TI3002C_Etapa 4.1-Base Datos_1

April 9, 2024

1 4.1 Base de datos para el reto

Pandalytics - Equipo 1 * **A00832444** | Andrea Garza * **A01197991** | Hiram Maximiliano Muñoz Ramírez * **A00517124** | Erick Orlando Hernández Vallejo * **A01197655** | Raúl Isaí Murillo Alemán * **A01235692** | David Gerardo Martíne Hidrogo

Se definió una *pipeline* de procesamiento de datos en un *script* de Python independiente, con el cuál se tiene un proceso replicable para llegar a una base de datos limpia. Esta *pipeline* consta de 6 diferentes pasos:

- Importación de datos
- Concatenación de resúmenes de producción
- Separación de analisis de líneas de producción y sus defectos
- Selección y renombramiento de columnas
- Eliminación de columnas adicionales
- Conversión a formato *Feather*.

A continuación se muestra un resumen de todo el proceso de la *pipeline* de datos:

1.1 1.1 Importación de datos

La importación de datos consiste en importar todos los datos relevantes de los archivos de excel proveídos por Ternium. Hay un total de 12 archivos diferentes.Cada uno de estos archivos contiene varias *worksheets* dentro. Para la mayoría, solamente necesitamos importar una de estas. Solamente el archivo de Análisis de consumo de pintura requirió importar mas *worksheets*:

- Pinturas y revestidos
  - Pintado 1 UNI Agosto
- Análisis de consumo de pintura
  - Base de datos por pintura
  - Metros
  - Rendimientos INDU
- 10 archivos de resúmenes de producción
  - Resúmen producción

1.2 1.2 Concatenación de resúmenes de producción

Para unificar toda la información de resúmenes de producción, concatenamos todos estas tablas, para así acabar con una base de datos más cohesiva y extensiva con la cual trabajar. Se ignoro el índice original de las 10 tablas, ya que realmente no tenían ningún significado.

1.3 1.3 Separación de analisis de líneas de producción y sus defectos

Los resúmenes de producción contienen un grupo de columnas que se repiten 5 veces. Estás columnas representan hasta 5 posibles defectos en cada línea de producción. Por lo tanto, estas columnas representan una relación uno a muchos entre una línea de producción y sus posibles defectos. Se definió una función que separa estos grupos de 5 columnas hacia una nueva tabla de defectos, con cada defecto recibiendo una referencia hacia su respectiva línea de producción. Finalmente, se eliminaron las columnas de defectos de la tabla original, ya que ahora se pueden referenciar por medio de una operación de merge con la nueva tabla de defectos.

1.4 1.4 Selección y renombramiento de columnas

Se seleccionaron columnas relevantes de cada base de datos para dejar en la base de datos. Estas columnas fueron renombradas para mejor documentar su propósito. A continuación se tiene una lista de las columnas renombradas para cada base de datos, así como las razones por las cuales las seleccionamos.

1.4.1 1.4.1 Pinturas y revestidos

- **'Denominación objeto' por 'production_line':** Posible identificador para cada línea de producción
- **'Material' por 'paint_id'**: Pintura usada, posible identificador para pinturas
- **'Texto breve de material' por 'paint_name'**: Nombre de la pintura usada, posible identificador para pinturas
- **'Valor var.' por 'monetary_value_usd'**: Valor monetario en dólares, útil para calcular costos
- **'Ctd.total reg.' por 'total_liters_used'**: Litros usados, dato clave
- **'Planta' por 'production_plant'**: Identificación de planta, para filtrar datos
- **'Usuario' por 'user'**: Usuario a cargo del proceso de pintado
- **'Proveedor' por 'supplier'**: Proveedor, para análisis de datos agrupados
- **'Registrado' por 'date'**: Fecha de los datos, dato clave
- **'Hora' por 'hour'**: Hora de los datos, dato clave
- **'Precio' por 'price_per_liter'**: Precio de cada litro, útil para calcular costos
- **'Linea' por 'line'**: Línea al que pertenece el proceso

1.4.2 1.4.2 Análisis de consumo de pintura

- **'Linea' por 'production_line'**: Línea de producción
- **'Mes' por 'month'**: Mes del consumo de pintura, dato clave
- **'Mes num' por 'month_number'**: Número del mes
- **'Pintura' por 'paint'**: Pintura, posible identificador para pinturas
- **'Real' por 'real_consumption'**: Consumo real, dato clave
- **'Teo' por 'theoretical_consumption'**: Consumo tórico, dato clave
- **'Dif' por 'consumption_difference'**: Diferencia de consumos, dato clave
- **'Rendimeinto Std' por 'average_yield'**: Rendimiento promedio, dato clave
- **'Rendimeinto Real' por 'real_yield'**: Rendimiento real, dato clave
- **'Diferencia de Rendimiento' por 'yield_difference'**: Diferencia de rendimientos, dato clave

- **'Metros cuadrados reales' por 'real_produced_square_meters'**: Metros cuadrados reales producidos, dato clave

1.4.3 1.4.3 Rendimientos de pinturas por metro cuadrado

- **'Linea' por 'production_line'**: Identificador para cada línea de producción
- **'Mes' por 'month'**: Mes del dato
- **'Num mes' por 'month_number'**: Numero del mes
- **'Pintura' por 'paint_name'**: Nombre de la pintura
- **'Metros cuadrados reales (m2)' por 'real_square_meters'**: Metros cuadrados de rendimiento de la pintura

1.4.4 1.4.4 Rendimientos de pinturas metros cuadrados por litro

- **'Pintura' por 'paint_name'**: Nombre de la pintura
- **'Clave' por 'paint_code'**: Identificador de la pintura
- **'Rendimiento Canning [m2/L]' por 'paint_performance_m2/l'**: Rendimiento de metros cuadrados por litro

1.4.5 1.4.5 Resumen de producción

- **'Linea' por 'production_line'**: Posible identificador para cada línea de producción
- **'Material Entrada' por 'input_material_code'**: Id del material de entrada, dato clave
- **'Material Salida' por 'output_material_code'**: Id del material de salida, dato clave
- **'Fecha Inicio' por 'start_date'**: Fecha de inicio de la producción, dato clave
- **'Fecha Fin' por 'end_date'**: Fecha de fin de la producción, dato clave
- **'Cliente' por 'client_name'**: Cliente, posible agrupación
- **'Código Clear Inf' por 'inferior_clear_code'**: Código del *clear* inferior
- **'Código Clear Sup' por 'superior_clear_code'**: Código del *clear* superior
- **'Ancho 1' por 'width1_mm'**: Ancho 1, dato clave
- **'Ancho 2' por 'width2_mm'**: Ancho 2, dato clave
- **'Ancho 3' por 'width3_8mm'**: Ancho 3, dato clave
- **'Ancho' por 'width_mm'**: Ancho promedio, dato clave
- **'Espesor 1' por 'thickness1_mm'**: Espesor 1, dato clave
- **'Espesor 2' por 'thickness2_mm'**: Espesor 2, dato clave
- **'Espesor 3' por 'thickness3_mm'**: Espesor 3, dato clave
- **'Espesor' por 'thickness_mm'**: Espesor promedio, dato clave
- **'Peso Entrada' por 'input_weight_kg'**: Peso de entrada, dato clave
- **'Peso' por 'weight_kg'**: Peso de salida, dato clave
- **'Largo' por 'length_m'**: largo total, dato clave
- **'Color Inferior' por 'inferior_color_code'**: Código del color inferior
- **'Color Superior' por 'superior_color_code'**: Código del color superior
- **'Primer Superior' por 'superior_primer_code'**: Código del *primer* superior
- **'Primer Inferior' por 'inferior_primer_code'**: Código del *primer* inferior
- **'Ruta Teórica' por 'route'**: Ruta teórica de la producción, nos sirve para filtrar las líneas de producción de UNI

1.4.6 1.4.6 Defectos de producción

- **'Codigo defecto' por 'defect_code'**: Posible identificador para tipo de defecto
- **'Defecto' por 'defect_name'**: Nombre de defecto
- **'Ubicacion' por 'location'**: Ubicación en la línea donde sucedió el defecto
- **'Es Contencion' por 'is_containment'**: Si el defecto fue contención de un suceso
- **'Es Prevencion' por 'is_preventive'**: Si el defecto fue prevención de un riesgo
- **'Intensidad' por 'intensity'**: Intensidad del suceso
- **'Cara' por 'face'**: Cara donde sucedió el defecto
- **'Lado' por 'side'**: Lado donde sucedió el defecto
- **'Frecuencia' por 'frequency'**: Frecuencia del defecto
- **'Fecha Registro' por 'register_date'**: Fecha del suceso

1.5 1.5 Eliminación de columnas adicionales

Una vez seleccionadas y renombradas las columnas relevantes, se excluyeron de las tablas todas las columnas adicionales.

1.6 1.6 Conversión a formato *feather*

Feather es un format de archivos que permite almacenar tablas o *dataframes* de una manera eficiente y agnostica al lenguaje. Para el reto, es mucho mas eficiente trabajar con los datos en formato *Feather* que directamente con los archivos de Excel, por lo cual se decidió utilizar este formato para almacenar los artefactos generados por la *pipeline* de datos.

El proceso de escribir una tabla a formato *Feather* automáticamente infiere los tipos de todas las columnas, convirtiendo las columnas númericas a sus respectivos tipos, p. ej. '1.02' a tipo *float* y '232' a tipo *int*. Cualquier otra columna será convertida a tipo *object*, con las cuales se debe llevar a cabo una conversión manual. Esta conversión es realizada en secciones posteriores de este documento.

[1]:
```
import pandas as pd
import pyarrow.feather as feather

paint_analysis_df = pd.read_excel('ternium-data/Exp_2_Pintura_Estructura␣
 ↪Analisis de consumo y rendimiento mensual.xlsx',
                                  sheet_name='Base de datos por pintura')
paint_square_meters_df = pd.read_excel('ternium-data/Exp_2_Pintura_Estructura␣
 ↪Analisis de consumo y rendimiento mensual.xlsx',
                                       sheet_name='Metros')
paint_performance_df = pd.read_excel('ternium-data/Exp_2_Pintura_Estructura␣
 ↪Analisis de consumo y rendimiento mensual.xlsx',
                                     sheet_name='Rendimientos INDU')


coating_dfs = pd.read_excel('ternium-data/P12. Pinturas Revestidos Mx␣
 ↪Jul20-Ago23_psch.xlsx',
                                        sheet_name=[
                                           'Pinturas Revestidos Mx Ene21 a '
```

```
                                        ], header=1).values()

production_dfs = [
    pd.read_excel(
        'ternium-data/Archivos piso planta reales/
 ↪ResumenProduccion_AbrilPintado1.xlsx',
        sheet_name='ResumenProduccion'
    ),
    pd.read_excel(
        'ternium-data/Archivos piso planta reales/
 ↪ResumenProduccion_AbrilPintado2.xlsx',
        sheet_name='ResumenProduccion'
    ),
    pd.read_excel(
        'ternium-data/Archivos piso planta reales/
 ↪ResumenProduccion_MayoPintado1.xlsx',
        sheet_name='ResumenProduccion'
    ),
    pd.read_excel(
        'ternium-data/Archivos piso planta reales/
 ↪ResumenProduccion_MayoPintado2.xlsx',
        sheet_name='ResumenProduccion'
    ),
    pd.read_excel(
        'ternium-data/Archivos piso planta reales/
 ↪ResumenProduccion_JunioPintado1.xlsx',
        sheet_name='ResumenProduccion'
    ),
    pd.read_excel(
        'ternium-data/Archivos piso planta reales/
 ↪ResumenProduccion_JunioPintado2.xlsx',
        sheet_name='ResumenProduccion'
    ),
    pd.read_excel(
        'ternium-data/Archivos piso planta reales/
 ↪ResumenProduccion_JulioPintado1.xlsx',
        sheet_name='ResumenProduccion'
    ),
    pd.read_excel(
        'ternium-data/Archivos piso planta reales/
 ↪ResumenProduccion_JulioPintado2.xlsx',
        sheet_name='ResumenProduccion'
    ),
    pd.read_excel(
        'ternium-data/Archivos piso planta reales/
 ↪ResumenProduccion_AgostoPintado1.xlsx',
```

```
        sheet_name='ResumenProduccion'
    ),
    pd.read_excel(
        'ternium-data/Archivos piso planta reales/
 ↪ResumenProduccion_AgostoPintado2.xlsx',
        sheet_name='ResumenProduccion'
    ),
    pd.read_excel(
        'ternium-data/Archivos piso planta reales/
 ↪ResumenProduccion_Enero_Diciembre_2022_Pintado1_UNI.xlsx'
    ),
    pd.read_excel(
        'ternium-data/Archivos piso planta reales/
 ↪ResumenProduccion_Enero_Diciembre_2022_Pintado2_UNI.xlsx'
    )
]
```

[2]:
```
defects_cols = [
    'Codigo defecto',
    'Defecto',
    'Ubicacion',
    'Es Contencion',
    'Es Prevencion',
    'Es Mejora',
    'Intensidad',
    'Cara',
    'Lado',
    'Linea Origen',
    'Dictamen',
    'Resolución',
    'Calidad Sugerida',
    'Siguiente Accion',
    'Siguiente Proceso',
    'Mts Ini',
    'Mts Fin',
    'Frecuencia',
    'Causa',
    'Accion Preventiva',
    'Accion Correctiva',
    'Observaciones',
    'Fecha Registro',
    'Registro Automatico',
    'Código Equipo',
]


def extract_defects_df(production_df: pd.DataFrame):
```

```
    """
    This function aims to extract defects data from a production DataFrame, assuming it has specified columns labeled
    with defect types and corresponding numerical identifiers in brackets like ' (1)', ' (2)', etc. for multiple observations.

    It drops NA values and concatenates the data to form a unified DataFrame that captures all the defects detail.
    The final DataFrame is indexed serially.

    Parameters:

    production_df (DataFrame): pandas DataFrame with production data, expected to include defects_cols with numerical identifiers.

    Returns:
    defects_df: A DataFrame containing the combined defects data from all the identities.
    """
    defects_df_1 = production_df[map(lambda x: x + ' (1)', defects_cols)].dropna(thresh=1)
    defects_df_1.columns = defects_cols
    defects_df_2 = production_df[map(lambda x: x + ' (2)', defects_cols)].dropna(thresh=1)
    defects_df_2.columns = defects_cols
    defects_df_3 = production_df[map(lambda x: x + ' (3)', defects_cols)].dropna(thresh=1)
    defects_df_3.columns = defects_cols
    defects_df_4 = production_df[map(lambda x: x + ' (4)', defects_cols)].dropna(thresh=1)
    defects_df_4.columns = defects_cols
    defects_df_5 = production_df[map(lambda x: x + ' (5)', defects_cols)].dropna(thresh=1)
    defects_df_5.columns = defects_cols

    # the defect ID is the same as the id for its respective production line. As such, we can extract
    # it using reset_index to reference the original production line id in a new column.
    # then, the resulting dfs are concatenated while ignoring the existing index.
    defects_df = pd.concat([
        defects_df_1.reset_index(names='production_id'),
        defects_df_2.reset_index(names='production_id'),
        defects_df_3.reset_index(names='production_id'),
        defects_df_4.reset_index(names='production_id'),
```

```
        defects_df_5.reset_index(names='production_id'),
    ], ignore_index=True)

    return defects_df


def remove_defects_cols(production_id: pd.DataFrame):
    """
    Remove defects columns from the given production_id DataFrame.

    :param production_id: The DataFrame containing the production data.
    :return: The DataFrame with defects columns removed.
    """
    columns_to_remove = list(map(lambda x: x + ' (1)', defects_cols)) + list(
        map(lambda x: x + ' (2)', defects_cols)) + list(map(lambda x: x + '␣
 ↪(3)', defects_cols)) + list(
        map(lambda x: x + ' (4)', defects_cols)) + list(map(lambda x: x + '␣
 ↪(5)', defects_cols))
    return production_id.drop(columns=columns_to_remove)


coating_df = pd.concat(coating_dfs, ignore_index=True)

# Análisis de consumo

# join production summaries

production_df = pd.concat(production_dfs, ignore_index=True)

# extract defects from production summary
defects_df = extract_defects_df(production_df)



# clean up production production dataframe
production_df.index = production_df.index.astype(str)

production_df.rename(columns={
    'Material Entrada': 'input_material_code',
    'Material Salida': 'output_material_code',
    'Fecha Inicio': 'start_date',
    'Fecha Fin': 'end_date',
    'Cliente': 'client_name',
    'Código Clear Inf': 'inferior_clear_code',
    'Código Clear Sup': 'superior_clear_code',
    'Ancho 1': 'width1_mm',
    'Ancho 2': 'width2_mm',
```

```
    'Ancho 3': 'width3_mm',
    'Ancho': 'width_mm',
    'Espesor 1': 'thickness1_mm',
    'Espesor 2': 'thickness2_mm',
    'Espesor 3': 'thickness3_mm',
    'Espesor': 'thickness_mm',
    'Peso Entrada': 'input_weight_kg',
    'Usuario': 'user',
    'Peso': 'weight_kg',
    'Largo': 'length_m',
    'Color Inferior': 'inferior_color_code',
    'Color Superior': 'superior_color_code',
    'Primer Superior': 'superior_primer_code',
    'Primer Inferior': 'inferior_primer_code',
    'Ruta Teórica': 'route',
}, inplace=True, errors='raise')

production_df = production_df[[
    'input_material_code',
    'output_material_code',
    'start_date',
    'end_date',
    'client_name',
    'inferior_clear_code',
    'superior_clear_code',
    'width1_mm',
    'width2_mm',
    'width3_mm',
    'width_mm',
    'thickness1_mm',
    'thickness2_mm',
    'thickness3_mm',
    'thickness_mm',
    'input_weight_kg',
    'weight_kg',
    'length_m',
    'inferior_color_code',
    'superior_color_code',
    'superior_primer_code',
    'inferior_primer_code',
    'route',
]]

# clean up defects dataframe
defects_df.rename(columns={
    'Codigo defecto': 'defect_code',
    'Defecto': 'defect_name',
```

```
    'Ubicacion': 'location',
    'Es Contencion': 'is_containment',
    'Es Prevencion': 'is_preventive',
    'Intensidad': 'intensity',
    'Cara': 'face',
    'Lado': 'side',
    'Frecuencia': 'frequency',
    'Fecha Registro': 'register_date',
}, inplace=True, errors='raise')

defects_df = defects_df[[
    'defect_code',
    'defect_name',
    'location',
    'is_containment',
    'is_preventive',
    'intensity',
    'face',
    'side',
    'frequency',
    'register_date',
]]

# clean up paint coating dataframe
coating_df.rename(columns={
    'Denominación objeto': 'production_line',
    'Material': 'paint_id',
    'Texto breve de material': 'paint_name',
    '    Valor var.': 'monetary_value_usd',
    'Ctd.total reg.': 'total_liters_used',
    'Planta': 'production_plant',
    'Proveedor': 'supplier',
    'Usuario': 'user',
    'Registrado': 'date',
    'Hora': 'hour',
    'Precio': 'price_per_liter',
    'Línea': 'line',
}, inplace=True, errors='raise')

coating_df = coating_df[[
    'production_line',
    'paint_id',
    'paint_name',
    'monetary_value_usd',
    'total_liters_used',
    'production_plant',
    'supplier',
```

```
    'date',
    'hour',
    'price_per_liter',
]]

coating_df = coating_df[coating_df['production_plant'] == 'Uni']

coating_df['date'] = pd.to_datetime(coating_df['date'])

# clean up paint analysis dataframe
paint_analysis_df.rename(columns={
    'Linea ': 'production_line',
    'Mes': 'month',
    'Mes num': 'month_number',
    'Pintura': 'paint',
    'Real': 'real_consumption',
    'Teo': 'theoretical_consumption',
    'Dif': 'consumption_difference',
    'Rendimeinto Std': 'average_yield',
    'Rendimeinto Real': 'real_yield',
    'Diferencia de Rendimiento': 'yield_difference',
    'Metros cuadrados reales': 'real_produced_square_meters',
}, inplace=True, errors='raise')

paint_analysis_df = paint_analysis_df[[
    'production_line',
    'month',
    'month_number',
    'paint',
    'real_consumption',
    'theoretical_consumption',
    'consumption_difference',
    'average_yield',
    'real_yield',
    'yield_difference',
    'real_produced_square_meters'
]]

# clean up paint square meters dataframe
paint_square_meters_df.rename(columns={
    'Linea': 'production_line',
    'Mes': 'month',
    'Num mes': 'month_number',
    'Pintura': 'paint_name',
    'Metros cuadrados reales (m2)': 'real_square_meters',
}, inplace=True, errors='raise')
```

```
# clean up paint performance dataframe
paint_performance_df.rename(columns={
    'Pintura': 'paint_name',
    'Clave': 'paint_code',
    'Rendimiento Canning [m2/L]': 'paint_performance_m2/l'
}, inplace=True, errors='raise')

production_df['m2'] = production_df['length_m']*(production_df['width_mm']/1000)

# Separated production by paint type
inferior_clear_production_df = production_df.copy().
 ↪drop(columns=['superior_clear_code', 'inferior_color_code',␣
 ↪'superior_color_code', 'superior_primer_code', 'inferior_primer_code'],␣
 ↪axis=1).rename(columns={'inferior_clear_code': 'paint_code'})
superior_clear_production_df = production_df.copy().
 ↪drop(columns=['inferior_clear_code', 'inferior_color_code',␣
 ↪'superior_color_code', 'superior_primer_code', 'inferior_primer_code'],␣
 ↪axis=1).rename(columns={'superior_clear_code': 'paint_code'})
inferior_color_production_df = production_df.copy().
 ↪drop(columns=['inferior_clear_code','superior_clear_code', ␣
 ↪'superior_color_code', 'superior_primer_code', 'inferior_primer_code'],␣
 ↪axis=1).rename(columns={'inferior_color_code': 'paint_code'})
superior_color_production_df = production_df.copy().
 ↪drop(columns=['inferior_clear_code','superior_clear_code',␣
 ↪'inferior_color_code', 'superior_primer_code', 'inferior_primer_code'],␣
 ↪axis=1).rename(columns={'superior_color_code': 'paint_code'})
inferior_primer_production_df = production_df.copy().
 ↪drop(columns=['inferior_clear_code','superior_clear_code',␣
 ↪'inferior_color_code', 'superior_color_code', 'superior_primer_code'],␣
 ↪axis=1).rename(columns={'inferior_primer_code': 'paint_code'})
superior_primer_production_df = production_df.copy().
 ↪drop(columns=['inferior_clear_code','superior_clear_code',␣
 ↪'inferior_color_code', 'superior_color_code', 'inferior_primer_code'],␣
 ↪axis=1).rename(columns={'superior_primer_code': 'paint_code'})
separated_production_df = pd.concat([inferior_clear_production_df,␣
 ↪superior_clear_production_df, inferior_color_production_df,␣
 ↪superior_clear_production_df, inferior_primer_production_df,␣
 ↪superior_clear_production_df], ignore_index=True)
separated_production_df.dropna(subset='paint_code', inplace=True)

# Grouped paint production by paint and date
date_column = separated_production_df['start_date'].dt.date
paint_code_color = separated_production_df['paint_code']
paint_production_groups = separated_production_df[['length_m', 'm2',␣
 ↪'thickness_mm']].groupby(by=[paint_code_color, date_column])
paint_production_per_date_df = paint_production_groups.sum()
```

```
paint_production_per_date_df.index.names = ['paint_name', 'date']

# Grouped coating by paint and date
date_column = coating_df['date'].dt.date
paint_code = coating_df['paint_name']
coating_groups = coating_df[['total_liters_used']].groupby(by=[paint_code,␣
 ↪date_column])
coating_per_paint_date_df = coating_groups.sum()

# Joined paint production and coating, by paint and date
paint_per_date_df = paint_production_per_date_df.join(coating_per_paint_date_df)
paint_per_date_df['real_yield'] = paint_per_date_df['m2'] /␣
 ↪paint_per_date_df['total_liters_used']


# write all dataframes to disk
feather.write_feather(coating_df, 'data/pinturas_revestidos_jul20_ago23.
 ↪feather')

feather.write_feather(paint_analysis_df, 'data/analisis_consumo_pintura.
 ↪feather')
feather.write_feather(paint_square_meters_df, 'data/
 ↪pintura_metros_cuadrados_reales.feather')
feather.write_feather(paint_performance_df, 'data/rendimiento_pintura.feather')

feather.write_feather(production_df, 'data/production.feather')
feather.write_feather(defects_df, 'data/defects.feather')
```

```
/tmp/ipykernel_140901/3908150230.py:89: FutureWarning: The behavior of DataFrame
concatenation with empty or all-NA entries is deprecated. In a future version,
this will no longer exclude empty or all-NA columns when determining the result
dtypes. To retain the old behavior, exclude the relevant entries before the
concat operation.
  production_df = pd.concat(production_dfs, ignore_index=True)
/tmp/ipykernel_140901/3908150230.py:89: FutureWarning: The behavior of DataFrame
concatenation with empty or all-NA entries is deprecated. In a future version,
this will no longer exclude empty or all-NA columns when determining the result
dtypes. To retain the old behavior, exclude the relevant entries before the
concat operation.
  production_df = pd.concat(production_dfs, ignore_index=True)
/tmp/ipykernel_140901/3908150230.py:210: UserWarning: Parsing dates in %d.%m.%Y
format when dayfirst=False (the default) was specified. Pass `dayfirst=True` or
specify a format to silence this warning.
  coating_df['date'] = pd.to_datetime(coating_df['date'])
```

[3]:
```
coating_df[['paint_name', 'date', 'monetary_value_usd']].
 ↪groupby(by=['paint_name', 'date']).sum()
```

[3]:

paint_name	date	monetary_value_usd
0001-PRIMER 4457	2021-02-01	3936.33
	2021-02-02	1098.00
	2021-02-03	4282.20
	2021-02-04	3184.20
	2021-02-05	933.30
…		…
WHITE KRYSTAL KOTE	2023-08-18	12208.00
	2023-08-19	6864.00
	2023-08-25	26208.00
	2023-08-26	4680.00
WHITE TEXT KOTE	2020-07-28	14198.70

[15967 rows x 1 columns]

[4]:
```
coating_df[['paint_name', 'date', 'total_liters_used', 'monetary_value_usd']] .
 ↪groupby(by=['paint_name', 'date']) .sum()
```

[4]:

paint_name	date	total_liters_used	monetary_value_usd
0001-PRIMER 4457	2021-02-01	717.0	3936.33
	2021-02-02	200.0	1098.00
	2021-02-03	780.0	4282.20
	2021-02-04	580.0	3184.20
	2021-02-05	170.0	933.30
…		…	…
WHITE KRYSTAL KOTE	2023-08-18	980.0	12208.00
	2023-08-19	550.0	6864.00
	2023-08-25	2100.0	26208.00
	2023-08-26	375.0	4680.00
WHITE TEXT KOTE	2020-07-28	1590.0	14198.70

[15967 rows x 2 columns]

[5]:
```
# Ensure 'date' is of datetime type
coating_df['date'] = pd.to_datetime(coating_df['date'])

# Extract year from 'date'
coating_df['year'] = coating_df['date'].dt.year

# Calculate liters per USD (assuming 'monetary_value_usd' is the total cost for␣
 ↪'total_liters_used')
coating_df['liters_per_usd'] =  coating_df['monetary_value_usd']/
 ↪coating_df['total_liters_used']

# Group by 'paint_name', 'year', and apply describe on 'liters_per_usd'
```

```
liters_per_usd_description = coating_df.groupby(['paint_name',␣
 ↪'year'])['liters_per_usd'].describe()

# Display the results
print("Liters per USD Description by Paint and Year:")

liters_per_usd_description
```

```
Liters per USD Description by Paint and Year:
```

[5]:

		count	mean	std	min	25%	50%	\
paint_name	year							
0001-PRIMER 4457	2021	1381.0	5.683555	2.228726e-01	5.49	5.49	5.49	
	2022	577.0	6.374367	3.463144e-01	5.94	5.94	6.65	
	2023	730.0	6.812041	7.084498e-02	6.68	6.85	6.85	
0003-PRIMARIO 4435	2022	14.0	7.760000	2.413594e-15	7.76	7.76	7.76	
	2023	10.0	7.544000	9.811558e-02	7.43	7.43	7.62	
…		…	…	…	…	…	…	
WHITE KRYSTAL KOTE	2020	184.0	9.460000	2.012976e-15	9.46	9.46	9.46	
	2021	274.0	10.278978	4.386514e-01	9.96	9.96	9.96	
	2022	305.0	12.314754	7.006942e-01	11.64	11.64	11.64	
	2023	212.0	12.692830	1.513839e-01	12.48	12.48	12.80	
WHITE TEXT KOTE	2020	9.0	8.930000	0.000000e+00	8.93	8.93	8.93	

		75%	max
paint_name	year		
0001-PRIMER 4457	2021	5.94	5.940039
	2022	6.65	6.650000
	2023	6.85	6.850000
0003-PRIMARIO 4435	2022	7.76	7.760000
	2023	7.62	7.620000
…		…	…
WHITE KRYSTAL KOTE	2020	9.46	9.460000
	2021	10.88	10.880000
	2022	13.04	13.040000
	2023	12.80	12.800000
WHITE TEXT KOTE	2020	8.93	8.930000

```
[878 rows x 8 columns]
```

[6]:
```
#merge de coating and production
#database_df = coating_df.merge(separated_production_df, left_on='paint_name',␣
 ↪right_on='paint_code')
#database_df

#dataframes to excel
with pd.ExcelWriter('database.xlsx') as writer:
```

```
    coating_df.to_excel(writer, sheet_name='coating_db')
    production_df.to_excel(writer, sheet_name='production_db')
    defects_df.to_excel(writer, sheet_name='defects_db')
    paint_analysis_df.to_excel(writer, sheet_name='paint_analysis_db')
```

1.7 Contribución individual

Hiram Muñoz Calculo de variables diarias

Raúl Murillo Separación de production_df y exportación de los dataframes al archivo excel

Andrea Garza Importacion de datos nuevos y separacion de production_df

Erick Hernández Importación de datos nuevos, separacion de production_df y calculo de variables diarias

David Martínez Limpieza de fechas y calculo de variables diarias