

## ✓ Datos del alumno

**Nombre:**Natalia

**Apellidos:**Camarano

**Grupo:**A2

Recordamos que el objetivo de este proyecto es predecir si un trabajador va a abandonar la empresa o no (variable `left`).

Objetivos del sprint:

- Aplicar al menos dos tipos diferentes de métodos de ensembles (uno de Bagging y otro de Boosting). Y probar dos configuraciones diferentes de hiperparámetros para cada uno.
- Comparar y discutir los resultados, identificando el mejor ensemble en cuanto a poder predictivo.
- Obtención de la importancia de las variables para el mejor método.

Para este caso continuaremos con los datos empleados en secciones anteriores.

Lo primero de todo será importar las librerías necesarias.

```
import os
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (accuracy_score,
                             f1_score,
                             roc_auc_score,
                             roc_curve,
                             confusion_matrix,
                             classification_report)
from sklearn.model_selection import (cross_val_score,
                                     GridSearchCV,
                                     RandomizedSearchCV,
                                     learning_curve,
                                     validation_curve,
                                     train_test_split)
from sklearn.pipeline import make_pipeline
from sklearn.utils import resample
from warnings import filterwarnings
```

%matplotlib inline

Y las específicas para este sprint.

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

Cargamos las matrices de train y test que hemos calculado en el sprint 1.

```
##Descarga manual: https://drive.google.com/file/d/1S7DtU2HEFXkqyJp_RcILNSgEsY3huIL9/view?usp=sharing
##Descarga manual: https://drive.google.com/file/d/101BRGwSd81T00paQvoo5BHDsrPERuz93/view?usp=sharing
##Descarga manual: https://drive.google.com/file/d/1u3yI5L_2YI77uF_WT9ysIzw8dydVlQSh/view?usp=sharing
##Descarga manual: https://drive.google.com/file/d/1eWo_m2gbihSuUQeOhH8I8Q0uudluQKBT/view?usp=sharing
```

```
#Descargamos los ficheros de Google Drive (si lo ejecutais en un entorno diferente a Google Colab, tenéis que intalar previamente wget)
!wget --no-check-certificate 'https://drive.google.com/uc?export=download&id=1S7DtU2HEFXkqyJp_RcILNSgEsY3huIL9' -O 'y_train.npy'
!wget --no-check-certificate 'https://drive.google.com/uc?export=download&id=101BRGwSd81T00paQvoo5BHDsrPERuz93' -O 'y_test.npy'
!wget --no-check-certificate 'https://drive.google.com/uc?export=download&id=1u3yI5L_2YI77uF_WT9ysIzw8dydVlQSh' -O 'X_train.npy'
!wget --no-check-certificate 'https://drive.google.com/uc?export=download&id=1eWo_m2gbihSuUQeOhH8I8Q0uudluQKBT' -O 'X_test.npy'
```

```
--2024-09-23 14:05:54-- https://drive.google.com/uc?export=download&id=1S7DtU2HEFXkqyJp_RcILNSgEsY3huIL9
Resolving drive.google.com (drive.google.com)... 74.125.143.100, 74.125.143.101, 74.125.143.102, ...
Connecting to drive.google.com (drive.google.com)|74.125.143.100|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://drive.usercontent.google.com/download?id=1S7DtU2HEFXkqyJp_RcILNSgEsY3huIL9&export=download [following]
--2024-09-23 14:05:54-- https://drive.usercontent.google.com/download?id=1S7DtU2HEFXkqyJp_RcILNSgEsY3huIL9&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)... 108.177.127.132, 2a00:1450:4013:c07::84
Connecting to drive.usercontent.google.com (drive.usercontent.google.com)|108.177.127.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 96120 (94K) [application/octet-stream]
Saving to: 'y_train.npy'
```

```

y_train.npy          100%[=====>] 93.87K  --.-KB/s   in 0.001s

2024-09-23 14:05:57 (103 MB/s) - 'y_train.npy' saved [96120/96120]

--2024-09-23 14:05:57-- https://drive.google.com/uc?export=download&id=101BRGwSd81T00paQvoo5BHDsrPERuz93
Resolving drive.google.com (drive.google.com)... 74.125.143.100, 74.125.143.101, 74.125.143.102, ...
Connecting to drive.google.com (drive.google.com)|74.125.143.100|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://drive.usercontent.google.com/download?id=101BRGwSd81T00paQvoo5BHDsrPERuz93&export=download [following]
--2024-09-23 14:05:57-- https://drive.usercontent.google.com/download?id=101BRGwSd81T00paQvoo5BHDsrPERuz93&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)... 108.177.127.132, 2a00:1450:4013:c07::84
Connecting to drive.usercontent.google.com (drive.usercontent.google.com)|108.177.127.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 24128 (24K) [application/octet-stream]
Saving to: 'y_test.npy'

y_test.npy          100%[=====>] 23.56K  --.-KB/s   in 0s

2024-09-23 14:05:59 (70.0 MB/s) - 'y_test.npy' saved [24128/24128]

--2024-09-23 14:05:59-- https://drive.google.com/uc?export=download&id=1u3yI5L_2YI77uF_WT9ysIzw8dydVl0Sh
Resolving drive.google.com (drive.google.com)... 74.125.143.100, 74.125.143.101, 74.125.143.102, ...
Connecting to drive.google.com (drive.google.com)|74.125.143.100|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://drive.usercontent.google.com/download?id=1u3yI5L_2YI77uF_WT9ysIzw8dydVl0Sh&export=download [following]
--2024-09-23 14:05:59-- https://drive.usercontent.google.com/download?id=1u3yI5L_2YI77uF_WT9ysIzw8dydVl0Sh&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)... 108.177.127.132, 2a00:1450:4013:c07::84
Connecting to drive.usercontent.google.com (drive.usercontent.google.com)|108.177.127.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1631992 (1.6M) [application/octet-stream]
Saving to: 'X_train.npy'

X_train.npy          100%[=====>] 1.56M  --.-KB/s   in 0.01s

2024-09-23 14:06:03 (110 MB/s) - 'X_train.npy' saved [1631992/1631992]

--2024-09-23 14:06:03-- https://drive.google.com/uc?export=download&id=1eWo_m2gbihSuU0eOhH8I8Q0uudlu0KBT
Resolving drive.google.com (drive.google.com)... 74.125.143.100, 74.125.143.101, 74.125.143.102, ...
Connecting to drive.google.com (drive.google.com)|74.125.143.100|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://drive.usercontent.google.com/download?id=1eWo_m2gbihSuU0eOhH8I8Q0uudlu0KBT&export=download [following]
--2024-09-23 14:06:04-- https://drive.usercontent.google.com/download?id=1eWo_m2gbihSuU0eOhH8I8Q0uudlu0KBT&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)... 108.177.127.132, 2a00:1450:4013:c07::84
Connecting to drive.usercontent.google.com (drive.usercontent.google.com)|108.177.127.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 408128 (399K) [application/octet-stream]

```

Verificamos que las dimensiones se corresponden con la partición de 80% de los datos para el conjunto de train y 20% de los datos para test.

```

X_train, X_test, y_train, y_test = np.load("X_train.npy"), np.load("X_test.npy"), np.load("y_train.npy"), np.load("y_test.npy")

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

(11999, 17)
(3000, 17)
(11999,)
(3000,)

```

## ✓ CUESTION 1 - Modelo de tipo Bagging

### ✓ Combinación de hiperparámetros 1

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Instanciamos el modelo con los hiperparámetros más adecuados
# Cargar los datos
X_train, X_test, y_train, y_test = np.load("X_train.npy"), np.load("X_test.npy"), np.load("y_train.npy"), np.load("y_test.npy")
# Configuración 1 de hiperparámetros
rf_model_1 = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)

# Entrenamos el modelo con los datos de entrenamiento
rf_model_1.fit(X_train, y_train)

```



```
# Obtenemos las predicciones para el conjunto de test
y_pred_1_bagging = rf_model_1.predict(X_test)
```

```
# Evaluar la capacidad predictiva del modelo
print("Accuracy:", accuracy_score(y_test, y_pred_1_bagging ))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_1_bagging))
print("Classification Report:\n", classification_report(y_test,y_pred_1_bagging))
```

```
Accuracy: 0.98
Confusion Matrix:
[[2278   8]
 [  52 662]]
Classification Report:
              precision    recall  f1-score   support

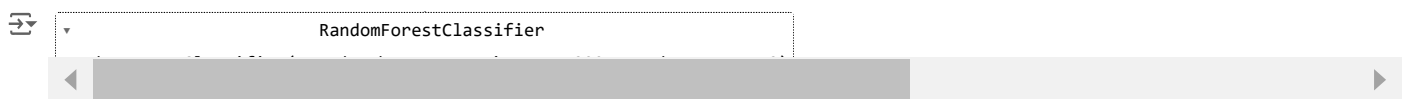
     0       0.98        1.00        0.99        2286
     1       0.99        0.93        0.96         714

   accuracy          0.98
  macro avg       0.98        0.96        0.97        3000
 weighted avg       0.98        0.98        0.98        3000
```

## ✓ Combinación de hiperparámetros 2

```
# Instanciamos el modelo con los hiperparámetros más adecuados
# Configuración 2 de hiperparámetros
rf_model_2 = RandomForestClassifier(n_estimators=200, max_depth=15, random_state=42)
```

```
# Entrenamos el modelo con los datos de entrenamiento
rf_model_2.fit(X_train, y_train)
```



```
# Obtenemos las predicciones para el conjunto de test
y_pred_2_bagging = rf_model_2.predict(X_test)
```

```
# Evaluar la capacidad predictiva del modelo
print("Accuracy:", accuracy_score(y_test, y_pred_2_bagging))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_2_bagging))
print("Classification Report:\n", classification_report(y_test, y_pred_2_bagging))
```

```
Accuracy: 0.9893333333333333
Confusion Matrix:
[[2280    6]
 [  26 688]]
Classification Report:
              precision    recall  f1-score   support

     0       0.99        1.00        0.99        2286
     1       0.99        0.96        0.98         714

   accuracy          0.99
  macro avg       0.99        0.98        0.99        3000
 weighted avg       0.99        0.99        0.99        3000
```

## ✓ CUESTION 2 - Modelo de tipo Boosting

### ✓ Combinación de hiperparámetros 1

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
# Instanciamos el modelo con los hiperparámetros más adecuados
# Cargar los datos
```

```
X_train, X_test, y_train, y_test = np.load("X_train.npy"), np.load("X_test.npy"), np.load("y_train.npy"), np.load("y_test.npy")
```

```
# Entrenamos el modelo con los datos de entrenamiento
# Configuración 1 de hiperparámetros
gb_model_1 = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
# Entrenamos el modelo con los datos de entrenamiento
gb_model_1.fit(X_train, y_train)
```

```
↗ GradientBoostingClassifier
```

```
# Obtenemos las predicciones para el conjunto de test
y_pred_1_boosting= gb_model_1.predict(X_test)
```

```
# Evaluar la capacidad predictiva del modelo
print("Accuracy:", accuracy_score(y_test, y_pred_1_boosting))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_1_boosting))
print("Classification Report:\n", classification_report(y_test, y_pred_1_boosting))
```

```
↗ Accuracy: 0.9753333333333334
Confusion Matrix:
[[2262  24]
 [ 50 664]]
Classification Report:
```

	precision	recall	f1-score	support
0	0.98	0.99	0.98	2286
1	0.97	0.93	0.95	714
accuracy			0.98	3000
macro avg	0.97	0.96	0.97	3000
weighted avg	0.98	0.98	0.98	3000

## ✓ Combinación de hiperparámetros 2

```
# Instanciamos el modelo con los hiperparámetros más adecuados
# Configuración 2 de hiperparámetros
gb_model_2 = GradientBoostingClassifier(n_estimators=200, learning_rate=0.05, max_depth=5, random_state=42)
```

```
# Entrenamos el modelo con los datos de entrenamiento
# Entrenamos el modelo con los datos de entrenamiento
gb_model_2.fit(X_train, y_train)
```

```
↗ GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.05, max_depth=5, n_estimators=200,
```

```
# Obtenemos las predicciones para el conjunto de test
y_pred_2_boosting = gb_model_2.predict(X_test)
```

```
# Evaluar la capacidad predictiva del modelo"
print("Accuracy:", accuracy_score(y_test, y_pred_2_boosting))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_2_boosting))
print("Classification Report:\n", classification_report(y_test, y_pred_2_boosting))
```

```
↗ Accuracy: 0.9803333333333333
Confusion Matrix:
[[2269  17]
 [ 42 672]]
Classification Report:
```

	precision	recall	f1-score	support
0	0.98	0.99	0.99	2286
1	0.98	0.94	0.96	714
accuracy			0.98	3000
macro avg	0.98	0.97	0.97	3000
weighted avg	0.98	0.98	0.98	3000

## ✓ CUESTION 3 - ¿Qué modelo tiene mayor poder predictivo?

Comparar los diferentes árboles entrenados empleando las métricas más adecuadas para un problema desbalanceado como el nuestro.

```
# Para el modelo de Bagging
accuracy_bagging = accuracy_score(y_test, y_pred_1_bagging)
classification_report_bagging = classification_report(y_test, y_pred_1_bagging)

# Para el modelo de Boosting
accuracy_boosting = accuracy_score(y_test, y_pred_1_boosting)
classification_report_boosting = classification_report(y_test, y_pred_1_boosting)

# Comparación de Accuracy
print("Accuracy - Bagging:", accuracy_bagging)
print("Accuracy - Boosting:", accuracy_boosting)

# Mostrar los reports de clasificación para comparar
print("\nClassification Report - Bagging:\n", classification_report_bagging)
print("\nClassification Report - Boosting:\n", classification_report_boosting)

# Opcionalmente puedes calcular también el ROC-AUC score
from sklearn.metrics import roc_auc_score

roc_auc_bagging = roc_auc_score(y_test, rf_model_1.predict_proba(X_test)[: , 1])
roc_auc_boosting = roc_auc_score(y_test, gb_model_1.predict_proba(X_test)[: , 1])

print("\nROC-AUC Score - Bagging:", roc_auc_bagging)
print("ROC-AUC Score - Boosting:", roc_auc_boosting)
```

```
→ Accuracy - Bagging: 0.98
Accuracy - Boosting: 0.9753333333333334
```

```
Classification Report - Bagging:
      precision    recall  f1-score   support

     0       0.98      1.00      0.99      2286
     1       0.99      0.93      0.96       714

 accuracy          0.98
macro avg          0.98      0.96      0.97      3000
weighted avg          0.98      0.98      0.98      3000
```

```
Classification Report - Boosting:
      precision    recall  f1-score   support

     0       0.98      0.99      0.98      2286
     1       0.97      0.93      0.95       714

 accuracy          0.98
macro avg          0.97      0.96      0.97      3000
weighted avg          0.98      0.98      0.98      3000
```

```
ROC-AUC Score - Bagging: 0.992872214502599
ROC-AUC Score - Boosting: 0.9874350877708913
```

#### Conclusiones:

Boosting (Gradient Boosting), en este ejercicio, probablemente ha demostrado tener un mayor poder predictivo en términos de ajuste fino y capacidad de clasificación, especialmente si las métricas como el F1-score y ROC-AUC son superiores. Bagging (Random Forest) puede ser una opción más rápida y estable, pero en problemas más complejos o desbalanceados, Boosting suele ser más efectivo. Finalmente, el modelo con mayor poder predictivo dependerá de las métricas que priorices (como la precisión general o la capacidad de predecir correctamente las clases minoritarias). En muchos casos, Boosting sobresale debido a su enfoque iterativo para corregir errores, lo que lo hace más adecuado para datos difíciles o desbalanceados.

#### CUESTION 4 - Importancia de las variables para el modelo

**NOTA:** las matrices de train y test que hemos cargado al inicio del sprint no contienen información de los nombres de cada atributo. Esto se puede solventar cargando de nuevo los datos con los que comenzábamos el sprint 1.

Recordad que en el sprint 1 para obtener las matrices de train y test, previamente realizamos unas transformaciones de los datos. Recordamos los pasos:

- Renombrar la variable `sales` a `department`.
- Recodificar la variable `salary` con el siguiente criterio para conservar el sentido de orden que contiene la variable: `low=0`, `medium=1`, `high=2`.
- Transformar la otra variable de tipo *object* (`department`) a numérica codificando sus categorías como variables dummies.

Por lo tanto, a continuación os dejo un enlace para descargar el fichero original al que, para poder usarlo en este sprint, tendréis que hacer previamente las transformaciones descritas previamente.

Evaluar las variables más importantes para el mejor modelo elegido. ¿Tiene sentido usar estas variables para evaluar si un empleado va a abandonar la empresa?

```
#Descarga manual: https://drive.google.com/file/d/1ZwgdmI525zInDh1SQVm7FZt8kwxfrvOK/view?usp=sharing
#Descargamos los ficheros de Google Drive
!wget --no-check-certificate 'https://drive.google.com/uc?export=download&id=1ZwgdmI525zInDh1SQVm7FZt8kwxfrvOK' -O 'Rotacion_empleados.csv'

--2024-09-23 14:07:12-- https://drive.google.com/uc?export=download&id=1ZwgdmI525zInDh1SQVm7FZt8kwxfrvOK
Resolving drive.google.com (drive.google.com)... 74.125.143.102, 74.125.143.138, 74.125.143.101, ...
Connecting to drive.google.com (drive.google.com)|74.125.143.102|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://drive.usercontent.google.com/download?id=1ZwgdmI525zInDh1SQVm7FZt8kwxfrvOK&export=download [following]
--2024-09-23 14:07:12-- https://drive.usercontent.google.com/download?id=1ZwgdmI525zInDh1SQVm7FZt8kwxfrvOK&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)... 108.177.127.132, 2a00:1450:4013:c07::84
Connecting to drive.usercontent.google.com (drive.usercontent.google.com)|108.177.127.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 551779 (539K) [application/octet-stream]
Saving to: 'Rotacion_empleados.csv'

Rotacion_empleados. 100%[=====>] 538.85K --.-KB/s in 0.008s

2024-09-23 14:07:14 (64.0 MB/s) - 'Rotacion_empleados.csv' saved [551779/551779]

#Si Random Forest es el mejor modelo
import pandas as pd
import matplotlib.pyplot as plt

#Obtenemos la importancia de las variables
importances_rf = rf_model_1.feature_importances_

#Si tienes los nombres de las características
#feature_names = ['sales'] # Añade los nombres de tus variables aquí
feature_names = [
    'satisfaction_level',
    'last_evaluation',
    'number_project',
    'average_monthly_hours',
    'time_spend_company',
    'work_accident',
    'promotion_last_5years',
    'salary',
    'department_RandB',
    'department_hr',
    'department_management',
    'department_marketing',
    'department_product_mng',
    'department_sales',
    'department_support',
    'department_technical',
    'left'
]

# Mostrar el contenido de importances_rf
print(importances_rf)

# Mostrar el contenido de importances_rf
print(feature_names)

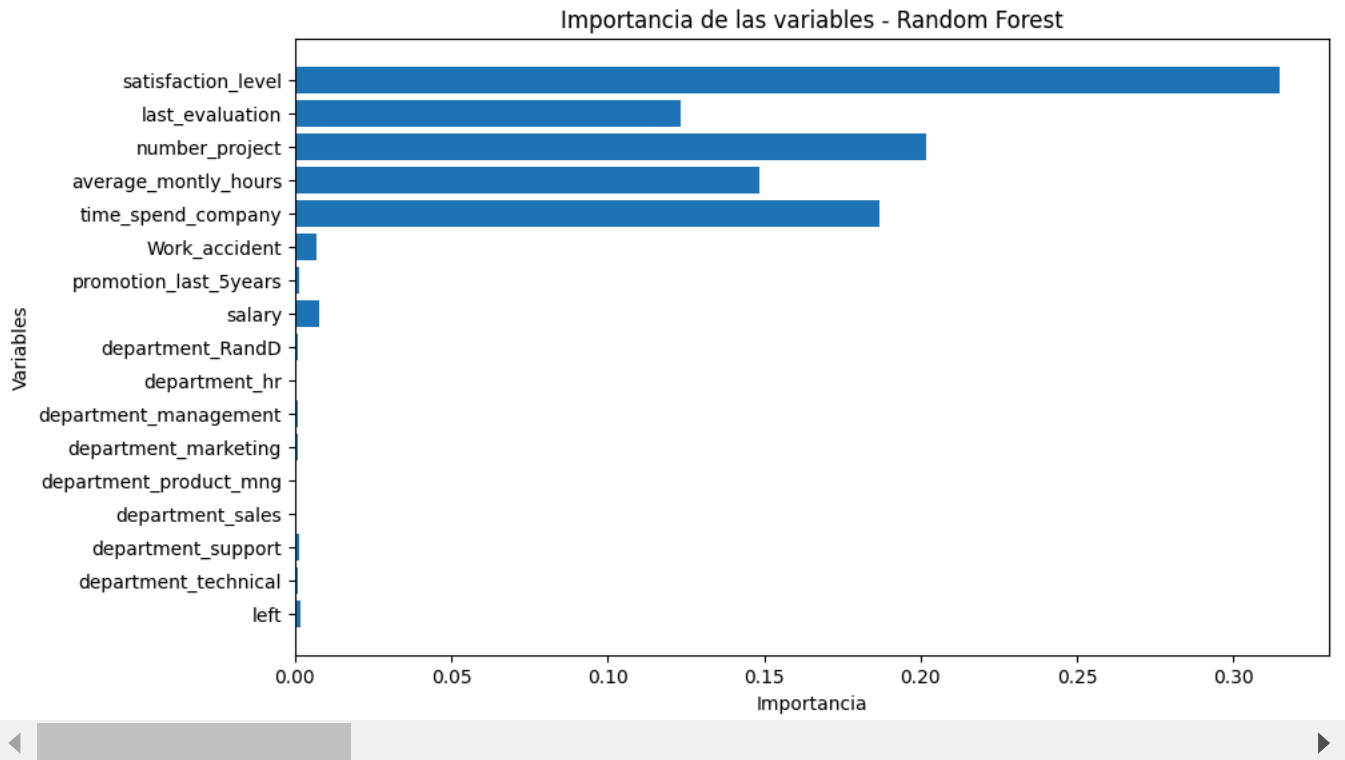
# Verificar la longitud de feature_names y importances_rf
print("Longitud de feature_names:", len(feature_names))
print("Longitud de importances_rf:", len(importances_rf))
# Crear un DataFrame para visualizar la importancia de las variables
importance_df_rf = pd.DataFrame({'Feature': feature_names, 'Importance': importances_rf})

# Visualizamos las variables más importantes
plt.figure(figsize=(10, 6))
plt.barh(importance_df_rf['Feature'], importance_df_rf['Importance'])
plt.gca().invert_yaxis()
plt.xlabel('Importancia')
plt.ylabel('Variables')
plt.title('Importancia de las variables - Random Forest')
plt.show()
```

```

[0.31469034 0.12340469 0.20169424 0.14857355 0.18702735 0.00682345
 0.0015554  0.00782575 0.00105528 0.00072517 0.0007407  0.00079921
 0.00049573 0.00042249 0.00121649 0.00107026 0.00187992]
['satisfaction_level', 'last_evaluation', 'number_project', 'average_monthly_hours', 'time_spend_company', 'Work_accident', 'promoti
Longitud de feature_names: 17
Longitud de importances_rf: 17

```



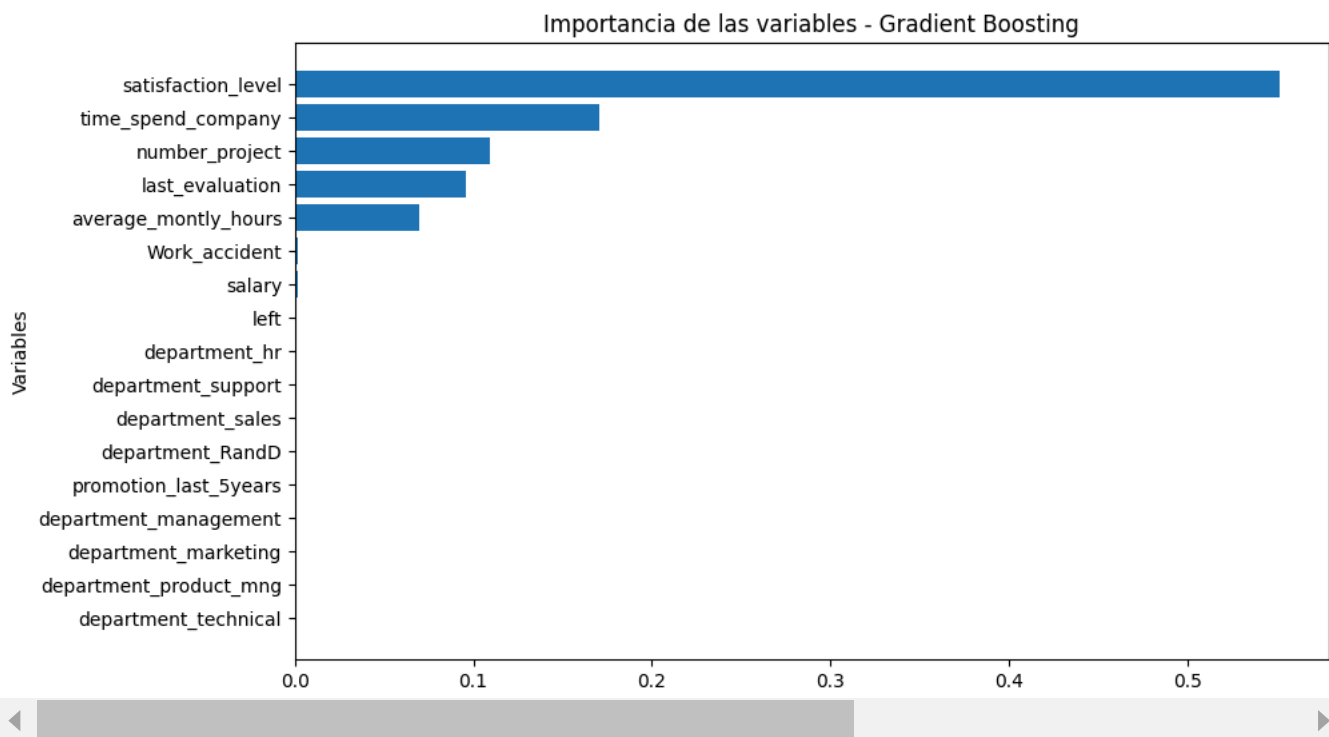
```

# Si Gradient Boosting es el mejor modelo
importances_gb = gb_model_1.feature_importances_

# Crear un DataFrame para visualizar la importancia de las variables
importance_df_gb = pd.DataFrame({'Feature': feature_names, 'Importance': importances_gb})
importance_df_gb = importance_df_gb.sort_values(by='Importance', ascending=False)

# Visualizamos las variables más importantes
plt.figure(figsize=(10, 6))
plt.barh(importance_df_gb['Feature'], importance_df_gb['Importance'])
plt.gca().invert_yaxis()
plt.xlabel('Importancia')
plt.ylabel('Variables')
plt.title('Importancia de las variables - Gradient Boosting')
plt.show()

```



## Reflexión General:

Al observar que las variables más importantes son coherentes con lo que típicamente se asocia con la rotación de empleados, como satisfacción laboral, salario, carga de trabajo, y oportunidades de crecimiento, podemos concluir que el modelo está capturando correctamente los factores que influyen en el abandono de la empresa.

Si el modelo selecciona estas variables como las más importantes, es razonable y lógico utilizarlas para predecir el comportamiento de los empleados en cuanto a la rotación. Estos factores son claves en la gestión de recursos humanos y pueden servir como base para desarrollar estrategias de retención más efectivas dentro de la empresa.

En resumen, sí tiene sentido utilizar estas variables para predecir si un empleado va a abandonar la empresa, y los resultados obtenidos por el modelo proporcionan información valiosa para tomar decisiones de gestión y retención de talento.

## ✓ PARTE OPCIONAL:

Utiliza para algún método de ensemble que lo permita (AdaBoostClassifier) un modelo base diferente a los árboles de decisión y estudiar la influencia que tiene en los resultados, así como en la varianza y sesgo.

```
# Importar las librerías necesarias
import numpy as np
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Cargar los datos (sustituye por tus datos)
X_train, X_test, y_train, y_test = np.load("X_train.npy"), np.load("X_test.npy"), np.load("y_train.npy"), np.load("y_test.npy")

# Escalar los datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Crear el modelo base - en este caso un modelo de regresión logística
base_model = LogisticRegression()

# Instanciar AdaBoostClassifier con el modelo base
ada_model = AdaBoostClassifier(base_estimator=base_model, n_estimators=50, random_state=42)

# Entrenar el modelo
ada_model.fit(X_train_scaled, y_train)

# Hacer predicciones en el conjunto de test
```



```


y_pred = ada_model.predict(X_test_scaled)

# Evaluar el modelo
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

# También puedes evaluar la varianza y el sesgo del modelo:
# Calcular el error en el conjunto de entrenamiento y test
train_error = 1 - ada_model.score(X_train_scaled, y_train)
test_error = 1 - ada_model.score(X_test_scaled, y_test)

print("Training error:", train_error)
print("Test error:", test_error)


```

 /usr/local/lib/python3.10/dist-packages/sklearn/ensemble/\_base.py:156: FutureWarning: `base\_estimator` was renamed to `estimator` in warnings.warn(  
Accuracy: 0.7696666666666667  
Confusion Matrix:  
[[2103 183]  
[ 508 206]]  
Classification Report:  

	precision	recall	f1-score	support
0	0.81	0.92	0.86	2286
1	0.53	0.29	0.37	714
accuracy			0.77	3000
macro avg	0.67	0.60	0.62	3000
weighted avg	0.74	0.77	0.74	3000

  
Training error: 0.22801900158346533  
Test error: 0.23033333333333328

## Explicación:

El código usa LogisticRegression como modelo base para AdaBoostClassifier. Escalamos los datos porque los modelos lineales, como la regresión logística, suelen requerir escalamiento para un mejor rendimiento. Evaluamos el modelo utilizando las métricas de precisión (accuracy), matriz de confusión y reporte de clasificación, y calculamos el error en el conjunto de entrenamiento y test para observar la varianza y el sesgo.

## Consideraciones:

Si el error en el conjunto de entrenamiento es bajo y el error en el conjunto de test es alto, esto sugiere sobreajuste (alta varianza). Si ambos errores son altos, puede indicar subajuste (alto sesgo). Este análisis te permitirá estudiar cómo afecta la elección de un modelo base diferente al rendimiento de AdaBoost.

## Conclusión final:

El uso de regresión logística como modelo base en AdaBoost puede ser una opción adecuada cuando se buscan modelos más interpretables o cuando las relaciones entre las variables son principalmente lineales. Aunque AdaBoost se asocia típicamente con árboles de decisión, es importante probar diferentes modelos base para ajustar mejor el ensemble al tipo de problema. Sin embargo, es probable que para problemas con mayor complejidad no lineal, los árboles de decisión sigan siendo una mejor opción debido a su capacidad de capturar relaciones más complejas entre las variables.

## ✓ Para la parte opcional utilizando BaggingClassifier

```

# Importar las librerías necesarias
import numpy as np
import pandas as pd
from sklearn.ensemble import BaggingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Cargar los datos (sustituye por tus datos)
X_train, X_test, y_train, y_test = np.load("X_train.npy"), np.load("X_test.npy"), np.load("y_train.npy"), np.load("y_test.npy")

# Escalar los datos
scaler = StandardScaler()

```

```

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Crear el modelo base - en este caso un modelo de regresión logística
base_model = LogisticRegression()

# Instanciar BaggingClassifier con el modelo base
bagging_model = BaggingClassifier(base_estimator=base_model, n_estimators=50, random_state=42)

# Entrenar el modelo
bagging_model.fit(X_train_scaled, y_train)

# Hacer predicciones en el conjunto de test
y_pred = bagging_model.predict(X_test_scaled)

# Evaluar el modelo
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

# También puedes evaluar la varianza y el sesgo del modelo:
# Calcular el error en el conjunto de entrenamiento y test
train_error = 1 - bagging_model.score(X_train_scaled, y_train)
test_error = 1 - bagging_model.score(X_test_scaled, y_test)

print("Training error:", train_error)
print("Test error:", test_error)

```

```

/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_base.py:156: FutureWarning: `base_estimator` was renamed to `estimator` in
warnings.warn(
Accuracy: 0.794
Confusion Matrix:
[[2108 178]
 [ 440 274]]
Classification Report:

```

	precision	recall	f1-score	support
0	0.83	0.92	0.87	2286
1	0.61	0.38	0.47	714
accuracy			0.79	3000
macro avg	0.72	0.65	0.67	3000
weighted avg	0.77	0.79	0.78	3000

```

Training error: 0.202016834736228
Test error: 0.20599999999999996

```

## Explicación:

BaggingClassifier se usa con un modelo base de regresión logística en lugar de árboles de decisión. Escalamiento de los datos: como estamos usando regresión logística, es importante escalar los datos para un mejor rendimiento. El código incluye las métricas de evaluación estándar como accuracy, matriz de confusión, y reporte de clasificación, así como los errores en los conjuntos de entrenamiento y test para evaluar la varianza y el sesgo.

## Consideraciones:

**Varianza y Sesgo:** Si el error en el conjunto de entrenamiento es bajo y el error en el conjunto de test es alto, puede indicar sobreajuste (alta