SAP Data Cleaning Project

Summary and goals

Project Objective
Clean and standardize SAP data to ensure quality, consistency, and accuracy for analysis and decision-making.

Project Steps

1. Load Data & Libraries

- Import raw SAP datasets and required Python libraries.

2. Clean Each DataFrame

- Remove unnecessary columns
- Handle missing values
- Standardize column names
- Filter relevant data

3. DataFrame Interactions

- Add last maintenance year
- Calculate equipment age
- Identify decommissioning requests

5. Save Cleaned Data

- Export structured DataFrames for easy re-importing

Modular Functions
Cleaning functions are in `data_cleaning_utils.py` for reusability.

This ensures an efficient, reproducible cleaning process for accurate insights.

Part 1: Load DataFrames and Import Libraries

In [57]:
```
%load_ext autoreload
%autoreload 2

import importlib
import pandas as pd
import data_cleaning_utils as f
import os
```

```
from tabulate import tabulate
importlib.reload(f)
import numpy as  np
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [58]:
```
# Define the folder where the files are located
folder_name = "DF_original"

# List of base names for the DataFrames
df_names = {
    "df_equipment",
    "df_orders",
    "df_equipment_decomission_request",
    "df_storage",
    "df_dolar"
}

# Dynamically load the DataFrames and assign them as individual variables

for name in df_names:
    file_path = os.path.join("../",folder_name, f"{name}.csv")
    globals()[name] = pd.read_csv(file_path)

print("File loading completed.")
```

```
File loading completed.
```

Part 2: Cleaning Each DataFrame Separately

Each DataFrame is cleaned separately by:

- Running the basic_data_checks function to identify and handle missing values, duplicates, and constant columns.
- Dropping unnecessary columns to keep only relevant data.
- Standardizing column names for consistency across DataFrames.
- Performing additional transformations as needed to prepare the data for analysis.

Cleaning Storage DataFrame

This DataFrame contains key information related to storage locations in hospitals, including:

- Center Name: Full name of the hospital.
- Hospital Name: A simplified hospital name used for merging with other DataFrames.
- Storage Data: Storage unit number.
- Responsible Hospital: The individual assigned to manage the storage unit.

These fields are cleaned and standardized to ensure compatibility with other datasets during merging.

In [59]:
```
df_storage.head()
```

Out[59]:

	Responsible	Storage	Hospital
0	R3	A3	H2
1	R3	A3	H42
2	R7	A7	H6
3	R1	A1	H31
4	R1	A1	H22

In [60]:
```
df_storage = f.basic_data_checks(df_storage)
```

```
Initial DataFrame shape: (52, 3)
Total missing values in the DataFrame: 16
Missing values per column:
Storage     15
Hospital     1
dtype: int64
No duplicate rows found in the DataFrame.

Data types of columns:
Responsible    object
Storage        object
Hospital       object
dtype: object
```

Cleaning Decommission Request DataFrame

This DataFrame contains information about equipment for which a formal decommissioning request has been submitted.
By analyzing this data, we can verify whether any of these decommissioned devices are still present in the active equipment database.

Although the dataset contains extensive information, we are specifically interested in the following fields:

- **Equipment_ID:** Unique identifier for each piece of equipment (renamed from `Title` for consistency).
- **Created:** Date when the decommissioning request was submitted.
- **Modified:** Date when the equipment's status was last updated.
- **Status:** Current status of the equipment in the decommissioning process.

In [61]:
```
df_equipment_decomission_request.head(2)
```

Out[61]:

	Equipment_ID	Created	Modified	Status
0	NaN	2022-11-09 10:00:31	2022-11-09 11:57:41	Finalizado
1	NaN	2022-11-09 10:04:27	2022-12-28 16:27:09	Finalizado

We observed that some records in the **'Status'** column have missing values.
This may indicate that the decommissioning request has not yet been reviewed or initiated.

Cleaning Orders DataFrame

This DataFrame contains information about maintenance and service orders assigned to medical equipment.
When a **healthcare professional** (e.g., doctor or nurse) reports an issue requiring technical assistance, an **ICOR** order is generated.
Additional orders may be created for **preventive maintenance (IPRV)** and **third-party service agreements (IABN).**

Relevant Columns:

- **Order:** Unique identifier for each service request.
- **Class:** Type of order (ICOR, IPRV, IABN).
- **Equipment_ID:** Unique identifier of the equipment.
- **Execution Date → Year:** Original execution date, converted to **year format**.
- **Cost:** Cost of the order in USD.

Filtering Criteria:

We are only interested in:

- Orders that are linked to a valid **equipment ID**.
- Orders where the **technical location** (`Ubicac.técnica`) starts with "IC".

This filtering ensures that only relevant maintenance records are considered in the analysis.

In [62]:
```
#drop na
df_orders = df_orders.dropna(subset=['Equipment_ID'])
```

In [63]:
```
df_dolar.head(2)
```

Out[63]:

	Day	Value
0	2011-01-03	4.11
1	2011-01-04	4.12

In [64]:
```
## Use the function to add the "Dollar price of the day" column
df_orders = f.add_dollar_day_price(df_orders, df_dolar,date_col_main="Year",
```

```
                               date_col_dolar="Day",
                               price_col_dolar="Value",
                               new_col='Dollar_day_price')
```

In [65]:
```
df_orders.columns
```

Out[65]:
```
Index(['Order', 'Class', 'Year', 'Equipment_ID', 'Cost', 'Day', 'Value',
       'Dollar_day_price'],
      dtype='object')
```

In [66]:
```
df_orders.head(2)
```

Out[66]:

	Order	Class	Year	Equipment_ID	Cost	Day	Value	Dollar_day_price
0	9000002	ICOR	2008-01-12	T11-VS-SVYS	0.000	2011-01-02	4.13	3.0
1	9000001	ICOR	2008-01-17	T84-UCU	15957.297	2011-01-02	4.13	3.0

In [67]:
```
df_orders['Cost_in_dollar'] = (df_orders['Cost'] / df_orders['Dollar_day_price']).r
```

In [68]:
```
df_orders.drop(columns=['Order'], inplace=True)
```

In [69]:
```
df_orders.head(2)
```

Out[69]:

	Class	Year	Equipment_ID	Cost	Day	Value	Dollar_day_price	Cost_in_dollar
0	ICOR	2008-01-12	T11-VS-SVYS	0.000	2011-01-02	4.13	3.0	0.0
1	ICOR	2008-01-17	T84-UCU	15957.297	2011-01-02	4.13	3.0	5319.1

In [70]:
```
## Drop rows with missing equipment values
# Ensuring only orders with assigned equipment are kept
#df_orders.dropna(subset=["Equipment_ID"], inplace=True)

## Convert date to year format
# Ensuring 'Year' column contains only the year extracted from the date
df_orders["Year"] = pd.to_datetime(df_orders["Year"], errors="coerce").dt.year

## Perform basic data checks
# Running the function to check for missing values, duplicates, and data consistenc
df_orders = f.basic_data_checks(df_orders, remove_duplicates=True)
```

```
Initial DataFrame shape: (106609, 8)
No missing values found in the DataFrame.
Found 3648 duplicate rows in the DataFrame.
Duplicates removed.
New DataFrame shape: (102961, 8)

Data types of columns:
Class                        object
Year                          int32
Equipment_ID                 object
Cost                        float64
Day                  datetime64[ns]
Value                       float64
Dollar_day_price            float64
Cost_in_dollar              float64
dtype: object
```

Now I want to add a column for the case that the order needed a repair (Cost > 0) and those that didnt

In [71]:
```
# Add a new column "OrderWithRepair" that is True if "Costo" > 0, otherwise False
df_orders_icor = df_orders[df_orders["Class"] == "ICOR"]
df_orders["OrderWithRepair"] = df_orders_icor["Cost"] > 0
```

Cleaning equipment DataFrame

This DataFrame contains detailed information about medical equipment. The following fields have been selected and renamed to ensure consistency across datasets: **Relevant Columns:**

- **Equipment_ID:** Unique identifier for each equipment item.
- **Description:** General description of the equipment.
- **Type:** Equipment category.
- **Subtype:** Subcategory within the equipment type.
- **Manufacturer:** Name of the equipment manufacturer.
- **Model:** Equipment model.
- **Serial_number:** Equipment serial number.
- **ABC:** Inventory classification based on management principles.
- **Purchase_value:** Original purchase value of the equipment.
- **UT:** Internal code used to derive the responsible entity (EC or IC), hospital, and department.
- **Purchase_date → Purchase_year:** The year the equipment was acquired. If missing, it is set to **2005**, as the acquisition occurred before the system's implementation in 2008.
- **UT_info:** Additional details extracted from the UT code.
- **Device_status:** Operational status of the equipment.
- **System_status:** System-defined status assigned to the equipment.

The cleaning process standardizes formats, fills missing values (such as purchase year), and extracts relevant fields from **UT** to support further analysis.

In [72]:
```
df_equipment.head(1)
```

Out[72]:

	Equipment_ID	Type	Subtype	Serial_number	ABC	Purchase_value	Purchase_date	Dev
0	St12-ITYWKVTK	NaN	St12	ITYWKVTK	NaN	0.0	NaN	

In [73]:
```
## Convert date columns to datetime format
# Ensuring proper date format and handling errors by coercing invalid values to NaT
df_equipment["Purchase_date"] = pd.to_datetime(df_equipment["Purchase_date"], error

## Extract year from dates
# Replacing missing purchase years with 2005, as these items were acquired before t
df_equipment["Purchase_year"] = df_equipment["Purchase_date"].dt.year.fillna(2005).

#Those with year prior to 2005 lets put it as 2005 as well

df_equipment["Purchase_year"] = df_equipment["Purchase_date"].dt.year.fillna(2005).



## Drop original date columns
# Removing full date columns as only the year is needed for analysis
df_equipment.drop(columns=["Purchase_date"], inplace=True)
```

In [74]:
```
df_equipment = f.basic_data_checks(df_equipment)
```

```
Initial DataFrame shape: (14591, 13)
Total missing values in the DataFrame: 1353
Missing values per column:
Type                2
Subtype            39
Serial_number      11
ABC                44
Responsible       227
Hospital          248
Department        675
Manufacturer        2
Model             105
dtype: int64
No duplicate rows found in the DataFrame.

Data types of columns:
Equipment_ID       object
Type               object
Subtype            object
Serial_number      object
ABC                object
Purchase_value    float64
Device_status      object
Responsible        object
Hospital           object
Department         object
Manufacturer       object
Model              object
Purchase_year       int32
dtype: object
```

During the data cleaning process, we identified several missing values across different columns.
This highlights **inconsistencies and gaps in the database**, indicating issues in how equipment data is recorded and maintained.

Critical Missing Data: Hospital Field

One of the most relevant missing data cases is in the **Hospital** field.
Every piece of equipment should have a **Hospital** assigned, as it indicates its location and responsibility.

After consultation, we confirmed that **equipment marked for system removal had their Hospital removed**.
This suggests that missing Hospital values may correspond to equipment that was supposed to be decommissioned but still appears in the database.

This inconsistency should be addressed to ensure accurate tracking and reporting.

In [75]:
```
df_equipment_missing_hospital = df_equipment[df_equipment['Hospital'].isnull()]
```

Incorrect Purchase Year Detected

During data validation, we identified an equipment entry with a **purchase year of 2201**, which is clearly an error.
To correct this, we will **replace it with 2017**, a reasonable estimate, to avoid inconsistencies in the analysis.

In [76]:
```
df_equipment["Purchase_year"].value_counts().sort_index
```

Out[76]:
```
<bound method Series.sort_index of Purchase_year
2012    1855
2008    1721
2005    1250
2014    1248
2013    1215
2017     949
2018     897
2015     852
2024     770
2016     626
2011     608
2019     512
2023     501
2020     464
2022     270
2010     251
2021     249
2009     213
2025      79
2007      52
2000       7
2006       1
2201       1
Name: count, dtype: int64>
```

In [77]:
```
df_equipment[df_equipment["Purchase_year"] == 2201]
```

Out[77]:

	Equipment_ID	Type	Subtype	Serial_number	ABC	Purchase_value	Device_status	F
4807	T115-UWVWVF	T115	St23	UWVWVF	C	495.0	Normal	

In [78]:
```
df_equipment.loc[df_equipment["Purchase_year"] == 2201, "Purchase_year"] = 2017
```

Add Purchase_value to those that are 0

In [79]:
```
promedio_por_Model = df_equipment[df_equipment['Purchase_value'] > 0].groupby('Mode

    # Fill in 0 values with the corresponding average
df_equipment['Purchase_value'] = df_equipment.apply(
```

```
        lambda row: promedio_por_Model[row['Model']] if row['Purchase_value'] == 0
        axis=1
    )
```

Part 3: Interaction Between Tables

In this section, we enrich the **Equipment** and **Orders** DataFrames by integrating key information from other tables to support deeper analysis.

Steps:

1. **Add the Year of the Last Order**
    - Extracted from the **Orders** DataFrame.
    - Enables tracking of the most recent maintenance activity or reported issue for each equipment item.
    - If NAN, we assign 2000
2. **Calculate Equipment Age at the Time of the Order**
    - Computed as the difference between the **order year** and the **purchase year** (`Purchase_year`).
    - Useful for identifying maintenance patterns based on equipment age.
3. **Flag Equipment with Decommissioning Requests**
    - A new column is added to the **Equipment** DataFrame.
    - Indicates whether the equipment appears in the **Decommission Request** table.

These enhancements improve data completeness and provide valuable insights for equipment lifecycle and maintenance analysis.

Add Column: Year of Last Order

In [80]:
```
df_equipment.columns
```

Out[80]:
```
Index(['Equipment_ID', 'Type', 'Subtype', 'Serial_number', 'ABC',
       'Purchase_value', 'Device_status', 'Responsible', 'Hospital',
       'Department', 'Manufacturer', 'Model', 'Purchase_year'],
      dtype='object')
```

In [81]:
```
## Get the last recorded order year for each equipment
# First, compute the maximum year of order per equipment in the 'Ordenes' DataFrame
df_orders["Last_order_year"] = df_orders.groupby("Equipment_ID")["Year"].transform(

# Create a summarized DataFrame containing only the latest order year per equipment
df_ultima_orden = df_orders.groupby("Equipment_ID")["Last_order_year"].max().reset_

## Merge the last order year into the Equipos DataFrame
# Performing a left join to retain all equipment records, even those without an ord
```

```python
df_equipment = df_equipment.merge(df_ultima_orden, on="Equipment_ID", how="left")

## Handle missing values in 'Last_order_year'
# If an equipment has no recorded orders, replace NaN with "No orders"
df_equipment["Last_order_year"] = df_equipment["Last_order_year"].fillna(2000).asty

# Print verification sample
print(df_equipment[["Equipment_ID", "Last_order_year"]].head())

## Convert the 'Last_order_year' column to a numeric type
# This ensures consistency, forcing invalid entries to NaN for further handling
df_equipment["Last_order_year"] = pd.to_numeric(df_equipment["Last_order_year"], er
```

```
   Equipment_ID  Last_order_year
0  St12-ITYWKVTK             2024
1       St9-UUWW             2000
2       T8-GFVWU             2000
3       T10-YYTF             2000
4   T10-PIUIVFUW             2024
```

In [82]:
```python
df_equipment.columns
```

Out[82]:
```
Index(['Equipment_ID', 'Type', 'Subtype', 'Serial_number', 'ABC',
       'Purchase_value', 'Device_status', 'Responsible', 'Hospital',
       'Department', 'Manufacturer', 'Model', 'Purchase_year',
       'Last_order_year'],
      dtype='object')
```

Add column: Equipment Age at the Time of the Order

In [83]:
```python
## Create copies of original DataFrames to avoid modifying them directly
df_orders2 = df_orders.copy()
df_equipment2 = df_equipment.copy()

## Merge 'Purchase_year' from df_equipment into df_orders
# This allows us to calculate the equipment's age at the time of the order
df_orders2 = df_orders2.merge(df_equipment2[['Equipment_ID', 'Purchase_year']], on=

## Calculate equipment age at the time of the order
# Subtracting the purchase year from the order year
df_orders2['Equipment_age'] = df_orders2['Year'] - df_orders2['Purchase_year']

## Display a preview of the updated DataFrame
df_orders2.head(3)
```

Out[83]:

	Class	Year	Equipment_ID	Cost	Day	Value	Dollar_day_price	Cost_in_dollar	O
0	ICOR	2008	T11-VS-SVYS	0.000	2011-01-02	4.13	3.0	0.0	
1	ICOR	2008	T84-UCU	15957.297	2011-01-02	4.13	3.0	5319.1	
2	ICOR	2008	T52-UKV	0.000	2011-01-02	4.13	3.0	0.0	

We observed that some equipment records in the **orders dataset** do not have a purchase year.
However, we already assigned a purchase year to all active equipment in the **Equipos** DataFrame.

This discrepancy occurs because maintenance orders are not only for **active equipment** but also for those that have already been **decommissioned**.

To address this, we will separate the datasets to distinguish between **orders for active equipment** and **orders for decommissioned equipment**.

In [84]:

```
## Separate orders for active and decommissioned equipment

# Keep only orders where the equipment has a recorded purchase year (active equipme
df_orders = df_orders2[df_orders2['Purchase_year'].notna()]

# Create a separate DataFrame for orders linked to decommissioned equipment (missin
df_orders_decommissioned = df_orders2[df_orders2['Purchase_year'].isna()]

df_orders_all = df_orders2.copy()
```

Lets do analize this new column

In [85]:

```
df_orders["Equipment_age"].describe()
```

Out[85]:

```
count    102961.000000
mean          6.224658
std           4.103337
min         -11.000000
25%           3.000000
50%           6.000000
75%           9.000000
max          24.000000
Name: Equipment_age, dtype: float64
```

If its -1 it might be because of how we split the data, lets change those to 0 and eliminate the rest of the negative values

In [86]:
```
df_orders.loc[df_orders["Equipment_age"] == -1, "Equipment_age"] = 0
df_orders_all.loc[df_orders_all["Equipment_age"] == -1, "Equipment_age"] = 0
df_orders = df_orders[df_orders["Equipment_age"] >= 0]
df_orders_all = df_orders_all[df_orders_all["Equipment_age"] >= 0]
df_orders["Equipment_age"].describe()
```

Out[86]:
```
count    102860.000000
mean          6.235962
std           4.089587
min           0.000000
25%           3.000000
50%           6.000000
75%           9.000000
max          24.000000
Name: Equipment_age, dtype: float64
```

Add Column: Equipment with Decommissioning Request

In [88]:
```
## Merge to Identify Equipment with a Decommissioning Request
# Performing a left merge between 'df_equipment' and 'df_equipment_decomission_requ
# to check which equipment appears in the decommissioned equipment dataset.
df_equipment = df_equipment.merge(df_equipment_decomission_request[['Equipment_ID']
                              on='Equipment_ID',
                              how='left',
                              indicator=True)

## Create a New Column to Indicate Decommissioning Request
# Mapping the '_merge' column:
# - "both" → Equipment exists in both datasets (has a decommissioning request) → 1
# - "left_only" → Equipment exists only in 'df_equipment' (no decommissioning reque
df_equipment["Decommission_request"] = df_equipment["_merge"].map({"both": 1, "left

## Remove the '_merge' Column
# This column was generated by the merge and is no longer needed
df_equipment.drop(columns=["_merge"], inplace=True)

## Create a Subset of Equipment with a Decommissioning Request
# Filtering only the equipment that has a decommissioning request (Decommission_req
df_equipment_with_decommission_request = df_equipment[df_equipment["Decommission_re

## Check the Shape of the New DataFrame
df_equipment_with_decommission_request.shape

## Verify the Output
# Uncomment to inspect the first few rows with the new 'Decommission_request' colum
print(df_equipment[["Equipment_ID", "Decommission_request"]].head())
```

```
     Equipment_ID  Decommission_request
0  St12-ITYWKVTK                   0.0
1       St9-UUWW                   0.0
2       T8-GFVWU                   0.0
3       T10-YYTF                   0.0
4   T10-PIUIVFUW                   0.0
```

In [ ]:

Part 5: Save the DataFrames and Import Process

In this section, we finalize the cleaning and standardization process by **saving the cleaned DataFrames** for future analysis.
The saved files will be structured and formatted to ensure **easy re-importing** without requiring additional preprocessing.

In [109…

```
import os

## Create a folder for storing cleaned DataFrames
# Ensures the directory 'DF_cleaned' exists before saving files
folder_name = "DF_cleaned"

## Dictionary containing the DataFrames to be exported
dfs = {
    "df_equipment": df_equipment,
    "df_orders": df_orders,
    "df_orders_decommissioned": df_orders_decommissioned,
    "df_orders_all": df_orders_all,
    "df_equipment_decomission_request": df_equipment_decomission_request,
    "df_storage": df_storage,
    "df_equipment_missing_hospital": df_equipment_missing_hospital,
}

## Save each DataFrame as a CSV file inside the specified folder
for name, df in dfs.items():
    file_path = os.path.join("..",folder_name, f"{name}.csv")
    df.to_csv(file_path, index=False)

print("Export process completed successfully.")
```

```
Export process completed successfully.
```