Test a Perceptual Phenomenon

January 21, 2018

0.0.1 Analyzing the Stroop Effect

Perform the analysis in the space below. Remember to follow the instructions and review the project rubric before submitting. Once you've completed the analysis and write up, download this file as a PDF or HTML file and submit in the next section.

(1) What is the independent variable? What is the dependent variable?

Independent variable: Congruence of words Dependent variable: Response time

(2) What is an appropriate set of hypotheses for this task? What kind of statistical test do you expect to perform? Justify your choices.

We want to see if there is a difference in response time when participants are shown words in different considitions, in this case, words whose names match the colors in which they are printed and words whose names do not match the colors in which they are printed.

We can run a hypothesis testing on the mean response time associated with each condition and see if there is a difference. The hypotheses should be set up as the followings (ţ here represents average response time):

H0: There is no significant difference in the population average response time on stating the colors in a congruent or incongruent condition

H1: There is significant difference in the population average response time on stating the colors in a congruent or incongruent condition

Since we do not know anything about the population (Including the standard deviation), and we have a relatively small sample with a size of 24 participants, performing a z-test is not really a good call.

And we are going to perform a paired t-test as this is a comparison of two different methods of measurement where the measurements are applied to the same subjects.

(3) Report some descriptive statistics regarding this dataset. Include at least one measure of central tendency and at least one measure of variability. The name of the data file is 'stroopdata.csv'.

In [1]:
```
### Import packages that are needed for the analysis

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import scipy.stats as st
sns.set_style('darkgrid')
%matplotlib inline
```

In [2]:
```
### Preliminary investigation on the dataset

df = pd.read_csv('stroopdata.csv')

df.isnull().any()
```

Out[2]:
```
Congruent      False
Incongruent    False
dtype: bool
```

In [3]:
```
df.describe()
```

Out[3]:

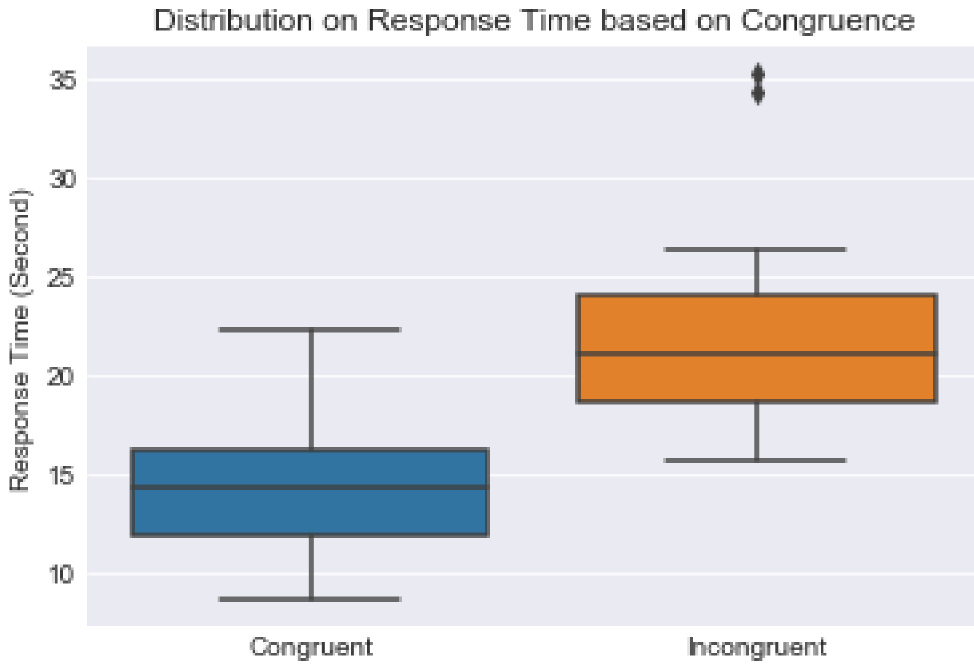
	Congruent	Incongruent
count	24.000000	24.000000
mean	14.051125	22.015917
std	3.559358	4.797057
min	8.630000	15.687000
25%	11.895250	18.716750
50%	14.356500	21.017500
75%	16.200750	24.051500
max	22.328000	35.255000

In [4]:
```
### Get the inter-quatile range

print(st.iqr(df.Congruent), st.iqr(df.Incongruent))
```

```
4.3055 5.33475
```

In [5]:
```
sns.boxplot(data=df)
plt.title('Distribution on Response Time based on Congruence')
plt.ylabel('Response Time (Second)')
```

Out[5]:
```
<matplotlib.text.Text at 0x117758d30>
```



Based on the preliminary investigation, people obviously need more time to respond when being shown incongruent words. On average, they need about 8 more seconds to respond. There is also a greater standard deviation when people see incongruent words. The SD is about 1.2 higher. On top of that, the inter-quatile range is 5.335 seconds (Incongruent), compared to 4.3055 seconds (Congruent). In a nutshell, there is a greater spread with incongruence.

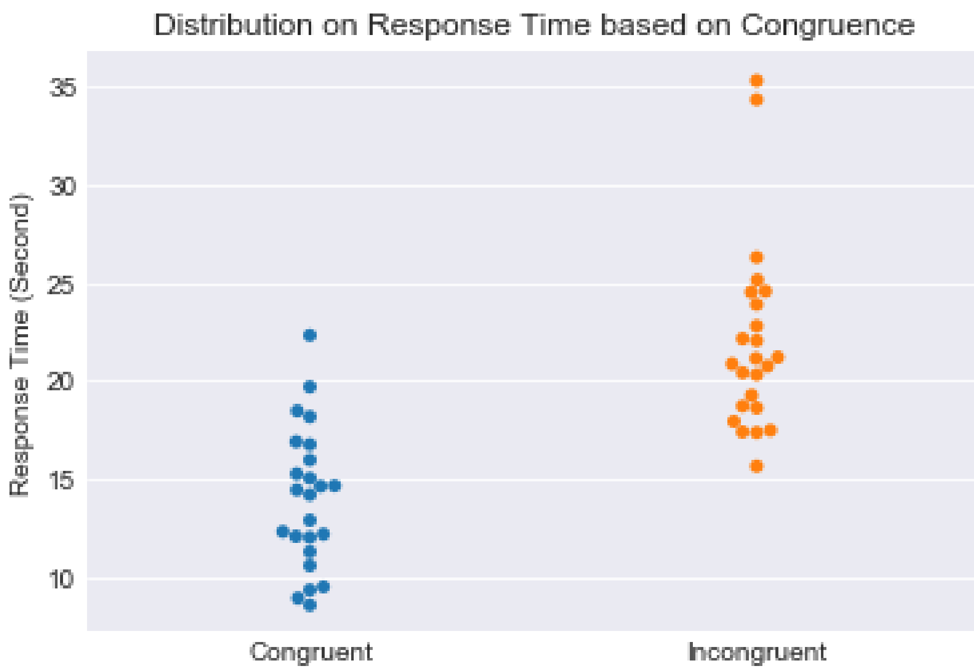
If we dive deeper and look at the boxplot, we can see that the congruent group performs obviously better than the incongruent group do, in a sense of response time. 75% participants of the congruent group responded at a time which only less than 25% participants of the incongruent group could do.

Although at this point we cannot draw any conslusions yet, this first glance of the dataset already gives us a general idea of the impact of congruence on people's response time.

(4) Provide one or two visualizations that show the distribution of the sample data. Write one or two sentences noting what you observe about the plot or plots.

```
In [6]: ### Build the visualizations here

        sns.swarmplot(data=df)
        plt.title('Distribution on Response Time based on Congruence')
        plt.ylabel('Response Time (Second)')
```

```
Out[6]: <matplotlib.text.Text at 0x117861ac8>
```



The swarmplot aligns with the boxplot above which shows that people obviously need more time to respond when being shown incongruent words.

(5) Now, perform the statistical test and report the results. What is the confidence level and your critical statistic value? Do you reject the null hypothesis or fail to reject it? Come to a conclusion in terms of the experiment task. Did the results match up with your expectations?

```
In [7]: ### 95% Confidence level and its associated critical values

        st.t.interval(alpha=0.95, df=23)

Out[7]: (-2.0686576104190406, 2.0686576104190406)

In [8]: ### 99% Confidence level and its associated critical values

        st.t.interval(alpha=0.99, df=23)

Out[8]: (-2.8073356837675227, 2.8073356837675227)

In [9]: ### T-statistic and P-value

        t_stat, p_val = st.ttest_rel(df.Incongruent, df.Congruent)

        print('t-statistic = %6.3f, p-value = %6.4f' % (t_stat, p_val))

t-statistic =  8.021, p-value = 0.0000
```

The t-statistic is equal to 8.021. This test statistic tells us how much the sample mean deviates from the null hypothesis.

If the t-statistic lies outside the quantiles (Critical values) of the t-distribution corresponding to our confidence level and degrees of freedom, we reject the null hypothesis.

A p-value of 0.0000 means we'd expect to see data as extreme as our sample due to chance about basically 0% of the time assuming the null hypothesis is true.

In this case, the p-value is lower than our significance level (1 - confidence level or 0.05, even 0.01) so we should reject the null hypothesis.

Let's double check by performing bootstrap sampling.

```
In [10]: ### Create a mean difference function.

         def diff_of_means(data_1, data_2):
             """Difference in means of two arrays."""

             # The difference of means of data_1, data_2: diff
             diff = np.mean(data_1) - np.mean(data_2)

             return diff
```

```
In [11]: ### Create a bootstrap function for drawing bootstrap replicates.

         def bootstrap_replicate_1d(data, func):
             return func(np.random.choice(data, size=len(data)))

         def draw_bs_reps(data, func, size=1):
             """Draw bootstrap replicates."""

             # Initialize array of replicates
             bs_replicates = np.empty(size)

             # Generate replicates
             for i in range(size):
                 bs_replicates[i] = bootstrap_replicate_1d(data, func)

             return bs_replicates
```

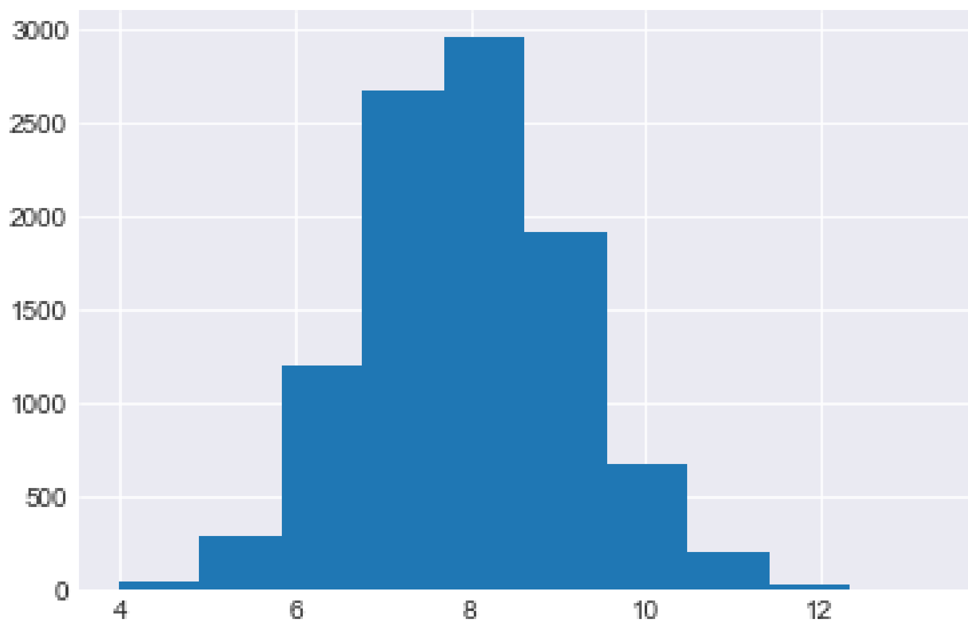
```
In [12]: # Compute the difference of the means
         mean_diff = diff_of_means(df.Incongruent, df.Congruent)

         # Get bootstrap replicates of means
         bs_replicates_incon = draw_bs_reps(df.Incongruent, np.mean, 10000)
         bs_replicates_con = draw_bs_reps(df.Congruent, np.mean, 10000)

         # Compute samples of difference of means
         bs_diff_replicates = bs_replicates_incon - bs_replicates_con

         # Compute 95% confidence interval
```

```
conf_int = np.percentile(bs_diff_replicates, [2.5, 97.5])

# Print the results
print('Difference of means =', mean_diff)
print('95% confidence interval =', conf_int)
```

```
Difference of means = 7.964791666666665
95% confidence interval = [  5.70418958  10.45997396]
```

```
In [13]: ### Bootstrap replicates distribution

         plt.hist(bs_diff_replicates)
```

```
Out[13]: (array([   41.,   287.,  1201.,  2680.,  2957.,  1913.,   682.,   204.,
                     29.,     6.]),
          array([  3.98633333,   4.91452917,   5.842725  ,   6.77092083,
                   7.69911667,   8.6273125 ,   9.55550833,  10.48370417,
                  11.4119    ,  12.34009583,  13.26829167]),
          <a list of 10 Patch objects>)
```

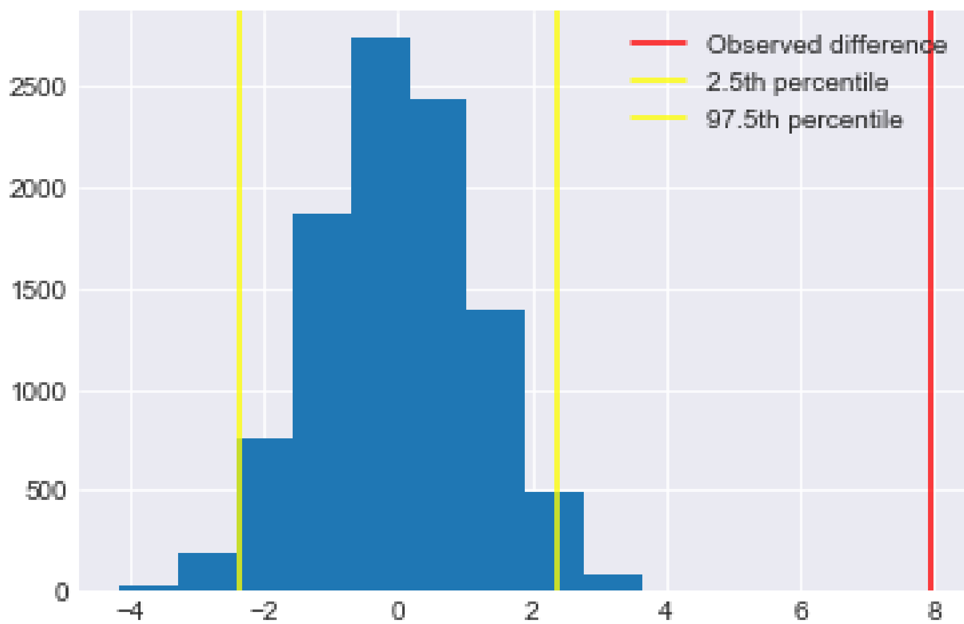



```
In [14]: ### Bootstrap replicates standard deviation

         bs_diff_replicates.std()
```

```
Out[14]: 1.1918606855744385
```

```
In [15]: ### Sampling distribution under null hypothesis

         null_vals = np.random.normal(0, bs_diff_replicates.std(), len(bs_diff_replicates))

         ptile_low = np.percentile(null_vals, 2.5)

         ptile_high = np.percentile(null_vals, 97.5)

         plt.hist(null_vals)
         plt.axvline(mean_diff, color='red', label='Observed difference')
         plt.axvline(ptile_low, color='yellow', label='2.5th percentile')
         plt.axvline(ptile_high, color='yellow', label='97.5th percentile')
         plt.legend()
```

```
Out[15]: <matplotlib.legend.Legend at 0x117b876a0>
```




```
In [16]: np.percentile(null_vals, [2.5, 97.5])
```

```
Out[16]: array([-2.34706717,  2.38581464])
```

The result aligns with the one from our t-test, in which we reject the null hypothesis.