03.1_Division_TrainValTest

Objetivo

Cargar el dataset limpio de la encuesta NFCS 2021, explorar rápidamente su estructura, derivar algunas variables básicas y dividir el conjunto de datos en particiones estratificadas de entrenamiento, validación y prueba para su posterior modelado.

Entradas (Inputs)

- `data/processed/final/NFCS_2021_final_clean.csv`

Salidas (Outputs)

- `data/splits/final/X_train.parquet`
- `data/splits/final/y_train.parquet`
- `data/splits/final/X_val.parquet`
- `data/splits/final/y_val.parquet`
- `data/splits/final/X_test.parquet`
- `data/splits/final/y_test.parquet`

+ Código + Texto

Resumen Ejecutivo

- El notebook aborda la preparación de los datos limpios de la encuesta NFCS 2021 para un proyecto de modelado de riesgo financiero.
- Se importa el dataset desde Google Drive y se definen las librerías clave (pandas, NumPy, scikit-learn, pyarrow).
- Se crea la variable **FL_SCORE** (suma de aciertos en preguntas de alfabetización financiera) y una métrica de **PORTFOLIO_DIVERSITY**.
- Se especifican las columnas de identificación (`NFCSID`), el objetivo (`B10`) y el peso de muestreo (`WGT1`).
- Se construye la matriz de características **X** y el vector objetivo **y**, eliminando IDs y columnas no predictivas.
- Se realiza una división estratificada en tres conjuntos: entrenamiento (≈70 %), validación (≈15 %) y test (≈15 %) usando `train_test_split`.
- Los tamaños resultantes son **X_train** (1976×92), **X_val** (424×92) y **X_test** (392×92), manteniendo la distribución de la variable objetivo.
- Finalmente, se guardan los tres "splits" en formato Parquet mediante **pyarrow** para facilitar su carga en etapas posteriores.

1. Carga de dependencias y montaje de Google Drive

Agrupa e importa todas las librerías necesarias (estándar, terceros y locales) y monta Google Drive para acceder a los datos.

```
import sys
from pathlib import Path

from google.colab import drive
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

# Configuración de semilla para reproducibilidad
RANDOM_STATE = 42

# Montar Google Drive
drive.mount('/content/drive', force_remount=True)

# Añadir la raíz del proyecto al path para poder importar 'config'
ROOT_PATH_STR = '/content/drive/MyDrive/TFM-AntonioEsquinas'
if ROOT_PATH_STR not in sys.path:
    sys.path.append(ROOT_PATH_STR)

# Importar las rutas necesarias desde el archivo de configuración
# Se importa la ruta del dataset procesado final y la de los splits finales.
from config import FINAL_PROCESSED_DATA_DIR, FINAL_SPLITS_DIR, METADATA_DIR

# Definir la ruta del archivo de entrada usando las variables de config
DATA_FILE_PATH = FINAL_PROCESSED_DATA_DIR / 'NFCS_2021_final_clean.csv'

# Leer el dataset
df = pd.read_csv(DATA_FILE_PATH)
print(f" Dataset cargado con {df.shape[0]} filas y {df.shape[1]} columnas")

print('\n Primeras filas del dataset:')
display(df.head())

print('\n Información general:')
display(df.info())

print('\n Estadísticas descriptivas:')
display(df.describe())

# Conteo de valores nulos
TARGET = 'B10' # Asegúrate de que esta es tu variable objetivo
na_pct_full = df.isna().mean().sort_values(ascending=False)
print('\n Top 10 variables con más % de NA:')
display(na_pct_full.head(10))

# Distribución de la variable objetivo
```

```
    if TARGET in df.columns:
        print(f"\n Distribución de la variable objetivo '{TARGET}':")
        display(df[TARGET].value_counts(normalize=True))
    else:
        print(f"\n La columna {TARGET} no existe en el dataset.")
```

```
Mounted at /content/drive
✅ Módulo de configuración cargado y estructura de carpetas asegurada.
Dataset cargado con 2824 filas y 92 columnas

Primeras filas del dataset:
```

	NFCSID	A1	A2	A3	B2_1	B2_2	B2_3	B2_4	B2_5	B2_20	...	G12	G13	H31
0	2.021010e+09	1.0	1.0	1.0	1.0	2.0	1.0	2.0	2.0	1.0	...	2.0	3.0	1.0
1	2.021010e+09	1.0	1.0	1.0	1.0	2.0	1.0	2.0	2.0	1.0	...	3.0	3.0	1.0
2	2.021010e+09	1.0	1.0	1.0	1.0	1.0	1.0	2.0	2.0	2.0	...	2.0	4.0	2.0
3	2.021010e+09	2.0	2.0	1.0	2.0	2.0	1.0	2.0	2.0	2.0	...	2.0	3.0	2.0
4	2.021010e+09	1.0	1.0	1.0	1.0	2.0	1.0	2.0	2.0	2.0	...	3.0	2.0	2.0

5 rows × 92 columns

```
 Información general:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2824 entries, 0 to 2823
Data columns (total 92 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   NFCSID       2824 non-null   float64
 1   A1           2824 non-null   float64
 2   A2           2824 non-null   float64
 3   A3           2824 non-null   float64
 4   B2_1         2824 non-null   float64
 5   B2_2         2824 non-null   float64
 6   B2_3         2824 non-null   float64
 7   B2_4         2824 non-null   float64
 8   B2_5         2824 non-null   float64
 9   B2_20        2824 non-null   float64
 10  B2_23        2824 non-null   float64
 11  B2_24        2824 non-null   float64
 12  B30          2824 non-null   float64
 13  B31          2824 non-null   float64
 14  B3           2824 non-null   float64
 15  B32          2824 non-null   float64
 16  B4           2824 non-null   float64
 17  B10          2824 non-null   float64
 18  B11          2824 non-null   float64
 19  B35          2824 non-null   float64
 20  B23          2824 non-null   float64
 21  B24          2824 non-null   float64
 22  B25          2824 non-null   float64
 23  B26          2824 non-null   float64
 24  C22_1        2824 non-null   float64
 25  C22_2        2824 non-null   float64
 26  C22_3        2824 non-null   float64
 27  C22_4        2824 non-null   float64
 28  C24          2824 non-null   float64
 29  C25          2824 non-null   float64
 30  C26          2824 non-null   float64
 31  C30          2824 non-null   float64
 32  C7           2824 non-null   float64
 33  D1_1         2824 non-null   float64
 34  D1_2         2824 non-null   float64
```

```
35  D2          2824 non-null   float64
36  D3          2824 non-null   float64
37  D21         2824 non-null   float64
38  D30         2824 non-null   float64
39  D5          2824 non-null   float64
40  E1_1        2824 non-null   float64
41  E5          2824 non-null   float64
42  E6          2824 non-null   float64
43  F30_1       2824 non-null   float64
44  F30_2       2824 non-null   float64
45  F30_3       2824 non-null   float64
46  F30_4       2824 non-null   float64
47  F30_5       2824 non-null   float64
48  F30_6       2824 non-null   float64
49  F30_7       2824 non-null   float64
```

El resultado muestra la distribución de la variable `proportion` en el DataFrame inicial, donde la clase 3.0 representa el 54.99 % de los datos, seguida de la clase 2.0 con el 27.34 %, la clase 4.0 con el 9.56 % y la clase 1.0 con el 8.11 %. Este desbalance de clases es clave a la hora de estratificar los datos o aplicar técnicas de balanceo para evitar sesgos en el modelo.

2.Definición de la variable FL_SCORE

Define el nombre de la variable que almacenará la puntuación de alfabetización financiera.

```
# FL_SCORE = suma de aciertos en preguntas de alfabetización financiera
literacy_items = ['G4','G5','G6','G7','G8','G11','G12','G13','G21']
literacy_present = [col for col in literacy_items if col in df.columns]
if literacy_present:
    df['FL_SCORE'] = df[literacy_present].sum(axis=1)
    print("Variable FL_SCORE creada correctamente.")
else:
    print("Atención: Ninguno de los literacy_items está en df. No se creó FL_SCORE.")

# PORTFOLIO_DIVERSITY = número de activos marcados (B2_)
asset_cols = [c for c in df.columns if c.startswith('B2_')]
if asset_cols:
    df['PORTFOLIO_DIVERSITY'] = (df[asset_cols] == 1).sum(axis=1)
    print("Variable PORTFOLIO_DIVERSITY creada correctamente.")
else:
    print("Atención: No se encontraron columnas B2_*. No se creó PORTFOLIO_DIVERSITY.")

# TRADER_SCORE = inversamente proporcional a B3 + ajuste por B31
if 'B3' in df.columns and 'B31' in df.columns:
    B3_norm = df['B3'].replace({98: np.nan, 99: np.nan})
    df['TRADER_SCORE'] = (5 - B3_norm).add(df['B31'].map({1:1, 2:0}), fill_value=0)
    print("Variable TRADER_SCORE creada correctamente.")
else:
    print("Atención: Las columnas B3 y/o B31 no existen. No se creó TRADER_SCORE.")

# **Verificamos que efectivamente se hayan agregado** las nuevas columnas a df
print("Columnas actuales de df:")
print(df.columns.tolist())
print("Total columnas en df:", len(df.columns))
print("Dimensiones tras derivar variables:", df.shape)
```

```
83  G13         2824 non-null   float64
84  H31         2824 non-null   float64
85  Age1        2824 non-null   float64
86  S_Gender2   2824 non-null   float64
87  S_Age       2824 non-null   float64
88  S_Ethnicity 2824 non-null   float64
89  S_Education 2824 non-null   float64
90  S_Income    2824 non-null   float64
91  B4_log      2824 non-null   float64
dtypes: float64(92)
memory usage: 2.0 MB
None
Variable FL_SCORE creada correctamente.
Variable PORTFOLIO_DIVERSITY creada correctamente.
Variable TRADER_SCORE creada correctamente.
Columnas actuales de df:
['INFCSID', 'A1', 'A2', 'A3', 'B2_1', 'B2_2', 'B2_3', 'B2_4', 'B2_5', 'B2_20', 'B2_23', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B9', 'B10', 'B11', 'B12', 'B13', 'B14', 'B15', 'B16', 'B17', 'B18', 'B19', 'B20', 'B21', 'B22', 'B23', 'B24', 'B25', 'B26', 'B27', 'B28', 'B29', 'B30', 'B31', 'B32', 'B33', 'B34', 'B35', 'B36', 'B37', 'B38', 'B39', 'B40', 'B41', 'B42', 'B43', 'B44', 'B45', 'B46', 'B47', 'B48', 'B49', 'B50', 'B51', 'B52', 'B53', 'B54', 'B55', 'B56', 'B57', 'B58', 'B59', 'B60', 'B61', 'B62', 'B63', 'B64', 'B65', 'B66', 'B67', 'B68', 'B69', 'B70', 'B71', 'B72', 'B73', 'B74', 'B75', 'B76', 'B77', 'B78', 'B79', 'B80', 'B81', 'B82', 'B83', 'B84', 'B85', 'B86', 'B87', 'B88', 'B89', 'B90', 'B91', 'FL_SCORE', 'PORTFOLIO_DIVERSITY', 'TRADER_SCORE']
Total columnas en df: 95
Dimensiones tras derivar variables: (2824, 95)
```

3. Definir variables de identificación

Crea la lista ID_VARS con las columnas que se usarán como identificadores únicos.

```
ID_VARS = ['NFCSID']
TARGET = 'B10'
WEIGHT_VARS = ['WGT1']  # Puedes quitar si no usas pesos

# Verificar que TARGET existe
if TARGET not in df.columns:
    raise ValueError(f"La columna TARGET '{TARGET}' no se encuentra en el dataset.")
```

4. Construcción de la matriz de características (X) y vector objetivo (y)

Elimina de df las columnas de identificación, objetivo y pesos; añade variables derivadas si existen; define y; y muestra las columnas resultantes de X.

```
# Construir X partiendo de df: quitamos ID, TARGET, WEIGHT_VARS
X = df.drop(columns=ID_VARS + [TARGET] + WEIGHT_VARS, errors='ignore')

# Forzar que las derivadas estén en X (si existen en df)
derived_vars = ['FL_SCORE', 'PORTFOLIO_DIVERSITY', 'TRADER_SCORE']
for v in derived_vars:
    if v in df.columns and v not in X.columns:
        X[v] = df[v]

# Definir y
y = df[TARGET]

# Verificar
print("\n Mostrar columnas del df para verificar:")
print(X.columns.tolist())
print("Total columnas en X:", len(X.columns))
print("Shape de X:", X.shape)
print("Shape de y:", y.shape)
```

```
 Mostrar columnas del df para verificar:
['A1', 'A2', 'A3', 'B2_1', 'B2_2', 'B2_3', 'B2_4', 'B2_5', 'B2_20', 'B2_23', 'B2_24',
Total columnas en X: 92
Shape de X: (2824, 92)
Shape de y: (2824,)
```

5. División estratificada de los datos

Realiza una división estratificada en tres pasos: extrae primero el conjunto de prueba (15%), luego el de validación (~15%), y mantiene el restante como entrenamiento; finalmente muestra las formas de cada subconjunto

```
# División estratificada en Train / Validation / Test

# Primer split: 15% test
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size=0.15, stratify=y, random_state=RANDOM_STATE
)
# Segundo split: ~15% validación
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.1765, stratify=y_temp, random_state=RANDOM_STATE
)

print(" Shapes resultantes:")
print(f"   • X_train: {X_train.shape},  y_train: {y_train.shape}")
print(f"   • X_val:   {X_val.shape},    y_val:   {y_val.shape}")
print(f"   • X_test:  {X_test.shape},   y_test:  {y_test.shape}")
```

```
 Shapes resultantes:
   • X_train: (1976, 92),  y_train: (1976,)
   • X_val:   (424, 92),    y_val:   (424,)
   • X_test:  (424, 92),   y_test:  (424,)
```

6. Instalación de pyarrow y guardado de los splits en formato Parquet

Se asegura de que pyarrow esté instalado y guarda X_train, y_train, X_val, y_val, X_test y y_test en archivos Parquet en la ruta configurada, confirmando al final la operación.

```
# Guardado de splits (VERSIÓN FINAL)

# Asegurarse de que la librería para Parquet esté instalada
!pip install pyarrow -q

# Guardar splits en Drive usando la ruta FINAL desde config.py
X_train.to_parquet(FINAL_SPLITS_DIR / 'X_train.parquet')
y_train.to_frame().to_parquet(FINAL_SPLITS_DIR / 'y_train.parquet')

X_val.to_parquet(FINAL_SPLITS_DIR / 'X_val.parquet')
```