

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Mounted at /content/gdrive

```
# Importing the necessary libraries
import time
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df = pd.read_csv("/content/gdrive/MyDrive/datathon_ws/df_NUEVO (1).csv").copy()
```

```
df.head()
```

	idListaCobro	idCredito	consecutivoCobro	idBanco_x	montoExigible	montoCobrar	montoCobrado	fechaCobroBanco	idRespue
0	155938	738973	41396434	2	622.87	622.87	0.00	NaN	
1	155938	739017	41396435	2	1069.11	1069.11	0.00	NaN	
2	155939	739185	41396436	2	4340.83	4340.83	4340.83	02/01/2025	
3	155940	732324	41396437	2	2134.21	2134.21	0.00	NaN	
4	155940	737028	41396438	2	815.76	815.76	0.00	NaN	

5 rows × 24 columns

```
df = pd.read_csv("/content/gdrive/MyDrive/datathon_ws/df_NUEVO (1).csv").copy()
```

```
df.head()
```

	idListaCobro	idCredito	consecutivoCobro	idBanco_x	montoExigible	montoCobrar	montoCobrado	fechaCobroBanco	idRespue
0	155938	738973	41396434	2	622.87	622.87	0.00	NaN	
1	155938	739017	41396435	2	1069.11	1069.11	0.00	NaN	
2	155939	739185	41396436	2	4340.83	4340.83	4340.83	02/01/2025	
3	155940	732324	41396437	2	2134.21	2134.21	0.00	NaN	
4	155940	737028	41396438	2	815.76	815.76	0.00	NaN	

5 rows × 24 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2114142 entries, 0 to 2114141
Data columns (total 24 columns):
#   Column                Dtype
---  ----
0   idListaCobro          int64
1   idCredito             int64
2   consecutivoCobro      int64
3   idBanco_x            int64
4   montoExigible         float64
5   montoCobrar           float64
6   montoCobrado          float64
7   fechaCobroBanco       object
8   idRespuestaBanco      float64
9   caso_exitoso          int64
10  fechaCreacionLista     object
11  idBanco_y             int64
12  fechaEnvioCobro        object
13  idEmisora             int64
14  id_estrategia         int64
15  servicio              int64
```

```

16 tiempo_feedback      int64
17 Hora_inicio          object
18 Hora_fin             object
19 pagare               float64
20 capital              float64
21 fechaAperturaCredito object
22 CobroExito           float64
23 CobroDevuelta        float64
dtypes: float64(8), int64(10), object(6)
memory usage: 387.1+ MB

# =====
# 1) Imports y carga de datos
# =====
import pandas as pd
from pathlib import Path
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.metrics import roc_auc_score, classification_report

# -----
# Carga tu muestra (CSV o XLSX)  ->  df
# -----

# =====
# 2) Feature engineering ligero
# =====
# => Fechas -> datetime
# -----
# Fechas -> tipo datetime (formato dd/mm/yyyy)
# -----
df["fechaEnvioCobro"] = pd.to_datetime(df["fechaEnvioCobro"],
                                     dayfirst=True, errors="coerce")
df["fechaAperturaCredito"] = pd.to_datetime(df["fechaAperturaCredito"],
                                             dayfirst=True, errors="coerce")

# => Días transcurridos desde origen del crédito
df["dias_desde_origen"] = (df["fechaEnvioCobro"] - df["fechaAperturaCredito"]).dt.days

# => Hora del envío de cobro (ej. 0-23)
df["hora_envio"] = pd.to_datetime(df["Hora_inicio"]).dt.hour

# =====
# 3) Selección de columnas (sin fugas de información)
# =====
cols_drop = [
    "idListaCobro", "montoCobrado", "fechaCobroBanco", "idRespuestaBanco",
    "fechaCreacionLista", "idBanco_x", "idBanco_y", "Hora_inicio", "Hora_fin",
    "fechaAperturaCredito", "fechaEnvioCobro" # reemplazadas por 'hora_envio' y 'dias_desde_origen'
]
df = df.drop(columns=cols_drop)


# Columnas restantes ➡ variables de entrada
cat_cols = ["idEmisora", "id_estrategia", "servicio"]
num_cols = [
    "consecutivoCobro", "montoExigible", "montoCobrar",
    "tiempo_feedback", "hora_envio", "pagare", "capital",
    "dias_desde_origen", "CobroExito", "CobroDevuelta"
]
target = "caso_exitoso"

# =====
# 4) Pre-procesador: One-Hot para categóricas, passthrough numéricas
# =====
pre = ColumnTransformer(
    transformers=[
        ("cat", OneHotEncoder(handle_unknown="ignore", sparse_output=True)
, cat_cols)
    ],
    remainder="passthrough" # deja las columnas numéricas tal cual
)
```

```
# =====
# 5) Modelo XGBoost + Pipeline
# =====
xgb = XGBClassifier(
    n_estimators=600,
    max_depth=6,
    learning_rate=0.08,
    subsample=0.8,
    colsample_bytree=0.8,
    eval_metric="logloss",
    random_state=42,
    n_jobs=-1
)

pipe = Pipeline([
    ("prep", pre),
    ("clf", xgb)
])

# =====
# 6) Split train / test y entrenamiento
# =====
X_train, X_test, y_train, y_test = train_test_split(
    df[cat_cols + num_cols], df[target],
    test_size=0.20, random_state=42, stratify=df[target]
)
```

 <ipython-input-56-9486403f1217>:34: UserWarning: Could not infer format, so each element will be parsed individually, falling back to object dtype. This behavior can be avoided by specifying a format.
df["hora_envio"] = pd.to_datetime(df["Hora_inicio"]).dt.hour

✓ Entrenamiento de modelo


```
# 1) Nuevo modelo con peso de clases
ratio = (y_train == 0).sum() / (y_train == 1).sum() # ≈23
xgb = XGBClassifier(
    n_estimators=800, max_depth=6, learning_rate=0.06,
    subsample=0.8, colsample_bytree=0.8,
    scale_pos_weight=ratio,
    eval_metric='aucpr', random_state=42, n_jobs=-1
)
pipe.set_params(clf=xgb).fit(X_train, y_train)

# 2) Curva P-R para elegir umbral
from sklearn.metrics import precision_recall_curve, auc
proba = pipe.predict_proba(X_test)[:,-1]
prec, rec, thr = precision_recall_curve(y_test, proba)

# Selecciona el threshold donde recall ~0.60 (ejemplo)
target_recall = 0.60
idx = next(i for i, r in enumerate(rec) if r < target_recall) - 1
best_thr = thr[idx]

print("Nuevo threshold:", round(best_thr,3),
      " Precision:", round(prec[idx],2),
      " Recall:", round(rec[idx],2))

# 3) Clasificaciones finales
y_pred = (proba >= best_thr).astype(int)
print(classification_report(y_test, y_pred))
```

 Nuevo threshold: 0.845 Precision: 0.46 Recall: 0.6

	precision	recall	f1-score	support
0	0.98	0.97	0.98	405177
1	0.46	0.60	0.52	17652
accuracy			0.95	422829
macro avg	0.72	0.78	0.75	422829

weighted avg 0.96 0.95 0.96 422829

✓ Prueba de modelo para un ID random

```
import numpy as np
import pandas as pd

# -----
# 1) Elegir un registro aleatorio
# (si solo quieres segundo cobro, descomenta la línea filtrada)
# -----
df_test_case = df[df["consecutivoCobro"] == 2].sample(1, random_state=None)
df_test_case = df.sample(1, random_state=None)

# -----
# 2) Preparar la fila base SIN 'id_estrategia'
# -----
base_row = df_test_case.iloc[0].copy()
if "id_estrategia" not in base_row.index:
    raise KeyError("'id_estrategia' no está en la fila: revisa columnas.")
base_row = base_row.drop(labels="id_estrategia")

# -----
# 3) Lista real de tus 12 estrategias (ajusta a tus IDs)
# Si no estás seguro, puedes extraer las 12 principales del DataFrame:
estrategias_12 = sorted(df["id_estrategia"].unique())[:12]
# -----

def predict_for_all_strategies(pipeline, base_row, strategies,
                              cat_cols, num_cols):
    """Devuelve prob_exito por estrategia para una sola fila."""
    rows = []
    for strat in strategies:
        r = base_row.copy()
        r["id_estrategia"] = strat
        rows.append(r)

    batch = pd.DataFrame(rows)
    # reindex garantiza orden y crea NaN (numéricas) si faltan columnas
    batch = batch.reindex(columns=cat_cols + num_cols)

    # Probabilidades
    probs = pipeline.predict_proba(batch)[ :, 1]
    return (
        pd.DataFrame({"id_estrategia": strategies, "prob_exito": probs})
        .sort_values("prob_exito", ascending=False)
        .reset_index(drop=True)
    )

# -----
# 4) Ejecutar y mostrar
# -----
resultado = predict_for_all_strategies(
    pipeline=pipe,
    base_row=base_row,
    strategies=estrategias_12,
    cat_cols=cat_cols,
    num_cols=num_cols
)

print("Crédito ejemplo (idCredito={}):".format(df_test_case["idCredito"].values[0]))
print(resultado)
```

```
➡ Crédito ejemplo (idCredito=664299):
   id_estrategia  prob_exito
0              1    0.054097
1              2    0.052852
2             11    0.050263
3              4    0.047007
4              7    0.046924
5              6    0.045366
6              9    0.045108
```

7

5 0.039510

✓ Prediccion de probabilidades

```
import numpy as np
import pandas as pd

# 1) Elegir un registro aleatorio (o consecutivoCobro == 2 si lo prefieres)
df_test_case = df.sample(1, random_state=None)
# df_test_case = df[df["consecutivoCobro"] == 2].sample(1, random_state=None)

# Extrae el idCredito del registro
credit_id = df_test_case["idCredito"].values[0]

# 2) Preparar la fila base SIN 'id_estrategia'
base_row = df_test_case.iloc[0].copy()
if "id_estrategia" not in base_row.index:
    raise KeyError("'id_estrategia' no está en la fila: revisa columnas.")
base_row = base_row.drop(labels="id_estrategia")

# 3) Lista real de tus 12 estrategias
estrategias_12 = sorted(df["id_estrategia"].unique())[:12]

def predict_for_all_strategies(pipeline, base_row, strategies,
                               cat_cols, num_cols):
    """Devuelve DataFrame con id_estrategia y prob_exito."""
    rows = []
    for strat in strategies:
        r = base_row.copy()
        r["id_estrategia"] = strat
        rows.append(r)
    batch = pd.DataFrame(rows)
    batch = batch.reindex(columns=cat_cols + num_cols)
    probs = pipeline.predict_proba(batch)[:, 1]
    return pd.DataFrame({
        "id_estrategia": strategies,
        "prob_exito": probs
    })

# 4) Ejecutar y enriquecer con idCredito
resultado = predict_for_all_strategies(
    pipeline=pipe,
    base_row=base_row,
    strategies=estrategias_12,
    cat_cols=cat_cols,
    num_cols=num_cols
)

# Añade idCredito y reordena columnas
resultado["idCredito"] = credit_id
resultado = resultado[["idCredito", "id_estrategia", "prob_exito"]]

# Muestra el DataFrame final
print(resultado)
```

```
↗
```

	idCredito	id_estrategia	prob_exito
0	745301	1	0.302873
1	745301	2	0.342250
2	745301	4	0.303285
3	745301	5	0.250488
4	745301	6	0.307992
5	745301	7	0.227003
6	745301	9	0.277258
7	745301	11	0.224693

```
resultado.info()
```

```
↗
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 2 columns):
#   Column      Non-Null Count  Dtype
---  -
#   Column      Non-Null Count  Dtype
```

```

0    id_estrategia    8 non-null    int64
1    prob_exito      8 non-null    float32
dtypes: float32(1), int64(1)
memory usage: 228.0 bytes

```

✓ Añadir columnas de prediccion de probabilidades a cada registro

```

import numpy as np
import pandas as pd

# 1) Define tus 12 estrategias
estrategias_12 = sorted(df["id_estrategia"].unique())[:12]

# 2) Asegúrate de que 'id_estrategia' esté en cat_cols
if "id_estrategia" not in cat_cols:
    cat_cols = cat_cols + ["id_estrategia"]

# 3) Para cada estrategia, calcula las probabilidades y añade una columna nueva
for strat in estrategias_12:
    # Crea una copia ligera donde fijas esa estrategia para todas las filas
    tmp = df.copy()
    tmp["id_estrategia"] = strat

    # Reindexa según el orden de columnas que espera el pipeline
    batch = tmp.reindex(columns=cat_cols + num_cols)

    # Predice la probabilidad de éxito (clase 1)
    probs = pipe.predict_proba(batch)[:, 1]

    # Añade la columna al DataFrame original
    df[f"prob_exito_{strat}"] = probs

# Ahora df tiene, además de sus columnas originales, una columna por cada estrategia:
#   prob_exito_<id_estrategia>
print(df.head())

...

```

```
df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2114142 entries, 0 to 2114141
Data columns (total 23 columns):
#   Column          Dtype
---  ---
0    idCredito      int64

```

```

1 consecutivoCobro      int64
2 montoExigible         float64
3 montoCobrar           float64
4 caso_exitoso          int64
5 idEmisora             int64
6 id_estrategia         int64
7 servicio              int64
8 tiempo_feedback       int64
9 pagare                float64
10 capital              float64
11 CobroExito            float64
12 CobroDevuelta         float64
13 dias_desde_origen     int64
14 hora_envio            int32
15 prob_exit_1           float32
16 prob_exit_2           float32
17 prob_exit_4           float32
18 prob_exit_5           float32
19 prob_exit_6           float32
20 prob_exit_7           float32
21 prob_exit_9           float32
22 prob_exit_11          float32
dtypes: float32(8), float64(6), int32(1), int64(8)
memory usage: 298.4 MB

```

✓ Añade dos columnas a df:

- best_strategy: id de la estrategia (1,2,4,5,6,7,9,11) con mayor utilidad.
- best_utility: valor de la utilidad correspondiente.

```

import pandas as pd

def seleccionar_mejor_estrategia(df):
    """
    Añade dos columnas a df:
    - best_strategy: id de la estrategia (1,2,4,5,6,7,9,11) con mayor utilidad.
    - best_utility: valor de la utilidad correspondiente.
    """
    M = df['montoCobrar']
    Ce = df['CobroExito']
    Cf = df['CobroDevuelta']
    λ = 1.0

    # Lista de estrategias disponibles (coincide con sufijos en prob_exit_*)
    estrategias = [1,2,4,5,6,7,9,11]

    # Calculamos utilidades en forma vectorizada
    utilidades = {}
    for s in estrategias:
        p = df[f'prob_exit_{s}']
        U_s = p * M - λ * (p * Ce + (1 - p) * Cf)
        utilidades[s] = U_s

    util_df = pd.DataFrame(utilidades)          # columnas = {1,2,4,...}
    df['best_strategy'] = util_df.idxmax(axis=1) # el índice (s) con mayor U_s
    df['best_utility'] = util_df.max(axis=1)     # valor de U_s máximo

    return df

# df = pd.read_csv('tus_datos.csv')
df2 = seleccionar_mejor_estrategia(df)
# Ahora df['best_strategy'] y df['best_utility'] están poblados.

```



```

-----
KeyError                                Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
    3804         try:
-> 3805             return self._engine.get_loc(casted_key)
    3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'prob_exito_1'

```

The above exception was the direct cause of the following exception:

```

KeyError                                Traceback (most recent call last)
-----
3 frames
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
    3810         ):
    3811             raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
    3813         except TypeError:
    3814             # If we have a listlike key, _check_indexing_error will raise

KeyError: 'prob_exito_1'

```

Next steps: [Explain error](#)

df2.head()



	idCredito	consecutivoCobro	montoExigible	montoCobrar	caso_exitoso	idEmisora	id_estrategia	servicio	tiempo_feedbac
0	738973	41396434	622.87	622.87	0	5	9	1	180
1	739017	41396435	1069.11	1069.11	0	5	9	1	180
2	739185	41396436	4340.83	4340.83	1	5	9	1	180
3	732324	41396437	2134.21	2134.21	0	5	9	1	180
4	737028	41396438	815.76	815.76	0	5	9	1	180

5 rows × 25 columns

import pandas as pd

```

def seleccionar_mejor_estrategia(df):
    """
    Añade tres columnas a df:
    - best_strategy: id de la estrategia con mayor utilidad.
    - best_utility: utilidad de esa estrategia.
    - real_utility: utilidad de la estrategia efectivamente usada (id_estrategia).
    """
    M = df['montoCobrar']
    Ce = df['CobroExito']
    Cf = df['CobroDevuelta']
    λ = 1.0

    # Estrategias candidatas (coinciden con sufijos en prob_exito_*)
    estrategias = [1,2,4,5,6,7,9,11]

    # 1) Calcular utilidades vectorizadas para cada estrategia
    utilidades = {}
    for s in estrategias:
        p_s = df[f'prob_exito_{s}']
        U_s = p_s * M - λ * (p_s * Ce + (1 - p_s) * Cf)
        utilidades[s] = U_s

    util_df = pd.DataFrame(utilidades, index=df.index)

    # 2) Mejor estrategia por fila

```



```

df['best_strategy'] = util_df.idxmax(axis=1)
df['best_utility'] = util_df.max(axis=1)

# 3) Utilidad de la estrategia real (id_estrategia)
# Primero, revisamos que todas las reales estén en candidatas
missing = set(df['id_estrategia'].unique()) - set(estrategias)
if missing:
    raise ValueError(f"Estrategias reales no contempladas: {missing}")

# Ahora extraemos de util_df la columna que coincida con id_estrategia
# y la guardamos en real_utility
df['real_utility'] = df.apply(
    lambda row: utilidades[row['id_estrategia']].loc[row.name],
    axis=1
)

return df

# Ejemplo de uso:
df3 = seleccionar_mejor_estrategia(df)
# Ahora df2 contiene best_strategy, best_utility y real_utility.

```

✓ Se agrega una columna de incremento en ganancia que se calcula al dividir la mejor utilidad entre la utilidad real

```
df3.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2114142 entries, 0 to 2114141
Data columns (total 26 columns):
 #   Column                Dtype
---  -
 0   idCredito             int64
 1   consecutivoCobro      int64
 2   montoExigible         float64
 3   montoCobrar           float64
 4   caso_exitoso          int64
 5   idEmisora             int64
 6   id_estrategia         int64
 7   servicio              int64
 8   tiempo_feedback       int64
 9   pagare                float64
10   capital               float64
11   CobroExito            float64
12   CobroDevuelta         float64
13   dias_desde_origen     int64
14   hora_envio            int32
15   prob_exit_1           float32
16   prob_exit_2           float32
17   prob_exit_4           float32
18   prob_exit_5           float32
19   prob_exit_6           float32
20   prob_exit_7           float32
21   prob_exit_9           float32
22   prob_exit_11          float32
23   best_strategy         int64
24   best_utility          float64
25   real_utility          float64
dtypes: float32(8), float64(8), int32(1), int64(9)
memory usage: 346.8 MB

```

```
import numpy as np
```

```


# Evitamos división por cero convirtiendo ceros en NaN
df['utilidad_real_nonzero'] = df['real_utility'].replace(0, np.nan)

# Calculamos el incremento
df['incremento_ganancia'] = df['best_utility'] / df['utilidad_real_nonzero']

# (Opcional) Si quieres que en lugar de NaN aparezca un cero cuando utilidad_real era 0:
df['incremento_ganancia'] = df['incremento_ganancia'].fillna(0)


```

```
df['incremento_ganancia'].describe()
```




incremento_ganancia	
count	2.114142e+06
mean	1.341169e+00
std	6.064812e+01
min	-8.524075e+04
25%	1.111595e+00
50%	1.277486e+00
75%	1.481894e+00
max	5.389683e+03

```
df.columns
```



```
Index(['idCredito', 'consecutivoCobro', 'montoExigible', 'montoCobrar',
      'caso_exitoso', 'idEmisora', 'id_estrategia', 'servicio',
      'tiempo_feedback', 'pagare', 'capital', 'CobroExito', 'CobroDevuelta',
      'dias_desde_origen', 'hora_envio', 'prob_exit_1', 'prob_exit_2',
      'prob_exit_4', 'prob_exit_5', 'prob_exit_6', 'prob_exit_7',
      'prob_exit_9', 'prob_exit_11', 'best_strategy', 'best_utility',
      'real_utility', 'utilidad_real_nonzero', 'incremento_ganancia'],
      dtype='object')
```

```
df.head()
```



	idCredito	consecutivoCobro	montoExigible	montoCobrar	caso_exitoso	idEmisora	id_estrategia	servicio	tiempo_feedback
0	738973	41396434	622.87	622.87	0	5	9	1	180
1	739017	41396435	1069.11	1069.11	0	5	9	1	180
2	739185	41396436	4340.83	4340.83	1	5	9	1	180
3	732324	41396437	2134.21	2134.21	0	5	9	1	180
4	737028	41396438	815.76	815.76	0	5	9	1	180

5 rows × 28 columns

```
# 1. Selecciona las columnas de interés en un nuevo DataFrame
```

```
cols = [
    'idCredito',
    'id_estrategia',
    'best_strategy',
    'best_utility',
    'real_utility',
    'incremento_ganancia'
]
df_export = df[cols].copy()
```

```
# 2. Comprueba que se ven bien
```

```
df_export.head()
```

	idCredito	id_estrategia	best_strategy	best_utility	real_utility	incremento_ganancia	
0	738973	9	4	167.766917	127.475233	1.316075	
1	739017	9	5	283.036571	218.480336	1.295478	
2	739185	9	1	3439.631086	3185.209769	1.079876	
3	732324	9	6	549.388294	480.747936	1.142778	
4	737000	9	11	147.771556	100.777000	1.151000	

```
df_export.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2114142 entries, 0 to 2114141
Data columns (total 6 columns):
#   Column                Dtype
---  ----
0   idCredito              int64
1   id_estrategia          int64
2   best_strategy          int64
3   best_utility           float64
4   real_utility           float64
5   incremento_ganancia    float64
dtypes: float64(3), int64(3)
memory usage: 96.8 MB
```

Exportamos un excel con la columna idcredito y las columnas 'best_strategy',

- ✓ 'best_utility', 'real_utility', 'utilidad_real_nonzero', 'incremento_ganancia' en un nuevo dataframe