2024

Guía de Entrenamiento: Preparación y Limpieza de Datos (ETL, Transformación de Datos)

FELIPE ALEJANDRO RODRIGUEZ LEMOS

SENA - CEAI

Guía de Entrenamiento: Preparación y Limpieza de Datos (ETL, Transformación de Datos)

1. Introducción al ETL

¿Qué es ETL?

ETL es un proceso en el flujo de trabajo de datos que implica tres etapas clave: Extracción (Extract), Transformación (Transform), y Carga (Load). Es fundamental para la integración y preparación de datos para análisis y reportes. Las etapas se definen de la siguiente manera:

Extracción (Extract): Consiste en extraer datos desde diversas fuentes, como bases de datos, archivos de texto, hojas de cálculo, y APIs. El objetivo es reunir los datos en bruto necesarios para su posterior procesamiento.

Transformación (Transform): En esta etapa, los datos extraídos se procesan para convertirlos en un formato adecuado para su análisis. Esto puede incluir limpiar datos, normalizar valores, transformar formatos, y agregar datos nuevos.

Carga (Load): Finalmente, los datos transformados se cargan en un sistema de almacenamiento, como un almacén de datos (Data Warehouse) o una base de datos, para su uso posterior.

Importancia del ETL en el análisis de datos

El proceso ETL es crucial porque garantiza que los datos sean precisos, completos y estén en un formato que facilite su análisis. Sin un proceso ETL adecuado, los datos pueden contener errores, duplicados o estar en formatos incompatibles, lo que puede llevar a análisis incorrectos y decisiones empresariales erróneas.

2. Extracción de Datos

Fuentes de Datos Comunes
Los datos pueden provenir de varias fuentes, tales como:
Bases de Datos Relacionales (MySQL, PostgreSQL, Oracle): Estructurados en tablas, accesibles a través de consultas SQL.
Bases de Datos NoSQL (MongoDB, Cassandra): Almacenan datos no estructurados o semi-estructurados.
Archivos: CSV, Excel, JSON, XML.
APIs: Servicios que permiten acceder a datos a través de solicitudes HTTP.
Web Scraping: Extracción de datos directamente desde sitios web.

Técnicas de Extracción

1. Extracción desde un archivo CSV usando Python

Los archivos CSV (Comma-Separated Values) son uno de los formatos más comunes para almacenar datos tabulares. A continuación, se muestra cómo extraer datos de un archivo CSV utilizando Python y la librería `pandas`.

Código python

```
import pandas as pd

# Leer datos desde un archivo CSV
ruta_csv = 'datos.csv'
datos = pd.read_csv(ruta_csv)

 # Mostrar las primeras filas del dataset
print(datos.head())
```

Explicación del Código:
- `import pandas as pd`: Importa la librería `pandas`, una herramienta poderosa para manipulación y análisis de datos.
- `pd.read_csv(ruta_csv)`: Utiliza la función `read_csv` para leer los datos del archivo CSV especificado en `ruta_csv` y cargarlos en un DataFrame de `pandas`.
- `datos.head()`: Muestra las primeras cinco filas del DataFrame para dar una vista rápida de los datos.

2. Extracción desde una base de datos SQL usando Python

Para conectarse a una base de datos SQL y extraer datos, se puede usar la librería `SQLAlchemy` junto con `pandas`. Aquí hay un ejemplo de cómo extraer datos de una tabla en una base de datos MySQL.

Código Python

```
from sqlalchemy import create_engine

# Crear una conexión a la base de datos
usuario = 'root'
contraseña = 'password'
host = 'localhost'
nombre_bd = 'mi_base_de_datos'

 # Crear cadena de conexión
cadena_conexion = f'mysql+pymysql://{usuario}:{contraseña}@{host}/{nombre_bd}'
motor = create_engine(cadena_conexion)

 # Ejecutar consulta SQL y cargar los datos en un DataFrame
consulta = 'SELECT * FROM nombre_tabla'
datos_sql = pd.read_sql(consulta, motor)

# Mostrar las primeras filas del dataset
print(datos_sql.head())
```

Explicación del Código:
- `create_engine`: Crea una conexión a la base de datos utilizando SQLAlchemy. La cadena de conexión incluye el nombre de usuario, la contraseña, el host y el nombre de la base de datos.
- `pd.read_sql`: Ejecuta una consulta SQL y carga los resultados en un DataFrame de `pandas`.

Ejercicio Práctico de Extracción

Descripción del Ejercicio: Extraer datos de un archivo CSV que contiene información de ventas y de una base de datos SQL con información de clientes. Los datos extraídos se utilizarán para análisis de ventas y perfil de clientes.

1. Extracción de Datos de Ventas desde un CSV:
   - Archivo CSV: `ventas.csv` con columnas `id_venta`, `fecha`, `producto`, `cantidad`, `precio`.

2. Extracción de Datos de Clientes desde una Base de Datos SQL:
   - Base de datos: `clientes_db`.
   - Tabla: `clientes` con columnas `id_cliente`, `nombre`, `email`, `fecha_registro`.

Código python

```
# Extracción desde CSV
ventas = pd.read_csv('ventas.csv')
print("Datos de Ventas:")
print(ventas.head())

# Extracción desde SQL
motor_sql = create_engine('mysql+pymysql://usuario:password@localhost/clientes_db')
clientes = pd.read_sql('SELECT * FROM clientes', motor_sql)
print("\nDatos de Clientes:")
print(clientes.head())
```

3. Limpieza de Datos

La limpieza de datos es crucial para asegurar que los datos sean precisos y utilizables. Los datos sin limpiar pueden contener errores, inconsistencias y valores faltantes que pueden afectar la calidad del análisis.

Identificación de Problemas Comunes
1. Datos Faltantes: Ausencia de valores en ciertas columnas.
2. Datos Duplicados: Registros repetidos que pueden distorsionar los resultados.
3. Inconsistencias: Diferencias en formatos de datos (e.g., fechas en distintos formatos).
4. Errores Tipográficos: Errores de escritura que afectan la calidad de los datos.

Cuando se prepara un dataset para análisis de datos y machine learning, es crucial revisar y corregir ciertos errores e inconsistencias. Aquí tienes una lista de aspectos clave a revisar:

Lista de Errores e Inconsistencias a Revisar en un Dataset

1. Valores Faltantes (Missing Values)
   - Errores: Celdas vacías o NaN.
   - Soluciones: Imputación de valores, eliminación de filas/columnas con muchos valores faltantes, o análisis de patrones en los datos faltantes.

2. Valores Duplicados
   - Errores: Filas o registros duplicados en el dataset.
   - Soluciones: Eliminar duplicados para evitar sesgos en el análisis.

3. Errores de Codificación
   - Errores: Problemas de codificación de caracteres que resultan en caracteres ilegibles o errores de lectura.
   - Soluciones: Detectar y ajustar la codificación de los archivos, usar codificaciones alternativas si es necesario.

4. Inconsistencias en el Formato de Datos
   - Errores: Diferentes formatos para fechas, números, o cadenas en el mismo campo.
   - Soluciones: Estandarizar formatos de fechas, números y otros datos.

5. Valores Atípicos (Outliers)
   - Errores: Valores que se desvían significativamente de la media o distribución esperada.
   - Soluciones: Identificar y analizar si deben ser eliminados, transformados, o si son representativos.

6. Errores de Entrada de Datos
   - Errores: Errores tipográficos, datos incorrectos o inconsistentes.
   - Soluciones: Validar y corregir los datos, normalizar entradas (por ejemplo, nombres en mayúsculas/minúsculas).

7. Datos No Relevantes o Irrelevantes
   - Errores: Columnas o filas que no aportan valor al análisis o modelo.
   - Soluciones: Eliminar o ignorar datos irrelevantes para mejorar la eficiencia del análisis.

8. Normalización y Estandarización
   - Errores: Datos en diferentes escalas o unidades.
   - Soluciones: Normalizar o estandarizar datos para asegurar que todos los atributos contribuyan de manera equitativa al análisis.

9. Errores en Etiquetas o Categorías
   - Errores: Categorías incorrectas, inconsistentes o mal codificadas.
   - Soluciones: Revisar y corregir categorías y etiquetas para asegurar consistencia.

10. Valores Anómalos en Datos Categóricos
   - Errores: Valores categóricos inesperados o faltantes.
   - Soluciones: Revisar y ajustar valores categóricos para que sean consistentes y completos.

11. Errores de Redondeo y Precisión
   - Errores: Problemas con la precisión y redondeo de números.
   - Soluciones: Ajustar la precisión y el redondeo para que se alineen con los requisitos del análisis.

12. Desbalanceo de Clases (Class Imbalance)
   - Errores: Distribución desigual de clases en problemas de clasificación.
   - Soluciones: Técnicas de balanceo como sobremuestreo o submuestreo.

13. Datos Temporales Inconsistentes
   - Errores: Inconsistencias en las fechas y tiempos.
   - Soluciones: Verificar y corregir registros de tiempo y fecha, asegurando coherencia.

14. Errores en Datos Geoespaciales
   - Errores: Datos geoespaciales incorrectos o inconsistentes.
   - Soluciones: Validar y corregir coordenadas y ubicaciones geográficas.

15. Consistencia entre Dataset y Metadatos
   - Errores: Inconsistencias entre los datos y la descripción en los metadatos.
   - Soluciones: Asegurar que los metadatos coincidan con la estructura y contenido real del dataset.

Técnicas de Limpieza

1. Eliminación de Duplicados

Código python

```
# Ejemplo de eliminación de duplicados en un DataFrame
datos = pd.DataFrame({
    'id': [1, 2, 2, 3, 4],
    'nombre': ['Ana', 'Juan', 'Juan', 'María', 'Luis']
})

print("Antes de eliminar duplicados:")
print(datos)

# Eliminar duplicados basados en la columna 'id'
datos_sin_duplicados = datos.drop_duplicates(subset='id')

print("\nDespués de eliminar duplicados:")
print(datos_sin_duplicados)
```

Explicación del Código:
- `drop_duplicates`: Elimina filas duplicadas en el DataFrame. En este ejemplo, se eliminan duplicados basados en la columna 'id'.

2. Manejo de Valores Faltantes

Código python

```
# Ejemplo de manejo de valores faltantes
datos = pd.DataFrame({
    'nombre': ['Ana', 'Juan', None, 'María', 'Luis'],
    'edad': [23, None, 30, 22, None]
})

print("Datos originales con valores faltantes:")
print(datos)

# Rellenar valores faltantes en la columna 'edad' con la media de la columna
datos['edad'].fillna(datos['edad'].mean(), inplace=True)

# Eliminar filas con valores faltantes en la columna 'nombre'
datos.dropna(subset=['nombre'], inplace=True)

print("\nDatos después de manejar valores faltantes:")
print(datos)
```

Explicación del Código:
- `fillna`: Rellena los valores faltantes con un valor específico. En este caso, la media de la columna 'edad'.
- `dropna`: Elimina las filas con valores faltantes en una columna específica.

Ejercicio Práctico de Limpieza

Descripción del Ejercicio: Limpieza de un dataset de ventas para eliminar duplicados y manejar valores faltantes.

1. Eliminar duplicados en el dataset de ventas basado en `id_venta`.
2. Rellenar valores faltantes en la columna `cantidad` con el valor medio.

3. Eliminar filas con valores faltantes en la columna `producto`.

Código Python

```
# Supongamos un dataset de ventas con duplicados y valores faltantes
ventas = pd.DataFrame({
    'id_venta': [1, 2, 2, 3, 4],
    'producto': ['A', 'B', None, 'D', 'E'],
    'cantidad': [10, None, 5, 2, 3],
    'precio': [100, 150, 150, 200, 250]
})

print("Datos de Ventas Originales:")
print(ventas)

# Eliminar duplicados basados en 'id_venta'
ventas = ventas.drop_duplicates(subset='id_venta')

# Rellenar valores faltantes en 'cantidad' con la media
ventas['cantidad'].fillna(ventas['cantidad'].mean(), inplace=True)

# Eliminar filas con valores faltantes en 'producto'
ventas.dropna(subset=['

producto'], inplace=True)

print("\nDatos de Ventas después de limpieza:")
print(ventas)
```

4. Transformación de Datos

La transformación implica modificar los datos para ajustarlos a un formato que facilite el análisis o para integrarlos en un sistema de almacenamiento específico.

Tipos de Transformación

1. Transformaciones Básicas: Incluyen cambios sencillos como filtrar, seleccionar columnas específicas, o convertir tipos de datos.
2. Transformaciones Avanzadas: Involucran operaciones más complejas como agregar datos (suma, promedio), pivoteo de datos, y creación de nuevas columnas basadas en cálculos o condiciones.

Técnicas de Transformación

1. Filtrado y Selección de Columnas

Código python

```
# Supongamos un dataset de ventas
ventas = pd.DataFrame({
    'id_venta': [1, 2, 3, 4],
    'producto': ['A', 'B', 'C', 'D'],
    'cantidad': [10, 20, 15, 25],
    'precio': [100, 150, 120, 200]
})
# Filtrar ventas con 'cantidad' mayor a 15
ventas_filtradas = ventas[ventas['cantidad'] > 15]

# Seleccionar columnas 'producto' y 'cantidad'
ventas_seleccionadas = ventas[['producto', 'cantidad']]

print("Ventas Filtradas:")
print(ventas_filtradas)
print("\nVentas Seleccionadas:")
print(ventas_seleccionadas)
```

Explicación del Código:

- `ventas[ventas['cantidad'] > 15]`: Filtra las filas donde la columna 'cantidad' es mayor a 15.
- `ventas[['producto', 'cantidad']]`: Selecciona solo las columnas 'producto' y 'cantidad' del DataFrame.

2. Agregación de Datos

Código Python

```
# Dataset de ventas
ventas = pd.DataFrame({
    'producto': ['A', 'B', 'A', 'C', 'B', 'A'],
    'cantidad': [10, 5, 8, 7, 12, 6]
})

# Calcular el total de ventas por producto
ventas_agrupadas = ventas.groupby('producto').sum()

print("Total de Ventas por Producto:")
print(ventas_agrupadas)
```

Explicación del Código:

- `groupby('producto')`: Agrupa los datos por la columna 'producto'.
- `sum()`: Suma los valores de las columnas numéricas dentro de cada grupo.

Ejercicio Práctico de Transformación

Descripción del Ejercicio: Transformar un dataset de ventas para analizar la cantidad total vendida por producto y el precio promedio por producto.

1. Calcular la cantidad total vendida por cada producto.
2. Calcular el precio promedio por producto.

Código Python

```
# Dataset de ventas con cantidades y precios
ventas = pd.DataFrame({
    'producto': ['A', 'B', 'A', 'C', 'B', 'A'],
    'cantidad': [10, 5, 8, 7, 12, 6],
    'precio': [100, 150, 100, 200, 150, 100]
})

 # Calcular la cantidad total vendida por producto
total_cantidad = ventas.groupby('producto')['cantidad'].sum()

# Calcular el precio promedio por producto
precio_promedio = ventas.groupby('producto')['precio'].mean()

print("Cantidad Total Vendida por Producto:")
print(total_cantidad)
print("\nPrecio Promedio por Producto:")
print(precio_promedio)
```

5. Carga de Datos

Después de extraer, limpiar y transformar los datos, el paso final es cargarlos en un sistema de almacenamiento para su uso posterior.

Opciones de Almacenamiento
1. Bases de Datos Relacionales: Almacenan datos en tablas con relaciones predefinidas (e.g., MySQL, PostgreSQL).
2. Bases de Datos NoSQL: Manejan grandes volúmenes de datos no estructurados (e.g., MongoDB).
3. Data Warehouses: Optimizados para almacenamiento y consulta de grandes volúmenes de datos (e.g., Amazon Redshift, Google BigQuery).

Técnicas de Carga

1. Carga de Datos en una Base de Datos SQL usando Python

Código python

```
from sqlalchemy import create_engine

# Supongamos un dataset limpio y transformado
datos = pd.DataFrame({
    'id_venta': [1, 2, 3, 4],
    'producto': ['A', 'B', 'C', 'D'],
    'cantidad': [10, 20, 15, 25],
    'precio': [100, 150, 120, 200]
})

# Crear conexión a la base de datos
motor_sql = create_engine('mysql+pymysql://usuario:password@localhost/mi_base_de_datos')

# Cargar datos en una tabla llamada 'ventas_procesadas'
datos.to_sql('ventas_procesadas', motor_sql, if_exists='replace', index=False)

print("Datos cargados en la base de datos 'mi_base_de_datos', tabla 'ventas_procesadas'")
```

Explicación del Código:
- `to_sql`: Método de `pandas` para cargar datos desde un DataFrame a una tabla SQL. El parámetro `if_exists='replace'` indica que se reemplace la tabla si ya existe.

6. Herramientas ETL y Automatización

Plataformas ETL Populares
Apache NiFi: Plataforma para automatización de flujos de trabajo de datos.
Talend: Software de integración de datos con una interfaz visual para diseño de procesos ETL.
Alteryx: Herramienta de análisis de datos con capacidades de ETL.

Uso de Python para ETL
Python es una herramienta poderosa para implementar procesos ETL debido a su flexibilidad y amplia gama de librerías como `pandas`, `SQLAlchemy`, y `requests`.

Ejercicio Práctico de Automatización

Descripción del Ejercicio: Crear un script Python para automatizar un flujo ETL simple que extrae datos de un archivo CSV, los limpia, los transforma y los carga en una base de datos SQL.

Código python

```
import pandas as pd
from sqlalchemy import create_engine

# Paso 1: Extracción - Leer datos desde un archivo CSV
ventas = pd.read_csv('ventas.csv')

# Paso 2: Limpieza - Eliminar duplicados y manejar valores faltantes
ventas = ventas.drop_duplicates(subset='id_venta')
ventas['cantidad'].fillna(ventas['cantidad'].mean(), inplace=True)
ventas.dropna(subset=['producto'], inplace=True)

# Paso 3: Transformación - Calcular la cantidad total vendida por producto
ventas_agrupadas = ventas.groupby('producto')['cantidad'].sum().reset_index()

# Paso 4: Carga - Cargar datos en una base de datos SQL
motor_sql =
create_engine('mysql+pymysql://usuario:password@localhost/mi_base_de_datos')
ventas_agrupadas.to_sql('ventas_resumen', motor_sql, if_exists='replace',
index=False)
print("Flujo ETL completado y datos cargados en la tabla 'ventas_resumen'")
```

7. Casos de Uso en la Industria

Análisis de Clientes: Uso de ETL para consolidar y analizar datos de clientes provenientes de múltiples fuentes, permitiendo a las empresas entender mejor su base de clientes y personalizar sus ofertas.
Integración de Datos de Ventas: Empresas utilizan ETL para combinar datos de ventas de diferentes sistemas (e.g., puntos de venta, e-commerce) para tener una visión integral de las ventas y tomar decisiones estratégicas.
Monitoreo de Sensores en IoT: ETL es utilizado para procesar y almacenar datos de sensores, facilitando el análisis de patrones y el mantenimiento predictivo en industrias como manufactura y salud.

8. Mejores Prácticas en ETL

1. Documentación y Mantenimiento: Documentar los procesos ETL para facilitar su mantenimiento y actualización.
2. Optimización del Rendimiento: Usar técnicas de indexación, partición de datos, y optimización de consultas SQL para mejorar la eficiencia de los procesos ETL.
3. Control de Calidad de Datos: Implementar validaciones y verificaciones de calidad para asegurar la precisión y consistencia de los datos a lo largo del proceso ETL.