# Why and How to Adjust P-values in Multiple Hypothesis Testing

P-values below a certain threshold are often used as a method to select relevant features. Advice below suggests how to use them correctly.

Igor Šegota · Follow

Published in Towards Data Science · 9 min read · May 5



Photo by the author. Taken at Westfield UTC Mall, La Jolla, California.

Multiple hypothesis testing occurs when we repeatedly test models on a number of features, as the probability of obtaining one or more false discoveries increases with the number of tests. For example, in the field of genomics, scientists often want to test whether any of the thousands of genes have a significantly different activity in an outcome of interest. Or whether jellybeans cause acne.

In this blog post, we will cover few of the popular methods used to account for multiple hypothesis testing by adjusting model p-values:

1. False Positive Rate (FPR)
2. Family-Wise Error Rate (FWER)
3. False Discovery Rate (FDR)

and explain when it makes sense to use them.

This document can be summarized in the following image:

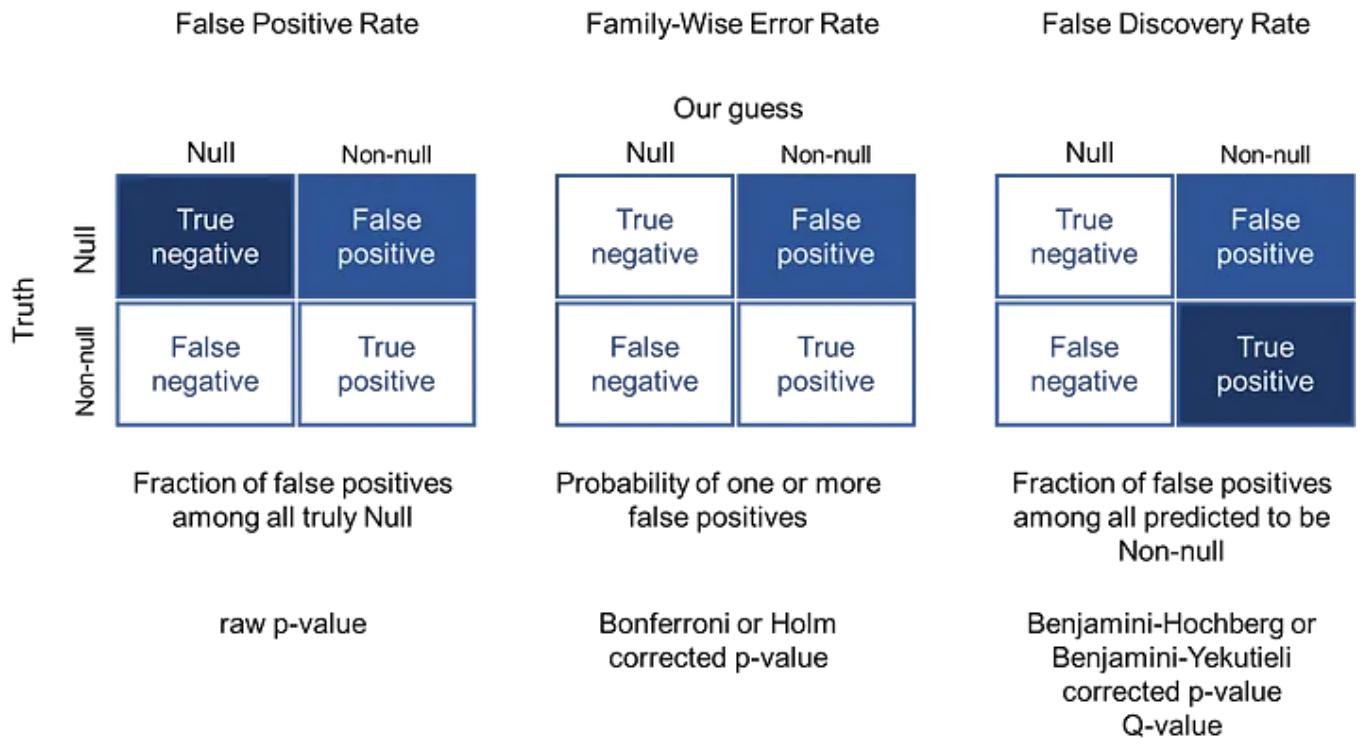



Image by the author.

## Create test data

We will create a simulated example to better understand how various manipulation of p-values can lead to different conclusions. To run this code, we need Python with `pandas`, `numpy`, `scipy` and `statsmodels` libraries installed.

For the purpose of this example, we start by creating a Pandas DataFrame of 1000 features. 990 of which (99%) will have their values generated from a Normal distribution with mean = 0, called a Null model. (In a function `norm.rvs()` used below, mean is set using a `loc` argument.) The remaining 1% of the features will be generated from a Normal distribution mean = 3, called a Non-Null model. We will use these as representing interesting features that we would like to discover.

```
import pandas as pd
import numpy as np
from scipy.stats import norm
from statsmodels.stats.multitest import multipletests

np.random.seed(42)

n_null = 9900
n_nonnull = 100

df = pd.DataFrame({
    'hypothesis': np.concatenate((
        ['null'] * n_null,
        ['non-null'] * n_nonnull,
    )),
    'feature': range(n_null + n_nonnull),
    'x': np.concatenate((
        norm.rvs(loc=0, scale=1, size=n_null),
        norm.rvs(loc=3, scale=1, size=n_nonnull),
    ))
})
```

For each of the 1000 features, p-value is a probability of observing the value at least as large, if we assume it was generated from a Null distribution.

P-values can be calculated from a cumulative distribution ( `norm.cdf()` from `scipy.stats` ) which represents the probability of obtaining a value equal to or **less than** the one observed. Then to calculate the p-value we calculate `1 - norm.cdf()` to find the probability **greater than** the one observed:

```
df['p_value'] = 1 - norm.cdf(df['x'], loc = 0, scale = 1)
df
```

| | hypothesis | feature | x | p_value |
|---|---|---|---|---|
| 0 | null | 0 | 0.496714 | 0.309695 |
| 1 | null | 1 | -0.138264 | 0.554984 |
| 2 | null | 2 | 0.647689 | 0.258593 |
| 3 | null | 3 | 1.523030 | 0.063876 |
| 4 | null | 4 | -0.234153 | 0.592567 |
| ... | ... | ... | ... | ... |
| 9995 | non-null | 9995 | 4.301102 | 0.000008 |
| 9996 | non-null | 9996 | 1.001655 | 0.158255 |
| 9997 | non-null | 9997 | 2.294683 | 0.010876 |
| 9998 | non-null | 9998 | 3.495766 | 0.000236 |
| 9999 | non-null | 9999 | 3.644388 | 0.000134 |

10000 rows × 4 columns

## False Positive Rate

The first concept is called a False Positive Rate and is defined as a fraction of null hypotheses that we flag as “significant” (also called Type I errors). The p-values we calculated earlier can be interpreted as a false positive rate by their very definition: they are probabilities of obtaining a value at least as large as a specified value, when we sample a Null distribution.

For illustrative purposes, we will apply a common (magical 🧙) p-value threshold of 0.05, but any threshold can be used:

```
df['is_raw_p_value_significant'] = df['p_value'] <= 0.05
df.groupby(['hypothesis', 'is_raw_p_value_significant']).size()
```

```
hypothesis  is_raw_p_value_significant
non-null    False                          8
            True                          92
null        False                       9407
```

```
              True                          493
dtype: int64
```

notice that out of our 9900 null hypotheses, 493 are flagged as "significant". Therefore, a False Positive Rate is: FPR = 493 / (493 + 9407) = 0.053.

The main problem with FPR is that in a real scenario we do not a priori know which hypotheses are null and which are not. Then, the raw p-value on its own (False Positive Rate) is of limited use. In our case when the fraction of non-null features is very small, most of the features flagged as significant will be null, because there are many more of them. Specifically, out of 92 + 493 = 585 features flagged true ("positive"), only 92 are from our non-null distribution. That means that a majority or about 84% of reported significant features (493 / 585) are false positives!

So, what can we do about this? There are two common methods of addressing this issue: instead of False Positive Rate, we can calculate Family-Wise Error Rate (FWER) or a False Discovery Rate (FDR). Each of these methods takes the set of raw, unadjusted, p-values as an input, and produces a new set of "adjusted p-values" as an output. These "adjusted p-values" represent estimates of *upper bounds* on FWER and FDR. They can be obtained from `multipletests()` function, which is part of the `statsmodels` Python library:

```
def adjust_pvalues(p_values, method):
   return multipletests(p_values, method = method)[1]
```

## Family-Wise Error Rate

Family-Wise Error Rate is a probability of falsely rejecting one or more null hypotheses, or in other words flagging true Null as Non-null, or a probability of seeing one or more false positives.

When there is only one hypothesis being tested, this is equal to the raw p-value (false positive rate). However, the more hypotheses are tested, the more likely we are going to get one or more false positives. There are two popular ways to estimate FWER: Bonferroni and Holm procedures. Although neither Bonferroni nor Holm procedures make any assumptions about the dependence of tests run on individual features, they will be overly conservative. For example, in the extreme case when all of the features are identical (same model repeated 10,000 times), no correction is needed. While in the other extreme, where no features are correlated, some type of correction is required.

### Bonferroni procedure

One of the most popular methods for correcting for multiple hypothesis testing is a Bonferroni procedure. The reason this method is popular is because it is very easy to calculate, even by hand. This procedure multiplies each p-value by the total number of tests performed or sets it to 1 if this multiplication would push it past 1.

```
df['p_value_bonf'] = adjust_pvalues(df['p_value'], 'bonferroni')
df.sort_values('p_value_bonf')
```

| | hypothesis | feature | x | p_value | is_raw_p_value_significant | p_value_bonf |
|---|---|---|---|---|---|---|
| 9907 | non-null | 9907 | 5.322609 | 5.114466e-08 | True | 0.000511 |
| 9942 | non-null | 9942 | 5.022174 | 2.554492e-07 | True | 0.002554 |
| 9943 | non-null | 9943 | 4.831177 | 6.786409e-07 | True | 0.006786 |
| 9941 | non-null | 9941 | 4.801528 | 7.872958e-07 | True | 0.007873 |
| 9976 | non-null | 9976 | 4.674271 | 1.475000e-06 | True | 0.014750 |
| ... | ... | ... | ... | ... | ... | ... |
| 3336 | null | 3336 | -1.365824 | 9.140029e-01 | False | 1.000000 |
| 3337 | null | 3337 | -0.148969 | 5.592111e-01 | False | 1.000000 |
| 3338 | null | 3338 | 0.502784 | 3.075579e-01 | False | 1.000000 |
| 3331 | null | 3331 | -1.545730 | 9.389151e-01 | False | 1.000000 |
| 9999 | non-null | 9999 | 3.644388 | 1.340142e-04 | True | 1.000000 |

10000 rows × 6 columns

## Holm procedure

Holm's procedure provides a correction that is more powerful than Bonferroni's procedure. The only difference is that the p-values are not all multiplied by the total number of tests (here, 10000). Instead, each sorted p-value is multiplied progressively by a decreasing sequence 10000, 9999, 9998, 9997, ..., 3, 2, 1.

```
df['p_value_holm'] = adjust_pvalues(df['p_value'], 'holm')
df.sort_values('p_value_holm').head(10)
```

| | hypothesis | feature | x | p_value | is_raw_p_value_significant | p_value_bonf | p_value_holm |
|---|---|---|---|---|---|---|---|
| 9907 | non-null | 9907 | 5.322609 | 5.114466e-08 | True | 0.000511 | 0.000511 |
| 9942 | non-null | 9942 | 5.022174 | 2.554492e-07 | True | 0.002554 | 0.002554 |
| 9943 | non-null | 9943 | 4.831177 | 6.786409e-07 | True | 0.006786 | 0.006785 |
| 9941 | non-null | 9941 | 4.801528 | 7.872958e-07 | True | 0.007873 | 0.007871 |
| 9976 | non-null | 9976 | 4.674271 | 1.475000e-06 | True | 0.014750 | 0.014744 |
| 9964 | non-null | 9964 | 4.589147 | 2.225301e-06 | True | 0.022253 | 0.022242 |
| 9974 | non-null | 9974 | 4.515318 | 3.161090e-06 | True | 0.031611 | 0.031592 |
| 9990 | non-null | 9990 | 4.433625 | 4.633087e-06 | True | 0.046331 | 0.046298 |
| 9909 | non-null | 9909 | 4.414029 | 5.073215e-06 | True | 0.050732 | 0.050692 |
| 9916 | non-null | 9916 | 4.316007 | 7.943832e-06 | True | 0.079438 | 0.079367 |

We can verify this ourselves: the last 10th p-value on this output is multiplied by 9991: 7.943832e-06 * 9991 = 0.079367. Holm's correction is also the default method for adjusting p-values in `p.adjust()` function in R language.

If we again apply our p-value threshold of 0.05, let's take a look how these adjusted p-values affect our predictions:

```
df['is_p_value_holm_significant'] = df['p_value_holm'] <= 0.05
df.groupby(['hypothesis', 'is_p_value_holm_significant']).size()
```

```
hypothesis  is_p_value_holm_significant
non-null    False                          92
            True                            8
null        False                        9900
dtype: int64
```

These results are much different than when we applied the same threshold to the raw p-values! Now, only 8 features are flagged as "significant", and all 8 are correct — they were generated from our Non-null distribution. This is

because the probability of getting even one feature flagged incorrectly is only 0.05 (5%).

However, this approach has a downside: it failed to flag other 92 Non-null features as significant. While it was very stringent to make sure none of the null features slipped in, it was able to find only 8% (8 out of 100) non-null features. This can be seen as taking a different extreme than the False Positive Rate approach.

Is there a more middle ground? The answer is "yes", and that middle ground is False Discovery Rate.

## False Discovery Rate

What if we are OK with letting some false positives in, but capturing more than single-digit percent of true positives? Maybe we are OK with having *some* false positive, just not that many that they overwhelm all of the features we flag as significant — as was the case in the FPR example.

This can be done by controlling for False Discovery Rate (rather than FWER or FPR) at a specified threshold level, say 0.05. False Discovery Rate is defined a fraction of false positives among all features flagged as positive: FDR = FP / (FP + TP), where FP is the number of False Positives and TP is the number of True Positives. By setting FDR threshold to 0.05, we are saying we are OK with having 5% (on average) false positives among all of our features we flag as positive.

There are several methods to control FDR and here we will describe how to use two popular ones: Benjamini-Hochberg and Benjamini-Yekutieli procedures. Both of these procedures are similar although more involved

than FWER procedures. They still rely on sorting the p-values, multiplying them with a specific number, and then using a cut-off criterion.

### Benjamini-Hochberg procedure

Benjamini-Hochberg (BH) procedure assumes that each of the tests are *independent*. Dependent tests occur, for example, if the features being tested are correlated with each other. Let's calculate the BH-adjusted p-values and compare it to our earlier result from FWER using Holm's correction:

```
df['p_value_bh'] = adjust_pvalues(df['p_value'], 'fdr_bh')
df[['hypothesis', 'feature', 'x', 'p_value', 'p_value_holm', 'p_value_bh']] \
    .sort_values('p_value_bh') \
    .head(10)
```

| | hypothesis | feature | x | p_value | p_value_holm | p_value_bh |
|---|---|---|---|---|---|---|
| 9907 | non-null | 9907 | 5.322609 | 5.114466e-08 | 0.000511 | 0.000511 |
| 9942 | non-null | 9942 | 5.022174 | 2.554492e-07 | 0.002554 | 0.001277 |
| 9941 | non-null | 9941 | 4.801528 | 7.872958e-07 | 0.007871 | 0.001968 |
| 9943 | non-null | 9943 | 4.831177 | 6.786409e-07 | 0.006785 | 0.001968 |
| 9976 | non-null | 9976 | 4.674271 | 1.475000e-06 | 0.014744 | 0.002950 |
| 9964 | non-null | 9964 | 4.589147 | 2.225301e-06 | 0.022242 | 0.003709 |
| 9974 | non-null | 9974 | 4.515318 | 3.161090e-06 | 0.031592 | 0.004516 |
| 9990 | non-null | 9990 | 4.433625 | 4.633087e-06 | 0.046298 | 0.005637 |
| 9909 | non-null | 9909 | 4.414029 | 5.073215e-06 | 0.050692 | 0.005637 |
| 9995 | non-null | 9995 | 4.301102 | 8.497538e-06 | 0.084890 | 0.007097 |

```
df['is_p_value_holm_significant'] = df['p_value_holm'] <= 0.05
df.groupby(['hypothesis', 'is_p_value_holm_significant']).size()
```

```
hypothesis  is_p_value_holm_significant
non-null    False                          92
            True                            8
null        False                        9900
dtype: int64
```

```
df['is_p_value_bh_significant'] = df['p_value_bh'] <= 0.05
df.groupby(['hypothesis', 'is_p_value_bh_significant']).size()
```

```
hypothesis  is_p_value_bh_significant
non-null    False                          67
            True                           33
null        False                        9898
            True                            2
dtype: int64
```

BH procedure now correctly flagged 33 out of 100 non-null features as significant — an improvement from the 8 with the Holm's correction. However, it also flagged 2 null features as significant. So, out of the 35 features flagged as significant, the fraction of incorrect features is: 2 / 33 = 0.06 so 6%.

Note that in this case we have 6% FDR rate, even though we aimed to control it at 5%. FDR will be controlled at a 5% rate *on average*: sometimes it may be lower and sometimes it may be higher.

### Benjamini-Yekutieli procedure

Benjamini-Yekutieli (BY) procedure controls FDR regardless of whether tests are independent or not. Again, it is worth noting that all of these procedures try to establish *upper bounds* on FDR (or FWER), so they may be less or more

conservative. Let's compare the BY procedure with a BH and Holm procedures above:

```
df['p_value_by'] = adjust_pvalues(df['p_value'], 'fdr_by')
df[['hypothesis', 'feature', 'x', 'p_value', 'p_value_holm', 'p_value_bh', 'p_va
    .sort_values('p_value_by') \
    .head(10)
```

| | hypothesis | feature | x | p_value | p_value_holm | p_value_bh | p_value_by |
|---|---|---|---|---|---|---|---|
| 9907 | non-null | 9907 | 5.322609 | 5.114466e-08 | 0.000511 | 0.000511 | 0.005006 |
| 9942 | non-null | 9942 | 5.022174 | 2.554492e-07 | 0.002554 | 0.001277 | 0.012501 |
| 9943 | non-null | 9943 | 4.831177 | 6.786409e-07 | 0.006785 | 0.001968 | 0.019264 |
| 9941 | non-null | 9941 | 4.801528 | 7.872958e-07 | 0.007871 | 0.001968 | 0.019264 |
| 9976 | non-null | 9976 | 4.674271 | 1.475000e-06 | 0.014744 | 0.002950 | 0.028873 |
| 9964 | non-null | 9964 | 4.589147 | 2.225301e-06 | 0.022242 | 0.003709 | 0.036301 |
| 9974 | non-null | 9974 | 4.515318 | 3.161090e-06 | 0.031592 | 0.004516 | 0.044199 |
| 9990 | non-null | 9990 | 4.433625 | 4.633087e-06 | 0.046298 | 0.005637 | 0.055172 |
| 9909 | non-null | 9909 | 4.414029 | 5.073215e-06 | 0.050692 | 0.005637 | 0.055172 |
| 9916 | non-null | 9916 | 4.316007 | 7.943832e-06 | 0.079367 | 0.007097 | 0.069458 |

```
df['is_p_value_by_significant'] = df['p_value_by'] <= 0.05
df.groupby(['hypothesis', 'is_p_value_by_significant']).size()
```

```
hypothesis  is_p_value_by_significant
non-null    False                        93
            True                          7
null        False                      9900
dtype: int64
```

BY procedure is stricter in controlling FDR; in this case even more so than the Holm's procedure for controlling FWER, by flagging only 7 non-null features as significant! The main advantage of using it is when we know the data may contain a high number of correlated features. However, in that case we may also want to consider filtering out correlated features so that we do not need to test all of them.

## Summary

At the end, the choice of procedure is left to the user and depends on what the analysis is trying to do. Quoting Benjamini, Hochberg (Royal Stat. Soc. 1995):

> *Often the control of the FWER is not quite needed. The control of the FWER is important when a conclusion from the various individual inferences is likely to be erroneous when at least one of them is.*
>
> *This may be the case, for example, when several new treatments are competing against a standard, and a single treatment is chosen from the set of treatments which are declared significantly better than the standard.*

In other cases, where we may be OK to have some false positives, FDR methods such as BH correction provide less stringent p-value adjustments and may be preferrable if we primarily want to increase the number of true positives that pass a certain p-value threshold.

There are other adjustment methods not mentioned here, notably a q-value which is also used for FDR control, and at the time of writing exists only as an R package.

Statistics | Hypothesis Testing | False Positive | P Value | Data Science

## More from the list: "NLP"

Curated by Himanshu Birla

 Jon Gi... in Towards Data ...

**Characteristics of Word Embeddings**

✦ · 11 min read · Sep 4, 2021

 Jon Gi... in Towards Data ...

**The Word2vec Hyperparameters**

✦ · 6 min read · Sep 3, 2021

 Jon Gi... in

**The Word2ve**

✦ · 15 min rea

View list

# Written by Igor Šegota

Follow

72 Followers · Writer for Towards Data Science

Bioinformatics Data Scientist, PhD in Physics, R ❤ Python

## More from Igor Šegota and Towards Data Science

Igor Šegota in Towards Data Science

## Unbox the Cox: Intuitive Guide to Cox Regressions

How do hazards and maximum likelihood estimates predict event rankings?

10 min read · Jun 6

82 1

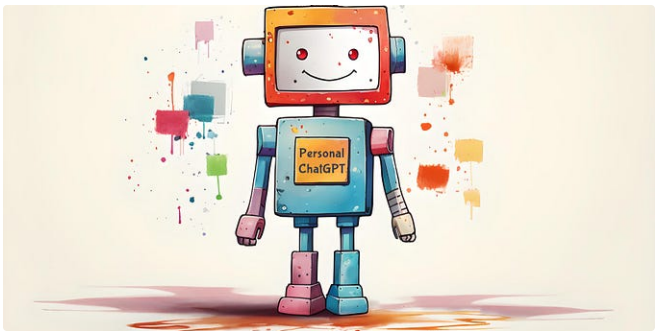

Antonis Makropoulos in Towards Data Science

## How to Build a Multi-GPU System for Deep Learning in 2023

This story provides a guide on how to build a multi-GPU system for deep learning and...

10 min read · Sep 17

549 11




Robert A. Gonsalves in Towards Data Science

## Your Own Personal ChatGPT

How you can fine-tune OpenAI's GPT-3.5 Turbo model to perform new tasks using you...

· 15 min read · Sep 8

595 7

Igor Šegota in Towards Data Science

## Unbox the Cox: A Hidden Dark Secret of Cox Regression

Why perfect predictors result in a p-value of 0.93 ?

8 min read · Jun 27

70

See all from Igor Šegota | See all from Towards Data Science

## Recommended from Medium

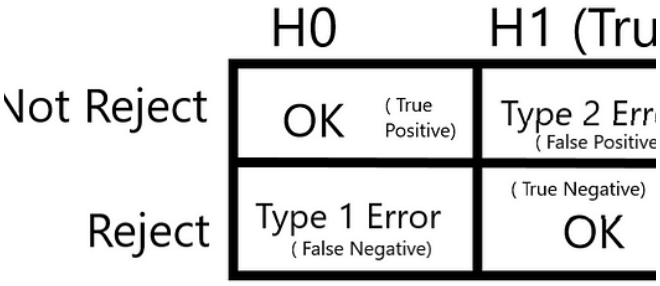

Data Analysis

### How I Chose Between Factor Analysis and Principal Componen...

Key Takeaways

7 min read · Sep 26

23




Kamna Sinha in Data At The Core !

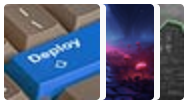
### Hypothesis Test And All About P Value,T test,Chi Square Test, Ano...

Hypothesis Test

4 min read · 6 days ago

22

## Lists



**Predictive Modeling w/ Python**

20 stories · 452 saves

**New_Reading_List**

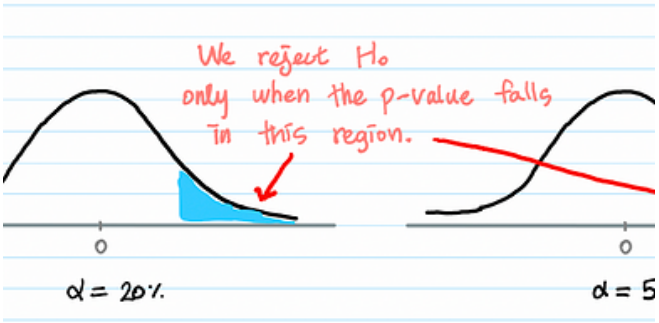
174 stories · 133 saves

**Practical Guides to Machine Learning**

10 stories · 519 saves

**Coding & Development**

11 stories · 200 saves



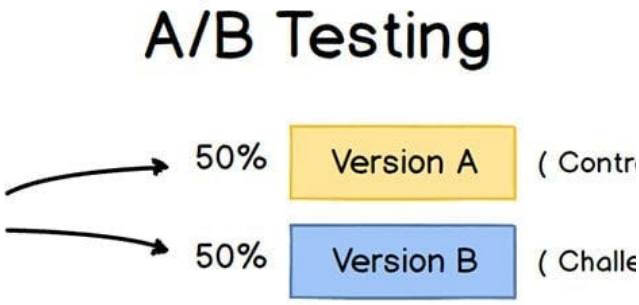

Ms Aerin in IntuitionMath

## Chi Square Test—Intuition, Examples, and Step-by-Step...

The best way to see if two variables are related.

· 15 min read · Feb 13

400 3




SharkYun

## A/B Testing— 6 Practical Steps of Implementation with Full...

What is A/B testing

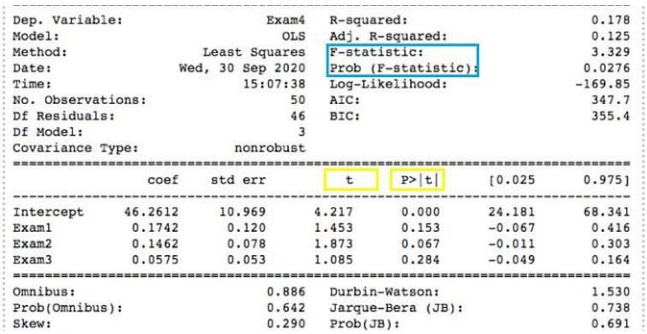
· 9 min read · May 11

9

Biman Chakraborty

## Two-Sample t Tests, Power, Effect Size and Sample Size Calculator i...

I was going over my daily dose of coffee while redaing the newspaper in the morning. A...



| | | | |
|---|---|---|---|
| Dep. Variable: | Exam4 | R-squared: | 0.178 |
| Model: | OLS | Adj. R-squared: | 0.125 |
| Method: | Least Squares | F-statistic: | 3.329 |
| Date: | Wed, 30 Sep 2020 | Prob (F-statistic): | 0.0276 |
| Time: | 15:07:38 | Log-Likelihood: | -169.85 |
| No. Observations: | 50 | AIC: | 347.7 |
| Df Residuals: | 46 | BIC: | 355.4 |
| Df Model: | 3 | | |
| Covariance Type: | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| Intercept | 46.2612 | 10.969 | 4.217 | 0.000 | 24.181 | 68.341 |
| Exam1 | 0.1742 | 0.120 | 1.453 | 0.153 | -0.067 | 0.416 |
| Exam2 | 0.1462 | 0.078 | 1.873 | 0.067 | -0.011 | 0.303 |
| Exam3 | 0.0575 | 0.053 | 1.085 | 0.284 | -0.049 | 0.164 |

| | | | |
|---|---|---|---|
| Omnibus: | 0.886 | Durbin-Watson: | 1.530 |
| Prob(Omnibus): | 0.642 | Jarque-Bera (JB): | 0.738 |
| Skew: | 0.290 | Prob(JB): | 0.691 |

Albane Colmenares

## The Statistical Foundation of Linear Regression: T-Tests, ANOVA, and...

In Kaggle's 2020 State of Data Science and Machine Learning Survey, it was reported th...

13 min read · Apr 30

5

6 min read · Sep 5

See more recommendations