

## 🎯 Proyecto Capstone: Predicción de Rotación de Empleados – Salifort Motors

🎯 Objetivo Diseñar un modelo predictivo que identifique si un empleado abandonará la empresa, utilizando variables como:

Departamento

Número de proyectos

Media de horas mensuales

Evaluación del desempeño

Entre otras

*PROPÓSITO Salifort Motors enfrenta una alta rotación de empleados. Esto afecta la productividad y eleva los costos de contratación y formación. El objetivo de este análisis es construir un modelo que prediga si un empleado abandonará la empresa. Esto permitirá tomar decisiones informadas y diseñar estrategias de retención más efectivas.*

### 🛠️ A.1. Importación de librerías y carga de datos

```
In [80]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Cargar el conjunto de datos de empleados
df = pd.read_csv('HR_comma_sep.csv')
df.head()
```

Out[80]:

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	time_spend_co
--	--------------------	-----------------	----------------	-----------------------	---------------

0	0.38	0.53	2	157	
1	0.80	0.86	5	262	
2	0.11	0.88	7	272	
3	0.72	0.87	5	223	
4	0.37	0.52	2	159	

### 🛠️ A.2. Análisis exploratorio (EDA)

```
In [81]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Valores nulos
print("🔍 Valores nulos por columna:")
```

```

print(df.isnull().sum())

# Tipos de datos
print("\n📊 Tipos de datos:")
print(df.dtypes)

# Mapeo de la variable 'left' para hacerlo más claro
df['left_label'] = df['left'].map({0: 'Se quedó', 1: 'Dejó la empresa'})

# Gráfico con etiquetas claras
plt.figure(figsize=(6,4))
sns.countplot(x='left_label', hue='left_label', data=df, palette='Set2', legend=False)
plt.title(' Distribución de empleados que dejaron la empresa')
plt.xlabel('Estado del empleado')
plt.ylabel('Cantidad de empleados')
plt.tight_layout()
plt.show()

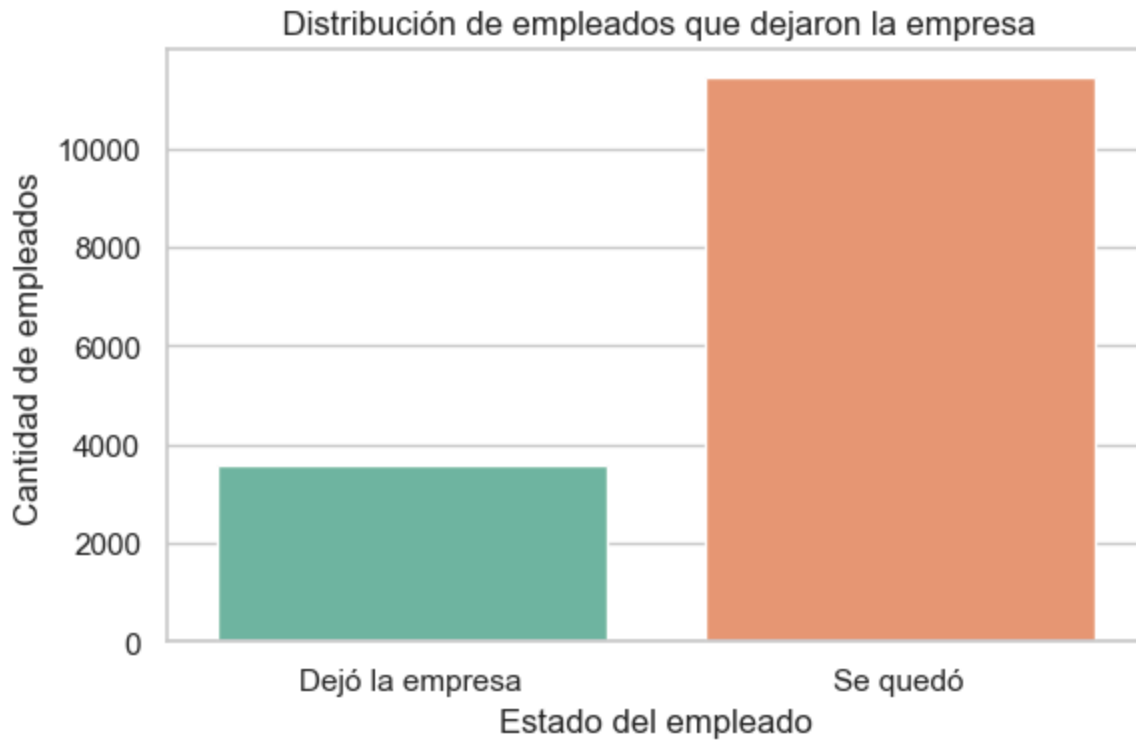
```

🔍 Valores nulos por columna:

satisfaction_level	0
last_evaluation	0
number_project	0
average_monthly_hours	0
time_spend_company	0
Work_accident	0
left	0
promotion_last_5years	0
Department	0
salary	0
dtype:	int64

📊 Tipos de datos:

satisfaction_level	float64
last_evaluation	float64
number_project	int64
average_monthly_hours	int64
time_spend_company	int64
Work_accident	int64
left	int64
promotion_last_5years	int64
Department	object
salary	object
dtype:	object



```
In [82]: import matplotlib.pyplot as plt

# Lista de columnas y títulos traducidos
columnas = {
    'satisfaction_level': 'Nivel de satisfacción',
    'last_evaluation': 'Última evaluación',
    'number_project': 'Número de proyectos',
    'average_monthly_hours': 'Horas mensuales promedio',
    'time_spend_company': 'Años en la empresa'
}

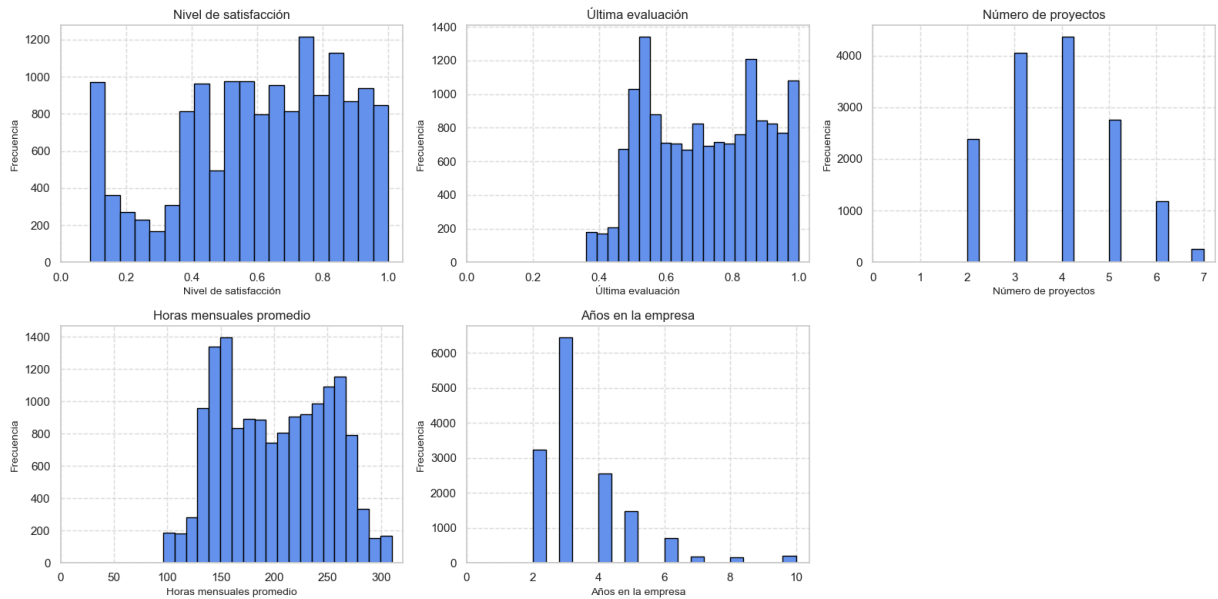
# Crear subgráficos
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(16, 8))
axs = axs.flatten()

# Dibujar histogramas uno por uno
for i, (col, titulo) in enumerate(columnas.items()):
    ax = axs[i]
    ax.hist(df[col], bins=20, edgecolor='black', color='cornflowerblue')
    ax.set_title(titulo, fontsize=12)
    ax.set_xlabel(titulo, fontsize=10)
    ax.set_ylabel('Frecuencia', fontsize=10)
    ax.grid(True, linestyle='--', alpha=0.6)
    ax.set_xlim(left=0) # Para evitar que empiece en negativo

# Eliminar subgráfico sobrante si hay menos de 6
for j in range(len(columnas), len(axs)):
    fig.delaxes(axs[j])

plt.tight_layout()
plt.suptitle('Distribución de variables numéricas del personal', fontsize=14, y=1.0)
plt.show()
```

Distribución de variables numéricas del personal



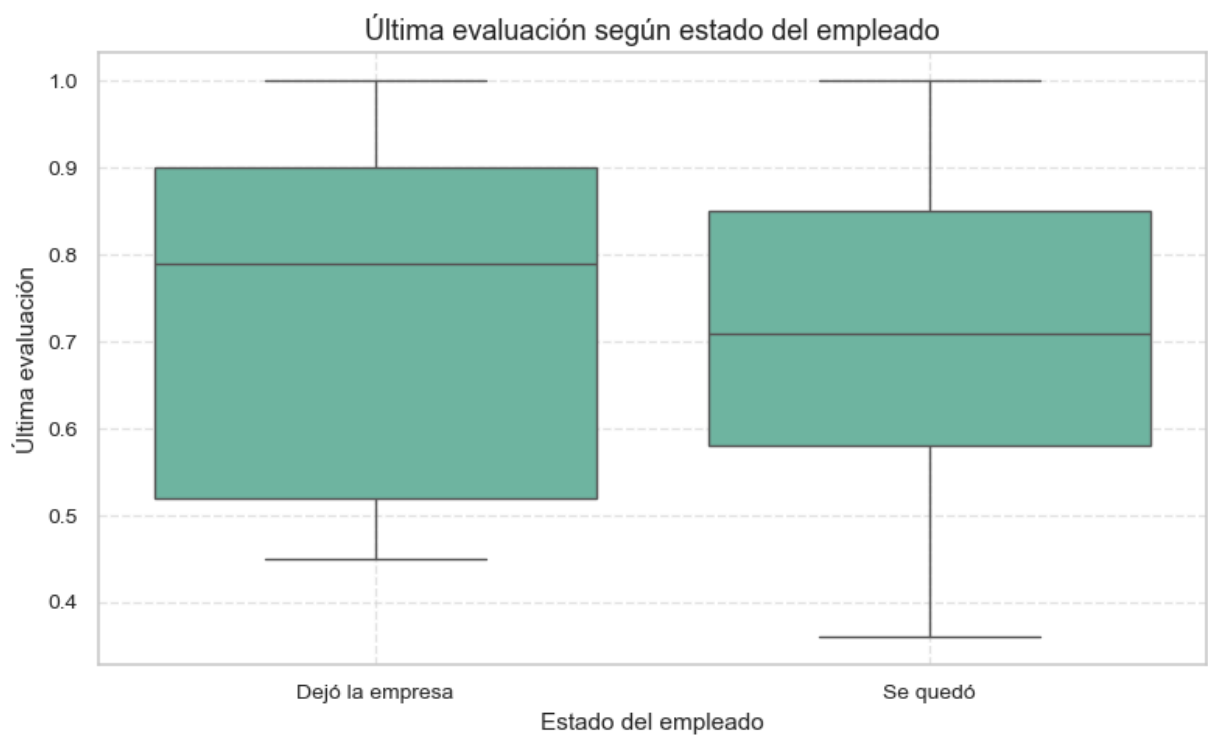
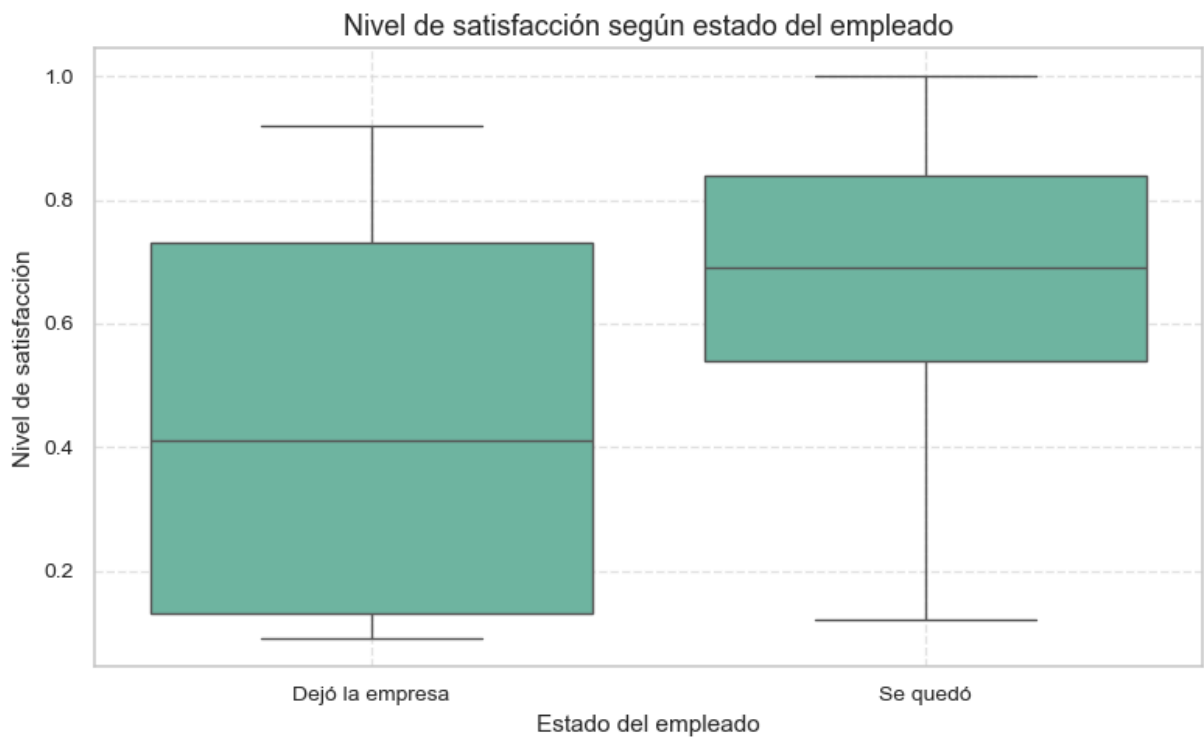
```
In [83]: import seaborn as sns
import matplotlib.pyplot as plt

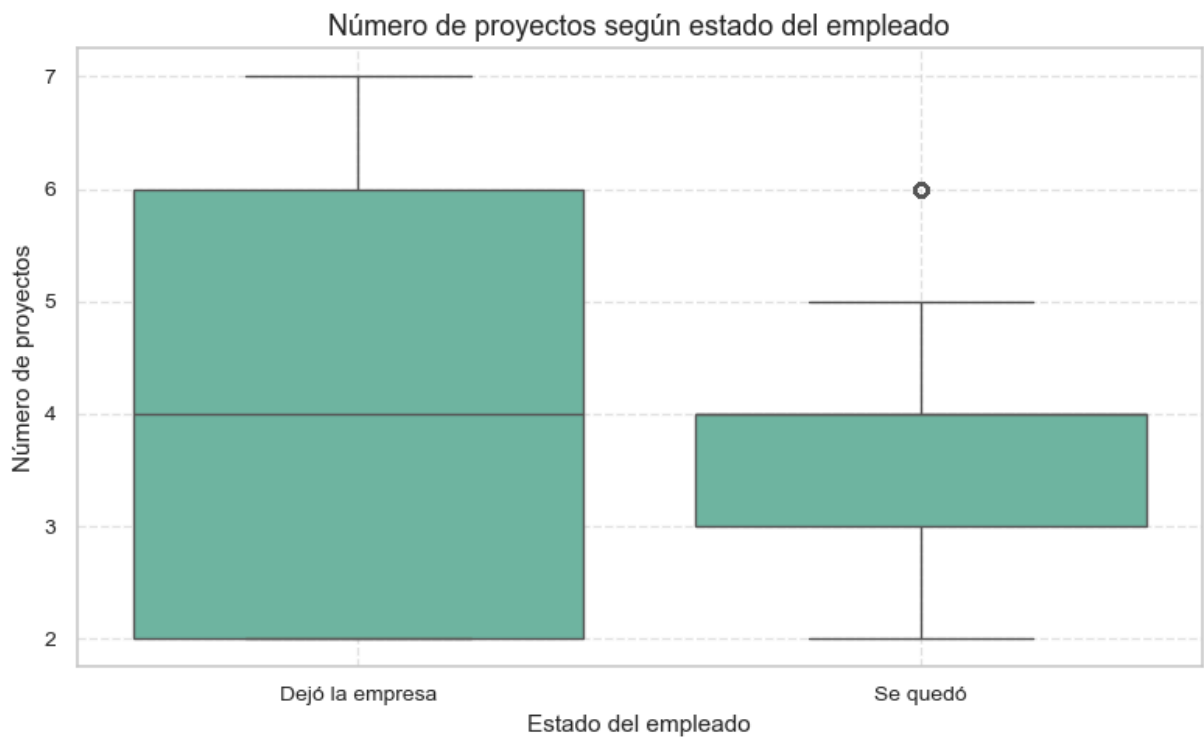
# Diccionario de variables traducidas
columnas = {
    'satisfaction_level': 'Nivel de satisfacción',
    'last_evaluation': 'Última evaluación',
    'number_project': 'Número de proyectos',
    'average_monthly_hours': 'Horas mensuales promedio',
    'time_spend_company': 'Años en la empresa'
}

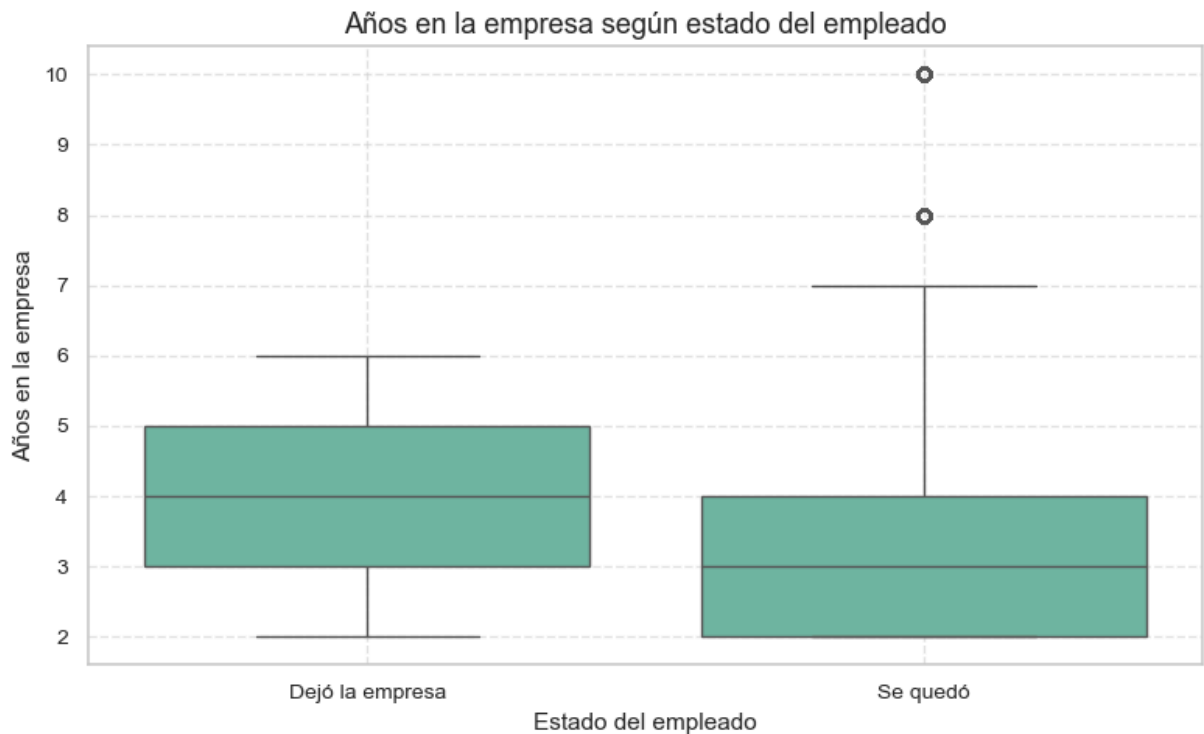
# Mapear valores de la variable 'left' para mostrar etiquetas claras
df['Estado empleado'] = df['left'].map({0: 'Se quedó', 1: 'Dejó la empresa'})

# Estilo visual general
sns.set(style='whitegrid', palette='Set2')

# Crear boxplots en bucle
for col, titulo in columnas.items():
    plt.figure(figsize=(8, 5))
    sns.boxplot(x='Estado empleado', y=col, data=df)
    plt.title(f'{titulo} según estado del empleado', fontsize=13)
    plt.xlabel('Estado del empleado', fontsize=11)
    plt.ylabel(titulo, fontsize=11)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()
```







```
In [84]: import seaborn as sns
import matplotlib.pyplot as plt

# Traducción de columnas
col_rename = {
    'satisfaction_level': 'Satisfacción',
    'last_evaluation': 'Evaluación',
    'number_project': 'Número de proyectos',
    'average_monthly_hours': 'Horas mensuales',
    'time_spend_company': 'Años en la empresa',
    'Work_accident': 'Accidente laboral',
    'left': 'Abandono',
    'promotion_last_5years': 'Promoción reciente'
}

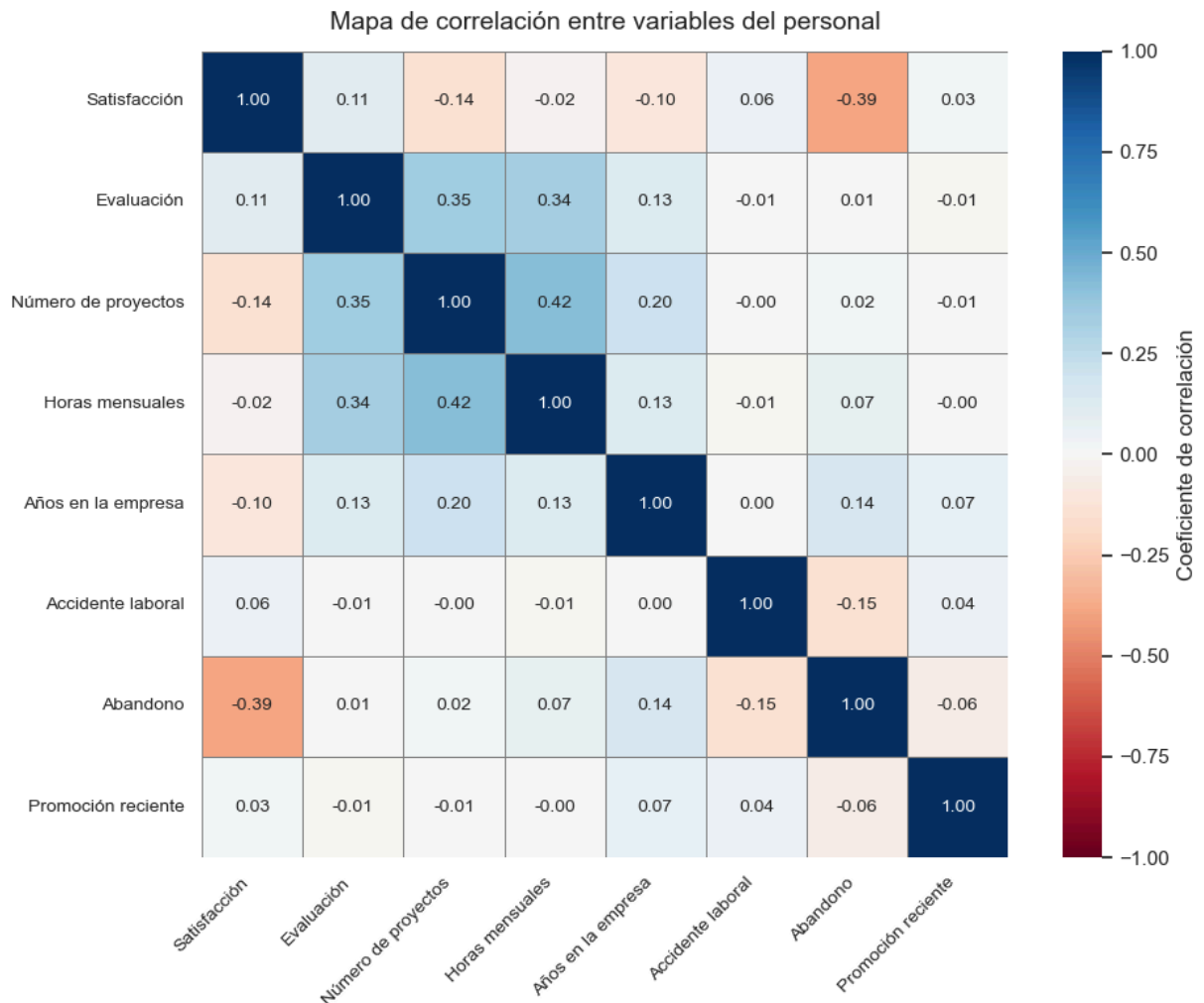
# Renombrar columnas
df_cor = df.rename(columns=col_rename)
corr = df_cor[list(col_rename.values())].corr()

# 📊 Crear heatmap con paleta invertida
plt.figure(figsize=(10, 8))
sns.set(font_scale=1.0)
sns.heatmap(
    corr,
    annot=True,
    fmt=".2f",
    cmap="RdBu", # 🔄 Invertimos el esquema: Lo positivo es azul y lo negativo rojo
    vmin=-1, vmax=1,
    square=True,
    linewidths=0.5,
    linecolor='gray',
    annot_kws={"size": 10},
    cbar_kws={"label": "Coeficiente de correlación"}
)
```

```

)
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(rotation=0, fontsize=10)
plt.title('Mapa de correlación entre variables del personal', fontsize=14, pad=12)
plt.tight_layout()
plt.show()

```



```

In [31]: import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.patches import Patch

# Correlación con 'left'
corr_numericas = df.select_dtypes(include='number').corr()
correlaciones = corr_numericas['left'].drop('left').sort_values()
colores = ['steelblue' if val < 0 else 'salmon' for val in correlaciones]

# Crear gráfico
fig, ax = plt.subplots(figsize=(9, 5))
correlaciones.plot(kind='barh', color=colores, ax=ax)

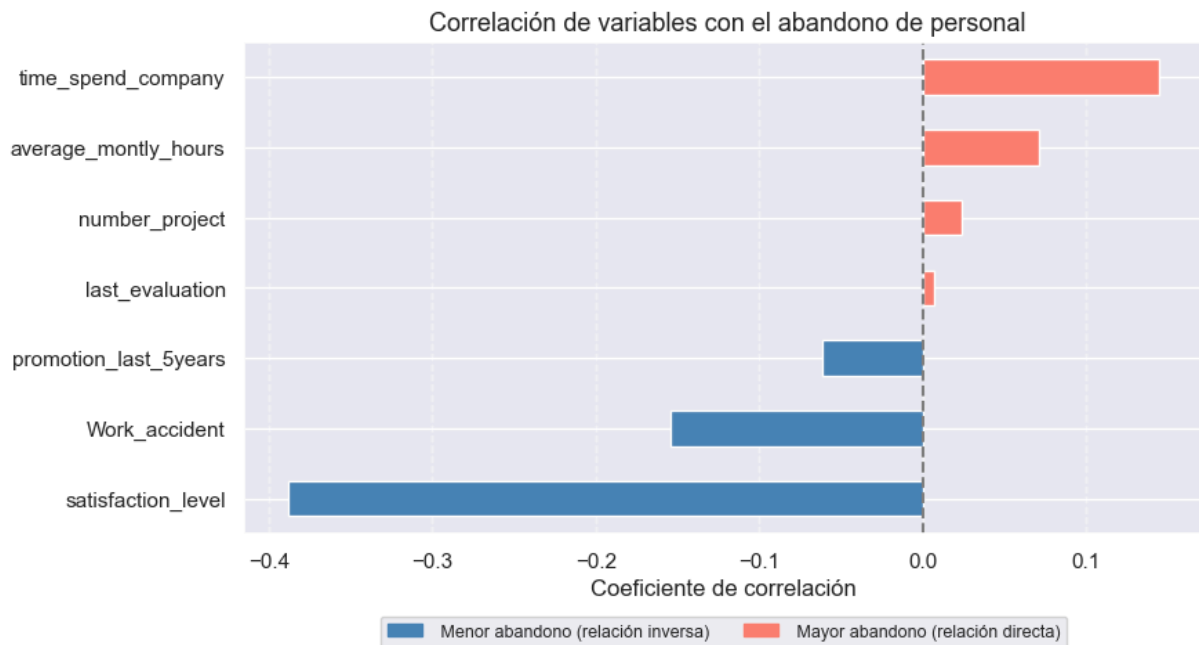
# Configurar título, ejes y grid
ax.set_title('Correlación de variables con el abandono de personal', fontsize=13)
ax.set_xlabel('Coeficiente de correlación')
ax.axvline(0, color='gray', linestyle='--')
ax.grid(True, axis='x', linestyle='--', alpha=0.5)

```



```
# ✓ Leyenda bien alineada
leyenda = [
    Patch(color='steelblue', label=' Menor abandono (relación inversa)'),
    Patch(color='salmon', label=' Mayor abandono (relación directa)')
]
ax.legend(handles=leyenda, loc='upper center', bbox_to_anchor=(0.5, -0.15),
          ncol=2, frameon=True, fontsize=9)

plt.tight_layout()
plt.show()
```



### A.3 – Preparación de datos

```
In [85]: import time
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

inicio = time.time()
print("🕒 Iniciando proceso de preparación de datos...")

# Codificar variables categóricas
le_dep = LabelEncoder()
le_sal = LabelEncoder()

df['Department'] = le_dep.fit_transform(df['Department'])
df['salary'] = le_sal.fit_transform(df['salary'])

print("✅ Variables categóricas codificadas correctamente.")

# Separar variables predictoras y objetivo
X = df.drop('left', axis=1)
y = df['left']

# División de datos
```

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

fin = time.time()
tiempo_total = fin - inicio

print("✅ División en entrenamiento y prueba completada.")
print(f"🕒 Tiempo total de procesamiento: {tiempo_total:.2f} segundos")
```

- 🕒 Iniciando proceso de preparación de datos...
- ✅ Variables categóricas codificadas correctamente.
- ✅ División en entrenamiento y prueba completada.
- 🕒 Tiempo total de procesamiento: 0.02 segundos

#### A.4 – Modelado

```
In [86]: import time
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

print("🕒 Iniciando preprocesamiento...")

# Codificación de variables categóricas
le_dep = LabelEncoder()
le_sal = LabelEncoder()
df['Department'] = le_dep.fit_transform(df['Department'])
df['salary'] = le_sal.fit_transform(df['salary'])

# Verificamos que 'left' esté en formato numérico
if df['left'].dtype == 'object':
    df['left'] = LabelEncoder().fit_transform(df['left'])

# Seleccionar solo columnas numéricas
X = df.drop('left', axis=1).select_dtypes(include=['int64', 'float64'])
y = df['left']

print(f"📊 Se utilizarán {X.shape[1]} variables numéricas para el modelo.")

# División de datos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("📊 Datos divididos en entrenamiento (80%) y prueba (20%).")

# Función para entrenar y evaluar
def entrenar_y_evaluar(nombre_modelo, modelo):
    print(f"\n🕒 Entrenando modelo: {nombre_modelo}")
    inicio = time.time()
    modelo.fit(X_train, y_train)
    tiempo = time.time() - inicio
    y_pred = modelo.predict(X_test)
    print(f"🕒 Tiempo de entrenamiento: {tiempo:.2f} segundos")
    print(f"📄 Resultados para {nombre_modelo}:\n")
    print(classification_report(y_test, y_pred, target_names=["Se quedó", "Se fue"])
```

```
# Entrenar modelos
entrenar_y_evaluar("Random Forest", RandomForestClassifier(random_state=42))
entrenar_y_evaluar("XGBoost", XGBClassifier(eval_metric='logloss', random_state=42))
entrenar_y_evaluar("Regresión Logística", LogisticRegression(max_iter=1000, random_
```

🔧 Iniciando preprocesamiento...

📊 Se utilizarán 9 variables numéricas para el modelo.  
 Datos divididos en entrenamiento (80%) y prueba (20%).

🚀 Entrenando modelo: Random Forest

🕒 Tiempo de entrenamiento: 0.98 segundos

📋 Resultados para Random Forest:

	precision	recall	f1-score	support
Se quedó	0.99	1.00	0.99	2294
Se fue	0.99	0.96	0.97	706
accuracy			0.99	3000
macro avg	0.99	0.98	0.98	3000
weighted avg	0.99	0.99	0.99	3000

🚀 Entrenando modelo: XGBoost

🕒 Tiempo de entrenamiento: 0.11 segundos

📋 Resultados para XGBoost:

	precision	recall	f1-score	support
Se quedó	0.99	1.00	0.99	2294
Se fue	0.98	0.96	0.97	706
accuracy			0.99	3000
macro avg	0.99	0.98	0.98	3000
weighted avg	0.99	0.99	0.99	3000

🚀 Entrenando modelo: Regresión Logística

🕒 Tiempo de entrenamiento: 0.39 segundos

📋 Resultados para Regresión Logística:

	precision	recall	f1-score	support
Se quedó	0.79	0.92	0.85	2294
Se fue	0.47	0.23	0.31	706
accuracy			0.76	3000
macro avg	0.63	0.57	0.58	3000
weighted avg	0.72	0.76	0.72	3000

```
In [95]: from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
```

```
# 1) Predicciones sobre el conjunto de test
```

```

y_pred = rf.predict(X_test)
y_prob = rf.predict_proba(X_test)[:, 1] # Para métricas probabilísticas

# 2) Métricas principales
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_prob)
matriz_conf = confusion_matrix(y_test, y_pred)
reporte = classification_report(y_test, y_pred, output_dict=False)

# 3) Mostrar resumen
print("🎯 Evaluación del modelo Random Forest")
print(f"✅ Accuracy      : {accuracy:.2%}")
print(f"✅ ROC AUC        : {roc_auc:.3f}")
print(f"\n📄 Classification Report:\n", reporte)

# 4) Mostrar matriz de confusión
import seaborn as sns
plt.figure(figsize=(5,4))
sns.heatmap(matriz_conf, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Se quedó', 'Se fue'],
            yticklabels=['Se quedó', 'Se fue'])
plt.title('Matriz de Confusión')
plt.xlabel('Predicción')
plt.ylabel('Valor real')
plt.tight_layout()
plt.show()

# 5) Curva ROC
RocCurveDisplay.from_estimator(rf, X_test, y_test)
plt.title('Curva ROC del modelo')
plt.grid(True, linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()

```

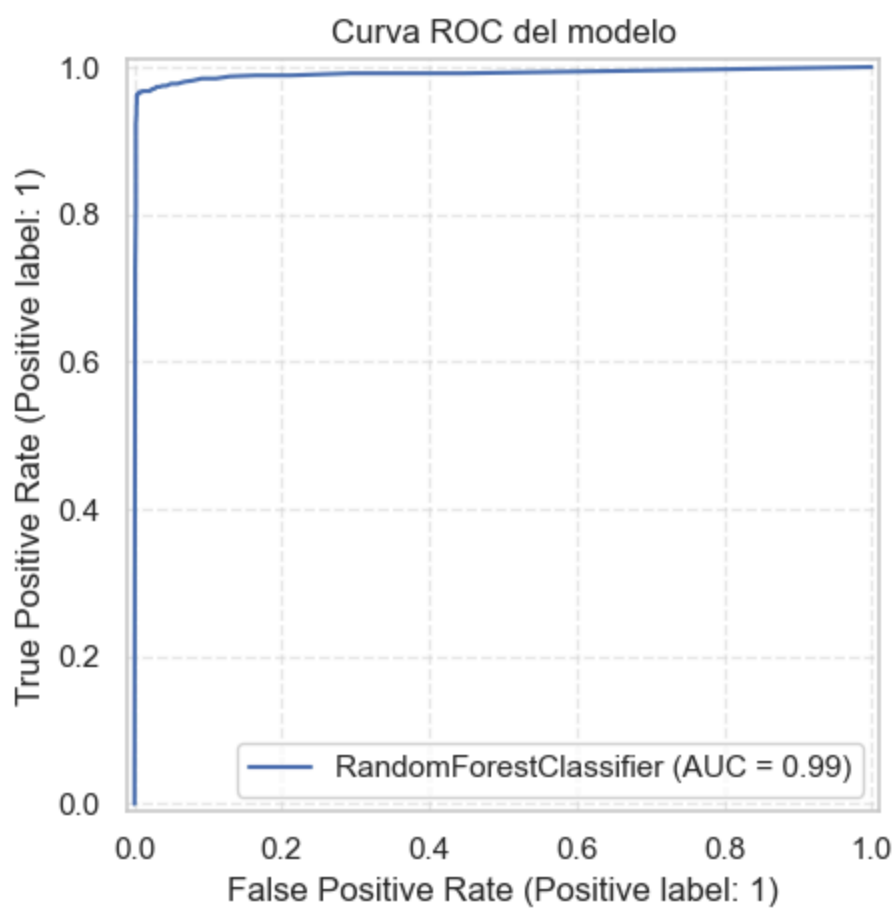
🎯 Evaluación del modelo Random Forest

✅ Accuracy : 98.83%

✅ ROC AUC : 0.991

📄 Classification Report:

	precision	recall	f1-score	support
0	0.99	1.00	0.99	2294
1	0.99	0.96	0.97	706
accuracy			0.99	3000
macro avg	0.99	0.98	0.98	3000
weighted avg	0.99	0.99	0.99	3000



```
In [87]: # Entrenar modelos y generar predicciones

# Random Forest
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
```

```

rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)

# XGBoost
from xgboost import XGBClassifier
xgb = XGBClassifier(eval_metric='logloss', random_state=42)
xgb.fit(X_train, y_train)
xgb_pred = xgb.predict(X_test)

# Regresión Logística
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(max_iter=1000, random_state=42)
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)

# Y ahora sí... La función para imprimir resultados
from sklearn.metrics import classification_report, accuracy_score

def imprimir_resultados(modelo_nombre, y_test, y_pred):
    print("\n" + "=" * 70)
    print(f"🔍 Resultados del modelo: {modelo_nombre.upper()}")
    print("-" * 70)

    accuracy = accuracy_score(y_test, y_pred)
    print(f"🎯 Precisión global (Accuracy): {accuracy:.4f}")


    if accuracy >= 0.95:
        print("🌟 Resultado excepcional: este modelo tiene un rendimiento casi perfecto")
    elif accuracy >= 0.85:
        print("👍 Buen desempeño, aunque hay espacio para perfeccionar la predicción")
    else:
        print("⚠️ El rendimiento es limitado. Podría mejorarse con más ajustes o datos")

    print("\n📊 Informe detallado por clase:\n")
    print(classification_report(y_test, y_pred, target_names=["Se quedó", "Se fue"]))
    print("=" * 70 + "\n")


# Ejecutar resultados
imprimir_resultados("Random Forest", y_test, rf_pred)
imprimir_resultados("XGBoost", y_test, xgb_pred)
imprimir_resultados("Regresión Logística", y_test, lr_pred)


```


=====

 Resultados del modelo: RANDOM FOREST

-----


 Precisión global (Accuracy): 0.9883

 Resultado excepcional: este modelo tiene un rendimiento casi perfecto.


 Informe detallado por clase:


	precision	recall	f1-score	support
Se quedó	0.99	1.00	0.99	2294
Se fue	0.99	0.96	0.97	706
accuracy			0.99	3000
macro avg	0.99	0.98	0.98	3000
weighted avg	0.99	0.99	0.99	3000


=====

 Resultados del modelo: XGBOOST

-----


 Precisión global (Accuracy): 0.9870

 Resultado excepcional: este modelo tiene un rendimiento casi perfecto.


 Informe detallado por clase:


	precision	recall	f1-score	support
Se quedó	0.99	1.00	0.99	2294
Se fue	0.98	0.96	0.97	706
accuracy			0.99	3000
macro avg	0.99	0.98	0.98	3000
weighted avg	0.99	0.99	0.99	3000


=====

 Resultados del modelo: REGRESIÓN LOGÍSTICA

-----

 Precisión global (Accuracy): 0.7583

 El rendimiento es limitado. Podría mejorarse con más ajustes o datos.

 Informe detallado por clase:

	precision	recall	f1-score	support
Se quedó	0.79	0.92	0.85	2294
Se fue	0.47	0.23	0.31	706
accuracy			0.76	3000
macro avg	0.63	0.57	0.58	3000
weighted avg	0.72	0.76	0.72	3000

```

=====

In [88]: import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as mtick

# -----
# Asumimos que ya tienes:
# nuevos_empleados → DataFrame con 'nombre', 'Probabilidad_de_irse' y 'Riesgo'
# colores          → dict {'Bajo':..., 'Medio':..., 'Alto':...}
# TOP_N            → número de empleados a mostrar
# -----

# 1) Seleccionar y ordenar Top N
top = (
    nuevos_empleados
    .nlargest(TOP_N, "Probabilidad_de_irse")
    .sort_values("Probabilidad_de_irse", ascending=True)
)

# 2) Estilo general
sns.set_style("whitegrid")

# 3) Figura con altura dinámica
height = TOP_N * 0.5 + 1.5 # 0.5" por barra + 1.5" de margen
fig, ax = plt.subplots(figsize=(10, height))

# 4) Dibujar barras horizontales con hue → activa La Leyenda
bars = sns.barplot(
    data=top,
    x="Probabilidad_de_irse",
    y="nombre",
    hue="Riesgo",
    palette=colores,
    dodge=False,
    ax=ax
)

# 5) Formatear ejes
ax.set_xlim(0, 1)
ax.xaxis.set_major_formatter(mtick.PercentFormatter(xmax=1))
ax.set_xlabel("Probabilidad de abandono", fontsize=12, weight="bold")
ax.set_ylabel("") # sin etiqueta redundante
ax.set_title(f"Top {TOP_N} empleados con mayor riesgo de abandono",
             fontsize=14, weight="bold")

# 6) Invertir Y para tener el mayor riesgo arriba
ax.invert_yaxis()

# 7) Anotar porcentaje al final de cada barra
for container in bars.containers:
    ax.bar_label(container, fmt="%.0f%%", padding=4, fontsize=10)

# 8) Leyenda externa

```

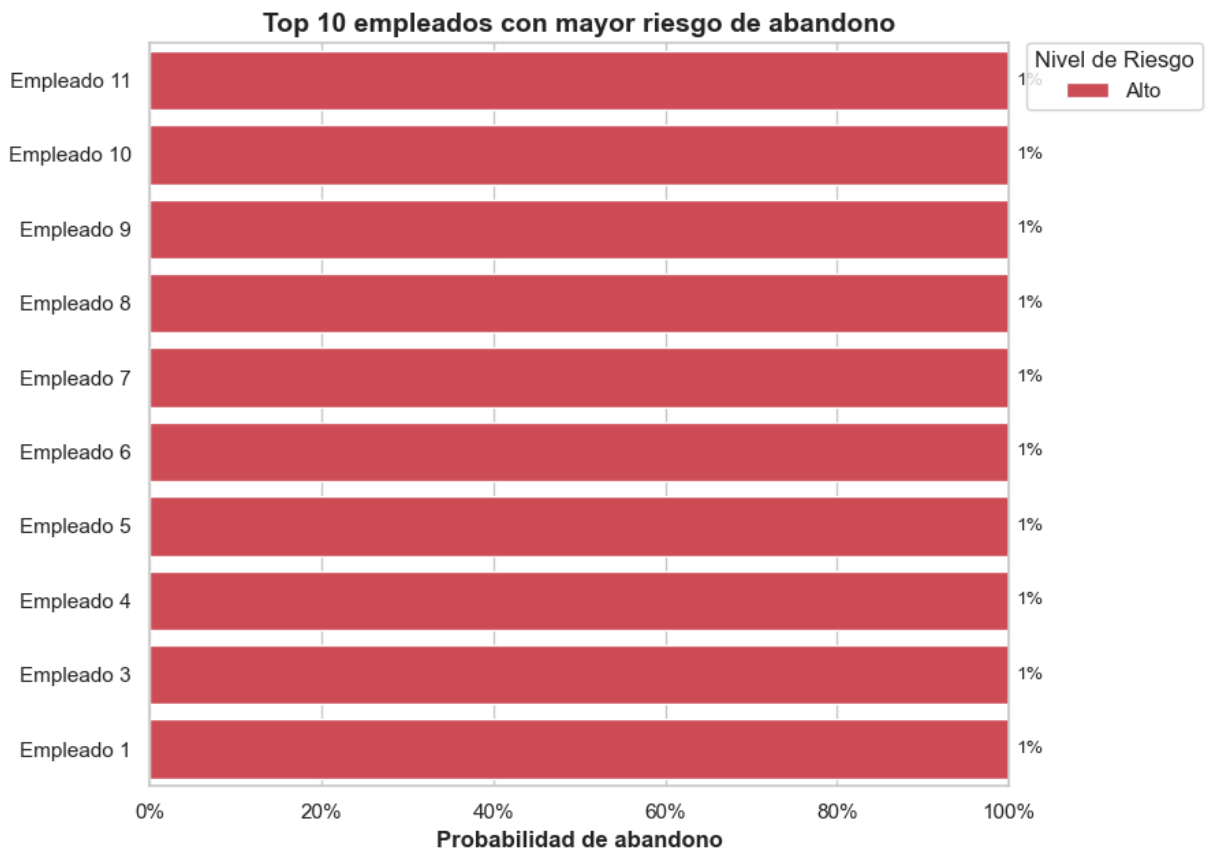


```

ax.legend(
    title="Nivel de Riesgo",
    bbox_to_anchor=(1.02, 1),
    loc="upper left",
    borderaxespad=0
)

# 9) Ajustes finales
plt.tight_layout()
plt.subplots_adjust(right=0.75) # dejar espacio para la leyenda
plt.show()

```



```

In [89]: import matplotlib.pyplot as plt

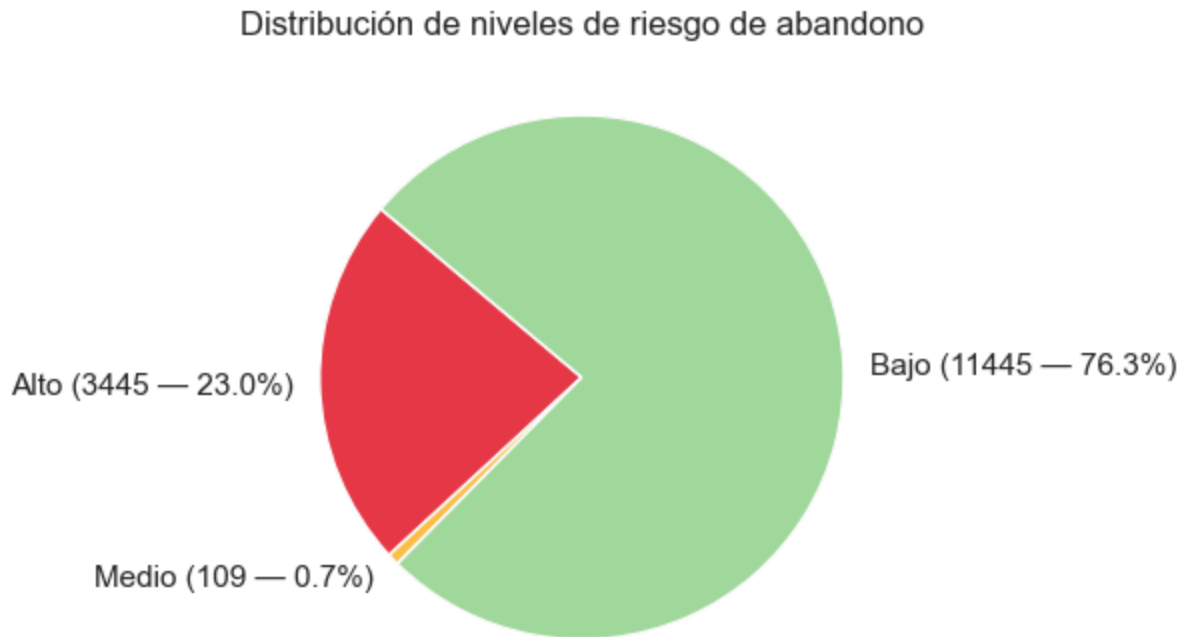
# 1) Contar cuántos empleados hay en cada nivel de Riesgo
counts = nuevos_empleados['Riesgo'].value_counts().reindex(['Alto', 'Medio', 'Bajo'])
total = counts.sum()

# 2) Colores iguales a los usados antes
colores = {'Bajo': '#A1D99B', 'Medio': '#FCBF49', 'Alto': '#E63946'}
mapped_colors = [colores[r] for r in counts.index]

# 3) Crear el pie chart
plt.figure(figsize=(6,6))
plt.pie(
    counts,
    labels=[f"{r} ({counts[r]} - {counts[r]/total*100:.1f}%)" for r in counts.index],
    colors=mapped_colors,
    startangle=140,
    wedgeprops={'edgecolor': 'white', 'linewidth': 1}
)

```

```
)
plt.title('Distribución de niveles de riesgo de abandono')
plt.tight_layout()
plt.show()
```



```
In [90]: import seaborn as sns
import matplotlib.pyplot as plt

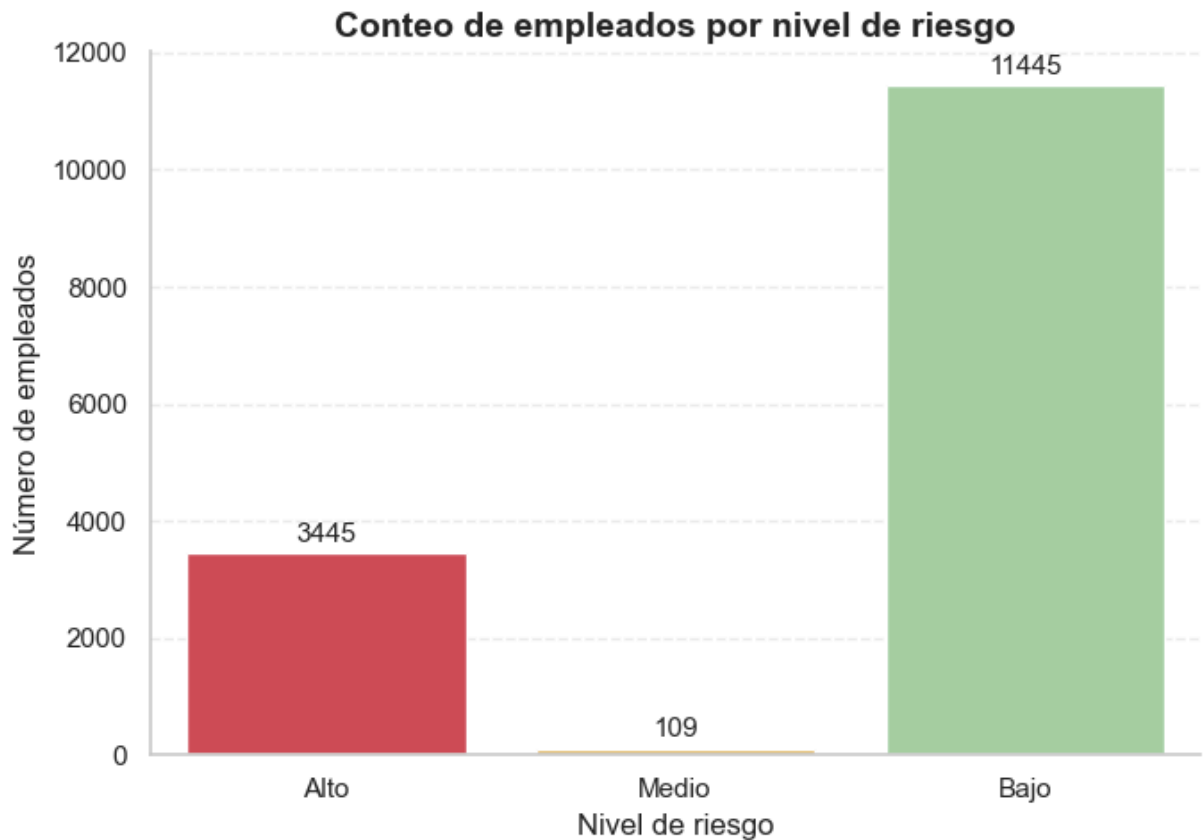
# Datos y colores
orden = ['Alto', 'Medio', 'Bajo']
valores = nuevos_empleados['Riesgo'].value_counts().reindex(orden, fill_value=0)

# Construir DataFrame temporal para graficar
import pandas as pd
df_plot = pd.DataFrame({
    'Riesgo': valores.index,
    'Conteo': valores.values
})

# Graficar de forma compatible con versiones nuevas
plt.figure(figsize=(7, 5))
barras = sns.barplot(
    data=df_plot,
    x='Riesgo',
    y='Conteo',
    hue='Riesgo',
    palette=colores,
    order=orden,
    legend=False
)

# Etiquetas sobre cada barra
for i, val in enumerate(df_plot['Conteo']):
    barras.text(i, val + 100, str(val), ha='center', va='bottom', fontsize=11)
```

```
# Estética
plt.title('Conteo de empleados por nivel de riesgo', fontsize=14, weight='bold')
plt.xlabel('Nivel de riesgo', fontsize=12)
plt.ylabel('Número de empleados', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.3)
sns.despine()
plt.tight_layout()
plt.show()
```



```
In [91]: print(nuevos_empleados['Riesgo'].value_counts())
```

```
Riesgo
Bajo    11445
Alto     3445
Medio     109
Name: count, dtype: int64
```

```
In [92]: import joblib
import pandas as pd

# -----
# 🧠 1. Guardar el modelo entrenado
# -----
joblib.dump(rf, 'modelo_random_forest.pkl')
print("✅ Modelo Random Forest guardado como 'modelo_random_forest.pkl'")

# -----
# 💾 2. Guardar Los LabelEncoders
# -----
```

```

joblib.dump(le_dep, 'encoder_departamento.pkl')
print("✅ Encoder de 'Department' guardado como 'encoder_departamento.pkl'")

joblib.dump(le_sal, 'encoder_salario.pkl')
print("✅ Encoder de 'salary' guardado como 'encoder_salario.pkl'")

# -----
# 📄 3. Guardar columnas usadas en entrenamiento
# -----
joblib.dump(list(X.columns), 'columnas_usadas_modelo.pkl')
print(f"✅ Columnas de entrada guardadas como 'columnas_usadas_modelo.pkl' ({len(X

# -----
# 📊 4. Guardar Los datos predichos con riesgo
# -----
archivo_pred = 'predicciones_empleados.csv'
nuevos_empleados.to_csv(archivo_pred, index=False)
print(f"✅ Archivo con predicciones guardado como '{archivo_pred}' - {nuevos_empleados

# -----
# 📋 Resumen final
# -----
print("\n🌟 Todo guardado correctamente. ¡Listo para reutilizar o desplegar en producción!")

```

✅ Modelo Random Forest guardado como 'modelo\_random\_forest.pkl'  
 ✅ Encoder de 'Department' guardado como 'encoder\_departamento.pkl'  
 ✅ Encoder de 'salary' guardado como 'encoder\_salario.pkl'  
 ✅ Columnas de entrada guardadas como 'columnas\_usadas\_modelo.pkl' (9 columnas)  
 ✅ Archivo con predicciones guardado como 'predicciones\_empleados.csv' – 14999 registros

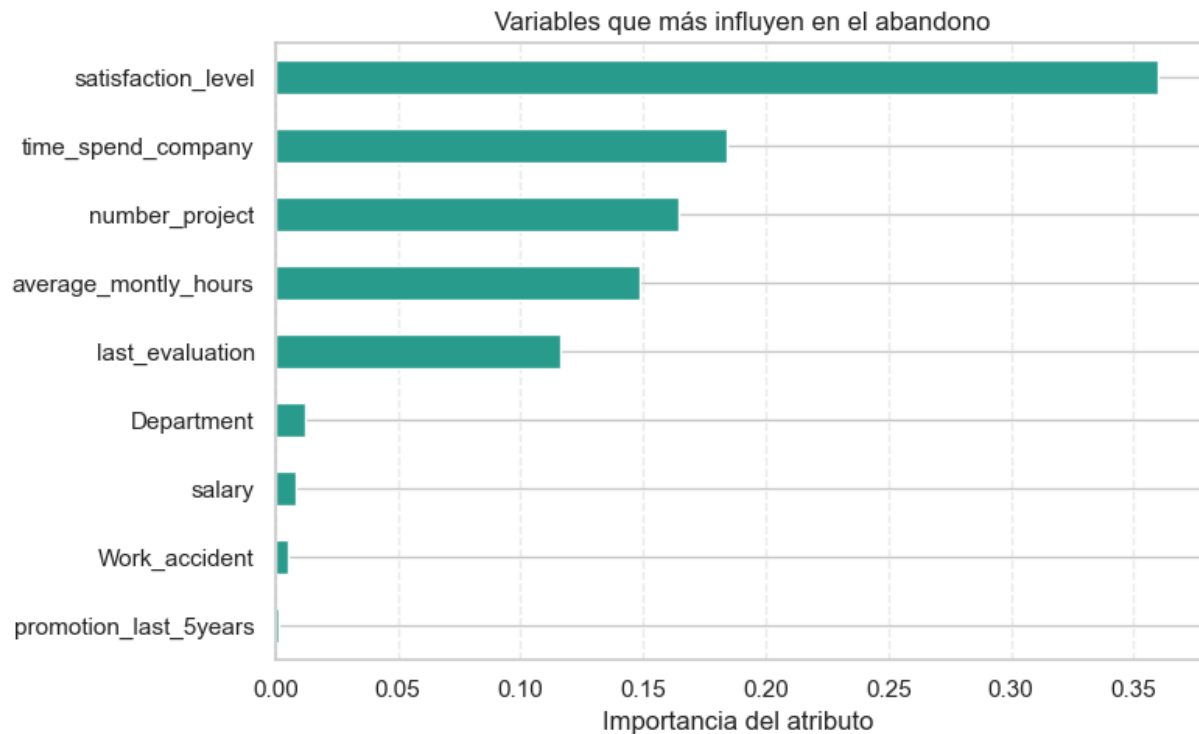
🌟 Todo guardado correctamente. ¡Listo para reutilizar o desplegar en producción!

```

In [94]: import pandas as pd
import matplotlib.pyplot as plt

importancias = pd.Series(rf.feature_importances_, index=X.columns).sort_values()
plt.figure(figsize=(8, 5))
importancias.plot(kind='barh', color='#2A9D8F')
plt.xlabel('Importancia del atributo')
plt.title(' Variables que más influyen en el abandono')
plt.grid(axis='x', linestyle='--', alpha=0.3)
plt.tight_layout()
plt.show()

```



In [97]: `!pip install openpyxl`

```
Collecting openpyxl
  Downloading openpyxl-3.1.5-py2.py3-none-any.whl.metadata (2.5 kB)
Collecting et_xmlfile (from openpyxl)
  Downloading et_xmlfile-2.0.0-py3-none-any.whl.metadata (2.7 kB)
Downloading openpyxl-3.1.5-py2.py3-none-any.whl (250 kB)
Downloading et_xmlfile-2.0.0-py3-none-any.whl (18 kB)
Installing collected packages: et-xmlfile, openpyxl
Successfully installed et-xmlfile-2.0.0 openpyxl-3.1.5
```

In [98]: `resumen = nuevos_empleados.groupby('Department')['Riesgo'].value_counts(normalize=True)`  
`resumen = (resumen * 100).round(1) # en porcentaje`  
`resumen.to_excel('resumen_riesgo_por_departamento.xlsx')`

In [ ]: