deuda_previsional_a_ANHO (1)

May 29, 2025

[ ]:
```
# 1. Instalar librerías necesarias
%%capture
!pip install swifter
!pip install unidecode
!pip install openpyxl
```

[ ]:
```
# 2.Montar google drive
from google.colab import drive
drive.mount('/content/drive')
```

[ ]:
```
# 3. Importar librerias
import os
import pandas as pd
import numpy as np
from unidecode import unidecode
import re
import swifter
from IPython.display import display, HTML
import openpyxl

print("Librerías importadas correctamente.")
```

```
Librerías importadas correctamente.
```

[ ]:
```
# 4. Definir la variable anho
anho = input("Ingresa el año a calcular la deuda previsional:")
anho
```

[ ]:
```
# 5. cargar csv con deuda a calcular y filtrar para 2020 hacia atrás

csv_file_path = '/content/drive/MyDrive/DEUDA_PREVISIONAL/CSV/
 ↪20250417_consolidado.csv'

df = pd.read_csv(csv_file_path, sep=";", encoding="utf-8", low_memory=False)

# Pasar a formato fecha
df['periodo_adeudado'] = pd.to_datetime(df['periodo_adeudado'], errors='coerce')
```

```
# Filtrar para años menores a anho
df_anho_consolidado = df[df['periodo_adeudado'].dt.year <= 2020].copy()

print(f"Filas en df_anho_consolidado (periodos <= 2020):␣
 ↪{len(df_anho_consolidado)}")
print(df_anho_consolidado.head())
```

[ ]:
```
# 6. leer csv con datos de reajuste, interes y recargo

# Load the CSV file into a pandas DataFrame
df_deuda_anho = pd.read_csv(f'/content/drive/MyDrive/DEUDA_PREVISIONAL/
 ↪datos_{anho}_deuda.csv', sep=",", encoding="utf-8", low_memory=False)

df_deuda_anho.columns = ["MES AÑO", "REAJUSTE", "INTERES", "RECARGO"]
print("\nColumnas del DataFrame df_deuda_anho renombradas:")
display(df_deuda_anho.head())
```

[ ]:
```
# 7. Convertir a numéricos los datos de las columnas del df_deuda_anho

# Lista de columnas a convertir
columnas_a_convertir = ['REAJUSTE', 'INTERES', 'RECARGO']

for col in columnas_a_convertir:
    if col in df_deuda_anho.columns:
        # Limpiar '%' y reemplazar ',' por '.'
        # Asegurar que se trate como string antes de reemplazar
        df_deuda_anho[col] = df_deuda_anho[col].astype(str).str.replace('%',␣
 ↪'', regex=False).str.replace(',', '.', regex=False)
        # Convertir a numérico, con errors='coerce' para manejar valores no␣
 ↪numéricos
        df_deuda_anho[col] = pd.to_numeric(df_deuda_anho[col], errors='coerce')
        # Dividir por 100 para convertir de porcentaje a factor decimal
        df_deuda_anho[col] = df_deuda_anho[col] / 100.0
    else:
        print(f"Advertencia: La columna '{col}' no se encontró en el DataFrame␣
 ↪df_deuda_anho.")

# Asegurar que 'MES AÑO' sea string y limpiar espacios
df_deuda_anho['MES AÑO'] = df_deuda_anho['MES AÑO'].astype(str).str.strip()


# Opcional: Muestra los tipos de datos después de la conversión
print("\nTipos de datos después de la conversión en df_deuda_anho:")
print(df_deuda_anho.dtypes)
```

```
# Opcional: Muestra las primeras filas para verificar los valores
print("\nDataFrame df_deuda_anho después de la conversión:")
print(df_deuda_anho.head())
```

0.1 Leer CSV

[ ]:
```
# 8. Procesamiento csv con las deudas

print("### Iniciando Carga y Preprocesamiento Base del CSV ###")
df_deuda_base = df_anho_consolidado.copy()
print(f"df_deuda_base cargado. Filas iniciales: {len(df_deuda_base)}")

df_deuda_base.columns = [str(col).strip().lower() for col in df_deuda_base.
 ↪columns]

MESES_ES_LIST_FORMAT = ["Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio",
                        "Julio", "Agosto", "Septiembre", "Octubre",␣
 ↪"Noviembre", "Diciembre"]
MESES_ES_REGEX_MAP_FORMAT = {
    "ENERO": "Enero", "FEBRERO": "Febrero", "MARZO": "Marzo", "ABRIL": "Abril",
    "MAYO": "Mayo", "JUNIO": "Junio", "JULIO": "Julio", "AGOSTO": "Agosto",
    "SEPTIEMBRE": "Septiembre", "OCTUBRE": "Octubre", "NOVIEMBRE": "Noviembre",
    "DICIEMBRE": "Diciembre",
    "ENE": "Enero", "FEB": "Febrero", "MAR": "Marzo", "ABR": "Abril",
    "MAY": "Mayo", "JUN": "Junio", "JUL": "Julio", "AGO": "Agosto",
    "SEP": "Septiembre", "OCT": "Octubre", "NOV": "Noviembre", "DIC":␣
 ↪"Diciembre"
}

def formatear_periodo_optimizado_csv(periodo_str_orig):
    if pd.isna(periodo_str_orig):
        return None
    # Si ya es datetime, formatearlo directamente
    if isinstance(periodo_str_orig, pd.Timestamp):
        return f"{MESES_ES_LIST_FORMAT[periodo_str_orig.month-1]}␣
 ↪{periodo_str_orig.year}"

    periodo_str = str(periodo_str_orig).strip()
    # Intenta convertir formatos comunes de fecha primero
    try:
        dt_obj = pd.to_datetime(periodo_str, errors='raise') # 'raise' para␣
 ↪probar el primer intento
        return f"{MESES_ES_LIST_FORMAT[dt_obj.month-1]} {dt_obj.year}"
    except ValueError:
        pass # Continúa con otros métodos si falla la conversión directa
```

```
    # Normalizar a mayúsculas y sin tildes para la comparación con regex
    periodo_str_norm = unidecode(periodo_str.upper())

    # Patrones regex más robustos
    # Formato MES AÑO (e.g., "MAYO 1997", "MAYO1997", "MAYO 1997")
    match_mes_ano = re.fullmatch(r"([A-Z]+)\s*(\d{4})", periodo_str_norm)
    if match_mes_ano:
        mes_detectado_upper = match_mes_ano.group(1)
        ano_str = match_mes_ano.group(2)
        if mes_detectado_upper in MESES_ES_REGEX_MAP_FORMAT:
            return f"{MESES_ES_REGEX_MAP_FORMAT[mes_detectado_upper]} {ano_str}"

    # Formato AÑO MES (e.g., "1997 MAYO", "1997MAYO")
    match_ano_mes = re.fullmatch(r"(\d{4})\s*([A-Z]+)", periodo_str_norm)
    if match_ano_mes:
        ano_str = match_ano_mes.group(1)
        mes_detectado_upper = match_ano_mes.group(2)
        if mes_detectado_upper in MESES_ES_REGEX_MAP_FORMAT:
            return f"{MESES_ES_REGEX_MAP_FORMAT[mes_detectado_upper]} {ano_str}"

    # Formato YYYYMM (e.g., "201403")
    match_yyyymm = re.fullmatch(r"(\d{4})(\d{2})", periodo_str)
    if match_yyyymm:
        try:
            year = int(match_yyyymm.group(1))
            month = int(match_yyyymm.group(2))
            if 1 <= month <= 12:
                return f"{MESES_ES_LIST_FORMAT[month-1]} {year}"
        except ValueError:
            pass
    return None # No se pudo formatear

print("Formateando 'periodo_adeudado' a 'periodo_formateado' en df_deuda_base...
 ↪")
if 'periodo_adeudado' not in df_deuda_base.columns:
     raise KeyError("La columna 'periodo_adeudado' no existe en df_deuda_base.")

# Aplicar la función de formateo
df_deuda_base['periodo_formateado'] = df_deuda_base['periodo_adeudado'].swifter.
 ↪apply(formatear_periodo_optimizado_csv)
print("'periodo_formateado' creado en df_deuda_base.")

# Verificar cuántos no se pudieron formatear
print(f"Periodos no formateados: {df_deuda_base['periodo_formateado'].isnull().
 ↪sum()}")
print("Ejemplos de periodos formateados y no formateados:")
print(df_deuda_base[['periodo_adeudado', 'periodo_formateado']].sample(10))
```

```
if 'monto_nominal_adeudado' not in df_deuda_base.columns:
    raise KeyError("La columna 'monto_nominal_adeudado' no existe en␣
 ↪df_deuda_base.")
df_deuda_base['monto_nominal_adeudado'] = pd.
 ↪to_numeric(df_deuda_base['monto_nominal_adeudado'], errors='coerce').
 ↪fillna(0.0)
print("'monto_nominal_adeudado' convertido a numérico en df_deuda_base.")

print("### Fin de Carga y Preprocesamiento Base del CSV ###")
```

[ ]:
```
# 9 - BUCLE PRINCIPAL DE CÁLCULO
print(f"### Iniciando Bucle Principal de Cálculo para el año anho ###")

# --- Función Auxiliar Específica para el Excel de anho ---
# df_excel_anho_global es el DataFrame cargado y preprocesado en las celdas 6 y␣
 ↪7
def obtener_valor_desde_excel_anho(periodo_deuda_mes_ano, df_excel_anho_global,␣
 ↪tipo_tasa_buscada):
    """
    Busca el valor de REAJUSTE, INTERES o RECARGO en el df_deuda_anho␣
 ↪(df_excel_anho_global).
    tipo_tasa_buscada debe ser 'REAJUSTE', 'INTERES', o 'RECARGO'.
    Retorna el valor numérico (factor) o 0.0 si no se encuentra o es NaN.
    Maneja diferencias de capitalización en los meses.
    """
    if pd.isna(periodo_deuda_mes_ano):
        return 0.0

    partes_periodo = periodo_deuda_mes_ano.split()
    if len(partes_periodo) == 2:
        mes_busqueda = unidecode(partes_periodo[0].lower()) # Convertir mes a␣
 ↪minúsculas y sin tildes
        ano_busqueda = partes_periodo[1]

        # Itera sobre el DataFrame de Excel preprocesado
        for index, row_excel in df_excel_anho_global.iterrows():
            mes_ano_excel = str(row_excel['MES AÑO']).strip() # 'MES AÑO' es la␣
 ↪columna clave en df_deuda_anho
            partes_excel = mes_ano_excel.split()
            if len(partes_excel) == 2:
                mes_excel_norm = unidecode(partes_excel[0].lower())
                ano_excel_norm = partes_excel[1]
```

```
                if mes_excel_norm == mes_busqueda and ano_excel_norm ==␣
 ↪ano_busqueda:
                    valor = row_excel[tipo_tasa_buscada] # tipo_tasa_buscada es␣
 ↪'REAJUSTE', 'INTERES', o 'RECARGO'
                    return valor if pd.notna(valor) else 0.0
    return 0.0
# --- Fin Función Auxiliar para anho ---

# Funciones de cálculo simplificadas para anho, usan el df_deuda_anho global
def calcular_reajuste_anho(row, df_global):
    if pd.isna(row['periodo_formateado']) or row['monto_nominal_adeudado'] == 0:
        return 0.0
    tasa_reajuste = obtener_valor_desde_excel_anho(row['periodo_formateado'],␣
 ↪df_global, "REAJUSTE")
    return row['monto_nominal_adeudado'] * tasa_reajuste

def calcular_interes_anho(row, df_global):
    if pd.isna(row['periodo_formateado']) or row['monto_nominal_adeudado'] == 0:
        return 0.0
    tasa_interes = obtener_valor_desde_excel_anho(row['periodo_formateado'],␣
 ↪df_global, "INTERES")
    return row['monto_nominal_adeudado'] * tasa_interes

def calcular_recargo_anho(row, df_global):
    if pd.isna(row['periodo_formateado']) or row['monto_nominal_adeudado'] == 0:
        return 0.0
    tasa_recargo = obtener_valor_desde_excel_anho(row['periodo_formateado'],␣
 ↪df_global, "RECARGO")
    return row['monto_nominal_adeudado'] * tasa_recargo

# Seleccionar columnas base para mantener en el resultado final
columnas_originales_ordenadas = [
    'rut_sostenedor', 'rut_trabajador', 'institucion', 'tipo_cuenta',
    'nom_com_sost', 'ano_traspaso', 'nombre_slep',
    'periodo_adeudado', 'periodo_formateado', 'monto_nominal_adeudado'
]
columnas_a_mantener_final = [col for col in columnas_originales_ordenadas if␣
 ↪col in df_deuda_base.columns]
df_calculo_final_acumulado = df_deuda_base[columnas_a_mantener_final].copy()

# procesar anho

print(f"\n--- PROCESANDO AÑO DE CÁLCULO: {anho} ---")
# La fecha de pago es referencial y no se usa directamente en las búsquedas del␣
 ↪Excel de anho
fecha_pago_actual = f"31.12.{anho}"
```

```
print(f"Fecha de pago (referencial) para {anho}: {fecha_pago_actual}")

# Definir nombres de columnas para el año anho
col_reajuste_anual = f"REAJUSTE_{anho}"
col_interes_anual = f"INTERES_{anho}"
col_recargo_anual = f"RECARGO_{anho}"
col_total_anual = f"TOTAL_{anho}"

print("Usando datos del DataFrame 'df_deuda_anho' (cargado y preprocesado␣
 ↪previamente).")

# Calcular Reajustes para anho
print(f"Calculando {col_reajuste_anual} (Swifter)...")
# Pasamos directamente df_deuda_anho (que es el global preprocesado)
df_calculo_final_acumulado[col_reajuste_anual] = df_calculo_final_acumulado.
 ↪swifter.apply(
    calcular_reajuste_anho,
    args=(df_deuda_anho,),
    axis=1
)
print(f"{col_reajuste_anual} calculados.")

# Calcular Intereses para anho
print(f"Calculando {col_interes_anual} (Swifter)...")
df_calculo_final_acumulado[col_interes_anual] = df_calculo_final_acumulado.
 ↪swifter.apply(
    calcular_interes_anho,
    args=(df_deuda_anho,),
    axis=1
)
print(f"Cálculo de {col_interes_anual} finalizado.")

# Calcular Recargos para anho
print(f"Calculando {col_recargo_anual} (Swifter)...")
df_calculo_final_acumulado[col_recargo_anual] = df_calculo_final_acumulado.
 ↪swifter.apply(
    calcular_recargo_anho,
    args=(df_deuda_anho,),
    axis=1
)
print(f"Cálculo de {col_recargo_anual} finalizado.")

# Calcular Columna Total para el año anho
df_calculo_final_acumulado[col_total_anual] = (
    df_calculo_final_acumulado['monto_nominal_adeudado'].fillna(0) +
    df_calculo_final_acumulado[col_reajuste_anual].fillna(0) +
    df_calculo_final_acumulado[col_interes_anual].fillna(0) +
```

```
    df_calculo_final_acumulado[col_recargo_anual].fillna(0)
)
print(f"Cálculo de {col_total_anual} finalizado para el año {anho}.")
print(f"--- FIN PROCESO AÑO {anho} ---")


# --- Fin del Bucle Principal ---
print("\n### Resumen del DataFrame de Deuda Calculada para anho ###")
print(f"Dimensiones del DataFrame final: {df_calculo_final_acumulado.shape}")
print("\nColumnas del DataFrame final:")
print(df_calculo_final_acumulado.columns.tolist())
print("\nPrimeras 5 filas del resultado:")
pd.options.display.float_format = '{:.2f}'.format
print(df_calculo_final_acumulado.head().to_string())
print("\nInformación del DataFrame final:")
df_calculo_final_acumulado.info(verbose=True)

print("\nConteo de NaNs por columnas calculadas para anho:")
cols_check_nan_anho = [
    f"REAJUSTE_{anho}", f"INTERES_{anho}", f"RECARGO_{anho}",
    f"TOTAL_{anho}"
]
cols_check_nan_existentes_anho = [col for col in cols_check_nan_anho if col in␣
 ↪df_calculo_final_acumulado.columns]

if cols_check_nan_existentes_anho:
    print(df_calculo_final_acumulado[cols_check_nan_existentes_anho].isnull().
 ↪sum())
else:
    print("No hay columnas calculadas para anho para verificar NaNs.")

print("\nVerificación de valores calculados (ejemplo para REAJUSTE_anho):")
print(df_calculo_final_acumulado[f"REAJUSTE_{anho}"].describe())
# Mostrar filas donde el REAJUSTE es significativamente diferente de cero o NaN␣
 ↪para verificar la búsqueda
print("\nEjemplos de Reajustes calculados (top 5 donde REAJUSTE_anho > 0):")
print(df_calculo_final_acumulado[df_calculo_final_acumulado[f"REAJUSTE_{anho}"]␣
 ↪> 0][
    ['periodo_formateado', 'monto_nominal_adeudado', f"REAJUSTE_{anho}",␣
 ↪f"INTERES_{anho}", f"RECARGO_{anho}", f"TOTAL_{anho}"]
].head())
print("\nEjemplos de periodos que podrían no encontrar match en el Excel␣
 ↪(REAJUSTE_anho es 0 y monto > 0):")
print(df_calculo_final_acumulado[
    (df_calculo_final_acumulado[f"REAJUSTE_{anho}"] == 0) &␣
 ↪(df_calculo_final_acumulado['monto_nominal_adeudado'] > 0)
```

```
]][['periodo_formateado', 'monto_nominal_adeudado', f"REAJUSTE_{anho}"]].head())


print(f"\n### Fin de Celda 9 (Bucle de Cálculo para {anho}) ###")
```

[ ]:
```
# 10. Guardar el DataFrame resultante
try:
    output_dir = "/content/drive/MyDrive/DEUDA_PREVISIONAL/RESULTADOS/"
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    # Nombre de archivo que indica el año específico anho y la fuente (Excel)
    nombre_base_archivo = f"deuda_calculada_ANIO_{anho}"
    nombre_archivo_salida_csv = os.path.join(output_dir,␣
 ↪f"{nombre_base_archivo}.csv")
    nombre_archivo_salida_excel = os.path.join(output_dir,␣
 ↪f"{nombre_base_archivo}.xlsx")

    # Guardar como CSV
    df_calculo_final_acumulado.to_csv(nombre_archivo_salida_csv, index=False,␣
 ↪sep=';', encoding = 'utf-8')
    print(f"\nDataFrame completo con cálculos para el año {anho} guardado en␣
 ↪CSV: {nombre_archivo_salida_csv}")

    # Guardar como Excel
    try:
        df_calculo_final_acumulado.to_excel(nombre_archivo_salida_excel,␣
 ↪index=False)
        print(f"DataFrame completo con cálculos para el año {anho} guardado en␣
 ↪Excel: {nombre_archivo_salida_excel}")
    except Exception as e_excel:
        print(f"\nError al guardar el archivo Excel: {e_excel}")
        print("Asegúrate de tener 'openpyxl' instalado y que el DataFrame no␣
 ↪sea demasiado grande para Excel.")

except Exception as e:
    print(f"\nError general al guardar archivos: {e}")
```