F RATIO UNDER THE NULL HYPOTHESIS

Packages used in this notebook

```julia
using Pkg
```

```julia
Pkg.activate("/Users/juan/Documents/Julia/Tutorials/DataScience/Project.toml")
```

```julia
using StatsPlots
```

```julia
using Distributions
```

```julia
using Random
```

```julia
using DataFrames
```

```julia
using StatsBase
```

```julia
using Statistics
```

```julia
using WebIO
```

```julia
using PlutoUI
```

📚 Table of Contents


```julia
PlutoUI.TableOfContents(title="📚 Table of Contents", indent=true, depth=4,
aside=true)
```

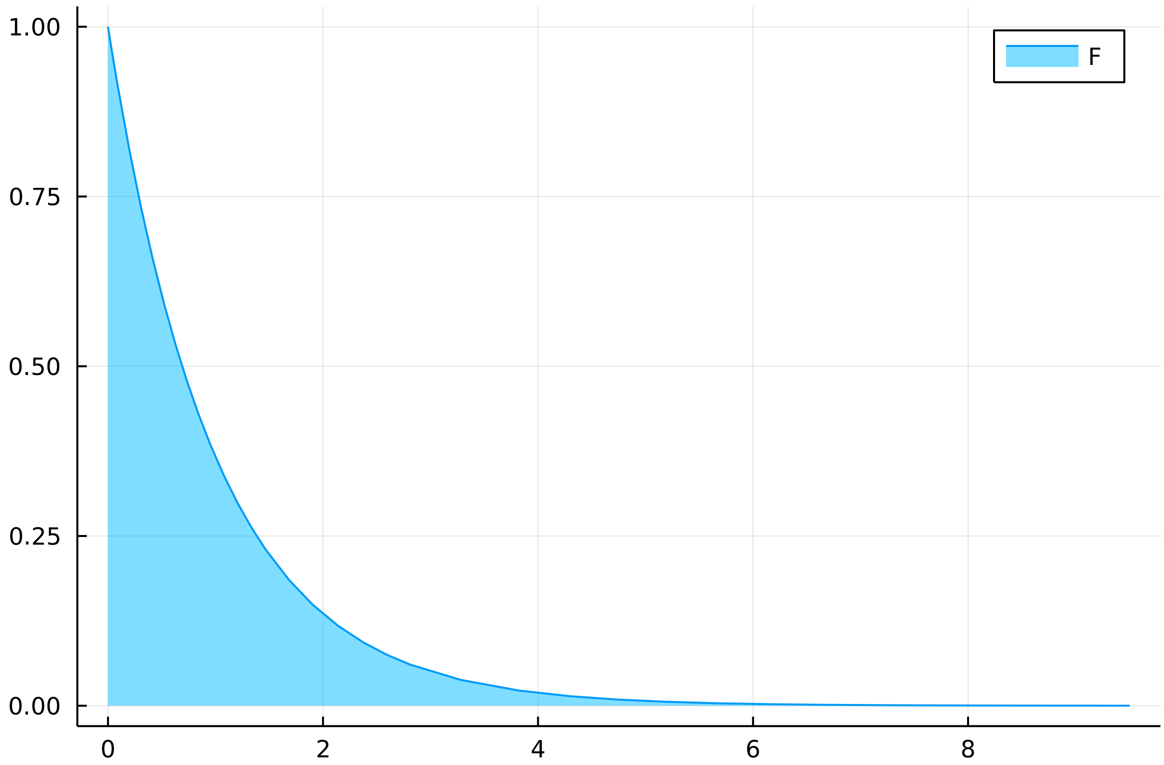
F distribution

2

```
@bind α Slider(2:5, show_value = true)
```

295

```
@bind β Slider(295:305, show_value = true)
```




```
StatsPlots.plot(Distributions.FDist(α, β), fill = (0, 0.5, :deepskyblue), label =
"F")
```

Data

Generate random values for mass for three groups, *I*, *II*, *III*. Mass values will be taken from normal distribution. The total sample size will be 300, with 100 subjects in each group.

n = 300

```
n = 300 # Sample size
```

n_group = 100

```
n_group = Int(n / 3)
```

id = 1:300

```
id = 1:n
```

```julia
group = vcat(
    repeat(["I"], n_group),
    repeat(["II"], n_group),
    repeat(["III"], n_group)
);
```

10

```julia
begin
    μ1 = 98
    μ2 = 100
    μ3 = 105
    σ1 = 20
    σ2 = 15
    σ3 = 10
end
```

[104.777, 118.715, 68.505, 151.497, 80.9689, 90.469, 97.1297, 134.322, 87.616, 114.919, 1

```julia
begin
    Random.seed!(12) # Reproducible results
    mass = vcat(
        rand(Distributions.Normal(μ1, σ1), n_group),
        rand(Distributions.Normal(μ2, σ2), n_group),
        rand(Distributions.Normal(μ3, σ3), n_group)
    )
end
```

df =

	ID	Group	Mass
1	1	"I"	104.777
2	2	"I"	118.715
3	3	"I"	68.505
4	4	"I"	151.497
5	5	"I"	80.9689
6	6	"I"	90.469
7	7	"I"	97.1297
8	8	"I"	134.322
9	9	"I"	87.616
10	10	"I"	114.919
	more		
300	300	"III"	94.6326

```julia
df = DataFrames.DataFrame(
    ID = id,
    Group = group,
    Mass = mass
)
```

It may be useful to generate sub-DataFrame objects, one for each group. To do this, we us eteh `filter` function.

	ID	Group	Mass
1	201	"III"	110.947
2	202	"III"	120.308
3	203	"III"	116.931
4	204	"III"	100.125
5	205	"III"	111.328
6	206	"III"	116.786
7	207	"III"	111.071
8	208	"III"	92.9178
9	209	"III"	109.014
10	210	"III"	105.275
more			
100	300	"III"	94.6326

```
begin
    group_I = filter(r -> r.Group == "I", df)
    group_II = filter(r -> r.Group == "II", df)
    group_III = filter(r -> r.Group == "III", df)
end
```

Although we generated the data manually, we extract the mass values from the DataFrame below as Vector objects.

```
md"Although we generated the data manually, we extract the mass values from the
DataFrame below as Vector objects."
```

```
[110.947, 120.308, 116.931, 100.125, 111.328, 116.786, 111.071, 92.9178, 109.014, 105.275
```

```
begin
    mass_all = collect(df.Mass) # All subjects
    mass_I = collect(group_I.Mass) # Only group I subjects
    mass_II = collect(group_II.Mass) # Only group II subjects
    mass_III = collect(group_III.Mass) # Only group III subjects
end
```

Summary statistics

```
mean_mass = 101.73033851002305
```

```
mean_mass = Statistics.mean(mass_all)
```

```
mean_mass_I = 99.71311763141634
```

```
mean_mass_I = mean(mass_I)
```

```
mean_mass_II = 100.52462781132863
```

```
mean_mass_II = mean(mass_II)
```

```
mean_mass_III = 104.95327008732414
```

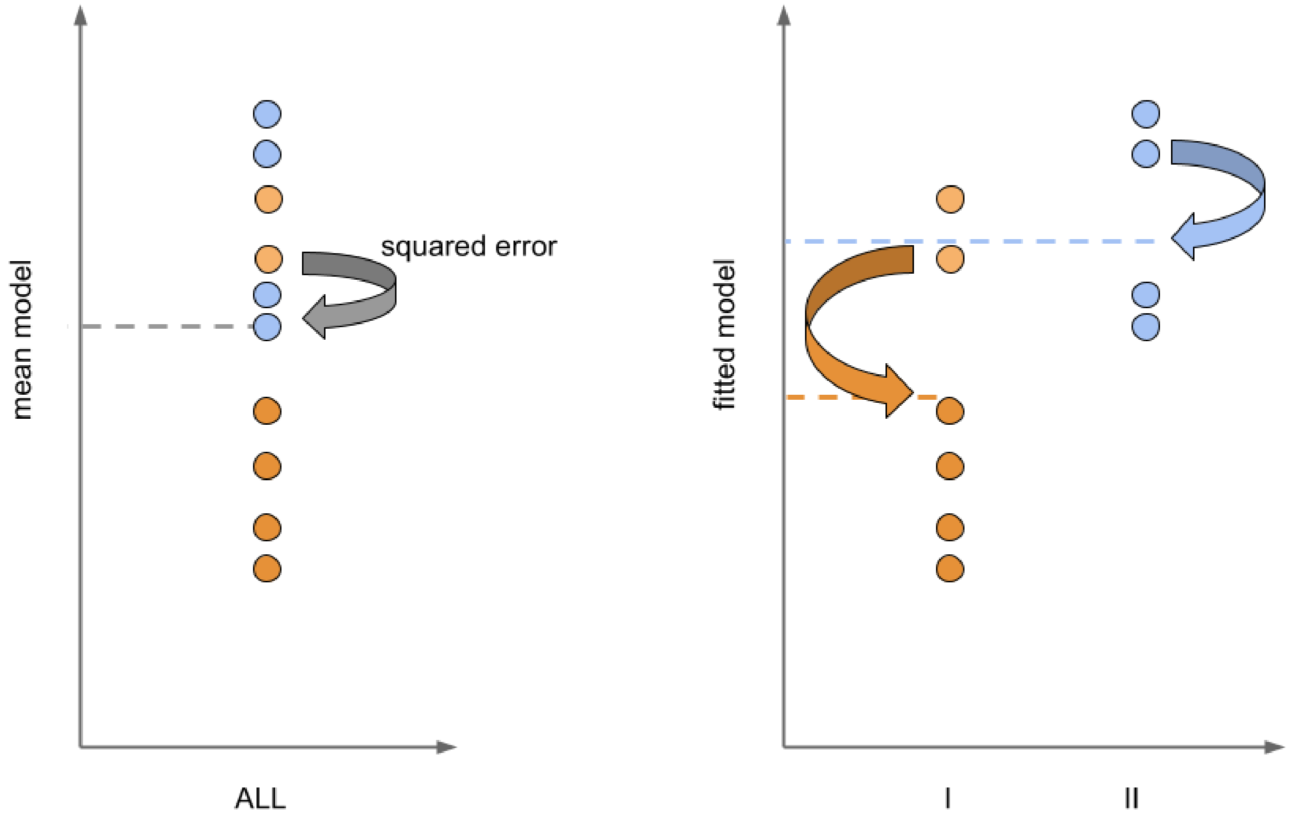
```
mean_mass_III = mean(mass_III)
```

F ratio

To compare these three (or more) means, we need a statistic. We use the *F* statistic. The *F* statistic requires a few values and two parameters. The values and parameters stem from two models (and hence the *ratio* between them).

In the image below, we see two models. The first (on the left) has the values for a variable for all the subjects. The *mean* model uses the overall mean as prediction for the variable. If we square the difference between each value and the model prediction (the mean), we get a squared error. Suming over all these squared errors gives us the sum of squared errors for the *mean* (worst) model. We denote this as the SSM.

On the right, we see the variable separated by group. In the image there are only two groups. Our example has three. Irrespective of the number of groups, we can build a model, where the mean of each group ois the predicted value for that group. We can the calculate a separate sum of squared errors for each and then sum over all the squared errors. This is termed teh sum of squared errors for the fitted model. We denote this by SSF.




```julia
PlutoUI.LocalResource("SquaredErrors.png")
```

As an example of the equation for the sum of squared errors we consider the SSM. We calculate a mean for the variable of interest for all the subjects combined. If we consider a vector of all these variable values $\mathbf{y}$ then we express the mean as $\bar{\mathbf{y}}$. The sum of squared errors for the mean (SSM) is the sum of the squared differences between the mean $\bar{\mathbf{y}}$ and each value of the variable y_i, where i is $1, 2, \ldots, n$ and n is the sample size. This is shown in (1).

$$SSM = \sum_{i=1}^{n} (\bar{y} - y_i)^2 \tag{1}$$

The two parameters reuqired for the F distribution relate to degrees of freedom. The equation for the F ratio has a numerator and denominator (each of which has a numerator and denominator). In the (overall) numerator, we have the difference between the number of parameters of the fitted model (3 in our case since we have three groups) and the number of parameters in the mean model (which is 1 since all subjects are placed together). In the denominator we have the difference between the overall sample size n and the number of parameters in the fitted model.

The equation for the F ratio is given below in (2).

$$F = \frac{\frac{SSM - SSF}{p_{\text{fit}} - p_{\text{mean}}}}{\frac{SSF}{n - p_{\text{fit}}}} \tag{2}$$

Below, we save the two parameter values.

```
3
```

```
begin
    p_mean = 1
    p_fit = 3
end
```

```
72297.54252117896
```

```
begin
    ssm = sum((mean_mass .- mass_all).^2)
    ssf_I = sum((mean_mass_I .- mass_I).^2)
    ssf_II = sum((mean_mass_II .- mass_II).^2)
    ssf_III = sum((mean_mass_III .- mass_III).^2)
    ssf = ssf_I + ssf_II + ssf_III
end
```

Below we assign the F ratio for our data to the computer variable `f_ratio` using equation (2).

```
f_ratio = 3.2679750310137083
```

```
f_ratio = ((ssm - ssf) / (p_fit - p_mean)) / ((ssf) / (n - p_fit))
```

Now we can express how likely this F ratio is.

Generating a sampling distribution by reassignement under the null hypothesis

Under the null hypothesis, which states that all three means are equal, we can randomly reassign subjects to the three groups. At each reassignment, we recaluclate the F ratio and append that to an (initially) empty vector, `f_ratio_sampling_5000`. This *builds* a sampling distribution of *F* ratio values. Below, we do this random reassignment 5000 times using a `for` loop.

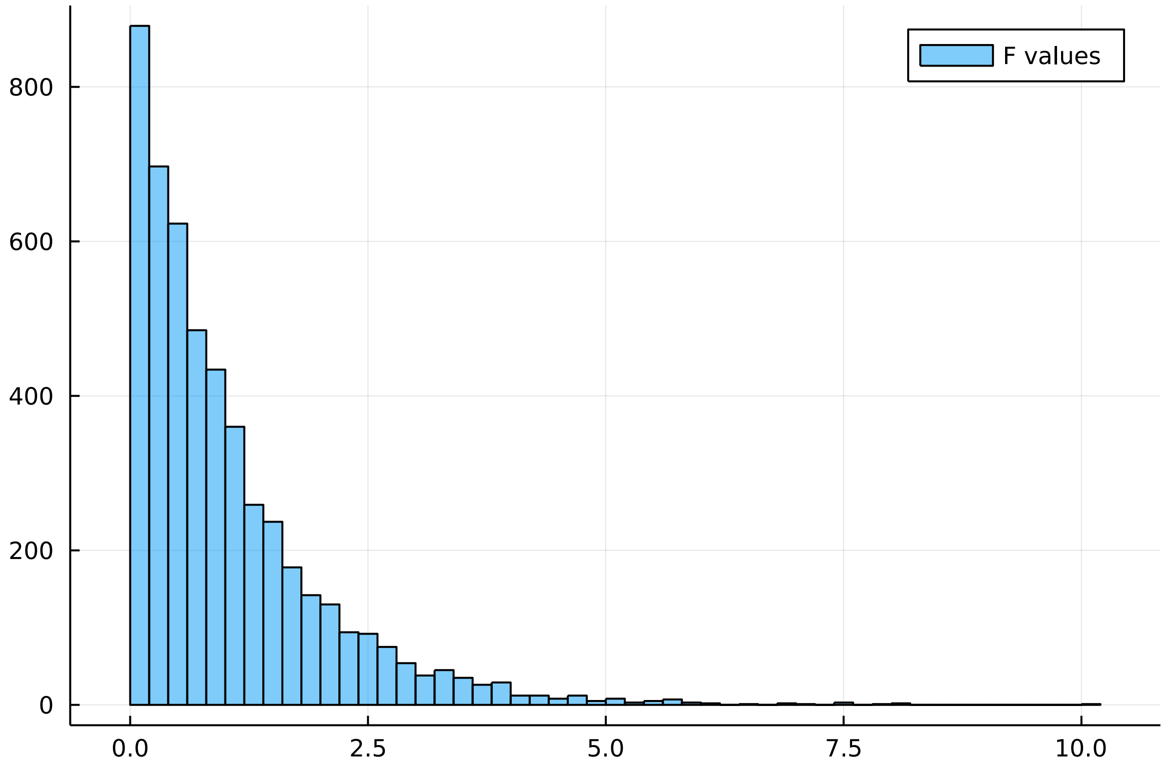
```
reassign = 5000
```

```
reassign = 5000 # Number of reasssignments
```

```
f_ratio_sampling_5000 = []; # Empty vector to hold all 20000 values
```

```julia
for i in 1:reassign
    shuffle_mass = Random.shuffle(mass_all)
    mean_shuffle_mass = mean(shuffle_mass)
    ss_mean_reassign = sum((mean_shuffle_mass .- shuffle_mass).^2)

    new_group_I = shuffle_mass[1:100]
    mean_new_group_I = mean(new_group_I)
    new_group_II = shuffle_mass[101:200]
    mean_new_group_II = mean(new_group_II)
    new_group_III = shuffle_mass[201:300]
    mean_new_group_III = mean(new_group_III)

    fit_I = sum((mean_new_group_I .- new_group_I).^2)
    fit_II = sum((mean_new_group_II .- new_group_II).^2)
    fit_III = sum((mean_new_group_III .- new_group_III).^2)
    ss_fit_reassign = fit_I + fit_II + fit_III

    reassign_f_ratio = ((ss_mean_reassign - ss_fit_reassign) / (p_fit - p_mean)) /
((ss_fit_reassign) / (n - p_fit))

    append!(f_ratio_sampling_5000, reassign_f_ratio)
end
```

We can create a histogram of all the F ratio's. Compare this to the initial plot, setting the parameters to 2 and 297.




```julia
StatsPlots.histogram(f_ratio_sampling_5000, nbins = 60, fillalpha = 0.5, label = "F
values")
```

Estimating probability of our F ratio

Finally, we can express how likely it was to have *found* our particular F ratio by expressing the fraction of resampled F ratio values were larger than ours.

```
0.041
```

```
sum(f_ratio_sampling_5000 .> f_ratio) / reassign
```

There we have it!