Consequences of p-Hacking, Detection of p-Hacking, and the Winner's Curse

Jeffrey Mei

May 9, 2023

Abstract

It has come to recent attention that a great deal of published research findings cannot be replicated [Ioannidis, 2018]. While the extent of the problem varies by field, the problem has been attributed to a variety of bad statistical practices: underpowered studies, data dredging, p-hacking – an entire lexicon developed explicitly to verbalize statistical malpractice. The issue has become so pronounced that some journals have resorted to banning p-values altogether. There have, however, been some attempts at rectifying the impacts p-hacking has on the literature through the analysis of p-curves [Simonsohn et al.,] [Head et al., 2015].

In this simulation study, we attempt to illustrate the severity of each of these bad statistical practices. When a researcher commits statistical sin, how big of an impact does that have on their scientific result in terms of the bias introduced? We address this question by roughly following the analyses presented by other authors in the literature [Stefan and Schönbrodt, 2023] [Button et al., 2013]. Moreover, we also address the question of how well can p-hacking detection procedures can detect p-hacking. For this, we introduce original simulations to assess the question – fusing results analyzed in the p-hacking and importing them into the p-hacking detection procedure.

1 Background

1.1 *P*-Hacking

P-hacking refers to a multitude of behaviors that yields an over-optimistic p-value. In its most nefarious form, a researcher could manipulate the data so that the resulting analysis yields a p-value below 0.05. However, often times, p-hacking is not so malicious. For example, an investigator may remove perceived "outliers," which may result in an inflation of an effect that is imagined by the investigator. Alternatively, an investigator may conduct an analysis mid-experiment, recognize that they have not achieved statistical significance, and thus increase the number of samples in hopes of reaching reaching a desired scientific conclusion.

In this report, we will investigate two forms of p-hacking and study their effect on false-positives. In particular, we investigate p-hacking by selecting reporting of dependent variable and p-hacking by data-peeking. In doing so, we illuminate which behaviors are most egregious. Later, in Section 1.3, we will investigate how powerful one particular p-hacking detection method is [Head et al., 2015].

1.1.1 Selective Reporting of Dependent Variable

One of the most common forms of p-hacking is to collect multiple dependent variables, test them against the independent variable, and only report the significant ones. In failing to report all variables tested, the investigator inflates the likelihood of reaching a statistically significant result. Another factor that impacts the number of false-positives that results from selective reporting of dependent variables is how correlated the dependent variables are.

That is, the more dependent variables there are, the more chances there are of finding a dependent variable that reaches statistical significant. However, the more correlated the dependent variables are, the less the dependent variables vary from one another. Therefore, one would expect that highly correlated dependent variables should not impact the false-positive rate too adversely.

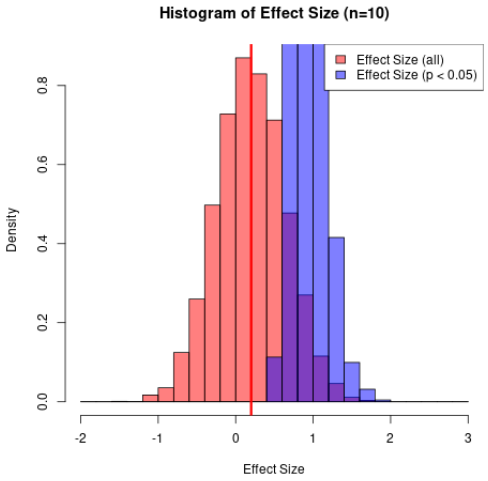
1.1.2 Data Peeking

Another common form of p-hacking includes data peeking, where investigators look at their data before the entire data collection process is complete. In doing so, this may influence their decision to stop collecting data when significance is reached. Alternatively, it could encourage the researcher to collect more data in hopes of finding statistical evidence supporting their hypothesis. In basing the sample size based on statistical significance, the chances of reaching a dubious statistically significant claim is inflated.

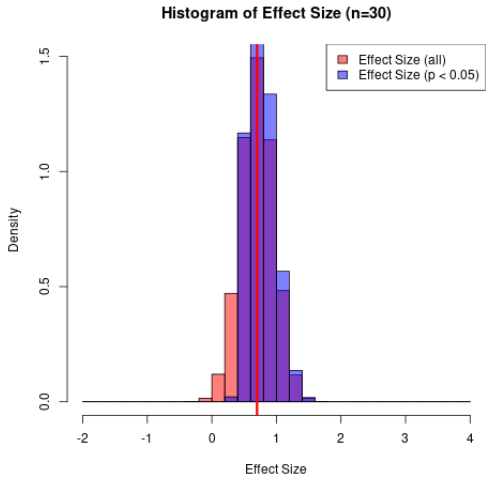
1.2 Winner's Curse

It is well known that underpowered studies inflate the number of false positive findings. However, another consequence is that the effect size tends to be inflated [Button et al., 2013]. The idea behind the winner's curse is that when sample sizes are small, it requires an exceptionally large signal to achieve statistical significance. Consequently, underpowered studies that are published tend to report overly optimistic effect sizes. Subsequent replication studies often result in a more accurate, but smaller effect size.

Figure 1 illustrates the winner's curse. In a simulation with $10,000$ replications, we see that of the simulations that were able to reach a statistical significance, the estimated effect size exceeded the true effect size. That is, the distribution of estimated effects illustrated in blue is shifted to the right of the true effect. We can also see in (b), where the true effect size is larger, the inflation of estimated effect size becomes less prominent as the power increases.



(a) red: $N(0,1)$; blue: $N(0.2,1)$; $n = 10$.



(b) red: $N(0,1)$; blue: $N(0.7,1)$; $n = 30$.

Figure 1: 10,000 simulations of one-sided t-tests with 0.05 significance level, comparing two samples of size n with different means. Vertical red line indicates true effect size.

1.3 Detecting P-Hacking

One explanation for the replicability crisis is the prevalence of p-hacking. When performing an analysis that hinges on a collection of p-valuse, such as with a meta-analysis, p-hacking may affect the conclusions. Consequently, it is of interest to analyze the extent of p-hacking in a corpus of literature.

Head et al. analyze the presence of p-hacking in various fields of research by text-mining and studying the p-curves – the distributions of p-values (see Figure 2) [Head et al., 2015]. If the null hypothesis is true, then one should expect that the distribution of p-values will be uniformly distributed (as seen in the purple histogram in Figure 2). However, If there is a true effect, then the p-values will become more right skewed as the effect gets stronger (as seen in the red and green histograms in Figure 2).

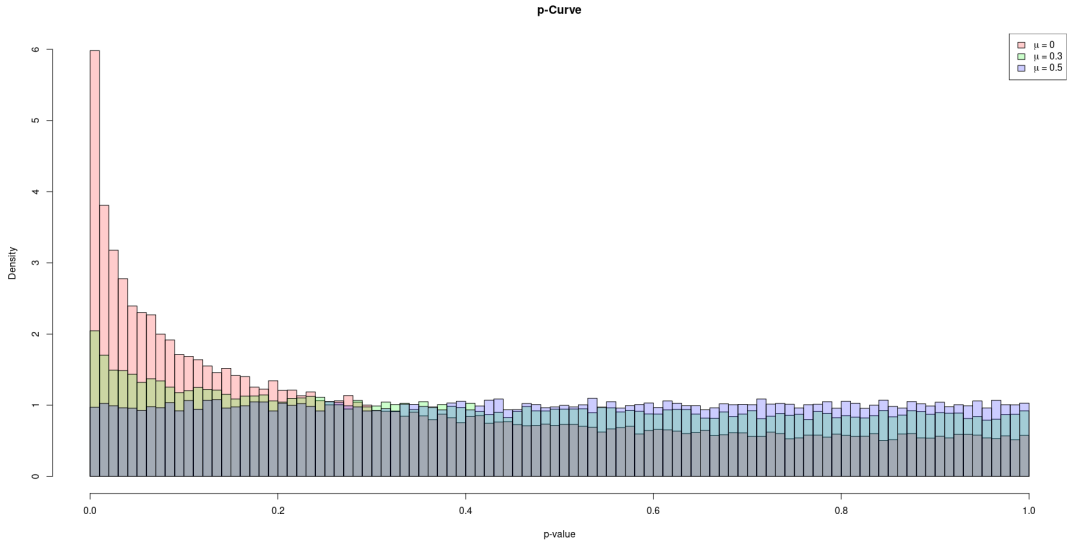
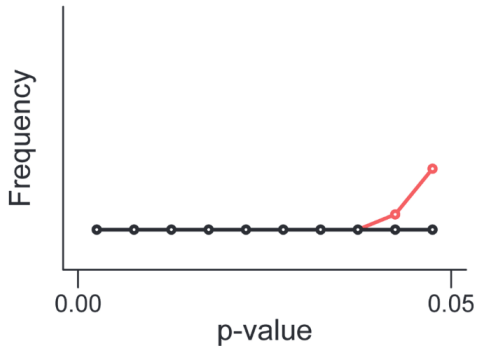
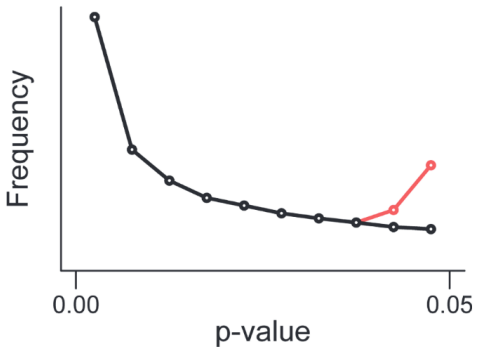


Figure 2: The p-curve illustrates how the data skews right as the effect size grows.



(a) The p-curve with no effect under p-hacking.



(b) The p-curve with true effect under p-hacking.

Figure 3: Figures 2a and 2b from [Head et al., 2015]

Consider a case where an investigator reaches a relatively low p-value, but still above 0.05, and the investigator p-hacks the result to reach statistical significance. While it may be difficult to catch the investigator in p-hacking, it is possible to detect p-hacking in a corpus of literature by analyzing the p-curve. In particular, there is a distinct bump in the p-curve just below 0.05, as seen in Figure 3. Presumably, this bump occurs because when a result that is slightly over 0.05 is p-hacked to reach statistical significance, then the proportion of p-values that are just above 0.05 is redistributed to be p-values below 0.05. That is, when p-hacking is prevalent in the literature, the number of p-values under 0.05 is inflated.

Now that we have seen that p-hacking leaves a distinct signature, one strategy to catch p-hacking is to compare the number of p-values just below 0.05 against the number of p-values that are more significant. In particular, in this study, we will compare the number of p-values that fall into the lower bin $0 < p < 0.025$ against the number of p-values that fall into the upper bin $0.025 \leq p < 0.05$. If there are more p-values in the upper bin than the lower bin, then this may be indicative of p-hacking. To compare the counts, we will apply a one-sided binomial test at a 0.05 significance level.

2 Methodology

In this simulation study, we use 10,000 simulations for each situation. The standard deviations are not reported here because they are all well below 10^{-3} and the results in this report are only reported up to the second decimal place.

We simulate various scenarios of p-hacking to understand the consequence each p-hacking method has on false positive rate. There are a number of simulation studies on p-hacking in the literature, but this set of simulations will be based from a comprehensive simulation study by Stefan and Schonbrodt (2023). We reproduce a small selection of their results in this study.

An additional consequence of p-hacking is that of the winner's curse. When there are small effect sizes (or no effect size), any perceived effect is exaggerated. We follow the analysis presented by Button et al. (2013).

Not only are we interested in the consequences of p-hacking, but another topic of interest is detecting p-hacking. In this study, we primarily study the method presented by Head et al. (2015), which uses a binomial test to assess the presence of p-hacking in a corpus. While the authors of that study introduce a method to detect p-hacking, they do not present any simulation results to validate the performance of their method. We test their procedure under various situations to assess the power of their procedure.

3 Results

3.1 Selective Reporting of Dependent Variables

We use two-sided t-tests between an independent sample against up to K dependent variables. We assume that the independent sample has a standard normal distribution and the K dependent response variables follow a multivariate normal distribution with mean 0 and covariance Σ, where $\Sigma_{ii} = 1$ and $\Sigma_{ij} = \rho$ varies between simulations for $i \neq j$. Since all of the samples have mean 0, there is no true effect and therefore any significant finding is a false positive.

We report the first significant result within the K dependent variables. If we have tried all K dependent variables and still do not find statistical significance, we report the smallest p-value.

	Correlation (ρ)					
n	0	0.1	0.3	0.5	0.7	0.9
10	0.17	0.17	0.15	0.14	0.12	0.08
30	0.19	0.18	0.16	0.14	0.12	0.09
50	0.18	0.18	0.16	0.14	0.12	0.09
100	0.19	0.17	0.15	0.14	0.11	0.09
300	0.18	0.17	0.16	0.14	0.12	0.09

(a) $K = 5$

	Correlation (ρ)					
n	0	0.1	0.3	0.5	0.7	0.9
10	0.29	0.27	0.23	0.19	0.15	0.10
30	0.28	0.27	0.23	0.20	0.15	0.10
50	0.29	0.27	0.24	0.19	0.15	0.11
100	0.29	0.28	0.23	0.20	0.15	0.11
300	0.29	0.27	0.23	0.20	0.15	0.10

(b) $K = 10$

Figure 4: False positive rate for number of dependent variables collected K

We see that as expected, as the correlation increases, the extent of the false-positive rate decreases. We also notice that the sample size bears little impact on the inflated false-positive rate. This result matches with the analysis done by Stefan et al. (2023).

Moreover, we see that by increasing the number of dependent variables that we collected and test against, the number of false-positives also increases. Therefore, the factors that impact the severity of the bias introduced from p-hacking via selective reporting of dependent variables include the number of dependent variables collected, and how correlated the dependent variables are.

3.2 Data Peeking

Here, we are assuming that the investigator is collecting data that is standard normal and tests for statistical significance using a two-sided one-sample t-test. That is, there is no true effect, and any significant finding is necessarily a false-positive. We also assume that the investigator analyzes the data for statistical significant in regular increments. In particular, we test the case where the investigator peeks at the data in increments of 1, 5, and 10 (admittedly, it would take a particularly unscrupulous investigator to check for statistical significance after the collection of every data point and at such regular intervals). If statistical significance is reached, the data collection process stops, and the significant p-value is reported. If after n points are collected, and statistical significance is not reached, the most significant p-value is reported.

	Peek Increments (k)		
n	1	5	10
10	0.21	0.10	0.05
30	0.28	0.16	0.11
50	0.33	0.19	0.14
100	0.37	0.23	0.19
300	0.47	0.35	0.27

Figure 5: False positive rates for varying sample size n and peek increments K

It is clear that as the fixed sample size increases, there are more opportunities to data-peek. Therefore, we see that the false-positive rate increases with maximum sample size n. In addition, we also see that the more frequent the data-peeking, the higher the inflation of false-positives.

3.3 Winner's Curse

Somewhat different from the previous two sections, we analyze the effect the winner's curse has on reported research findings. Again, we assume the data is normally distributed with variance 1. However, in this case, we vary the true effect size so that it is non-zero.

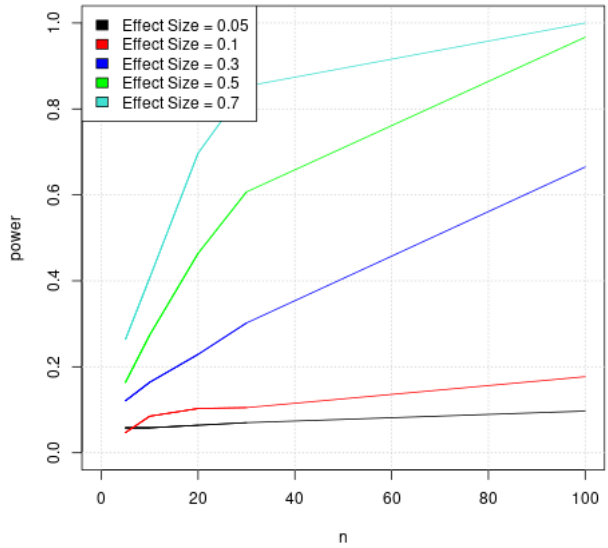


Figure 6: Power analysis with varying levels of effect size and sample size.

	Effect Size (θ)				
Sample Size	0.05	0.1	0.3	0.5	0.7
5	1.18	1.12	1.00	0.85	0.74
10	0.83	0.82	0.64	0.51	0.39
20	0.59	0.56	0.41	0.27	0.16
30	0.49	0.44	0.29	0.16	0.07
100	0.25	0.21	0.07	0.01	0.00

Figure 7: Each entry indicates the bias in effect size $bias(\theta) = \hat{\theta} - \theta$

From the above table, we see that indeed, as the power of a scenario considered decreases, the effect size becomes increasingly biased. In the worst case scenario, where a particular study has a tiny sample size and another tiny effect size, the bias is the largest. In contrast, it is worthwhile to note that it requires a fairly large sample size and effect size to tame the bias in the exaggeration of the effect.

3.4 *P*-Hacking Detection

In this set of simulations, we analyzed the ability of the p-hacking detection procedure described by Head et al. (2015) to detect p-hacking when there are varying levels of

data-peeking. That is, we use a one-sided binomial test to determine whether or not the upper bin has more p-values than the lower bin. We are considering a particularly contrived scenario where every p-value considered is p-hacked. Consequently, whenever the binomial test results in a significant p-value, the test has successfully determined that the collection of p-values is indeed p-hacked.

In the tables presented in Figure 8 show the results of the simulation. We vary the number of p-values analyzed, m. If we were trying to determine if p-hacking is prevalent in the cancer research literature for example, m would represent the number of p-values in the corpus of medical literature considered. In contrast, n would represent the maximum sample size from data-peeking. Lastly, k is the frequency at which the data is analyzed for statistical significance in data-peeking.

	Peek Increment (k)		
m	1	5	10
50	0.24	0.01	0.00
100	0.50	0.02	0.01
500	0.99	0.07	0.03

(a) $n = 10$

	Peek Increment (k)		
m	1	5	10
50	0.52	0.05	0.02
100	0.82	0.07	0.02
500	1.00	0.24	0.08

(b) $n = 20$

	Peek Increment (k)		
m	1	5	10
50	0.81	0.14	0.05
100	0.99	0.31	0.09
500	1.00	0.88	0.37

(c) $n = 50$

Figure 8: p-hacking detection results on data-peeking style p-hacking

From the results, we see the desired pattern that as the severity of p-hacking increases, the better our test is at detecting p-hacking. However, it looks like there must be a large amount of p-hacking for the procedure to yield a significant result. For example, if *every* researcher ($m = 50$) in a field of study ran experiment where up to 50 observations are taken, and every researcher data-peeks every 5 observations (behavior that would yield a +10% inflation of false-positives according to Figure 5), our p-hacking detection procedure would only detect p-hacking 14% of the time. Although, the test appears to struggle when the number of p-values under consideration is small, the power of the test improves significantly as the number of p-values in the corpus increase.

Therefore, these simulations indicate that it would require either a significant amount of p-hacking in the literature, or a large corpus of p-values to detect p-hacking effectively.

4 Conclusion

All in all, there are a multitude of statistical issues that plague the scientific research enterprise. In this report, we study a few of them in detail using simulations. In particular, we have examined the effects of p-hacking on the number of dubious claims. We have also analyzed the winner's curse on the exaggeration of proclaimed effects. Finally, we present original simulations on how effective a test proposed by Head et al. (2015) is in detecting p-hacking.

We see that of the two p-hacking scenarios that we considered, data-peeking seems to yield particularly inflated false-positives as compared to p-hacking by selectively reporting dependent variables. Many of the data-peeking scenarios that are considered in this study would only be realistic in the case that an investigator is particularly unscrupulous and persistent. In the context of the entirety of science, one should hope that these characters are in the minority. However, even under more modest scenarios where investigators peek infrequently, it can be expected that the number of dubious claims is inflated by two or three times what it should be.

While data-peeking appears to be the worser of the two types of p-hacking considered in this study, selective reporting of dependent variables also yields a substantial inflation of dubious claims. While I assume that most investigators know that data-peeking is

improper statistical technique, I surmise that a smaller number of investigators are wary of the dangers of selective reporting of dependent variables. The simulations considered here paint a grim picture where even if the dependent variables considered are highly correlated ($\rho = 0.9$), the number of false-positives nearly doubles by considering multiple dependent variables.

Moreover, whether claims are dubious, or real but small, our simulations indicate that whatever reported the effect is, it is likely to be exaggerated by the winner's curse. In fact, it requires a fairly large powered test to tame the bias incurred. Therefore, one conclusion from these results is that one should be wary of effect sizes cited in the literature unless the study is well-powered.

Finally, while p-hacking may be pervasive in the literature, some authors have proposed means of testing for the presence of p-hacking. We analyze one such procedure and find that it requires a significant amount of p-hacking to detect p-hacking. We considered a particularly contrived scenario where every p-value is p-hacked to evaluate the test. Moreover, due to time constraints, the scenarios considered in this report only included scenarios where the true effect is non-existant. In reality, it would be difficult to imagine that there would be so much funding and research interest in a completely imagined effect. With that said, since a true effect introduces a right skew, one can expect that the power of the test suffers. Therefore, the situations considered in this study can be viewed as a upper bound on performance of detecting p-hacking.

In the future, it would be interesting to consider more p-hacking scenarios, and to make them more realistic. For example, it is also possible that investigators do not stick to one strategy of p-hacking, but also engage in others. Alternatively, it would be interesting to consider the p-hacking detection method in a more realistic scenario where p-values follow from a mixed distribution such that some p-values are generated fairly, and another proportion of p-values are p-hacked.

In conclusion, while the replicability crisis rages, it is clear that more work must be done to address the p-hacking and other statistical malpractice prevalent in the scientific literature. This study is only one analysis that illustrates the consequences of statistical malpractice, and analyzes a method to detect it.

References

[Button et al., 2013] Button, K. S., Ioannidis, J. P. A., Mokrysz, C., Nosek, B. A., Flint, J., Robinson, E. S. J., and Munafò, M. R. (2013). Power failure: why small sample size undermines the reliability of neuroscience. *Nature Reviews Neuroscience*, 14(5):365–376. Number: 5 Publisher: Nature Publishing Group.

[Head et al., 2015] Head, M. L., Holman, L., Lanfear, R., Kahn, A. T., and Jennions, M. D. (2015). The Extent and Consequences of P-Hacking in Science. *PLOS Biology*, 13(3):e1002106. Publisher: Public Library of Science.

[Ioannidis, 2018] Ioannidis, J. P. (2018). Why most published research findings are false. In *Getting to Good: Research Integrity in the Biomedical Sciences*, volume 2, pages 2–8. Springer International Publishing. Issue: 8 ISSN: 15491277.

[Simonsohn et al.,] Simonsohn, U., Nelson, L. D., and Simmons, J. P. P-Curve: A Key to the File-Drawer.

[Stefan and Schönbrodt, 2023] Stefan, A. M. and Schönbrodt, F. D. (2023). Big little lies: a compendium and simulation of p-hacking strategies. *Royal Society Open Science*, 10(2):220346. Publisher: Royal Society.

5 Appendix

```
# ================================
# Figure 1: Distribution of P-Values
# ================================


# Set Parameters
N = 10000
n_seq = c(10, 30, 50, 100, 300)
```

```
n = 30
mu = 0

# Run Test
sim_ttest <- function(n, mu){

    # Generate Data
    X = rnorm(n, mean=mu)
    # Evalute P-Value
    return(t.test(X)$p.val)
}



# ====================================
# Figure 2: P-Values With Real Effect
# ====================================
breaks = seq(0, 1, 0.01)
N = 10000

# Simulate Values
pvals0 = replicate(N, sim_ttest(n, mu=0))
pvals1 = replicate(N, sim_ttest(n, mu=0.1))
pvals2 = replicate(N, sim_ttest(n, mu=0.2))

# Generate Histogram Counts
pvals0_ct = hist(pvals0, breaks=breaks, freq=F,
                 main='Distribution of p-Values',
                 col=rgb(1, 0, 0, 0.2), ylim=c(0, 2.5))$counts / N * 100
pvals1_ct = hist(pvals1, breaks=breaks, freq=F, add=T,
                 main='Distribution of p-Values',
                 col=rgb(0, 1, 0, 0.2))$counts / N * 100
```

```
pvals2_ct = hist(pvals2, breaks=breaks, freq=F,
                 main='Distribution of p-Values', add=T,
                 col=rgb(0, 0, 1, 0.2))$counts / N * 100

# Plot P-Curves
plot(NA,
     xlim=c(range(breaks)),
     ylim=c(0, max(pvals0_ct, pvals1_ct, pvals2_ct)),
     main='P-Curves', xlab='p-Value', ylab='Density')

lines(breaks[-length(breaks)] + 0.025, pvals0_ct, col=1)
lines(breaks[-length(breaks)] + 0.025, pvals1_ct, col=2)
lines(breaks[-length(breaks)] + 0.025, pvals2_ct, col=3)
points(breaks[-length(breaks)] + 0.025, pvals0_ct, col=1)
points(breaks[-length(breaks)] + 0.025, pvals1_ct, col=2)
points(breaks[-length(breaks)] + 0.025, pvals2_ct, col=3)
grid()
legend('topright', c(expression(paste(mu, ' = 0')),
                     expression(paste(mu, ' = 0.3')),
                     expression(paste(mu, ' = 0.5'))),
       fill=c('black', 'red', 'green'))


# ================================
# Figure 2: P-Values under P-Hacking
# ================================
# Run Test
sim_ttest_phacked <- function(n, mu, phack_prob=0){

    # Generate Data
    X = rnorm(n, mean=mu)
```

```
    # Evalute P-Value
    pval = t.test(X)$p.val

    # Phack
    phack_flag = rbinom(1, 1, phack_prob)
    if((phack_flag == 1) & (0.05 <= pval) & (pval <= 0.1)){
        pval = max(0, pval - abs(rnorm(1, mean=0.01, sd=0.05)))
    }
    return(pval)
}

# Simulate Values
N = 50000
breaks = seq(0, 1, 0.01)
phack_prob = 0.5
pvals0 = replicate(N, sim_ttest_phacked(n, mu=0, phack_prob))
pvals1 = replicate(N, sim_ttest_phacked(n, mu=0.1, phack_prob))
pvals2 = replicate(N, sim_ttest_phacked(n, mu=0.2, phack_prob))


# Generate Histogram Counts
pvals2_ct = hist(pvals2, breaks=breaks, freq=F,
                 main='p-Curve',
                 ylim=c(0, max(pvals0_ct, pvals1_ct, pvals2_ct)),
                 xlim=c(0, 1),
                 xlab='p-value', ylab='Density',
                 col=rgb(1, 0, 0, 0.2))$counts / N * 100
pvals1_ct = hist(pvals1, breaks=breaks, freq=F, add=T,
                 col=rgb(0, 1, 0, 0.2))$counts / N * 100
pvals0_ct = hist(pvals0, breaks=breaks, freq=F, add=T,
                 col=rgb(0, 0, 1, 0.2))$counts / N * 100
legend('topright', c(expression(paste(mu, ' = 0')),
```

```
                    expression(paste(mu, ' = 0.3')),
                    expression(paste(mu, ' = 0.5'))),
       fill=c(rgb(1, 0, 0, 0.2),
              rgb(0, 1, 0, 0.2),
              rgb(0, 0, 1, 0.2)))
abline(v=0.05, col='red')
abline(v=0.1, col='blue')


# Plot P-Curves
plot(NA,
     xlim=c(0, 0.2),
     ylim=c(0, max(pvals0_ct, pvals1_ct, pvals2_ct)),
     main='P-Curve', xlab='p-Value', ylab='Density')

lines(breaks[-length(breaks)], pvals0_ct, col=1)
lines(breaks[-length(breaks)], pvals1_ct, col=2)
lines(breaks[-length(breaks)], pvals2_ct, col=3)
points(breaks[-length(breaks)], pvals0_ct, col=1)
points(breaks[-length(breaks)], pvals1_ct, col=2)
points(breaks[-length(breaks)], pvals2_ct, col=3)
grid()
legend('topright', c(expression(paste(mu, ' = 0')),
                    expression(paste(mu, ' = 0.3')),
                    expression(paste(mu, ' = 0.5'))),
       fill=c('black', 'red', 'green'))


# Compare P-Hacked Curve to Regular Curve
breaks=seq(0, 1, 0.025)
pvals0 = replicate(N, sim_ttest_phacked(n, mu=0, phack_prob=0))
pvals1 = replicate(N, sim_ttest_phacked(n, mu=0, phack_prob=0.5))
```

```
hist(pvals0, breaks=breaks, freq=F,
     main='p-Curve',
     ylim=c(0, 2.5),
     xlim=c(0, 0.2),
     xlab='p-value', ylab='Density',
     col=rgb(1, 0, 0, 0.2))$counts / N * 100
hist(pvals1, breaks=breaks, freq=F, add=T,
     col=rgb(0, 1, 0, 0.2))$counts / N * 100
abline(v=0.05, col='red')
abline(v=0.1, col='blue')
legend('topright', c('not p-hacked',
                     'p-hacked'),
       fill=c(rgb(1, 0, 0, 0.2),
              rgb(0, 1, 0, 0.2)))
```

```
# ========================
# Establish Power of Test
# ========================
# - vary effect size

breaks = seq(0, 1, 0.025)
pvals0 = replicate(N, sim_ttest_phacked(n, mu=0, phack_prob=0))
pvals1 = replicate(N, sim_ttest_phacked(n, mu=0, phack_prob=0.5))

pval0_ct = hist(pvals0, breaks=breaks, freq=F,
                main='p-Curve',
                ylim=c(0, 2.5),
                xlim=c(0, 0.2),
                xlab='p-value', ylab='Density',
```

```
                col=rgb(1, 0, 0, 0.2))$counts
pval1_ct = hist(pvals1, breaks=breaks, freq=F, add=T,
                col=rgb(0, 1, 0, 0.2))$counts

df = data.frame(cbind(breaks, pval0_ct))



pval0_ct[2]
pval0_ct[3]
binom.test()



# Test with Few Sample Sizes



# ================================================
# Analyze P-Curve with Multiple Dependent Variables
# ================================================
selectively_report_dependent_var <- function(X, Y){

    # Set Variables
    n = nrow(Y)
    p = ncol(Y)

    # Save P-Values
    pvals = rep(NA, p)
    for(i in 1:p){
        pvals[i] = t.test(X, Y[,i])$p.val
    }
    return(pvals)
}
```

```
sim_pfishing <- function(n, S){

    # Generate Data
    p = nrow(S)
    X = rnorm(n)
    Y = mvrnorm(n, mu=rep(0, p), Sigma=S)

    # T-Test
    pvals = selectively_report_dependent_var(X, Y)
    is_sig = which(pvals < 0.05)

    if(length(is_sig) > 0){  # no significant values
        pval = pvals[is_sig[1]]
    }else{
        pval = min(pvals)
    }
        # take first pvalue
    return(pval)
}

find_first_sig <- function(pvals){

    is_sig = which(pvals < 0.05)

    if(length(is_sig) > 0){  # no significant values
        pval = pvals[is_sig[1]]
    }else{
        pval = min(pvals)
    }
    return(pval)
}
```

```
# ================================
# Figure 3: Testing for P-Hacking
# ================================
'''
Assuming fixed sample size of 30
- generate M p-values
- look at distribution
- attempt to detect p-hacking
- used author recommendation of 0.005 bin_size
'''

# Optional Stopping:
data_peeking <- function(n, step=1){

    # Generate Random Data
    X = rnorm(n)

    # Evaluate P-Values
    if(step == 1){
        steps = seq(step, n, by=step)
        pval = sapply(steps[-1], function(k) t.test(X[1:k])$p.val)
    }else{
        steps = seq(step, n, by=step)
        pval = sapply(steps, function(k) t.test(X[1:k])$p.val)
    }
    sig_idx = which(pval < 0.05)

    # Return
    #return(length(sig_idx) > 0)
    return(pval)
}
```

```
phack_detect <- function(pvals, bin_size=0.005){

    # Get Counts
    breaks = seq(0, 1, bin_size)
    cts = hist(pvals, breaks=breaks, plot=F)$counts

    # Locate Bins
    sig_bin_idx = which(breaks == 0.05)
    l_bin_idx = sig_bin_idx - 2
    r_bin_idx = sig_bin_idx - 1
    l_bin = cts[l_bin_idx] # 0.04  - 0.045
    r_bin = cts[r_bin_idx] # 0.045 - 0.05

    # Calculate p-Value
    pval = binom.test(c(l_bin, r_bin), alternative='less')$p.val
    return(pval)
}

sim_phack_detect <- function(M, n, step, bin_size=0.005){

    # Generate P-Hacked P-Values
    #pvals = replicate(M, sim_pfishing(n, S))

    # For Data-Peeking
    phacked_pvals = replicate(M, data_peeking(n, step=step), simplify=F)
    reported_pvals = unlist(lapply(phacked_pvals, function(x) find_first_sig(x)))

    # Detect P-Hacking
    pval = phack_detect(reported_pvals, bin_size=bin_size)
    #pval = phack_detect(reported_pvals, bin_size=0.005)
```

```
    return(pval)
}


N = 1000 # number of simulations
M = 100 # number of p-values in corpus
n = 10 # fixed sample size in each individual test
p = 10 # number of dependent variables collected
step = 5
#S = matrix(0.1, nrow=p, ncol=p)
#diag(S) = 1

# Simulate Detecting P-Hack
#pvals = replicate(N, sim_phack_detect(M, n, step, bin_size=0.005))
pvals = replicate(N, sim_phack_detect(M, n, step, bin_size=0.025))

# Calculate FPR
sum(pvals < 0.05) / N
hist(pvals, breaks=breaks)


# Draw Histogram
hist(reported_pvals, breaks=breaks/2)
abline(v=0.05, col='red')
abline(v=0.025, col='red')


# ==============
# Power Analysis
# ==============
# - vary sample size
# - bin_size
```

```
# - amount of p-hacking
#   - number of dependent variables
# - can be masked by real effect


bin_seq = c(0.005, 0.025)
M_seq = c(50, 100, 500)
step_seq = c(1, 5, 10)
n = 20

bin_size = 0.025
fpr_df = data.frame(matrix(NA, nrow=length(M_seq), ncol=length(step_seq)))
sd_df = data.frame(matrix(NA, nrow=length(M_seq), ncol=length(step_seq)))
for(i in 1:length(M_seq)){
    for(j in 1:length(step_seq)){
        print(paste(i, j))

        M = M_seq[i]
        step = step_seq[j]
        pvals = replicate(N, sim_phack_detect(M, n, step, bin_size=bin_size))

        # Save Results
        fpr_df[i, j] = sum(pvals < 0.05) / N
        sd_df[i, j] = sd(pvals)
        sd_df[i, j] = sd((pvals < 0.05) / N)
    }
}

breaks = seq(0, 1, 0.025)
hist(pvals, breaks=breaks, xlim=c(0, 0.2))

rownames(fpr_df) = M_seq
```

```
colnames(fpr_df) = step_seq

rownames(sd_df) = M_seq
colnames(sd_df) = step_seq


# Plot Results
fpr_df1 = data.frame(matrix(c(0.24, 0.01, 0.00,
                              0.50, 0.02, 0.01,
                              0.99, 0.07, 0.03),
                            nrow=length(M_seq), ncol=length(step_seq),
                            byrow=T))

fpr_df2 = data.frame(matrix(c(0.52, 0.05, 0.02,
                              0.82, 0.07, 0.02,
                              1.00, 0.24, 0.08),
                            nrow=length(M_seq), ncol=length(step_seq),
                            byrow=T))

fpr_df3 = data.frame(matrix(c(0.81, 0.14, 0.05,
                              0.99, 0.31, 0.09,
                              1.00, 0.88, 0.37),
                            nrow=length(M_seq), ncol=length(step_seq),
                            byrow=T))
rownames(fpr_df1) = M_seq
colnames(fpr_df1) = step_seq
rownames(fpr_df2) = M_seq
colnames(fpr_df2) = step_seq
rownames(fpr_df3) = M_seq
colnames(fpr_df3) = step_seq

fpr_df1 = cbind(fpr_df1, n=rep(10, 3), m=M_seq)
```

```
fpr_df2 = cbind(fpr_df2, n=rep(20, 3), m=M_seq)
fpr_df3 = cbind(fpr_df3, n=rep(50, 3), m=M_seq)

fpr_df = rbind(fpr_df1, fpr_df2, fpr_df3)

fpr_df[fpr_df$m == 50, ]
plot(NA, xlim=c(), ylim=c())

t_test <- function(x, alpha=0.0025, mu0=0){

    n = length(x)
    t_crit = qt(1 - alpha, df=n-1)
    t_stat = (mean(x) - mu0) / (sd(x) / sqrt(n))

    pval = pt(t_stat, n-1)
    return(pval)
}



p_hack <- function(){
    n = 30
    n0 = 5
    x = rnorm(n)
    pval = sapply(n0:n, function(n) t_test(x[1:n]))
    return(pval)
}



# ==============
# Plot P-Fishing
# ==============
plot(n0:n, p_hack(), type='l', ylim=c(0, 1))
```

```
for(i in 1:5){
    lines(n0:n, p_hack())
}
abline(h=0.05)
grid()



# =============
# Verify T-Test
# =============
N = 1000
pval = replicate(N, t_test(rnorm(n)))
sum(pval < 0.05) / N

'''
P-Hacking Variation:
- get data first
- ask questions later
'''



# Model Parameters
n = 20
p = 10

b_signal = c(3, -1, 2)
b_noise = rep(0, p - 3)

# Generate Model
B = c(b_signal, b_noise)
X = matrix(rnorm(n * p), nrow=n, ncol=p)
```

```
e = rnorm(n)
Y = X %*% B + e
alpha = 0.05

# Analyze P-Values
summary(lm(Y ~ X))


sim_lm <- function(n, p, B){

    # Generate Model
    X = matrix(rnorm(n * p), nrow=n, ncol=p)
    e = rnorm(n)
    Y = X %*% B + e

    # Analyze P-Values
    fit = summary(lm(Y ~ X))$coeff
    pvals = fit[,4]
    return(pvals)
}

sim_obj = replicate(100, sim_lm(n, p, B))

# Proportion Selected
apply(sim_obj, 1, function(x) sum(x < alpha) / length(x))


# Number of False Covariates Chosen
apply(sim_obj, 1, function(x) sum(x < alpha) / length(x))
```

```
noise_coef = sim_obj[(length(b_signal) + 1):(length(B) + 1), ]
apply(noise_coef, 2, function(x) sum(x < alpha))
    # number of false positives


# ------------------------------------------------------------------------

'''
Experimental Rerun
'''

n = 100
alpha = 0.05

phacking_rerun <- function(n, alpha=0.05){

    # Sample
    x = rnorm(n) # control
    y = rnorm(n) # treatment

    test = t.test(x, y)
    pval = test$p.val

    # Rerun Test if not Significant
    if(pval > alpha){
        # Sample
        x = rnorm(n) # control
        y = rnorm(n) # treatment

        # Test
        test = t.test(x, y)
        pval = test$p.val
```

```
    }
    return(pval)
}


N = 10000
sim = replicate(N, phacking_rerun(n, alpha))
hist(sim, freq=F, breaks=10,
     main='P-Hacking (Experimental Rerun)',
     xlab='P-Value', ylab='')
sum(sim < alpha) / N



filename = 'phacking_exp_rerun.png'
filepath = paste('/home/jmei/Documents/Notes/Courses/STAT675_Computing/Project/Manusc
png(filepath)
# plot
hist(sim, freq=F, breaks=20,
     main='P-Hacking (Experimental Rerun)',
     xlab='P-Value', ylab='')
dev.off()


'''
Conclusions:
- doubles chances of finding significant finding
'''


'''
Experimental Rerun 2:
```

```
- you get a smallish p-value (<0.10)
- you want to collect just a little bit more data (10%)
'''

phacking_fishing <- function(n, alpha=0.05){

    # Sample
    x = rnorm(n) # control
    y = rnorm(n) # treatment

    test = t.test(x, y)
    pval = test$p.val

    # Rerun Test if not Significant
    smallish_pval = 0.10
    if((pval > alpha) & (pval <= smallish_pval)){

        # Sample
        a_little_more = 0.1
        x = c(x, rnorm(a_little_more * n)) # control
        y = c(y, rnorm(a_little_more * n)) # treatment

        # Test
        test = t.test(x, y)
        pval = test$p.val
    }
    return(pval)
}

N = 10000
n = 10
alpha = 0.05
```

```
sim = replicate(N, phacking_fishing(n, alpha))
hist(sim, freq=F, breaks=20)
sum(sim < alpha) / N


filename = 'phacking_exp_fishing.png'
filepath = paste('/home/jmei/Documents/Notes/Courses/STAT675_Computing/Project/Manusc
#png(filepath)
# plot
hist(sim, freq=F, breaks=20,
     main='P-Hacking (Fishing)',
     xlab='P-Value', ylab='')
#dev.off()

# ------------------------------------------------------------------------

# Optional Stopping:
data_peeking <- function(n, step=1){

    # Generate Random Data
    X = rnorm(n)

    # Evaluate P-Values
    if(step == 1){
        steps = seq(step, n, by=step)
        pval = sapply(steps[-1], function(k) t.test(X[1:k])$p.val)
    }else{
        steps = seq(step, n, by=step)
        pval = sapply(steps, function(k) t.test(X[1:k])$p.val)
    }
    sig_idx = which(pval < 0.05)
```

```
    # Return
    #return(length(sig_idx) > 0)
    return(pval)
}


# Set Parameters
N = 1000
n = 30
n_seq = c(10, 30, 50, 100, 300)
step_seq = c(1, 5, 10)

#
df = data.frame(matrix(NA, ncol=length(step_seq), nrow=length(n_seq)) )
colnames(df) = step_seq
rownames(df) = n_seq

# Run Simulation
for(i in 1:length(n_seq)){
    for(j in 1:length(step_seq)){

        # Run Simulation
        sim_obj = replicate(N, data_peeking(n_seq[i], step_seq[j]))
        fp_rate = sum(sim_obj) / N

        # Save in Table
        df[i, j] = fp_rate
    }
}

# ====================================
# Simulate Multiple Dependent Variables (HARPing)
```

```
# ====================================
selectively_report_dependent_var <- function(X, Y){

    # Set Variables
    n = nrow(Y)
    p = ncol(Y)

    # Save P-Values
    pvals = rep(NA, p)
    for(i in 1:p){
        pvals[i] = t.test(X, Y[,i])$p.val
    }
    return(pvals)
}

sim_pfishing <- function(n, S){

    # Generate Data
    p = nrow(S)
    X = rnorm(n)
    Y = mvrnorm(n, mu=rep(0, p), Sigma=S)

    # T-Test
    pvals = selectively_report_dependent_var(X, Y)
    return(sum(pvals < 0.05))
}

n_seq = c(10, 30, 50, 100, 300)

# Correlation
library('MASS')
```

```
# Number of Dependent Variables
N = 10000
p = 10
n_seq = c(10, 30, 50, 100, 300)
r_seq = c(0, 0.1, 0.3, 0.5, 0.7, 0.9)
fp_rate = matrix(NA, nrow=length(n_seq), ncol=length(r_seq))
sd_df = matrix(NA, nrow=length(n_seq), ncol=length(r_seq))
for(i in 1:length(n_seq)){
    for(j in 1:length(r_seq)){

        # Set Variables
        n = n_seq[i]
        r = r_seq[j]

        # Define Correlation
        S = matrix(r, nrow=p, ncol=p)
        diag(S) = 1

        # Run Simulation
        sim_obj = replicate(N, sim_pfishing(n, S))
        sd_df[i, j] = sd((sim_obj > 0) / N)
        fp_rate[i, j] = sum(sim_obj > 0) / N
    }
}


foo = data.frame(fp_rate)
colnames(foo) = r_seq
rownames(foo) = n_seq
```