Laboratorio Nacional de Políticas Públicas

CIDE CENTRO DE INVESTIGACIÓN Y DOCENCIA ECONÓMICAS A.C.

# Introducción a regex con rebus

**Laboratorio Estadística.**

**Noviembre, 2019**

**M.C. JORGE JUVENAL CAMPOS FERREIRA.**
Asistente de investigación.
Laboratorio Nacional de Políticas Públicas
CIDE

# Introducción

## Requerimientos para la sesión:

**Instalar los paquetes {rebus}, {stringr} y {htmltools}.**

# Introducción

**Como vimos la clase pasada con Sebastián, las funciones de la librería stringr funcionan con algo llamado "patrones".**

```
str_view(string, pattern, match = NA)

str_view_all(string, pattern, match = NA)
```

**Sin embargo... ¿qué son y cómo podemos generar esos patrones?**

**Para eso, en programación contamos con una herramienta muy poderosa conocida como *regex* o *expresiones regulares*.**

# Regex

**¿Cómo se ven las expresiones regulares?**

**Las regex son cadenas de texto que, a través de símbolos predefinidos, nos sirven para detectar un patrón en un texto.**

# Regex

**¿Cómo se ven las expresiones regulares?**

**Las regex son cadenas de texto que, a través de símbolos predefinidos, nos sirven para detectar un patrón en un texto.**

**Ejemplo:**

```
(?<![\\w\\d])Retweet(?![\\w\\d])(\\s[\\d\\.(K|M)?]+)?
```

**(Regex para capturar el numero de retweets de una base de datos en TW)**

# Regex

**¿Cómo se ven las expresiones regulares?**

**Las regex son cadenas de texto que, a través de símbolos predefinidos, nos sirven para detectar un patrón en un texto.**

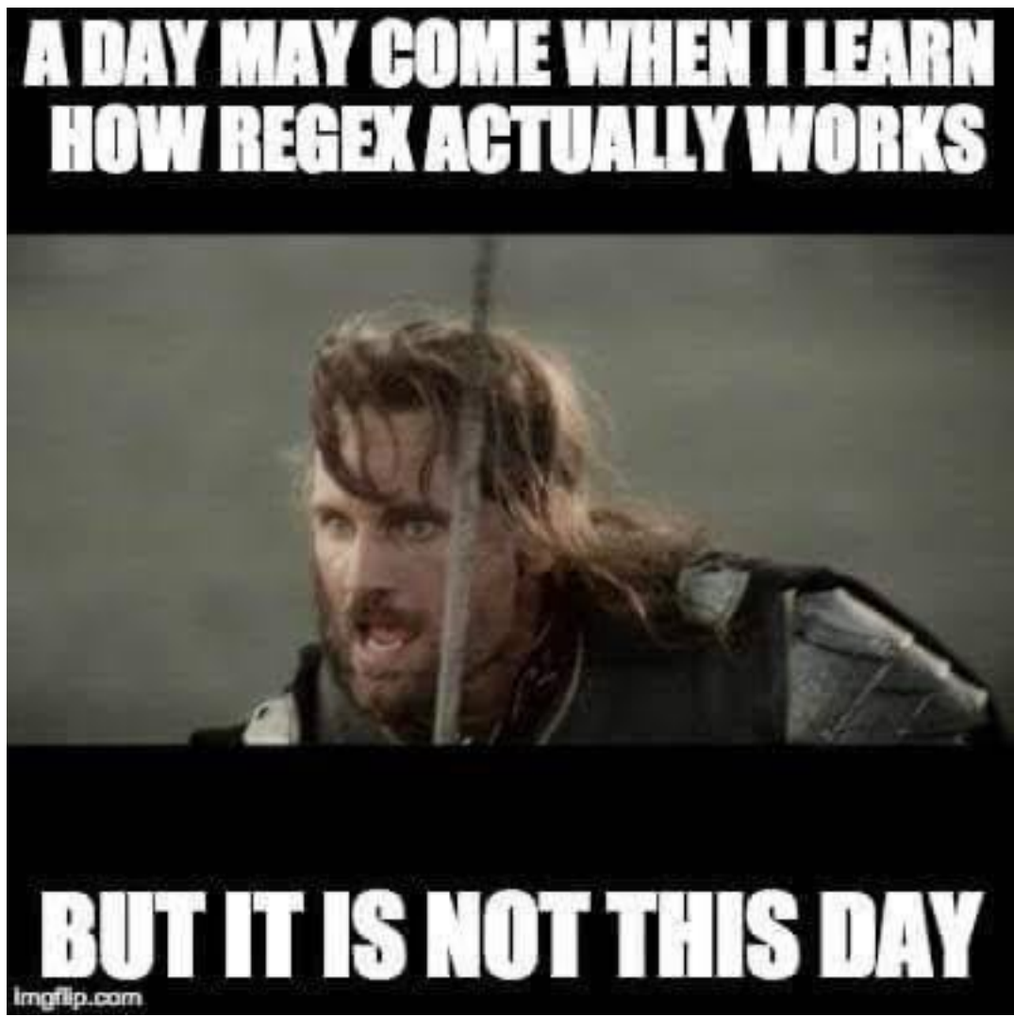
**Ejemplo:**

```
(?<![\\w\\d])Retweet(?![\\w\\d])(\\s[\\d\\.(K|M)?]+)?
```

**(Regex para capturar el numero de retweets de una base de datos en TW)**


# Regex

**Las regex son conceptos complicados (incluso para los programadores), así que tómenlo con calma.** 🙂







# {rebus}

**Un programador de R diseñó una forma menos dolorosa para poder aprender (y utilizar) estos conceptos sin tanto sufrimiento.**

**La librería {rebus} nos da la facilidad de construir expresiones regulares de manera más intuitiva, al mismo tiempo en que nos permite ir aprendiendo a trabajar con las regex crudas.**

**`rebus: Build Regular Expressions in a Human Readable Way`**

Build regular expressions piece by piece using human readable code. This package is designed for interactive use.

# Objetos pre-programados {rebus}

| Patrón | Expresión Regular | Objeto rebus |
|---|---|---|
| Inicio de un string o cadena de texto | ^ | START |
| Final de un string | $ | END |
| Cualquier carácter sencillo | . | ANY_CHAR |
| Punto literal, gorrito o signo de pesos | \. \^ \$ | DOT, CARAT, DOLLAR |

# Objetos pre-programados {rebus}

| Patrón | Expresión Regular | Objeto rebus |
|---|---|---|
| Espacio | \s | SPC |
| Dígito | \d | DGT |
| Letra | \w | WRD |
| Minúsculas ASCII | a-z | ASCI_LOWER |

# Funciones de repetición {rebus}

| Patrón | Expresión Regular | Función rebus |
|---|---|---|
| Opcional | ? | optional() |
| Zero o más | * | zero_or_more() |
| Uno o más | + | one_or_more() |
| Entre *n* y *m* veces | {n}{m} | repeated() |

# Funciones:

## **stringr::str_view(***string, pattern, match***)**
## **stringr::str_view_all(***string, pattern, match***)**

**Esta función nos permite probar nuestros intentos de expresiones regulares. “Para atrapar lo que queremos atrapar.”**

```
str_view_all(contact, pattern =  DGT , match = TRUE)
```

```
Call me at 555-555-0191
123 Main St
(555) 555 0191
Phone: 555.555.0191 Mobile: 555.555.0192
```

# Función char_class()

**Esta función sirve para definir un conjunto de caracteres que van a formar parte del patrón. Por ejemplo:**

```
library(rebus)
c <- char_class("aeiouAEIOUñ@")
str_view_all("Estos niñ@s son mis Alumnos", pattern = c)
```

**En este caso, creamos un objeto en el cual el patrón a detectar va a ser todas las vocales, minúsculas y mayúsculas, la letra ñ y el arroba. Abajo, podemos ver lo que captura este patrón:**

# Función %R% (pipa rebus, concatenar)

**Esta función sirve para concatenar objetos rebus, para poder armar patrones compuestos y más complejos.**

**Ejemplo de uso:**

```
pat <- START %R% WRD %R% DGT %R% capture(one_or_more(SPC)) %R% END
```

**Patrón compuesto.**

# Manos a la obra

**A continuación vamos a llevar a cabo un ejemplo**

# Manos a la obra

## 1. Leemos las librerías

```
library(rebus)
library(stringr)
```

## 2. Generamos texto

```
# Some strings to practice with
x <- c("cat", "coat", "scotland", "tic toc")


# Print END
END
```

## 3. Primer patrón!
## Generamos un patrón de las palabras que empiezan con la letra “c”

```
# Run me
str_view(x, pattern = START %R% "c")
```

# Ejemplos

## 4. Resultado
## Se marca en obscuro la letra c inicial.

```
# Run me
str_view(x, pattern = START %R% "c")
```

cat

coat

scotland

tic toc

# Ejemplos

### 5. Ahora, las que terminen en “-at”

```
# Match the strings that end with "at"
str_view(x, pattern = "at" %R% END)
```

```
cat
coat
scotland
tic toc
```

# Ejemplos

## 6. Palabras que llevan un caracter, y luego llevan una “t”

```
x <- c("cat", "coat", "scotland", "tic toc")

# Match any character followed by a "t"
str_view(x, pattern =  ANY_CHAR %R% "t")
```

cat

coat

scotland

tic toc

# Ejemplos

## 7. Palabras de exactamente 3 caracteres.

```
# Match a string with exactly three characters
str_view(x, pattern = START %R% ANY_CHAR %R% ANY_CHAR %R% ANY_CHAR
%R% END)
```

```
cat

coat

scotland

tic toc
```

# Ejercicio.


**Abramos RStudio y corramos el ejemplo que les envié a su correo.**