)$$

$$= \int_0^{-\frac{1}{c}\log(\gamma/2)} f_1(t)\,dt \qquad = 1 - e^{-2c\cdot\left(-\frac{1}{c}\log\left(\frac{\gamma}{2}\right)\right)}$$

$$= 1 - \frac{\gamma^2}{4}$$

$$= 1 - (1-\alpha)^2$$

102_hw1

September 20, 2022

1 FA 22 Data 102 Homework 1

Derek Derui Wang, derekderuiwang@berkeley.edu

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from datascience import *

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from scipy.stats import norm
import timeit
import hashlib
%matplotlib inline

sns.set(style="dark")
plt.style.use("ggplot")
```

1.1 Question 2

[2]:
```
df = pd.read_csv("policez.csv", index_col = 0)
df
```

[2]:
```
             x
1     2.411365
2     0.160788
3    -0.852171
4     0.151016
5     1.836084
…          …
2745  2.008162
2746  0.963842
2747 -2.735369
2748  1.074744
```

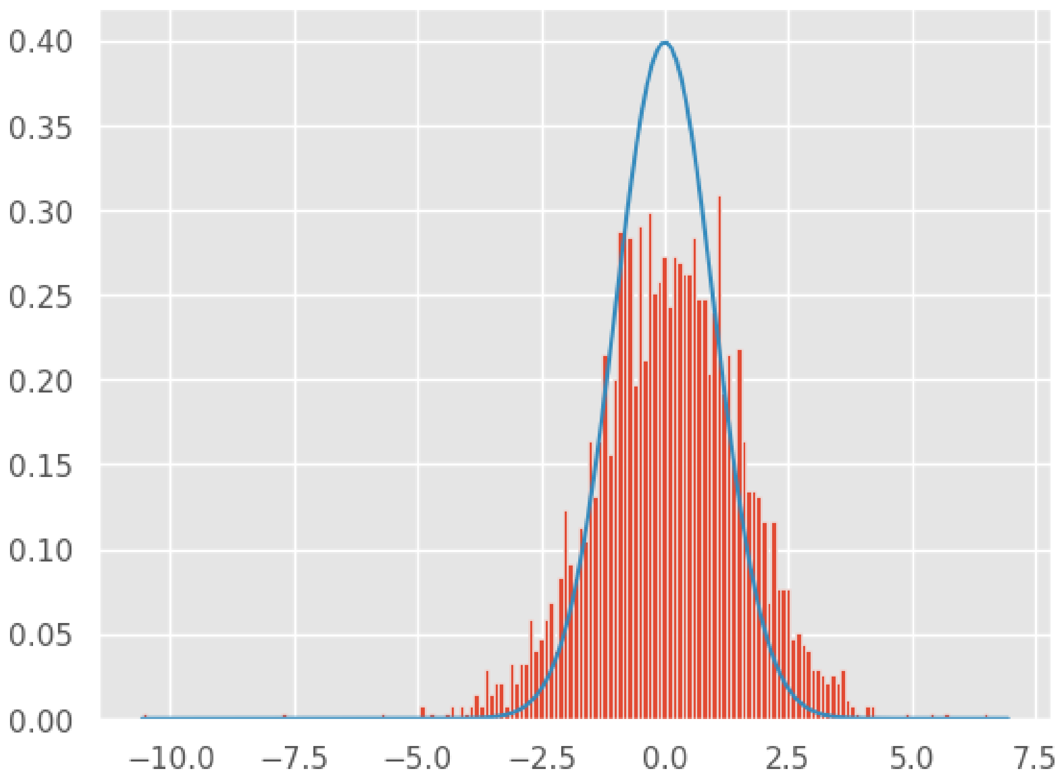
```
2749 -1.978600

[2749 rows x 1 columns]
```

1.1.1 a) Normalized Histogram

In one plot, make a normalized histogram of the z-scores and a line plot of the pdf of the theoretical null N (0, 1). Describe how the fit looks.

[3]:
```python
x_axis = np.arange(min(df['x']), max(df['x']), 0.1)
plt.hist(df, x_axis, density = True)
plt.plot(x_axis, norm.pdf(x_axis,
                          0, 1))
```

[3]:
```
[<matplotlib.lines.Line2D at 0x7fd5cd23f820>]
```



The normalized histogram of the z-scores does not fit the line plot of the pdf of the theoretical null N (0, 1) well. There are fewer distributions around the mean(0), and more at around -1.5 and + 1.5

1.1.2 b) Compute p-values

Compute p-values $P_i = \phi(-z_i)$ (where ϕ is the standard normal CDF) and then apply the BH procedure with $\alpha = 0.2$. Plot the sorted p-values as well as the decision boundary. How many discoveries did you make?

[4]:
```
neg_z = -df['x']
neg_z
```

[4]:
```
1      -2.411365
2      -0.160788
3       0.852171
4      -0.151016
5      -1.836084
          …
2745   -2.008162
2746   -0.963842
2747    2.735369
2748   -1.074744
2749    1.978600
Name: x, Length: 2749, dtype: float64
```

[5]:
```
p_val = norm.cdf(neg_z, loc = 0, scale = 1)
p_val
```

[5]:
```
array([0.00794646, 0.43613015, 0.80294036, …, 0.99688448, 0.1412447 ,
       0.97606948])
```

[6]:
```
df['p-val'] = p_val
df
```

[6]:
```
             x     p-val
1     2.411365  0.007946
2     0.160788  0.436130
3    -0.852171  0.802940
4     0.151016  0.439982
5     1.836084  0.033173
…          …         …
2745  2.008162  0.022313
2746  0.963842  0.167562
2747 -2.735369  0.996884
2748  1.074744  0.141245
2749 -1.978600  0.976069

[2749 rows x 2 columns]
```

[7]:
```
p_sorted = df.sort_values(by = ['p-val']).reset_index(drop = True)
m = len(p_sorted)
```

```python
k = np.arange(1, m+1) # index of each test in sorted order
p_sorted['k'] = k
alpha = .2
p_sorted
```

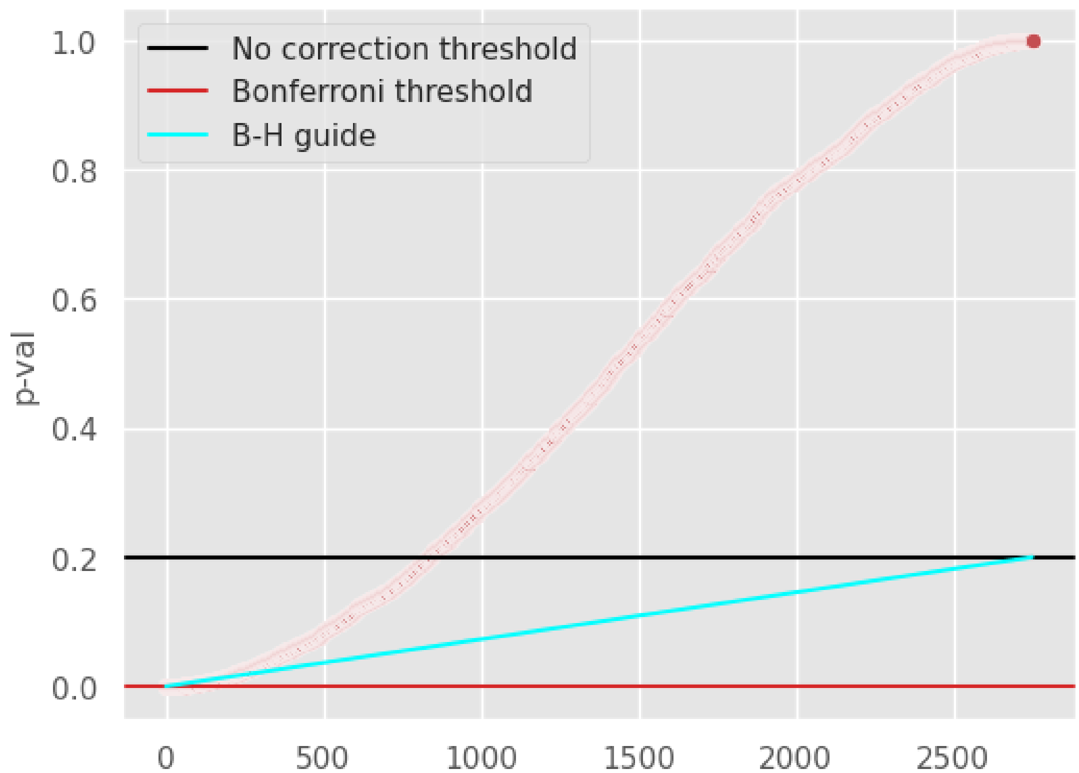
[7]:

	x	p-val	k
0	6.952483	1.794564e-12	1
1	6.506210	3.853529e-11	2
2	5.703130	5.881356e-09	3
3	5.360467	4.150348e-08	4
4	4.934229	4.023405e-07	5
…	…	…	…
2744	-4.917371	9.999996e-01	2745
2745	-4.937951	9.999996e-01	2746
2746	-5.724645	1.000000e+00	2747
2747	-7.705945	1.000000e+00	2748
2748	-10.563937	1.000000e+00	2749

[2749 rows x 3 columns]

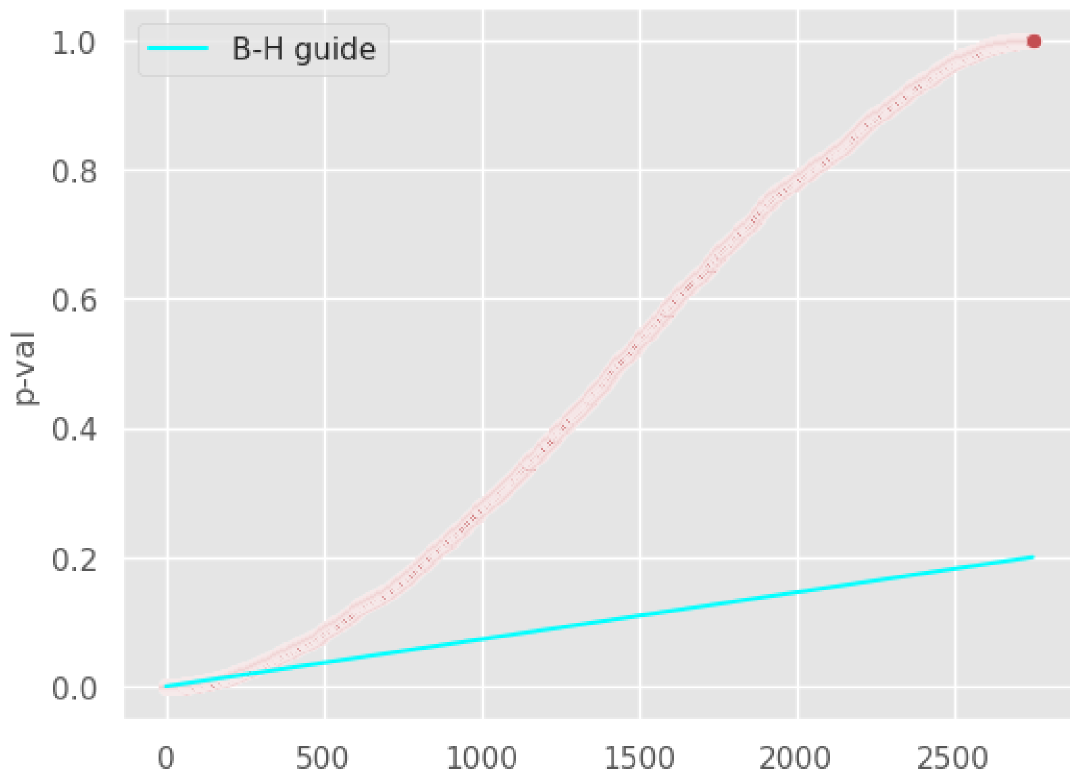
[8]:
```python
sns.scatterplot(x=k, y=p_sorted['p-val'], color = 'r')
plt.axhline(alpha, label='No correction threshold', color='black')
plt.axhline(alpha / len(p_sorted), label='Bonferroni threshold', color='tab:
↪red')
plt.plot(k, k/len(p_sorted)* alpha, label='B-H guide', color='cyan')
plt.legend()
```

[8]: <matplotlib.legend.Legend at 0x7fc926d71ca0>

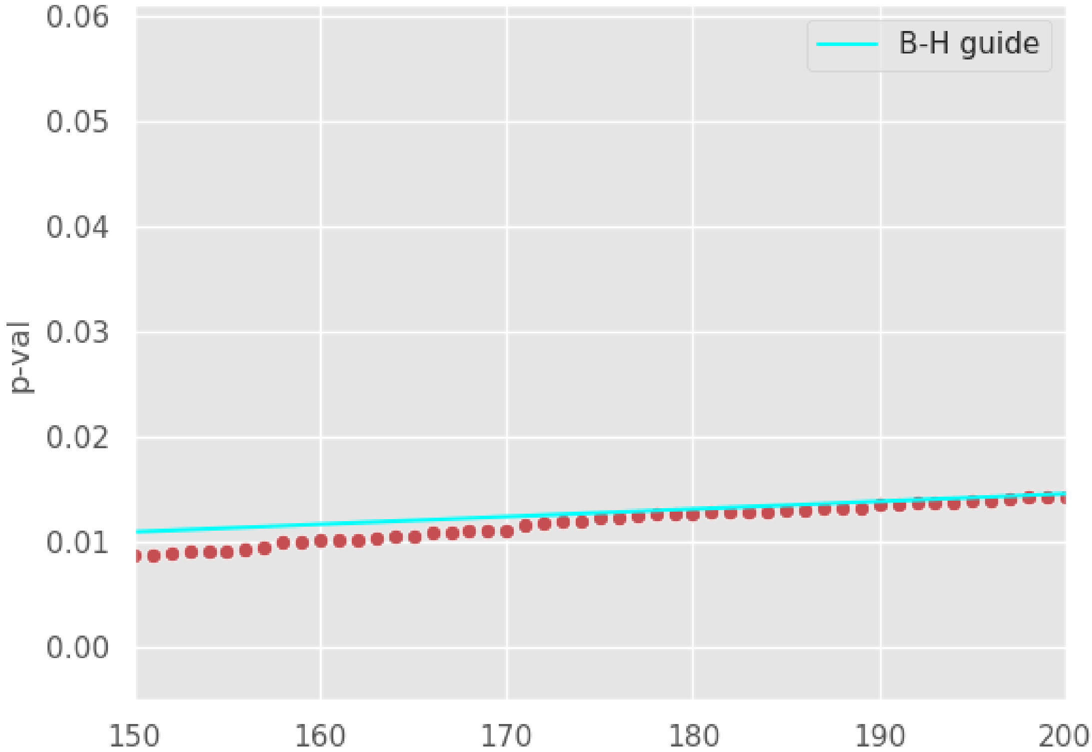


[9]:
```
sns.scatterplot(x=k, y=p_sorted['p-val'], color = 'r')
plt.plot(k, k/len(p_sorted)* alpha, label='B-H guide', color='cyan')
plt.legend()
```

[9]: <matplotlib.legend.Legend at 0x7fc926cf7070>

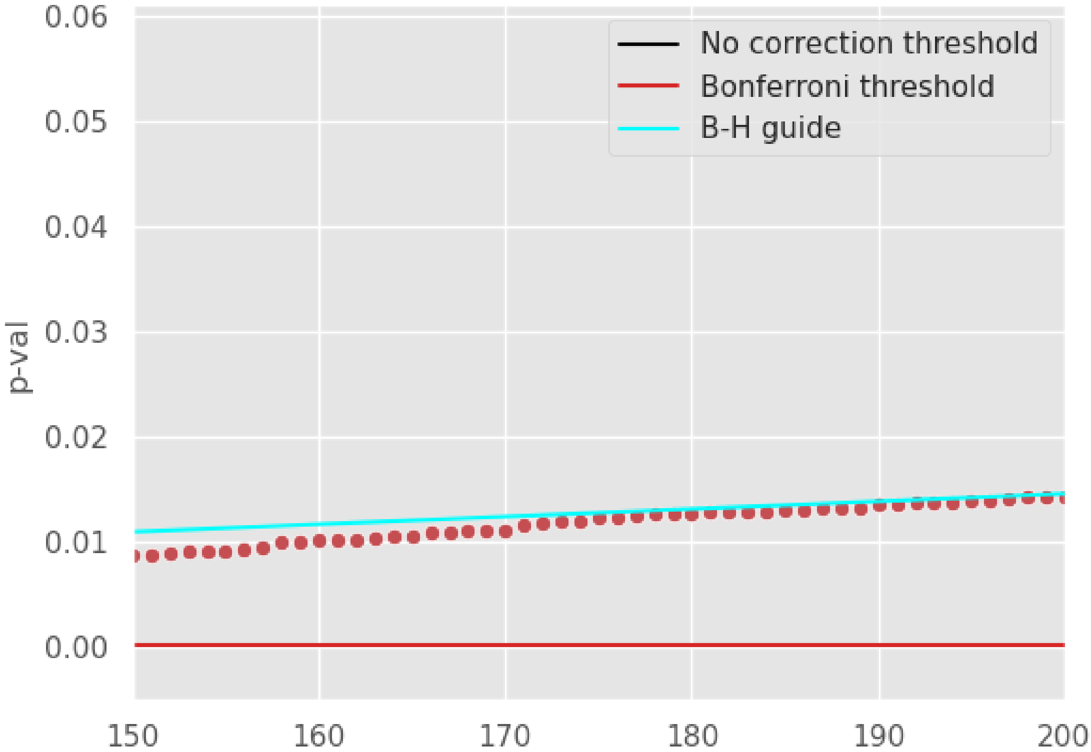


[10]:
```
sns.scatterplot(x=k, y=p_sorted['p-val'], color = 'r')
plt.plot(k, k/len(p_sorted)* alpha, label='B-H guide', color='cyan')
plt.legend()
plt.axis([150, 200, -0.005, .061]);
```



[11]:
```
plt.axis([150, 200, -0.005, .061]);
sns.scatterplot(x=k, y=p_sorted['p-val'], color = 'r')
plt.axhline(alpha, label='No correction threshold', color='black')
plt.axhline(alpha / len(p_sorted), label='Bonferroni threshold', color='tab:
    ↪red')
plt.plot(k, k/len(p_sorted)* alpha, label='B-H guide', color='cyan')
plt.legend()
```

[11]: <matplotlib.legend.Legend at 0x7fc926c4f220>



[12]:
```
p_sorted['B-H'] =  p_sorted['p-val'] - p_sorted['k']/len(p_sorted)* alpha
p_sorted[p_sorted['B-H']>=-0.00009]
```

[12]:

	x	p-val	k	B-H
0	6.952483	1.794564e-12	1	-0.000073
201	2.176679	1.475225e-02	202	0.000056
202	2.170774	1.497412e-02	203	0.000205
203	2.167356	1.510385e-02	204	0.000262
204	2.161422	1.533138e-02	205	0.000417
…	…	…	…	…
2744	-4.917371	9.999996e-01	2745	0.800291
2745	-4.937951	9.999996e-01	2746	0.800218
2746	-5.724645	1.000000e+00	2747	0.800146
2747	-7.705945	1.000000e+00	2748	0.800073
2748	-10.563937	1.000000e+00	2749	0.800000

[2549 rows x 4 columns]

[13]:
```
p_sorted[p_sorted['k'] == 201]
```

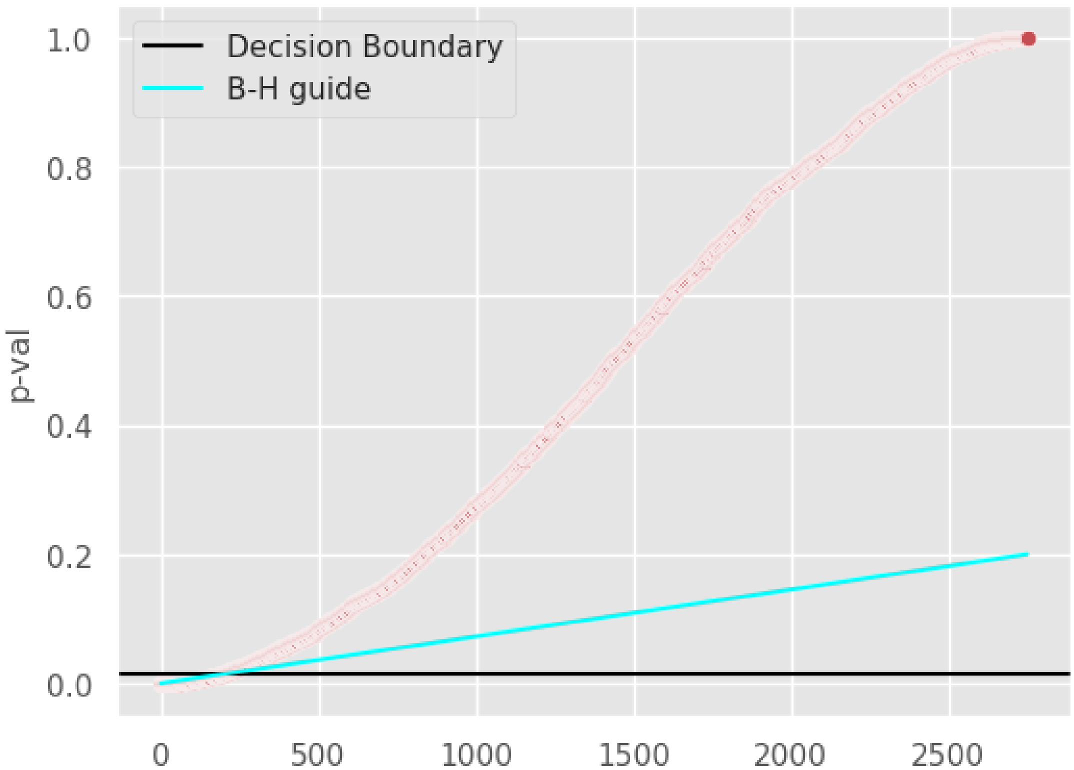
[13]:

	x	p-val	k	B-H
200	2.186698	0.014382	201	-0.000241

2.2 RESPONSE: k = 201, $p - val_{B-H} = 0.014382$, 201 Discoveries were made

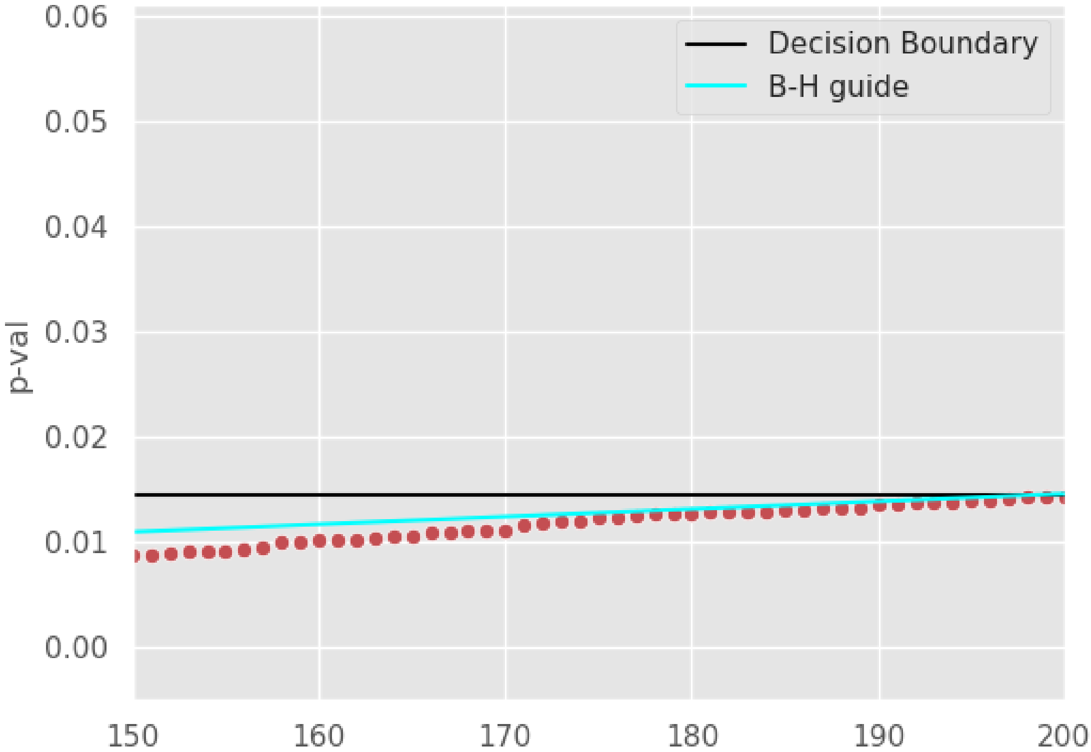
[14]:
```
sns.scatterplot(x=k, y=p_sorted['p-val'], color = 'r')
plt.axhline(0.014382, label='Decision Boundary', color='black')
plt.plot(k, k/len(p_sorted)* alpha, label='B-H guide', color='cyan')
plt.legend()
```

[14]: <matplotlib.legend.Legend at 0x7fc926c4fac0>



[15]:
```
plt.axis([150, 200, -0.005, .061]);
sns.scatterplot(x=k, y=p_sorted['p-val'], color = 'r')
plt.axhline(0.014382, label='Decision Boundary', color='black')
plt.plot(k, k/len(p_sorted)* alpha, label='B-H guide', color='cyan')
plt.legend()
```

[15]: <matplotlib.legend.Legend at 0x7fc926ce2640>



1.1.3 C) Empirical Null

Looking at the data, we can get a better fit to the distribution of z-scores if we use $N(0.10, 1.40^2)$, called the empirical null (instead of the theoretical null from part (a)). Repeat steps (a) and (b), treating the empirical null as the null distribution.

[16]:
```
df = pd.read_csv("policez.csv", index_col = 0)
df
```

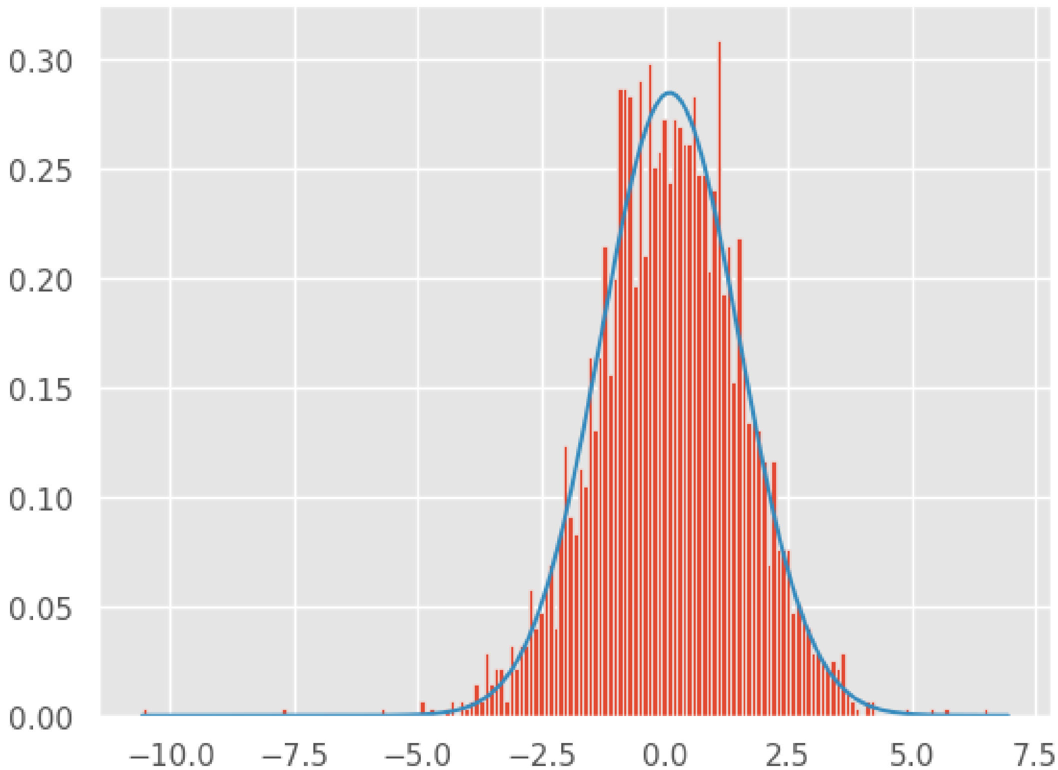
[16]:

	x
1	2.411365
2	0.160788
3	-0.852171
4	0.151016
5	1.836084
…	…
2745	2.008162
2746	0.963842
2747	-2.735369
2748	1.074744
2749	-1.978600

```
[2749 rows x 1 columns]
```

[17]:
```
x_axis = np.arange(min(df['x']), max(df['x']), 0.1)
plt.hist(df['x'], x_axis, density = True)
plt.plot(x_axis, norm.pdf(x_axis,
                          0.10, 1.40))
```

[17]:
```
[<matplotlib.lines.Line2D at 0x7fc926cdef40>]
```



[18]:
```
neg_z = -df['x']
neg_z
```

[18]:
```
1       -2.411365
2       -0.160788
3        0.852171
4       -0.151016
5       -1.836084
           …
2745    -2.008162
2746    -0.963842
2747     2.735369
```

```
2748   -1.074744
2749    1.978600
Name: x, Length: 2749, dtype: float64
```

[19]:
```
p_val = norm.cdf(neg_z, loc = 0.10 , scale = 1.40)
df['p-val'] = p_val
df
```

[19]:
```
             x     p-val
1     2.411365  0.036420
2     0.160788  0.426114
3    -0.852171  0.704458
4     0.151016  0.428852
5     1.836084  0.083345
…          …        …
2745  2.008162  0.066055
2746  0.963842  0.223661
2747 -2.735369  0.970110
2748  1.074744  0.200706
2749 -1.978600  0.910179

[2749 rows x 2 columns]
```

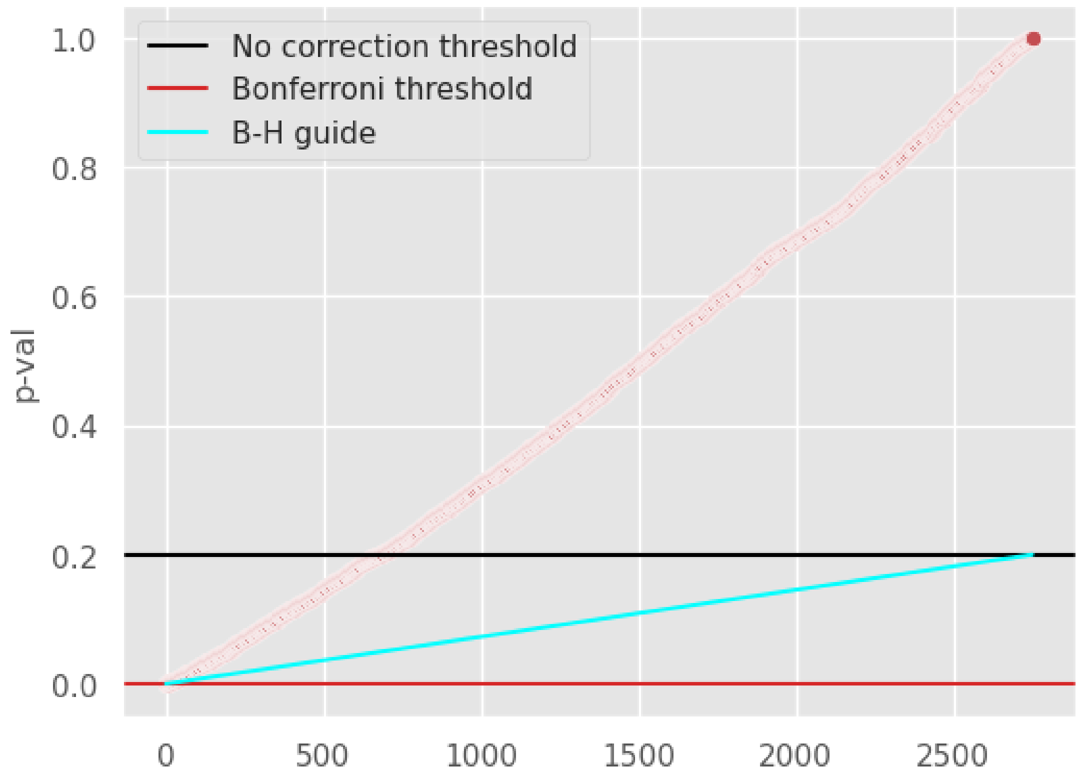
[20]:
```
p_sorted = df.sort_values(by = ['p-val']).reset_index(drop = True)
m = len(p_sorted)
k = np.arange(1, m+1) # index of each test in sorted order
p_sorted['k'] = k
alpha = .2
p_sorted
```

[20]:
```
              x         p-val     k
0      6.952483  2.358410e-07     1
1      6.506210  1.186659e-06     2
2      5.703130  1.698380e-05     3
3      5.360467  4.803008e-05     4
4      4.934229  1.616499e-04     5
…           …             …    …
2744  -4.917371  9.997102e-01  2745
2745  -4.937951  9.997255e-01  2746
2746  -5.724645  9.999706e-01  2747
2747  -7.705945  1.000000e+00  2748
2748 -10.563937  1.000000e+00  2749

[2749 rows x 3 columns]
```

[21]:
```
sns.scatterplot(x=k, y=p_sorted['p-val'], color = 'r')
plt.axhline(alpha, label='No correction threshold', color='black')
```

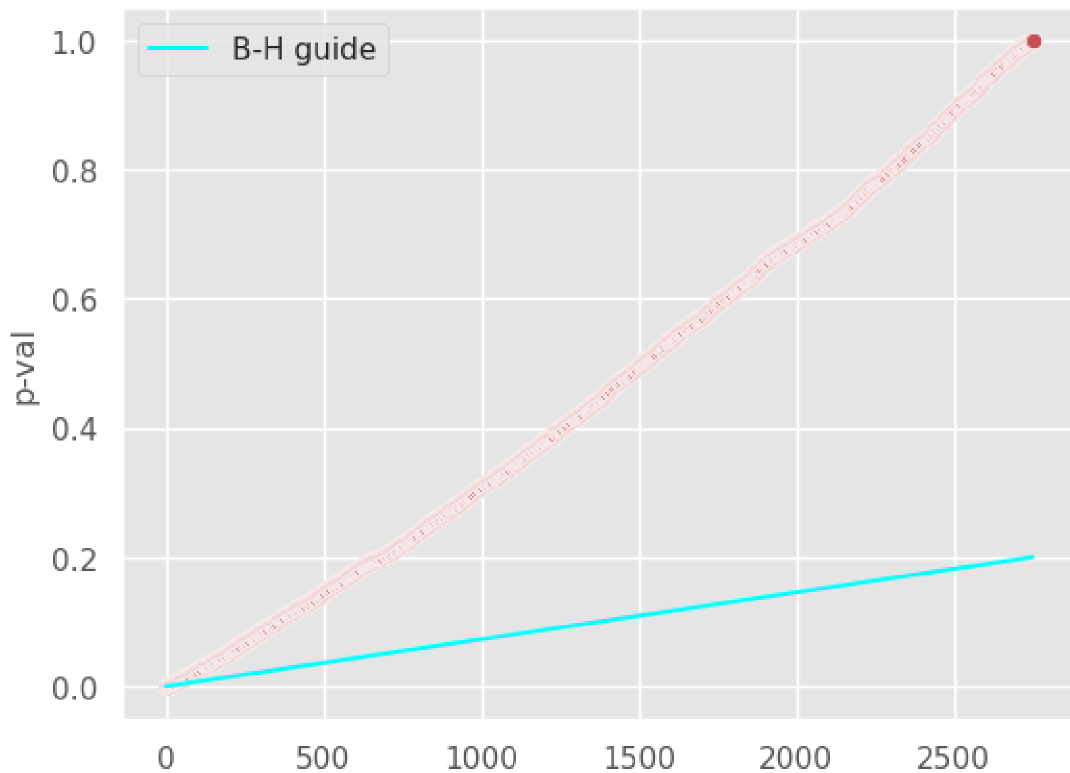
```
plt.axhline(alpha / len(p_sorted), label='Bonferroni threshold', color='tab:
 ↪red')
plt.plot(k, k/len(p_sorted)* alpha, label='B-H guide', color='cyan')
plt.legend()
```

[21]: <matplotlib.legend.Legend at 0x7fc92699c160>

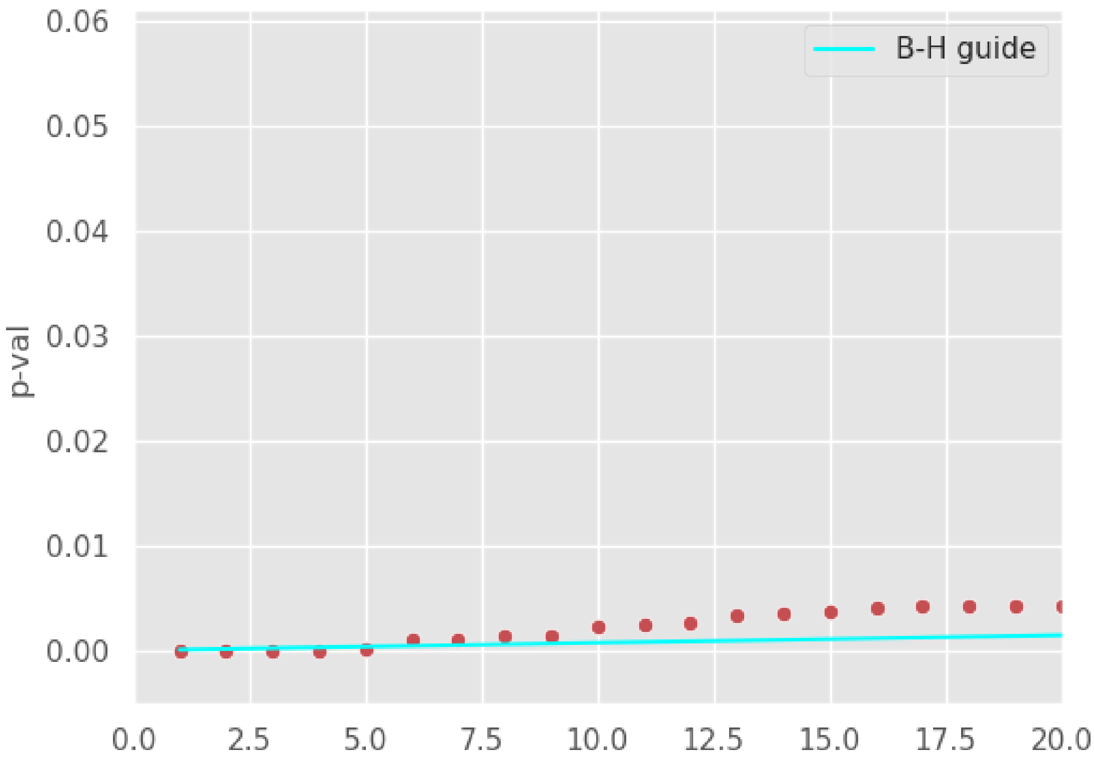


[22]:
```
sns.scatterplot(x=k, y=p_sorted['p-val'], color = 'r')
plt.plot(k, k/len(p_sorted)* alpha, label='B-H guide', color='cyan')
plt.legend()
```

[22]: <matplotlib.legend.Legend at 0x7fc9268f7340>



[23]:
```
sns.scatterplot(x=k, y=p_sorted['p-val'], color = 'r')
plt.plot(k, k/len(p_sorted)* alpha, label='B-H guide', color='cyan')
plt.legend()
plt.axis([0,20, -0.005, .061]);
```



[24]:
```
p_sorted['B-H'] =  p_sorted['p-val'] - p_sorted['k']/len(p_sorted)* alpha
p_sorted[p_sorted['B-H']>=-0.00009]
```

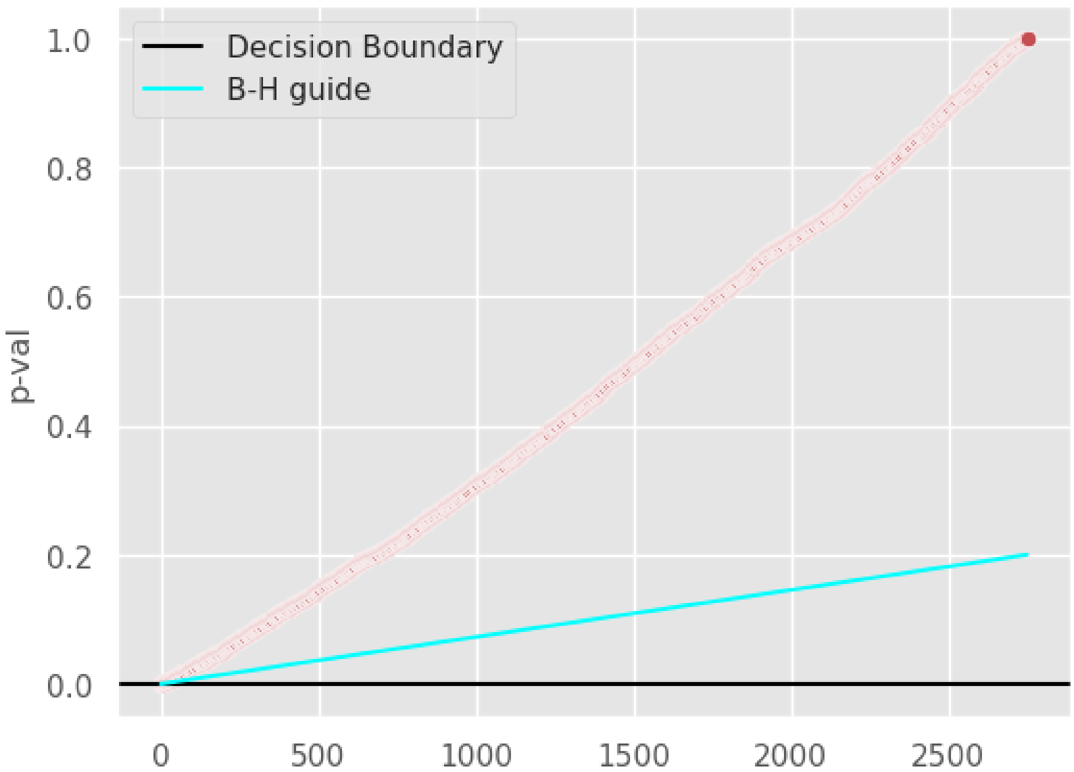
[24]:
```
              x         p-val     k       B-H
0      6.952483  2.358410e-07     1 -0.000073
5      4.224927  1.003367e-03     6  0.000567
6      4.214797  1.028083e-03     7  0.000519
7      4.105680  1.332025e-03     8  0.000750
8      4.067040  1.457999e-03     9  0.000803
…           …             …   …     …
2744  -4.917371  9.997102e-01  2745  0.800001
2745  -4.937951  9.997255e-01  2746  0.799944
2746  -5.724645  9.999706e-01  2747  0.800116
2747  -7.705945  1.000000e+00  2748  0.800073
2748 -10.563937  1.000000e+00  2749  0.800000

[2745 rows x 4 columns]
```

[25]:
```
p_sorted[p_sorted['k'] == 5]
```

[25]:
```
          x     p-val  k       B-H
4  4.934229  0.000162  5 -0.000202
```

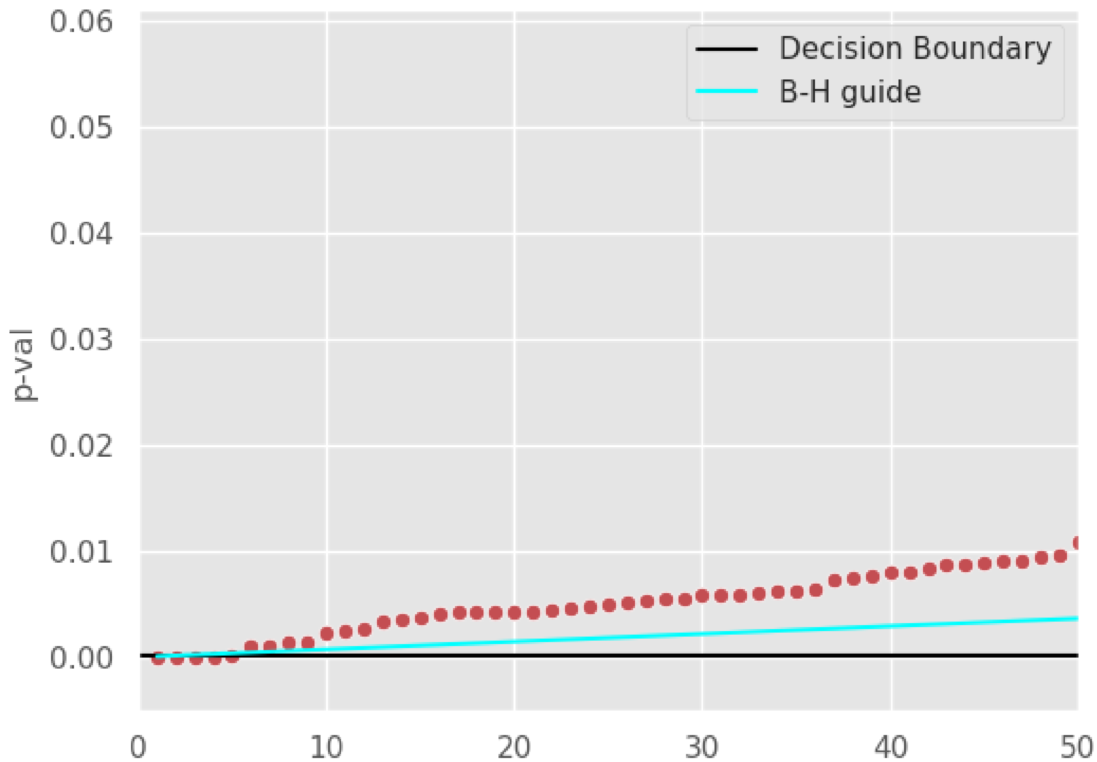
[26]:
```
sns.scatterplot(x=k, y=p_sorted['p-val'], color = 'r')
plt.axhline(0.000162, label='Decision Boundary', color='black')
plt.plot(k, k/len(p_sorted)* alpha, label='B-H guide', color='cyan')
plt.legend()
```

[26]: <matplotlib.legend.Legend at 0x7fc926879b80>



[27]:
```
plt.axis([0, 50, -0.005, .061]);
sns.scatterplot(x=k, y=p_sorted['p-val'], color = 'r')
plt.axhline(0.000162, label='Decision Boundary', color='black')
plt.plot(k, k/len(p_sorted)* alpha, label='B-H guide', color='cyan')
plt.legend()
```

[27]: <matplotlib.legend.Legend at 0x7fc92679b220>



2.3 RESPONSE: k = 5, $p - val_{B-H} = 0.000162$, 5 Discoveries were made

1.2 d) Theoretical Null

Assume the observed data is normally distributed, there is some racial bias existing, and a permutation test will be performed to get the empirical null. Limitation of Theoretical null is that it does not fit the data well, and has the assumption of normally distributed that is hard to reach. There might be covariants, or co-linearity between variables, also Omitted Variable Bias.

I would use the empirical null, because it is a valid method and only have 5 cases discovered, which is much smaller than using the theoretical null. Other limitations include small sample size, ambiguous standard and variation/bias during data collection

1.3 Question 3 p-values, FDR and FWER

[28]:
```
df = Table.read_table("adult.csv", index_col = 0)
df
```

[28]:

index	Age	Post HS?	Ever Married?	Gender	Hours per Week	50K?
38	51	1	1	Male	40	1
23	20	0	0	Female	30	0

```
22     | 58    | 1          | 1               | Female | 40              | 0
77     | 23    | 1          | 0               | Female | 35              | 0
80     | 27    | 0          | 0               | Male   | 1               | 0
32     | 39    | 1          | 1               | Male   | 40              | 0
59     | 24    | 0          | 0               | Male   | 40              | 0
57     | 49    | 1          | 0               | Female | 40              | 0
35     | 35    | 0          | 1               | Male   | 50              | 0
21     | 40    | 1          | 1               | Male   | 40              | 0
… (80 rows omitted)
```

1.3.1 a) Permutation Test Function

[29]:
```python
def mean_different(table, label, numeral):
    reduced = table.select(numeral,label)
    means_table = reduced.group(label, np.average)
    means = means_table.column(1)
    return means.item(1) - means.item(0)

def one_simulation(dummy, numeral):
    table = df.select(numeral, dummy)
    shuffled_labels = table.sample(with_replacement=False).column(dummy)

    # table of birth weights and shuffled labels
    shuffled_table = table.select(numeral).with_column(
        'Shuffled Label', shuffled_labels)

    return mean_different(shuffled_table, 'Shuffled Label', numeral)
```

[43]:
```python
def avg_difference_in_means(dummy, numeral):

    differences = make_array()
    observed = mean_different(df, dummy, numeral)
    repetitions = 25000
    for i in np.arange(repetitions):
        new_difference = one_simulation(dummy, numeral)
        differences = np.append(differences, new_difference)
    empirical_p = np.count_nonzero(differences >= observed) / repetitions
    return empirical_p, differences
```

1.3.2 b) 8 p-values

[44]:
```python
gender_age, gender_age_diff = avg_difference_in_means("Gender", "Age")
gender_hpw, gender_hpw_diff = avg_difference_in_means("Gender", "Hours per␣
↪Week")
```

[45]:
```python
gender_age_diff
```

[45]:
```
array([ 0.35817308,  2.57572115,  3.0625    , …, -3.10336538,
       -3.15745192,  5.55048077])
```

[32]:
```python
ps_age = avg_difference_in_means("Post HS?", "Age")
ps_hpw = avg_difference_in_means("Post HS?", "Hours per Week")
```

[34]:
```python
married_age = avg_difference_in_means("Ever Married?", "Age")
married_hpw = avg_difference_in_means("Ever Married?", "Hours per Week")
```

[35]:
```python
k_age = avg_difference_in_means("50K?", "Age")
k_hpw = avg_difference_in_means("50K?", "Hours per Week")
```

[36]:
```python
print("P-values for average difference in Age between Male and Female is:",␣
 ↪gender_age)
print("P-values for average difference in Hours Per Week between Male and␣
 ↪Female is:", gender_hpw)
print("P-values for average difference in Age between people went to HS and␣
 ↪Didn't is:", ps_age)
print("P-values for average difference in Hours Per Week between people went to␣
 ↪HS and Didn't is:", ps_hpw)
print("P-values for average difference in Age between Married vs not Married is:
 ↪", married_age)
print("P-values for average difference in Hours Per Week between Married vs not␣
 ↪Married is:", married_hpw)
print("P-values for average difference in Age between people with 50K+ Salaries␣
 ↪and people without is:", k_age)
print("P-values for average difference in Hours Per Week between people with␣
 ↪50K+ Salaries and people without is:", k_hpw)
```

```
P-values for average difference in Age between Male and Female is: 0.0218
P-values for average difference in Hours Per Week between Male and Female is:
0.00588
P-values for average difference in Age between people went to HS and Didn't is:
0.35788
P-values for average difference in Hours Per Week between people went to HS and
Didn't is: 0.53764
P-values for average difference in Age between Married vs not Married is: 0.0
P-values for average difference in Hours Per Week between Married vs not Married
is: 0.05072
P-values for average difference in Age between people with 50K+ Salaries and
people without is: 0.0
P-values for average difference in Hours Per Week between people with 50K+
Salaries and people without is: 0.12248
```

1.3.3 c) Naive P-Val Threshold: 0.05

With a Naive P-value Threshold 0.05, following null hypothesis can be rejected: - No age difference between Male and Female - No difference in Working hour per week between Male and Female - No age difference between people married vs not married - No difference in Working hour per week between people married vs not married - No age difference between people making 50k+ salary vs people making less than 50k

1.3.4 d) FWER <= 0.05,

[ ]:
```
x = 0.05/25000
print("P-val threshold with Bonferroni Correction is:", x)
```

P-value threshold with Bonferroni correction is 0.000002. Under this p-value threshold, following null hypothesis can be rejected: - No age difference between people married vs not married - No age difference between people making 50k+ salary vs people making less than 50k

1.3.5 e) FDR <= 0.05

B-H algorithm gives a p-val in between α (0.05) and Bonferroni Correction. So two null hypothesis rejected under p-val with Bonferroni Correction will be rejected with B-H Algorithm's p-val threshold.

Need to calculate the B-H p-val of the 3 other Null hypothesis, which include: - Age Difference between Male and Female - Working Hour per Week between Male and Female - Working Hour per Week between people married vs not married

[87]:
```
def b_h_avg_difference_in_means(dummy, numeral, alpha):

    differences = make_array()
    observed = mean_different(df, dummy, numeral)
    repetitions = 25000

    for i in np.arange(repetitions):
        new_difference = one_simulation(dummy, numeral)
        differences = np.append(differences, new_difference)
    p_val = make_array()
    for i in differences:
        p_val = np.append(p_val, (np.count_nonzero(differences >= i) /
 ↪repetitions))
    p_val.sort()
    result_table = Table().with_columns("p-value", p_val)

    k = np.arange(1, repetitions+1)
    result_table['k'] = k
    result_table['b_h'] = result_table['k']/len(p_val)* alpha
    result_table['diff'] = result_table['p-value'] - result_table['b_h']
    return result_table
```

[88]:
```python
gender_age_bh = b_h_avg_difference_in_means("Gender", "Age", 0.05)
```

[89]:
```python
gender_age_bh
```

[89]:
```
p-value | k    | b_h     | diff
4e-05   | 1    | 2e-06   | 3.8e-05
8e-05   | 2    | 4e-06   | 7.6e-05
0.00012 | 3    | 6e-06   | 0.000114
0.00016 | 4    | 8e-06   | 0.000152
0.0002  | 5    | 1e-05   | 0.00019
0.00024 | 6    | 1.2e-05 | 0.000228
0.00028 | 7    | 1.4e-05 | 0.000266
0.00032 | 8    | 1.6e-05 | 0.000304
0.0004  | 9    | 1.8e-05 | 0.000382
0.0004  | 10   | 2e-05   | 0.00038
… (24990 rows omitted)
```

[90]:
```python
gender_hour_bh = b_h_avg_difference_in_means("Gender", "Hours per Week", 0.05)
```

[91]:
```python
gender_hour_bh.item(100)
```

```
/opt/conda/lib/python3.9/site-packages/datascience/tables.py:222: FutureWarning:
Implicit column method lookup is deprecated.
  warnings.warn("Implicit column method lookup is deprecated.", FutureWarning)
```

[91]:
```
p-value | k    | b_h      | diff
0.0042  | 101  | 0.000202 | 0.003998
```

[92]:
```python
married_hour_bh = b_h_avg_difference_in_means("Ever Married?", "Hours per␣
↪Week", 0.05)
```

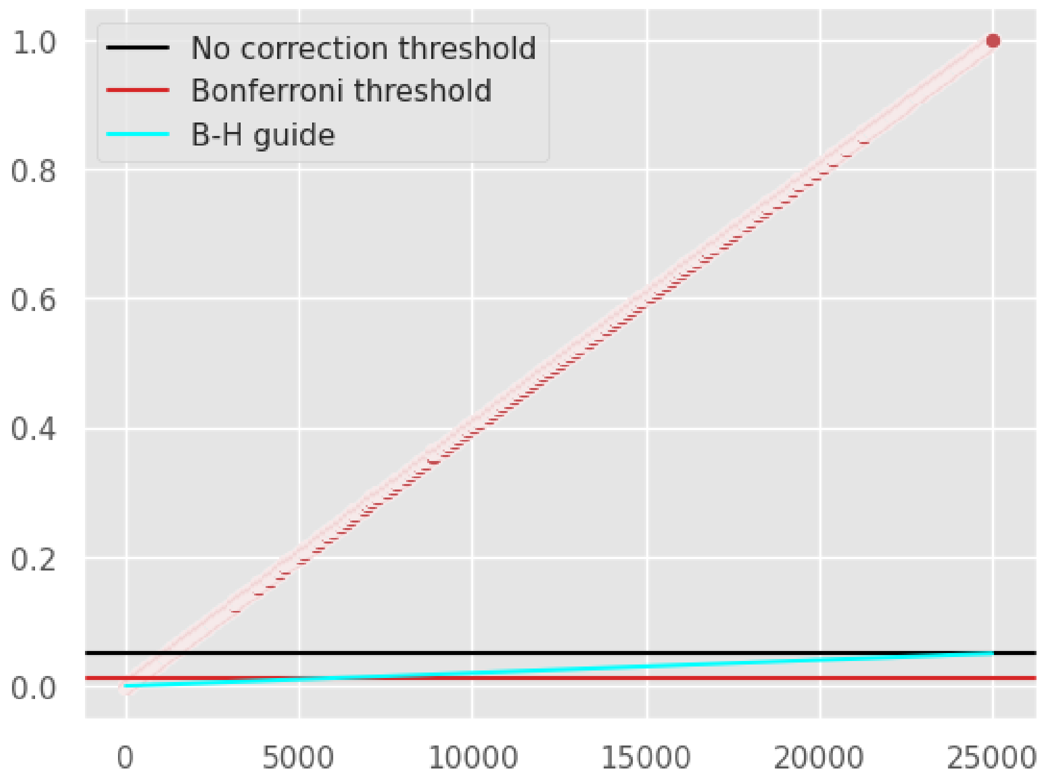
[93]:
```python
married_hour_bh
```

[93]:
```
p-value | k    | b_h     | diff
4e-05   | 1    | 2e-06   | 3.8e-05
8e-05   | 2    | 4e-06   | 7.6e-05
0.00012 | 3    | 6e-06   | 0.000114
0.0002  | 4    | 8e-06   | 0.000192
0.0002  | 5    | 1e-05   | 0.00019
0.00028 | 6    | 1.2e-05 | 0.000268
0.00028 | 7    | 1.4e-05 | 0.000266
0.00032 | 8    | 1.6e-05 | 0.000304
0.00036 | 9    | 1.8e-05 | 0.000342
0.0004  | 10   | 2e-05   | 0.00038
… (24990 rows omitted)
```

[94]:

```
alpha = 0.05
sns.scatterplot(x = gender_hour_bh['k'], y=gender_hour_bh['p-value'], color =
 ↪'r')
plt.axhline(alpha, label='No correction threshold', color='black')
plt.axhline(alpha / len(gender_hour_bh), label='Bonferroni threshold',
 ↪color='tab:red')
plt.plot(gender_hour_bh['k'], gender_hour_bh['b_h'], label='B-H guide',
 ↪color='cyan')
plt.legend()
```

[94]: <matplotlib.legend.Legend at 0x7fc92428db20>



[ ]:

f)
FWER is the lowest bound because it is calculated assuming all cases are exclusive, but that's usually not true. And it is about the probability of at least one FP, which makes this value extremely small. The FWER has a fixed p-val cut off for different features, but B-H gives different p-val cut offs. As a result, FWER rejects more null hypothesis.

g) Marriage Status can be binarized according to current marriage status instead of ever married. This is because