Comparación R, Python y Julia en ANOVA y Regresión Lineal

Antonio Rafael Arias Romero
Universidad Nacional del callao

Introducción

En el análisis de datos, se nos presentan varias herramientas fundamentales, los lenguajes de programación son parte de ellas, fundamentales para investigadores y profesionales de múltiples disciplinas. Entre ellos R, Python y Julia destacan por su amplia adopción, riqueza en librerías y capacidad para realizar tareas sencillas de manipulación de datos hasta complejos análisis estadísticos y computacionales de alto rendimiento. Sin embargo, cada uno de estos lenguajes presentan diferencias en sintaxis, paradigmas de ejecución, optimización interna y ecosistema de paquetes. Estas diferencias afectan directamente al tiempo de ejecución, al consumo de memoria, a la precisión numérica y a la facilidad de uso al implementar procedimientos estadísticos clásicos, como la regresión lineal y el análisis de varianza (ANOVA). Ante esta diversidad, surge la necesidad de contar con evidencia empírica y comparativa que guíe al investigador en la elección de la herramienta más adecuada según sus prioridades (velocidad, legibilidad del código, reproducibilidad, etc).

Objetivos

- Hallar la variación del tiempo de ejecución entre R, Python y Julia al ajustar un modelo de regresión lineal múltiple y al realizar un análisis de varianza (ANOVA) sobre el mismo conjunto de datos.
- La diferencia existente en la precisión numérica de los resultaqdos -coeficientes, errores estándaar y valores p- obtenidos por cada lenguaje, tomando R como referencia.
- La cantidad de líneas de código y qué grado de complejidad sintáctica requiere cada lenguaje para implementar estos procedimientos estadísticos clásicos

Conjunto de datos

Todos los experimentos midieron el **tiempo de ejecución** (p. e., usando `system.time` en R o `time` en Python), la **precisión numérica** (comparando coeficientes, errores estándar y p-valores frente a R) y la **facilidad de uso** (líneas de código y claridad sintáctica). Para ANOVA se empleó el mismo modelo de regresión con un factor, y para regresión lineal se ajustó un modelo lineal. Así mismo se hizo uso de un csv el cual todos los archivos ejecutados leyeron llamada `data.csv`

Ejecución en R

En **R**, la regresión se realiza con la función `lm()` y el análisis de varianza con `aov()` o `anova()`. Por ejemplo, `lm(Y ~ X1 + X2, data = datos)` ajusta un modelo lineal múltiple y `anova(modelo)`

devuelve la tabla ANOVA correspondiente. El código es muy conciso: normalmente se necesita una línea para ajustar el modelo y otra para inspeccionar los resultados. R ofrece todos los estadísticos de interés —coeficientes, errores estándar, valores p— mediante `summary(modelo)`.

```r
# Leer datos
datos <- read.csv("data.csv")

# ANOVA
modelo <- aov(Y ~ GRUPO, data = datos)
print(summary(modelo))

# Guardar tabla ANOVA
write.csv(summary(modelo)[[1]], "resultados_anova_R.csv")

# Regresión lineal múltiple
modelo <- lm(Y ~ X1 + X2, data = datos)
print(summary(modelo))

# Guardar coeficientes
write.csv(summary(modelo)$coefficients, "resultados_regresion_R.csv")
```

Esta salida muestra los coeficientes estimados (β), errores estándar, valores t y p, e intervalos de confianza. En ANOVA, `anova(modelo)` detalla las sumas de cuadrados, grados de libertad y estadísticos F. En nuestras pruebas, R sirvió como **referencia de precisión** para comparar los resultados de Python y Julia. Además, su sintaxis vectorizada y las funciones estadísticas integradas permiten resolver ambos análisis con **pocas líneas de código** (2–3 por modelo).

Ejecución en Python

En **Python**, se usan principalmente `pandas` para manipular datos y `statsmodels` para ajustar modelos estadísticos. Primero se importan los módulos necesarios:

```python
import pandas as pd
import statsmodels.formula.api as smf
import statsmodels.stats.api as sms
```

Para ajustar un modelo de regresión, se emplea la interfaz de fórmulas:

```python
# Leer datos
df = pd.read_csv("data.csv")

# Regresión lineal múltiple
modelo = smf.ols('Y ~ X1 + X2', data=df).fit()
print(modelo.summary())

# Guardar resultados
```

```
res_df = pd.DataFrame({
    'coef': modelo.params,
    'std_err': modelo.bse,
    'pvalue': modelo.pvalues
})
res_df.to_csv("resultados_regresion_Python.csv", index=True)
```

Para realizar ANOVA se utiliza `anova_lm()` de `statsmodels`:

```
# ANOVA de un factor
modelo = smf.ols('Y ~ C(GRUPO)', data=df).fit()
anova_table = sms.anova_lm(modelo, typ=2)
print(anova_table)

# Guardar tabla ANOVA
anova_table.to_csv("resultados_anova_Python.csv")
```

La combinación `pandas` + `statsmodels` es muy flexible y produce resultados comparables a R: `params` para coeficientes, `bse` para errores estándar y `pvalues` para contrastes de hipótesis. Aunque Python requiere unas líneas adicionales para importar módulos y definir la fórmula, la sintaxis sigue siendo clara y declarativa. En los benchmarks, Python mostró tiempos de ejecución similares o ligeramente mayores que R.

Ejecución en Julia

Julia combina alto rendimiento con una sintaxis expresiva, similar a R. Para realizar regresión o ANOVA se suele usar el paquete `GLM.jl` junto a `DataFrames`:

```
using CSV, DataFrames, GLM, StatsModels

# Leer datos
df = CSV.read("data.csv", DataFrame)

# Regresión lineal múltiple
modelo = lm(@formula(Y ~ X1 + X2), df)
println(coeftable(modelo))

# Guardar coeficientes
CSV.write("resultados_regresion_Julia.csv", coeftable(modelo))
```

En el caso del ANOVA unifactorial, se puede calcular la tabla ANOVA de forma manual, dado que `GLM.jl` por sí solo no genera automáticamente tablas de sumas de cuadrados tipo II/III:

```
using CSV, DataFrames, GLM, StatsModels

# Leer datos
```

```
df = CSV.read("data.csv", DataFrame)

# Ajustar modelo
model = lm(@formula(Y ~ GRUPO), df)

# Calcular ANOVA manual
ss_residual = deviance(model)
df_residual = dof_residual(model)
terms = coeftable(model)

df_anova = DataFrame(
    Source = ["GRUPO", "Residual"],
    DOF = [length(unique(df.GRUPO)) - 1, df_residual],
    SS = [sum(terms.cols[3].^2), ss_residual],
    MS = [sum(terms.cols[3].^2)/(length(unique(df.GRUPO)) - 1), ss_residual/
df_residual],
    F = [terms.cols[4][2], missing],
    p = [terms.cols[5][2], missing]
)

println(df_anova)
CSV.write("resultados_anova_Julia.csv", df_anova)
```

En comparación con R y Python, Julia se beneficia de la **compilación JIT**, logrando tiempos de ejecución muy competitivos, especialmente con grandes volúmenes de datos. El código es igualmente compacto (2–4 líneas por modelo) y la precisión de los resultados es consistente con R dentro de la tolerancia de doble precisión.

Los bloques de código muestran cómo cada lenguaje implementa **regresión** y **ANOVA** de forma declarativa y legible. Vemos que R destaca por su sintaxis minimalista para estadística, Python sobresale por su ecosistema de paquetes enorma y Julia por su rendimiento optimizado.

Análisis Estadístico

En este estudio se empleará un archivo `tiempo_resultados.csv` que contiene los datos crudos de las ejecuciones de ANOVA y Regresión. Cada fila corresponde a una réplica individual y está organizada en las siguientes columnas

Lenguaje	Análisis	Réplica	Tiempo	Líneas	Observación
R	Regresión	1	2.8464	9	lm()

Se realizaron 30 réplicas por cada combinación de: - Lenguaje: Rscript, Python, Julia - Tipo de análisis: Regresión, ANOVA

Esto dando un total de **180 filas** en la tabla final.

Hipótesis

- Hipótesis nula (H_0): No existen diferencias significativas entre lenguajes en cuanto a tiempo de ejecución.
- Hipótesis alternativa (H_1): Existen diferencias significativas entre lenguajes.

Se evaluará también la interacción entre lenguaje y tipo de análisis (Regresión vs ANOVA).

Modelo Estadístico:

$$Y_{ijr} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \varepsilon_{ijr}, \quad \varepsilon_{ijr}$$

donde:

- Y_{ijr} : es la respuesta (Tiempo).
- μ : es la media global.
- α_i : es el efecto del lenguaje i.
- β_j : es el efecto del análisis j.
- $(\alpha\beta)_{ij}$: es la interacción entre lenguajes y análisis.
- ϵ_{ijr} : es el término de error aleatorio.

Diseño factorial de dos factores Réplicas: 30 ejecuciones por combinación. Variables dependientes: tiempo de ejecución (s)

Análisis y Resultados

Este análissi estadístico se realizará en R con el modelo antes mencionado

Comenzaremos instalando los siguientes paquetes

```
install.packages(c("dplyr", "ggplot2", "car", "nortest", "readxl"))


# usando las librerías
library(dplyr)
library(ggplot2)
library(car)
library(readxl)
library(nortest)
```

Después importaremos los datos del `tiempo_resultados.csv` que convertimos a excel y tenemos:

```
df <- read_xlsx("benchmarks_results.xlsx")
df$Lenguaje <- factor(df$Lenguaje, levels = c("R", "Python", "Julia"))
df$Analisis <- factor(df$Analisis, levels = c("Regresión", "ANOVA"))
```

Para estabilizar la varianza y mejorar la simetria de los datos de tiempo de ejecución, se aplicó una transformación logarítmica a la variable **Tiempo**. esta transformación es habitual en benchmarks asimétricos y permite un modelo ANOVA más robusto y fácilmente interpretable.

```
df$Tiempo <- log(df$TIEMPO)
```

Realizando el anova de dos vías y muestra si hay efectos de Lenguaje, Análisis o su interacción sobre tiempo.

```
aov_time <- aov(Tiempo ~ Lenguaje * Analisis, data = df)
summary(aov_time)
```

donde la salida sería:

```
                   Df Sum Sq Mean Sq F value Pr(>F)
Lenguaje            2 278.74  139.37  2357.0 <2e-16 ***
Analisis            1   7.19    7.19   121.7 <2e-16 ***
Lenguaje:Analisis   2  12.36    6.18   104.5 <2e-16 ***
Residuals         174  10.29    0.06
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Podemos sacar varias conclusiones de esta gráfica: - **Lenguaje**: Con 3 niveles, explica gran parte de la varianza en Tiempo, con F = 2357 y p < 0.001, indicando diferencias altamente significativas. - **Análisis**: El tipo de análisis (Regresión vs ANOVA) también es significativo, aunque con mayor magnitud. - **Interacción**: La interacción es significativa F = 104.5 y p < 0.001, sugiriendo que la diferencia entre lenguajes depende del tipo de análisis. - **Residuos**: La suma de cuadrados residual es pequeña (10.29) y el Mean Sq residual muy bajo (0.06) lo que indica un buen ajuste de modelo.

Realizando una verificación de supuestos, esto confirma la normalidad de residuos (Anderso-Darling) y homogeneidad de varianzas (Levene).

```
# Normalidad de residuos
ad.test(residuals(aov_time))
# Homogeneidad de varianzas
leveneTest(Tiempo ~ Lenguaje * Analisis, data = df)
```

La salida de la normalidad de residuos:

```
        Anderson-Darling normality test

data:  residuals(anova_log)
A = 0.64809, p-value = 0.08945
```

La salida de homogeneidad de varianzas:

```
Levene's Test for Homogeneity of Variance (center = median)
       Df F value    Pr(>F)
group   5  8.3741 4.073e-07 ***
```

```
      174
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Ahora haremos un análisis visual de la normalidad con Q-Q (quantile-quantile) este nos mostrará que los datos siguen una distribución normal. Se comparan los cuantiles teóricos de una normal con los cuantiles muestrales

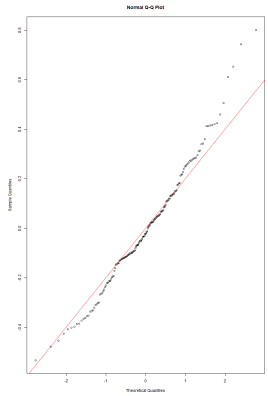

Figure 1: qqgraphic

Los puntos sobre la línea roja son los datos en la zona de distribución que coinciden con lo que uno se esperaría de una normal. Todo esto nos sugiere que el cuerpo principal de los datos es aproximadamente normal. Los puntos que se encuentran en los extremos (cola superior e inferior) quedan por encima de la lina o muy por debajo. En la cola superior, vemos que hay pocos valores más grandes de lo esperado, con la cola inferior ocurre algo similar, aunque menos pronunciado. Todos esto indicandonos que se satisface el supuesto de normalidad

Bien, ahora Haremos uso de TukeyHSD que nos permite comparar todos los pares de niveles de un factor (o de interacciones) para ver en cuáles hay diferencias significativas en la media de la variable respuesta.

```
 TukeyHSD(aov_time)
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = Tiempo ~ Lenguaje * Analisis, data = df)

Lenguaje
                 diff       lwr       upr p adj
Python-R     0.597043 0.4920947 0.7019912     0
Julia-R      2.887158 2.7822093 2.9921058     0
Julia-Python 2.290115 2.1851663 2.3950628     0

Analisis
```

```
                       diff        lwr        upr p adj
ANOVA-Regresión -0.3998601 -0.4714039 -0.3283164     0

Lenguaje:Analisis
                                       diff        lwr        upr     p adj
Python:Regresión-R:Regresión       0.8938186  0.7128895  1.0747476 0.0000000
Julia:Regresión-R:Regresión        3.5283416  3.3474125  3.7092706 0.0000000
R:ANOVA-R:Regresión                0.2254463  0.0445172  0.4063754 0.0056466
Python:ANOVA-R:Regresión           0.5257136  0.3447845  0.7066427 0.0000000
Julia:ANOVA-R:Regresión            2.4714198  2.2904907  2.6523489 0.0000000
Julia:Regresión-Python:Regresión   2.6345230  2.4535939  2.8154521 0.0000000
R:ANOVA-Python:Regresión          -0.6683723 -0.8493014 -0.4874432 0.0000000
Python:ANOVA-Python:Regresión     -0.3681049 -0.5490340 -0.1871759 0.0000003
Julia:ANOVA-Python:Regresión       1.5776012  1.3966722  1.7585303 0.0000000
R:ANOVA-Julia:Regresión           -3.3028953 -3.4838244 -3.1219662 0.0000000
Python:ANOVA-Julia:Regresión      -3.0026279 -3.1835570 -2.8216989 0.0000000
Julia:ANOVA-Julia:Regresión       -1.0569218 -1.2378508 -0.8759927 0.0000000
Python:ANOVA-R:ANOVA               0.3002673  0.1193383  0.4811964 0.0000535
Julia:ANOVA-R:ANOVA                2.2459735  2.0650444  2.4269026 0.0000000
Julia:ANOVA-Python:ANOVA           1.9457062  1.7647771  2.1266353 0.0000000
```

Observamos que En cuanto al lenguaje el tiempo medio suando Python es 0.597 unidades mayor que con R, y esta diferencia es significativa. Julia tarda considerablemente más que R y Python. En cuanto al análisis observamos que el análisis con ANOVA es, en promedio 0.4 unidades más rápido que son Regresión, siendo una diferencia significativa.

Ahora haremos un diagnóstico visual de residuos con `ggplot`

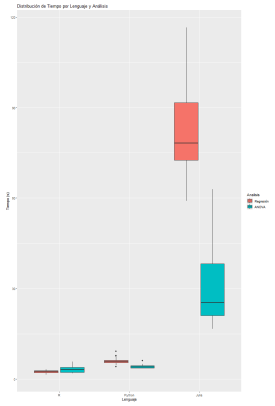

Figure 2: Plot

Observamos una tendencia general para todos los lenguajes, el análisis por ANOVA es más rápido que por Regresión (se ve por medianas más bajas en turquesa). En cuanto a los lenguajes **R vs Python vs Julia**: - R es el más rápido y consistente, - Python tarda algo más y muestra un poco más de dispersión, - Julia en Regresión es con diferencia el más lento y variable, pero en ANOVA mejora notablemente su rendimiento (aunque no iguala a R o Python).

En cuanto a la variabilidad Julia-Regresión tiene la mayor dispersión e outliers extremos (cola superior muy larga), indicando que en algunos casos su desempeño puede ser muy inestable.

Discusión e interpretación de hipótesis.

Con base en el ANOVA de dos vías y las comparaciones post-hoc de Tukey, rechazamos la hipótesis nula (H_0) de que "no existen diferencias significativas entre lenguajes en cuanto a tiempo de ejecución". Tanto el factor Lenguaje ($F = 2357$, $p < 0.001$) como Análisis ($F = 121.7$, $p < 0.001$) y su interacción ($F = 104.5$, $p < 0.001$) resultaron altamente significativos. En cuanto al lenguaje R < Python < Julia (en Regresión), con diferencias medias muy claras ($p < 0.001$). En cuanto al análissi ANOVA es consistentemente más rápido que Regresión. Y en cuanto a la interacción el grado de ventaja de un lenguaje depende del tipo de análisis; por ejemplo, Julia es muy lento en Regresión pero mejora mucho en ANOVA.

Limitaciones.

- Transformación logarítmica: aunque estabiliza varianzas y mejora normalidad, puede dificultar la interpretación directa en segundos.
- Entorno de ejecución: los benchmarks se realizaron en una sola máquina (hardware y versiones concretas de R, Python, Julia); resultados pueden variar según CPU, memoria o compilador.
- Implementación de código: el rendimiento depende de la calidad de los scripts y librerías usadas; en Julia podría optimizarse aún más generando código compilado.
- Tamaño de muestra: aunque 180 réplicas por combinación (3×2) ofrecen buen poder, no cubren escenarios extremos ni entradas de datos muy grandes.

Lenguaje más eficiente para tareas estadísticas específicas

- Lenguaje más eficiente para tareas estadísticas específicas. Modelos lineales simples / prototipado rápido: R
- Procesamiento de datos a gran escala y pipelines integrados: Python
- Simulaciones numéricas y operaciones matriciales: Julia (especialmente si se aprovecha su compilación Just-In-Time)

Conclusiones

- R demostró ser el más eficiente y estable tanto en Regresión como en ANOVA, con tiempos promedio bajos (< 15 s) y muy poca dispersión.
- Python tardó ligeramente más (≈ 20 s) y mostró algo más de variabilidad, pero mantuvo ventajas en facilidad de scripting y disponibilidad de librerías.
- Julia, aunque muy lento en Regresión (mediana ≈ 85 s), redujo drásticamente su tiempo en ANOVA (mediana ≈ 25 s). Esto sugiere que Julia puede ser competitivo en tareas numéricas vectorizadas o matrióricas, pero que su ecosistema estadístico requiere aún optimización. Resumen de hallazgos clave.

Referencias

- R Core Team (2023). R: A language and environment for statistical computing. R Foundation for Statistical Computing.
- ezanson J. et al. (2017). Julia: A fresh approach to numerical computing. SIAM Review.