INGENIERIA DEL DATO

BOXPLOT

```
import matplotlib.pyplot as plt
columnas_boxplot = [
    'Nivel de importancia del salario',
    'Nivel de percepción de justicia salarial',
    'Nivel de ingreso bruto anual',
    'Antigüedad en el puesto actual',
    'Nivel de compromiso (engagement)',
    'Nivel de satisfacción general'
]
plt.figure(figsize=(10, 6))

box = plt.boxplot(df[columnas_boxplot].values,
                  patch_artist=True,
                  boxprops=dict(color='blue', facecolor='white'),
                  medianprops=dict(color='green'),
                  whiskerprops=dict(color='blue'),
                  capprops=dict(color='blue'),
                  flierprops=dict(markeredgecolor='black'),
                  widths=0.6)

plt.xticks(range(1, len(columnas_boxplot) + 1), columnas_boxplot, rotation=45,
ha='right')
plt.title('Boxplot de variables numéricas')
plt.grid(True)
plt.tight_layout()
```

```
plt.show()
```

ESTADÍSTICA DESCRIPTIVA

```
for col in df.select_dtypes(include='number').columns:
    print(f"\n--- {col} ---")
    print(f'Media: {df[col].mean():.2f}")
    print(f'Mediana: {df[col].median():.2f}")
    print(f'Desviación estándar: {df[col].std():.2f}")
    print(f'Mínimo: {df[col].min()}")
    print(f'Máximo: {df[col].max()}")
```

ANALISIS DEL DATO

BASE DE DATOS INE:

TASA DE ROTACIÓN EN EL TIEMPO

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_excel('BASE DE DATOS REAL.xlsx')

df['Fecha'] = pd.to_datetime(df['Fecha'])
plt.figure(figsize=(10, 5))
plt.plot(df['Fecha'], df['Tasa de Rotación (%)'], color='orange')
plt.title('Tasa de rotación en el tiempo')
plt.xlabel('Fecha')
plt.ylabel('%')
plt.grid(True)
plt.tight_layout()
```

```
plt.show()
```

MATRIZ DE CORRELACIÓN

```
import seaborn as sns

columnas_macro = [
    'Coste laboral mensual', 'Coste salarial mensual',
    'Salario medio Hombres', 'Salario medio Mujeres',
    'Número de Vacantes', 'Tasa de paro - Ambos sexos (%)',
    'Tasa de Temporalidad (%)', 'Tasa de Rotación (%)', 'PIB'
]

corr = df[columnas_macro].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='RdBu_r', center=0, square=True,
            fmt=".2f", linewidths=0.5, cbar_kws={'label': 'Correlación'})
plt.title('Correlaciones entre variables macroeconómicas')
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

BASE DE DATOS FINAL:

```
import pandas as pd
df = pd.read_excel('/content/BASE DE DATOS FINALISIMA TFG.xlsx')
df.head()
```

```
X = df.drop('Intención_Permanencia', axis=1)
y = df['Intención_Permanencia']
from sklearn.model_selection import train_test_split

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

MATRIZ DE CORRELACIÓN DE LA INTENCIÓN DE PERMANENCIA

```
import matplotlib.pyplot as plt
columnas_boxplot = [
    'Nivel de importancia del salario',
    'Nivel de percepción de justicia salarial',
    'Nivel de ingreso bruto anual',
    'Antigüedad en el puesto actual',
    'Nivel de compromiso (engagement)',
    'Nivel de satisfacción general'
]
plt.figure(figsize=(10, 6))

box = plt.boxplot(df[columnas_boxplot].values,
                  patch_artist=True,  # Necesario para aplicar color a cada parte
                  boxprops=dict(color='blue', facecolor='white'),
                  medianprops=dict(color='green'),
                  whiskerprops=dict(color='blue'),
                  capprops=dict(color='blue'),
                  flierprops=dict(markeredgecolor='black'),
                  widths=0.6)
```

```
plt.xticks(range(1, len(columnas_boxplot) + 1), columnas_boxplot, rotation=45,
ha='right')

plt.title('Boxplot de variables numéricas')

plt.grid(True)

plt.tight_layout()

plt.show()
```

REGRESIÓN LOGÍSTICA

```
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import classification_report, roc_auc_score, roc_curve

import matplotlib.pyplot as plt


logreg = LogisticRegression(max_iter=1000)

logreg.fit(X_train, y_train)


y_pred_log = logreg.predict(X_test)

y_proba_log = logreg.predict_proba(X_test)[:, 1]


print(classification_report(y_test, y_pred_log))

print("AUC:", roc_auc_score(y_test, y_proba_log))
```

MATRIZ DE CONFUSIÓN

```
from sklearn.metrics import confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt


matriz = confusion_matrix(y_test, y_pred_log)

labels = ['No permanece', 'Permanece']
```

```
plt.figure(figsize=(6, 5))
sns.heatmap(matriz, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.title('Matriz de Confusión - Regresión Logística')
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.tight_layout()
plt.show()
```

CURVA ROC

```
fpr, tpr, _ = roc_curve(y_test, y_proba_log)
plt.plot(fpr, tpr)
plt.title('Curva ROC - Regresión Logística')
plt.xlabel('Falsos positivos')
plt.ylabel('Verdaderos positivos')
plt.grid()
plt.show()
from sklearn.ensemble import RandomForestClassifier
```

TOP 10 VARIABLES MÁS INFLUYENTES EN LA INTENCIÓN DE PERMANENCIA

Con la variable Riesgo_salida:

```
logreg_con = LogisticRegression(max_iter=1000)
logreg_con.fit(X_train, y_train)

coef_con = pd.Series(logreg_con.coef_[0], index=X.columns)
top_con = coef_con.abs().sort_values(ascending=False).head(10)
top_con_signed = coef_con[top_con.index]
```

```
plt.figure(figsize=(8, 5))
top_con_signed.sort_values().plot(kind='barh', color='teal')
plt.title('Top 10 Variables más Influyentes')
plt.xlabel('Valor del Coeficiente')
plt.tight_layout()
plt.show()
```

Sin la variable Riesgo_salida:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

X_sin_riesgo = X.drop('Riesgo_salida', axis=1)

X_train_sin, X_test_sin, y_train_sin, y_test_sin = train_test_split(X_sin_riesgo, y,
test_size=0.3, random_state=42)

logreg_sin = LogisticRegression(max_iter=1000)
logreg_sin.fit(X_train_sin, y_train_sin)

coeficientes = pd.Series(logreg_sin.coef_[0], index=X_sin_riesgo.columns)
top_coef = coeficientes.abs().sort_values(ascending=False).head(10)
top_coef_signed = coeficientes[top_coef.index]
```

```
plt.figure(figsize=(8, 5))
top_coef_signed.sort_values().plot(kind='barh', color='teal')
plt.title('Top 10 Variables más Influyentes en la Intención de Permanencia')
plt.xlabel('Valor del Coeficiente')
plt.tight_layout()
plt.show()
```

COEFICIENTES B Y EXP(B)

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train, y_train)

coef = logreg.coef_[0]
variables = X.columns

tabla_coef = pd.DataFrame({
    'Variable': variables,
    'Coeficiente (β)': coef,
    'exp(β)': np.exp(coef)
})

tabla_coef = tabla_coef.dropna()
```

```
top10 = tabla_coef.reindex(tabla_coef['Coeficiente
(β)'].abs().sort_values(ascending=False).head(10).index)

top10 = top10.sort_values(by='Coeficiente (β)',
ascending=False).reset_index(drop=True)

top10[['Coeficiente (β)', 'exp(β)']] = top10[['Coeficiente (β)', 'exp(β)']].round(4)

top10
```

RANDOM FOREST CLASSIFIER

```
rf = RandomForestClassifier(random_state=42)

rf.fit(X_train, y_train)


y_pred_rf = rf.predict(X_test)

y_proba_rf = rf.predict_proba(X_test)[:, 1]


print(classification_report(y_test, y_pred_rf))

print("AUC:", roc_auc_score(y_test, y_proba_rf))
```

MATRIZ DE CONFUSIÓN

```
matriz_rf = confusion_matrix(y_test, y_pred_rf)


plt.figure(figsize=(6, 5))

sns.heatmap(matriz_rf, annot=True, fmt='d', cmap='Blues',

        xticklabels=['No', 'Sí'], yticklabels=['No', 'Sí'])

plt.title('Matriz de Confusión - Random Forest')

plt.xlabel('Predicción')

plt.ylabel('Real')

plt.tight_layout()

plt.show()
```

CURVA ROC

```
fpr, tpr, _ = roc_curve(y_test, y_proba_rf)
plt.plot(fpr, tpr)
plt.title('Curva ROC - Random Forest')
plt.xlabel('Falsos positivos')
plt.ylabel('Verdaderos positivos')
plt.grid()
plt.show()
```

GRID SEARCH

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

param_grid = {
    'n_estimators': [100, 250, 500, 750, 1000],
    'max_features': [5, 10, 'sqrt', 'log2']
}

rf = RandomForestClassifier(random_state=42)

grid = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='accuracy', return_train_score=False)
grid.fit(X_train, y_train)

resultados = pd.DataFrame(grid.cv_results_)
resultados_ordenados = resultados[['param_n_estimators', 'param_max_features', 'mean_test_score']]
resultados_ordenados = resultados_ordenados.sort_values(by='mean_test_score', ascending=False)
resultados_ordenados.head(10)
```

```
import seaborn as sns
import matplotlib.pyplot as plt


pivot = resultados_ordenados.pivot(index='param_max_features',
columns='param_n_estimators', values='mean_test_score')


plt.figure(figsize=(8, 6))
scatter = plt.scatter(
    resultados_ordenados['param_n_estimators'],
    resultados_ordenados['param_max_features'].astype(str),
    c=resultados_ordenados['mean_test_score'],
    cmap='viridis',
    s=200,
    edgecolors='k'
)
plt.xlabel('n_estimators')
plt.ylabel('max_features')
plt.title('Grid Search: Accuracy por combinación de hiperparámetros')
cbar = plt.colorbar(scatter)
cbar.set_label('Accuracy')
plt.tight_layout()
plt.show()
```

RANDOM SEARCH

```
from sklearn.model_selection import RandomizedSearchCV
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
param_dist = {
    'n_estimators': np.arange(100, 1000, 50),
    'max_features': np.arange(3, X.shape[1], 5)
}

rf = RandomForestClassifier(random_state=42)

random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_dist,
    n_iter=10,
    cv=5,
    scoring='accuracy',
    return_train_score=True,
    random_state=42
)

random_search.fit(X_train, y_train)

resultados_random = pd.DataFrame(random_search.cv_results_)

tabla_random = resultados_random[['param_n_estimators', 'param_max_features',
'mean_test_score', 'mean_train_score']]
tabla_random = tabla_random.sort_values(by='mean_test_score', ascending=False)

tabla_random.head(10)
x_vals = tabla_random['param_n_estimators']
y_vals = tabla_random['param_max_features']
scores = tabla_random['mean_test_score']
```

```
plt.figure(figsize=(10, 6))
scatter = plt.scatter(
    x=x_vals,
    y=y_vals,
    c=scores,
    s=400,
    cmap='viridis',
    edgecolors='k'
)

plt.xlabel('n_estimators')
plt.ylabel('max_features')
plt.title('Random Search: Accuracy por combinación de hiperparámetros')

cbar = plt.colorbar(scatter)
cbar.set_label('mean_test_score')

plt.grid(True)
plt.tight_layout()
plt.show()
```

IMPORTANCIA VARIABLES

```
# 3. Separar variable objetivo y predictoras
X = df.drop('Intención_Permanencia', axis=1)
y = df['Intención_Permanencia']

# 4. Eliminar 'Riesgo_salida'
```

```
X_sin_riesgo = X.drop('Riesgo_salida', axis=1)


# 5. Dividir en train/test

X_train, X_test, y_train, y_test = train_test_split(X_sin_riesgo, y, test_size=0.3,
random_state=42)


# 6. Entrenar modelo

modelo = LogisticRegression(max_iter=1000)

modelo.fit(X_train, y_train)


#  Obtener coeficientes

coef = pd.Series(modelo.coef_[0], index=X_sin_riesgo.columns)

top10 = coef.abs().sort_values(ascending=False).head(10)

top10_vars = coef.loc[top10.index]


# Crear gráfico

plt.figure(figsize=(10, 6))

top10_vars.sort_values().plot(kind='barh')

plt.title('Top 10 Variables más Influyentes (sin Riesgo_salida)')

plt.xlabel('Valor del Coeficiente')

plt.axvline(0, color='black', linewidth=0.8)

plt.tight_layout()

plt.show()


RANDOM SEARCH

from sklearn.model_selection import RandomizedSearchCV

import numpy as np


param_dist = {
```

```
    'n_estimators': np.arange(100, 500, 50),
    'max_features': np.arange(3, X.shape[1], 3)
}

random_search = RandomizedSearchCV(RandomForestClassifier(random_state=42),
                                   param_distributions=param_dist,
                                   n_iter=20, cv=5, random_state=42)
random_search.fit(X_train, y_train)

print("Mejores parámetros:", random_search.best_params_)
print("Mejor score:", random_search.best_score_)
```

IMPORTANCIA VARIABLE

Con riesgo_salida:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

importancia = rf.feature_importances_
importancia_df = pd.DataFrame({'Variable': X.columns, 'Importancia': importancia})
importancia_df = importancia_df.sort_values(by='Importancia',
ascending=False).head(10)

plt.figure(figsize=(8, 5))
sns.barplot(data=importancia_df, x='Importancia', y='Variable', color='steelblue')
```

```
plt.title('Top 10 variables más influyentes en la permanencia ')
plt.xlabel('Importancia')
plt.ylabel('Variable')
plt.tight_layout()
plt.show()
```

Sin Riesgo_salida:

```
X_sin_riesgo = X.drop('Riesgo_salida', axis=1)

X_train_sin, X_test_sin, y_train_sin, y_test_sin = train_test_split(X_sin_riesgo, y, test_size=0.3, random_state=42)

rf_sin = RandomForestClassifier(random_state=42)
rf_sin.fit(X_train_sin, y_train_sin)

importancia_sin = rf_sin.feature_importances_
importancia_df_sin = pd.DataFrame({'Variable': X_sin_riesgo.columns, 'Importancia': importancia_sin})
importancia_df_sin = importancia_df_sin.sort_values(by='Importancia', ascending=False).head(10)

plt.figure(figsize=(8, 5))
sns.barplot(data=importancia_df_sin, x='Importancia', y='Variable', color='steelblue')
plt.title('Top 10 variables más influyentes en la permanencia (sin Riesgo_salida)')
plt.xlabel('Importancia')
plt.ylabel('Variable')
plt.tight_layout()
plt.show()
```

GRAFICOS DEPENDENCIA PARCIAL

```
from sklearn.inspection import PartialDependenceDisplay
import matplotlib.pyplot as plt

variables_pdp = [
    'Nivel de satisfacción general',
    'Nivel de compromiso (engagement)',
    'Nivel del liderazgo ejercido por los directivos de la empresa',
    'Percepción de crecimiento profesional',
    'Nivel de flexibilidad laboral',
    'Importancia_Mejores beneficios',
    'Importancia_Teletrabajo',
    'Nivel de ingreso bruto anual',
    'Nivel de percepción de justicia salarial',
    'Nivel de importancia del salario'
]

for var in variables_pdp:
    fig, ax = plt.subplots(figsize=(6, 4))
    PartialDependenceDisplay.from_estimator(rf, X, [var], ax=ax)
    ax.set_title(f'Efecto parcial de {var}')
    plt.tight_layout()
    plt.show()
```

IMPORTANCIA VARIABLES ECONÓMICAS vs EMOCIONALES:

```
X = df.drop(columns=["Intención_Permanencia"])
y = df["Intención_Permanencia"]
```

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


modelo_rf = RandomForestClassifier(n_estimators=100, random_state=42)

modelo_rf.fit(X_train, y_train)


importancias = modelo_rf.feature_importances_

variables = X.columns

df_importancia = pd.DataFrame({'Variable': variables, 'Importancia': importancias})


variables_economicas = [

    'Nivel de ingreso bruto anual', 'Contrato Autónomo', 'Contrato Indefinido',

    'Contrato Temporal', 'Tamaño de la Empresa (10-50 empleados)',

    'Tamaño de la Empresa (51-250 empleados)', 'Tamaño de la Empresa ( <10 empleados)',

    'Tamaño de la Empresa ( >250 empleados)', 'Beneficio_Acciones, stock options','Fideliza_Salario'

    'Beneficio_Comidas subvenicionadas', 'Beneficio_Seguro médico','Importancia_Aumento salarial'

    'Beneficio_Plan de pensiones', 'Fideliza_Beneficios adicionales(tickets de comida, seguro médico...)' 'Beneficio_Transporte', 'Nivel de percepción de justicia salarial',  'Nivel de importancia del salario'

]


variables_emocionales = [v for v in variables if v not in variables_economicas]


econ_total = df_importancia[df_importancia['Variable'].isin(variables_economicas)]['Importancia'].sum()

emoc_total = df_importancia[df_importancia['Variable'].isin(variables_emocionales)]['Importancia'].sum()
```

```
df_tipo = pd.DataFrame({
    'Tipo': ['Económicas', 'Emocionales'],
    'Importancia': [econ_total, emoc_total]
})

plt.figure(figsize=(6,4))
sns.barplot(data=df_tipo, x='Tipo', y='Importancia', palette='Set2')
plt.title("Importancia Total de Variables Económicas vs. Emocionales")
plt.ylim(0, 1)
plt.ylabel("Importancia agregada (Random Forest)")
plt.show()
```

MEDIA DE LA IMPORTANCIA ECONOMICAS VS EMOCIONALES

```
variables_economicas = [
    'Nivel de ingreso bruto anual', 'Contrato Autónomo', 'Contrato Indefinido',
    'Contrato Temporal', 'Tamaño de la Empresa (10-50 empleados)',
    'Tamaño de la Empresa (51-250 empleados)', 'Tamaño de la Empresa ( <10 empleados)',
    'Tamaño de la Empresa ( >250 empleados)', 'Beneficio_Acciones, stock options',
    'Fideliza_Salario', 'Beneficio_Comidas subvenicionadas', 'Beneficio_Seguro médico',
    'Importancia_Aumento salarial', 'Beneficio_Plan de pensiones',
    'Fideliza_Beneficios adicionales(tickets de comida, seguro médico...)',
    'Beneficio_Transporte', 'Nivel de percepción de justicia salarial',
    'Nivel de importancia del salario'
]

variables_emocionales = [v for v in variables if v not in variables_economicas]
```

```
media_econ = df_importancia[df_importancia['Variable'].isin(variables_economicas)]['Importancia'].mean()
media_emoc = df_importancia[df_importancia['Variable'].isin(variables_emocionales)]['Importancia'].mean()


df_tipo_media = pd.DataFrame({
    'Tipo': ['Económicas', 'Emocionales'],
    'Importancia': [media_econ, media_emoc]
})


plt.figure(figsize=(6,4))
sns.barplot(data=df_tipo_media, x='Tipo', y='Importancia', palette='Set2')
plt.title("Media de Importancia por Variable - Económicas vs. Emocionales")
plt.ylabel("Media de Importancia (Random Forest)")
plt.ylim(0, df_tipo_media['Importancia'].max() + 0.005)
plt.tight_layout()
plt.show()
```