

```

1 # -*- coding: utf-8 -*-
2
3 ##### SCRIPT 20 - PYTHON
4 # GENERACION LIBRERIA CASOS SIMILARES
5 # PARA EJECUTAR EN GPT SIN ACCESO A PYTHON
6 # =====
7
8
9
10 # =====
11 # Generar biblioteca de casos en Markdown para modo "light" sin Python
12 # =====
13
14 import pandas as pd
15 import numpy as np
16 import json
17 from pathlib import Path
18
19 # -----
20 # 1. Rutas y configuración básica
21 # -----
22
23 excel_path =
24     "/content/drive/MyDrive/_Pipeline_produccion_prediccion/df_test_train_v8.xlsx"
25 output_md_path =
26     "/content/drive/MyDrive/_Pipeline_produccion/casos_referencia_light.md"
27
28 # Columna de ID "de negocio" (si no existe se usará el índice del DataFrame)
29 id_col = "Identificador"           # si no existe, se usará el índice
30
31 # Columna con el cluster final (objetivo usado para entrenar RF)
32 cluster_col = "Cluster_6"
33
34 # Columna con el ahorro final post-licitación
35 ahorro_col = "Ahorro_final"
36
37 # Variables PRE-licitación según tu tabla (numéricas y categóricas)
38 pre_numeric_cols = [
39     "Presupuesto_licitacion_lote_c",      # presupuesto base de licitación (€)
40     "C_precio_p",                         # ponderación criterio precio (%)
41     "Plazo_m",                            # plazo del contrato (meses)
42     "N_lotes",                            # nº de lotes
43     "N_CPV",                             # nº de CPVs
44     "N_clasi_empresa",                  # nº clasificaciones exigidas
45     "C_juicio_valor_p_c",                # ponderación juicio de valor (%)
46     "Intervalo_lici_d_c",                # periodo de presentación ofertas (días)
47 ]
48
49 pre_categ_cols = [
50     "Tipo_de_contrato_c",
51     "Tipo_de_Administracion_c",
52     "Tipo_de_procedimiento_c",
53     "Tramitacion_c",
54     "Codigo_Postal_c",
55     "Tipo_ganador_lote_c",
56 ]
57
58 # Variables económico-competitivas principales para similitud
59 sim_vars = [
60     "Presupuesto_licitacion_lote_c",
61     "N_ofertantes",
62     "Plazo_m",
63     "Baja_p",
64     "C_precio_p",
65 ]
66
67 # Parámetro de saturación para similitud (%)
68 D_MAX = 5.0 # distancias >= D_MAX → similitud ≈ 0
69
70 # -----
71 # 2. Cargar DataFrame y verificar columnas

```

```

71 # -----
72
73 df = pd.read_excel(excel_path)
74
75 # Comprobaciones básicas
76 for col in [cluster_col, ahorro_col]:
77     if col not in df.columns:
78         raise ValueError(
79             f"La columna '{col}' no se encuentra en df_test_train_v8.xlsx. "
80             f"Ajusta 'cluster_col' / 'ahorro_col' en la cabecera del script."
81         )
82
83 # Avisos y limpieza de listas PRE
84 for col in pre_numeric_cols + pre_categ_cols:
85     if col not in df.columns:
86         print(f"AVISO: la columna PRE '{col}' no está en el DataFrame y se omitirá.")
87 pre_numeric_cols = [c for c in pre_numeric_cols if c in df.columns]
88 pre_categ_cols = [c for c in pre_categ_cols if c in df.columns]
89
90 # Avisos y limpieza de variables de similitud
91 for col in sim_vars:
92     if col not in df.columns:
93         print(f"AVISO: la columna de similitud '{col}' no está en el DataFrame y se
94         omitirá.")
95 sim_vars = [c for c in sim_vars if c in df.columns]
96
97 if not sim_vars:
98     raise ValueError("No queda ninguna variable en 'sim_vars'; revisa que existan en
99     el Excel.")
100
101 # -----
102
103 stats = {}
104 for col in sim_vars:
105     stats[col] = {
106         "mean": float(df[col].mean()),
107         "std": float(df[col].std(ddof=0)), # desviación poblacional
108     }
109
110 for col in sim_vars:
111     mean = stats[col]["mean"]
112     std = stats[col]["std"]
113     if std == 0 or np.isnan(std):
114         df[f"z_{col}"] = np.nan
115         print(f"AVISO: desviación nula/NaN en {col}; z_{col} se rellena con NaN.")
116     else:
117         df[f"z_{col}"] = (df[col] - mean) / std
118
119
120 def fmt_float(x):
121     """Formateo seguro para floats/ints → float con 6 decimales, o None si NaN."""
122     if pd.isna(x):
123         return None
124     return round(float(x), 6)
125
126
127 # -----
128 # 4. Cabecera del .md con instrucciones para el GPT
129 # -----
130
131 header_lines = []
132
133 header_lines.append("# Biblioteca de casos de referencia - Riesgo en licitaciones
134 # (modo light sin Python)\n")
135 header_lines.append("## Instrucciones para el GPT\n")
136 header_lines.append(
137     "Este archivo contiene una colección de casos históricos (TRAIN+TEST) del DSS de
riesgo en "
138     "licitaciones públicas. Cada caso está definido en un bloque `json` "
independiente.\n"

```

```

138 )
139
140 # Variables PRE
141 header_lines.append("### Variables de PRE-licitación a utilizar en la búsqueda\n")
142 header_lines.append(
143     "Debes buscar similitudes **solo** sobre variables de PRE-licitación. Para cada
144     caso, las "
145     "variables PRE disponibles (si existen en el dataset) son:\n"
146 )
147 if pre_numeric_cols:
148     header_lines.append(f"- Numéricas: {', '.join(pre_numeric_cols)}")
149 if pre_categ_cols:
150     header_lines.append(f"- Categóricas: {', '.join(pre_categ_cols)}")
151 header_lines.append("")
152 header_lines.append(
153     "En la búsqueda semántica, da más peso a las siguientes variables numéricas de
154     PRE, que son "
155     "especialmente representativas:\n"
156 )
157 header_lines.append("- `Presupuesto_licitacion_lote_c` (presupuesto base de
158     licitación del lote, €)")
159 header_lines.append("- `C_precio_p` (ponderación del precio en el criterio de
160     adjudicación, %)")
161 header_lines.append("- `Plazo_m` (plazo contractual en meses)\n")
162
163 # Variables económicas principales
164 header_lines.append("### Variables económico-competitivas principales\n")
165 header_lines.append(
166     "Cada caso incluye, además, las cinco variables económico-competitivas
167     principales, que usarás "
168     "tanto para la explicación como para el cálculo de similitud:\n"
169 )
170 for col in sim_vars:
171     header_lines.append(f"- `{col}`")
172 header_lines.append("- `Ahorro_final` (resultado post-licitación)\n")
173
174 # Medias y desviaciones
175 header_lines.append("### Coordenadas normalizadas para similitud\n")
176 header_lines.append(
177     "Para los casos que recuperes del índice (normalmente los **3 más parecidos** que
178     el RAG te "
179     "devuelva), utiliza las coordenadas normalizadas (`z_*`) de estas variables. Se
180     han calculado "
181     "con media 0 y desviación típica 1 sobre toda la base histórica.\n"
182 )
183 header_lines.append("Parámetros usados (media y desviación) por variable:\n")
184 for col in sim_vars:
185     header_lines.append(
186         f"- {col}: mean = {stats[col]['mean']:.6f}, std = {stats[col]['std']:.6f}"
187     )
188
189 header_lines.append("")
190 header_lines.append("### Cálculo del % de similitud entre un caso nuevo y un caso de
191     referencia\n")
192 header_lines.append("1. A partir de las respuestas del usuario, estima los valores
193     de:")
194 header_lines.append(f"    {', '.join(sim_vars)}")
195 header_lines.append(
196     "2. Calcula sus z-scores usando las medias y desviaciones anteriores:\n"
197     "    `z_x = (x - mean_x) / std_x`\n"
198 )
199 header_lines.append(
200     "3. Para cada caso de referencia recuperado del índice, usa los valores "
201     "`z_...` contenidos en el bloque JSON y computa la distancia euclídea:\n"
202 )
203 header_lines.append("    `d = sqrt( sum_k (z_k_nuevo - z_k_referencia)^2 )`\n")
204 header_lines.append(
205     "4. Convierte esta distancia en un porcentaje de similitud:\n"
206     "    `similitud = max(0, 100 * (1 - d / {D_MAX:.1f}))`\n"
207     "    donde `D_MAX = {D_MAX:.1f}` es un valor de saturación: distancias `d ≥

```

```

D_MAX` se consideran "
"similitud ≈ 0.\n"
)
header_lines.append(
    "Utiliza esta similitud solo para comparar **entre** los pocos casos devueltos
     por el RAG "
    "(por ejemplo, los 3 más cercanos) y selecciona el caso con mayor similitud como
     el más "
    "representativo."
)
header_lines.append(
    "Si los 3 casos pertenecen a clusters diferentes, adopta como *cluster de
     referencia* el "
    "cluster del caso con mayor similitud.\n"
)
header_lines.append("---\n")
header_lines.append("## Casos de referencia\n")
header_lines.append(
    "A continuación se listan todos los casos de la base de datos (TRAIN+TEST). "
    "Cada bloque `json` describe un expediente:\n"
)
header_text = "\n".join(header_lines)
# -----
# 5. Construcción de los bloques JSON por caso
# -----
lines = [header_text]

for idx, row in df.iterrows():
    # case_id: preferentemente Identificador si existe, si no, índice del DataFrame
    if id_col in df.columns:
        raw_id = row[id_col]
    else:
        raw_id = idx
    case_id = raw_id if not pd.isna(raw_id) else idx

    block = {
        "case_id": case_id,
        "dataset_index": int(idx),
        "cluster_final": row.get(cluster_col, None),
        "Ahorro_final": fmt_float(row.get(ahorro_col, np.nan)),
    }

    # Variables económico-competitivas + sus z-scores (si existen)
    for col in sim_vars:
        block[col] = fmt_float(row.get(col, np.nan))
        zcol = f"z_{col}"
        if zcol in df.columns:
            block[zcol] = fmt_float(row.get(zcol, np.nan))

    # Variables PRE-licitación agrupadas en 'pre_variables'
    pre_vars = {}
    for col in pre_numeric_cols + pre_categ_cols:
        val = row.get(col, np.nan)
        if isinstance(val, (np.floating, float, int, np.integer)):
            val_out = fmt_float(val)
        else:
            val_out = None if pd.isna(val) else val
        pre_vars[col] = val_out
    block["pre_variables"] = pre_vars

    # Serialización a JSON embebido en Markdown
    block_json = json.dumps(block, ensure_ascii=False, indent=2)
    lines.append(f"## CASE_ID = {case_id}\n\n```json\n{block_json}\n```\n")

md_text = "\n".join(lines)
Path(output_md_path).write_text(md_text, encoding="utf-8")

```

```
269     print(f"Archivo Markdown generado en:{output_md_path}")
```