# Diagnóstico Avanzado de Data Leakage

## Taxonomia de Data Leakage

### Leakage temporal

- **Definición formal:** Cuando dividimos nuestro dataset o hacemos k-fold validation, puede pasar que tengamos datos tanto de eventos del pasado como del futuro, haciendo que el modelo memorize lo que pasará en el futuro en lugar de predecirlo.
- **Ejemplo concreto:** Cuando queremos crear un modelo que prediga las acciones del SP500 o de bitcoin. Lo ideal sería por ejemplo que el modelo aprendiera solamente con datos de 2020 a 2024 y nosotros probaramos con datos del 2025, el problema es que muchas veces no tienen en cuenta la componente temporal y simplemente dividen el dataset como cualquier otro conjunto. Esto ocasiona que el modelo si vea datos de 2025, de tal forma que uno puede creer que funciona muy bien pero la realidad es que no.
- **Por qué es problemático:** El problema viene de hacernos creer que nuestro modelo es bastante bueno y que podemos predecir el futuro. Pero por ejemplo cuaando empezemos a usar el modelo con datos de 2026 no nos debería de sorprender que el modelo sea bastante malo.
- **Cómo detectarlo:** El modelo funciona sorprendentemente bien, esto es bastante raro en series temporales. También si tuviste la mala idea de ponerlo en producción deberías de ver que funciona mucho peor a los resultados en test. Otra técnica es el análisis de los residuos entre el valor predicho y el real, se debería de ver como forma de ruido.
- **Cómo prevenirlo:** Determinar un tiempo fijo, en donde de ese tiempo para atras solo sea para entrenar y de ese tiempo para adelante sea para probar el modelo

### Leakage por Preprocesamiento

- **Definición formal:** Es cuando le hacemos un preprocesamiento a los datos (imputar valores, transformaciones para el sesgo, escalamiento, etc) pero calculamos los valores con los que se hace el preprocesamiento (medias, desviaciones estandar, modas) con datos que deberían ir solo en test.
- **Ejemplo concreto:** Imaginemos que tenemos una columna en donde nos falta 30% de nuestros datos, esta columna es numérica y decidimos imputarlos con la media. Decidimos calcular esta media sobre TODOS nuestros datos e imputar, después contentos dividimos ya nuestro dataset en una parte para entrenamiento y otra para prueba, este es un claro ejemplo de data leakage por

preprocesamiento.

- **Por qué es problemático:** El problema radica en que los datos de entrenamiento se ven influenciados por los datos de test, cuando esto no debería de pasar. Para el modelo cuando vea los datos de test deberían de serle completamente desconocidos, en cambio con data leakage por preprocesamiento aprendió algunas características de los datos de test y cuando es momento de inferir se le hace bastante facil obtener un altísimo rendimiento.
- **Cómo detectarlo:** El modelo tiene un rendimiento demasiado bueno para ser verdad en los datos de test, practicamente igual al de entrenamiento. También si tenemos la mala suerte de ponerlo en producción se encontrará en esta ocasión con datos que ahora si nunca había visto y veremos que el rendimiento baja a comparacion del de test, esto en si no es malo si no es que se reporta normalmente cuanta confianza tiene el modelo, una confianza que no es verdadera.
- **Cómo prevenirlo:** Definir Pipelines con skilearn por ejemplo ayuda bastamte, en la pipeline decidimos que transformaciones hacer y una vez que lo tenemos listo hay que dividir el dataset en entrenamiento y prueba. Es tan sencillo como usar pipeline.fit para entrenamiento y pieline.predict para los datos de prueba.

## Leakage por Target Encoding

- **Definición formal:** Target encoding es una técnica que transforma variables categóricas en valores numéricos calculando estadísticas de la variable objetivo (por ejemplo, la media) para cada categoría. El leakage ocurre cuando estas estadísticas se calculan usando todo el dataset, incluyendo los datos de prueba, filtrando información del target que el modelo no debería conocer durante el entrenamiento.

- **Ejemplo concreto:** Imaginemos que tenemos un dataset de clientes y queremos predecir si compran o no un producto. Tenemos una variable categórica llamada ciudad. Si calculamos la media de compra por ciudad usando todos los datos el modelo conocera la información de los clientes de prueba. Por ejemplo si en la ciudad de Hermosillo el 80% de los clientes compran el modelo va a aprenderse la proporción en vez de patrones generales.

- **Por qué es problemático:** El modelo parece muy preciso en entrenamiento y validación, pero falla en datos nuevos. Esto se debe a que aprendió información del target que solo estaba disponible en el dataset completo, lo que produce una evaluación inflada y resultados poco realistas.

- **Cómo detectarlo:** Comparar desempeño entre entrenamiento y validación con datos “nuevos”: si hay un salto grande, podría indicar leakage. Revisar el pipeline de preprocesamiento: asegurarse de que estadísticas del target se calculen únicamente sobre el conjunto de entrenamiento, no usando datos futuros o de

prueba.

- **Cómo detectarlo:** Comparar desempeño entre entrenamiento y validación con datos “nuevos”: si hay un salto grande, podría indicar leakage. Revisar el pipeline de preprocesamiento: asegurarse de que estadísticas del target se calculen únicamente sobre el conjunto de entrenamiento, no usando datos futuros o de prueba.
- **Cómo prevenirlo:** Calcular el target encoding solo con datos de entrenamiento. Para cada fold en validación cruzada, calcular las estadísticas solo con el fold de entrenamiento, no con el de prueba. Aplicar técnicas de suavizado o regularización en el target encoding para reducir la dependencia de categorías con pocas muestras.

## Leakage por Features que contienen información futura

- **Definición formal:** Ocurre cuando una característica (feature) incluye información que solo estaría disponible después del momento en que se hace la predicción. Es decir, el modelo recibe datos “del futuro” que no existirían en un escenario real, lo que provoca predicciones irreales y sobreajuste.
- **Ejemplo concreto:** Si queremos predecir si un cliente pedirá un prestamo mañana e incluimos como feature saldo final del cliente en el banco mañana, estamos usando información que solo debería de conocerse después de la predicción, causando data leakage.
- **Por qué es problemático:** El principal problema es la falsa ilusión de que el modelo funciona muy bien, pero en realidad no tiene capacidad de generalizar.
- **Cómo detectarlo:** Revisamos el dataset y vemos si todas las características estarán disponibles cuando nos llegue un dato nuevo. También si hay alguna característica que tiene una correlacion muy alta con el target se podria estar filtrando información.
- **Cómo prevenirlo:** Asegurarse de usar solo información disponible hasta el momento de la predicción. Implementar pipelines que respeten el tiempo.

## Leakage por duplicación de registros

- **Definición formal:** Ocurre cuando hay instancias iguales o casi identicas tanto en el conjunto de entrenamiento como en el conjunto de test. Esto provoca que cuando el modelo aprendio a predecir correctamente una instancia en entrenamiento y luego viene una de test igual el modelo la predecira correctamente pero no por la generalizacion sino porque ya la ha visto.
- **Ejemplo concreto:** Supongamos que queremos predecir si una persona tiene cancer de pulmón. Ese paciente esta duplicado y esta tanto en train y en test. Entonces el modelo se fijará en las particularidades del paciente mas que en que

es lo que hace que una persona tiene cancer
- **Por qué es problemático:** Oculta problemas en la selección de características o el preprocesamiento. Da metricas infladas lo que da una falsa sensación de que el modelo funciona correctamente.
- **Cómo detectarlo:** Revisar duplicados antes de separar nuestros datos de entrenamiento y de prueba. Analizar correlaciones altas entre train y test.
- **Cómo prevenirlo:** Limpiando el dataset de duplicados antes de separar datos de entrenamiento y de prueba. También asegurarse que el mismo cliente quede ya sea en entrenamiento o en prueba.

## Leakage por estratificación incorrecta

- **Definición formal:** Sucede cuando los datos de entrenamiento y prueba no se dividen correctamente, rompiendo la independencia entre ellos. Esto puede suceder si se estratifica mal el dataset o si hay duplicados o grupos relacionados que aparecen en ambos conjuntos, filtrando información del target entre entrenamiento y prueba.
- **Ejemplo concreto:** Por ejemplo supongamos qu tenemos un dataset de clasificación. El objetivo es clasificar tipos de flores. Ademas supongamos que las clases no estan muy balanceadas, Si dividimos sin estratificar puede pasar que no haya instancias de una clase en el dataset de entrenamiento, así muy difícil para el modelo predecirlo.
- **Por qué es problemático:** Si no utilizamos las metricas correctas el modelo puede parecer que funciona, ya que la clase mas grande eclipsa a las demás, dandonos una falsa impresión de que el modelo clasifica muy bien.
- **Cómo detectarlo:** Revisar las distribuciones de las clases o de las características a estratificar, si la proporcion tanto en train como en test no es igual o muy parecida a la proporcion de todos los datos puede haber un problema de mal estratificado.
- **Cómo prevenirlo:** Utilizar estratificación al dividir los datos, por ejemplo con train_test_split en scikit-learn, para mantener la proporción de cada clase. Verificar siempre la distribución de clases y grupos después de hacer el split, antes de entrenar el modelo.

## Bibliografía

1. Amrane, S. (2025, febrero 15). *Data leakage in ML and DL: Understanding, detecting and fixing*. Medium. https://sirineamrane.medium.com/data-leakage-in-ai-understanding-detecting-and-fixing-cb2f87bd97b4

2. Built In. (2023). *Data Leakage in Machine Learning: Detect and Minimize Risk*. https://builtin.com/machine-learning/data-leakage

3. Microsoft Data Science Blog. (s.f.). *Stop the spill: The blueprint for eradicating data leakage*. Medium. https://medium.com/data-science-at-microsoft/stop-the-spill-the-blueprint-for-eradicating-data-leakage-6f924e543a95

4. Cross Validated. (2022, March 8). *Target encoding in test data and target leakage*. StackExchange. https://stats.stackexchange.com/questions/567095/target-encoding-in-test-data-and-target-leakage

# Metodología para evitar el Data Leakage

1. Dividir el dataset para entrenamiento y prueba

- El paso mas importante para evitar el data leakage.
- Asegurarnos de dividirlo estratíficadamente si es que lo necesita.
- Asegurarnos de que hay una clara división en el tiempo para el caso de datos temporales. En pocas palabras, train es el pasado mientras test son los valores mas recientes.

2. Preprocesamiento después de dividir el dataset

- Asegurarse que la normalización, estandarización e imputación se calculen solo usando datos de entrenamiento.
- Comprobar que no se calculen estadísticas globales antes de dividir los datos.

3. Orden de las operaciones en el pipeline de preprocesamiento:

- Verificar que todas las transformaciones y creaciones de feature se hagan después de dividir el dataset y solo se haga con la informacion de entrenamiento
- Confirmar que la creación de features no filtra información en el conjunto de prueba por medio de las correlaciones.

4. Características con correlación muy alta con el target

- Revisar las correlaciones entre cada feature y el target
- Revisar aquellas que tienen una correlacion mayor a 0.95 y analizar si hay información filtrada del target

5. Características con información temportal o futura

- Quedarse con características qu solo estarían disponibles con datos futuros.
- Confirmar que las secuencias de tiempo respetan la causalidad y no se utilice información futura

6. Duplicación de registros

- Detectar si hay registros idénticos o casi idénticos en entrenamiento y prueba.

- Si hay entidades repetidas usar splits por grupo para evitar que la información se filtre.

7. Codificación

- Verificar que target encoding u otros encodings que dependen del target se calculen solo en la parte de entrenamiento.
- Verificcar que en prueba se apliquen las estadísticas aprendidas en entrenamiento, sin recalcular usando los datos de prueba.

Esto es el pipeline general, a continuación se pone en forma de checklist.

# Checklist para evitar el Data Leakage

## 1. Dividir el dataset para entrenamiento y prueba

- ☐ Dividir el dataset (paso más importante para evitar el data leakage).
- ☐ Dividir estratificadamente si es necesario.
- ☐ Asegurar clara división temporal para datos secuenciales: train = pasado, test = valores más recientes.

## 2. Preprocesamiento después de dividir el dataset

- ☐ Normalización, estandarización e imputación calculadas solo con datos de entrenamiento.
- ☐ No calcular estadísticas globales antes de dividir los datos.

## 3. Orden de las operaciones en el pipeline de preprocesamiento

- ☐ Todas las transformaciones y creación de features después de dividir el dataset y solo con información de entrenamiento.
- ☐ Confirmar que la creación de features no filtra información en el conjunto de prueba por medio de correlaciones.

## 4. Características con correlación muy alta con el target

- ☐ Revisar correlaciones entre cada feature y el target.
- ☐ Analizar features con correlación mayor a 0.95 para detectar filtración de información del target.

## 5. Características con información temporal o futura

- ☐ Mantener solo características que estarían disponibles con datos futuros.

- ☐ Confirmar que las secuencias de tiempo respeten la causalidad y no se use información futura.

## 6. Duplicación de registros

- ☐ Detectar registros idénticos o casi idénticos en entrenamiento y prueba.
- ☐ Si hay entidades repetidas, usar splits por grupo para evitar filtración de información.

## 7. Codificación

- ☐ Verificar que target encoding u otros encodings dependientes del target se calculen solo en la parte de entrenamiento.
- ☐ En prueba, aplicar estadísticas aprendidas en entrenamiento sin recalcular usando datos de prueba.

# Caso de Estudio

Se presenta un ejemplo de un pipeline sencillo que contenga diferentes tipos de leakage. Para esto usaremos un dataset clásico, el del titanic.

In [1]:
```
# Importacion de librerias
import pandas as pd
import numpy as np
import seaborn as sns
```

In [2]:
```
# Cargamos nuestro dataset
df = sns.load_dataset('titanic')
```

In [3]:
```
# Vemos una vista de lo que tiene este dataset
print("Tamaño: ", df.shape)

print(df.info())

print(df.head())
```

```
Tamaño:  (891, 15)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype   
---  ------       --------------  -----   
 0   survived     891 non-null    int64   
 1   pclass       891 non-null    int64   
 2   sex          891 non-null    object  
 3   age          714 non-null    float64 
 4   sibsp        891 non-null    int64   
 5   parch        891 non-null    int64   
 6   fare         891 non-null    float64 
 7   embarked     889 non-null    object  
 8   class        891 non-null    category
 9   who          891 non-null    object  
 10  adult_male   891 non-null    bool    
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object  
 13  alive        891 non-null    object  
 14  alone        891 non-null    bool    
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
None
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third   
1         1       1  female  38.0      1      0  71.2833        C  First   
2         1       3  female  26.0      0      0   7.9250        S  Third   
3         1       1  female  35.0      1      0  53.1000        S  First   
4         0       3    male  35.0      0      0   8.0500        S  Third   

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

In [4]:
```
# Vamos a ver como se distribuye la variable objetivo.
print(df['survived'].value_counts(normalize=True) * 100)
```

```
survived
0    61.616162
1    38.383838
Name: proportion, dtype: float64
```

La mayoría no sobrevivieron. Veamos que columnas tienen variables nulas

In [5]:
```
print(df.isna().sum())
```

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

La variable deck tiene demasiados datos faltantes, vamos a eliminarla. En cambio a age y embark_town vamos a imputarlas

In [6]:
```
# Eliminamos la variable deck
df.drop('deck', axis=1, inplace=True)
```

In [7]:
```
# Imputamos la columna age
mean_age = df['age'].mean()
df['age'] = df['age'].fillna(mean_age)

# Imputamos la columna embarked
mode_embark_town = df['embark_town'].mode()[0]
df['embark_town'] = df['embark_town'].fillna(mode_embark_town)

# Imputamos la columna embarked
mode_embarked = df['embarked'].mode()[0]
df['embarked'] = df['embarked'].fillna(mode_embarked)
```

Ahora si, ya tenemos preprocesamos nuestros datos, ya no faltan valores nulos. Vemos que hay características como repetidas, es decir que nos dicen lo mismo, vamos a quedarnos con las que estan mas codificadas.

In [8]:
```
drop_cols = ['class', 'who', 'adult_male', 'embark_town', 'alone']
df.drop(drop_cols, axis=1, inplace=True)
df.columns
```

Out[8]:
```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'alive'],
      dtype='object')
```

Ahora vamos a codificar las variables sex y embarked para que puedan funcionar con un random forest

In [9]:
```
# Codificar sex como binaria
df['sex'] = df['sex'].map({'male': 0, 'female': 1})
```

```
# Codificar alive como binaria
df['alive'] = df['alive'].map({'no': 0, 'yes': 1})

# One-Hot Encoding para 'embarked'
df = pd.get_dummies(df, columns=['embarked'], drop_first=True)
```

Creamos nuestro conjunto de entrenamiento

In [10]:
```
X = df.drop('survived', axis=1)
y = df['survived']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random
```

Creamos nuestro modelo random forest

In [11]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

RFC = RandomForestClassifier(random_state=42)

RFC.fit(X_train, y_train)
y_pred = RFC.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy: ", acc)
```

```
Accuracy:  1.0
```

Obtuvimos un accuracy perfecto, el problema es que hicimos data leakage, hicimos todo el preprocesamiento antes de dividir nuestro dataset, por lo que nuestros datos de test fueron filtrados y el modelo sabe sobre ello. Otro detalle importante, no nos aseguramos de que no hubiera características que no deberían de venir cuando viene un nuevo dato, si queremos saber si una persona vive o no con nuestro modelo, es porque no tenemos ni la menor idea si esta vivo o muerto, pero dejamos la variable alive, lo cual no ayuda para nada, ya le estamos diciendo al modelo que solo se tiene que fijar en esa variable y nos la va a predecir perfectamente.

In [12]:
```
# Cargamos el dataset
df = sns.load_dataset('titanic')

# Eliminar columnas innecesarias o con demasiados nulos
df.drop('deck', axis=1, inplace=True)

# Eliminar columnas redundantes
drop_cols = ['class', 'who', 'adult_male', 'embark_town', 'alone', 'alive']
df.drop(drop_cols, axis=1, inplace=True)

# Separamos train y test antes de imputar
X = df.drop('survived', axis=1)
y = df['survived']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, stratify=y, random_state=42, test_size=0.2
```

```
)

# Imputamos usando solo estadísticas del conjunto de entrenamiento
mean_age = X_train['age'].mean()
mode_embarked = X_train['embarked'].mode()[0]

# Aplicar imputación a train y test
X_train['age'] = X_train['age'].fillna(mean_age)
X_test['age'] = X_test['age'].fillna(mean_age)

X_train['embarked'] = X_train['embarked'].fillna(mode_embarked)
X_test['embarked'] = X_test['embarked'].fillna(mode_embarked)

# Codificamos variables
X_train['sex'] = X_train['sex'].map({'male': 0, 'female': 1})
X_test['sex'] = X_test['sex'].map({'male': 0, 'female': 1})
X_train = pd.get_dummies(X_train, columns=['embarked'], drop_first=True)
X_test = pd.get_dummies(X_test, columns=['embarked'], drop_first=True)


# Entrenamos el modelo
RFC = RandomForestClassifier(random_state=42)
RFC.fit(X_train, y_train)

# Predicciones
y_pred = RFC.predict(X_test)

# Evaluación
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy SIN leakage: {acc:.4f}")
```

```
Accuracy SIN leakage: 0.8101
```

Ahora si se corrigió y vemos un accuracy mas realista, si hubieramos seguido la checklist desde un inicio, esta parte es la que nos hubiera prevenido a tener el data leakage.

- [ ] Normalización, estandarización e imputación calculadas solo con datos de entrenamiento.
- [ ] No calcular estadísticas globales antes de dividir los datos.
- [ ] Mantener solo características que estarían disponibles con datos futuros.