

```

1 # =====
2 # 1. LIBRERÍAS
3 # =====
4 import pandas as pd
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 from sklearn.preprocessing import RobustScaler
9 from sklearn.linear_model import Ridge
10 from sklearn.ensemble import RandomForestRegressor
11 from sklearn.neural_network import MLPRegressor
12 from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
13
14 from xgboost import XGBRegressor
15
16
17 # =====
18 # 2. CARGAR DATASET
19 # =====
20 df = pd.read_excel("DATASET803.xlsx")
21
22 df['timestamp'] = pd.to_datetime(df['timestamp'])
23 df = df.sort_values('timestamp').reset_index(drop=True)
24
25
26 # =====
27 # 3. CREAR LAGS DE KWh
28 # =====
29 LAGS = [1, 3, 6] # 10, 30 y 60 minutos
30
31 for lag in LAGS:
32     df[f'KWh_lag_{lag}'] = df['KWh'].shift(lag)
33
34 df = df.dropna().reset_index(drop=True)
35
36
37 # =====
38 # 4. FEATURES
39 # =====
40 base_features = [
41     'MES', 'DIA', 'HORA', 'MINUTO',
42     'HA1700867', 'HSA1080498', 'MCA1081129', 'MEA1081132', 'MHA1081128',
43     'PPA1081120', 'TA1700867', 'TSA1080498',
44     'V1HPA1081112', 'V2HPA1081121', 'VBA1081118',
45     'VHA1081114', 'VHA1081122', 'VPA1081115',
46     'uso_modem', 'uso_portatil_solo', 'uso_pc_escritorio',
47     'uso_portatil_pantalla', 'uso_pc_portatil_pantalla',
48     'uso_ventilador', 'NEV', 'ILUM', 'Bombillo Horno',
49     'horno_micro', 'AA'
50 ]
51
52 lag_features = [f'KWh_lag_{lag}' for lag in LAGS]
53 features_lag = base_features + lag_features
54
55 target = 'KWh'
56
57
58 # =====
59 # 5. SPLIT TEMPORAL - DÍA SIGUIENTE
60 # =====
61 last_day = df['timestamp'].dt.date.unique()[-1]
62
63 train_df = df[df['timestamp'].dt.date < last_day]
64 test_df = df[df['timestamp'].dt.date == last_day]
65
66 X_train_base = train_df[base_features]
67 X_train_lag = train_df[features_lag]
68 y_train = train_df[target]
69
70 X_test_base = test_df[base_features]
71 X_test_lag = test_df[features_lag]
72 y_test = test_df[target]
73
74

```

```

75 # =====
76 # 6. EJE TEMPORAL (10 MIN)
77 # =====
78 x_time = test_df['HORA'] * 60 + test_df['MINUTO']
79 xticks = np.arange(0, 24*60, 10)
80 xtick_labels = [str(int(t//60)) if t % 60 == 0 else "" for t in xticks]
81
82
83 # =====
84 # 7. FUNCIÓN MÉTRICAS
85 # =====
86 def metrics(y_true, y_pred):
87     return {
88         "RMSE": np.sqrt(mean_squared_error(y_true, y_pred)),
89         "MAE": mean_absolute_error(y_true, y_pred),
90         "R2": r2_score(y_true, y_pred)
91     }
92
93
94 # =====
95 # 8. REGRESIÓN LINEAL (RIDGE)
96 # =====
97 scaler_r1 = RobustScaler()
98 X_train_r1 = scaler_r1.fit_transform(X_train_base)
99 X_test_r1 = scaler_r1.transform(X_test_base)
100
101 r1 = Ridge(alpha=1.0)
102 r1.fit(X_train_r1, y_train)
103 y_pred_r1 = r1.predict(X_test_r1)
104 m_r1 = metrics(y_test, y_pred_r1)
105
106
107 # =====
108 # 9. MLP
109 # =====
110 scaler_mlp = RobustScaler()
111 X_train_mlp = scaler_mlp.fit_transform(X_train_base)
112 X_test_mlp = scaler_mlp.transform(X_test_base)
113
114 mlp = MLPRegressor(
115     hidden_layer_sizes=(100, 50),
116     activation='relu',
117     max_iter=2000,
118     early_stopping=True,
119     validation_fraction=0.15,
120     random_state=42
121 )
122
123 mlp.fit(X_train_mlp, y_train)
124 y_pred_mlp = mlp.predict(X_test_mlp)
125 m_mlp = metrics(y_test, y_pred_mlp)
126
127
128 # =====
129 # 10. RANDOM FOREST + LAGS
130 # =====
131 rf_lag = RandomForestRegressor(
132     n_estimators=800,
133     max_depth=18,
134     min_samples_leaf=5,
135     min_samples_split=10,
136     random_state=42,
137     n_jobs=-1
138 )
139
140 rf_lag.fit(X_train_lag, y_train)
141 y_pred_rf_lag = rf_lag.predict(X_test_lag)
142 m_rf_lag = metrics(y_test, y_pred_rf_lag)
143
144
145 # =====
146 # 11. XGBOOST + LAGS
147 # =====
148 xgb = XGBRegressor(
149     n_estimators=400,
150     max_depth=6,

```

```

151     learning_rate=0.05,
152     subsample=0.8,
153     colsample_bytree=0.8,
154     objective='reg:squarederror',
155     n_jobs=-1,
156     random_state=42
157 )
158
159 xgb.fit(X_train_lag, y_train)
160 y_pred_xgb = xgb.predict(X_test_lag)
161 m_xgb = metrics(y_test, y_pred_xgb)
162
163
164 # =====
165 # 12. TABLA COMPARATIVA FINAL
166 # =====
167 results_df = pd.DataFrame([
168     {"Modelo": "Regresión Lineal (Ridge)", **m_rl},
169     {"Modelo": "MLP", **m_mlp},
170     {"Modelo": "Random Forest + Lags", **m_rf_lag},
171     {"Modelo": "XGBoost + Lags", **m_xgb},
172 ])
173
174 for c in ["RMSE", "MAE", "R2"]:
175     results_df[c] = results_df[c].round(4)
176
177 print("\n==== TABLA COMPARATIVA FINAL =====")
178 print(results_df)
179
180
181 # =====
182 # 13. GRÁFICA FINAL
183 # =====
184 plt.figure(figsize=(14,5))
185
186 plt.plot(x_time, y_test.values, label="Real", linewidth=1.2)
187 plt.plot(x_time, y_pred_rl, label="RL (Ridge)", linewidth=1)
188 plt.plot(x_time, y_pred_mlp, label="MLP", linewidth=1)
189 plt.plot(x_time, y_pred_rf_lag, label="RF + Lags", linewidth=1.4)
190 plt.plot(x_time, y_pred_xgb, label="XGBoost + Lags", linewidth=1.6)
191
192 plt.xticks(xticks, xtick_labels)
193 plt.xlabel("Hora del día")
194 plt.ylabel("Consumo [kWh]")
195 plt.title("Predicción del día siguiente - Comparación final de modelos")
196 plt.grid(True, alpha=0.3)
197 plt.legend()
198 plt.show()

```

===== TABLA COMPARATIVA FINAL =====

	Modelo	RMSE	MAE	R2
0	Regresión Lineal (Ridge)	0.0118	0.0085	0.4888
1	MLP	0.0198	0.0158	-0.4365
2	Random Forest + Lags	0.0101	0.0065	0.6219
3	XGBoost + Lags	0.0104	0.0067	0.6054

Predicción del día siguiente - Comparación final de modelos



```

1 # =====
2 # 14. DESIGNBUILDER -> 10 MIN + MÉTRICAS (100 % AUTÓNOMO)
3 # =====
4
5 import pandas as pd
6 import numpy as np
7 import matplotlib.pyplot as plt
8 from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
9
10
11 # -----
12 # 0. FUNCIÓN DE MÉTRICAS (MISMA DEFINICIÓN)
13 # -----
14 def metrics(y_true, y_pred):
15     return {
16         "RMSE": np.sqrt(mean_squared_error(y_true, y_pred)),
17         "MAE": mean_absolute_error(y_true, y_pred),
18         "R2": r2_score(y_true, y_pred)
19     }
20
21
22 # -----
23 # 1. CARGAR DATASET REAL (10 MIN)
24 # -----
25 df_real = pd.read_excel("DATASETB803.xlsx")
26
27 df_real['timestamp'] = pd.to_datetime(df_real['timestamp'])
28 df_real = df_real.sort_values('timestamp').reset_index(drop=True)
29
30
31 # -----
32 # 2. DEFINIR DÍA DE TEST (ÚLTIMO DÍA DEL DATASET REAL)
33 # -----
34 test_day = df_real['timestamp'].dt.date.unique()[-1]
35
36 real_test_10 = df_real[
37     df_real['timestamp'].dt.date == test_day
38 ][['timestamp', 'KWh']]
39
40
41 # -----
42 # 3. CARGAR DESIGNBUILDER
43 # -----
44 df_db = pd.read_excel("DesignBuilderSub.xlsx")
45
46

```

```

47 # -----
48 # 4. PARSEO CORRECTO DEL TIEMPO DESIGNBUILDER
49 #   Year 1 Jan 01 12:30 a.m.
50 # -----
51 df_db['timestamp'] = (
52     df_db['Date/Time']
53     .str.replace(r"Year\s+\d+\s+", "", regex=True)
54     .str.replace("a.m.", "AM", regex=False)
55     .str.replace("p.m.", "PM", regex=False)
56 )
57
58 df_db['timestamp'] = pd.to_datetime(
59     df_db['timestamp'],
60     format="%b %d %I:%M %p"
61 )
62
63 # Año ficticio solo para alinear
64 df_db['timestamp'] = df_db['timestamp'].apply(lambda x: x.replace(year=2024))
65
66
67 # -----
68 # 5. SELECCIONAR kWh (YA INTEGRADO)
69 # -----
70 df_db = df_db[['timestamp', 'kWh']]
71 df_db = df_db.sort_values('timestamp').reset_index(drop=True)
72
73
74 # -----
75 # 6. DESAGREGAR DE 30 MIN → 10 MIN (CONSERVA ENERGÍA)
76 # -----
77 rows_10min = []
78
79 for _, row in df_db.iterrows():
80     base_time = row['timestamp']
81     kWh_10 = row['kWh'] / 3.0
82
83     for i in range(3):
84         rows_10min.append({
85             "timestamp": base_time + pd.Timedelta(minutes=10 * i),
86             "KWh_DB": kWh_10
87         })
88
89 df_db_10 = pd.DataFrame(rows_10min)
90
91
92 # -----
93 # 7. FILTRAR DESIGNBUILDER AL MISMO DÍA DE TEST
94 # -----
95 db_test_10 = df_db_10[
96     df_db_10['timestamp'].dt.date == test_day
97 ]
98
99
100 # -----
101 # 8. ALINEAR REAL VS DESIGNBUILDER (10 MIN)
102 # -----
103 df_cmp = pd.merge(
104     real_test_10,
105     db_test_10,
106     on='timestamp',
107     how='inner'
108 )
109
110 print(f"✓ Registros alineados DB vs Real (10 min): {len(df_cmp)}")
111
112
113 # -----
114 # 9. MÉTRICAS DESIGNBUILDER
115 # -----
116 m_db = metrics(df_cmp['KWh'], df_cmp['KWh_DB'])
117
118 print("\n===== MÉTRICAS DESIGNBUILDER (10 MIN - DÍA TEST) =====")
119 for k, v in m_db.items():
120     print(f"{k}: {v:.4f}")
121

```

```

122
123 # -----
124 # 10. GRÁFICA REAL VS DESIGNBUILDER (EJE X EN HORAS)
125 # -----
126 x_time = (
127     df_cmp['timestamp'].dt.hour * 60 +
128     df_cmp['timestamp'].dt.minute
129 )
130
131 xticks = np.arange(0, 24*60, 60) # cada hora
132 xtick_labels = [str(int(t//60)) for t in xticks]
133
134 plt.figure(figsize=(14,5))
135
136 plt.plot(x_time, df_cmp['KWh'], label="Real", linewidth=1.5)
137 plt.plot(x_time, df_cmp['KWh_DB'], label="DesignBuilder", linewidth=1.5)
138
139 plt.xticks(xticks, xtick_labels)
140 plt.xlabel("Hora del día")
141 plt.ylabel("Consumo [kWh]")
142 plt.title("Real vs DesignBuilder - Resolución 10 minutos (Día test)")
143 plt.grid(True, alpha=0.3)
144 plt.legend()
145 plt.show()
146
147 # -----
148 # 11. EXPORTAR A EXCEL CON TIEMPO CORREGIDO
149 # -----
150 # (Opcional) crear columnas de apoyo para que Excel muestre mejor
151 df_cmp_out = df_cmp.copy()
152 df_cmp_out["Fecha"] = df_cmp_out["timestamp"].dt.date
153 df_cmp_out["Hora"] = df_cmp_out["timestamp"].dt.strftime("%H:%M")
154
155 db_test_out = db_test_10.copy()
156 db_test_out["Fecha"] = db_test_out["timestamp"].dt.date
157 db_test_out["Hora"] = db_test_out["timestamp"].dt.strftime("%H:%M")
158
159 real_test_out = real_test_10.copy()
160 real_test_out["Fecha"] = real_test_out["timestamp"].dt.date
161 real_test_out["Hora"] = real_test_out["timestamp"].dt.strftime("%H:%M")
162
163 output_file = f"DB_vs_Real_{test_day}_10min.xlsx"
164
165 with pd.ExcelWriter(output_file, engine="openpyxl") as writer:
166     df_cmp_out.to_excel(writer, sheet_name="Comparacion_Real_vs_DB", index=False)
167     db_test_out.to_excel(writer, sheet_name="DesignBuilder_10min", index=False)
168     real_test_out.to_excel(writer, sheet_name="Real_10min", index=False)
169
170 print(f"✅ Excel generado: {output_file}")
171

```

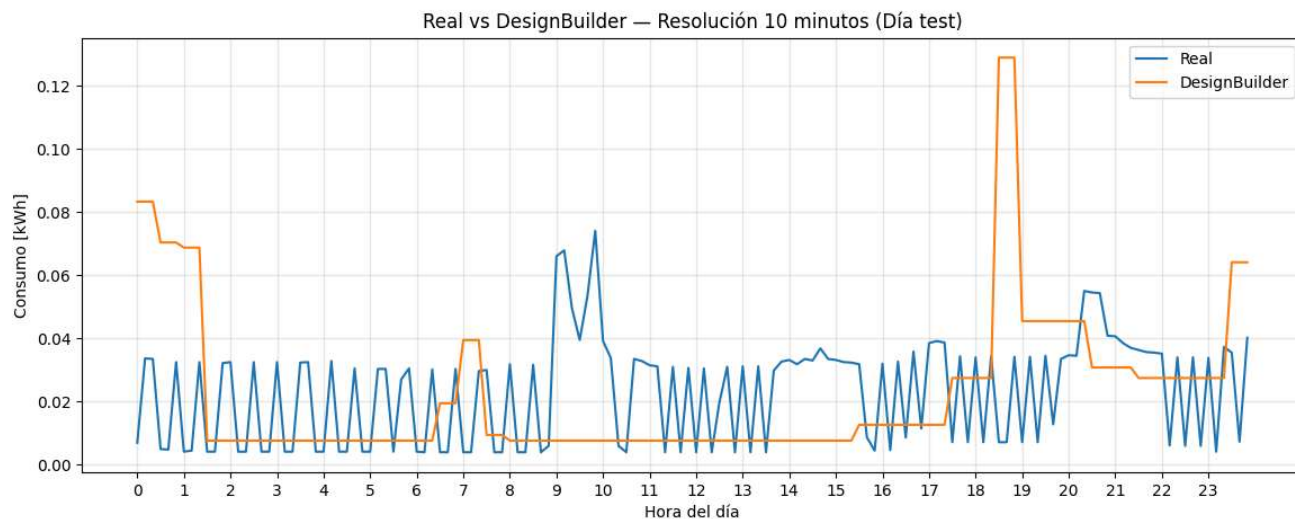
✓ Registros alineados DB vs Real (10 min): 144

===== MÉTRICAS DESIGNBUILDER (10 MIN - DÍA TEST) =====

RMSE: 0.0301

MAE: 0.0217

R2: -2.3380



✓ Excel generado: DB_vs_Real_2024-11-30_10min.xlsx

```

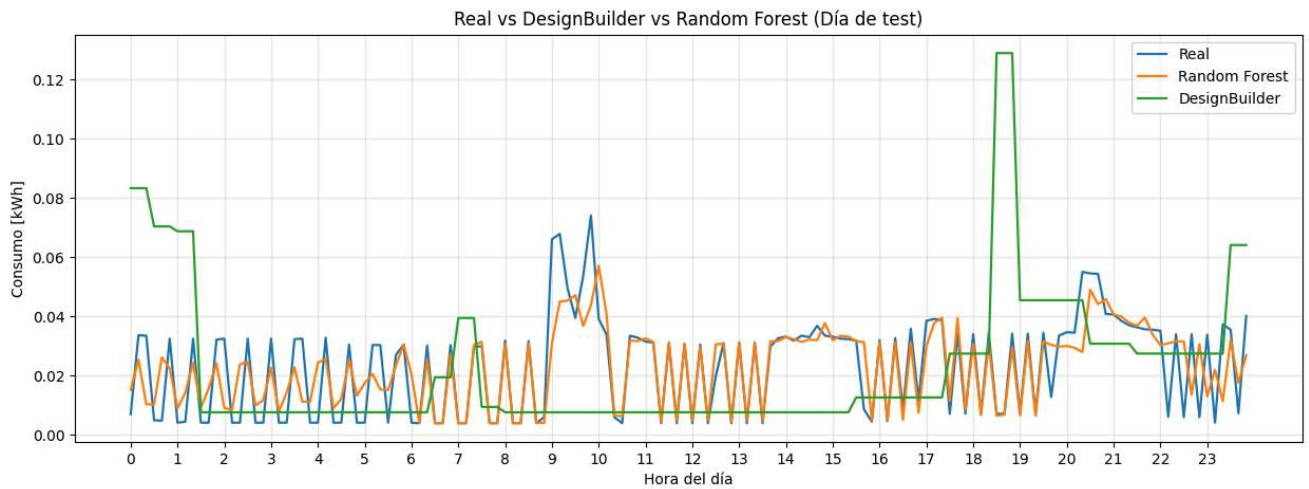
1 # =====
2 # GRÁFICA FINAL: REAL vs DESIGNBUILDER vs RANDOM FOREST
3 # =====
4
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8
9 # -----
10 # 1. DEFINIR DÍA DE TEST (ÚLTIMO DÍA)
11 # -----
12 test_day = df_real['timestamp'].dt.date.unique()[-1]
13
14
15 # -----
16 # 2. DATOS REALES DEL DÍA TEST (10 MIN)
17 # -----
18 real_day = df_real[df_real['timestamp'].dt.date == test_day][
19     ['timestamp', 'KWh']
20 ].reset_index(drop=True)
21
22
23 # -----
24 # 3. PREDICCIÓN RANDOM FOREST
25 #     y_pred_rf_lag ya corresponde a este día
26 # -----
27 rf_day = real_day.copy()
28 rf_day['RF'] = y_pred_rf_lag
29
30
31 # -----
32 # 4. DESIGNBUILDER YA ALINEADO (df_cmp)
33 #     df_cmp: timestamp | KWh | KWh_DB
34 # -----
35 db_day = df_cmp.copy()
36
37
38 # -----
39 # 5. EJE X EN HORAS DEL DÍA
40 # -----
41 x_rf = rf_day['timestamp'].dt.hour * 60 + rf_day['timestamp'].dt.minute
42 x_db = db_day['timestamp'].dt.hour * 60 + db_day['timestamp'].dt.minute
43
44 xticks = np.arange(0, 24*60, 60)
45 xtick_labels = [str(int(t // 60)) for t in xticks]

```

```

46
47
48 # -----
49 # 6. GRÁFICA FINAL
50 # -----
51 plt.figure(figsize=(15,5))
52
53 plt.plot(x_rf, rf_day['KWh'], label="Real", linewidth=1.6)
54 plt.plot(x_rf, rf_day['RF'], label="Random Forest", linewidth=1.6)
55 plt.plot(x_db, db_day['KWh_DB'], label="DesignBuilder", linewidth=1.6)
56
57 plt.xticks(xticks, xtick_labels)
58 plt.xlabel("Hora del día")
59 plt.ylabel("Consumo [kWh]")
60 plt.title("Real vs DesignBuilder vs Random Forest (Día de test)")
61 plt.grid(True, alpha=0.3)
62 plt.legend()
63 plt.show()
64

```



```

1 import numpy as np
2 import pandas as pd
3
4 # =====
5 # FUNCIÓN % ERROR POR PUNTO
6 # =====
7 def ape_percent(y_true, y_pred, eps=1e-10):
8     y_true = np.array(y_true, dtype=float)
9     y_pred = np.array(y_pred, dtype=float)
10    denom = np.maximum(np.abs(y_true), eps)
11    return (np.abs(y_pred - y_true) / denom) * 100.0
12
13 # =====
14 # DATAFRAME CON LOS DATOS DE LA GRÁFICA
15 # =====
16 df_grafica = pd.DataFrame({
17     "timestamp": test_df["timestamp"].values, # tiempo real
18     "min_dia": x_time.values if hasattr(x_time, "values") else np.array(x_time),
19     "Real": y_test.values,
20
21     "RL_Ridge": y_pred_rl,
22     "MLP": y_pred_mlp,
23     "RF_Lags": y_pred_rf_lag,
24     "XGBoost_Lags": y_pred_xgb,
25 })
26
27 # Orden por timestamp (por seguridad)
28 df_grafica = df_grafica.sort_values("timestamp").reset_index(drop=True)
29
30 # =====
31 # % ERROR POR PUNTO (APE%) PARA CADA MODELO
32 # =====
33 df_grafica["APE_RL_%"] = ape_percent(df_grafica["Real"], df_grafica["RL_Ridge"])
34 df_grafica["APE_MLP_%"] = ape_percent(df_grafica["Real"], df_grafica["MLP"])

```



```

35 df_grafica["APE_RF_%"] = ape_percent(df_grafica["Real"], df_grafica["RF_Lags"])
36 df_grafica["APE_XGB_%"] = ape_percent(df_grafica["Real"], df_grafica["XGBoost_Lags"])
37
38 # =====
39 # EXPORTAR A EXCEL
40 # =====
41 with pd.ExcelWriter("datos_grafica_modelos.xlsx", engine="openpyxl") as writer:
42     df_grafica.to_excel(writer, index=False, sheet_name="Datos_grafica")
43
44 print("✅ Excel generado: datos_grafica_modelos.xlsx")
45 print(df_grafica.head())
46

```

✅ Excel generado: datos_grafica_modelos.xlsx

	timestamp	min_dia	Real	RL_Ridge	MLP	RF_Lags	\
0	2024-11-30 00:00:00	0	0.007000	0.022765	0.011269	0.015241	
1	2024-11-30 00:10:00	10	0.033667	0.022766	0.007804	0.025416	
2	2024-11-30 00:20:00	20	0.033500	0.022761	0.003917	0.010383	
3	2024-11-30 00:30:00	30	0.005000	0.022801	-0.002820	0.010295	
4	2024-11-30 00:40:00	40	0.004833	0.022547	-0.006475	0.026206	

	XGBoost_Lags	APE_RL_%	APE_MLP_%	APE_RF_%	APE_XGB_%
0	0.017246	225.220450	60.982101	117.728144	146.373617
1	0.027831	32.376780	76.820445	24.508243	17.333024
2	0.007171	32.057329	88.308906	69.006747	78.595313
3	0.005472	356.021867	156.390000	105.895747	9.438235
4	0.028829	366.488898	233.961806	442.188670	496.452877

```

1 # =====
2 # DEMANDA COMPUTACIONAL DE LOS MODELOS
3 # Tiempo de entrenamiento y tiempo de inferencia
4 # =====
5
6 # -----
7 # 1. LIBRERÍAS
8 # -----
9 import time
10 import numpy as np
11 import pandas as pd
12
13 from sklearn.preprocessing import RobustScaler
14 from sklearn.linear_model import Ridge
15 from sklearn.ensemble import RandomForestRegressor
16 from sklearn.neural_network import MLPRegressor
17 from xgboost import XGBRegressor
18
19
20 # -----
21 # 2. CARGAR DATASET
22 # -----
23 df = pd.read_excel("DATASET803.xlsx")
24 df["timestamp"] = pd.to_datetime(df["timestamp"])
25 df = df.sort_values("timestamp").reset_index(drop=True)
26
27
28 # -----
29 # 3. CREAR LAGS
30 # -----
31 LAGS = [1, 3, 6]
32
33 for lag in LAGS:
34     df[f"KWh_lag_{lag}"] = df["KWh"].shift(lag)
35
36 df = df.dropna().reset_index(drop=True)
37
38
39 # -----
40 # 4. FEATURES
41 # -----
42 base_features = [
43     'MES', 'DIA', 'HORA', 'MINUTO',
44     'HA1700867', 'HSA1080498', 'MCA1081129', 'MEA1081132', 'MHA1081128',
45     'PPA1081120', 'TA1700867', 'TSA1080498',
46     'V1HPA1081112', 'V2HPA1081121', 'VBA1081118',
47     'VHA1081114', 'VHA1081122', 'VPA1081115',
48     'uso_modem', 'uso_portatil_solo', 'uso_pc_escritorio',
49     'uso_portatil_pantalla', 'uso_pc_portatil_pantalla'.

```

```

14     'uso_ventilador', 'NEV', 'ILUM', 'Bombillo Horno',
50     'horno micro', 'AA'
51 ]
52 ]
53
54 lag_features = [f"KWh_lag_{lag}" for lag in LAGS]
55 features_lag = base_features + lag_features
56
57 target = "KWh"
58
59
60 # -----
61 # 5. SPLIT TEMPORAL
62 # -----
63 last_day = df["timestamp"].dt.date.unique()[-1]
64
65 train_df = df[df["timestamp"].dt.date < last_day]
66 test_df = df[df["timestamp"].dt.date == last_day]
67
68 X_train_base = train_df[base_features]
69 X_test_base = test_df[base_features]
70
71 X_train_lag = train_df[features_lag]
72 X_test_lag = test_df[features_lag]
73
74 y_train = train_df[target]
75
76
77 # -----
78 # 6. ESCALADO (RL y MLP)
79 # -----
80 scaler_rl = RobustScaler()
81 X_train_rl = scaler_rl.fit_transform(X_train_base)
82 X_test_rl = scaler_rl.transform(X_test_base)
83
84 scaler_mlp = RobustScaler()
85 X_train_mlp = scaler_mlp.fit_transform(X_train_base)
86 X_test_mlp = scaler_mlp.transform(X_test_base)
87
88
89 # -----
90 # 7. FUNCIÓN PARA MEDIR TIEMPOS
91 # -----
92 def medir_tiempos(crear_modelo, Xtr, ytr, Xte, n_runs=3):
93     tiempos_train = []
94     tiempos_pred = []
95
96     for _ in range(n_runs):
97         modelo = crear_modelo()
98
99         t0 = time.perf_counter()
100         modelo.fit(Xtr, ytr)
101         tiempos_train.append(time.perf_counter() - t0)
102
103         t1 = time.perf_counter()
104         modelo.predict(Xte)
105         tiempos_pred.append(time.perf_counter() - t1)
106
107     return np.mean(tiempos_train), np.mean(tiempos_pred)
108
109
110 # -----
111 # 8. DEFINICIÓN DE MODELOS
112 # -----
113 def RL():
114     return Ridge(alpha=1.0)
115
116 def MLP():
117     return MLPRegressor(
118         hidden_layer_sizes=(100, 50),
119         activation="relu",
120         max_iter=2000,
121         early_stopping=True,
122         validation_fraction=0.15,
123         random_state=42
124     )
125

```

```

126 def RF():
127     return RandomForestRegressor(
128         n_estimators=800,
129         max_depth=18,
130         min_samples_leaf=5,
131         min_samples_split=10,
132         random_state=42,
133         n_jobs=-1
134     )
135
136 def XGB():
137     return XGBRegressor(
138         n_estimators=400,
139         max_depth=6,
140         learning_rate=0.05,
141         subsample=0.8,
142         colsample_bytree=0.8,
143         objective="reg:squarederror",
144         random_state=42,
145         n_jobs=-1
146     )
147
148
149 # -----
150 # 9. EJECUCIÓN
151 # -----
152 N_RUNS = 3
153
154 resultados = []
155
156 t_train, t_pred = medir_tiempos(RL, X_train_rl, y_train, X_test_rl, N_RUNS)
157 resultados.append(["Regresión Lineal (Ridge)", t_train, t_pred])
158
159 t_train, t_pred = medir_tiempos(MLP, X_train_mlp, y_train, X_test_mlp, N_RUNS)
160 resultados.append(["MLP", t_train, t_pred])
161
162 t_train, t_pred = medir_tiempos(RF, X_train_lag, y_train, X_test_lag, N_RUNS)
163 resultados.append(["Random Forest + Lags", t_train, t_pred])
164
165 t_train, t_pred = medir_tiempos(XGB, X_train_lag, y_train, X_test_lag, N_RUNS)
166 resultados.append(["XGBoost + Lags", t_train, t_pred])

```