Examen Parcial 2

Dulce María Mayorga

15 de mayo de 2025

1 Examen Parcial 2

1.0.1 About Dataset

Machine Predictive Maintenance Classification Dataset Since real predictive maintenance datasets are generally difficult to obtain and in particular difficult to publish, we present and provide a synthetic dataset that reflects real predictive maintenance encountered in the industry to the best of our knowledge.

The dataset consists of 10 000 data points stored as rows with 14 features in columns

- UID: unique identifier ranging from 1 to 10000
- productID: consisting of a letter L, M, or H for low (50% of all products), medium (30%), and high (20%) as product quality variants and a variant-specific serial number
- air temperature [K]: generated using a random walk process later normalized to a standard deviation of 2 K around 300 K
- process temperature [K]: generated using a random walk process normalized to a standard deviation of 1 K, added to the air temperature plus 10 K.
- rotational speed [rpm]: calculated from powepower of 2860 W, overlaid with a normally distributed noise
- torque [Nm]: torque values are normally distributed around 40 Nm with an Ïf = 10 Nm and no negative values.
- tool wear [min]: The quality variants H/M/L add 5/3/2 minutes of tool wear to the used tool in the process. and a
- 'machine failure' label that indicates, whether the machine has failed in this particular data point for any of the following failure modes are true.

1.0.2 Important :

There are two Targets - Do not make the mistake of using one of them as feature, as it will lead to leakage. - Target : Failure or Not - Failure Type : Type of Failure

> **Nota importante:** Para este examen vamos a utilizar la variable objetivo, **Target** es binaria si la maquina fallo o no, la otra variable hay que eliminarla ya que puede causar data leakage
>
> **Pregunta 1:** Investigar y explicar en un parrafo en formato markdown que es el **data leakage** y como esta puede afectar a nuestro modelo de ML, especificamente en este caso porque hay que eliminar una de las variables objetivo

¿Qué es el data leakage y cómo afecta a un modelo de ML?

El *data leakage* pasa cuando un modelo aprende con información que no debería conocer, como datos del futuro o variables que están directamente relacionadas con lo que queremos predecir. Esto hace que el modelo parezca muy bueno al entrenarse, pero falle al aplicarse a nuevos datos. En este modelo específico, tener tanto la columna `Target` como `Failure Type` puede causar leakage, porque ambas contienen información del fallo. Para evitarlo, hay que elegir solo una como variable objetivo y eliminar la otra.

1.1 Carga de datos

1.2 Preprocesamiento de datos

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   UDI                      10000 non-null  int64
 1   Product ID               10000 non-null  object
 2   Type                     10000 non-null  object
 3   Air temperature [K]      10000 non-null  float64
 4   Process temperature [K]  10000 non-null  float64
 5   Rotational speed [rpm]   10000 non-null  int64
 6   Torque [Nm]              10000 non-null  float64
 7   Tool wear [min]          10000 non-null  int64
 8   Target                   10000 non-null  int64
 9   Failure Type             10000 non-null  object
dtypes: float64(3), int64(4), object(3)
memory usage: 781.4+ KB
```

1.2.1 Validacion de datos Null o dupicados

```
UDI                        0
Product ID                 0
Type                       0
Air temperature [K]        0
Process temperature [K]    0
Rotational speed [rpm]     0
Torque [Nm]                0
Tool wear [min]            0
Target                     0
Failure Type               0
dtype: int64

np.int64(0)
```

1.2.2 Entendiendo los tipos de datos y valores unicos

```
UDI                          int64
Product ID                  object
Type                        object
Air temperature [K]        float64
Process temperature [K]    float64
Rotational speed [rpm]       int64
Torque [Nm]                float64
Tool wear [min]              int64
Target                       int64
Failure Type                object
dtype: object
```

```
UDI                        10000
Product ID                 10000
Type                           3
Air temperature [K]           93
Process temperature [K]       82
Rotational speed [rpm]       941
Torque [Nm]                  577
Tool wear [min]              246
Target                         2
Failure Type                   6
dtype: int64
```

```
Type
L    6000
M    2997
H    1003
Name: count, dtype: int64
```

Pregunta 2: Que entiendes de el analisis anterior

Del análisis previo, se puede concluir que los datos están limpios, ya que no hay valores nulos ni registros duplicados. A su vez, se pudo identificar los tipos de datos que tenemos tales como datos numéricos y categóricos.

Además, al observar los valores únicos, se nota que hay variables como `UDI` y `Product ID` que son identificadores únicos y probablemente no sean útiles para el modelo, por lo que es conveniente eliminarlos.

En la columna `Type` se encuentran tres tipos de productos: `L`, `M` y `H`, con una distribución desigual (mayoría de tipo `L`). Este desbalance es importante y se debe tomar en cuenta cuando entrenemos nuestro modelo, ya que puede afectar el rendimiento del clasificador.

1.2.3 Eliminar identificadores

Elimina las variables - UDI - Product ID - Failure Type

Tip: utiliza la función `drop`de pandas

```
['Type', 'Air temperature [K]', 'Process temperature [K]', 'Rotational speed
[rpm]', 'Torque [Nm]', 'Tool wear [min]', 'Target']
```

1.2.4 Verificamos si el dataset esta balancado

Tip: Utiliza la funcion de pandas `value_counts()`

```
Target
0    9661
1     339
Name: count, dtype: int64
```

> **Pregunta 3:** Esta balanceado o no el dataset, que estrategias hay para un dataset no balanceado

Al revisar los valores de la variable `Target`, vemos que el dataset está desbalanceado: hay 9661 casos de clase 0 y solo 339 de clase 1. Esto puede ser un problema para entrenar nuestro modelo de Machine Learning, ya que pueden aprender a predecir siempre la clase mayoritaria y aún así tener buena precisión, sin realmente aprender a detectar la clase minoritaria.

Para lidiar con este desequilibrio, se pueden usar varias estrategias, como: - **Submuestreo (undersampling):** de la clase mayoritaria. - **Sobremuestreo (oversampling):** de la clase minoritaria, usando por ejemplo técnicas como SMOTE. - **Ajustar los pesos del modelo:** para que preste más atención a la clase minoritaria. - **Elegir métricas de evaluación más adecuadas:**, como `f1-score`, `recall` o `AUC`, en lugar de solo precisión.

1.2.5 Creamos dos variables nuevas

- temperature_difference: diferencia entre la temperatura de el aire y la del proceso
- Mechanical Power (W):

$$P = \frac{T \cdot 2\pi \cdot N}{60}$$

Donde:

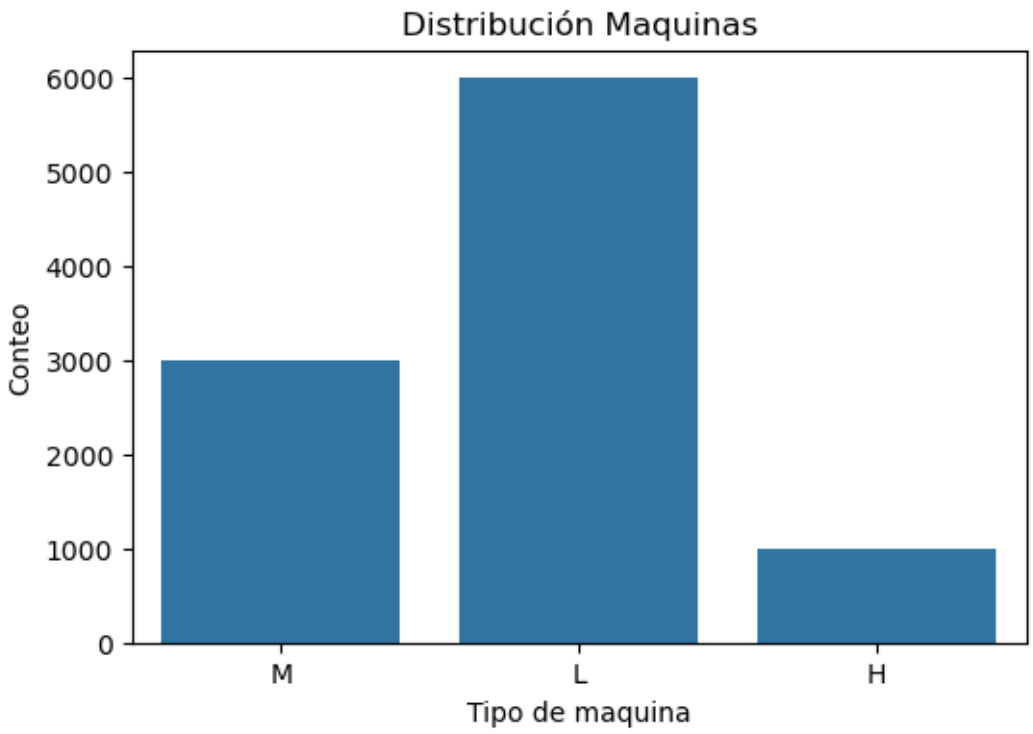
- P es la potencia en W,
- T es el torque en Nm,
- N es la velocidad en rpm,
- $\frac{2\pi}{60}$ convierte rpm a rad/s.

Aplicandolo a la data:

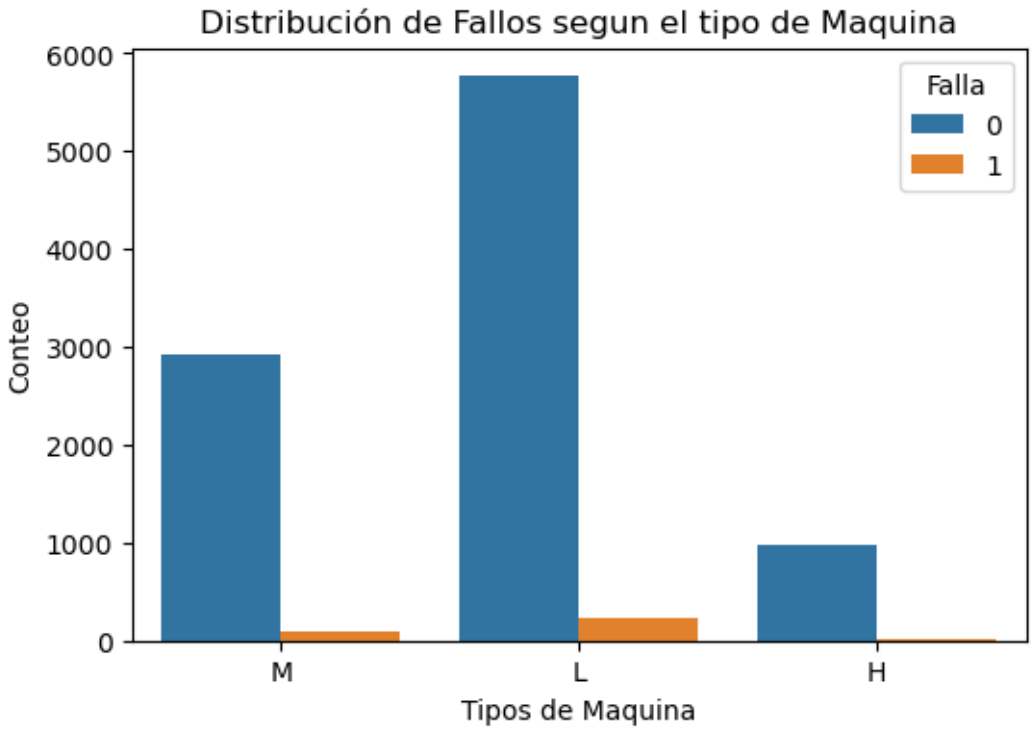
$$\text{Potencia (W)} = \frac{\text{Torque (Nm)} \cdot \text{Velocidad (rpm)} \cdot 2 \cdot \pi}{60.4}$$

1.3 Analisis de Datos Exploratorio

Visualización de Tipos de maquina Grafica la distribución de tipos de maquina

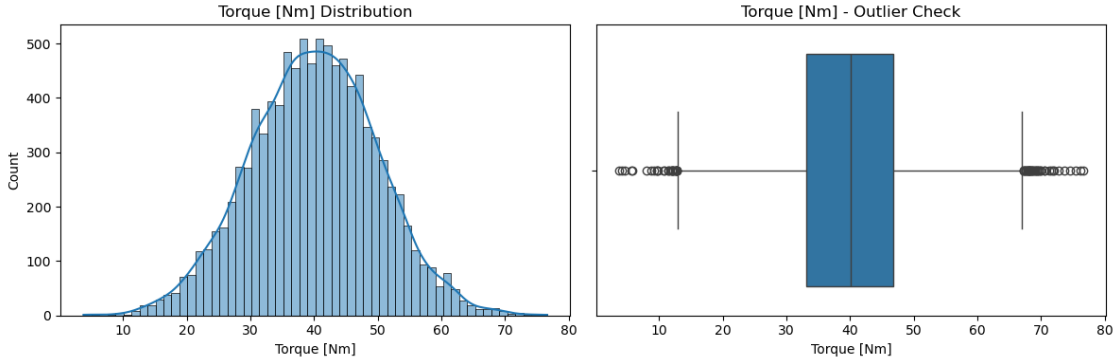


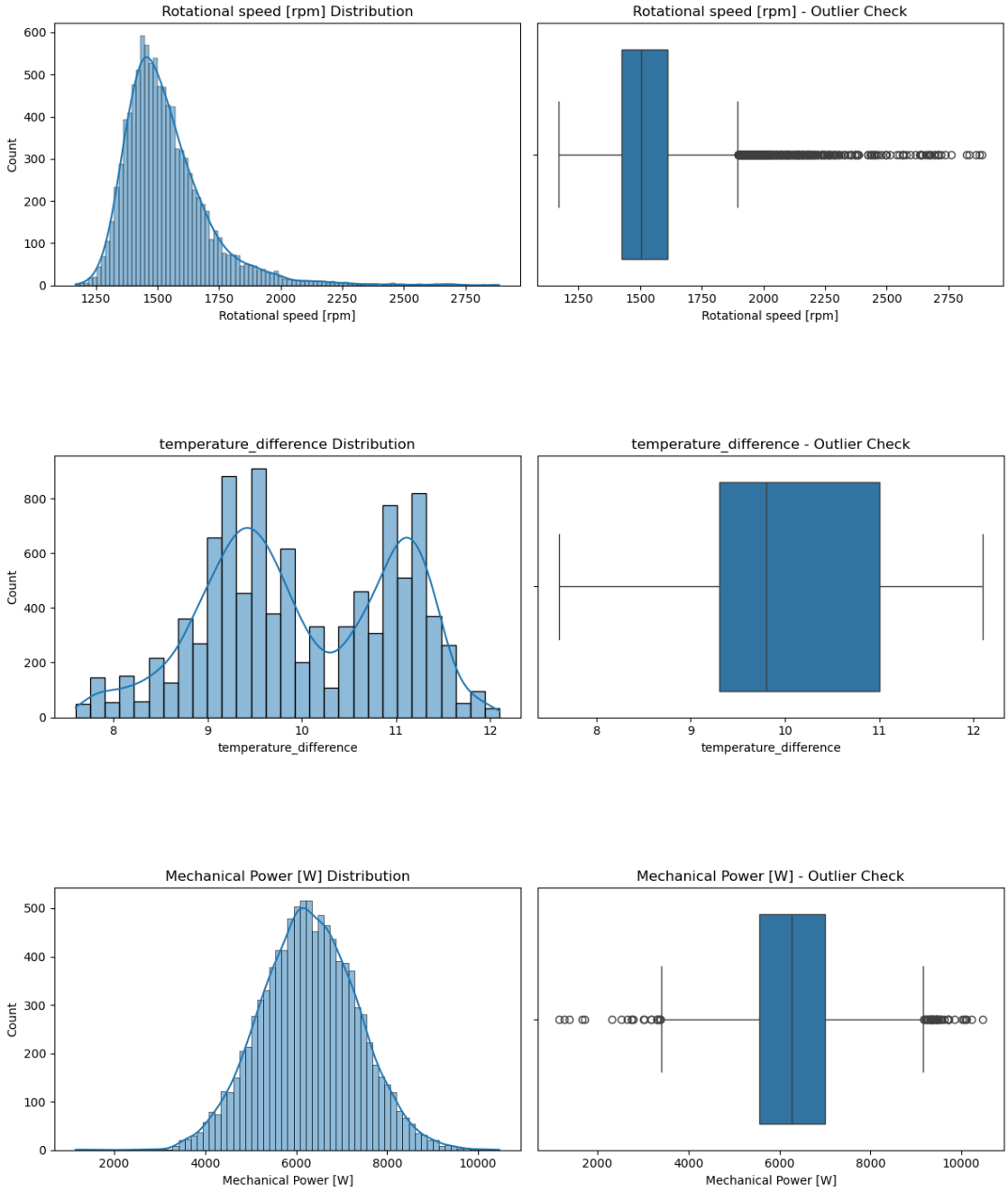
Visualización de la distribución de fallos entre tipos de productos Muestra cómo se distribuyen los fallos entre los tipos: ¿es posible que algunos tipos de maquina fallan más?



1.3.1 Grafico de distribuciones

Graficamos las distribuciones de las variables numericas para observar anomalias y patrones en los datos.





Pregunta 4: Describe que entiendes de la grafica anterior

Torque [Nm]

- **Histograma:** La forma es bastante simétrica, similar a una campana. La mayoría de los valores se agrupan entre 30 y 50 Nm, con un pico claro cerca de los 40 Nm.
- **Boxplot:** Aunque la distribución se ve normal, aparecen muchos valores atípicos (outliers) a ambos extremos. Esto podría representar condiciones especiales o errores que sería bueno

revisar antes de modelar.

Rotational speed [rpm]

- **Histograma:** Está sesgado hacia la derecha. Se observa una gran concentración de datos entre 1300 y 1700 rpm. Luego, los valores disminuyen de forma más gradual.
- **Boxplot:** Se identifican bastantes outliers en el extremo derecho. Aunque la mayoría de datos están bien agrupados, estos valores extremos pueden influir negativamente si no se gestionan.

temperature_difference

- **Histograma:** Muestra una distribución bimodal, es decir, con dos picos, lo que sugiere la presencia de dos grupos de condiciones diferentes en el dataset.
- **Boxplot:** A pesar de los dos grupos en el histograma, no hay outliers extremos. Esto indica que los datos están dentro de un rango relativamente controlado.

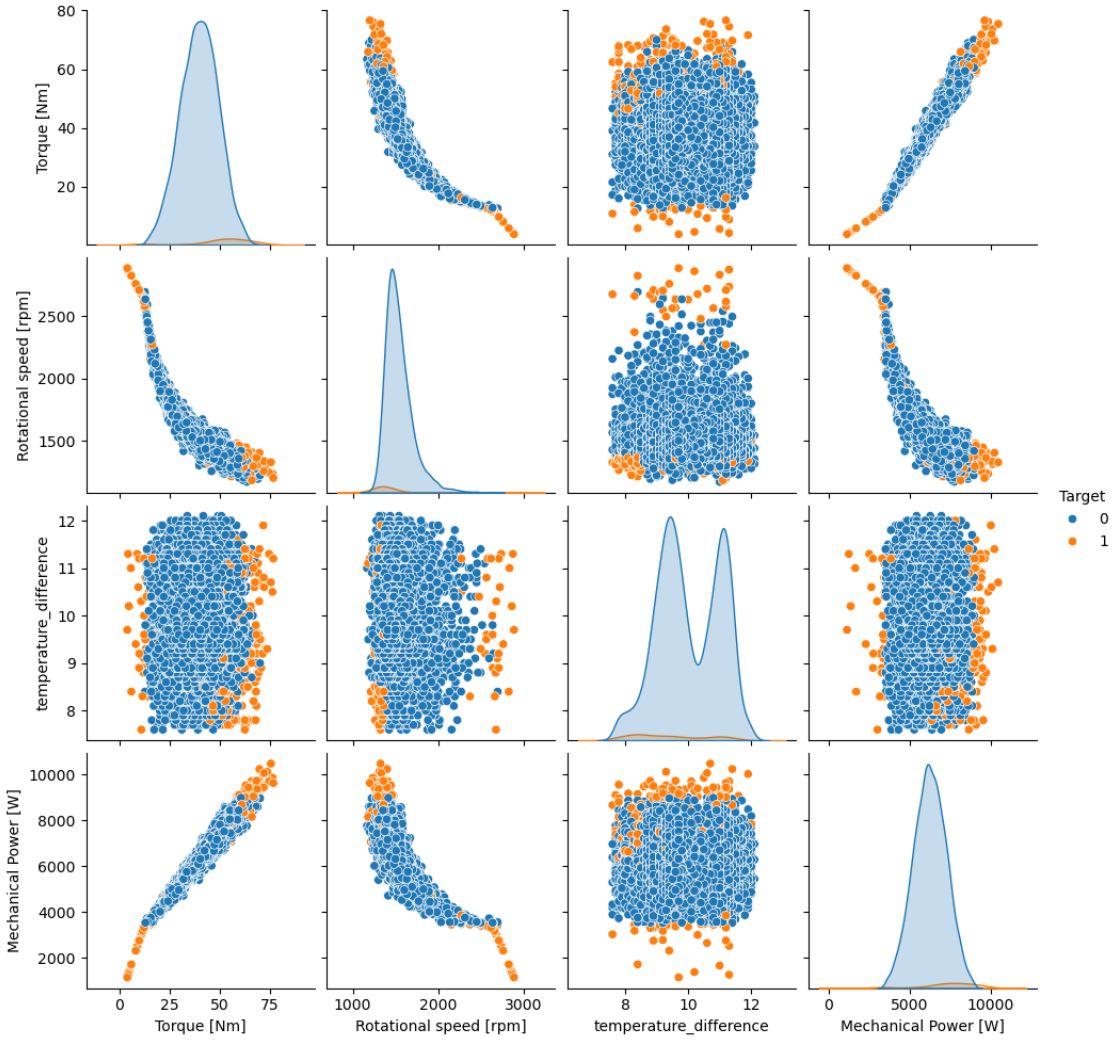
Mechanical Power [W]

- **Histograma:** Tiene una forma de campana bastante clara, centrada entre 5000 y 7000 W. Se puede decir que está cerca de una distribución normal.
- **Boxplot:** Hay varios outliers en ambos extremos. Aunque no son muchos, conviene tenerlos presentes porque podrían influir en la predicción si no se manejan correctamente.

Las variables analizadas presentan distribuciones en su mayoría normales o cercanas a normales. Algunas, como `rotational speed` o `mechanical power`, tienen valores atípicos importantes. Además, `temperature_difference` resalta por su distribución bimodal, lo que podría indicar diferentes estados de operación en el proceso. Es fundamental considerar estos detalles antes de aplicar modelos de Machine Learning, ya que los outliers y las formas de las distribuciones pueden afectar fuertemente los resultados.

1.3.2 Diagrama de pares para relaciones de características¶

Muestra la interacción entre las características coloreadas por falla.



1.3.3 Correlacion entre variables numericas

<Axes: >

Pregunta 5: Investiga sobre la matriz de confusión, que interpretas de estos resultados

Una matriz de confusión es una herramienta que nos permite evaluar el rendimiento de un modelo de clasificación. Nos muestra cuántas predicciones fueron correctas y cuántas no, separadas por clases. Se compone de 4 valores principales:
- **Verdaderos positivos (VP):** el modelo predijo correctamente la clase positiva.
- **Falsos positivos (FP):** el modelo predijo positivo cuando era negativo.
- **Falsos negativos (FN):** el modelo predijo negativo cuando era positivo.
- **Verdaderos negativos (VN):** el modelo predijo correctamente la clase negativa.

En los resultados mostrados: - Modelos como Gradient Boosting y Random Forest tienen un buen equilibrio: cometen pocos errores tanto en clases positivas como negativas. - En contraste, Logistic Regression y SGD tienen muchos más falsos positivos (por ejemplo, SGD predijo erróneamente 2035 positivos que eran realmente negativos), lo que demuestra que aunque mejoró el recall, se sacrificó mucha precisión.

2 Entrenando el Modelo

2.0.1 Separando el dataset

Utiliza la función `train_test_split` de sklearn utiliza 20% de los datos para el test set

2.1 Codificación de variables categoricas

Modelos de ML

```
======== Training model: Logistic Regression ============
Test data ROC AUC: 0.93
======== Training model: Logistic Regression CV ============
Test data ROC AUC: 0.93
======== Training model: SGD ============
Test data ROC AUC: 0.93
======== Training model: Random Forest ============
Test data ROC AUC: 0.97
======== Training model: Gradient Boosting ============
Test data ROC AUC: 0.98
======== Training model: Decision Tree ============
Test data ROC AUC: 0.84
```

```
Model Performance:
======== Model: Gradient Boosting ============
  Precision: 0.964286
  Recall:    0.794118
  F1 Score:  0.870968
  ROC AUC:   0.975680
======== Model: Random Forest ============
  Precision: 0.934783
  Recall:    0.632353
  F1 Score:  0.754386
  ROC AUC:   0.966501
======== Model: Decision Tree ============
  Precision: 0.758065
  Recall:    0.691176
  F1 Score:  0.723077
  ROC AUC:   0.841706
======== Model: Logistic Regression CV ============
  Precision: 0.181269
  Recall:    0.882353
  F1 Score:  0.300752
  ROC AUC:   0.933100
======== Model: Logistic Regression ============
  Precision: 0.180723
  Recall:    0.882353
  F1 Score:  0.300000
  ROC AUC:   0.932971
======== Model: SGD ============
  Precision: 0.113636
  Recall:    0.955882
  F1 Score:  0.203125
  ROC AUC:   0.930155
```

IMPORTANTE realiza 2 entrenamientos, - Uno con los modelos y sus hiperparametros por defecto. Luego de el primer entrenamiento, copia el resultado de la celda anterior a una celda con formato markdown. - Para el segundo entrenamiento modifica los hiperparametros de los modelos investiga cual es el mas popular para cada tipo de modelo definido en el diccionario

¿Mejoro algo al ajustar hiperparametros?

Tip: utiliza el hiper-parametro `class_weight='balanced'`

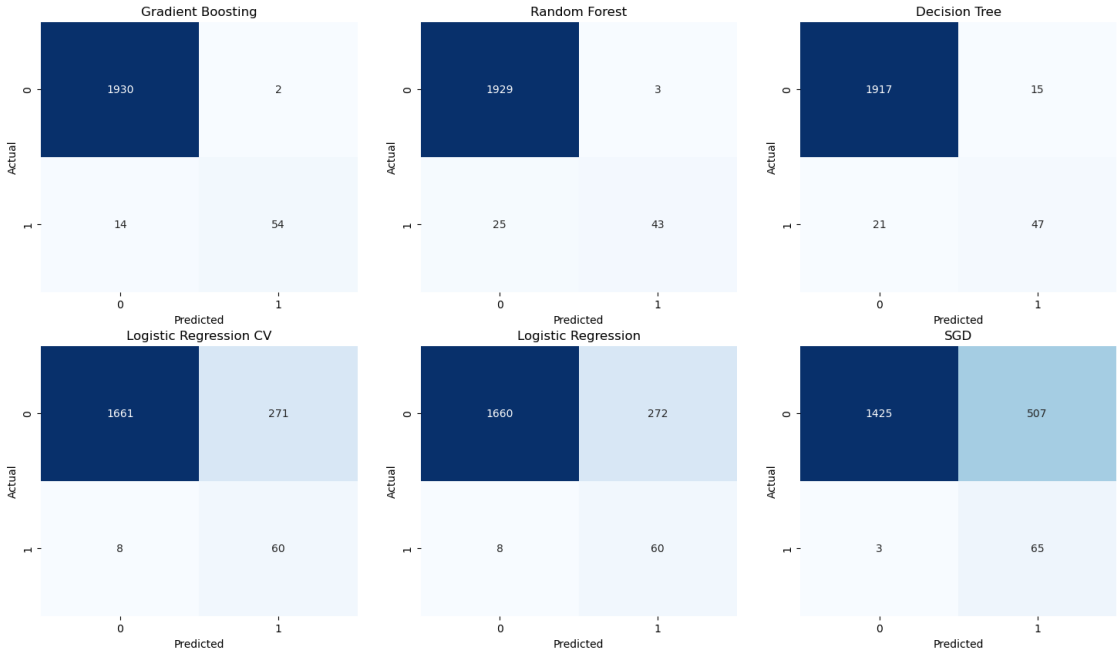
2.1.1 Model Performance (Primer entrenamiento)

```
======== Model: Gradient Boosting ============
Precision:     0.964286
Recall:        0.794118
F1 Score:      0.870968
ROC AUC:       0.975680

======== Model: Random Forest ============
Precision:     0.925926
Recall:        0.735294
F1 Score:      0.819672
ROC AUC:       0.964385

======== Model: Decision Tree ============
Precision:     0.742857
Recall:        0.764706
F1 Score:      0.753623
ROC AUC:       0.877695

======== Model: SGD ============
Precision:     0.636364
Recall:        0.308824
F1 Score:      0.415842
ROC AUC:       0.916043

======== Model: Logistic Regression ============
Precision:     0.619048
Recall:        0.191176
F1 Score:      0.292135
ROC AUC:       0.924560

======== Model: Logistic Regression CV ============
Precision:     0.666667
Recall:        0.147059
F1 Score:      0.240964
ROC AUC:       0.919711
```

2.1.2 ¿Mejoró algo al ajustar hiperparámetros?

Sí, ajustar los hiperparámetros ayudó bastante, sobre todo en modelos como **Logistic Regression** y **SGD**, que al inicio no detectaban bien los fallos (tenían `recall` muy bajo). Al usar `class_weight='balanced'`, estos modelos mejoraron mucho en `recall` y `f1_score`, es decir, ahora reconocen mejor los casos positivos.

Por otro lado, modelos como **Gradient Boosting** o **Random Forest** ya eran bastante buenos desde el principio, así que sus mejoras fueron pequeñas.



Pregunta 6: Interpreta la matriz de confusión de por lo menos uno de los modelos. ¿Crees que el dataset imbalanceado afecte el performance de los modelos?

2.1.3 Análisis de matriz de confusión (Primer entrenamiento)

Al observar la matriz de confusión del modelo **Logistic Regression**, podemos ver que predice correctamente la mayoría de los valores de la clase 0 (negativos), pero tiene un desempeño limitado para la clase 1 (positivos). Aunque logra identificar correctamente 60 casos positivos, se equivoca con 272 casos que clasifica como positivos cuando en realidad no lo son.

Esto indica que el modelo está **sesgado hacia la clase mayoritaria**, lo cual es un comportamiento común cuando el dataset está **desbalanceado**, como es este caso (muchos más ceros que unos). Modelos como Gradient Boosting y Random Forest también presentan este sesgo, aunque en menor medida gracias a su mayor capacidad de generalización.

Por lo tanto, el desbalance de clases sí afecta al rendimiento de los modelos, haciendo que fallen más al identificar correctamente la clase minoritaria.

2.1.4 Balanceo de datos

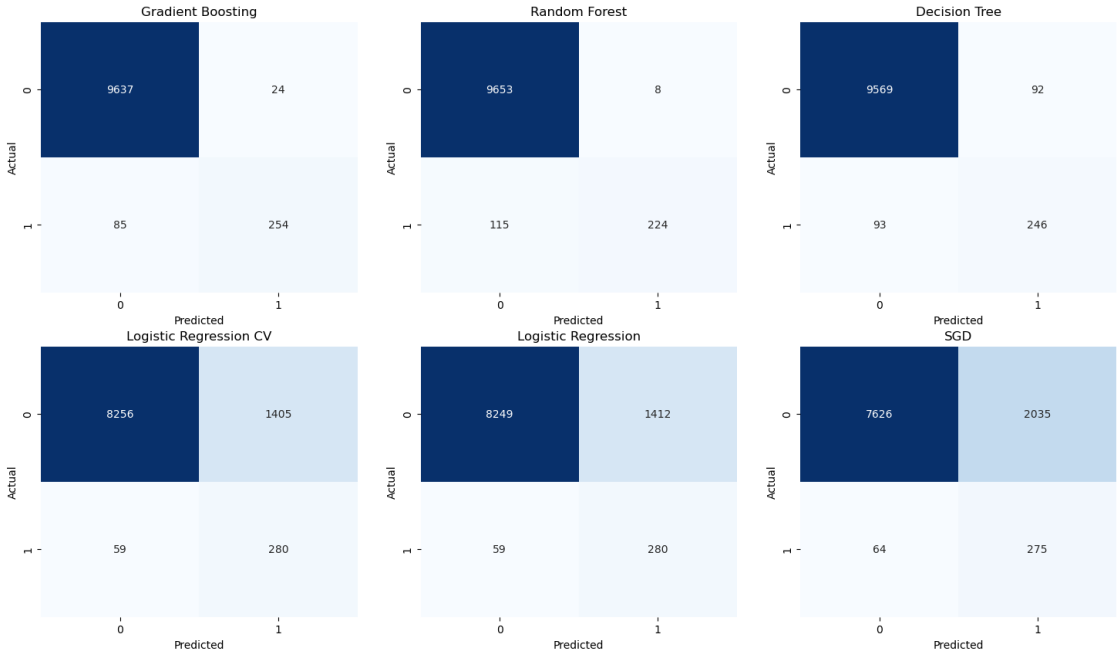
Vamos a utilizar `StratifiedKFold` ya que tenemos el dataset imbalanceado > **Pregunta 7:** Investiga que es el StratifiedKFold y en que nos puede ayudar con un dataset imbalanceado

`StratifiedKFold` es una técnica de validación cruzada que divide el conjunto de datos en múltiples "folds" (partes), asegurándose de que la proporción de clases (por ejemplo, 0 y 1) se mantenga constante en cada uno de ellos. Es decir, cada fold tendrá una representación similar de las clases, tal como ocurre en el conjunto original.

Esto es especialmente útil cuando trabajamos con **datasets desbalanceados**, ya que evita que alguna partición quede con muy pocos ejemplos de la clase minoritaria. Gracias a esto, el modelo puede aprender de forma más equilibrada y las métricas que obtenemos son más realistas y confiables.

```
======== Evaluating model: Logistic Regression with 5-Fold Stratified CV
============
  Fold 1 - F1: 0.27, AUC: 0.92
  Fold 2 - F1: 0.27, AUC: 0.92
  Fold 3 - F1: 0.27, AUC: 0.93
  Fold 4 - F1: 0.28, AUC: 0.93
  Fold 5 - F1: 0.28, AUC: 0.92
======== Evaluating model: Logistic Regression CV with 5-Fold Stratified CV
============
  Fold 1 - F1: 0.27, AUC: 0.92
  Fold 2 - F1: 0.28, AUC: 0.92
  Fold 3 - F1: 0.27, AUC: 0.93
  Fold 4 - F1: 0.28, AUC: 0.93
  Fold 5 - F1: 0.29, AUC: 0.92
======== Evaluating model: SGD with 5-Fold Stratified CV ============
  Fold 1 - F1: 0.18, AUC: 0.83
  Fold 2 - F1: 0.31, AUC: 0.91
  Fold 3 - F1: 0.18, AUC: 0.90
  Fold 4 - F1: 0.16, AUC: 0.91
  Fold 5 - F1: 0.39, AUC: 0.91
======== Evaluating model: Random Forest with 5-Fold Stratified CV ============
  Fold 1 - F1: 0.80, AUC: 0.96
  Fold 2 - F1: 0.80, AUC: 0.97
  Fold 3 - F1: 0.79, AUC: 0.97
  Fold 4 - F1: 0.82, AUC: 0.96
  Fold 5 - F1: 0.70, AUC: 0.96
======== Evaluating model: Gradient Boosting with 5-Fold Stratified CV
============
  Fold 1 - F1: 0.83, AUC: 0.99
  Fold 2 - F1: 0.85, AUC: 0.98
  Fold 3 - F1: 0.79, AUC: 0.98
  Fold 4 - F1: 0.85, AUC: 0.98
  Fold 5 - F1: 0.79, AUC: 0.98
======== Evaluating model: Decision Tree with 5-Fold Stratified CV ============
```

```
  Fold 1 - F1: 0.81, AUC: 0.89
  Fold 2 - F1: 0.75, AUC: 0.86
  Fold 3 - F1: 0.72, AUC: 0.84
  Fold 4 - F1: 0.69, AUC: 0.86
  Fold 5 - F1: 0.68, AUC: 0.84

Model Performance:
======== Model: Gradient Boosting ============
  Precision: 0.916676
  Recall:    0.749254
  F1 Score:  0.822337
  ROC AUC:   0.981615
======== Model: Random Forest ============
  Precision: 0.968930
  Recall:    0.660887
  F1 Score:  0.782713
  ROC AUC:   0.967346
======== Model: Decision Tree ============
  Precision: 0.733134
  Recall:    0.725812
  F1 Score:  0.728292
  ROC AUC:   0.858144
======== Model: Logistic Regression CV ============
  Precision: 0.166383
  Recall:    0.825944
  F1 Score:  0.276870
  ROC AUC:   0.923779
======== Model: Logistic Regression ============
  Precision: 0.165641
  Recall:    0.825944
  F1 Score:  0.275863
  ROC AUC:   0.923607
======== Model: SGD ============
  Precision: 0.149890
  Recall:    0.811062
  F1 Score:  0.241674
  ROC AUC:   0.893470
```



Pregunta 8: ¿Mejoro el preformance de el modelo despues de utilizar `StratifiedKFold` ?

Sí, el rendimiento de los modelos mejoró después de aplicar `StratifiedKFold`. Podemos verlo especialmente en los modelos de regresión logística (simple y con validación cruzada) y el modelo SGD, que en el primer entrenamiento tenían dificultades para identificar correctamente la clase minoritaria (los unos).

Después de usar `StratifiedKFold`, la distribución de clases en cada partición fue más equilibrada, lo que permitió a los modelos aprender mejor los patrones de ambas clases. Por ejemplo, en Logistic Regression CV se pasó de 60 verdaderos positivos a 280, lo cual es una mejora significativa en la capacidad del modelo para identificar los casos positivos.

`StratifiedKFold` permitión reducir el sesgo hacia la clase mayoritaria y mejoró el recall y el balance general en los modelos más afectados por el desbalance.