Hafta 5 (Measurement Problems & Hypothesis Tests)

@mebaysan

24/09/2021

Bu haftanın veri seti

- ab_testing.xlsx
- amazon_review.csv
- course_review.csv
- imdb_ratings.csv
- product_sorting.csv

İlgili Okuma Listesi

-

Benim Yazdığım Yazılar

-

Measurement Problems (Ölçüm Problemleri)

Bir Ürünü Satın Aldıran Nedir?

Satın alma kararlarını etkileyen bazı faktörler vardır. Bu faktörlerin son yıllarda en önemli sayılabilecek olanı **Social Proof**'tur. Topluluğun kanaati diyebiliriz. Bu oldukça önemlidir. Peki bu neden önemlidir? **The Wisdom of Crowds** yani kalabalıkların bilgeliği diyebiliriz. "Onlar yaptıysa bir bildiği vardır" vb.

Product Rating

- Average
  - Direkt ortalamaya göre

```
df["Rating"].mean()
```

- Time-Based Weighted Average
  - (Analiz tarihi) - (rating tarihi)
  - Zaman projeksiyonunda (yıl||ay||gün) verdiğimiz ağırlıklara göre sonucu hesaplarız

- ÖR: Eğer zaman farkı 30 günden az ise %30 etkili olsun gibi

```
df['Timestamp'] = pd.to_datetime(df['Timestamp'])

current_date = pd.to_datetime('2021-02-10 0:0:0')

df["days"] = (current_date - df['Timestamp']).dt.days

df.loc[df["days"] <= 30, "Rating"].mean() * 28 / 100 + \
df.loc[(df["days"] > 30) & (df["days"] <= 90), "Rating"].mean() * 26 / 100 + \
df.loc[(df["days"] > 90) & (df["days"] <= 180), "Rating"].mean() * 24 / 100 + \
df.loc[(df["days"] > 180), "Rating"].mean() * 22 / 100
```

- User-Based Weighted Average
  - Bu tekniğin en önemli sorusu şudur: Her kullanıcının verdiği puan aynı mıdır?
  - User Quality Score olarak da bilinmektedir
  - ÖR: Kullanıcının bir online kursa katılımı %10 veya daha az ise bunun sıralamaya etkisi %20 olsun

```
df.loc[df["Progress"] <= 10, "Rating"].mean() * 22 / 100 + \
df.loc[(df["Progress"] > 10) & (df["Progress"] <= 45), "Rating"].mean() * 24 / 100 + \
df.loc[(df["Progress"] > 45) & (df["Progress"] <= 75), "Rating"].mean() * 26 / 100 + \
df.loc[(df["Progress"] > 75), "Rating"].mean() * 28 / 100
```

- Weighted Rating
  - User Quality Score ve Zaman Ağırlıklı skorları hesaplar ikisine de ağırlık verir (%X kadar alıp) bunları toplarsak ağırlıklı ratingi hesaplamış oluruz

```
def course_weighted_rating(dataframe, time_w=50, user_w=50):
    return time_based_weighted_average(dataframe) * time_w / 100 + user_based_weighted_average(dataframe) * user_w / 100
```

Product Sorting

- Sorting by Rating
  - Sadece rating'e göre sıralarız.
  - Sadece rating parametresine göre sıralayamıyor olmamız gerekir. Bu durumda ratingi yüksek olan fakat yorum sayısı az olanlar yukarıda gözükecek. Bu sebeple social proof etkisiz kalacaktır

```
df.sort_values("rating", ascending=False).head(20)
```

- Sorting by Comment Count or Purchase Count

```
df.sort_values("purchase_count", ascending=False).head(20)

df.sort_values("commment_count", ascending=False).head(20)
```

- Sorting by Rating, Comment and Purchase

- En önemli social proof metriklerinden biri yorum sayısıdır.

```
# rating 1-5 arasında olduğu için satın alma sayılarını ve yorum sayılarını 1-5 arasında standartlaştırıyoruz
df["purchase_count_scaled"] = MinMaxScaler(feature_range=(1, 5)). \
    fit(df[["purchase_count"]]). \
    transform(df[["purchase_count"]])

df["commment_count_scaled"] = MinMaxScaler(feature_range=(1, 5)). \
    fit(df[["commment_count"]]). \
    transform(df[["commment_count"]])


(df["commment_count_scaled"] * 32 / 100 + # %32 olacak şekilde ağırlıklandırdık
 df["purchase_count_scaled"] * 26 / 100 +
 df["rating"] * 42 / 100)
```

- Sorting by Bayesian Average Rating Score (Sorting Products with 5 Star Rated)
  - İstatistiki olarak olasılıksal skorlama ve sıralama

```
def bayesian_average_rating(n, confidence=0.95):
    """
    Olasılıksal

    Parameters
    ----------
    n: list or df
        puanların frekanslarını tutar.
        Örnek: [2, 40, 56, 12, 90] 2 tane 1 puan, 40 tane 2 puan, ... , 90 tane 5 puan.
    confidence: float
        güven aralığı

    Returns
    -------
    BAR score: float


    """

    # rating'lerin toplamı sıfır ise sıfır dön.
    if sum(n) == 0:
        return 0
    # eşsiz yıldız sayısı. 5 yıldızdan da puan varsa 5 olacaktır.
    K = len(n)
    # 0.95'e göre z skoru.
    z = st.norm.ppf(1 - (1 - confidence) / 2)
    # toplam rating sayısı.
    N = sum(n)
    first_part = 0.0
    second_part = 0.0
    # index bilgisi ile birlikte yıldız sayılarını gez.
    # formülasyondaki hesapları gerçekleştir.
    for k, n_k in enumerate(n):
        first_part += (k + 1) * (n[k] + 1) / (N + K)
        second_part += (k + 1) * (k + 1) * (n[k] + 1) / (N + K)
    score = first_part - z * math.sqrt((second_part - first_part * first_part) / (N + K + 1))
    return score



df["bar_sorting_score"] = df.apply(lambda x: bayesian_average_rating(x[["1_point",
                                                                        "2_point",
                                                                        "3_point",
                                                                        "4_point",
                                                                        "5_point"]]), axis=1)
```

- Hybrid Sorting: BAR Score + Diğer Faktörler
  - İstersek hem ağırlıklı skorlama hem de bayesian skorlamayı hesaplar ve bunları ağırlıklandırır kullanırız

```
def hybrid_sorting_score(dataframe, bar_w=60, wss_w=40):
    bar_score = dataframe.apply(lambda x: bayesian_average_rating(x[["1_point",
                                                                     "2_point",
                                                                     "3_point",
                                                                     "4_point",
                                                                     "5_point"]]), axis=1)
    wss_score = weighted_sorting_score(dataframe)

    return bar_score*bar_w/100 + wss_score*wss_w/100


df["hybrid_sorting_score"] = hybrid_sorting_score(df)
```

- IMDB Sorting Tekniği
  - IMDB sıralama için 2015'e kadar bu yöntemi uyguluyordu
    - weighted_rating = (v/(v+M) * r) + (M/(v+M) * C)
    - r = vote average
    - v = vote count
    - M = minimum votes required to be listed in the Top 250
    - C = the mean vote across the whole report (currently 7.0)

```
def weighted_rating(
    r,  # filmin oy ortalaması
    v,  # filmin oy sayısı
    M,  # gereken minimum oy sayısı
    C  # genel kitlenin ortalaması
):
    return (v / (v + M) * r) + (M / (v + M) * C)


df["weighted_rating"] = weighted_rating(df["vote_average"], df["vote_count"], M, C)
```

Review Sorting

Yorum sıralamak için bazı teknikler vardır:

- Score Up-Down Diff
  - Bu yöntem çok kullanılmamalıdır
  - (Kaç adet Up rate var) - (Kaç adet Down rate var)
  - (up ratings) − (down ratings)
  - Bu yöntem aşağıdaki 2 yorum sıralanmasında sorun teşkil edecektir
    - Review 1: 600 up 400 down total 1000
    - Review 2: 5500 up 4500 down total 10000

```
def score_up_down_diff(up, down):
    return up - down

score_up_down_diff(600, 400)
```

- Average rating
  - Bu yöntem çok kullanılmamalıdır
  - (Kaç adet Up rate var) / (Toplam rate sayısı)
  - (up ratings) / (total ratings)
  - Bu yöntem aşağıdaki 2 yorum sıralanmasında sorun teşkil edecektir
    - Review 1: 2 up 0 down total 2
    - Review 2: 100 up 1 down total 101

```
def score_average_rating(up, down):
    if up + down == 0:
        return 0
    return up / (up + down)

score_average_rating(600, 400)
```

- Wilson Lower Bound Score
  - Bu yöntemi kullanmak için elimizde binary bir case olmalıdır:
    - Up - Down
    - Like - Dislike
    - Yes - No
  - 2 parametresi vardır
    - p = ilgilenen olay / tüm olay
    - p için güven aralığı

```
def wilson_lower_bound(up, down, confidence=0.95):
    """
    Wilson Lower Bound Score hesapla

    - Bernoulli parametresi p için hesaplanacak güven aralığının alt sınırı WLB skoru olarak kabul edilir.
    - Hesaplanacak skor ürün sıralaması için kullanılır.

    - Not:
    Eğer skorlar 1-5 arasıdaysa 1-3 negatif, 4-5 pozitif olarak işaretlenir ve bernoulli'ye uygun hale getirilebilir.
    Bu beraberinde bazı problemleri de getirir. Bu sebeple bayesian average rating yapmak gerekir.

    Parameters
    ----------
    up: int
        up count
    down: int
        down count
    confidence: float
        confidence

    Returns
    -------
```

```
    wilson score: float

    """
    n = up + down
    if n == 0:
        return 0
    z = st.norm.ppf(1 - (1 - confidence) / 2)
    phat = 1.0 * up / n
    return (phat + z * z / (2 * n) - z * math.sqrt((phat * (1 - phat) + z * z / (4 * n)) / n)) / (1 + z * z / n)
```

Temel İstatistik Kavramları

> Without a grounding in Statistics, a Data Scientist is a Data Lab Assistant. - Martyn Jones

Olasılık ve istatistikte temel amacımız; belirsizlik altında karar vermek üzere belirsizliği azaltmaya çalışmaktır.

Betimsel İstatistikler

- Ortalama
- Medyan
- Mod
- Kartiller
- Değişim Aralığı
- Standart Sapma
- Korelasyon

Ortalama aykırı değerlerden etkilenir. Bu sebeple aykırı değerleri olan verilerin ortalaması için medyan kullanmak daha iyidir.

Standart sapma da aslında bir ortalamadır. Merkezden olan sapmanın bir ölçüsüdür. Sapmaların ortalamasıdır.

Confidence Intervals (Güven Aralıkları)

Anakütle parametresinin tahmini değerini (istatistik) kapsayabilecek iki sayıdan oluşan bir aralık bulunmasıdır.

CI For	Sample Statistic	Margin of Error	Use When
Population mean (μ)	$\bar{x}$	$\pm z^* \frac{\sigma}{\sqrt{n}}$	X is normal, or $n \geq 30$; σ known
Population mean (μ)	$\bar{x}$	$\pm t^*_{n-1} \frac{s}{\sqrt{n}}$	$n < 30$, and/or σ unknown
Population proportion (p)	$\hat{p}$	$\pm z^* \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$	$n\hat{p}$, $n(1-\hat{p}) \geq 10$
Difference of two population means ($\mu_1 - \mu_2$)	$\bar{x}_1 - \bar{x}_2$	$\pm z^* \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$	Both normal distributions or $n_1, n_2 \geq 30$; σ_1, σ_2 known
Difference of two population means $\mu_1 - \mu_2$	$\bar{x}_1 - \bar{x}_2$	$\pm t^*_{n_1+n_2-2} \sqrt{\frac{(n_1-1)s_1^2 + (n_2-1)s_2^2}{n_1+n_2-2}}$	$n_1, n_2 < 30$; and/or $\sigma_1 = \sigma_2$ unknown
Difference of two proportions ($p_1 - p_2$)	$\hat{p}_1 - \hat{p}_2$	$\pm z^* \sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}}$	$n\hat{p}$, $n(1-\hat{p}) \geq 10$ for each group

Örnek

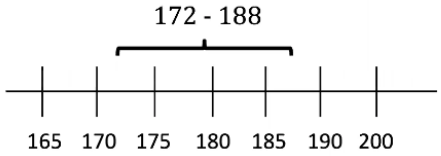
Web sitesinde geçirlen ortalama süre 180 ve standar sapma 40. Fakat; "bu 180'in etrafında hangi aralıkta en çok vakit geçirme süresi var" sorusunun cevabını güven aralıklarıyla veririz.

"web sitesinde geçirilen süre %5 hata payıyla (ki bu da %95 güvenilirlik) 172 ile 188 arasındadır" dediğimiz zaman 100 kullanıcıdan 95'i %5 hata payı ile 172 ile 188 saniye arasında sitede vakit geçirir diyebiliriz.

Web sitesinde geçirilen ortalama sürenin güven aralığı nedir?

Ortalama: 180 saniye

Standart sapma: 40 saniye



Python'da Güven Aralığı

```
import statsmodels.stats.api as sms

df = sns.load_dataset("tips")

# * %95 güvenilirlikle 'total_bill' değişkeninin güven aralığı => kitlenin çoğunluğu bu aralıkta hesap ödüyor
sms.DescrStatsW(df["total_bill"]).tconfint_mean()

>>> (18.66333170435847, 20.908553541543164)
```

Hypothesis Testing (Hipotez Testleri)

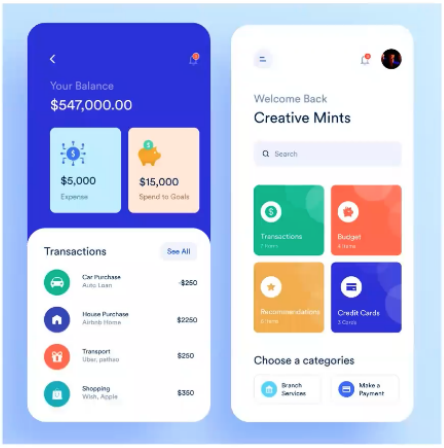
Bir inanışı, bir savı test etmek için kullanılan istatistiksel yöntemlerdir.

Grup karşılaştırmalarında temel amaç olası farklılıkların şans eser ortaya çıkıp çıkmadığını göstermeye çalışmaktır.

Örnek: Aşağıdaki soruyu cevaplamak için hipotez testleri yapmamız gerekir. Belki ortalama şans eseri artmış olabilir.

Mobil uygulamada yapılan arayüz değişikliği sonrasında kullanıcıların uygulamada geçirdikleri günlük ortalama süre arttı mı?

Tasarım 1: 55 dk



Tasarım 2: 58 dk

dribbble.cor

AB Testi (Bağımsız İki Örneklem T Testi)

İki grup ortalaması arasında karşılaştırma yapılmak istendiğinde kullanılır.

$H_0: \mu_1 = \mu_2$ $H_0: \mu_1 <= \mu_2$ $H_0: \mu_1 >= \mu_2$

$H_1: \mu_1 \neq \mu_2$ $H_1: \mu_1 > \mu_2$ $H_1: \mu_1 < \mu_2$

Örnek sayıları aynı, varyanslar homojen ise:

$$t = \frac{\overline{X}_1 - \overline{X}_2}{S_p\sqrt{\frac{2}{n}}}, \quad S_p = \sqrt{\frac{s^2_{X1} + s^2_{X2}}{2}}$$

Örnek sayısı farklı, varyanslar homojen ise:

$$t = \frac{\overline{X}_1 - \overline{X}_2}{S_p\sqrt{\frac{1}{n_1}+\frac{1}{n_2}}}, \qquad S_p = \sqrt{\frac{(n_1-1)s^2_{X1} + (n_2-1)s^2_{X2}}{n_1+n_2-2}}$$

Örnek sayıları farklı varyanslar homojen değil ise:

$$t = \frac{\overline{X}_1 - \overline{X}_2}{S_{\overline{\Delta}}}, \qquad S_{\overline{\Delta}} = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

Bağımsız İki Örneklem T Testi'nin Varsayımları

- Normallik
  - Elimizdeki 2 grubunda dağılımlarının normal olduğu varsayımıdır
- Varyans Homojenliği
  - 2 grubun varyanslarının birbirine benzerliğini ifade ediyor

Testi Uygulama Adımları:

1. Varsayım Kontrolü

a. Normallik Varsayımı

i. Shapiro Wilks testi, elimizdeki dağılımın normal olup olmadığını test eder

```
from scipy.stats import shapiro
 # H0: Normal dağılım varsayımı sağlanmaktadır.
 # H1: Normal dağılım varsayımı sağlanmamaktadır.

 test_stat, pvalue = shapiro(df.loc[df["smoker"] == "Yes", "total_bill"])
 print('Test Stat = %.4f, p-value = %.4f' % (test_stat, pvalue))

 # p-value < ise 0.05'ten HO RED.
 # p-value < değilse 0.05 H0 REDDEDILEMEZ.
```

b. Varyans Homojenliği

i. Levene testi iki grubun varyansının homojen olup olmadığını test eder

```
from scipy.stats import levene
 # H0: Varyanslar Homojendir
 # H1: Varyanslar Homojen Değildir

 test_stat, pvalue = levene(df.loc[df["smoker"] == "Yes", "total_bill"],
                            df.loc[df["smoker"] == "No", "total_bill"])
 print('Test Stat = %.4f, p-value = %.4f' % (test_stat, pvalue))

 # p-value < ise 0.05 'ten HO RED.
 # p-value < değilse 0.05 H0 REDDEDILEMEZ.
```

2. Hipotezin Uygulanması

a. Varsayımlar sağlanıyorsa bağımsız iki örneklem t testi

i. parametrik test

```
from scipy.stats import ttest_ind
 # H0: M1 = M2 (... iki grup ortalamaları arasında ist ol.anl.fark yoktur.)
 # H1: M1 != M2 (...vardır)

 ############################
 # Varsayımlar sağlanıyorsa bağımsız iki örneklem t testi (parametrik test)
 ############################


 test_stat, pvalue = ttest_ind(df.loc[df["smoker"] == "Yes", "total_bill"],
                               df.loc[df["smoker"] == "No", "total_bill"],
                               equal_var=True)

 print('Test Stat = %.4f, p-value = %.4f' % (test_stat, pvalue))


 # p-value < ise 0.05 'ten HO RED.
 # p-value < değilse 0.05 H0 REDDEDILEMEZ.
```

b. Varsayımlar sağlanmıyorsa mannwhitneyu testi

i. non-parametrik test

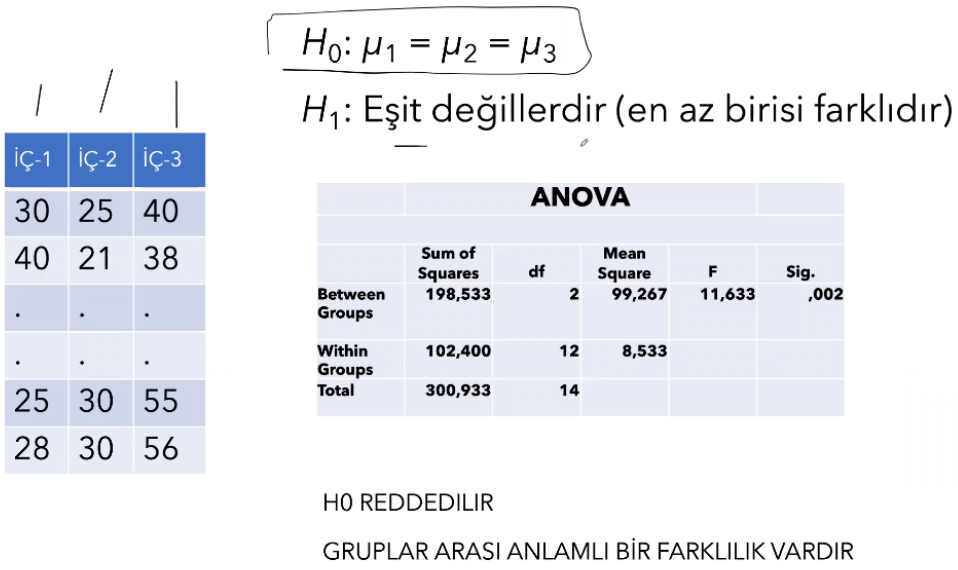
```
from scipy.stats import mannwhitneyu
 # H0: M1 = M2 (... iki grup ortalamaları arasında ist ol.anl.fark yoktur.)
 # H1: M1 != M2 (...vardır)
```

```
############################
# Varsayımlar sağlanmıyorsa mannwhitneyu testi (non-parametrik test)
############################

test_stat, pvalue = mannwhitneyu(df.loc[df["smoker"] == "Yes", "total_bill"],
                                 df.loc[df["smoker"] == "No", "total_bill"])


print('Test Stat = %.4f, p-value = %.4f' % (test_stat, pvalue))
```

NOT:

- Normallik sağlanmıyorsa direkt 2 numara (non-parametrik /mannwhitneyu). Varyans Homojenliği sağlanmıyor fakat normallik sağlanıyorsa 1 numaraya (bağımsız iki örneklem t testi) arguman girilir.
- Normallik incelemesi öncesi aykırı değer incelemesi ve düzeltmesi yapmak faydalı olabilir.

İkiden Fazla Grup Ortalaması Karşılaştırma (ANOVA - Analysis of Variance)



İÇ-1	İÇ-2	İÇ-3
30	25	40
40	21	38
.	.	.
.	.	.
25	30	55
28	30	56

$H_0: \mu_1 = \mu_2 = \mu_3$

H_1: Eşit değillerdir (en az birisi farklıdır)

ANOVA

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	198,533	2	99,267	11,633	,002
Within Groups	102,400	12	8,533		
Total	300,933	14			

H0 REDDEDILIR

GRUPLAR ARASI ANLAMLI BİR FARKLILIK VARDIR

ANOVA Uygulama Adımları:

1. Varsayım Kontrolü
    a. Normallik Varsayımı
        i. Shapiro

```
from scipy.stats import shapiro
 df = sns.load_dataset("tips")
```

```
 # H0: Normal dağılım varsayımı sağlanmaktadır.

 for group in list(df["day"].unique()):
     pvalue = shapiro(df.loc[df["day"] == group, "total_bill"])[1]
     print(group, 'p-value: %.4f' % pvalue)
```

b. Varyans Homojenliği Varsayımı

i. Levene

```
from scipy.stats import levene
 # H0: varyans homojenliği varsayımı sağlanmaktadır.
 test_stat, pvalue = levene(df.loc[df["day"] == "Sun", "total_bill"],
                            df.loc[df["day"] == "Sat", "total_bill"],
                            df.loc[df["day"] == "Thur", "total_bill"],
                            df.loc[df["day"] == "Fri", "total_bill"])
 print('Test Stat = %.4f, p-value = %.4f' % (test_stat, pvalue))
```

2. Hipotezin Uygulanması

a. Varsayım sağlanıyorsa:

i. one way anova (parametrik)

```
from scipy.stats import f_oneway
 # HO: Grup ortalamaları arasında ist ol anl fark yoktur:

 # parametrik anova testi:
 f_oneway(df.loc[df["day"] == "Thur", "total_bill"],
          df.loc[df["day"] == "Fri", "total_bill"],
          df.loc[df["day"] == "Sat", "total_bill"],
          df.loc[df["day"] == "Sun", "total_bill"])
```

b. Varsayım sağlanmıyorsa:

i. kruskal (non-parametrik)

```
from scipy.stats import kruskal
 # HO: Grup ortalamaları arasında ist ol anl fark yoktur:

 # Nonparametrik anova testi:
 kruskal(df.loc[df["day"] == "Thur", "total_bill"],
         df.loc[df["day"] == "Fri", "total_bill"],
         df.loc[df["day"] == "Sat", "total_bill"],
         df.loc[df["day"] == "Sun", "total_bill"])
```

İstersek ANOVA'yı burda kesebilir ve işimiz burda diyebiliriz. İstersek de şu soruyu sorabiliriz: Farklılık (varsa) hangisinden kaynaklı oluşmaktadır?

```
from statsmodels.stats.multicomp import MultiComparison
comparison = MultiComparison(df['total_bill'], df['day'])
tukey = comparison.tukeyhsd(0.05)
print(tukey.summary())

>>> print(tukey.summary())
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
```

```
group1 group2 meandiff p-adj   lower   upper  reject
----------------------------------------------------
   Fri    Sat   3.2898 0.4554 -2.4802  9.0598  False
   Fri    Sun   4.2584 0.2373 -1.5859 10.1028  False
   Fri   Thur   0.5312    0.9 -5.4437   6.506  False
   Sat    Sun   0.9686 0.8921 -2.6089  4.5462  False
   Sat   Thur  -2.7586 0.2375 -6.5456  1.0284  False
   Sun   Thur  -3.7273 0.0669 -7.6266  0.1721  False
----------------------------------------------------
```