# Expresiones Regulares y Loops

## Fundamentos teóricos para Ciencia de Datos

Nicolás Sidicaro
Abril 2025

# 1. Buenas prácticas para análisis de

## ¿Por qué importa el procesamiento de texto?

- Los datos textuales rara vez vienen "limpios" y estandarizados
- Pequeñas inconsistencias pueden generar grandes problemas analíticos
- La normalización facilita búsquedas, comparaciones y análisis
- Esencial como paso previo a cualquier análisis textual avanzado

## Conceptos claves:

- **Normalización**: Proceso para convertir texto a una forma estándar
- **Tokenización**: División del texto en unidades más pequeñas (palabras, frases)
- **Estandarización**: Aplicación de reglas consistentes para representar texto

# Normalización básica de texto

## Transformaciones fundamentales:

- **Conversión a mayúsculas o minúsculas**
  - Recomendación: convertir todo a minúsculas para análisis
  - Preservar mayúsculas solo cuando la capitalización es relevante (nombres propios)
- **Tratamiento de espacios**
  - Eliminar espacios iniciales y finales
  - Convertir espacios múltiples en espacios simples
  - Estandarizar separadores (tabulaciones, saltos de línea)
- **Eliminación de caracteres no deseados**
  - Caracteres no imprimibles
  - Símbolos específicos según contexto (emojis, caracteres especiales)
  - HTML y marcado si procede de fuentes web

# Tratamiento de caracteres especiales

## Normalización de caracteres acentuados:

- **Dos enfoques principales**:
  - **Conservar**: Mantener todos los acentos y diacríticos (mejor para análisis lingüístico)
  - **Eliminar**: Convertir a equivalentes sin acentos (mejor para búsquedas e indexación)
- **Consideraciones**:
  - Consistencia en todo el conjunto de datos
  - Contexto cultural y lingüístico del análisis
  - Capacidades del sistema y herramientas de análisis

## Manejo de codificación:

- Utilizar UTF-8 como estándar para compatibilidad universal
- Verificar y corregir problemas de codificación ("Ã±" por "ñ")
- Documentar las transformaciones realizadas

# Estandarización de formatos específicos

## Datos estructurados dentro de texto:

- **Fechas y horas**:
  - Convertir a un formato estándar (ISO: YYYY-MM-DD)
  - Considerar zonas horarias si son relevantes
- **Números y cantidades**:
  - Estandarizar separadores decimales (punto vs coma)
  - Normalizar representaciones de miles
  - Convertir fracciones a decimales
- **Información de contacto**:
  - Teléfonos: eliminar paréntesis, guiones, espacios
  - Códigos postales: formato consistente
  - Direcciones: estructura normalizada

# Consideraciones multilingüe

## Desafíos específicos:

- **Variaciones ortográficas regionales**
  - Español de España vs América Latina (coger/agarrar)
  - Inglés británico vs americano (colour/color)
- **Caracteres específicos de cada idioma**
  - Respeto a caracteres propios (ñ, ç, ß, etc.)
  - Consideración de alfabetos no latinos
- **Traducción vs análisis nativo**
  - Definir si se analizará cada idioma por separado
  - O si se traducirá todo a un idioma común
  - Impacto de la traducción automática en el análisis

# Más allá de la limpieza básica

## Técnicas complementarias:

- **Corrección ortográfica**
  - Detección y corrección de errores comunes
  - Consideración de jerga y términos técnicos
- **Normalización semántica**
  - Unificación de sinónimos
  - Resolución de abreviaturas y acrónimos
  - Identificación de entidades nombradas
- **Lematización/stemming**
  - Reducción a forma base de las palabras
  - Útil para análisis que ignora tiempos verbales o plurales

# 2. Expresiones Regulares: Introducción

## ¿Qué son las expresiones regulares?

- Lenguaje para describir patrones en texto
- Sistema universal presente en casi todos los lenguajes de programación
- Herramienta fundamental para manipulación y validación de texto
- Parte esencial del arsenal de cualquier científico de datos

## Aplicaciones en ciencia de datos:

- Validación de formatos (emails, teléfonos, URLs)
- Extracción de información estructurada desde texto no estructurado
- Limpieza y transformación de datos textuales

# Fundamentos de expresiones regulares

## Metacaracteres principales:

- `.` → Cualquier carácter excepto nueva línea
- `^` → Inicio de cadena/línea
- `$` → Final de cadena/línea
- `\` → Escape para caracteres especiales
- `|` → Alternativa (OR lógico)

## Cuantificadores:

- `*` → 0 o más repeticiones
- `+` → 1 o más repeticiones
- `?` → 0 o 1 ocurrencia (opcional)
- `{n}` → Exactamente n repeticiones
- `{n,m}` → Entre n y m repeticiones
- `{n,}` → Al menos n repeticiones

# Clases de caracteres y agrupamiento

## Clases de caracteres:

- `[abc]` → Cualquiera de estos caracteres
- `[^abc]` → Cualquier carácter excepto estos
- `[a-z]` → Rango de caracteres (minúsculas a-z)
- `[A-Z0-9]` → Letras mayúsculas o dígitos

## Clases predefinidas:

- `\d` → Dígito (`[0-9]`)
- `\w` → Carácter de palabra (`[a-zA-Z0-9_]`)
- `\s` → Espacio en blanco
- `\D`, `\W`, `\S` → Negaciones de las anteriores

## Grupos:

- `( ... )` → Captura un grupo
- `(?: ... )` → Grupo sin captura
- `\1`, `\2`, etc. → Referencias a grupos capturados

# Patrones comunes para análisis de texto

## Extracción de segmentos:

- Todo hasta el primer punto: `^[^.]*`
- Desde el inicio hasta cierta palabra: `^.*?palabra`
- Desde una palabra hasta el final: `palabra.*$`
- Entre dos delimitadores: `inicio(.*?)fin`

## Validación de formatos:

- Email básico: `^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9•–]+\.[a-zA-Z]{2,}$`
- URL simple: `https?://[a-zA-Z0-9•–]+\.[a-zA-Z]{2,}(/.*)?`
- Fecha (YYYY-MM-DD): `\d{4}-\d{2}-\d{2}`
- Teléfono argentino: `(\+54|0)?\s?9?\s?[1-9][0-9]{2}\s?[0-9]{3}\s?[0-9]{4}`

# Análisis web con expresiones regulares

## Patrones para URLs:

- Protocolo: `^(https?|ftp)://`
- Dominio y TLD: `[a-zA-Z0-9•–]+\.[a-zA-Z]{2,}`
- Parámetros de URL: `[\?&]([^=]+)=([^&]*)`
- Fragmento (anchor): `#([^#]*$)`

## Parámetros UTM en Google Analytics:

- Fuente: `utm_source=([^&]+)`
- Medio: `utm_medium=([^&]+)`
- Campaña: `utm_campaign=([^&]+)`
- Contenido: `utm_content=([^&]+)`
- Término: `utm_term=([^&]+)`

# Técnicas avanzadas

## Lookaheads y lookbehinds:

- **Lookahead positivo** `(?= ... )`: Coincide si lo que sigue cumple el patrón
- **Lookahead negativo** `(?! ... )`: Coincide si lo que sigue NO cumple el patrón
- **Lookbehind positivo** `(?<= ... )`: Coincide si lo precedente cumple el patrón
- **Lookbehind negativo** `(?<! ... )`: Coincide si lo precedente NO cumple el patrón

## Aplicaciones avanzadas:

- Validación de contraseñas complejas
- Extracción de información con contexto
- Análisis de documentos estructurados
- Procesamiento de logs y datos de eventos

# Expresiones regulares: Consideraciones

## Limitaciones y buenas prácticas:

- No es recomendable analizar HTML/XML solo con regex (usar parsers específicos)
- Las expresiones complejas pueden ser difíciles de mantener y depurar
- Dividir patrones complejos en partes más manejables
- Probar exhaustivamente con casos límite
- Documentar el propósito de expresiones complejas

## Herramientas de ayuda:

- Sitios como regex101.com para probar y depurar expresiones
- Generadores de expresiones regulares para casos comunes
- Visualizadores de patrones para entender el funcionamiento
- Cheatsheets específicas según el lenguaje/motor de regex

# 3. Loops y alternativas funcionales

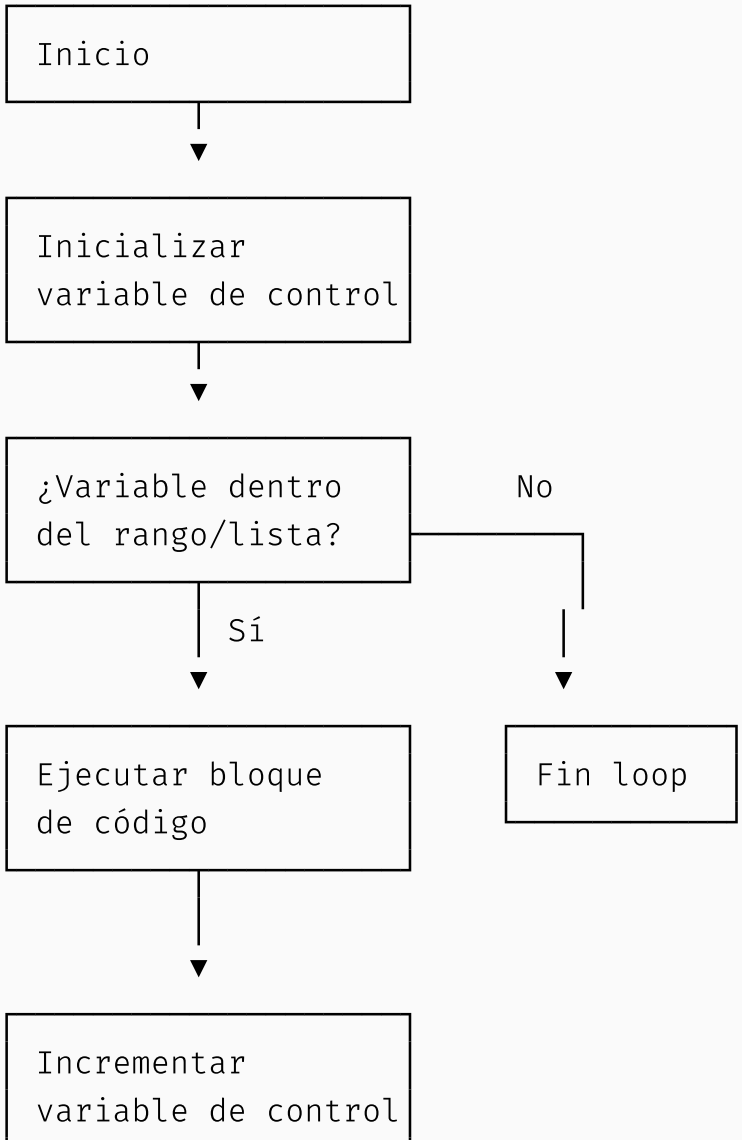
## ¿Qué son los loops?

- Estructuras que permiten repetir código varias veces
- Formas básicas de automatización en programación
- Elementos fundamentales para procesar colecciones de datos
- Presentes en todos los lenguajes de programación

## Tipos principales de loops:

- **For loop**: Iteración sobre una secuencia conocida
- **While loop**: Ejecución mientras se cumpla una condición

# Funcionamiento del For Loop

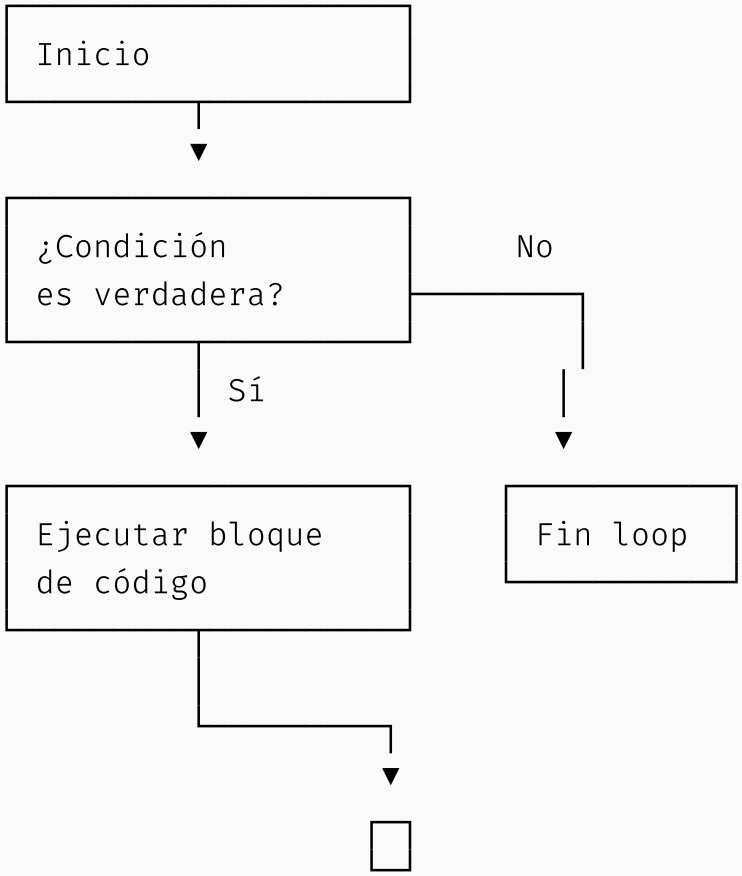


# For loop

## Características clave:

- **Número predefinido** de iteraciones
- La variable de control se **inicializa** al comienzo
- En cada ciclo se **comprueba** si debe continuar
- El bloque se ejecuta solo si la condición es verdadera
- La variable se **actualiza** automáticamente al final de cada ciclo

# Funcionamiento del While Loop



# While loop

## Características clave:

- **Evaluación de condición** al inicio de cada iteración
- Número de iteraciones **desconocido** de antemano
- **No se ejecuta ninguna vez** si la condición es falsa inicialmente
- Requiere que algo dentro del bloque eventualmente **cambie la condición**
- Riesgo de **bucle infinito** si la condición nunca cambia a falsa

# For loops vs While loops

## For loops:

- Utilizados cuando sabemos exactamente cuántas iteraciones necesitamos
- La variable de control se incrementa automáticamente
- Estructura más declarativa y generalmente más legible
- Ejemplo conceptual: "Para cada elemento en esta lista, haz algo"

## While loops:

- Utilizados cuando no sabemos cuántas iteraciones se necesitarán
- Requieren una condición de salida explícita
- Mayor flexibilidad pero mayor riesgo (loops infinitos)
- Ejemplo conceptual: "Mientras esta condición sea verdadera, sigue haciendo algo"

# Enfoque funcional: en vez de loops

## Programación funcional:

- Paradigma donde las funciones son ciudadanos de primera clase
- Enfoque declarativo vs imperativo de los loops
- Énfasis en inmutabilidad y funciones puras
- Favorece composición sobre instrucciones paso a paso

## Ventajas del enfoque funcional:

- Código más conciso y expresivo
- Menos propenso a errores (sin efectos secundarios)
- Mejor paralelización potencial
- Mayor nivel de abstracción (puede ser desventaja)

# Apply vs Map: ¿Cuándo usar cada uno?

## Familia apply en R:

- Conjunto de funciones para aplicar operaciones a matrices, listas, etc.
- `apply()` para matrices y arrays
- `lapply()` para listas, devuelve lista
- `sapply()` para listas, intenta simplificar el resultado
- `tapply()` para aplicar funciones por grupos

## Funciones map (purrr):

- Alternativa más moderna y consistente a la familia apply
- Tipado más consistente (`map_dbl`, `map_chr`, etc.)
- Mejor integración con el tidyverse
- Sintaxis más intuitiva con fórmulas y funciones anónimas

# Loops vs Funcionales: Guía de selección

## Cuándo usar loops tradicionales:

- Cuando la lógica de iteración es compleja o tiene múltiples condiciones
- Al modificar la estructura que se está iterando
- Cuando se requiere control fino sobre el flujo (break, continue)
- Al trabajar con operaciones que tienen efectos secundarios
- Para algoritmos con lógica que depende del estado anterior

## Cuándo usar enfoque funcional (apply/map):

- Para operaciones simples y uniformes en cada elemento
- Cuando la operación es "pura" (sin efectos secundarios)
- Para transformaciones de datos directas
- Cuando se prioriza legibilidad y concisión
- En operaciones que se benefician de paralelización

# 4. Introducción a ETL

## ¿Qué es ETL?

- **E**xtract, **T**ransform, **L**oad (Extraer, Transformar, Cargar)
- Proceso central en data warehousing e integración de datos
- Pipeline para mover datos entre sistemas
- Componente esencial en arquitecturas de datos modernas

## Importancia del ETL:

- Unifica datos de múltiples fuentes
- Garantiza calidad y consistencia
- Prepara datos para análisis
- Mantiene históricos y trazabilidad
- Automatiza flujos de información

# Fase 1: Extracción de datos

## Fuentes comunes:

- Bases de datos relacionales (MySQL, PostgreSQL, SQL Server)
- APIs (REST, SOAP, GraphQL)
- Archivos planos (CSV, Excel, TXT)
- Fuentes no estructuradas (logs, emails, documentos)
- Sistemas legacy y ERP

## Consideraciones principales:

- Volumen de datos y estrategia de extracción
- Frecuencia (batch vs streaming)
- Gestión de credenciales y seguridad
- Validación inicial de datos extraídos
- Registro de metadatos (timestamps, proveniencia)

# Fase 2: Transformación de datos

## Tipos de transformaciones:

- **Limpieza**: Manejo de valores nulos, duplicados, outliers
- **Estandarización**: Formatos, unidades, codificaciones
- **Normalización/Desnormalización**: Estructura optimizada
- **Enriquecimiento**: Agregación de datos externos
- **Agregación**: Resúmenes estadísticos
- **Derivación**: Cálculo de nuevas variables

## Arquitecturas de transformación:

- ETL tradicional: Transformación en ambiente intermedio
- ELT moderno: Carga primero, transforma después
- Streaming: Transformación en tiempo real

# Fase 3: Carga de datos

## Destinos habituales:

- Data warehouses (Snowflake, Redshift, BigQuery)
- Data lakes (S3, Azure Data Lake)
- Bases de datos analíticas
- Plataformas BI (Tableau, Power BI)
- Sistemas operacionales

## Estrategias de carga:

- **Full load**: Reemplazo completo de los datos destino
- **Incremental**: Solo nuevos registros o cambios
- **Merge/Upsert**: Actualización e inserción combinadas
- **Histórica**: Preservación de todos los estados anteriores

# Herramientas y plataformas ETL

## Categorías principales:

- **Herramientas tradicionales**:
  - Informatica PowerCenter
  - IBM DataStage
  - Microsoft SSIS
- **Plataformas cloud-native**:
  - AWS Glue
  - Azure Data Factory
  - Google Cloud Dataflow
- **Frameworks de código abierto**:
  - Apache Airflow
  - Apache NiFi
  - dbt (data build tool)

# Desafíos comunes en procesos ETL

## Puntos críticos:

- **Gestión de excepciones**: Manejo de errores y casos extremos
- **Escalabilidad**: Adaptación a volúmenes crecientes
- **Latencia**: Reducción de tiempo entre extracción y disponibilidad
- **Gobernanza**: Control de acceso y cumplimiento normativo
- **Monitoreo**: Detección proactiva de problemas
- **Calidad de datos**: Validación continua

## Mejores prácticas:

- Diseño modular y reutilizable
- Documentación exhaustiva
- Testing automatizado
- Control de versiones
- Orquestación centralizada

# Tendencias actuales en ETL

## Evolución del paradigma:

- **ETL vs ELT**: Transformación tras la carga en entornos cloud
- **Datos en tiempo real**: Shift desde batch a streaming
- **DataOps**: Aplicación de principios DevOps a datos
- **ETL sin código/bajo código**: Democratización del desarrollo
- **ETL como código**: Infraestructura declarativa y versionada
- **Procesamiento federado**: Transformación donde residen los datos

## Integración con ecosistemas modernos:

- ETL para data science y machine learning
- Integración con data mesh y arquitecturas descentralizadas
- Compatibilidad con formatos y lagos de datos abiertos