Información General

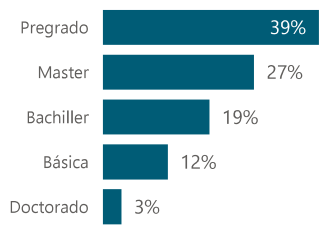
Pregrado
Nivel Educativo Promedio

$65 mil
Salario Mensual Promedio

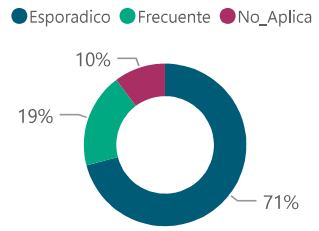
9
Distancia Casa-Trabajo Prom. KM

2
Promedio Años para Promoción

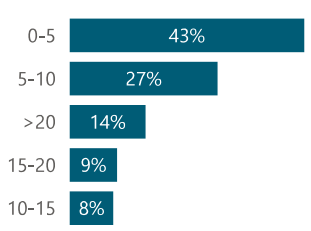
Nivel Educativo



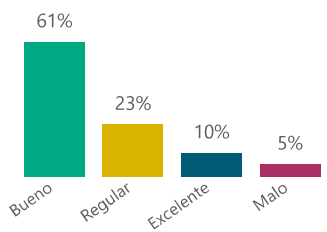
Viaje de Negocios



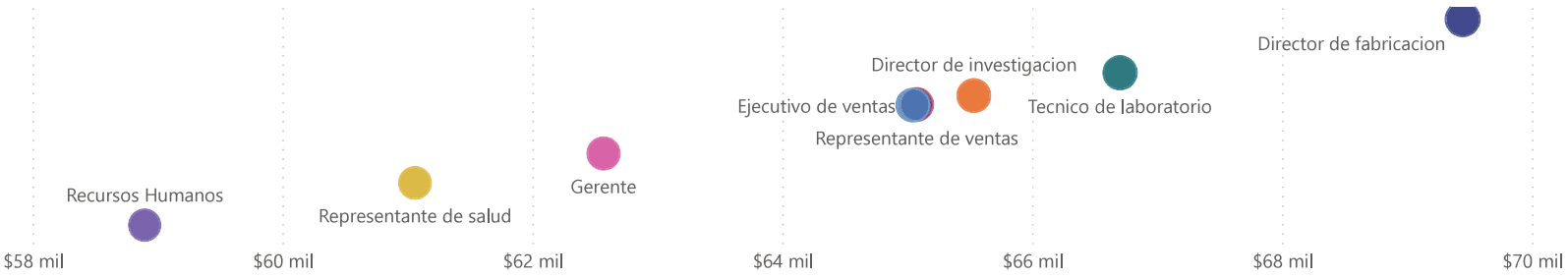
Distancia al Trabajo en KM



Balance Vida Laboral/Personal



Salario Mensual por Cargo



Desercion
- No
- Si

Area
- Investigacion & Desarrollo
- Recursos Humanos
- Ventas

Nivel Educativo
- Básica
- Bachiller
- Pregrado
- Master
- Doctorado

Rol
- Cientifico investigador
- Director de fabricacion
- Director de investigacion
- Ejecutivo de ventas
- Gerente
- Recursos Humanos
- Representante de salud
- Representante de ventas
- Tecnico de laboratorio

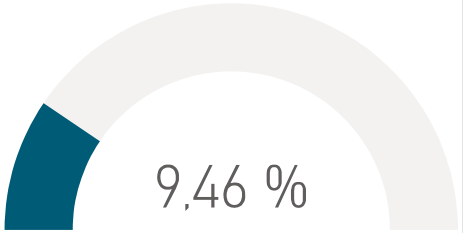
Indicadores de Desempeño

25
Promedio Anual de Ausencias

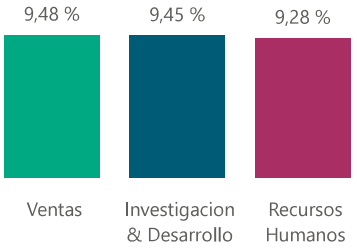
7
Prom. Años en la compañía

11
Promedio Años de Experiencia

Promedio de Ausentismo

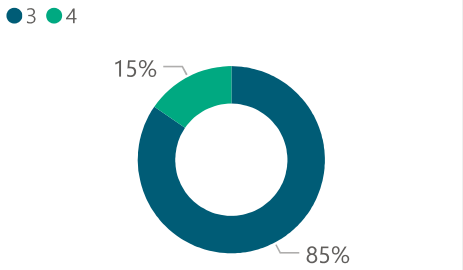


Ausentismo por Área

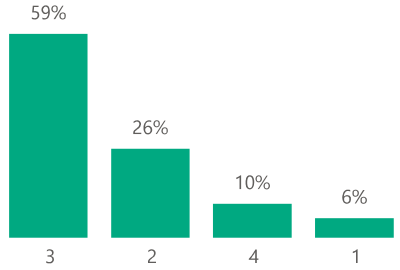


Promedio Horas Laborales

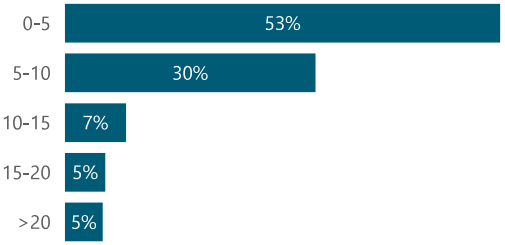
Desempeño



Compromiso del Empleado



Años en la Compañía



Desercion
No
Si

Area
Investigacion & Desarrollo
Recursos Humanos
Ventas

Nivel Educativo
Básica
Bachiller
Pregrado
Master
Doctorado

Rol
Cientifico investigador
Director de fabricacion
Director de investigacion
Ejecutivo de ventas
Gerente
Recursos Humanos
Representante de salud
Representante de ventas
Tecnico de laboratorio

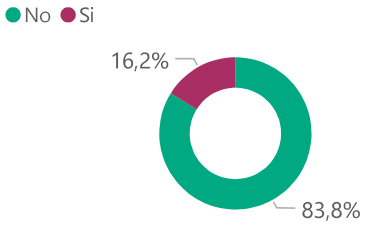
Reporte de Deserción

4.300	695
Total Empleados	Retiros

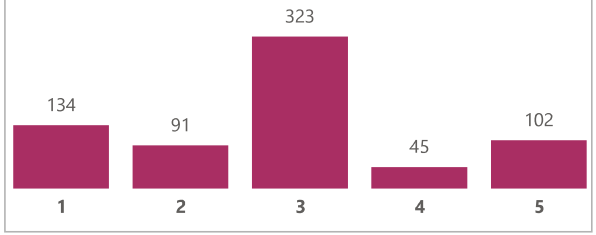
Retiros por Género

34	5
Edad Promedio Retirados	Prom. de Antigüedad Ret.

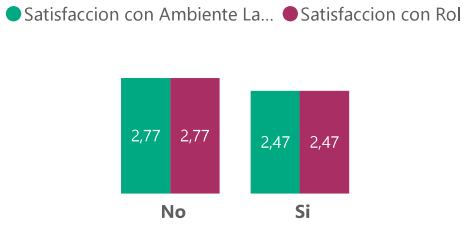
Deserción General



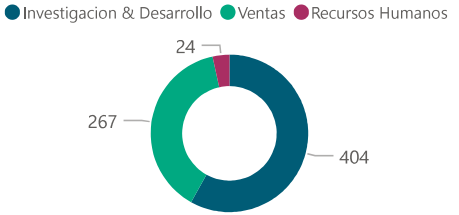
Retiros por nivel de cargo



Promedio encuestas segun Deserción

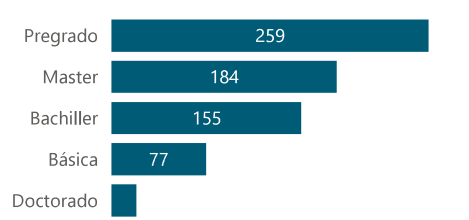


Deserción por Área



Deserción por Cargo

Deserción por Nivel Educativo



Genero
- Femenino
- Masculino

Area
- Investigacion & Desarrollo
- Recursos Humanos
- Ventas

Nivel Educativo
- Básica
- Bachiller
- Pregrado
- Master
- Doctorado

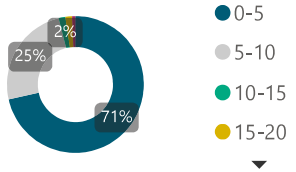
Rol
- Cientifico investigador
- Director de fabricacion
- Director de investigacion
- Ejecutivo de ventas
- Gerente
- Recursos Humanos
- Representante de salud
- Representante de ventas
- Tecnico de laboratorio

Alerta de Deserción

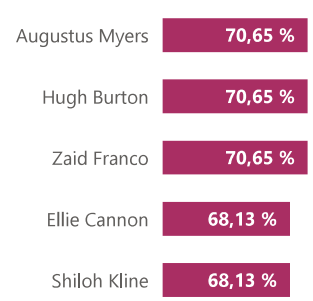
3.605	36,43 %	256
Total Emp. Activos	Riesgo Desercion Promedio	Empleados en Alerta

Riesgo de Retiro Promedio por Area

Alertas por Antiguedad (Años)



Top 5 Empleados en Riesgo



Alertas por Area

Nombre	Rol	Area	Antiguedad	Educación	Horas Prom/Dia	Distancia Casa Km	Satisfacción Laboral	Balance Laboral/Personal	Probabilidad Retiro
Aaden Bautista	Tecnico de laboratorio	Investigaci...	1	Pregrado	11	10	3	1	46,5 %
Aaliyah Christensen	Ejecutivo de ventas	Ventas	9	Bachiller	6	5	2	4	31,6 %
Aaliyah Higgins	Tecnico de laboratorio	Investigaci...	1	Bachiller	6	24	3	3	32,2 %
Abagail Barber	Ejecutivo de ventas	Ventas	8	Bachiller	6	3	3	4	18,5 %
Abbey Carr	Ejecutivo de ventas	Ventas	4	Master	6	8	3	3	32,5 %
Abbie West	Ejecutivo de ventas	Ventas	26	Bachiller	7	25	2	2	35,2 %
Abbigail Mendez	Representante de salud	Ventas	6	Pregrado	8	28	2	2	33,7 %
Abbigail Wilkerson	Ejecutivo de ventas	Ventas	7	Bachiller	11	10	1	3	47,0 %
Abby Andrews	Representante de ventas	Ventas	2	Pregrado	8	1	1	3	49,6 %
Abby Bolton	Representante de ventas	Ventas	1	Bachiller	7	9	1	2	46,6 %
Total			**7**		**8**	**9**	**3**	**3**	**36,4 %**

Area
- Investigacion & Desarrollo
- Recursos Humanos
- Ventas

Nivel Educativo
- Básica
- Bachiller
- Pregrado
- Master
- Doctorado

Rol
- Cientifico investigador
- Director de fabricacion
- Director de investigacion
- Ejecutivo de ventas
- Gerente
- Recursos Humanos
- Representante de salud
- Representante de ventas
- Tecnico de laboratorio

Modelamiento - Probabilidad de Deserción Candidatos

Preparación inicial

In [1]:
```
#Librerias numericas y graficos
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#Graficos
import plotly as py
import plotly.graph_objs as go

#Manejo de fechas
import datetime as dt

#Para Regresion Logistica
from sklearn.linear_model import LogisticRegression

#Modelos de Arboles
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

#Medidas
from sklearn import metrics
from sklearn import model_selection
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score

#Split X - y en conjuntos training y testing
from sklearn.model_selection import train_test_split

#Mostrar todas las columnas de frame
pd.set_option('display.max_columns', None)

#Asignar el formato de los decimales
np.set_printoptions(formatter={'float': lambda x: "{0:0.10f}".format(x)})
```

```
#Para datos desbalanceados
from pylab import rcParams

from imblearn.under_sampling import NearMiss
from imblearn.over_sampling import RandomOverSampler
from imblearn.combine import SMOTETomek
from imblearn.ensemble import BalancedBaggingClassifier
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import ADASYN

from collections import Counter
```

In [2]:
```
#Carga de datos
ruta = r'C:\Users\Jaime Andres Molina\Documents\Maestria\Proyecto Empresarial\Fuentes\01_Fuentes\all_attrition_modeling.csv'
#CSV
df = pd.read_csv (ruta, sep = ';')
```

In [3]:
```
#Tipos de datos
df.dtypes
```

Out[3]:

```
ID_empleado                   int64
Edad                          int64
Desercion                    object
Frecuencia_viaje             object
Area                         object
Distancia_Casa                int64
Nivel_Educativo               int64
Campo_Educativo              object
Recuento_Empleado             int64
Genero                       object
Nivel_Rol                     int64
Rol                          object
Estado_Civil                 object
Salario_Mensual             float64
Companias                     int64
Mayor18                      object
Porcentaje_Aumento_Salarial   int64
Horario                       int64
Nivel_Opcion_Acciones         int64
Anos_Trabajados               int64
Entrenamientos                int64
Antiguedad                    int64
Anos_Ultima_Promocion         int64
Anos_Lider_Actual             int64
Satisfaccion_Entorno          int64
Satisfaccion_Laboral          int64
Balance_VidaTrabajo           int64
CompromisoTrabajo             int64
Rendimiento                   int64
Prom_Horas                  float64
Porcentaje_Ausentismo       float64
Ausentismo                    int64
dtype: object
```

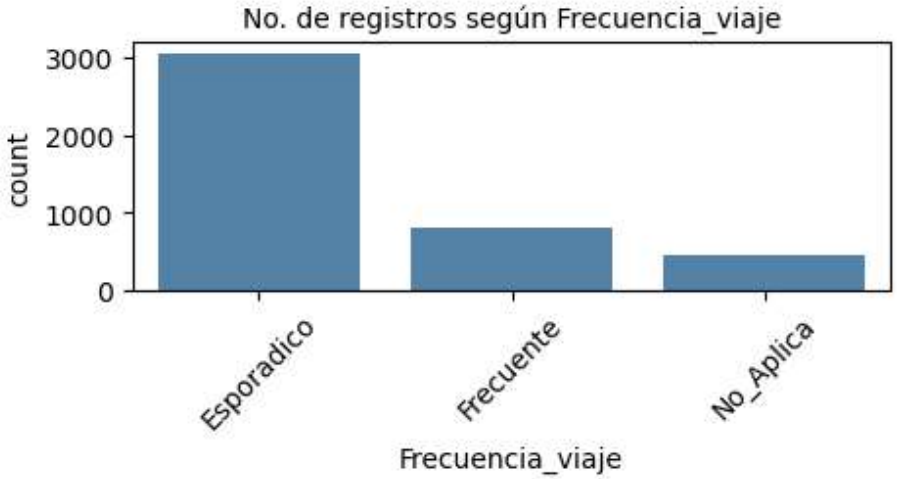
In [4]:

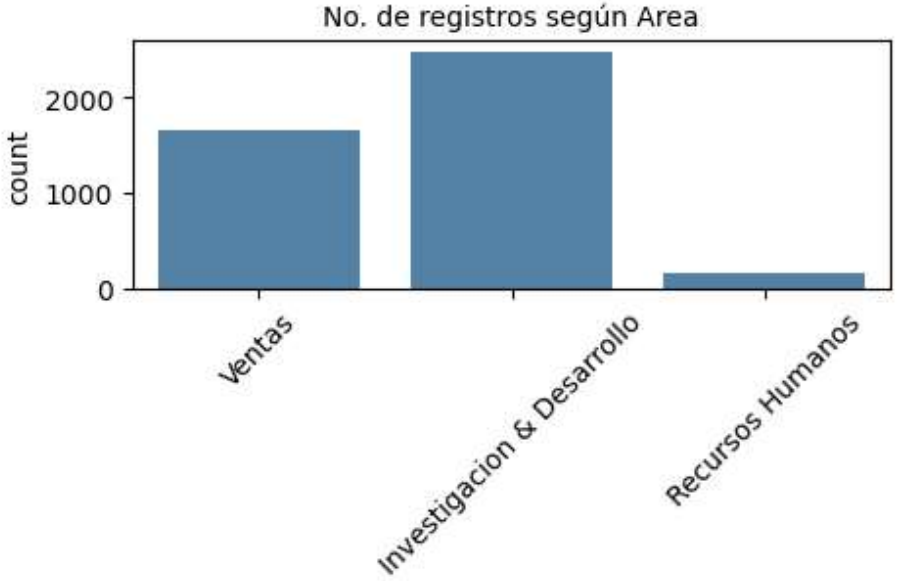
```
# Valores nulos por columna en porcentaje
df.isnull().sum()/df.shape[0]*100
```

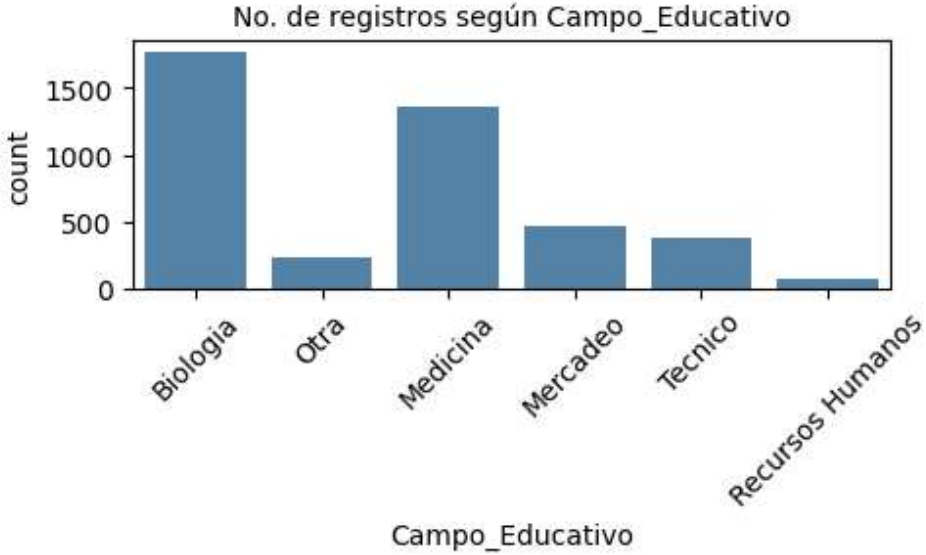
Out[4]:
```
ID_empleado                 0.0
Edad                        0.0
Desercion                   0.0
Frecuencia_viaje            0.0
Area                        0.0
Distancia_Casa              0.0
Nivel_Educativo             0.0
Campo_Educativo             0.0
Recuento_Empleado           0.0
Genero                      0.0
Nivel_Rol                   0.0
Rol                         0.0
Estado_Civil                0.0
Salario_Mensual             0.0
Companias                   0.0
Mayor18                     0.0
Porcentaje_Aumento_Salarial 0.0
Horario                     0.0
Nivel_Opcion_Acciones       0.0
Anos_Trabajados             0.0
Entrenamientos              0.0
Antiguedad                  0.0
Anos_Ultima_Promocion       0.0
Anos_Lider_Actual           0.0
Satisfaccion_Entorno        0.0
Satisfaccion_Laboral        0.0
Balance_VidaTrabajo         0.0
CompromisoTrabajo           0.0
Rendimiento                 0.0
Prom_Horas                  0.0
Porcentaje_Ausentismo       0.0
Ausentismo                  0.0
dtype: float64
```

In [5]:
```
#Histograma campos categoricos varias cajas
features=['Desercion', 'Frecuencia_viaje', 'Area','Campo_Educativo', 'Genero', 'Rol','Estado_Civil','Mayor18']
fig=plt.subplots(figsize=(11,14))
for i, j in enumerate(features):
    plt.subplot(4, 2, i+1)
    plt.subplots_adjust(hspace = 1.5)
    sns.countplot(x=j,data = df, color='steelblue')
    plt.xticks(rotation=45)
    plt.title("No. de registros según "+j, fontdict={'size': 10})
```

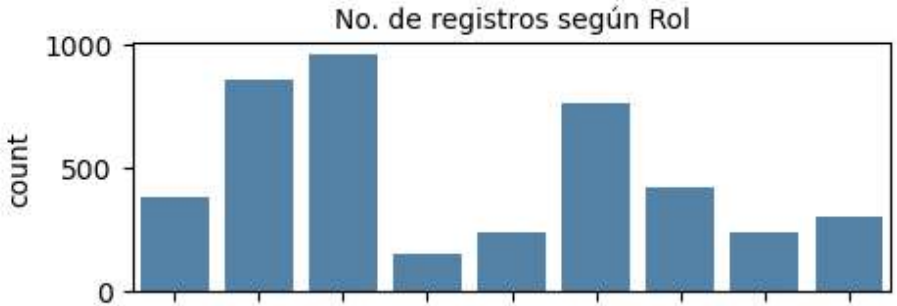

No. de registros según Desercion
count
3000
2000
1000
0
No
Si
Desercion

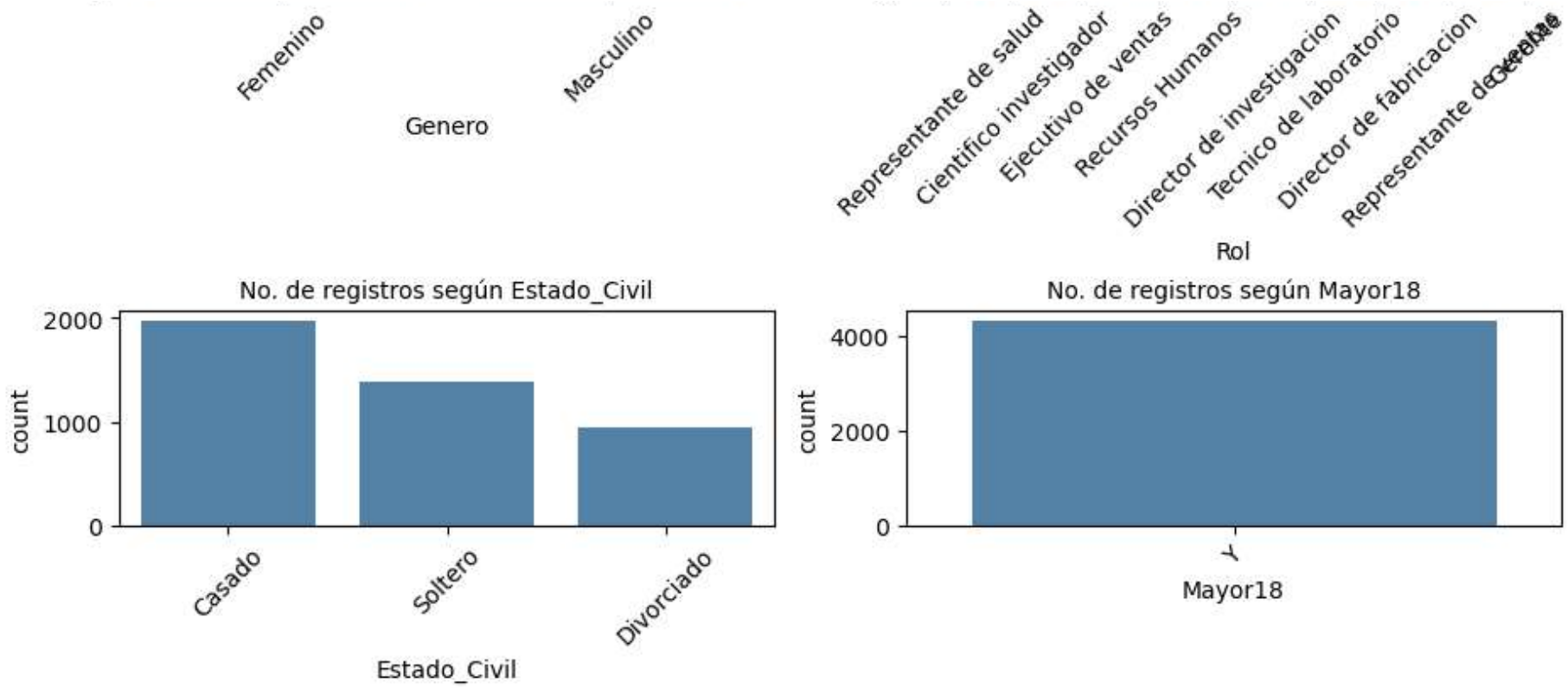

No. de registros según Frecuencia_viaje
count
3000
2000
1000
0
Esporadico
Frecuente
No_Aplica
Frecuencia_viaje


No. de registros según Area
count
2000
1000
0
Ventas
Investigacion & Desarrollo
Recursos Humanos
Area

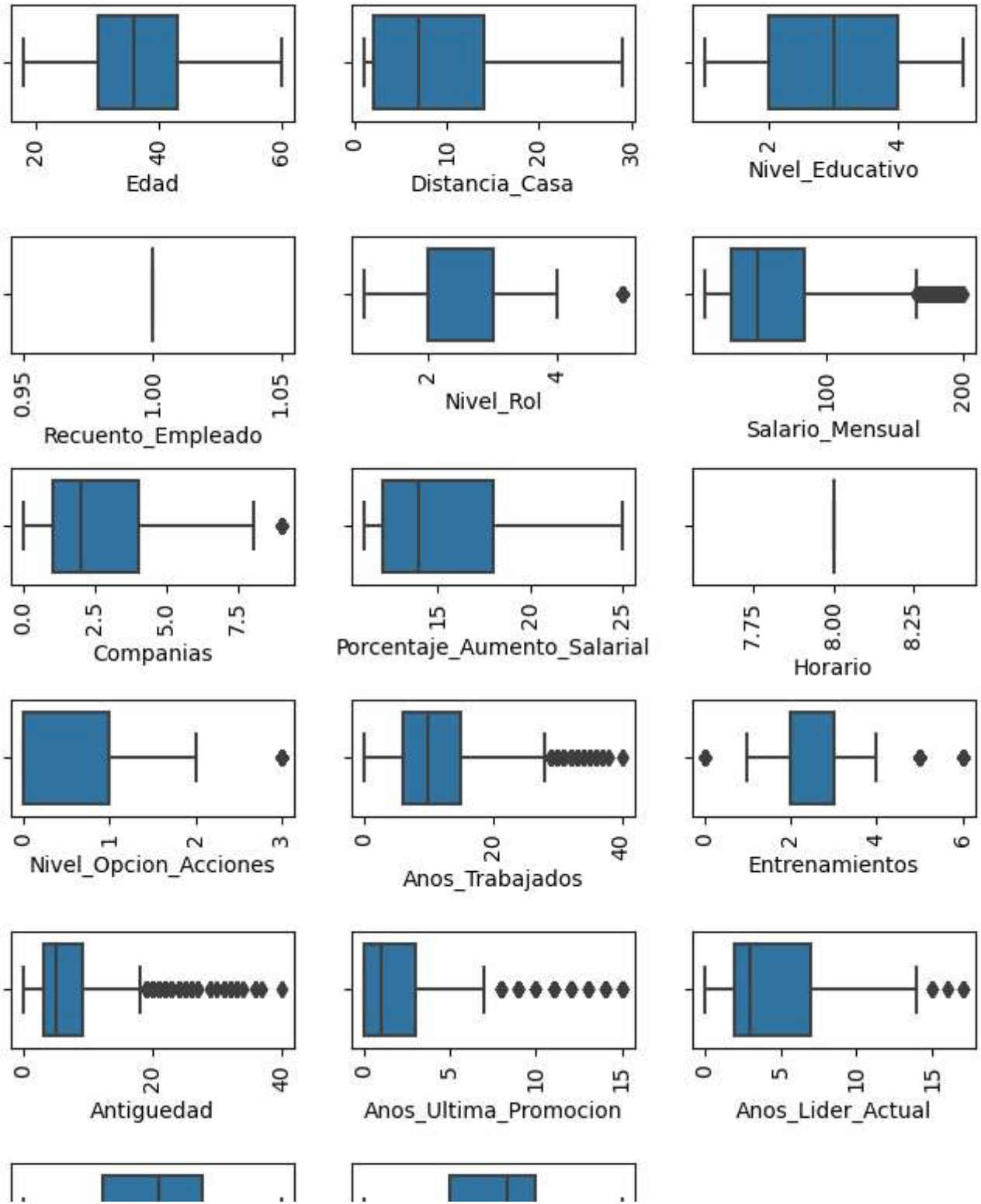

No. de registros según Campo_Educativo
count
1500
1000
500
0
Biologia
Otra
Medicina
Mercadeo
Tecnico
Recursos Humanos
Campo_Educativo

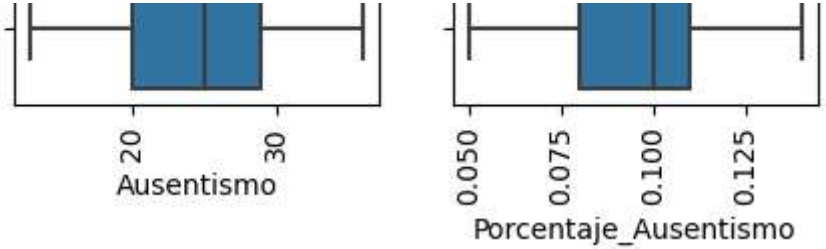

No. de registros según Genero
count
2000
1000
0


No. de registros según Rol
count
1000
500
0



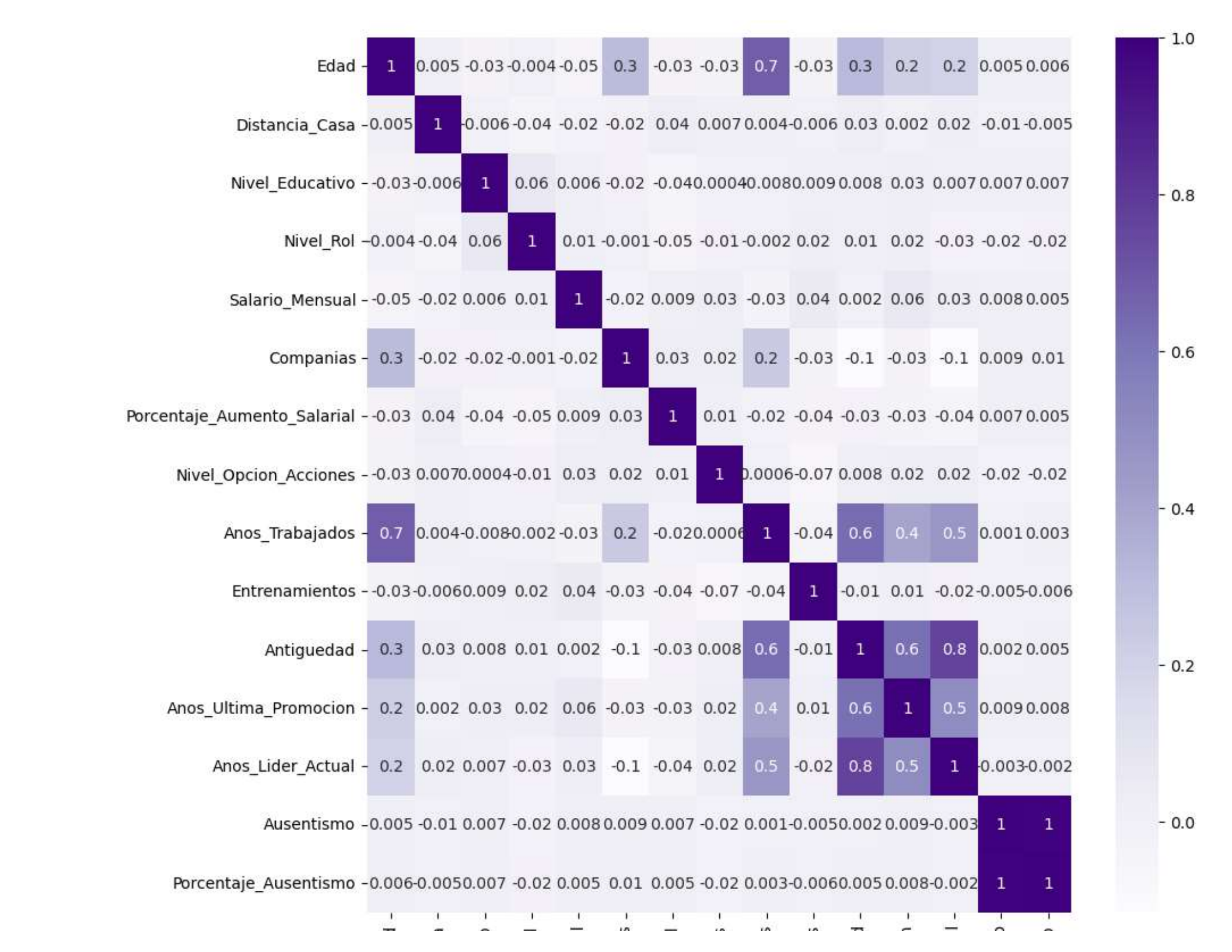
In [6]:
```
#Boxplots con seaborn multiples cajas
features=['Edad', 'Distancia_Casa', 'Nivel_Educativo', 'Recuento_Empleado', 'Nivel_Rol', 'Salario_Mensual',
       'Companias', 'Porcentaje_Aumento_Salarial', 'Horario', 'Nivel_Opcion_Acciones', 'Anos_Trabajados',
       'Entrenamientos', 'Antiguedad', 'Anos_Ultima_Promocion', 'Anos_Lider_Actual','Ausentismo',
        'Porcentaje_Ausentismo']
fig=plt.subplots(figsize=(8,30))
for i, j in enumerate(features):
    plt.subplot(16, 3, i+1)
    plt.subplots_adjust(hspace = 1.0)
    sns.boxplot(x=df[j])
    plt.xticks(rotation=90)
```

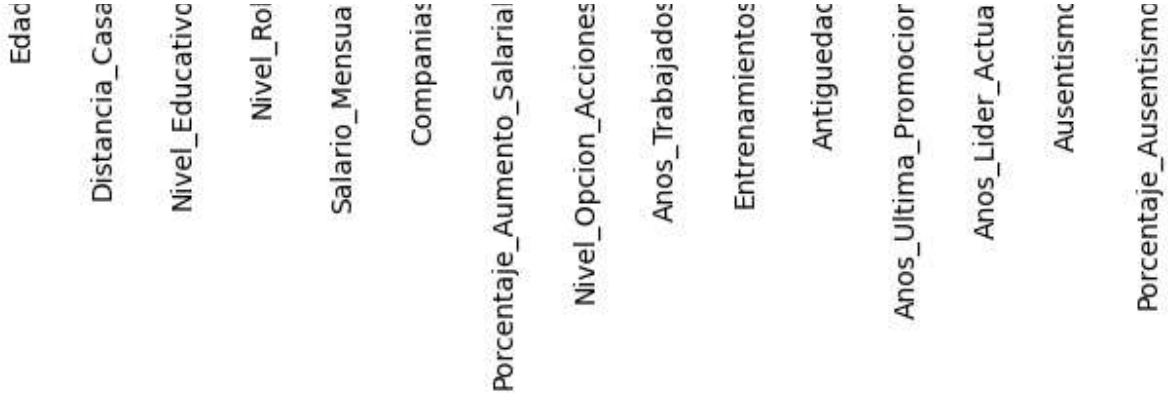

Edad
Distancia_Casa
Nivel_Educativo
Recuento_Empleado
Nivel_Rol
Salario_Mensual
Companias
Porcentaje_Aumento_Salarial
Horario
Nivel_Opcion_Acciones
Anos_Trabajados
Entrenamientos
Antiguedad
Anos_Ultima_Promocion
Anos_Lider_Actual



In [7]:
```
#Grafico de correlacion
corr_features=['Edad', 'Distancia_Casa', 'Nivel_Educativo', 'Nivel_Rol', 'Salario_Mensual',
               'Companias', 'Porcentaje_Aumento_Salarial', 'Nivel_Opcion_Acciones', 'Anos_Trabajados',
               'Entrenamientos', 'Antiguedad', 'Anos_Ultima_Promocion', 'Anos_Lider_Actual','Ausentismo',
               'Porcentaje_Ausentismo']

corr = df[corr_features].corr()
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values,
            #cmap=sns.diverging_palette(220, 10, as_cmap=True),
            annot=True, cmap="Purples", fmt = '.1g')
```

Out[7]:
```
<AxesSubplot:>
```


Edad 1 0.005 -0.03 -0.004 -0.05 0.3 -0.03 -0.03 0.7 -0.03 0.3 0.2 0.2 0.005 0.006
Distancia_Casa -0.005 1 -0.006 -0.04 -0.02 -0.02 0.04 0.007 0.004 -0.006 0.03 0.002 0.02 -0.01 -0.005
Nivel_Educativo -0.03 -0.006 1 0.06 0.006 -0.02 -0.04 0.0004 0.008 0.009 0.008 0.03 0.007 0.007 0.007
Nivel_Rol -0.004 -0.04 0.06 1 0.01 -0.001 -0.05 -0.01 -0.002 0.02 0.01 0.02 -0.03 -0.02 -0.02
Salario_Mensual -0.05 -0.02 0.006 0.01 1 -0.02 0.009 0.03 -0.03 0.04 0.002 0.06 0.03 0.008 0.005
Companias 0.3 -0.02 -0.02 -0.001 -0.02 1 0.03 0.02 0.2 -0.03 -0.1 -0.03 -0.1 0.009 0.01
Porcentaje_Aumento_Salarial -0.03 0.04 -0.04 -0.05 0.009 0.03 1 0.01 -0.02 -0.04 -0.03 -0.03 -0.04 0.007 0.005
Nivel_Opcion_Acciones -0.03 0.007 0.0004 -0.01 0.03 0.02 0.01 1 0.0006 -0.07 0.008 0.02 0.02 -0.02 -0.02
Anos_Trabajados 0.7 0.004 -0.008 -0.002 -0.03 0.2 -0.02 0.0006 1 -0.04 0.6 0.4 0.5 0.001 0.003
Entrenamientos -0.03 -0.006 0.009 0.02 0.04 -0.03 -0.04 -0.07 -0.04 1 -0.01 0.01 -0.02 -0.005 -0.006
Antiguedad 0.3 0.03 0.008 0.01 0.002 -0.1 -0.03 0.008 0.6 -0.01 1 0.6 0.8 0.002 0.005
Anos_Ultima_Promocion 0.2 0.002 0.03 0.02 0.06 -0.03 -0.03 0.02 0.4 0.01 0.6 1 0.5 0.009 0.008
Anos_Lider_Actual 0.2 0.02 0.007 -0.03 0.03 -0.1 -0.04 0.02 0.5 -0.02 0.8 0.5 1 -0.003 -0.002
Ausentismo -0.005 -0.01 0.007 -0.02 0.008 0.009 0.007 -0.02 0.001 -0.005 0.002 0.009 -0.003 1 1
Porcentaje_Ausentismo -0.006 -0.005 0.007 -0.02 0.005 0.01 0.005 -0.02 0.003 -0.006 0.005 0.008 -0.002 1 1
1.0
0.8
0.6
0.4
0.2
0.0



In [8]:
```
#Quitar columnas por nombre
cols_del = ['ID_empleado','Mayor18','Recuento_Empleado','Horario','Genero','Anos_Lider_Actual','Ausentismo',
            'Frecuencia_viaje','Porcentaje_Aumento_Salarial','Nivel_Opcion_Acciones','Entrenamientos','Antiguedad',
            'Anos_Ultima_Promocion','Satisfaccion_Entorno','Satisfaccion_Laboral','Balance_VidaTrabajo',
            'CompromisoTrabajo','Rendimiento','Prom_Horas','Porcentaje_Ausentismo']
df_in = df.loc[:, [name for name in df.columns if name not in cols_del]]
```

In [9]:
```
df_in.head()
```

Out[9]:

	Edad	Desercion	Area	Distancia_Casa	Nivel_Educativo	Campo_Educativo	Nivel_Rol	Rol	Estado_Civil	Salario_Mensual	Companias	A
0	51	No	Ventas	6	2	Biologia	2	Representante de salud	Casado	131.16	1	
1	31	Si	Investigacion & Desarrollo	10	1	Biologia	3	Cientifico investigador	Soltero	41.89	0	
2	32	No	Ventas	17	4	Otra	3	Ejecutivo de ventas	Casado	193.28	1	
3	38	No	Recursos Humanos	2	5	Biologia	3	Recursos Humanos	Casado	83.21	3	
4	32	No	Ventas	10	1	Medicina	3	Ejecutivo de ventas	Soltero	23.42	4	

In [10]:
```
#Dummificar variables seleccionadas
df_in= pd.get_dummies(df_in, columns=['Desercion', 'Area','Campo_Educativo', 'Rol', 'Estado_Civil'])
df_in.head()
```

Out[10]:

	Edad	Distancia_Casa	Nivel_Educativo	Nivel_Rol	Salario_Mensual	Companias	Anos_Trabajados	Desercion_No	Desercion_Si	Area_Investigacion & Desarrollo	Area_R H
0	51	6	2	2	131.16	1	1	1	0	0	
1	31	10	1	3	41.89	0	6	0	1	1	
2	32	17	4	3	193.28	1	5	1	0	0	
3	38	2	5	3	83.21	3	13	1	0	0	
4	32	10	1	3	23.42	4	9	1	0	0	

In [11]:
```
#Quitar columnas redundantes
cols_del_in = ['Desercion_No']
df_in = df_in.loc[:, [name for name in df_in.columns if name not in cols_del_in]]
```

Entrenamiento de Modelos

In [12]:
```
#Hay desbalanceo de clases en la variable objetivo
print(df_in.shape)
print(pd.value_counts(df_in['Desercion_Si'], sort = True))
```

```
(4300, 29)
0    3605
1     695
Name: Desercion_Si, dtype: int64
```

In [13]:
```
#Resultados con modelo normal sin estrategia de desbalance para tener punto de comparacion

#Definimos nuestras etiquetas y features
y = df_in['Desercion_Si']
X = df_in.drop('Desercion_Si', axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=16)
```

In [14]:
```
#Modelo de Regresion Logistica
logreg_model = LogisticRegression(C=1.0,penalty='l2',random_state=1,max_iter=10000)
logreg_model.fit(X_train, y_train)
```

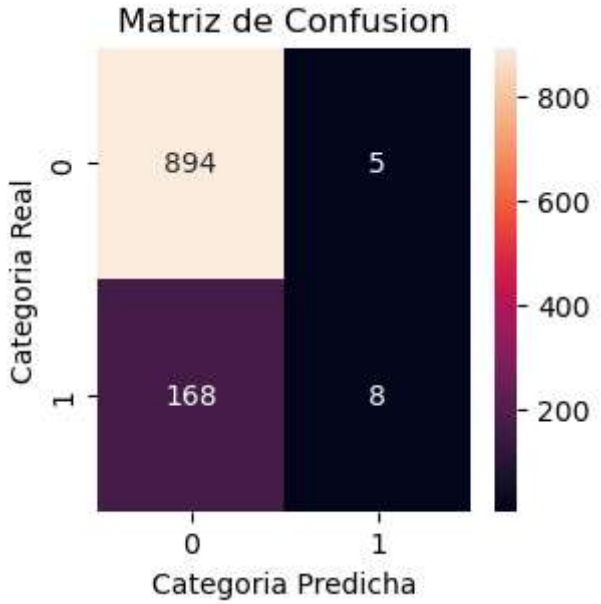
Out[14]:
```
LogisticRegression(max_iter=10000, random_state=1)
```

In [15]:
```
#Definimos funcion para mostrar los resultados
def mostrar_resultados(y_test, pred_y):
```

```
    conf_matrix = confusion_matrix(y_test, pred_y)
    plt.figure(figsize=(3, 3))
    sns.heatmap(conf_matrix, annot=True, fmt="d");
    plt.title("Matriz de Confusion")
    plt.ylabel('Categoria Real')
    plt.xlabel('Categoria Predicha')
    plt.show()
    print (classification_report(y_test, pred_y))
```

In [16]:
```
#Definimos funcion para medidas de comparación
def metricas_comparacion(y_test, pred_test, y_train, pred_train):
    #Accuracy
    accuracy_comparacion = metrics.accuracy_score(y_test, pred_test)
    print("Accuracy: ",accuracy_comparacion)

    #AUC test
    fpr, tpr, _ = metrics.roc_curve(y_test,  pred_test)
    auc_test = metrics.roc_auc_score(y_test, pred_test)
    print("AUC test: ",auc_test)

    #AUC train
    fpr_train, tpr_train, _train = metrics.roc_curve(y_train,  pred_train)
    auc_train = metrics.roc_auc_score(y_train, pred_train)
    print("AUC train: ",auc_train)

    #Delta AUC
    print("Delta AUC (overfitting): ",abs(auc_test-auc_train))
```

In [17]:
```
#Resultados de prediccion
pred_y = logreg_model.predict(X_test)
mostrar_resultados(y_test, pred_y)
#Recall de attrition de 18% a mejorar, activos en 98%
```



	precision	recall	f1-score	support
0	0.84	0.99	0.91	899
1	0.62	0.05	0.08	176
accuracy			0.84	1075
macro avg	0.73	0.52	0.50	1075
weighted avg	0.80	0.84	0.78	1075

Aplicando SMOTE sobre la base (oversampling)

In [18]:
```
sm = SMOTE(sampling_strategy='auto', k_neighbors=5, random_state=1)
X_sm, y_sm = sm.fit_resample(X, y)
```

In [19]:
```
#Nuevos registros por categoria
print(y_sm.shape)
y_sm.value_counts()
```

Out[19]:
```
(7210,)
0    3605
1    3605
Name: Desercion_Si, dtype: int64
```

In [20]:
```
#Dividir la base en train y test de SMOTE
X_train_sm, X_test_sm, y_train_sm, y_test_sm = train_test_split(X_sm, y_sm, test_size=0.25, random_state=28)
```

Aplicando ADASYN sobre la base (oversampling)

In [21]:
```
ada = ADASYN(sampling_strategy='auto', n_neighbors=5, random_state=1)
X_ada, y_ada = ada.fit_resample(X, y)
```

In [22]:
```
#Nuevos registros por categoria
print(y_ada.shape)
y_ada.value_counts()
```

Out[22]:
```
(7045,)
0    3605
1    3440
Name: Desercion_Si, dtype: int64
```

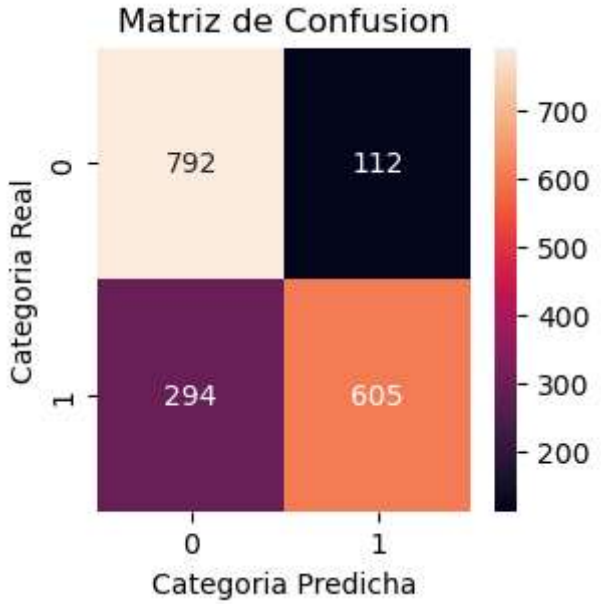
In [23]:
```
#Dividir la base en train y test de ADASYN
X_train_ada, X_test_ada, y_train_ada, y_test_ada = train_test_split(X_ada, y_ada, test_size=0.25, random_state=28)
```

Entrenamiento SMOTE Logistic Regression

In [24]:
```
#Modelo de Regresion Logistica SMOTE
logreg_model_sm = LogisticRegression(C=1.0,penalty='l2',random_state=1,max_iter=10000)
logreg_model_sm.fit(X_train_sm, y_train_sm)
```

Out[24]:
```
LogisticRegression(max_iter=10000, random_state=1)
```

In [25]:
```
#Prediccion sobre el test
pred_y_log_sm = logreg_model_sm.predict(X_test_sm)
mostrar_resultados(y_test_sm, pred_y_log_sm)
#Recall de attrition de 67%, activos en 88%

#Prediccion sobre el train
pred_y_log_sm_train = logreg_model_sm.predict(X_train_sm)
```



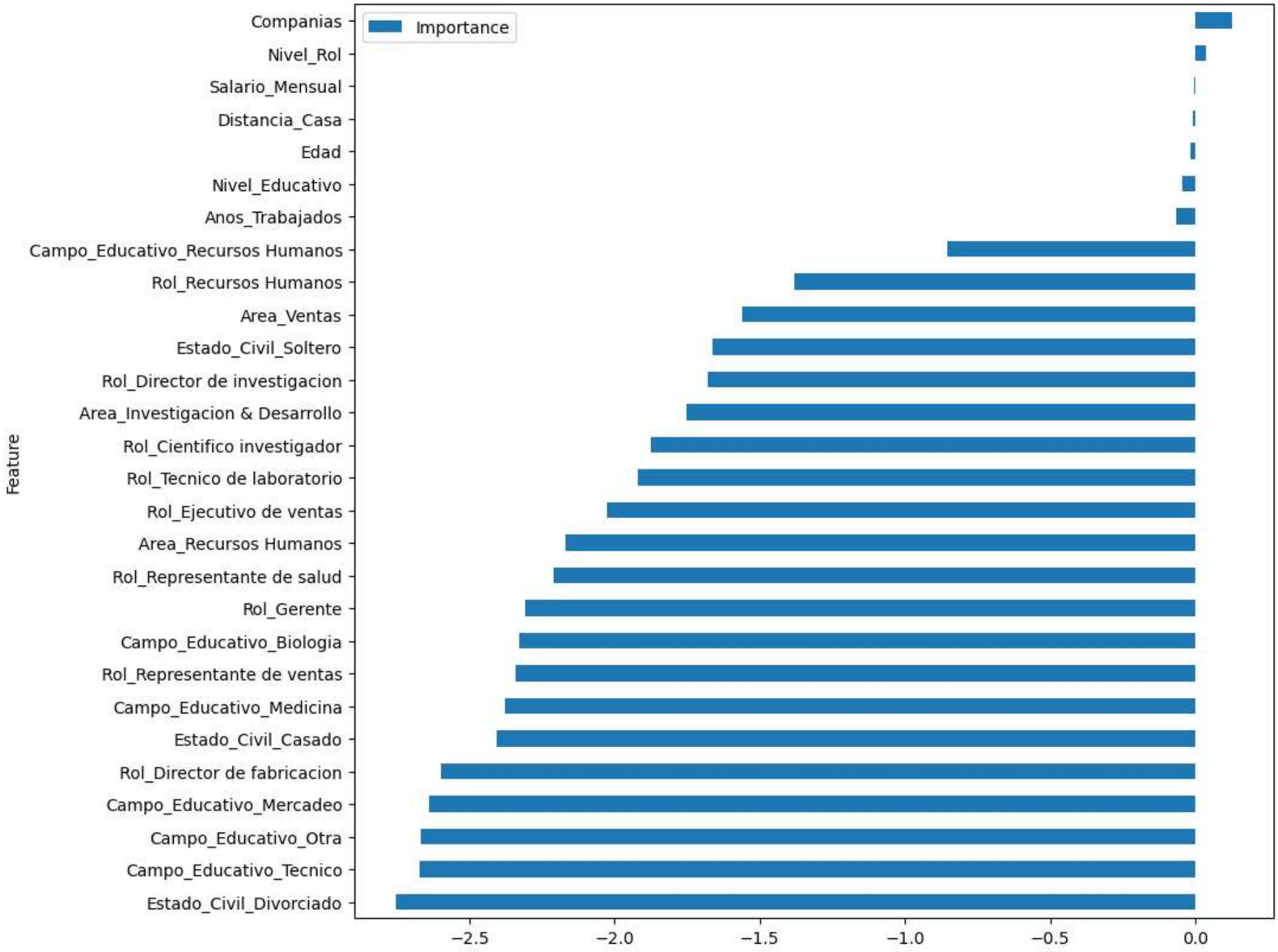
	precision	recall	f1-score	support
0	0.73	0.88	0.80	904
1	0.84	0.67	0.75	899
accuracy			0.77	1803
macro avg	0.79	0.77	0.77	1803
weighted avg	0.79	0.77	0.77	1803

In [26]:
```
metricas_comparacion(y_test_sm, pred_y_log_sm, y_train_sm, pred_y_log_sm_train)
```

```
Accuracy:  0.7748197448696617
AUC test:  0.774538080659927
AUC train:  0.7813070109261221
Delta AUC (overfitting):  0.006768930266195072
```

In [27]:
```
#Importancia de las variables
coefficients_log_sm = logreg_model_sm.coef_[0]

ft_importance_log_sm = pd.DataFrame({'Feature': X_sm.columns, 'Importance': coefficients_log_sm})
ft_importance_log_sm = ft_importance_log_sm.sort_values('Importance', ascending=True)
ft_importance_log_sm.plot(x='Feature', y='Importance', kind='barh', figsize=(10, 10))
```

Out[27]:
```
<AxesSubplot:ylabel='Feature'>
```



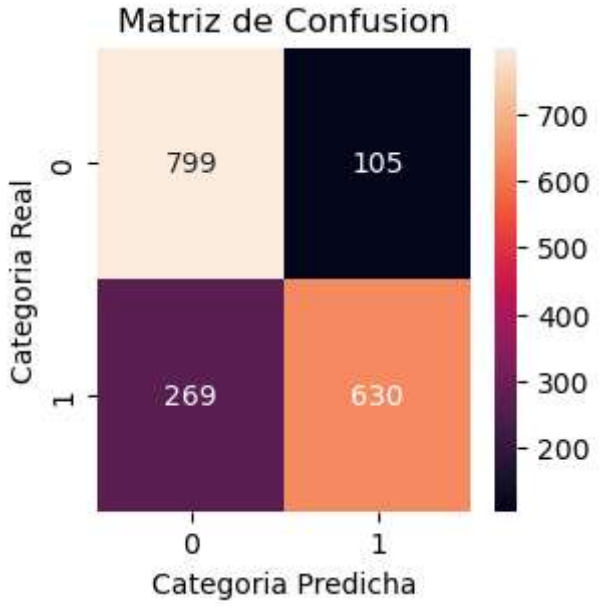
In [28]:
```
#Modelo de Regresion Logistica SMOTE (solo importantes)

#Quitar columnas con menos importancia
```

```
cols_del_imp = ['Nivel_Rol','Salario_Mensual','Distancia_Casa','Edad','Nivel_Educativo']

X_train_sm_imp = X_train_sm.loc[:, [name for name in X_train_sm.columns if name not in cols_del_imp]]
X_test_sm_imp = X_test_sm.loc[:, [name for name in X_test_sm.columns if name not in cols_del_imp]]
```

In [29]:
```
#Entrenar de nuevo
logreg_model_sm_imp = LogisticRegression(C=1.0,penalty='l2',random_state=1,max_iter=10000)
logreg_model_sm_imp.fit(X_train_sm_imp, y_train_sm)
```

Out[29]:
```
LogisticRegression(max_iter=10000, random_state=1)
```

In [30]:
```
#Prediccion sobre el test
pred_y_log_sm_imp = logreg_model_sm_imp.predict(X_test_sm_imp)
mostrar_resultados(y_test_sm, pred_y_log_sm_imp)
#Recall de attrition de 70%, activos en 88%

#Prediccion sobre el train
pred_y_log_sm_train_imp = logreg_model_sm_imp.predict(X_train_sm_imp)
```



```
              precision    recall  f1-score   support

           0       0.75      0.88      0.81       904
           1       0.86      0.70      0.77       899

    accuracy                           0.79      1803
   macro avg       0.80      0.79      0.79      1803
weighted avg       0.80      0.79      0.79      1803
```

In [31]:
```
metricas_comparacion(y_test_sm, pred_y_log_sm_imp, y_train_sm, pred_y_log_sm_train_imp)
```

```
Accuracy:  0.7925679423183583
AUC test:  0.79231410022936
AUC train:  0.7875896885252047
Delta AUC (overfitting):  0.0047244117041552736
```

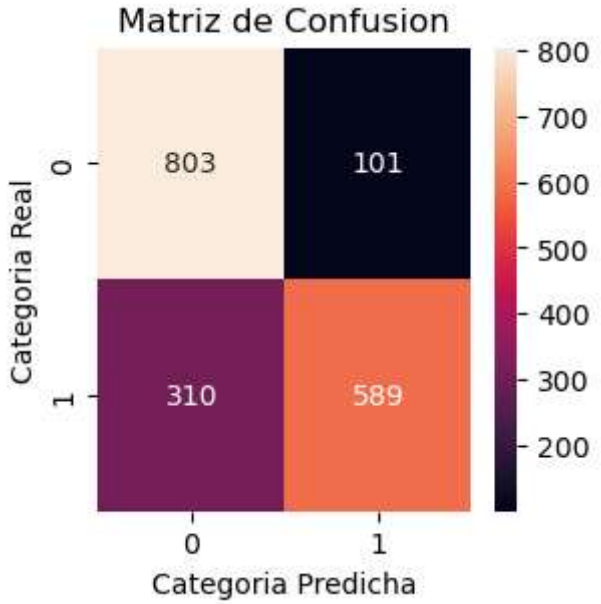
In [39]:
```
#Importancia de las variables
#coefficients_log_sm_imp = logreg_model_sm_imp.coef_[0]

#ft_importance_log_sm_imp = pd.DataFrame({'Feature': X_train_sm_imp.columns, 'Importance': coefficients_log_sm_imp})
#ft_importance_log_sm_imp = ft_importance_log_sm_imp.sort_values('Importance', ascending=True)
#ft_importance_log_sm_imp.plot(x='Feature', y='Importance', kind='barh', figsize=(10, 10))
```

Entrenamiento SMOTE Decision Tree

In [32]:
```
#Arbol de Decision individual SMOTE
dectree_model_sm = DecisionTreeClassifier(max_depth= 5, random_state = 1)
dectree_model_sm.fit(X_train_sm,y_train_sm)
```

Out[32]:
```
DecisionTreeClassifier(max_depth=5, random_state=1)
```

In [33]:
```
#Prediccion sobre el test
pred_y_dectree_sm = dectree_model_sm.predict(X_test_sm)
mostrar_resultados(y_test_sm, pred_y_dectree_sm)
#Recall de attrition de 66%, activos en 89%

#Prediccion sobre el train
pred_y_dectree_sm_train = dectree_model_sm.predict(X_train_sm)
```



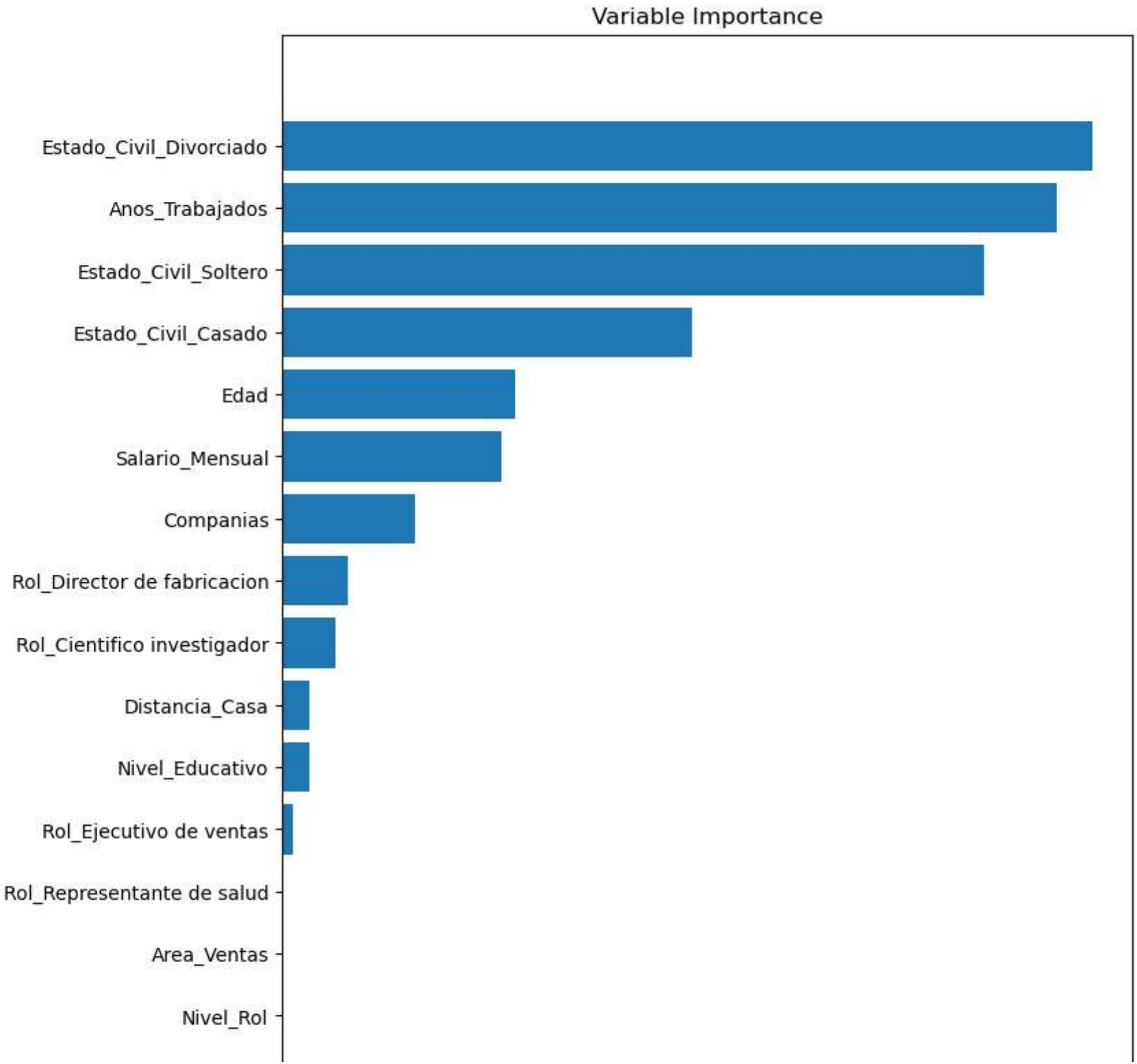
	precision	recall	f1-score	support
0	0.72	0.89	0.80	904
1	0.85	0.66	0.74	899
accuracy			0.77	1803
macro avg	0.79	0.77	0.77	1803
weighted avg	0.79	0.77	0.77	1803

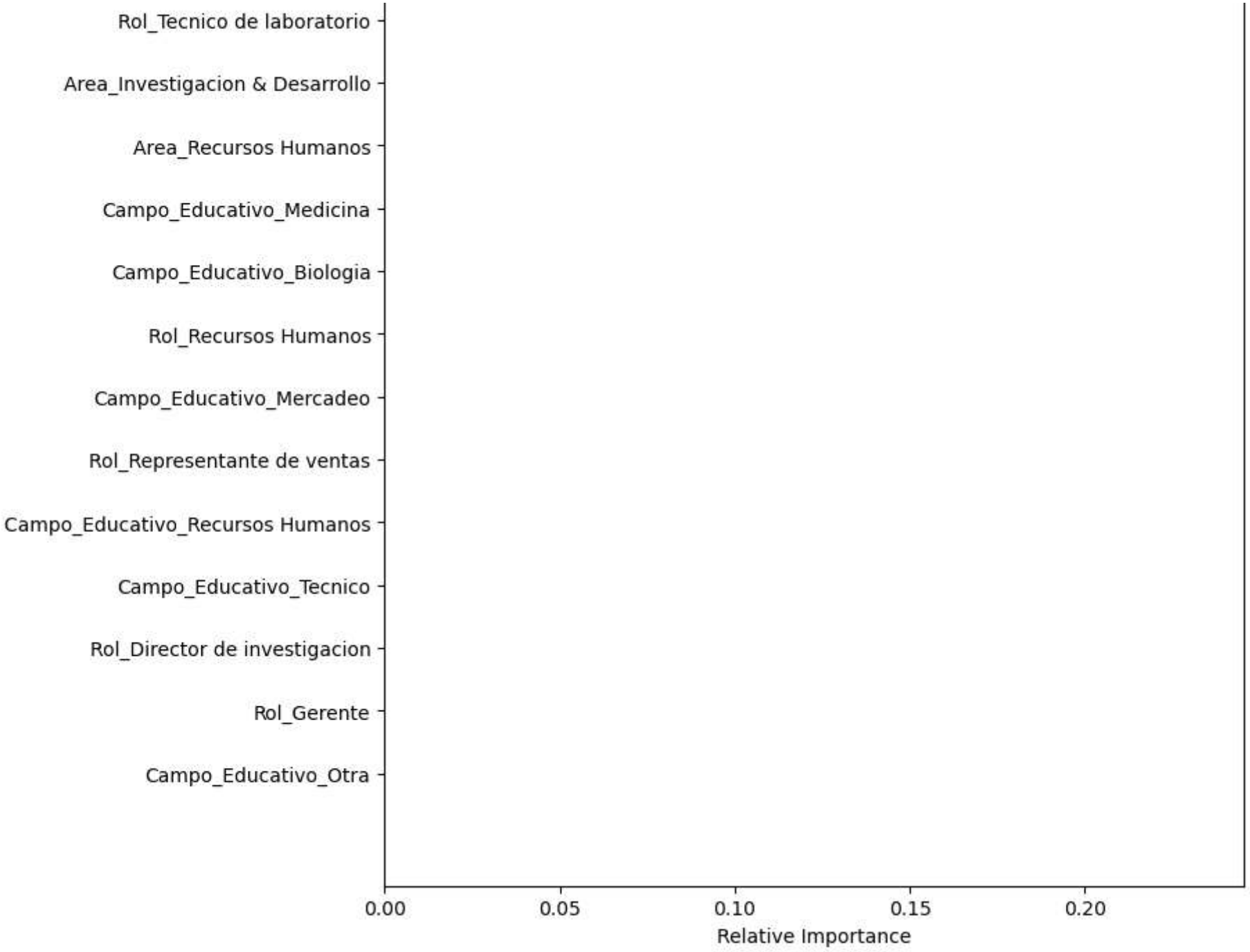
In [34]:
```
metricas_comparacion(y_test_sm, pred_y_dectree_sm, y_train_sm, pred_y_dectree_sm_train)
```

```
Accuracy:  0.7720465890183028
AUC test:  0.7717233750381447
AUC train:  0.7626369117348067
Delta AUC (overfitting):  0.009086463303337955
```

In [35]:
```
feature_importance = dectree_model_sm.feature_importances_
#Importancia relativa a la importancia maxima
feature_importance = 100.0 * (feature_importance / 100)
                             #feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5

plt.figure(figsize=(8, 18))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train_sm.keys()[sorted_idx])
plt.xlabel('Relative Importance')
```

```
plt.title('Variable Importance')
plt.show()
```


Variable Importance
Estado_Civil_Divorciado
Anos_Trabajados
Estado_Civil_Soltero
Estado_Civil_Casado
Edad
Salario_Mensual
Companias
Rol_Director de fabricacion
Rol_Cientifico investigador
Distancia_Casa
Nivel_Educativo
Rol_Ejecutivo de ventas
Rol_Representante de salud
Area_Ventas
Nivel_Rol



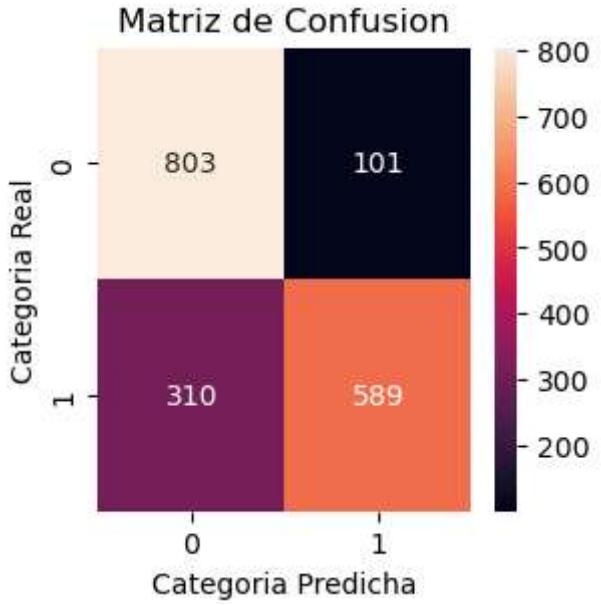
In [36]:
```
#Modelo de Decision Tree SMOTE (solo importantes)

#Quitar columnas con menos importancia
```

```
cols_del_imp = ['Rol_Representante de salud','Area_Ventas','Nivel_Rol','Rol_Tecnico de laboratorio',
                'Area_Investigacion & Desarrollo','Area_Recursos Humanos','Campo_Educativo_Medicina',
                'Campo_Educativo_Biologia','Rol_Recursos Humanos','Campo_Educativo_Mercadeo','Rol_Representante de ventas',
                'Campo_Educativo_Recursos Humanos','Campo_Educativo_Tecnico','Rol_Director de investigacion',
                'Rol_Gerente','Campo_Educativo_Otra']

X_train_sm_imp = X_train_sm.loc[:, [name for name in X_train_sm.columns if name not in cols_del_imp]]
X_test_sm_imp = X_test_sm.loc[:, [name for name in X_test_sm.columns if name not in cols_del_imp]]
```

In [37]:
```
#Entrenar de nuevo Arbol de Decision individual SMOTE
dectree_model_sm_imp = DecisionTreeClassifier(max_depth= 5, random_state = 1)
dectree_model_sm_imp.fit(X_train_sm_imp,y_train_sm)
```

Out[37]:
```
DecisionTreeClassifier(max_depth=5, random_state=1)
```

In [38]:
```
#Prediccion sobre el test
pred_y_dectree_sm_imp = dectree_model_sm_imp.predict(X_test_sm_imp)
mostrar_resultados(y_test_sm, pred_y_dectree_sm_imp)
#Recall de attrition de 66%, activos en 89%

#Prediccion sobre el train
pred_y_dectree_sm_train_imp = dectree_model_sm_imp.predict(X_train_sm_imp)
```



	precision	recall	f1-score	support
0	0.72	0.89	0.80	904
1	0.85	0.66	0.74	899
accuracy			0.77	1803
macro avg	0.79	0.77	0.77	1803
weighted avg	0.79	0.77	0.77	1803

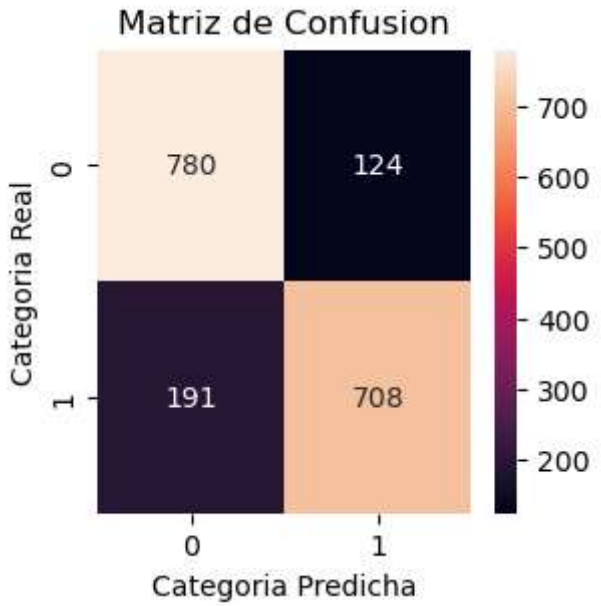
In [39]:
```
metricas_comparacion(y_test_sm, pred_y_dectree_sm_imp, y_train_sm, pred_y_dectree_sm_train_imp)
```

```
Accuracy:  0.7720465890183028
AUC test:  0.7717233750381447
AUC train:  0.7626369117348067
Delta AUC (overfitting):  0.009086463303337955
```

Entrenamiento SMOTE Random Forest

In [40]:
```
#Random Forest con SMOTE
rf_model_sm =RandomForestClassifier(max_depth=5,random_state=1, class_weight = 'balanced')
rf_model_sm.fit(X_train_sm,y_train_sm)

#Prediccion sobre test
y_pred_rf_sm = rf_model_sm.predict(X_test_sm)
```

In [41]:
```
mostrar_resultados(y_test_sm, y_pred_rf_sm)
#Attrition 79%, Activos 86%
```



	precision	recall	f1-score	support
0	0.80	0.86	0.83	904
1	0.85	0.79	0.82	899
accuracy			0.83	1803
macro avg	0.83	0.83	0.83	1803
weighted avg	0.83	0.83	0.83	1803

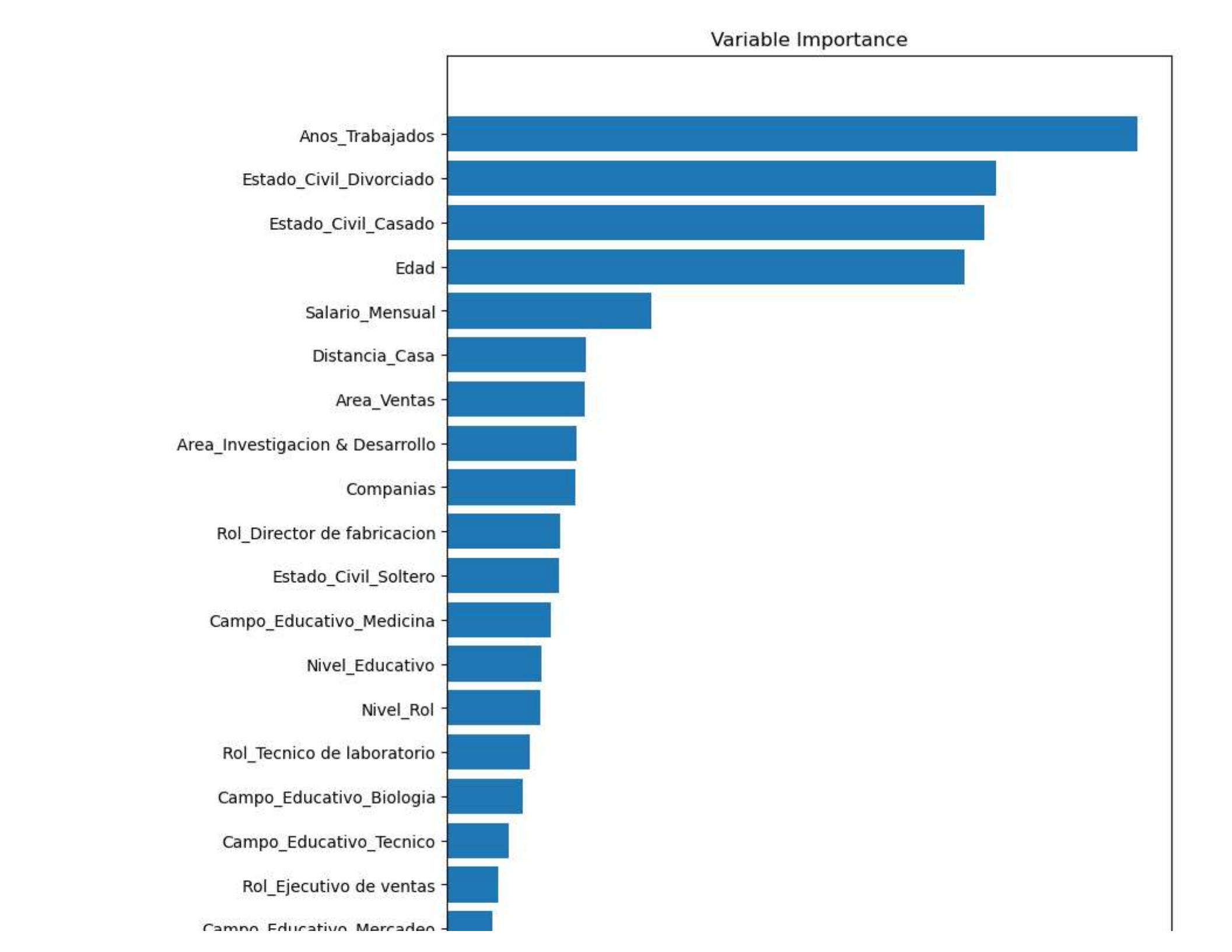
In [42]:
```
#Prediccion sobre el train
y_pred_rf_sm_train = rf_model_sm.predict(X_train_sm)
```

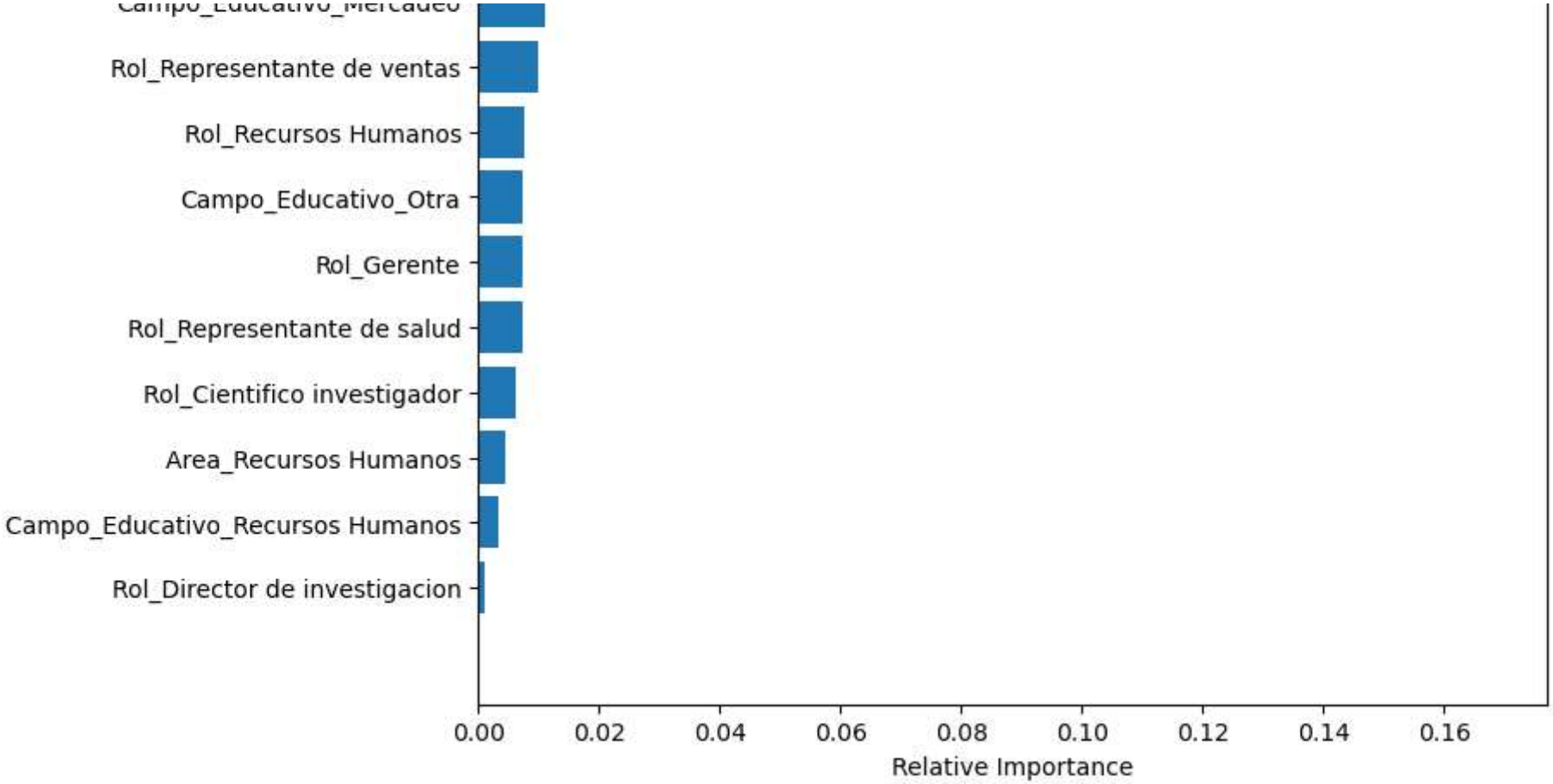
In [43]:
```
metricas_comparacion(y_test_sm, y_pred_rf_sm, y_train_sm, y_pred_rf_sm_train)
```

```
Accuracy:  0.8252911813643927
AUC test:  0.82518678571077
AUC train:  0.8173086368876545
Delta AUC (overfitting):  0.007878148823115438
```

In [44]:
```
#Importancia variables
feature_importance = rf_model_sm.feature_importances_
#Importancia relativa a la importancia maxima
feature_importance = 100.0 * (feature_importance / 100)
                              #feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
```

```
plt.figure(figsize=(8, 15))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train_ada.keys()[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```


Variable Importance
Anos_Trabajados
Estado_Civil_Divorciado
Estado_Civil_Casado
Edad
Salario_Mensual
Distancia_Casa
Area_Ventas
Area_Investigacion & Desarrollo
Companias
Rol_Director de fabricacion
Estado_Civil_Soltero
Campo_Educativo_Medicina
Nivel_Educativo
Nivel_Rol
Rol_Tecnico de laboratorio
Campo_Educativo_Biologia
Campo_Educativo_Tecnico
Rol_Ejecutivo de ventas

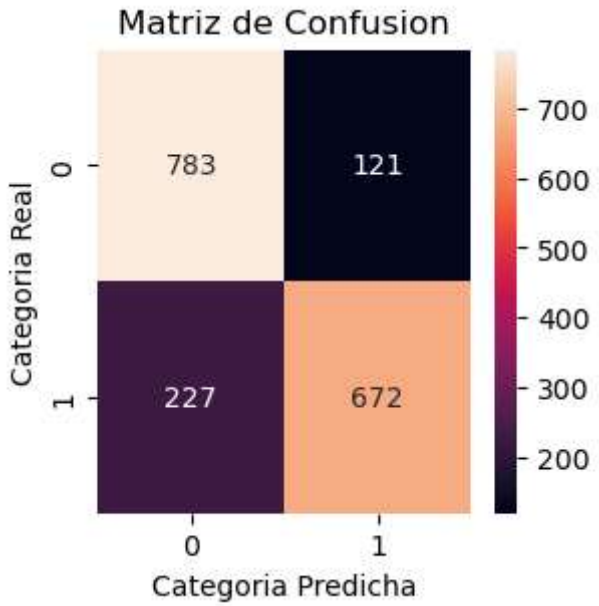


In [45]:
```
#Modelo de Random Forecast SMOTE (solo importantes)
# Segun la grafica las varaibles de rol, campo educativo, compromiso trabajo, nivel de acciones,
# rendimiento no son importantes, se deja frecuencia de viaje frecuente porque las demas estan arriba
# se quitan todos los roles y areas para ser consistentes

#Quitar columnas con menos importancia
cols_del_imp = ['Rol_Ejecutivo de ventas','Campo_Educativo_Mercadeo','Rol_Representante de ventas','Rol_Recursos_Humanos',
                'Campo_Educativo_Otra','Rol_Gerente','Rol_Representante de salud','Rol_Cientifico investigador',
                'Area_Recursos Humanos','Campo_Educativo_Recursos Humanos','Rol_Director de investigacion']

X_train_sm_imp = X_train_sm.loc[:, [name for name in X_train_sm.columns if name not in cols_del_imp]]
X_test_sm_imp = X_test_sm.loc[:, [name for name in X_test_sm.columns if name not in cols_del_imp]]
```

In [46]:
```
#Entrenar modelo de nuevo
rf_model_sm_imp =RandomForestClassifier(max_depth=5,random_state=1, class_weight = 'balanced')
rf_model_sm_imp.fit(X_train_sm_imp,y_train_sm)
```

```
#Prediccion sobre test
y_pred_rf_sm_imp = rf_model_sm_imp.predict(X_test_sm_imp)
```

In [47]:
```
mostrar_resultados(y_test_sm, y_pred_rf_sm_imp)
#Attrition 75%, Activos 87%
```



	precision	recall	f1-score	support
0	0.78	0.87	0.82	904
1	0.85	0.75	0.79	899
accuracy			0.81	1803
macro avg	0.81	0.81	0.81	1803
weighted avg	0.81	0.81	0.81	1803

In [48]:
```
#Prediccion sobre el train
y_pred_rf_sm_train_imp = rf_model_sm_imp.predict(X_train_sm_imp)
```

In [49]:
```
metricas_comparacion(y_test_sm, y_pred_rf_sm_imp, y_train_sm, y_pred_rf_sm_train_imp)
```

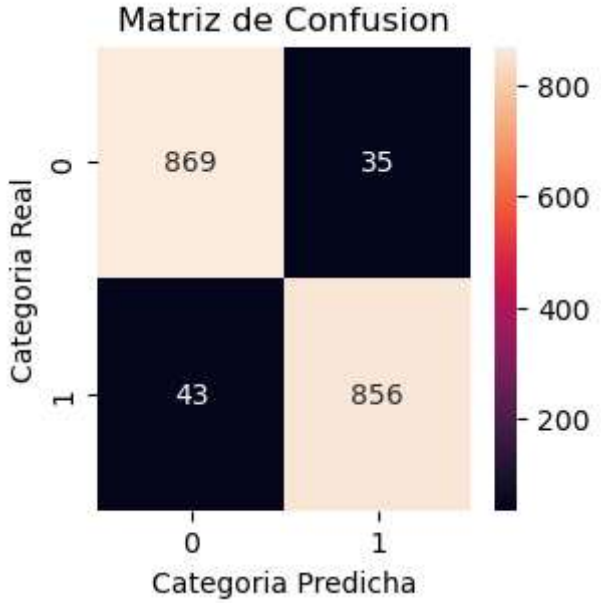
```
Accuracy:  0.8069883527454242
AUC test:  0.8068238308051227
AUC train:  0.8071545864729961
Delta AUC (overfitting):  0.00033075566787343913
```

Entrenamiento SMOTE Gradient Boosting

In [50]:
```
#Gradient Boosting individual
gbm_model_sm = GradientBoostingClassifier(max_depth = 5,max_features = 'auto',random_state = 1)
gbm_model_sm.fit(X_train_sm,y_train_sm)
```

Out[50]:
```
GradientBoostingClassifier(max_depth=5, max_features='auto', random_state=1)
```

In [51]:
```
#Prediccion sobre el test
pred_y_gbm_sm = gbm_model_sm.predict(X_test_sm)
mostrar_resultados(y_test_sm, pred_y_gbm_sm)
#Recall de attrition de 95%, activos en 96%

#Prediccion sobre el train
pred_y_gbm_sm_train = gbm_model_sm.predict(X_train_sm)
```

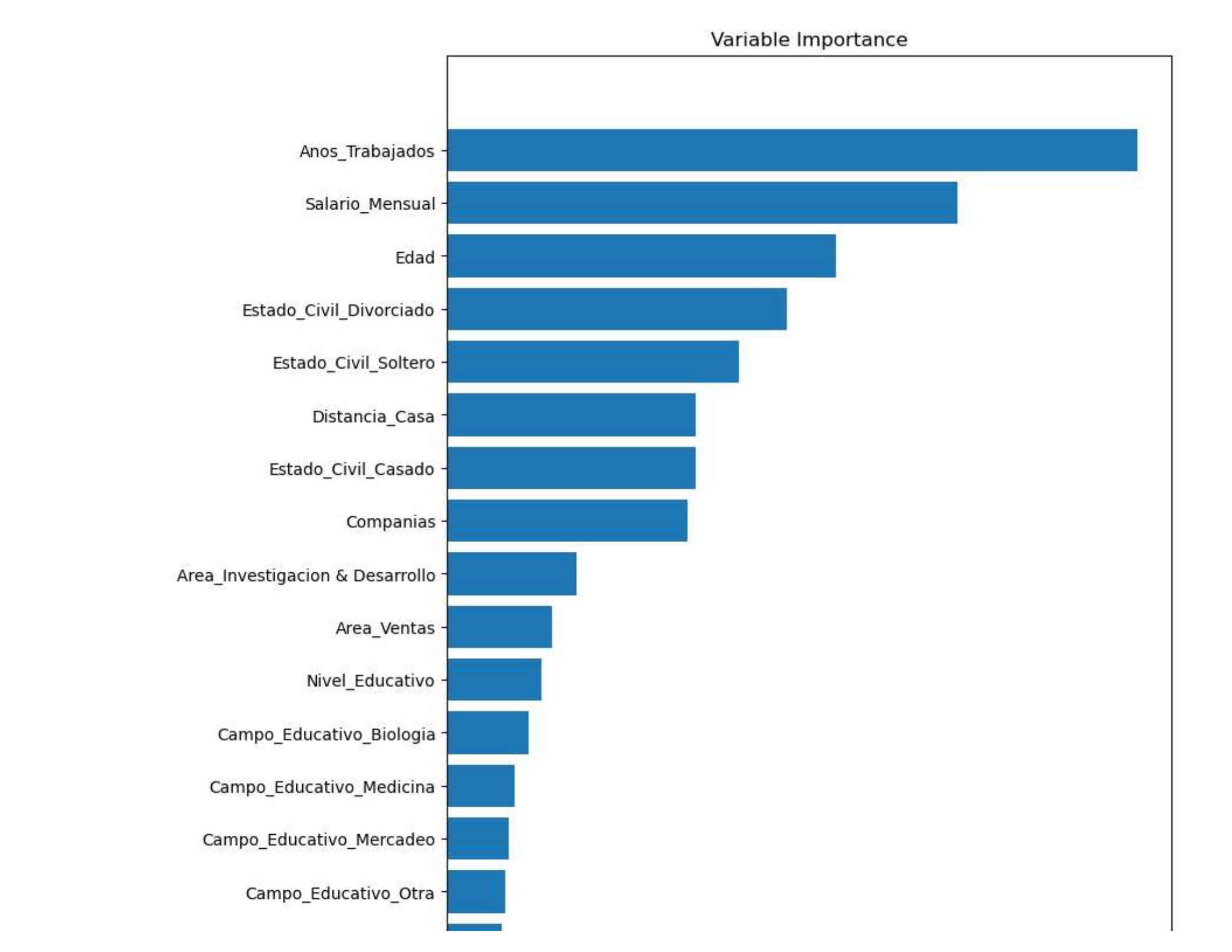


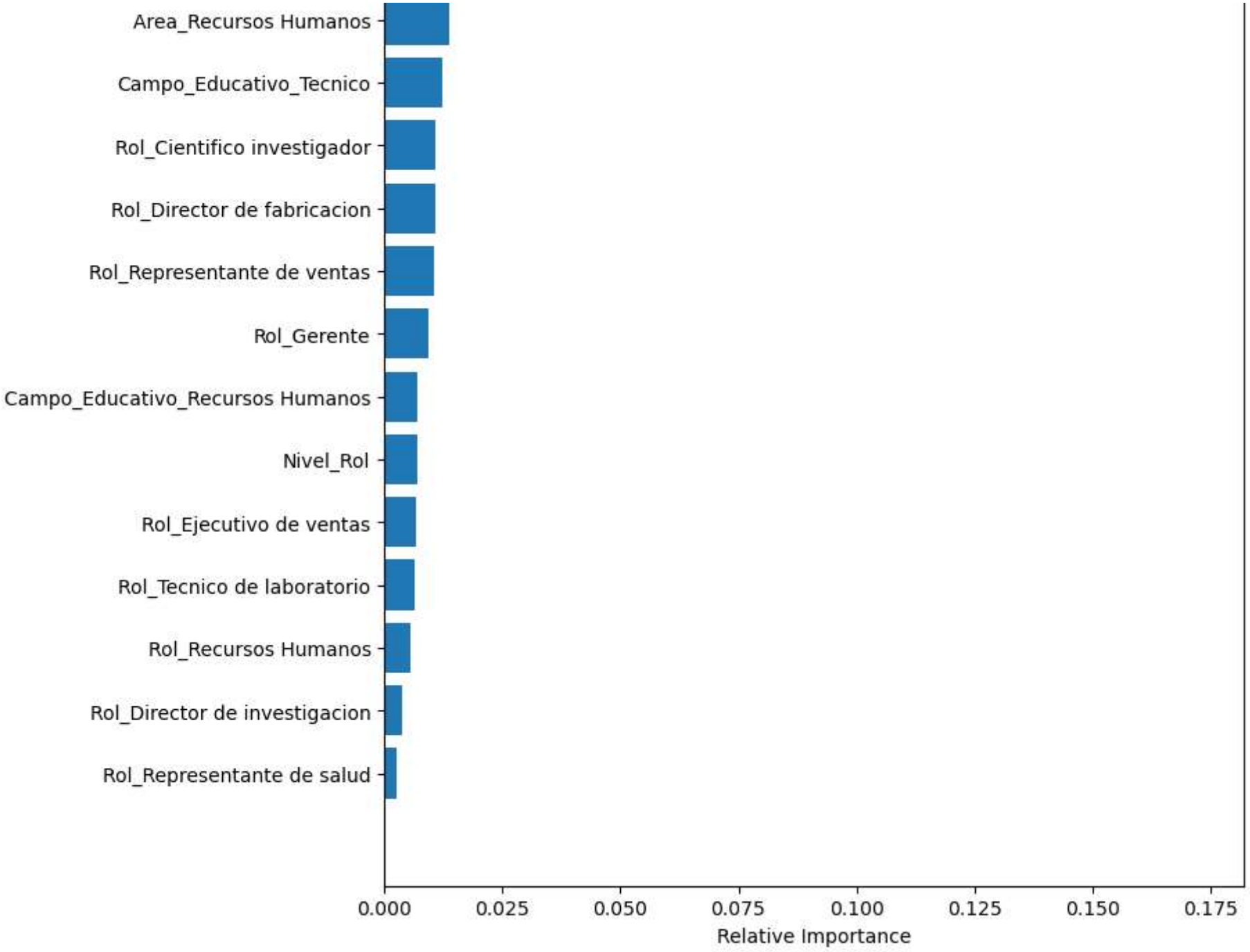
	precision	recall	f1-score	support
0	0.95	0.96	0.96	904
1	0.96	0.95	0.96	899
accuracy			0.96	1803
macro avg	0.96	0.96	0.96	1803
weighted avg	0.96	0.96	0.96	1803

In [52]:
```
metricas_comparacion(y_test_sm, pred_y_gbm_sm, y_train_sm, pred_y_gbm_sm_train)
```

```
Accuracy:  0.956738768718802
AUC test:  0.9567261312963272
AUC train:  0.9763338042656453
Delta AUC (overfitting):  0.01960767296931809
```

In [53]:
```
feature_importance = gbm_model_sm.feature_importances_
#Importancia relativa a la importancia maxima
feature_importance = 100.0 * (feature_importance / 100)
                              #feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5

plt.figure(figsize=(8, 18))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train_sm.keys()[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```


Variable Importance
Anos_Trabajados
Salario_Mensual
Edad
Estado_Civil_Divorciado
Estado_Civil_Soltero
Distancia_Casa
Estado_Civil_Casado
Companias
Area_Investigacion & Desarrollo
Area_Ventas
Nivel_Educativo
Campo_Educativo_Biologia
Campo_Educativo_Medicina
Campo_Educativo_Mercadeo
Campo_Educativo_Otra



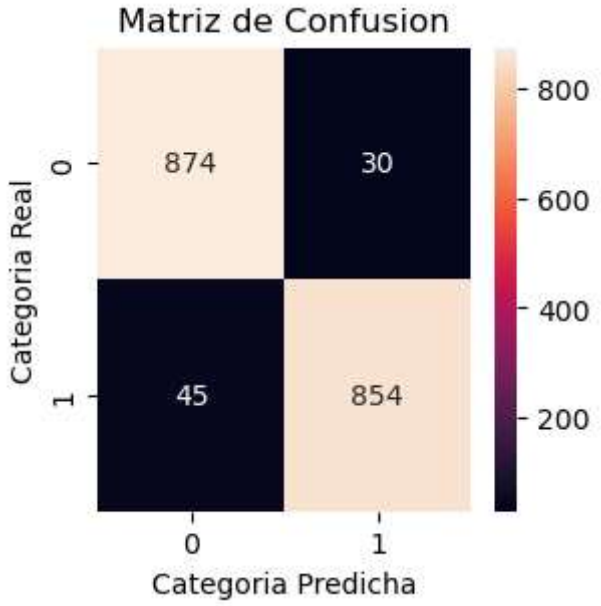
In [54]:
```
#Modelo de Gradient Boosting SMOTE (solo importantes)

#Quitar columnas con menos importancia
```

```
cols_del_imp = ['Area_Recursos_Humanos','Campo_Educativo_Tecnico','Rol_Cientifico investigador',
                'Rol_Director de fabricacion','Rol_Representante de ventas','Rol_Gerente',
                'Campo_Educativo_Recursos Humanos','Nivel_Rol','Rol_Ejecutivo de ventas','Rol_Tecnico de laboratorio',
                'Rol_Recursos Humanos','Rol_Director de investigacion','Rol_Representante de salud']

X_train_sm_imp = X_train_sm.loc[:, [name for name in X_train_sm.columns if name not in cols_del_imp]]
X_test_sm_imp = X_test_sm.loc[:, [name for name in X_test_sm.columns if name not in cols_del_imp]]
```

In [55]:
```
#Entrenar de nuevo
gbm_model_sm_imp = GradientBoostingClassifier(max_depth = 5,max_features = 'auto',random_state = 1)
gbm_model_sm_imp.fit(X_train_sm_imp,y_train_sm)
```

Out[55]:
```
GradientBoostingClassifier(max_depth=5, max_features='auto', random_state=1)
```

In [56]:
```
#Prediccion sobre el test
pred_y_gbm_sm_imp = gbm_model_sm_imp.predict(X_test_sm_imp)
mostrar_resultados(y_test_sm, pred_y_gbm_sm_imp)
#Recall de attrition de 95%, activos en 97%

#Prediccion sobre el train
pred_y_gbm_sm_train_imp = gbm_model_sm_imp.predict(X_train_sm_imp)
```



```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       904
           1       0.97      0.95      0.96       899

    accuracy                           0.96      1803
   macro avg       0.96      0.96      0.96      1803
weighted avg       0.96      0.96      0.96      1803
```

In [57]:
```
metricas_comparacion(y_test_sm, pred_y_gbm_sm_imp, y_train_sm, pred_y_gbm_sm_train_imp)
```

```
Accuracy:  0.9584026622296173
AUC test:  0.9583792709697108
AUC train:  0.9829880833054905
Delta AUC (overfitting):  0.024608812335779717
```

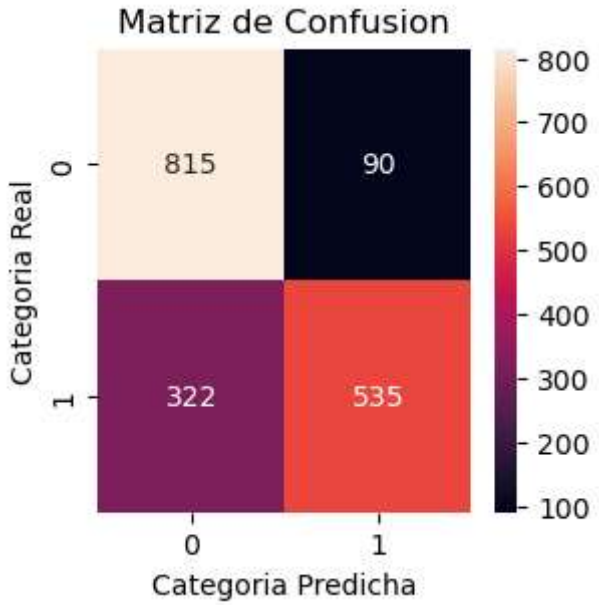
In [62]:
```
#feature_importance = gbm_model_sm_imp.feature_importances_
##Importancia relativa a la importancia maxima
#feature_importance = 100.0 * (feature_importance / 100)
#                             #feature_importance.max())
#sorted_idx = np.argsort(feature_importance)
#pos = np.arange(sorted_idx.shape[0]) + .5
#
#plt.figure(figsize=(8, 18))
#plt.barh(pos, feature_importance[sorted_idx], align='center')
#plt.yticks(pos, X_train_sm_imp.keys()[sorted_idx])
#plt.xlabel('Relative Importance')
#plt.title('Variable Importance')
#plt.show()
```

Entrenamiento ADASYN Logistic Regression

In [59]:
```
#Modelo de Regresion Logistica ADASYN
logreg_model_ada = LogisticRegression(C=1.0,penalty='l2',random_state=1,max_iter=10000)
logreg_model_ada.fit(X_train_ada, y_train_ada)
```

Out[59]:
```
LogisticRegression(max_iter=10000, random_state=1)
```

In [60]:
```
#Prediccion sobre el test
pred_y_log_ada = logreg_model_ada.predict(X_test_ada)
mostrar_resultados(y_test_ada, pred_y_log_ada)
#Recall de attrition de 62%, activos en 90%

#Prediccion sobre el train
pred_y_log_ada_train = logreg_model_ada.predict(X_train_ada)
```



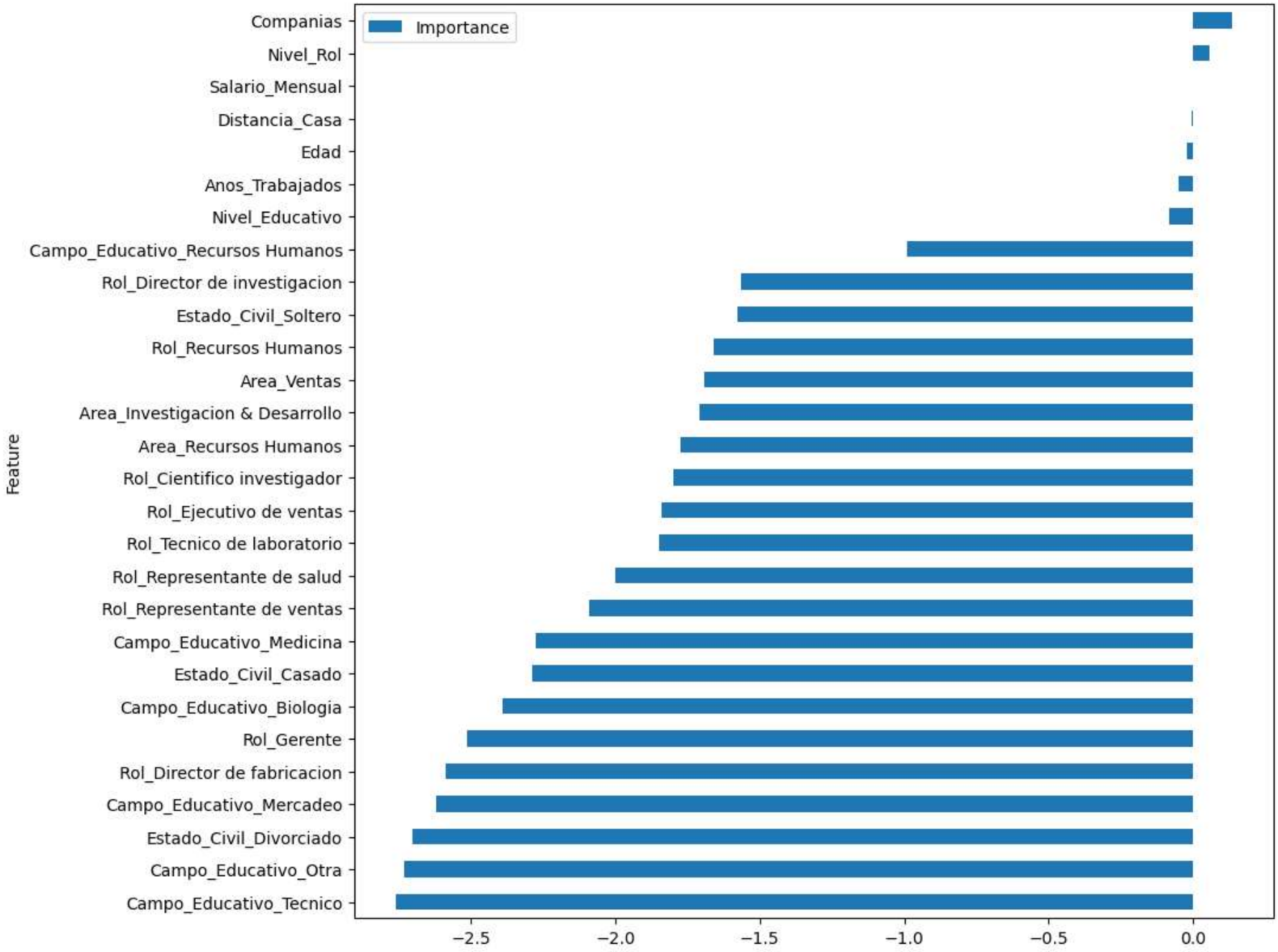
	precision	recall	f1-score	support
0	0.72	0.90	0.80	905
1	0.86	0.62	0.72	857
accuracy			0.77	1762
macro avg	0.79	0.76	0.76	1762
weighted avg	0.78	0.77	0.76	1762

In [61]:
```
metricas_comparacion(y_test_ada, pred_y_log_ada, y_train_ada, pred_y_log_ada_train)
```

```
Accuracy:  0.7661748013620885
AUC test:  0.7624115989865714
AUC train:  0.7754723190089046
Delta AUC (overfitting):  0.013060720022333161
```

In [62]:
```
#Importancia de las variables
coefficients_log_ada = logreg_model_ada.coef_[0]

ft_importance_log_ada = pd.DataFrame({'Feature': X_ada.columns, 'Importance': coefficients_log_ada})
ft_importance_log_ada = ft_importance_log_ada.sort_values('Importance', ascending=True)
ft_importance_log_ada.plot(x='Feature', y='Importance', kind='barh', figsize=(10, 10))
```

Out[62]:
```
<AxesSubplot:ylabel='Feature'>
```



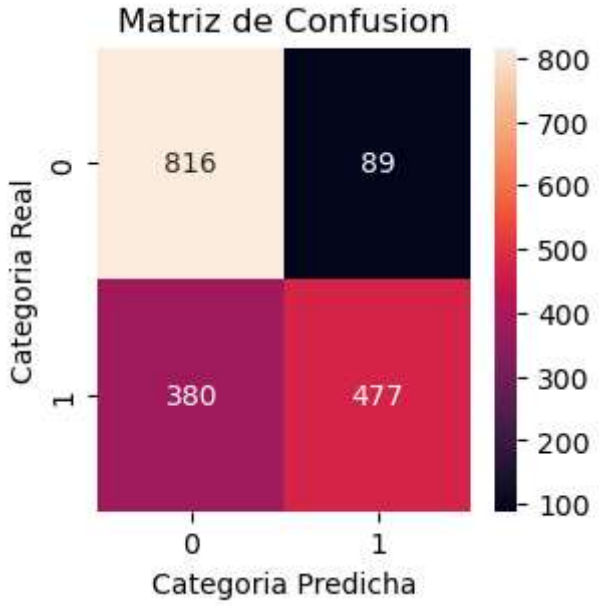
In [63]:
```
#Modelo de Regresion Logistica ADASYN (solo importantes)

#Quitar columnas con menos importancia
```

```
cols_del_imp =  ['Nivel_Rol','Salario_Mensual','Distancia_Casa','Edad','Anos_Trabajados']

X_train_ada_imp = X_train_ada.loc[:, [name for name in X_train_ada.columns if name not in cols_del_imp]]
X_test_ada_imp = X_test_ada.loc[:, [name for name in X_test_ada.columns if name not in cols_del_imp]]
```

In [64]:
```
#Entrenar de nuevo
logreg_model_ada_imp = LogisticRegression(C=1.0,penalty='l2',random_state=1,max_iter=10000)
logreg_model_ada_imp.fit(X_train_ada_imp, y_train_ada)
```

Out[64]:
```
LogisticRegression(max_iter=10000, random_state=1)
```

In [65]:
```
#Prediccion sobre el test
pred_y_log_ada_imp = logreg_model_ada_imp.predict(X_test_ada_imp)
mostrar_resultados(y_test_ada, pred_y_log_ada_imp)
#Recall de attrition de 56%, activos en 90%

#Prediccion sobre el train
pred_y_log_ada_train_imp = logreg_model_ada_imp.predict(X_train_ada_imp)
```



```
              precision    recall  f1-score   support

           0       0.68      0.90      0.78       905
           1       0.84      0.56      0.67       857

    accuracy                           0.73      1762
   macro avg       0.76      0.73      0.72      1762
weighted avg       0.76      0.73      0.73      1762
```

In [66]:
```
metricas_comparacion(y_test_ada, pred_y_log_ada_imp, y_train_ada, pred_y_log_ada_train_imp)
```

```
Accuracy:  0.7338251986379115
AUC test:  0.7291251120122231
AUC train:  0.7578500451671183
Delta AUC (overfitting):  0.028724933154895216
```

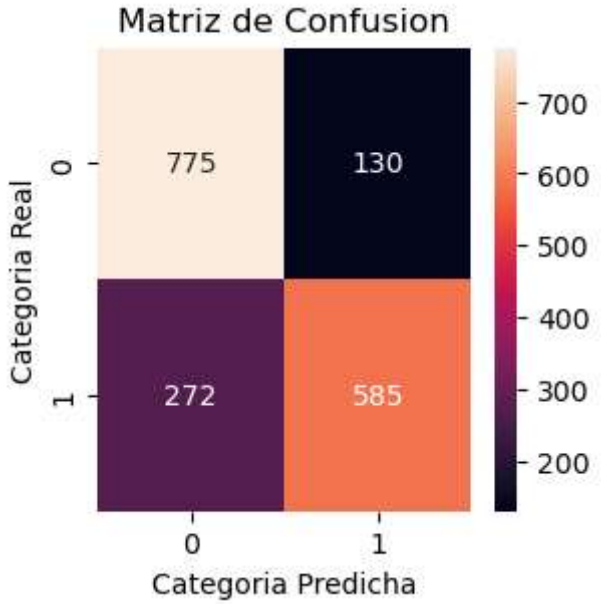
In [39]:
```
#Importancia de las variables
#coefficients_log_ada_imp = logreg_model_ada_imp.coef_[0]

#ft_importance_log_ada_imp = pd.DataFrame({'Feature': X_train_ada_imp.columns, 'Importance': coefficients_log_ada_imp})
#ft_importance_log_ada_imp = ft_importance_log_ada_imp.sort_values('Importance', ascending=True)
#ft_importance_log_ada_imp.plot(x='Feature', y='Importance', kind='barh', figsize=(10, 10))
```

Entrenamiento ADASYN Decision Tree

In [67]:
```
#Arbol de Decision individual ADASYN
dectree_model_ada = DecisionTreeClassifier(max_depth= 5, random_state = 1)
dectree_model_ada.fit(X_train_ada,y_train_ada)
```

Out[67]:
```
DecisionTreeClassifier(max_depth=5, random_state=1)
```

In [68]:
```
#Prediccion sobre el test
pred_y_dectree_ada = dectree_model_ada.predict(X_test_ada)
mostrar_resultados(y_test_ada, pred_y_dectree_ada)
#Recall de attrition de 68%, activos en 86%

#Prediccion sobre el train
pred_y_dectree_ada_train = dectree_model_ada.predict(X_train_ada)
```



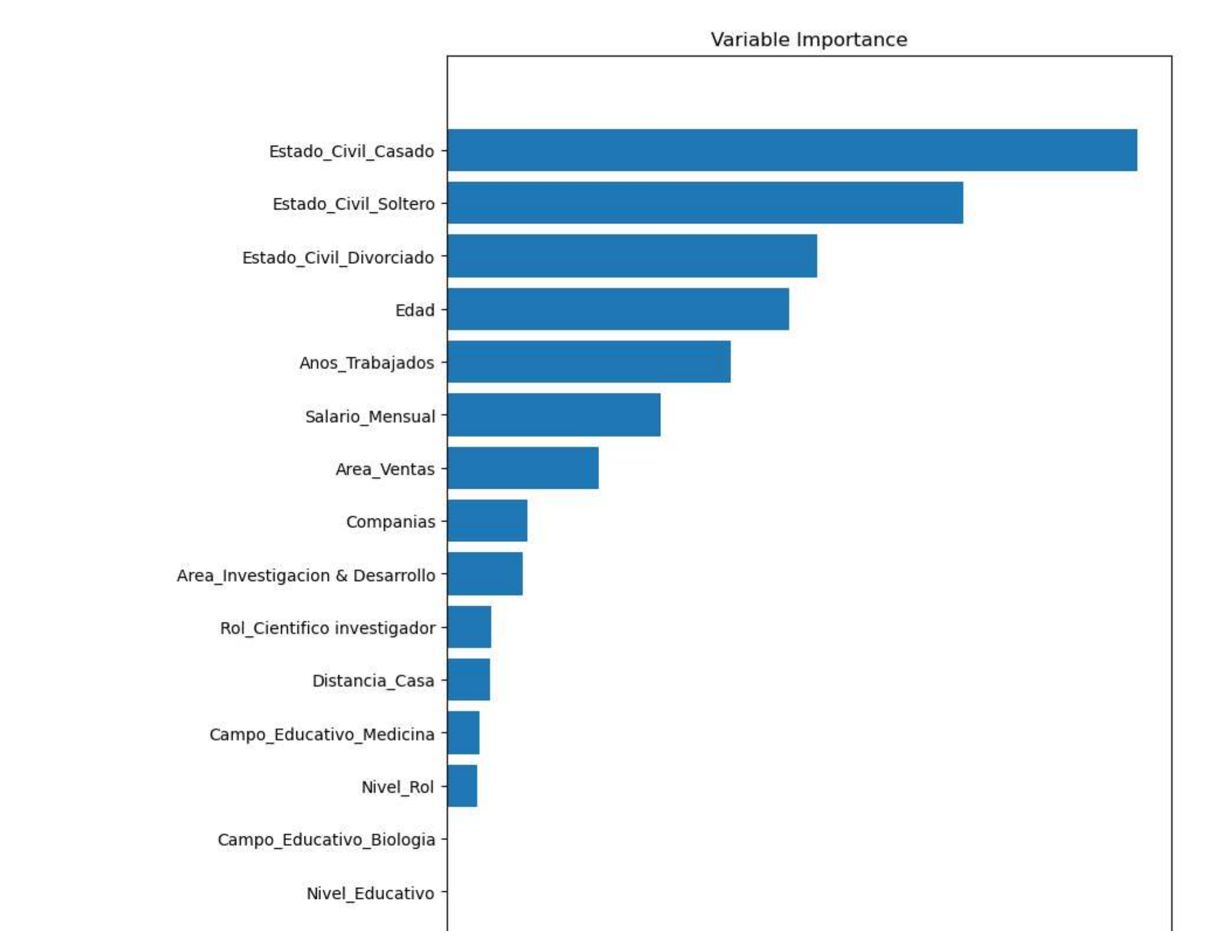
	precision	recall	f1-score	support
0	0.74	0.86	0.79	905
1	0.82	0.68	0.74	857
accuracy			0.77	1762
macro avg	0.78	0.77	0.77	1762
weighted avg	0.78	0.77	0.77	1762

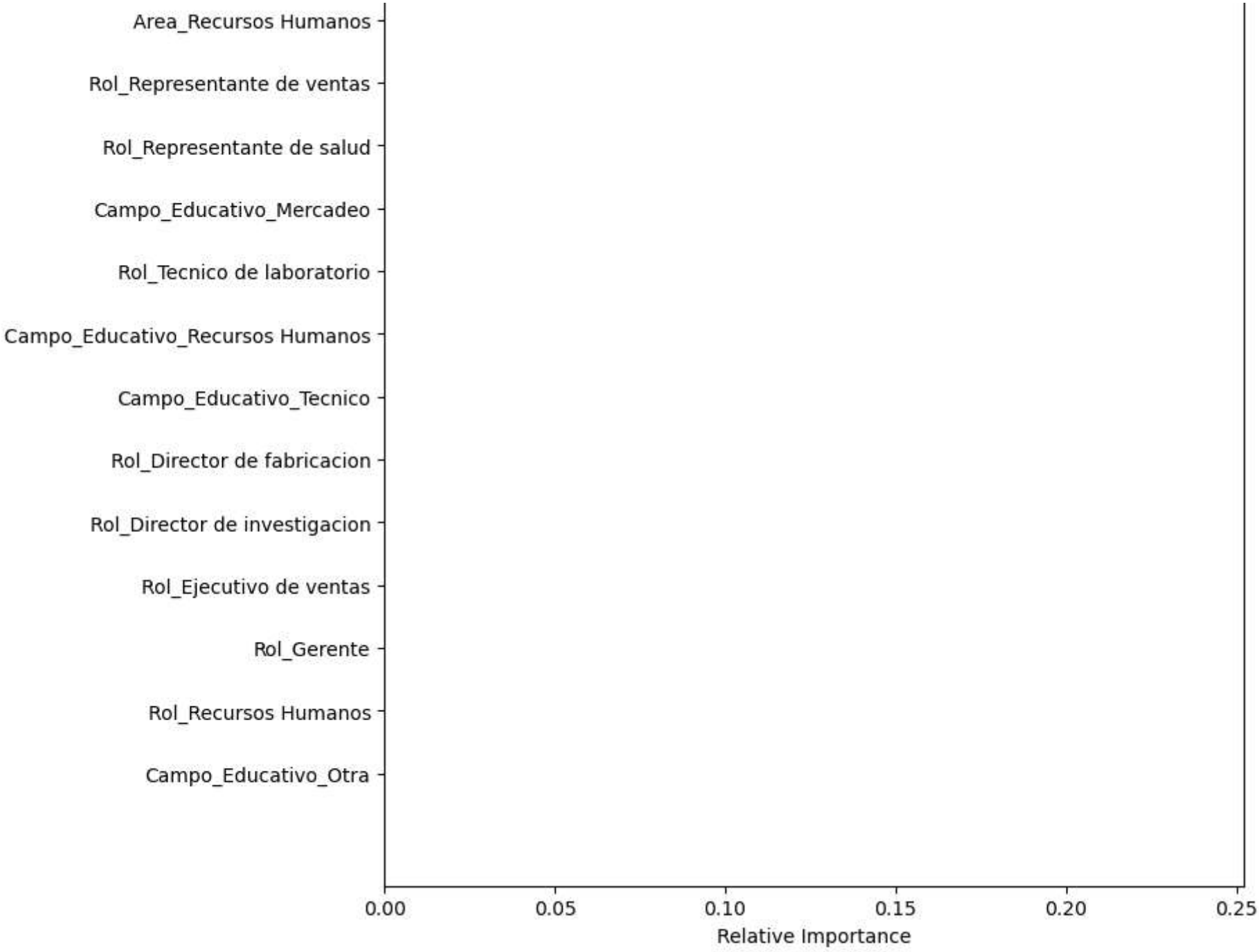
In [69]:
```
metricas_comparacion(y_test_ada, pred_y_dectree_ada, y_train_ada, pred_y_dectree_ada_train)
```

```
Accuracy:  0.771850170261067
AUC test:  0.7694836800608572
AUC train:  0.7671441476319525
Delta AUC (overfitting):  0.0023395324289047226
```

In [70]:
```
#Hallar variables importantes
feature_importance = dectree_model_ada.feature_importances_
#Importancia relativa a la importancia maxima
feature_importance = 100.0 * (feature_importance / 100)
                              #feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5

plt.figure(figsize=(8, 18))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train_sm.keys()[sorted_idx])
```

```
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```


Variable Importance
Estado_Civil_Casado
Estado_Civil_Soltero
Estado_Civil_Divorciado
Edad
Anos_Trabajados
Salario_Mensual
Area_Ventas
Companias
Area_Investigacion & Desarrollo
Rol_Cientifico investigador
Distancia_Casa
Campo_Educativo_Medicina
Nivel_Rol
Campo_Educativo_Biologia
Nivel_Educativo



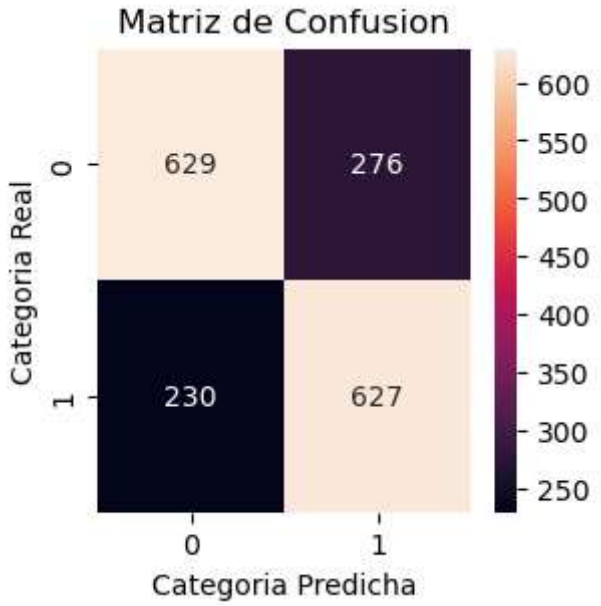
In [71]:
```
#Modelo de Decision Tree ADAYSIN(solo importantes)

#Quitar columnas con menos importancia
```

```
cols_del_imp = ['Campo_Educativo_Biologia','Nivel_Educativo','Area_Recursos Humanos','Rol_Representante de ventas',
                'Rol_Representante de salud','Campo_Educativo_Mercadeo','Rol_Tecnico de laboratorio',
                'Campo_Educativo_Recursos Humanos','Campo_Educativo Tecnico','Rol_Director de fabricacion',
                'Rol_Director de investigacion','Rol_Ejecutivo de ventas','Rol_Gerente','Rol_Recursos Humanos',
                'Campo_Educativo_Otra']

X_train_sm_imp = X_train_sm.loc[:, [name for name in X_train_sm.columns if name not in cols_del_imp]]
X_test_sm_imp = X_test_sm.loc[:, [name for name in X_test_sm.columns if name not in cols_del_imp]]
```

In [72]:
```
#Arbol de Decision individual ADASYN con variables importantes
dectree_model_ada_imp = DecisionTreeClassifier(max_depth= 5, random_state = 1)
dectree_model_ada_imp.fit(X_train_ada_imp,y_train_ada)
```

Out[72]:
```
DecisionTreeClassifier(max_depth=5, random_state=1)
```

In [73]:
```
#Prediccion sobre el test
pred_y_dectree_ada_imp = dectree_model_ada_imp.predict(X_test_ada_imp)
mostrar_resultados(y_test_ada, pred_y_dectree_ada_imp)
#Recall de attrition de 73%, activos en 70%

#Prediccion sobre el train
pred_y_dectree_ada_train_imp = dectree_model_ada_imp.predict(X_train_ada_imp)
```



```
              precision    recall  f1-score   support

           0       0.73      0.70      0.71       905
           1       0.69      0.73      0.71       857

    accuracy                           0.71      1762
   macro avg       0.71      0.71      0.71      1762
weighted avg       0.71      0.71      0.71      1762
```

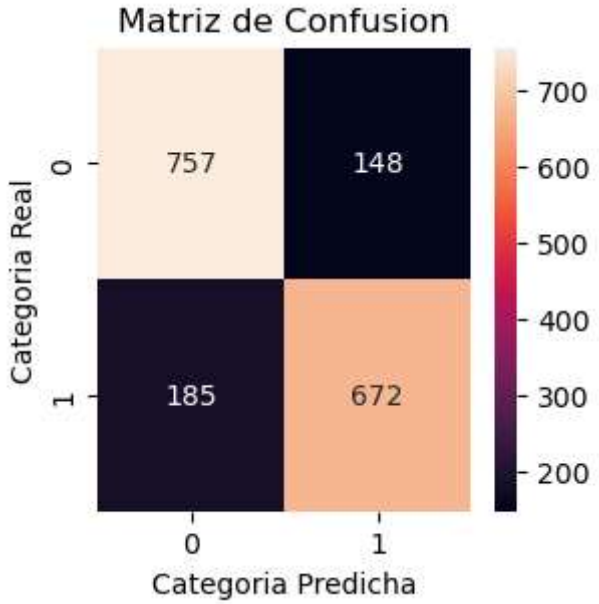
In [74]:
```
metricas_comparacion(y_test_ada, pred_y_dectree_ada_imp, y_train_ada, pred_y_dectree_ada_train_imp)
```

```
Accuracy:  0.7128263337116912
AUC test:  0.7133247806494453
AUC train:  0.7310878823073945
Delta AUC (overfitting):  0.017763101657949254
```

Entrenamiento ADASYN Random Forest

In [75]:
```
#Random Forest con ADASYN
rf_model_ada =RandomForestClassifier(max_depth=5,random_state=1, class_weight = 'balanced')
rf_model_ada.fit(X_train_ada,y_train_ada)

#Prediccion sobre test
y_pred_rf_ada = rf_model_ada.predict(X_test_ada)
```

In [76]:
```
mostrar_resultados(y_test_ada, y_pred_rf_ada)
#Attrition 78%, Activos 84%
```



	precision	recall	f1-score	support
0	0.80	0.84	0.82	905
1	0.82	0.78	0.80	857
accuracy			0.81	1762
macro avg	0.81	0.81	0.81	1762
weighted avg	0.81	0.81	0.81	1762

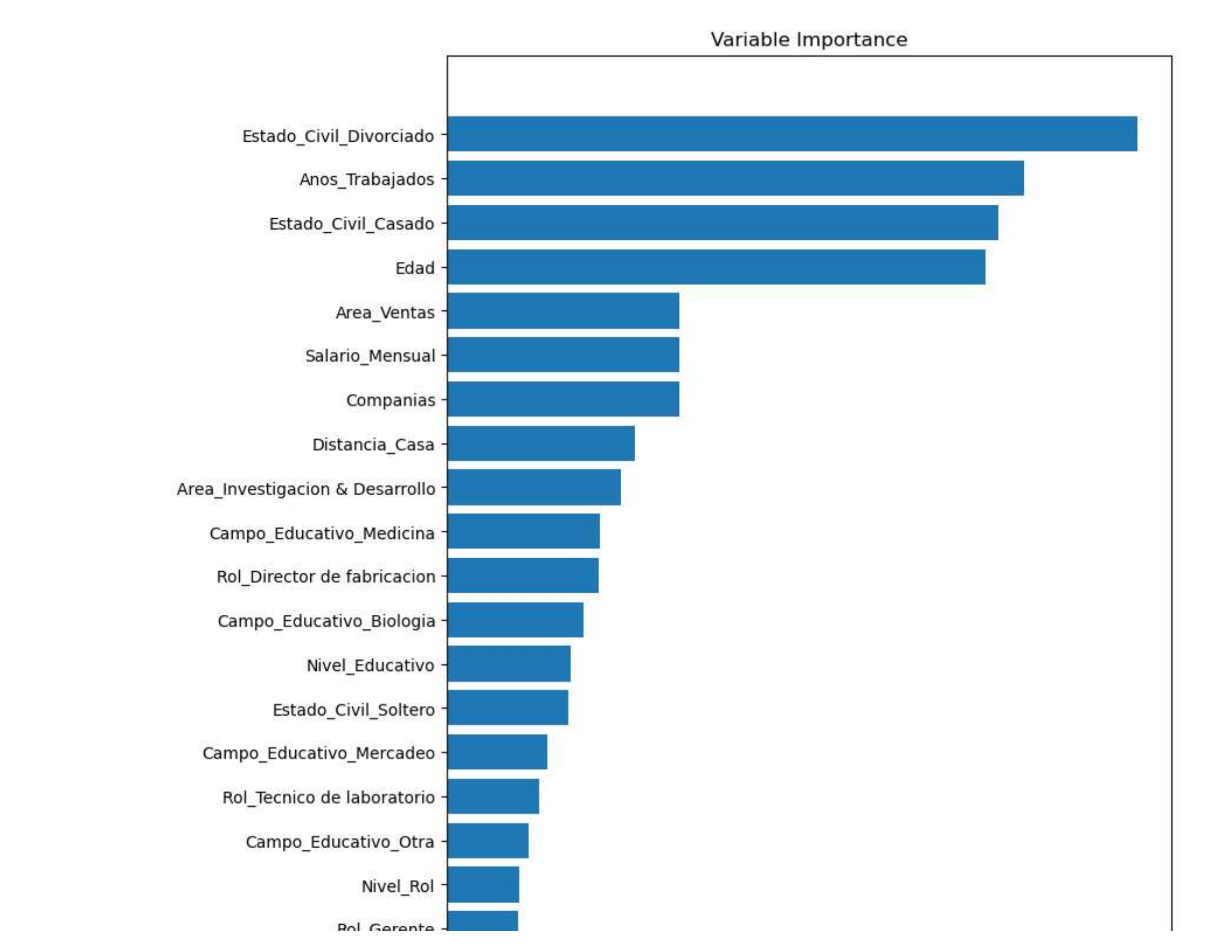
In [77]:
```
#Prediccion sobre el train
y_pred_rf_ada_train = rf_model_ada.predict(X_train_ada)
```

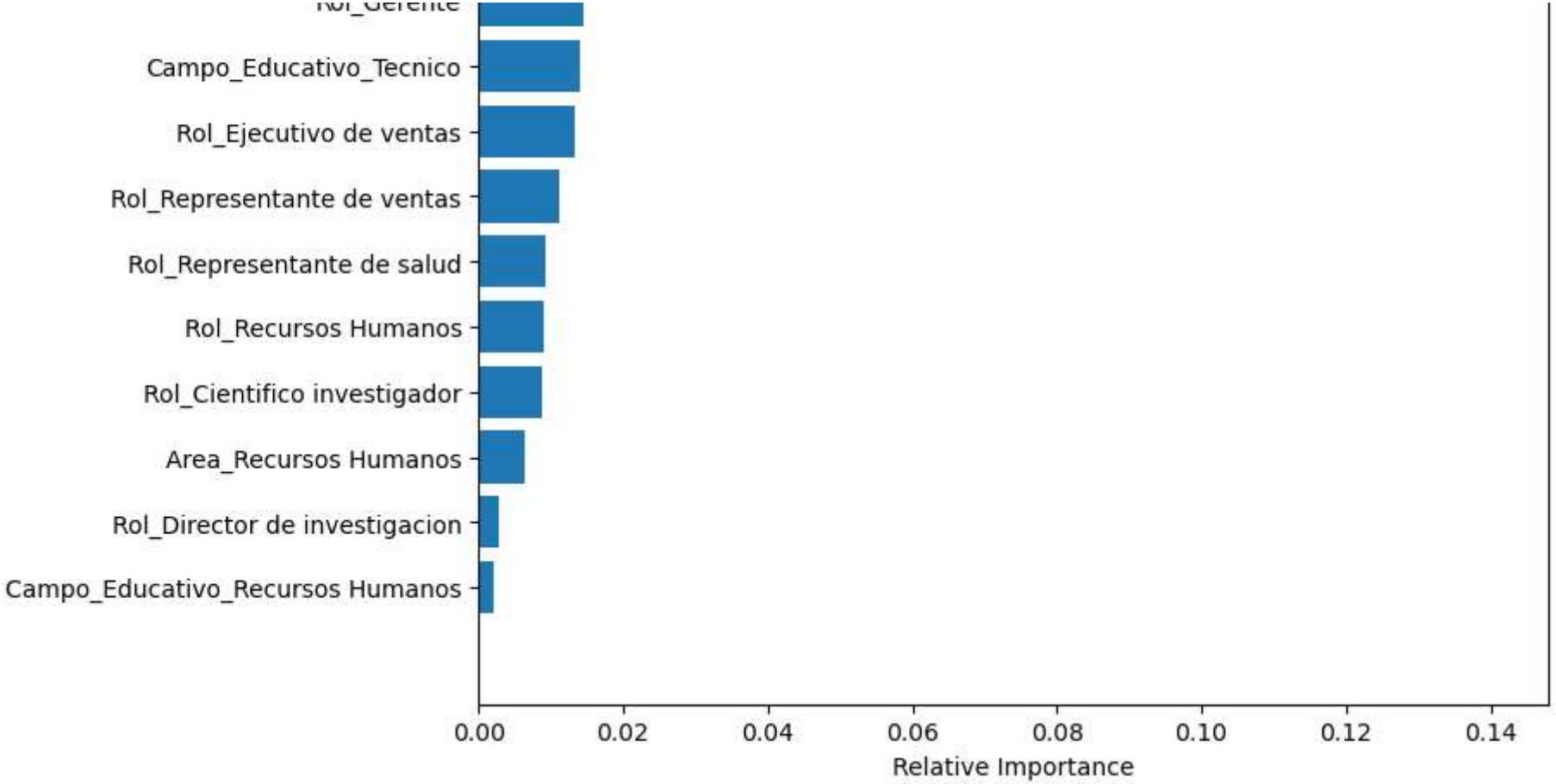
In [78]:
```
metricas_comparacion(y_test_ada, y_pred_rf_ada, y_train_ada, y_pred_rf_ada_train)
```

```
Accuracy:  0.8110102156640182
AUC test:  0.8102973884229324
AUC train:  0.8255194218608853
Delta AUC (overfitting):  0.01522203343795292
```

In [79]:
```
#Importancia variables
feature_importance = rf_model_ada.feature_importances_
#Importancia relativa a la importancia maxima
feature_importance = 100.0 * (feature_importance / 100)
                              #feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
```

```
plt.figure(figsize=(8, 15))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train_ada.keys()[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```

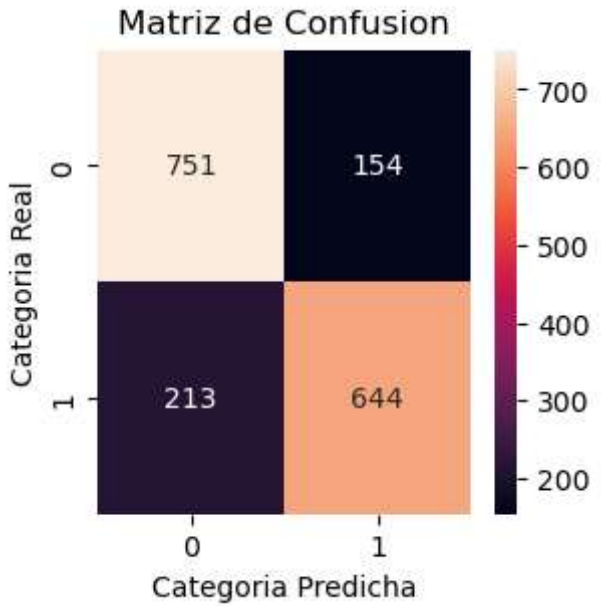

Variable Importance
Estado_Civil_Divorciado
Anos_Trabajados
Estado_Civil_Casado
Edad
Area_Ventas
Salario_Mensual
Companias
Distancia_Casa
Area_Investigacion & Desarrollo
Campo_Educativo_Medicina
Rol_Director de fabricacion
Campo_Educativo_Biologia
Nivel_Educativo
Estado_Civil_Soltero
Campo_Educativo_Mercadeo
Rol_Tecnico de laboratorio
Campo_Educativo_Otra
Nivel_Rol
Rol_Gerente



In [80]:
```
#Modelo de Random Forecast ADASYN (solo importantes)

#Quitar columnas con menos importancia
cols_del_imp= ['Nivel_Rol','Rol_Gerente','Campo_Educativo_Tecnico','Rol_Ejecutivo de ventas','Rol_Representante de ventas',
               'Rol_Representante de salud','Rol_Recursos Humanos','Rol_Cientifico investigador',
               'Area_Recursos Humanos','Rol_Director de investigacion','Campo_Educativo_Recursos Humanos']

X_train_ada_imp = X_train_ada.loc[:, [name for name in X_train_ada.columns if name not in cols_del_imp]]
X_test_ada_imp = X_test_ada.loc[:, [name for name in X_test_ada.columns if name not in cols_del_imp]]
```

In [81]:
```
#Entrenar el modelo de nuevo con variables importantes
#Random Forest individual
rf_model_ada_imp =RandomForestClassifier(max_depth=5,random_state=1, class_weight = 'balanced')
rf_model_ada_imp.fit(X_train_ada_imp,y_train_ada)

#Prediccion sobre test
y_pred_rf_ada_imp = rf_model_ada_imp.predict(X_test_ada_imp)
```

In [82]:
```
mostrar_resultados(y_test_ada, y_pred_rf_ada_imp)
#Attrition 75%, Activos 83%
```



	precision	recall	f1-score	support
0	0.78	0.83	0.80	905
1	0.81	0.75	0.78	857
accuracy			0.79	1762
macro avg	0.79	0.79	0.79	1762
weighted avg	0.79	0.79	0.79	1762

In [83]:
```
metricas_comparacion(y_test_ada, y_pred_rf_ada_imp, y_train_ada, y_pred_rf_ada_train)
```

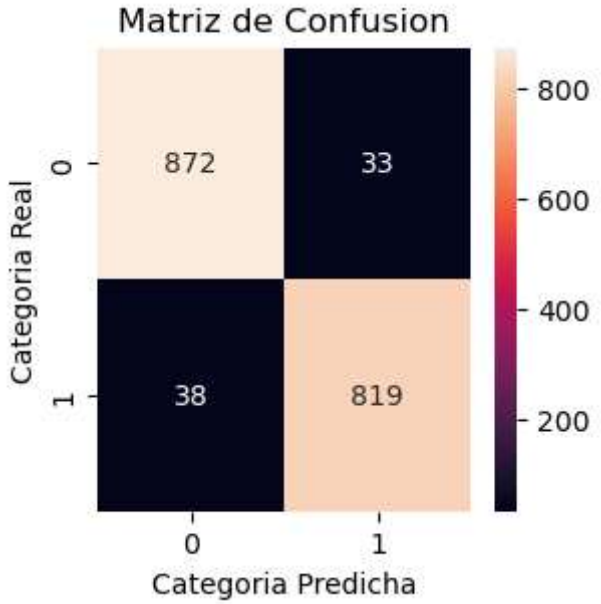
```
Accuracy:  0.7917139614074915
AUC test:  0.7906464152865257
AUC train:  0.8255194218608853
Delta AUC (overfitting):  0.034873006574359655
```

Entrenamiento ADASYN Gradient Boosting

In [84]:
```
#Gradient Boosting individual
gbm_model_ada = GradientBoostingClassifier(max_depth = 5,max_features = 'auto',random_state = 1)
gbm_model_ada.fit(X_train_ada,y_train_ada)
```

Out[84]:
```
GradientBoostingClassifier(max_depth=5, max_features='auto', random_state=1)
```

In [85]:
```
#Prediccion sobre el test
pred_y_gbm_ada = gbm_model_ada.predict(X_test_ada)
mostrar_resultados(y_test_ada, pred_y_gbm_ada)
#Recall de attrition de 96%, activos en 96%

#Prediccion sobre el train
pred_y_gbm_ada_train = gbm_model_ada.predict(X_train_ada)
```



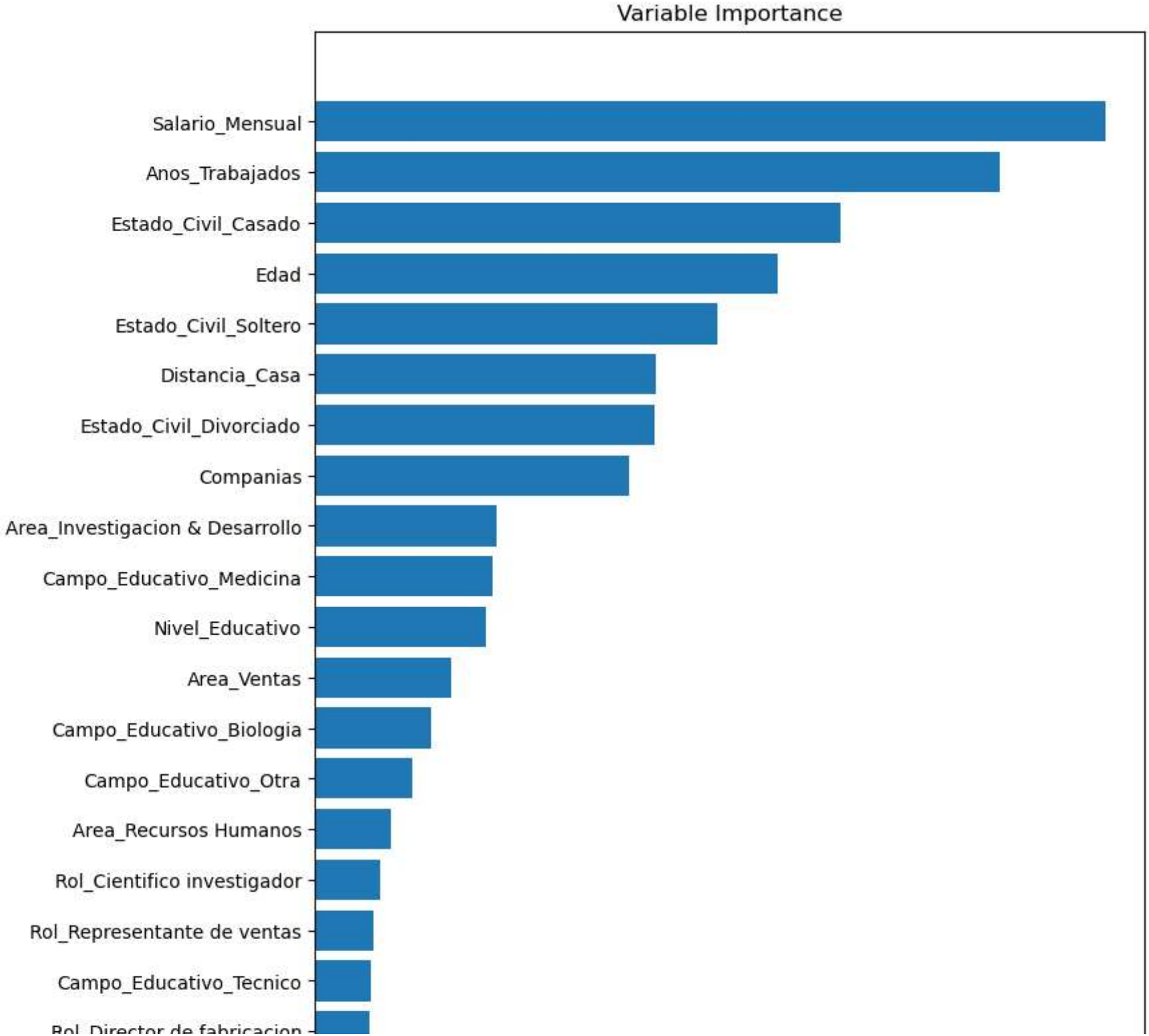
	precision	recall	f1-score	support
0	0.96	0.96	0.96	905
1	0.96	0.96	0.96	857
accuracy			0.96	1762
macro avg	0.96	0.96	0.96	1762
weighted avg	0.96	0.96	0.96	1762

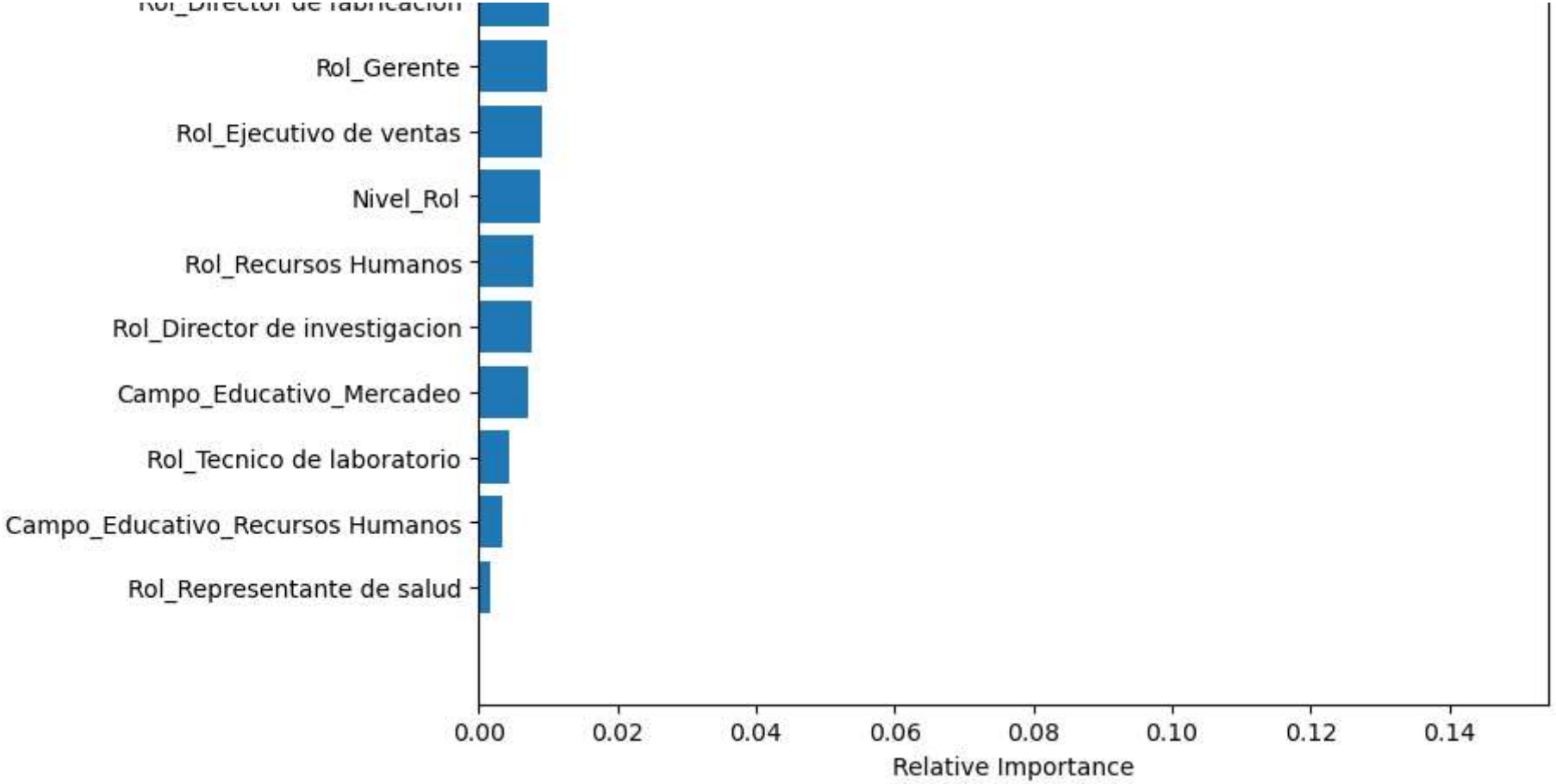
In [86]:
```
metricas_comparacion(y_test_ada, pred_y_gbm_ada, y_train_ada, pred_y_gbm_ada_train)
```

```
Accuracy:  0.9597048808172531
AUC test:  0.9595975940741506
AUC train:  0.9843276551813137
Delta AUC (overfitting):  0.024730061107163137
```

In [87]:

```
feature_importance = gbm_model_ada.feature_importances_
#Importancia relativa a la importancia maxima
feature_importance = 100.0 * (feature_importance / 100)
                             #feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5

plt.figure(figsize=(8, 15))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train_ada.keys()[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```


Variable Importance
Salario_Mensual
Anos_Trabajados
Estado_Civil_Casado
Edad
Estado_Civil_Soltero
Distancia_Casa
Estado_Civil_Divorciado
Companias
Area_Investigacion & Desarrollo
Campo_Educativo_Medicina
Nivel_Educativo
Area_Ventas
Campo_Educativo_Biologia
Campo_Educativo_Otra
Area_Recursos Humanos
Rol_Cientifico investigador
Rol_Representante de ventas
Campo_Educativo_Tecnico
Rol_Director de fabricacion

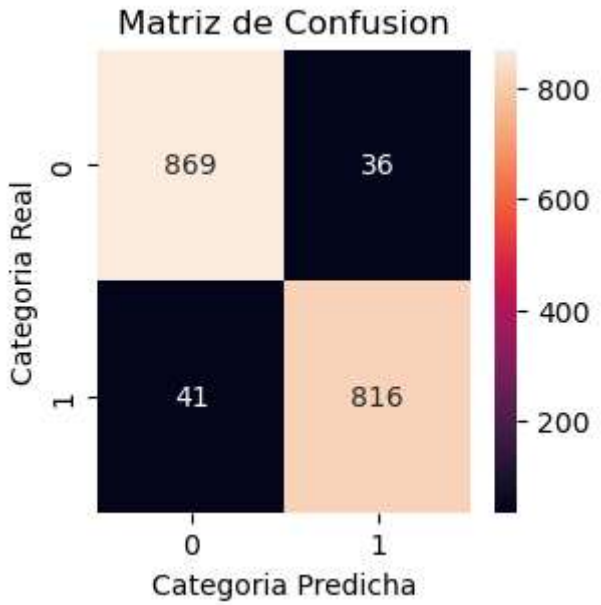


In [88]:
```
#Modelo de Gradient Boosting ADASYN (solo importantes)

#Quitar columnas con menos importancia
cols_del_imp = ['Area_Recursos Humanos','Rol_Cientifico investigador','Rol_Representante de ventas',
                'Campo_Educativo_Tecnico','Rol_Director de fabricacion','Rol_Gerente','Rol_Ejecutivo de ventas',
                'Nivel_Rol','Rol_Recursos Humanos','Rol_Director de investigacion','Campo_Educativo_Mercadeo',
                'Rol_Tecnico de laboratorio','Campo_Educativo_Recursos Humanos','Rol_Representante de salud']

X_train_ada_imp = X_train_ada.loc[:, [name for name in X_train_ada.columns if name not in cols_del_imp]]
X_test_ada_imp = X_test_ada.loc[:, [name for name in X_test_ada.columns if name not in cols_del_imp]]
```

In [89]:
```
#Entrenar de nuevo
gbm_model_ada_imp = GradientBoostingClassifier(max_depth = 5,max_features = 'auto',random_state = 1)
gbm_model_ada_imp.fit(X_train_ada_imp,y_train_ada)
```

Out[89]:
```
GradientBoostingClassifier(max_depth=5, max_features='auto', random_state=1)
```

In [90]:
```
#Prediccion sobre el test
pred_y_gbm_ada_imp = gbm_model_ada_imp.predict(X_test_ada_imp)
mostrar_resultados(y_test_ada, pred_y_gbm_ada_imp)
#Recall de attrition de 95%, activos en 96%

#Prediccion sobre el train
pred_y_gbm_ada_train_imp = gbm_model_ada_imp.predict(X_train_ada_imp)
```



	precision	recall	f1-score	support
0	0.95	0.96	0.96	905
1	0.96	0.95	0.95	857
accuracy			0.96	1762
macro avg	0.96	0.96	0.96	1762
weighted avg	0.96	0.96	0.96	1762

In [91]:
```
metricas_comparacion(y_test_ada, pred_y_gbm_ada_imp, y_train_ada, pred_y_gbm_ada_train_imp)
```

```
Accuracy:  0.9562996594778661
AUC test:  0.9561898437953287
AUC train:  0.9822402890695573
Delta AUC (overfitting):  0.026050445274228617
```

In [62]:
```
#feature_importance = gbm_model_ada_imp.feature_importances_
##Importancia relativa a la importancia maxima
#feature_importance = 100.0 * (feature_importance / 100)
```

```
#                               #feature_importance.max())
#sorted_idx = np.argsort(feature_importance)
#pos = np.arange(sorted_idx.shape[0]) + .5
#
#plt.figure(figsize=(8, 18))
#plt.barh(pos, feature_importance[sorted_idx], align='center')
#plt.yticks(pos, X_train_ada_imp.keys()[sorted_idx])
#plt.xlabel('Relative Importance')
#plt.title('Variable Importance')
#plt.show()
```

In [93]:
```
X_train.columns
```

Out[93]:
```
Index(['Edad', 'Distancia_Casa', 'Nivel_Educativo', 'Nivel_Rol',
       'Salario_Mensual', 'Companias', 'Anos_Trabajados',
       'Area_Investigacion & Desarrollo', 'Area_Recursos Humanos',
       'Area_Ventas', 'Campo_Educativo_Biologia', 'Campo_Educativo_Medicina',
       'Campo_Educativo_Mercadeo', 'Campo_Educativo_Otra',
       'Campo_Educativo_Recursos Humanos', 'Campo_Educativo_Tecnico',
       'Rol_Cientifico investigador', 'Rol_Director de fabricacion',
       'Rol_Director de investigacion', 'Rol_Ejecutivo de ventas',
       'Rol_Gerente', 'Rol_Recursos Humanos', 'Rol_Representante de salud',
       'Rol_Representante de ventas', 'Rol_Tecnico de laboratorio',
       'Estado_Civil_Casado', 'Estado_Civil_Divorciado',
       'Estado_Civil_Soltero'],
      dtype='object')
```

Búsqueda de mejor semilla sobre modelo elegido: Random Forest (ADASYN) con todas las variables

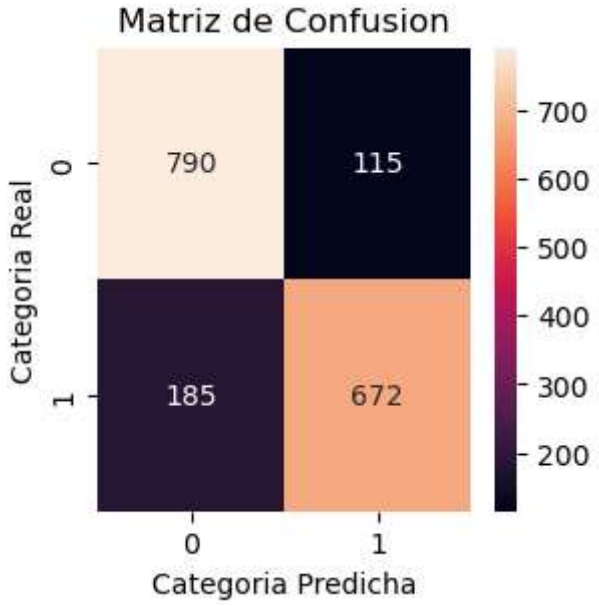
In [96]:
```
#Almacenamiento inicial de semilla y AUC
mejor_seed = 4455
mejor_auc = 0.8285294

#Iterador sobre semillas y predicciones
for i in range(2001,5001):
    #Random Forest individual
    rf_model_ada =RandomForestClassifier(max_depth=5,random_state=i, class_weight = 'balanced')
    rf_model_ada.fit(X_train_ada,y_train_ada)
    #Prediccion sobre test
    y_pred_rf_ada = rf_model_ada.predict(X_test_ada)
    #AUC test
    fpr, tpr, _ = metrics.roc_curve(y_test_ada,  y_pred_rf_ada)
    auc_test = metrics.roc_auc_score(y_test_ada, y_pred_rf_ada)
    if auc_test > mejor_auc:
        mejor_seed = i
```

```
        mejor_auc = auc_test

#Imprimir mejores resultados
print("Mejor Seed: ",mejor_seed)
print("Mejor AUC test: ",mejor_auc)

#corriendo hasta 5001
```

```
Mejor Seed:  4455
Mejor AUC test:  0.8285294326218274
```

In [97]:

```
#Resultados Random Forest ADASYN con semilla ganadora
rf_model_ada =RandomForestClassifier(max_depth=5,random_state=4455, class_weight = 'balanced')
rf_model_ada.fit(X_train_ada,y_train_ada)

#Prediccion sobre test
y_pred_rf_ada = rf_model_ada.predict(X_test_ada)

mostrar_resultados(y_test_ada, y_pred_rf_ada)
```



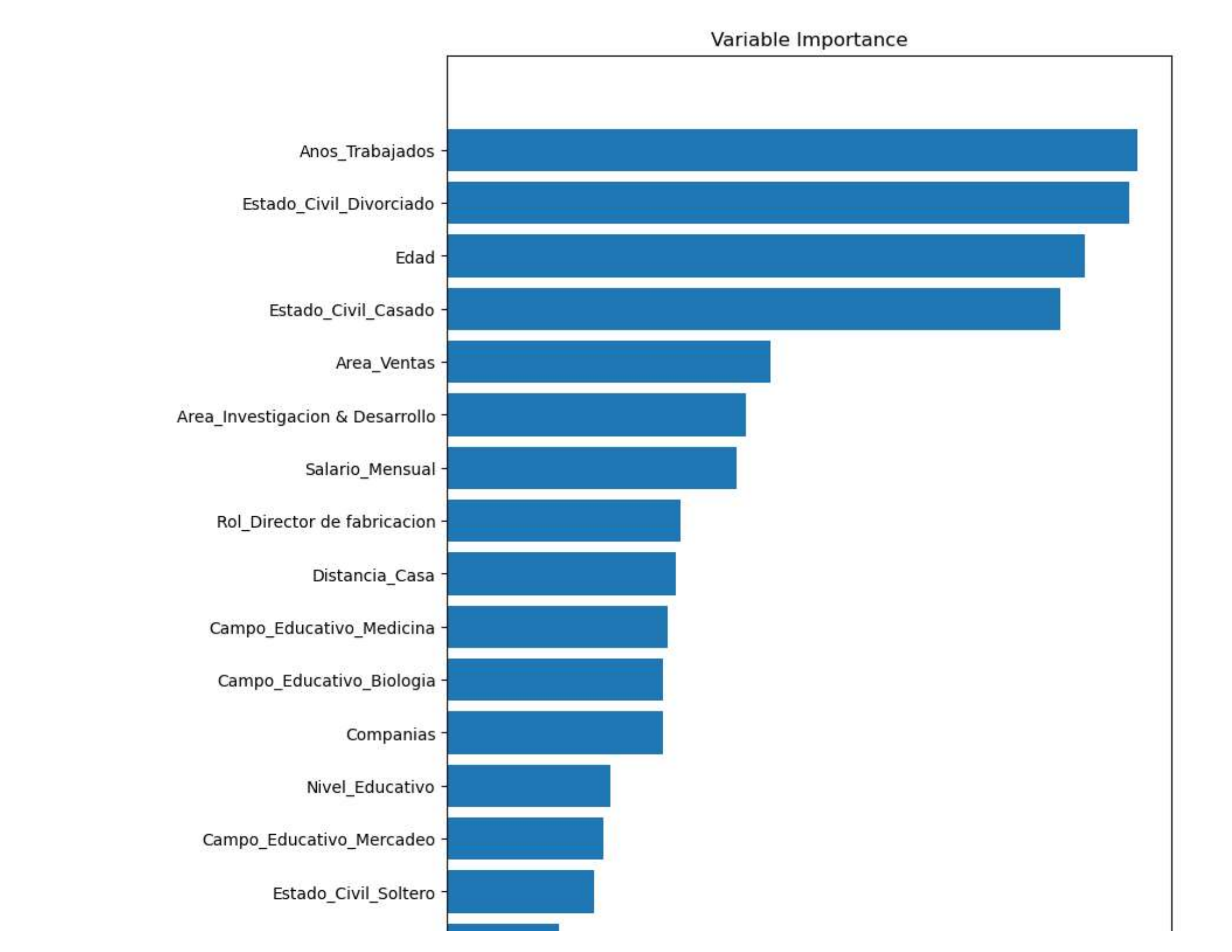
```
              precision    recall  f1-score   support

           0       0.81      0.87      0.84       905
           1       0.85      0.78      0.82       857

    accuracy                           0.83      1762
   macro avg       0.83      0.83      0.83      1762
weighted avg       0.83      0.83      0.83      1762
```

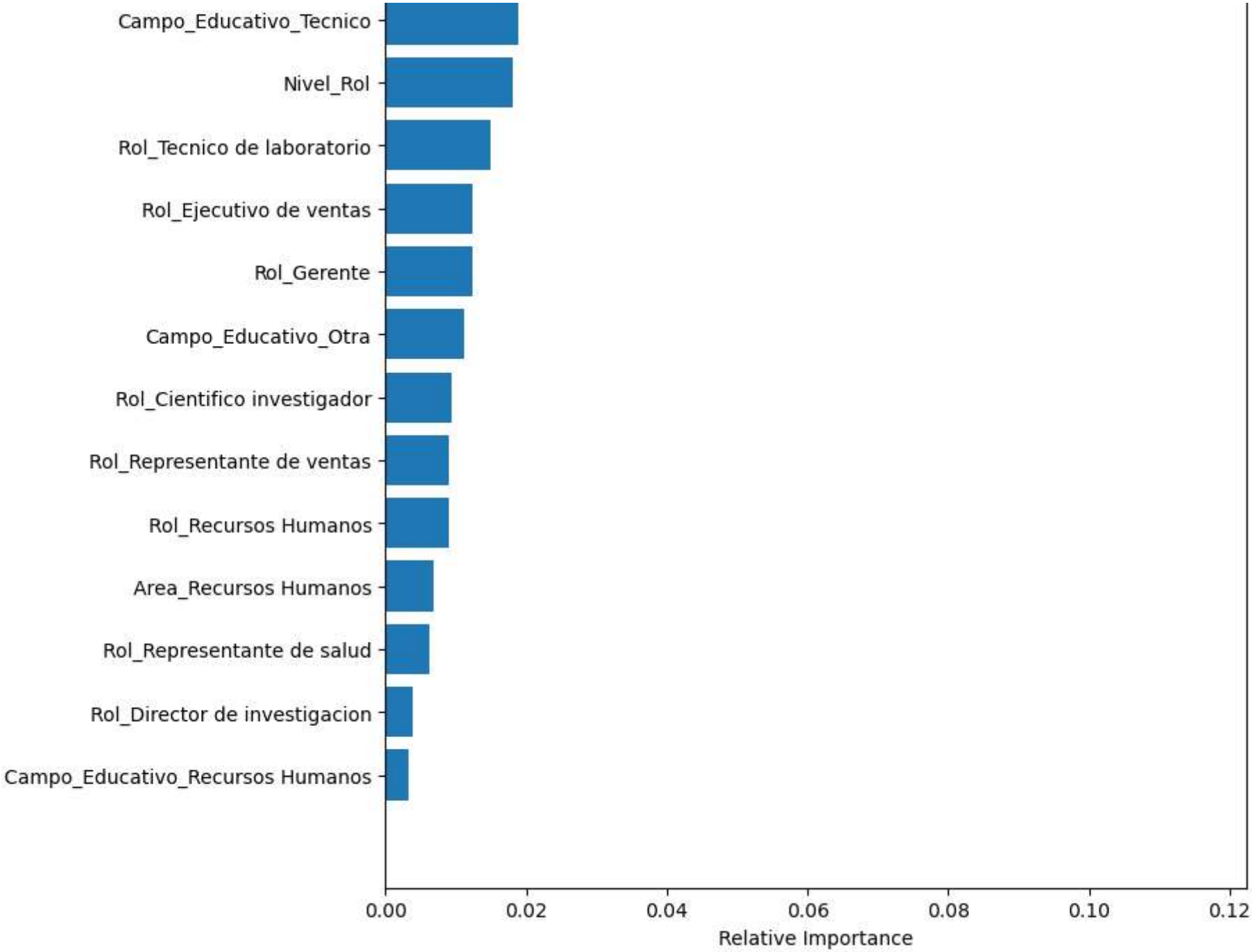
In [98]:
```
#Prediccion sobre el train
pred_y_rf_ada_train = rf_model_ada.predict(X_train_ada)
```

In [99]:
```
metricas_comparacion(y_test_ada, y_pred_rf_ada, y_train_ada, pred_y_rf_ada_train)
```

```
Accuracy:  0.8297389330306469
AUC test:  0.8285294326218274
AUC train:  0.8378429474770939
Delta AUC (overfitting):  0.009313514855266436
```

In [101…
```
#Hallar variables importantes
feature_importance = rf_model_ada.feature_importances_
#Importancia relativa a la importancia maxima
feature_importance = 100.0 * (feature_importance / 100)
                              #feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5

plt.figure(figsize=(8, 18))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train_ada.keys()[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```


Variable Importance
Anos_Trabajados
Estado_Civil_Divorciado
Edad
Estado_Civil_Casado
Area_Ventas
Area_Investigacion & Desarrollo
Salario_Mensual
Rol_Director de fabricacion
Distancia_Casa
Campo_Educativo_Medicina
Campo_Educativo_Biologia
Companias
Nivel_Educativo
Campo_Educativo_Mercadeo
Estado_Civil_Soltero


Campo_Educativo_Tecnico
Nivel_Rol
Rol_Tecnico de laboratorio
Rol_Ejecutivo de ventas
Rol_Gerente
Campo_Educativo_Otra
Rol_Cientifico investigador
Rol_Representante de ventas
Rol_Recursos Humanos
Area_Recursos Humanos
Rol_Representante de salud
Rol_Director de investigacion
Campo_Educativo_Recursos Humanos
0.00
0.02
0.04
0.06
0.08
0.10
0.12
Relative Importance

Modelamiento - Probabilidad de Deserción empleados activos

Preparación inicial

In [1]:
```
#Librerias numericas y graficos
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#Graficos
import plotly as py
import plotly.graph_objs as go

#Manejo de fechas
import datetime as dt

#Para Regresion Logistica
from sklearn.linear_model import LogisticRegression

#Modelos de Arboles
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

#Medidas
from sklearn import metrics
from sklearn import model_selection
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score

#Split X - y en conjuntos training y testing
from sklearn.model_selection import train_test_split

#Mostrar todas las columnas de frame
pd.set_option('display.max_columns', None)

#Asignar el formato de los decimales
np.set_printoptions(formatter={'float': lambda x: "{0:0.10f}".format(x)})
```

```
#Para datos desbalanceados
from pylab import rcParams

from imblearn.under_sampling import NearMiss
from imblearn.over_sampling import RandomOverSampler
from imblearn.combine import SMOTETomek
from imblearn.ensemble import BalancedBaggingClassifier
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import ADASYN

from collections import Counter
```

In [2]:
```
#Carga de datos
ruta = r'C:\Users\Jaime Andres Molina\Documents\Maestria\Proyecto Empresarial\Fuentes\01_Fuentes\all_attrition_modeling.csv'
#CSV
df = pd.read_csv (ruta, sep = ';')
```

In [3]:
```
#Tipos de datos
df.dtypes
```

Out[3]:

```
ID_empleado                   int64
Edad                          int64
Desercion                    object
Frecuencia_viaje             object
Area                         object
Distancia_Casa                int64
Nivel_Educativo               int64
Campo_Educativo              object
Recuento_Empleado             int64
Genero                       object
Nivel_Rol                     int64
Rol                          object
Estado_Civil                 object
Salario_Mensual             float64
Companias                     int64
Mayor18                      object
Porcentaje_Aumento_Salarial   int64
Horario                       int64
Nivel_Opcion_Acciones         int64
Anos_Trabajados               int64
Entrenamientos                int64
Antiguedad                    int64
Anos_Ultima_Promocion         int64
Anos_Lider_Actual             int64
Satisfaccion_Entorno          int64
Satisfaccion_Laboral          int64
Balance_VidaTrabajo           int64
CompromisoTrabajo             int64
Rendimiento                   int64
Prom_Horas                  float64
Porcentaje_Ausentismo       float64
Ausentismo                    int64
dtype: object
```

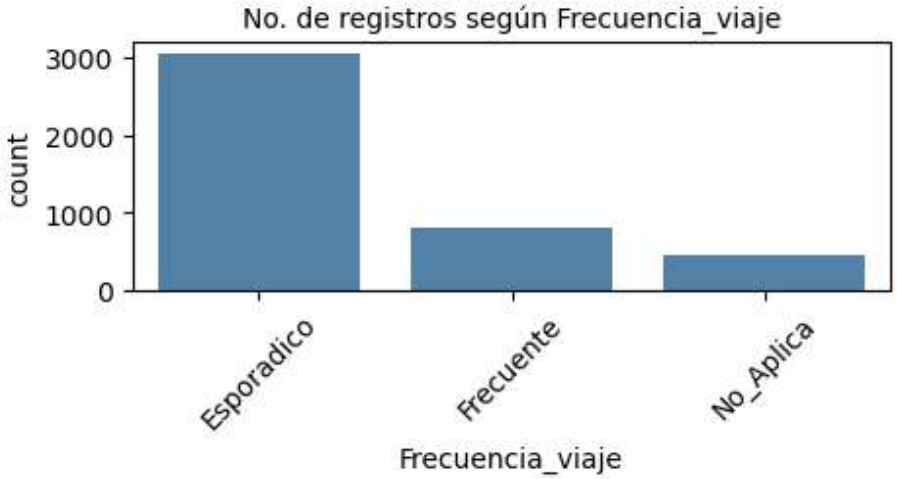
In [4]:

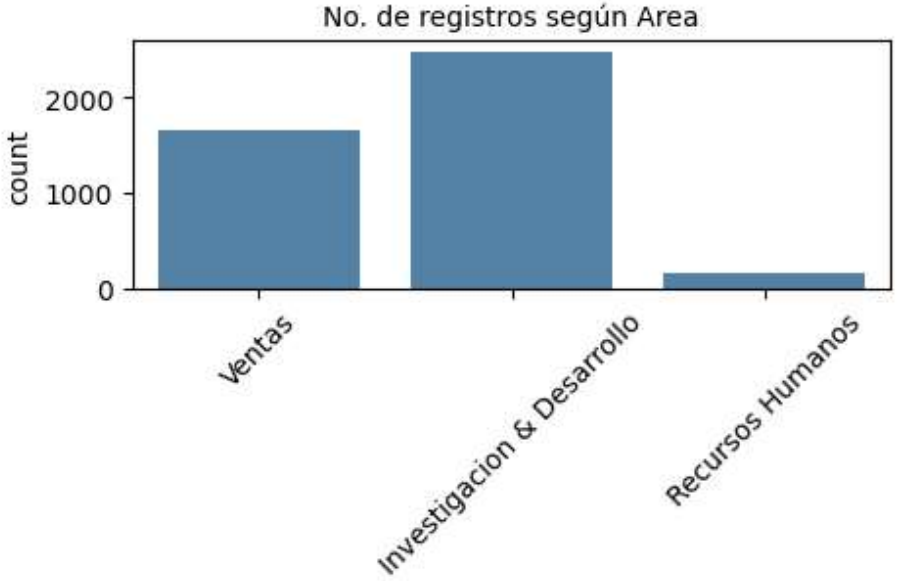
```
# Valores nulos por columna en porcentaje
df.isnull().sum()/df.shape[0]*100
```

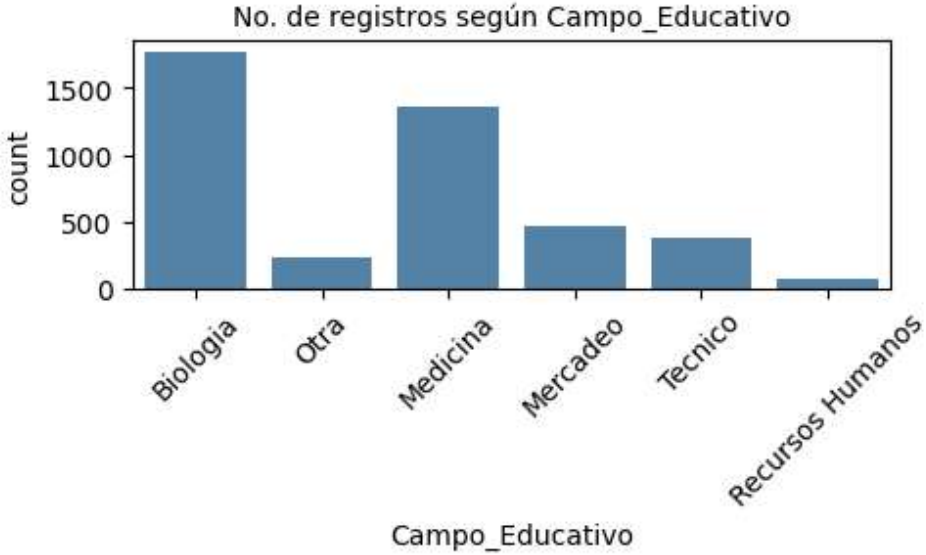
Out[4]:
```
ID_empleado                 0.0
Edad                        0.0
Desercion                   0.0
Frecuencia_viaje            0.0
Area                        0.0
Distancia_Casa              0.0
Nivel_Educativo             0.0
Campo_Educativo             0.0
Recuento_Empleado           0.0
Genero                      0.0
Nivel_Rol                   0.0
Rol                         0.0
Estado_Civil                0.0
Salario_Mensual             0.0
Companias                   0.0
Mayor18                     0.0
Porcentaje_Aumento_Salarial 0.0
Horario                     0.0
Nivel_Opcion_Acciones       0.0
Anos_Trabajados             0.0
Entrenamientos              0.0
Antiguedad                  0.0
Anos_Ultima_Promocion       0.0
Anos_Lider_Actual           0.0
Satisfaccion_Entorno        0.0
Satisfaccion_Laboral        0.0
Balance_VidaTrabajo         0.0
CompromisoTrabajo           0.0
Rendimiento                 0.0
Prom_Horas                  0.0
Porcentaje_Ausentismo       0.0
Ausentismo                  0.0
dtype: float64
```

In [5]:
```
#Histograma campos categoricos varias cajas
features=['Desercion', 'Frecuencia_viaje', 'Area','Campo_Educativo', 'Genero', 'Rol','Estado_Civil','Mayor18']
fig=plt.subplots(figsize=(11,14))
for i, j in enumerate(features):
    plt.subplot(4, 2, i+1)
    plt.subplots_adjust(hspace = 1.5)
    sns.countplot(x=j,data = df, color='steelblue')
    plt.xticks(rotation=45)
    plt.title("No. de registros según "+j, fontdict={'size': 10})
```

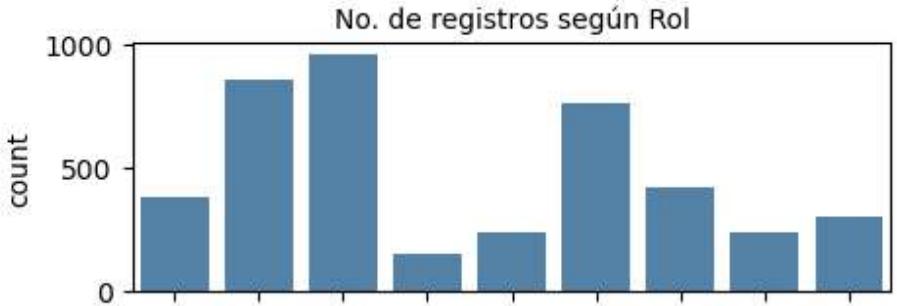

No. de registros según Desercion
count
3000
2000
1000
0
No
Si
Desercion

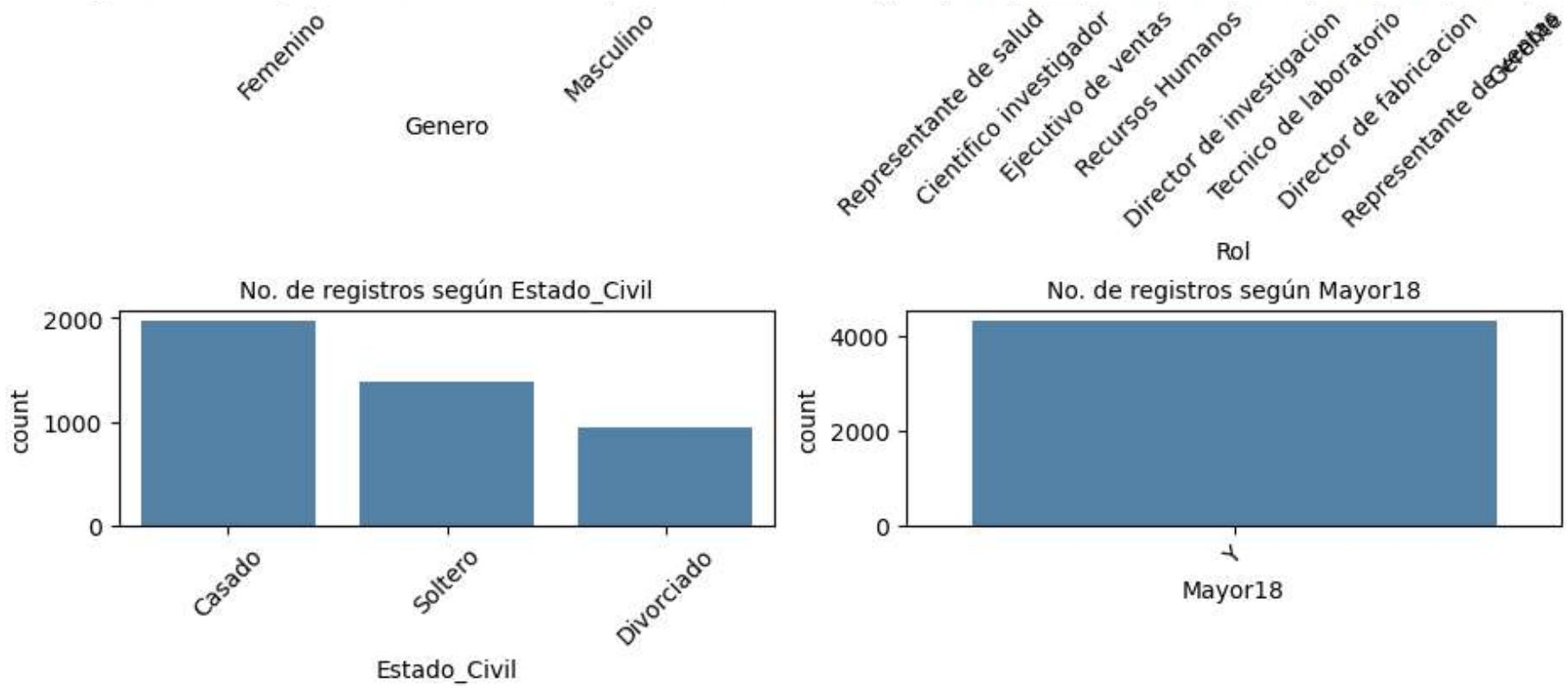

No. de registros según Frecuencia_viaje
count
3000
2000
1000
0
Esporadico
Frecuente
No_Aplica
Frecuencia_viaje


No. de registros según Area
count
2000
1000
0
Ventas
Investigacion & Desarrollo
Recursos Humanos
Area

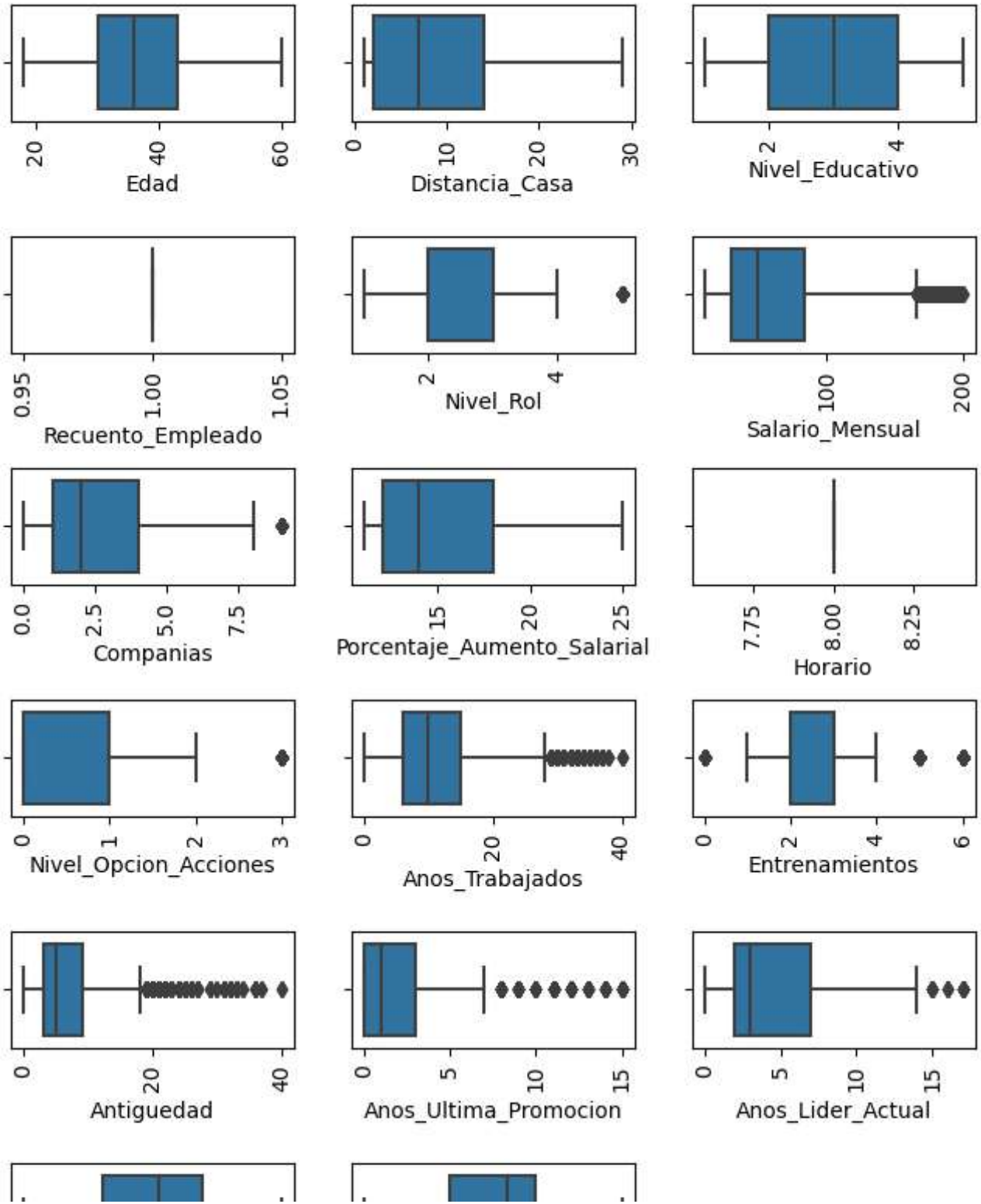

No. de registros según Campo_Educativo
count
1500
1000
500
0
Biologia
Otra
Medicina
Mercadeo
Tecnico
Recursos Humanos
Campo_Educativo

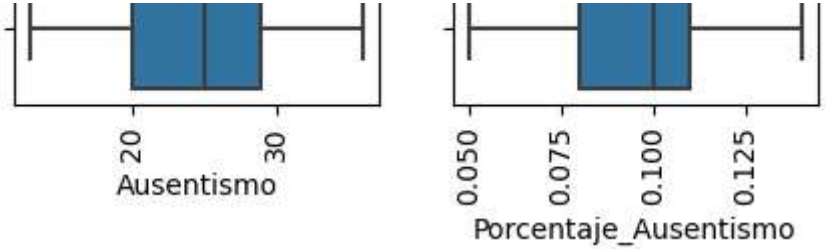

No. de registros según Genero
count
2000
1000
0


No. de registros según Rol
count
1000
500
0



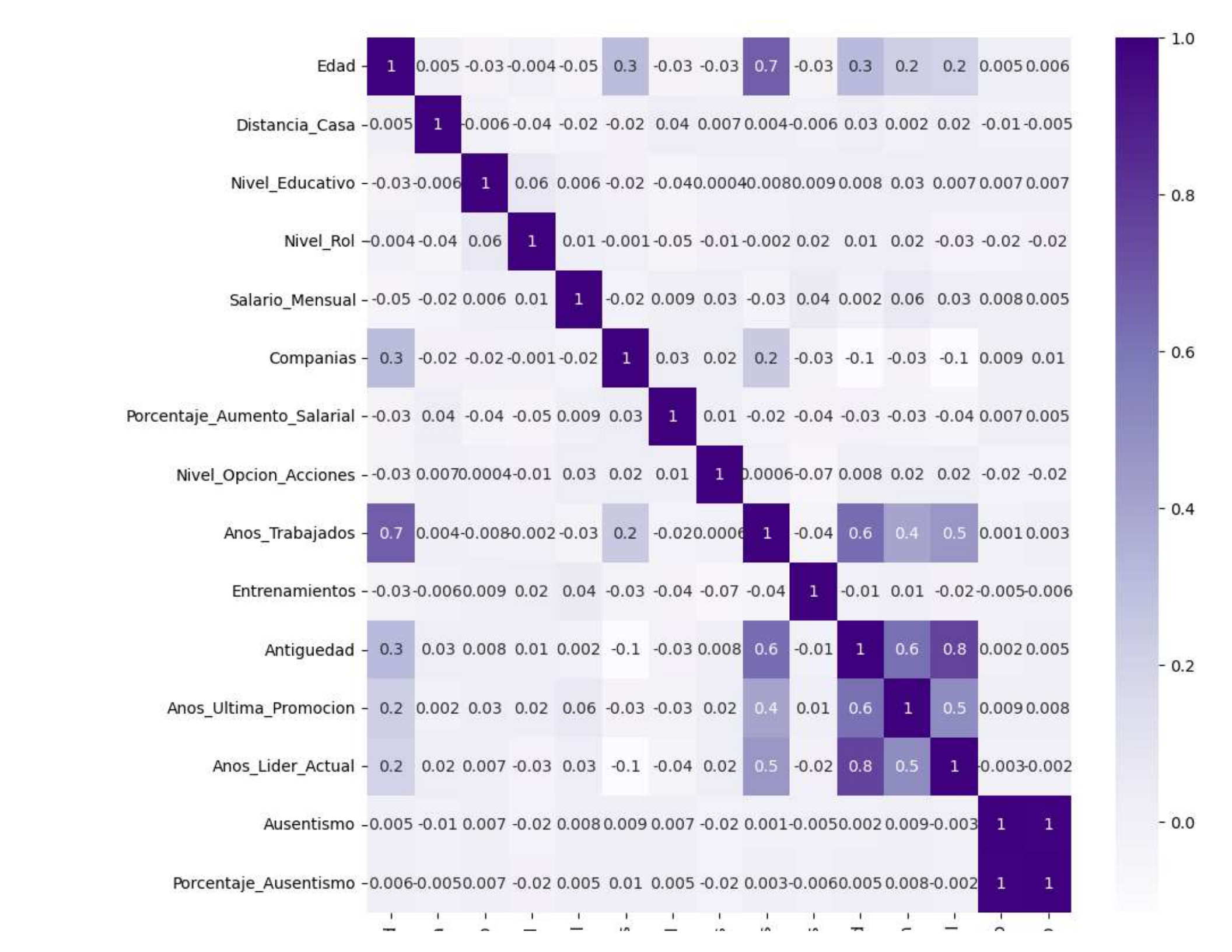
In [6]:
```
#Boxplots con seaborn multiples cajas
features=['Edad', 'Distancia_Casa', 'Nivel_Educativo', 'Recuento_Empleado', 'Nivel_Rol', 'Salario_Mensual',
       'Companias', 'Porcentaje_Aumento_Salarial', 'Horario', 'Nivel_Opcion_Acciones', 'Anos_Trabajados',
       'Entrenamientos', 'Antiguedad', 'Anos_Ultima_Promocion', 'Anos_Lider_Actual','Ausentismo',
        'Porcentaje_Ausentismo']
fig=plt.subplots(figsize=(8,30))
for i, j in enumerate(features):
    plt.subplot(16, 3, i+1)
    plt.subplots_adjust(hspace = 1.0)
    sns.boxplot(x=df[j])
    plt.xticks(rotation=90)
```


Edad
Distancia_Casa
Nivel_Educativo
Recuento_Empleado
Nivel_Rol
Salario_Mensual
Companias
Porcentaje_Aumento_Salarial
Horario
Nivel_Opcion_Acciones
Anos_Trabajados
Entrenamientos
Antiguedad
Anos_Ultima_Promocion
Anos_Lider_Actual



In [7]:
```
#Grafico de correlacion
corr_features=['Edad', 'Distancia_Casa', 'Nivel_Educativo', 'Nivel_Rol', 'Salario_Mensual',
               'Companias', 'Porcentaje_Aumento_Salarial', 'Nivel_Opcion_Acciones', 'Anos_Trabajados',
               'Entrenamientos', 'Antiguedad', 'Anos_Ultima_Promocion', 'Anos_Lider_Actual','Ausentismo',
               'Porcentaje_Ausentismo']

corr = df[corr_features].corr()
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values,
            #cmap=sns.diverging_palette(220, 10, as_cmap=True),
            annot=True, cmap="Purples", fmt = '.1g')
```

Out[7]:
```
<AxesSubplot:>
```


Edad 1 0.005 -0.03 -0.004 -0.05 0.3 -0.03 -0.03 0.7 -0.03 0.3 0.2 0.2 0.005 0.006
Distancia_Casa -0.005 1 -0.006 -0.04 -0.02 -0.02 0.04 0.007 0.004 -0.006 0.03 0.002 0.02 -0.01 -0.005
Nivel_Educativo -0.03 -0.006 1 0.06 0.006 -0.02 -0.04 0.0004 0.008 0.009 0.008 0.03 0.007 0.007 0.007
Nivel_Rol -0.004 -0.04 0.06 1 0.01 -0.001 -0.05 -0.01 -0.002 0.02 0.01 0.02 -0.03 -0.02 -0.02
Salario_Mensual -0.05 -0.02 0.006 0.01 1 -0.02 0.009 0.03 -0.03 0.04 0.002 0.06 0.03 0.008 0.005
Companias 0.3 -0.02 -0.02 -0.001 -0.02 1 0.03 0.02 0.2 -0.03 -0.1 -0.03 -0.1 0.009 0.01
Porcentaje_Aumento_Salarial -0.03 0.04 -0.04 -0.05 0.009 0.03 1 0.01 -0.02 -0.04 -0.03 -0.03 -0.04 0.007 0.005
Nivel_Opcion_Acciones -0.03 0.007 0.0004 -0.01 0.03 0.02 0.01 1 0.0006 -0.07 0.008 0.02 0.02 -0.02 -0.02
Anos_Trabajados 0.7 0.004 -0.008 -0.002 -0.03 0.2 -0.02 0.0006 1 -0.04 0.6 0.4 0.5 0.001 0.003
Entrenamientos -0.03 -0.006 0.009 0.02 0.04 -0.03 -0.04 -0.07 -0.04 1 -0.01 0.01 -0.02 -0.005 -0.006
Antiguedad 0.3 0.03 0.008 0.01 0.002 -0.1 -0.03 0.008 0.6 -0.01 1 0.6 0.8 0.002 0.005
Anos_Ultima_Promocion 0.2 0.002 0.03 0.02 0.06 -0.03 -0.03 0.02 0.4 0.01 0.6 1 0.5 0.009 0.008
Anos_Lider_Actual 0.2 0.02 0.007 -0.03 0.03 -0.1 -0.04 0.02 0.5 -0.02 0.8 0.5 1 -0.003 -0.002
Ausentismo -0.005 -0.01 0.007 -0.02 0.008 0.009 0.007 -0.02 0.001 -0.005 0.002 0.009 -0.003 1 1
Porcentaje_Ausentismo -0.006 -0.005 0.007 -0.02 0.005 0.01 0.005 -0.02 0.003 -0.006 0.005 0.008 -0.002 1 1
1.0
0.8
0.6
0.4
0.2
0.0

Edad
Distancia_Casa
Nivel_Educativo
Nivel_Rol
Salario_Mensua
Companias
Porcentaje_Aumento_Salarial
Nivel_Opcion_Acciones
Anos_Trabajados
Entrenamientos
Antiguedad
Anos_Ultima_Promocion
Anos_Lider_Actua
Ausentismo
Porcentaje_Ausentismo

In [8]:
```
#Quitar columnas por nombre
cols_del = ['ID_empleado','Mayor18','Recuento_Empleado','Horario','Genero','Anos_Lider_Actual','Ausentismo']
df_in = df.loc[:, [name for name in df.columns if name not in cols_del]]
```

In [9]:
```
df_in.head()
```

Out[9]:

	Edad	Desercion	Frecuencia_viaje	Area	Distancia_Casa	Nivel_Educativo	Campo_Educativo	Nivel_Rol	Rol	Estado_Civil	Salario_Mensu
0	51	No	Esporadico	Ventas	6	2	Biologia	2	Representante de salud	Casado	131.
1	31	Si	Frecuente	Investigacion & Desarrollo	10	1	Biologia	3	Cientifico investigador	Soltero	41.
2	32	No	Frecuente	Ventas	17	4	Otra	3	Ejecutivo de ventas	Casado	193.
3	38	No	No_Aplica	Recursos Humanos	2	5	Biologia	3	Recursos Humanos	Casado	83.
4	32	No	Esporadico	Ventas	10	1	Medicina	3	Ejecutivo de ventas	Soltero	23.

In [10]:
```
#Dummificar variables seleccionadas
df_in= pd.get_dummies(df_in, columns=['Desercion', 'Frecuencia_viaje', 'Area','Campo_Educativo',
                                      'Rol', 'Estado_Civil'])
df_in.head()
```

Out[10]:

	Edad	Distancia_Casa	Nivel_Educativo	Nivel_Rol	Salario_Mensual	Companias	Porcentaje_Aumento_Salarial	Nivel_Opcion_Acciones	Anos_Trabajados	Er
0	51	6	2	2	131.16	1	11	0	1	
1	31	10	1	3	41.89	0	23	1	6	
2	32	17	4	3	193.28	1	15	3	5	
3	38	2	5	3	83.21	3	11	3	13	
4	32	10	1	3	23.42	4	12	2	9	

In [11]:
```
#Quitar columnas redundantes
cols_del_in = ['Desercion_No']
df_in = df_in.loc[:, [name for name in df_in.columns if name not in cols_del_in]]
```

Entrenamiento de Modelos

In [12]:
```
#Hay desbalanceo de clases en la variable objetivo
print(df_in.shape)
print(pd.value_counts(df_in['Desercion_Si'], sort = True))
```

```
(4300, 44)
0    3605
1     695
Name: Desercion_Si, dtype: int64
```

In [13]:
```
#Resultados con modelo normal sin estrategia de desbalance para tener punto de comparacion

#Definimos nuestras etiquetas y features
y = df_in['Desercion_Si']
X = df_in.drop('Desercion_Si', axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=16)
```

In [97]:
```
#Modelo de Regresion Logistica
logreg_model = LogisticRegression(C=1.0,penalty='l2',random_state=1,max_iter=10000)
logreg_model.fit(X_train, y_train)
```

Out[97]:
```
LogisticRegression(max_iter=10000, random_state=1)
```

In [14]:
```
#Definimos funcion para mostrar los resultados
def mostrar_resultados(y_test, pred_y):
```

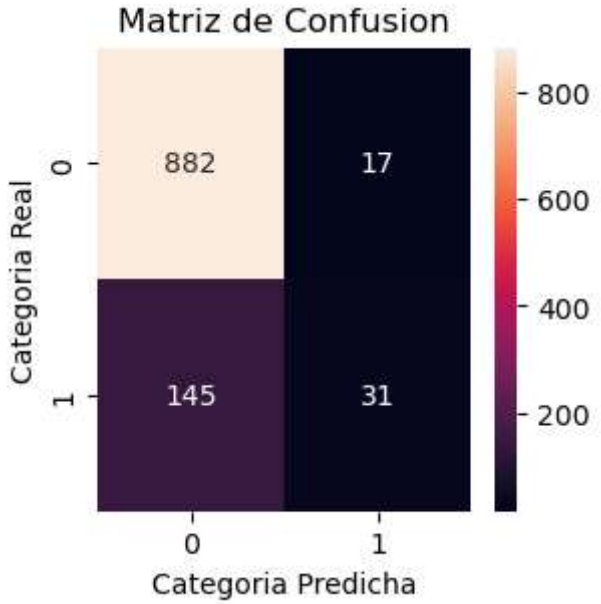
```
    conf_matrix = confusion_matrix(y_test, pred_y)
    plt.figure(figsize=(3, 3))
    sns.heatmap(conf_matrix, annot=True, fmt="d");
    plt.title("Matriz de Confusion")
    plt.ylabel('Categoria Real')
    plt.xlabel('Categoria Predicha')
    plt.show()
    print (classification_report(y_test, pred_y))
```

In [15]:

```
#Definimos funcion para medidas de comparación
def metricas_comparacion(y_test, pred_test, y_train, pred_train):
    #Accuracy
    accuracy_comparacion = metrics.accuracy_score(y_test, pred_test)
    print("Accuracy: ",accuracy_comparacion)

    #AUC test
    fpr, tpr, _ = metrics.roc_curve(y_test,  pred_test)
    auc_test = metrics.roc_auc_score(y_test, pred_test)
    print("AUC test: ",auc_test)

    #AUC train
    fpr_train, tpr_train, _train = metrics.roc_curve(y_train,  pred_train)
    auc_train = metrics.roc_auc_score(y_train, pred_train)
    print("AUC train: ",auc_train)

    #Delta AUC
    print("Delta AUC (overfitting): ",abs(auc_test-auc_train))
```

In [100…

```
#Resultados de prediccion
pred_y = logreg_model.predict(X_test)
mostrar_resultados(y_test, pred_y)
#Recall de attrition de 18% a mejorar, activos en 98%
```



	precision	recall	f1-score	support
0	0.86	0.98	0.92	899
1	0.65	0.18	0.28	176
accuracy			0.85	1075
macro avg	0.75	0.58	0.60	1075
weighted avg	0.82	0.85	0.81	1075

Aplicando SMOTE sobre la base (oversampling)

In [16]:
```
sm = SMOTE(sampling_strategy='auto', k_neighbors=5, random_state=1)
X_sm, y_sm = sm.fit_resample(X, y)
```

In [17]:
```
#Nuevos registros por categoria
print(y_sm.shape)
y_sm.value_counts()
```

Out[17]:
```
(7210,)
0    3605
1    3605
Name: Desercion_Si, dtype: int64
```

In [18]:
```
#Dividir la base en train y test de SMOTE
X_train_sm, X_test_sm, y_train_sm, y_test_sm = train_test_split(X_sm, y_sm, test_size=0.25, random_state=28)
```

Aplicando ADASYN sobre la base (oversampling)

In [19]:
```
ada = ADASYN(sampling_strategy='auto', n_neighbors=5, random_state=1)
X_ada, y_ada = ada.fit_resample(X, y)
```

In [20]:
```
#Nuevos registros por categoria
print(y_ada.shape)
y_ada.value_counts()
```

Out[20]:
```
(7096,)
0    3605
1    3491
Name: Desercion_Si, dtype: int64
```

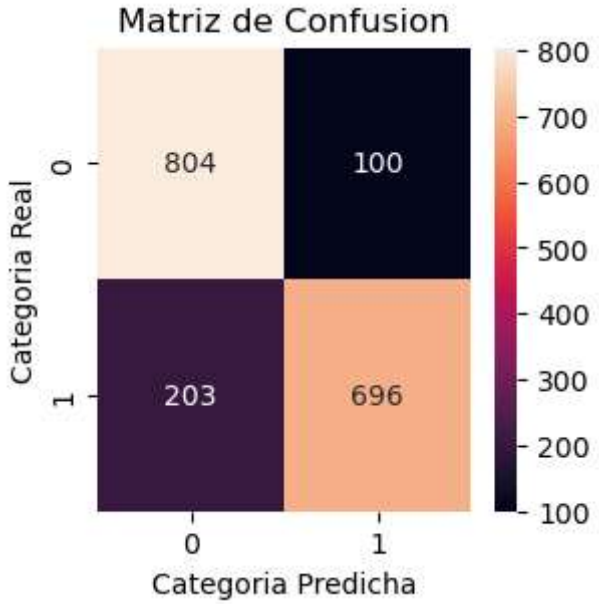
In [21]:
```
#Dividir la base en train y test de ADASYN
X_train_ada, X_test_ada, y_train_ada, y_test_ada = train_test_split(X_ada, y_ada, test_size=0.25, random_state=28)
```

Entrenamiento SMOTE Logistic Regression

In [107…
```
#Modelo de Regresion Logistica SMOTE
logreg_model_sm = LogisticRegression(C=1.0,penalty='l2',random_state=1,max_iter=10000)
logreg_model_sm.fit(X_train_sm, y_train_sm)
```

Out[107]:
```
LogisticRegression(max_iter=10000, random_state=1)
```

In [108…
```
#Prediccion sobre el test
pred_y_log_sm = logreg_model_sm.predict(X_test_sm)
mostrar_resultados(y_test_sm, pred_y_log_sm)
#Recall de attrition de 77%, activos en 89%

#Prediccion sobre el train
pred_y_log_sm_train = logreg_model_sm.predict(X_train_sm)
```



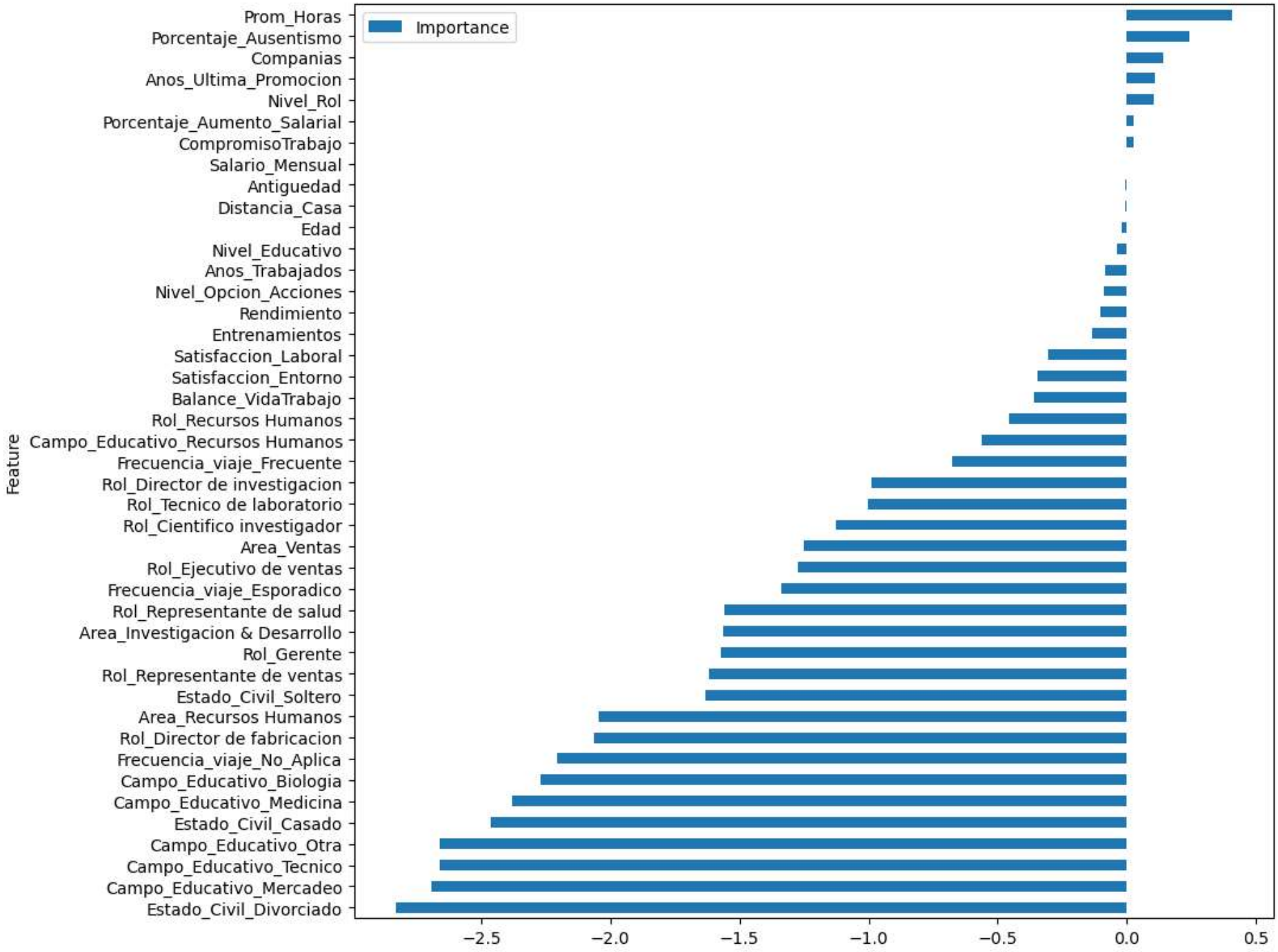
	precision	recall	f1-score	support
0	0.80	0.89	0.84	904
1	0.87	0.77	0.82	899
accuracy			0.83	1803
macro avg	0.84	0.83	0.83	1803
weighted avg	0.84	0.83	0.83	1803

In [109…
```
metricas_comparacion(y_test_sm, pred_y_log_sm, y_train_sm, pred_y_log_sm_train)
```

```
Accuracy:  0.831946755407654
AUC test:  0.831787039680274
AUC train:  0.8323102664064909
Delta AUC (overfitting):  0.0005232267262168522
```

In [111…
```
#Importancia de las variables
coefficients_log_sm = logreg_model_sm.coef_[0]

ft_importance_log_sm = pd.DataFrame({'Feature': X_sm.columns, 'Importance': coefficients_log_sm})
ft_importance_log_sm = ft_importance_log_sm.sort_values('Importance', ascending=True)
ft_importance_log_sm.plot(x='Feature', y='Importance', kind='barh', figsize=(10, 10))
```

Out[111]:
```
<AxesSubplot:ylabel='Feature'>
```



In [112…

```
#Modelo de Regresion Logistica SMOTE (solo importantes)

#Quitar columnas con menos importancia
```

```
cols_del_imp = ['Porcentaje_Aumento_Salarial','CompromisoTrabajo','Salario_Mensual','Antiguedad','Distancia_Casa']

X_train_sm_imp = X_train_sm.loc[:, [name for name in X_train_sm.columns if name not in cols_del_imp]]
X_test_sm_imp = X_test_sm.loc[:, [name for name in X_test_sm.columns if name not in cols_del_imp]]
```

In [113…

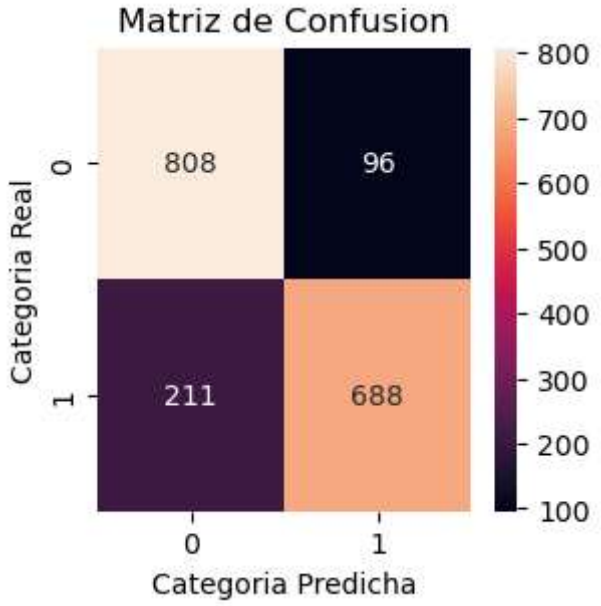
```
#Entrenar de nuevo
logreg_model_sm_imp = LogisticRegression(C=1.0,penalty='l2',random_state=1,max_iter=10000)
logreg_model_sm_imp.fit(X_train_sm_imp, y_train_sm)
```

Out[113]:

```
LogisticRegression(max_iter=10000, random_state=1)
```

In [114…

```
#Prediccion sobre el test
pred_y_log_sm_imp = logreg_model_sm_imp.predict(X_test_sm_imp)
mostrar_resultados(y_test_sm, pred_y_log_sm_imp)
#Recall de attrition de 77%, activos en 89%

#Prediccion sobre el train
pred_y_log_sm_train_imp = logreg_model_sm_imp.predict(X_train_sm_imp)
```



	precision	recall	f1-score	support
0	0.79	0.89	0.84	904
1	0.88	0.77	0.82	899
accuracy			0.83	1803
macro avg	0.84	0.83	0.83	1803
weighted avg	0.84	0.83	0.83	1803

In [115…
```
metricas_comparacion(y_test_sm, pred_y_log_sm_imp, y_train_sm, pred_y_log_sm_train_imp)
```

```
Accuracy:  0.8297282307265669
AUC test:  0.8295500408516838
AUC train:  0.8260279308558626
Delta AUC (overfitting):  0.0035221099958212543
```

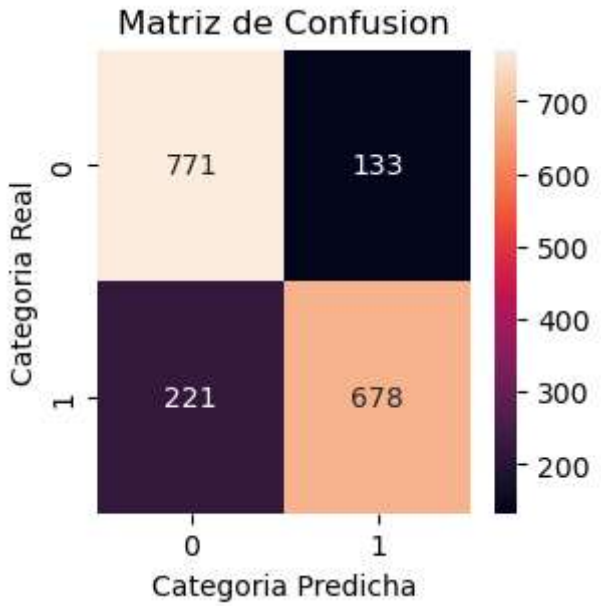
In [39]:
```
#Importancia de las variables
#coefficients_log_sm_imp = logreg_model_sm_imp.coef_[0]

#ft_importance_log_sm_imp = pd.DataFrame({'Feature': X_train_sm_imp.columns, 'Importance': coefficients_log_sm_imp})
#ft_importance_log_sm_imp = ft_importance_log_sm_imp.sort_values('Importance', ascending=True)
#ft_importance_log_sm_imp.plot(x='Feature', y='Importance', kind='barh', figsize=(10, 10))
```

Entrenamiento SMOTE Decision Tree

In [35]:
```
#Arbol de Decision individual SMOTE
dectree_model_sm = DecisionTreeClassifier(max_depth= 5, random_state = 1)
dectree_model_sm.fit(X_train_sm,y_train_sm)
```

Out[35]:
```
DecisionTreeClassifier(max_depth=5, random_state=1)
```

In [36]:
```
#Prediccion sobre el test
pred_y_dectree_sm = dectree_model_sm.predict(X_test_sm)
mostrar_resultados(y_test_sm, pred_y_dectree_sm)
#Recall de attrition de 75%, activos en 85%

#Prediccion sobre el train
pred_y_dectree_sm_train = dectree_model_sm.predict(X_train_sm)
```



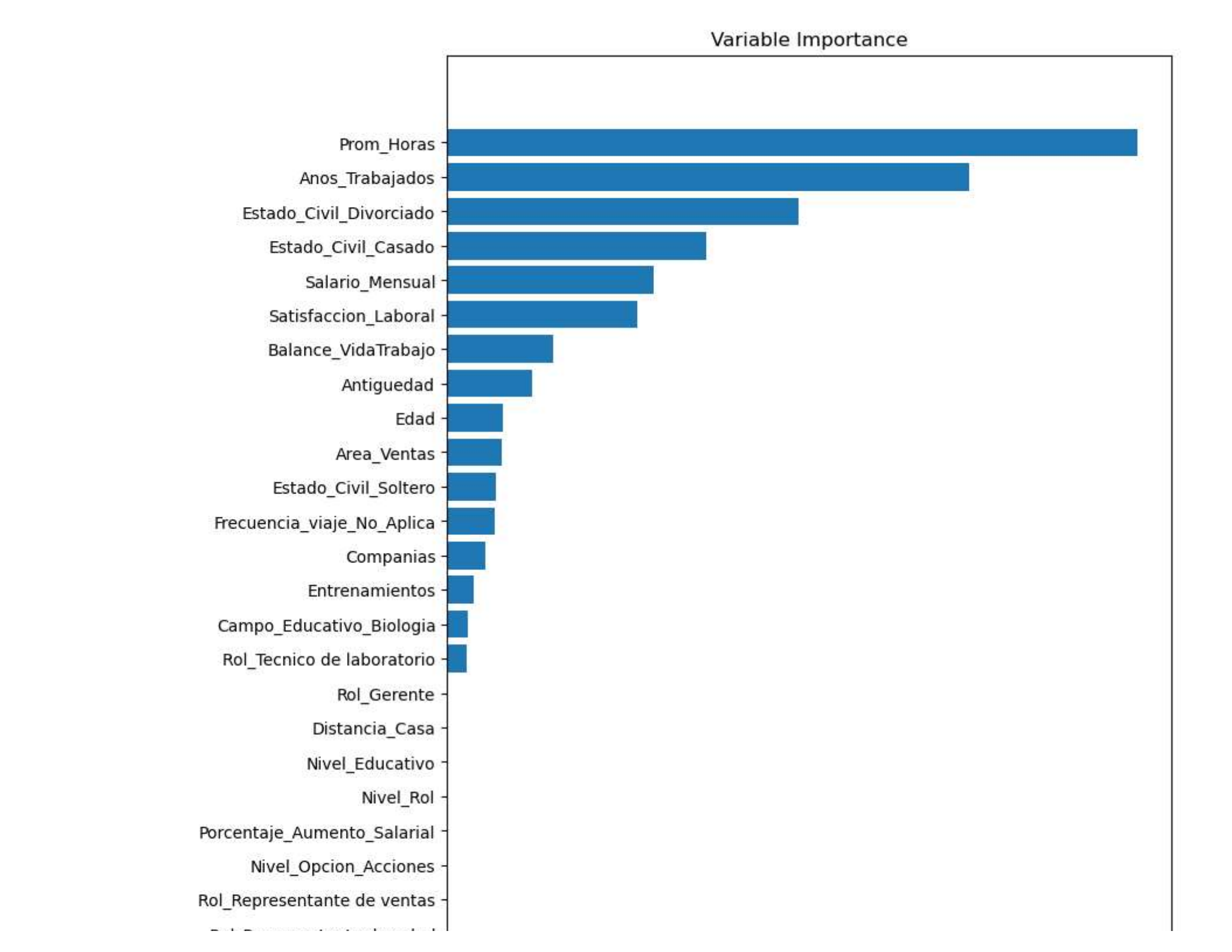
	precision	recall	f1-score	support
0	0.78	0.85	0.81	904
1	0.84	0.75	0.79	899
accuracy			0.80	1803
macro avg	0.81	0.80	0.80	1803
weighted avg	0.81	0.80	0.80	1803

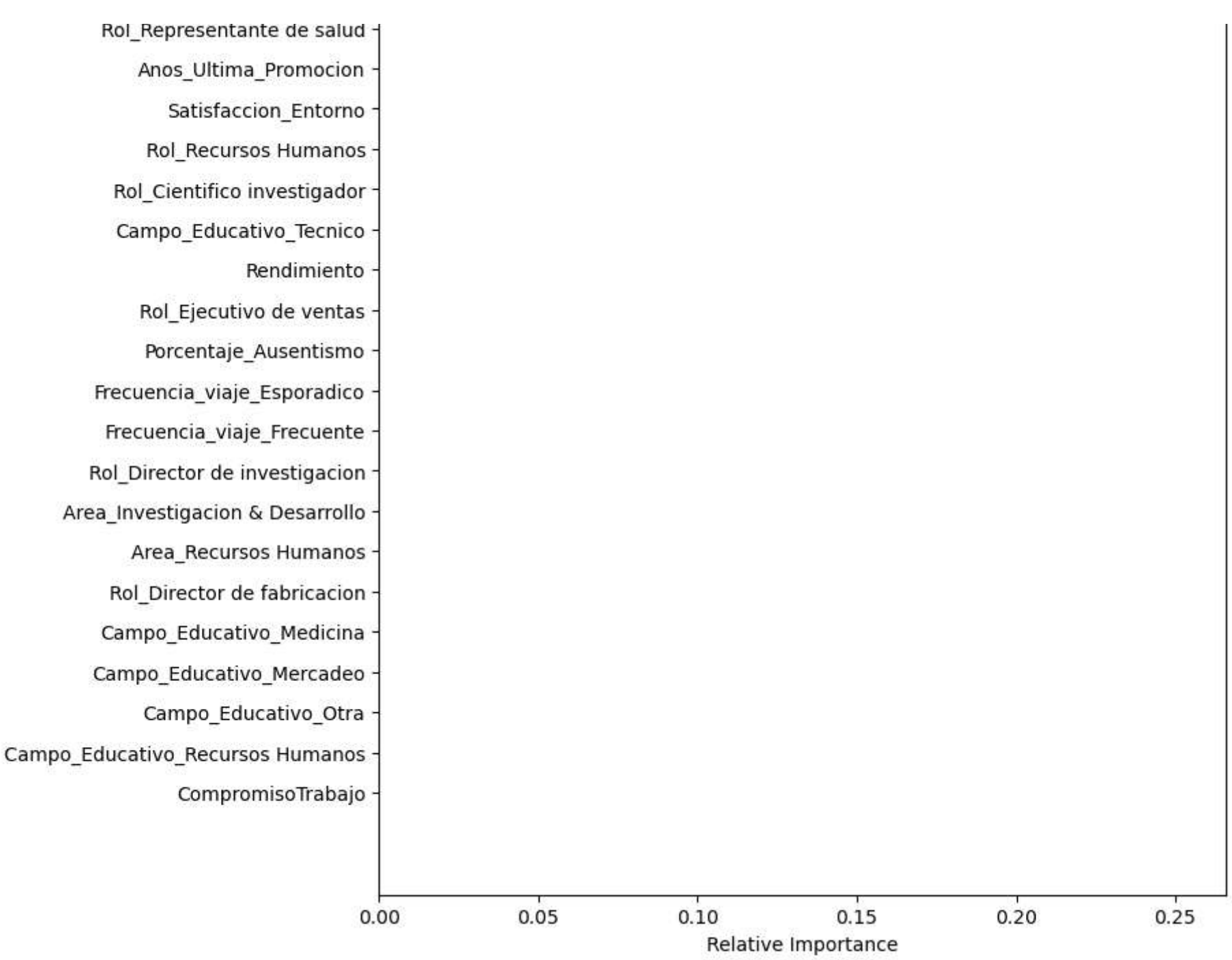
In [37]:
```
metricas_comparacion(y_test_sm, pred_y_dectree_sm, y_train_sm, pred_y_dectree_sm_train)
```

```
Accuracy:  0.8036605657237936
AUC test:  0.8035237038203706
AUC train:  0.8036363444816503
Delta AUC (overfitting):  0.00011264066127969485
```

In [38]:
```
feature_importance = dectree_model_sm.feature_importances_
#Importancia relativa a la importancia maxima
feature_importance = 100.0 * (feature_importance / 100)
                             #feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5

plt.figure(figsize=(8, 18))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train_sm.keys()[sorted_idx])
plt.xlabel('Relative Importance')
```

```
plt.title('Variable Importance')
plt.show()
```


Variable Importance
Prom_Horas
Anos_Trabajados
Estado_Civil_Divorciado
Estado_Civil_Casado
Salario_Mensual
Satisfaccion_Laboral
Balance_VidaTrabajo
Antiguedad
Edad
Area_Ventas
Estado_Civil_Soltero
Frecuencia_viaje_No_Aplica
Companias
Entrenamientos
Campo_Educativo_Biologia
Rol_Tecnico de laboratorio
Rol_Gerente
Distancia_Casa
Nivel_Educativo
Nivel_Rol
Porcentaje_Aumento_Salarial
Nivel_Opcion_Acciones
Rol_Representante de ventas



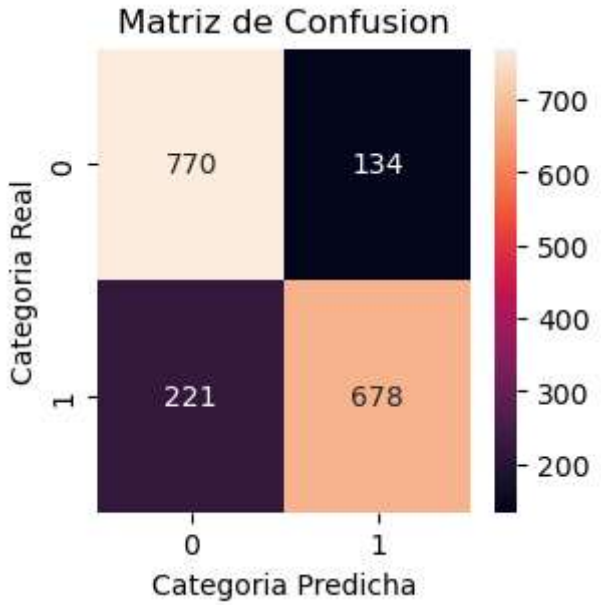
In [39]:
```
#Modelo de Decision Tree SMOTE (solo importantes)

#Quitar columnas con menos importancia
```

```
cols_del_imp = ['CompromisoTrabajo','Campo_Educativo_Recursos Humanos','Campo_Educativo_Otra','Campo_Educativo_Mercadeo',
                'Campo_Educativo_Medicina','Rol_Director de fabricacion','Area_Recursos Humanos',
                'Area_Investigacion & Desarrollo','Rol_Director de investigacion','Frecuencia_viaje_Frecuente',
                'Frecuencia_viaje_Esporadico','Porcentaje_Ausentismo','Rol_Ejecutivo de ventas','Rendimiento',
                'Campo_Educativo_Tecnico','Rol_Cientifico investigador','Rol_Recursos Humanos','Satisfaccion_Entorno',
                'Anos_Ultima_Promocion','Rol_Representante de salud','Rol_Representante de ventas','Nivel_Opcion_Acciones',
                'Porcentaje_Aumento_Salarial','Nivel_Rol','Nivel_Educativo','Distancia_Casa','Rol_Gerente',
                'Rol_Tecnico de laboratorio','Campo_Educativo_Biologia']

X_train_sm_imp = X_train_sm.loc[:, [name for name in X_train_sm.columns if name not in cols_del_imp]]
X_test_sm_imp = X_test_sm.loc[:, [name for name in X_test_sm.columns if name not in cols_del_imp]]
```

In [40]:
```
#Entrenar de nuevo Arbol de Decision individual SMOTE
dectree_model_sm_imp = DecisionTreeClassifier(max_depth= 5, random_state = 1)
dectree_model_sm_imp.fit(X_train_sm_imp,y_train_sm)
```

Out[40]:
```
DecisionTreeClassifier(max_depth=5, random_state=1)
```

In [42]:
```
#Prediccion sobre el test
pred_y_dectree_sm_imp = dectree_model_sm_imp.predict(X_test_sm_imp)
mostrar_resultados(y_test_sm, pred_y_dectree_sm_imp)
#Recall de attrition de 75%, activos en 85%

#Prediccion sobre el train
pred_y_dectree_sm_train_imp = dectree_model_sm_imp.predict(X_train_sm_imp)
```



	precision	recall	f1-score	support
0	0.78	0.85	0.81	904
1	0.83	0.75	0.79	899
accuracy			0.80	1803
macro avg	0.81	0.80	0.80	1803
weighted avg	0.81	0.80	0.80	1803

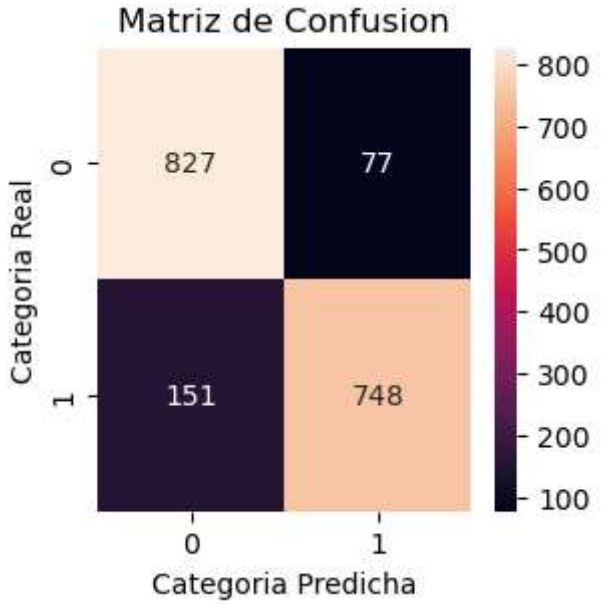
In [43]:
```
metricas_comparacion(y_test_sm, pred_y_dectree_sm_imp, y_train_sm, pred_y_dectree_sm_train_imp)
```

```
Accuracy:  0.8031059345535219
AUC test:  0.802970606475238
AUC train:  0.8032661112347047
Delta AUC (overfitting):  0.00029550475946671373
```

Entrenamiento SMOTE Random Forest

In [22]:
```
#Random Forest con SMOTE
rf_model_sm =RandomForestClassifier(max_depth=5,random_state=1, class_weight = 'balanced')
rf_model_sm.fit(X_train_sm,y_train_sm)

#Prediccion sobre test
y_pred_rf_sm = rf_model_sm.predict(X_test_sm)
```

In [23]:
```
mostrar_resultados(y_test_sm, y_pred_rf_sm)
```



	precision	recall	f1-score	support
0	0.85	0.91	0.88	904
1	0.91	0.83	0.87	899
accuracy			0.87	1803
macro avg	0.88	0.87	0.87	1803
weighted avg	0.88	0.87	0.87	1803

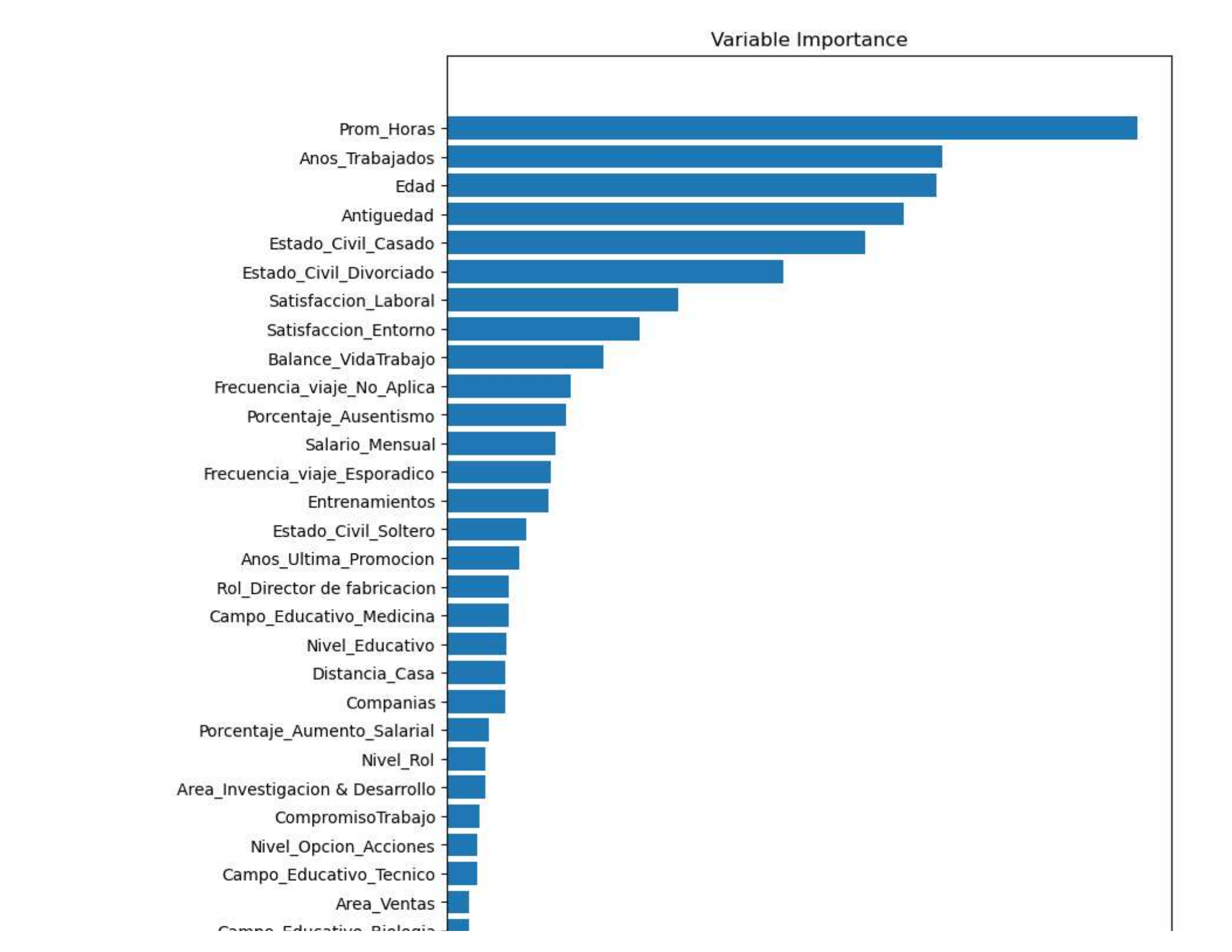
In [24]:
```
#Prediccion sobre el train
y_pred_rf_sm_train = rf_model_sm.predict(X_train_sm)
```

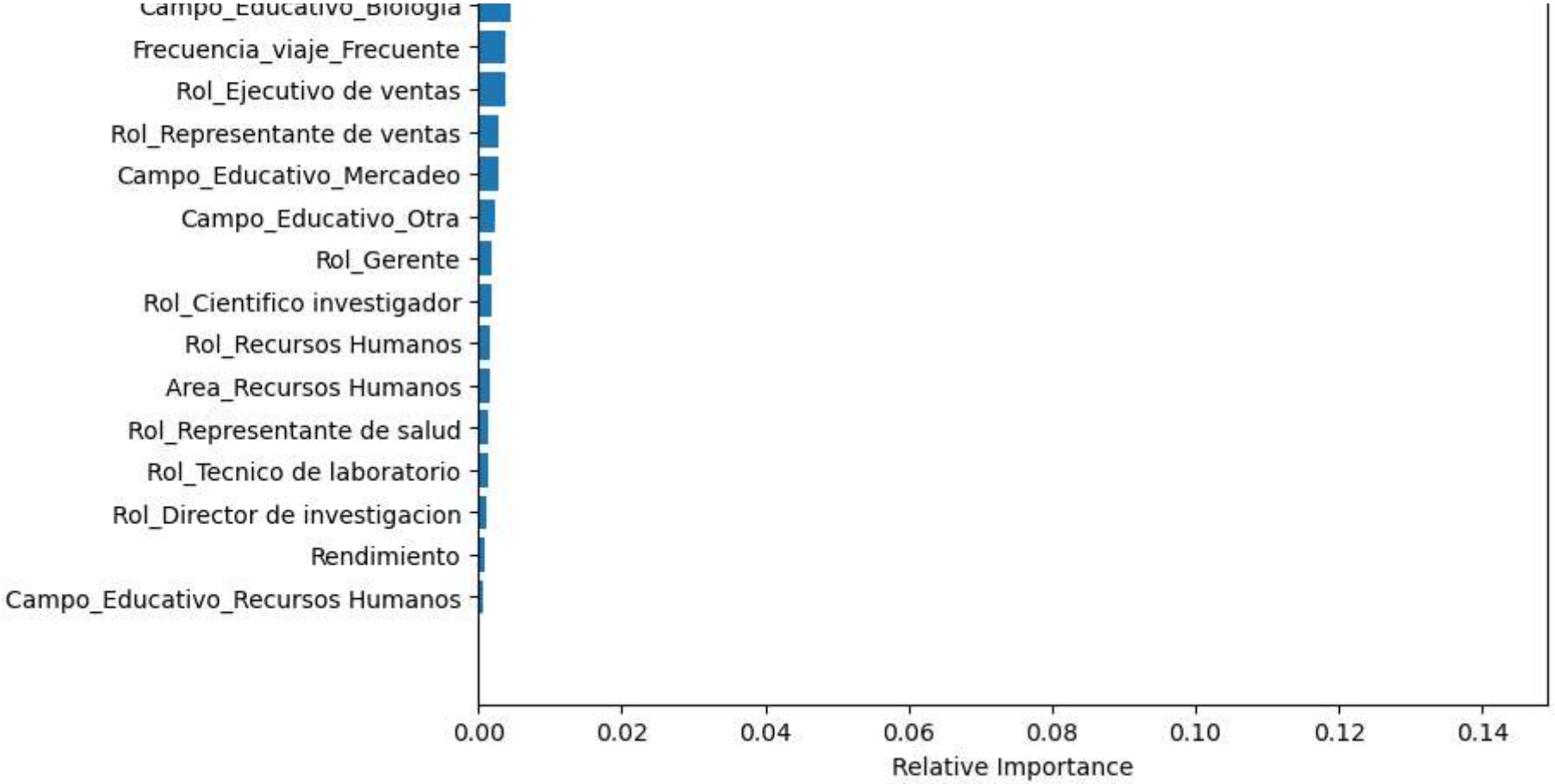
In [25]:
```
metricas_comparacion(y_test_sm, y_pred_rf_sm, y_train_sm, y_pred_rf_sm_train)
```

```
Accuracy:  0.8735440931780366
AUC test:  0.8734293019776151
AUC train:  0.8752039634933052
Delta AUC (overfitting):  0.001774661515690057
```

In [26]:
```
#Importancia variables
feature_importance = rf_model_sm.feature_importances_
#Importancia relativa a la importancia maxima
feature_importance = 100.0 * (feature_importance / 100)
                              #feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
```

```
plt.figure(figsize=(8, 15))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train_ada.keys()[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```


Variable Importance
Prom_Horas
Anos_Trabajados
Edad
Antiguedad
Estado_Civil_Casado
Estado_Civil_Divorciado
Satisfaccion_Laboral
Satisfaccion_Entorno
Balance_VidaTrabajo
Frecuencia_viaje_No_Aplica
Porcentaje_Ausentismo
Salario_Mensual
Frecuencia_viaje_Esporadico
Entrenamientos
Estado_Civil_Soltero
Anos_Ultima_Promocion
Rol_Director de fabricacion
Campo_Educativo_Medicina
Nivel_Educativo
Distancia_Casa
Companias
Porcentaje_Aumento_Salarial
Nivel_Rol
Area_Investigacion & Desarrollo
CompromisoTrabajo
Nivel_Opcion_Acciones
Campo_Educativo_Tecnico
Area_Ventas

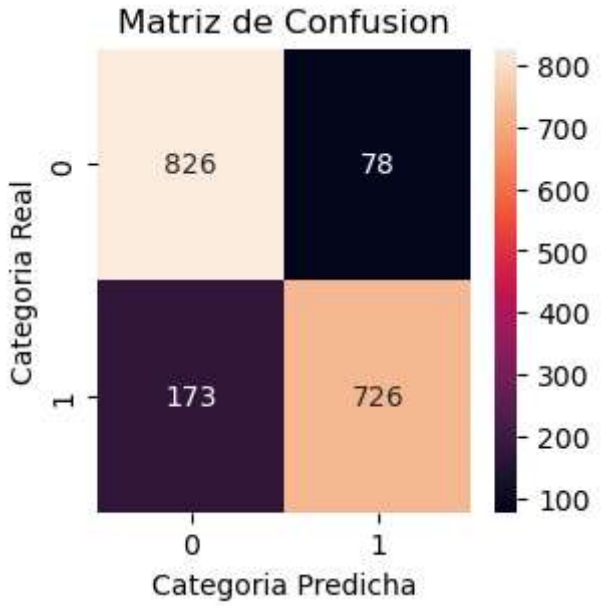


In [27]:

```
#Modelo de Random Forecast SMOTE (solo importantes)
# Segun la grafica las varaibles de rol, campo educativo, compromiso trabajo, nivel de acciones,
# rendimiento no son importantes, se deja frecuencia de viaje frecuente porque las demas estan arriba
# se quitan todos los roles y areas para ser consistentes

#Quitar columnas con menos importancia
cols_del_imp = ['Area_Investigacion & Desarrollo','CompromisoTrabajo','Nivel_Opcion_Acciones','Campo_Educativo_Tecnico',
                'Area_Ventas','Campo_Educativo_Biologia','Rol_Ejecutivo de ventas','Rol_Representante de ventas',
                'Campo_Educativo_Mercadeo','Campo_Educativo_Otra','Rol_Gerente','Rol_Cientifico investigador',
                'Rol_Recursos Humanos','Area_Recursos Humanos','Rol_Representante de salud','Rol_Tecnico de laboratorio',
                'Rol_Director de investigacion','Rendimiento','Campo_Educativo_Recursos Humanos',
                'Rol_Director de fabricacion','Campo_Educativo_Medicina']

X_train_sm_imp = X_train_sm.loc[:, [name for name in X_train_sm.columns if name not in cols_del_imp]]
X_test_sm_imp = X_test_sm.loc[:, [name for name in X_test_sm.columns if name not in cols_del_imp]]
```

In [28]:
```
#Entrenar modelo de nuevo
rf_model_sm_imp =RandomForestClassifier(max_depth=5,random_state=1, class_weight = 'balanced')
rf_model_sm_imp.fit(X_train_sm_imp,y_train_sm)

#Prediccion sobre test
y_pred_rf_sm_imp = rf_model_sm_imp.predict(X_test_sm_imp)
```

In [29]:
```
mostrar_resultados(y_test_sm, y_pred_rf_sm_imp)
```



	precision	recall	f1-score	support
0	0.83	0.91	0.87	904
1	0.90	0.81	0.85	899
accuracy			0.86	1803
macro avg	0.86	0.86	0.86	1803
weighted avg	0.86	0.86	0.86	1803

In [30]:
```
#Prediccion sobre el train
y_pred_rf_sm_train_imp = rf_model_sm_imp.predict(X_train_sm_imp)
```

In [31]:
```
metricas_comparacion(y_test_sm, y_pred_rf_sm_imp, y_train_sm, y_pred_rf_sm_train_imp)
```

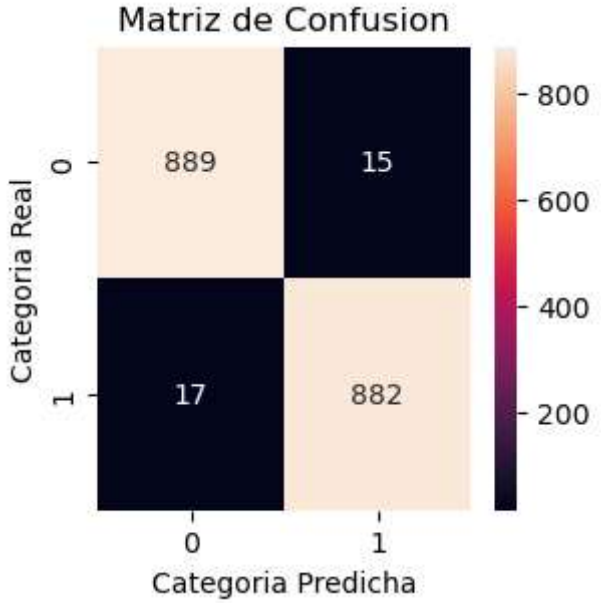
```
Accuracy:  0.8607875762617859
AUC test:  0.8606403870573991
AUC train:  0.8581964934287019
Delta AUC (overfitting):  0.0024438936286972357
```

Entrenamiento SMOTE Gradient Boosting

In [26]:
```
#Gradient Boosting individual
gbm_model_sm = GradientBoostingClassifier(max_depth = 5,max_features = 'auto',random_state = 1)
gbm_model_sm.fit(X_train_sm,y_train_sm)
```

Out[26]:
```
GradientBoostingClassifier(max_depth=5, max_features='auto', random_state=1)
```

In [27]:
```
#Prediccion sobre el test
pred_y_gbm_sm = gbm_model_sm.predict(X_test_sm)
mostrar_resultados(y_test_sm, pred_y_gbm_sm)
#Recall de attrition de 98%, activos en 98%

#Prediccion sobre el train
pred_y_gbm_sm_train = gbm_model_sm.predict(X_train_sm)
```

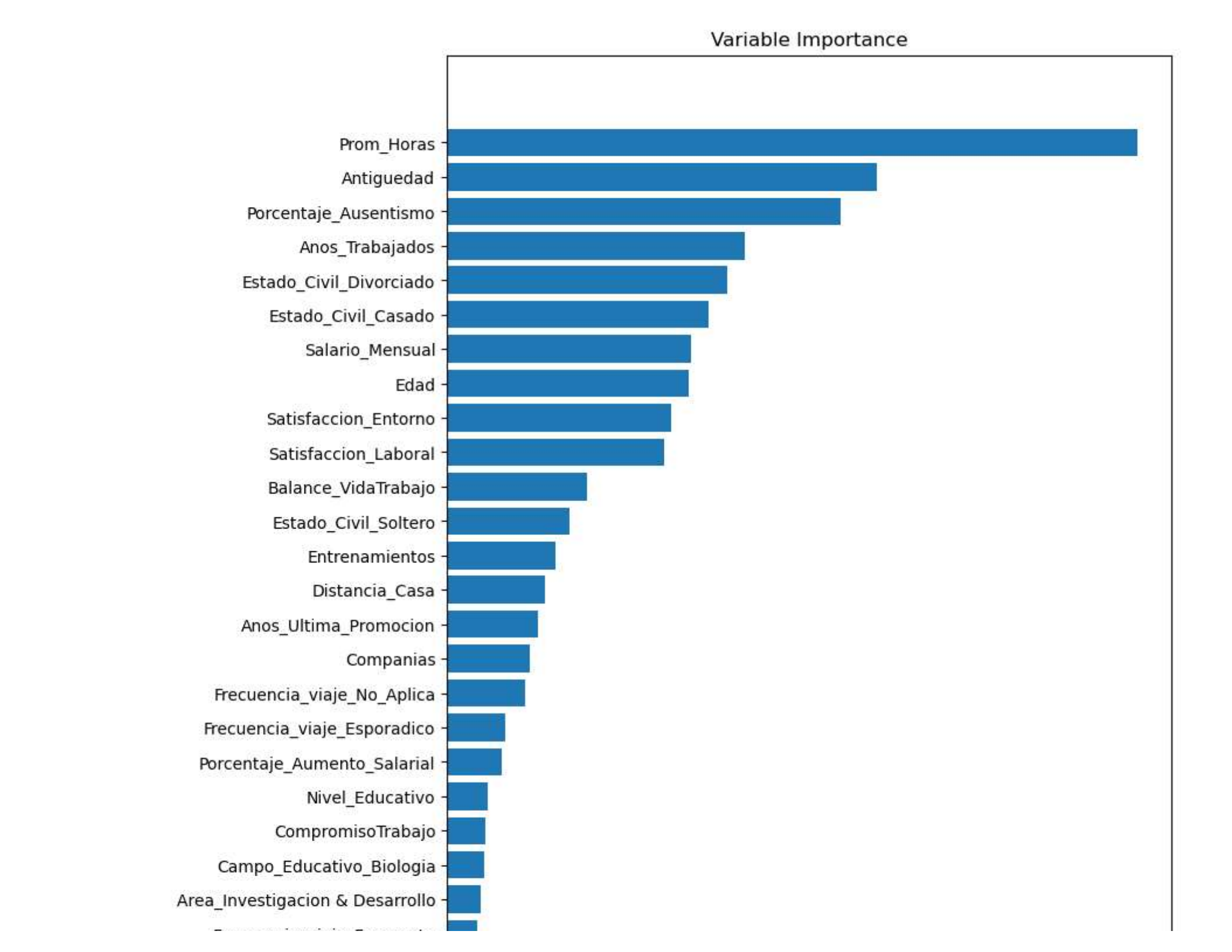


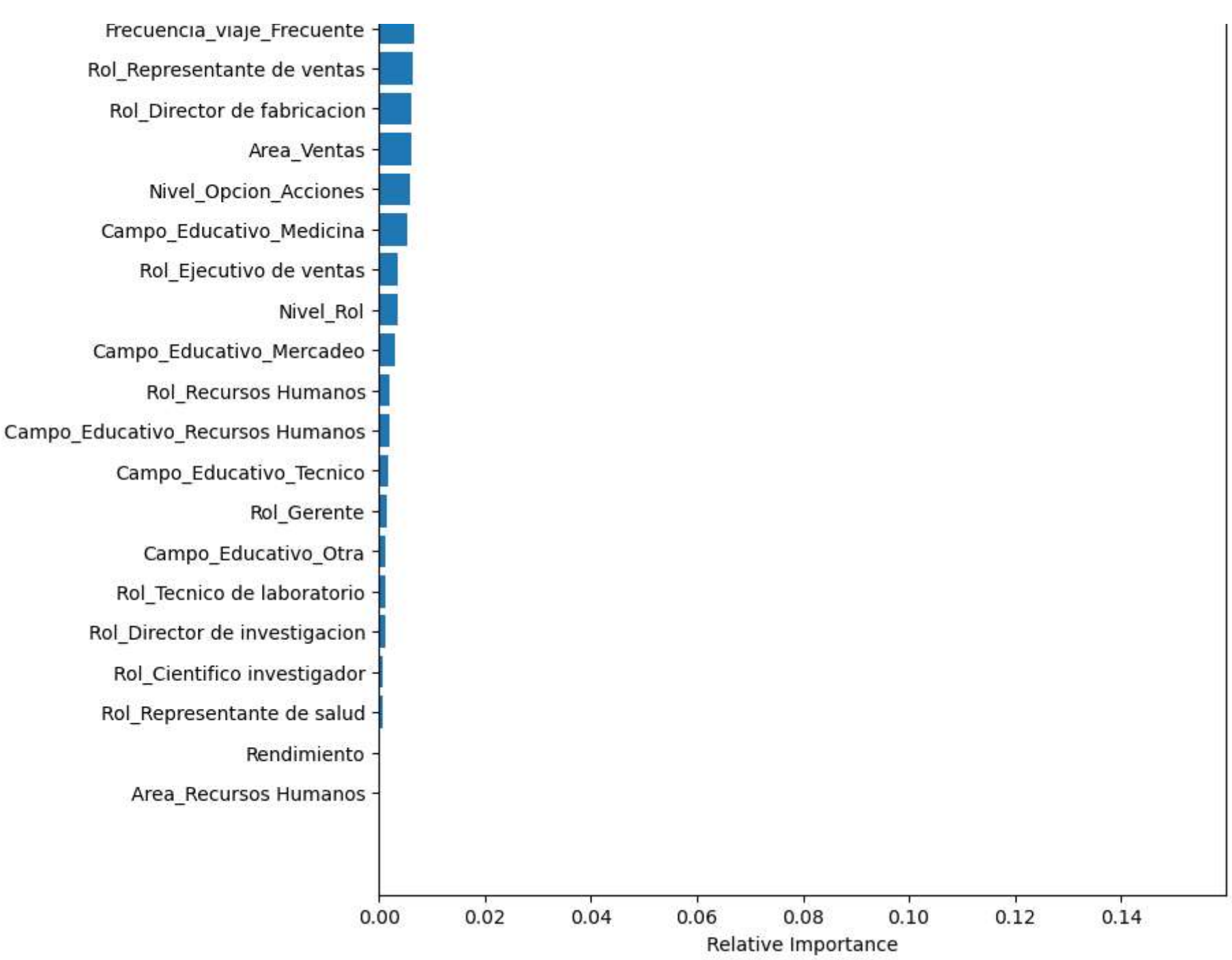
```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       904
           1       0.98      0.98      0.98       899

    accuracy                           0.98      1803
   macro avg       0.98      0.98      0.98      1803
weighted avg       0.98      0.98      0.98      1803
```

In [28]:
```
metricas_comparacion(y_test_sm, pred_y_gbm_sm, y_train_sm, pred_y_gbm_sm_train)
```

```
Accuracy:  0.9822518025513034
AUC test:  0.9822485898786262
AUC train:  0.9972273552293599
Delta AUC (overfitting):  0.014978765350733725
```

In [29]:
```
feature_importance = gbm_model_sm.feature_importances_
#Importancia relativa a la importancia maxima
feature_importance = 100.0 * (feature_importance / 100)
                              #feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5

plt.figure(figsize=(8, 18))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train_sm.keys()[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```


Variable Importance
Prom_Horas
Antiguedad
Porcentaje_Ausentismo
Anos_Trabajados
Estado_Civil_Divorciado
Estado_Civil_Casado
Salario_Mensual
Edad
Satisfaccion_Entorno
Satisfaccion_Laboral
Balance_VidaTrabajo
Estado_Civil_Soltero
Entrenamientos
Distancia_Casa
Anos_Ultima_Promocion
Companias
Frecuencia_viaje_No_Aplica
Frecuencia_viaje_Esporadico
Porcentaje_Aumento_Salarial
Nivel_Educativo
CompromisoTrabajo
Campo_Educativo_Biologia
Area_Investigacion & Desarrollo



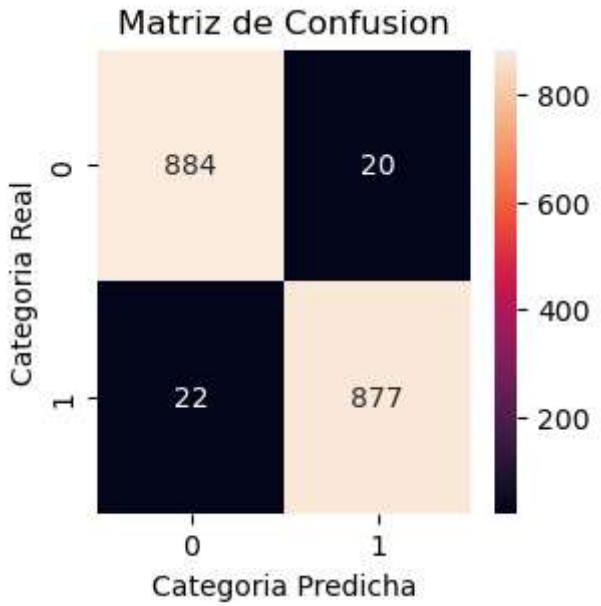
In [30]:
```
#Modelo de Gradient Boosting SMOTE (solo importantes)

#Quitar columnas con menos importancia
```

```
cols_del_imp = ['Area_Recursos Humanos','Rendimiento','Rol_Representante de salud',
                'Rol_Cientifico investigador','Rol_Director de investigacion','Rol_Tecnico de laboratorio',
               'Campo_Educativo_Otra','Rol_Gerente','Campo_Educativo_Tecnico','Campo_Educativo_Recursos Humanos',
               'Rol_Recursos Humanos','Campo_Educativo_Mercadeo','Nivel_Rol','Rol_Ejecutivo de ventas',
               'Campo_Educativo_Medicina','Nivel_Opcion_Acciones','Area_Ventas','Rol_Director de fabricacion',
               'Rol_Representante de ventas','Frecuencia_viaje_Frecuente','Area_Investigacion & Desarrollo',
               'Campo_Educativo_Biologia']

X_train_sm_imp = X_train_sm.loc[:, [name for name in X_train_sm.columns if name not in cols_del_imp]]
X_test_sm_imp = X_test_sm.loc[:, [name for name in X_test_sm.columns if name not in cols_del_imp]]
```

In [31]:
```
#Entrenar de nuevo
gbm_model_sm_imp = GradientBoostingClassifier(max_depth = 5,max_features = 'auto',random_state = 1)
gbm_model_sm_imp.fit(X_train_sm_imp,y_train_sm)
```

Out[31]:
```
GradientBoostingClassifier(max_depth=5, max_features='auto', random_state=1)
```

In [32]:
```
#Prediccion sobre el test
pred_y_gbm_sm_imp = gbm_model_sm_imp.predict(X_test_sm_imp)
mostrar_resultados(y_test_sm, pred_y_gbm_sm_imp)
#Recall de attrition de 98%, activos en 98%

#Prediccion sobre el train
pred_y_gbm_sm_train_imp = gbm_model_sm_imp.predict(X_train_sm_imp)
```



```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       904
           1       0.98      0.98      0.98       899

    accuracy                           0.98      1803
   macro avg       0.98      0.98      0.98      1803
weighted avg       0.98      0.98      0.98      1803
```

In [34]:
```
metricas_comparacion(y_test_sm, pred_y_gbm_sm_imp, y_train_sm, pred_y_gbm_sm_train_imp)
```

```
Accuracy:  0.9767054908485857
AUC test:  0.9767022355222617
AUC train:  0.9950062977961408
Delta AUC (overfitting):  0.01830406227387915
```

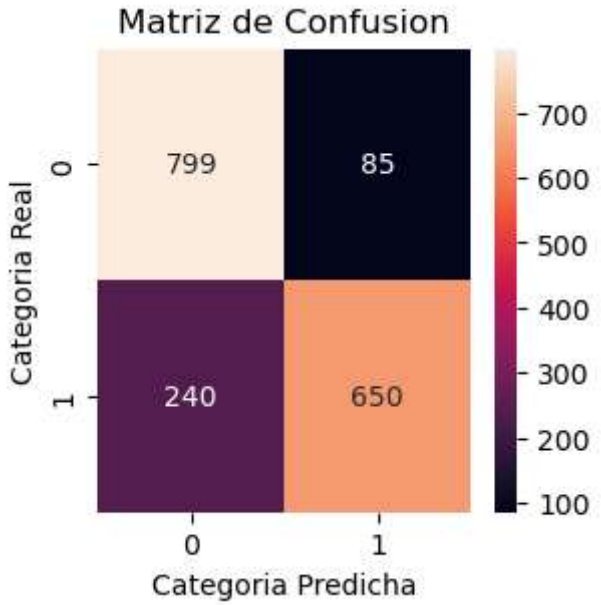
In [62]:
```
#feature_importance = gbm_model_sm_imp.feature_importances_
##Importancia relativa a la importancia maxima
#feature_importance = 100.0 * (feature_importance / 100)
#                             #feature_importance.max())
#sorted_idx = np.argsort(feature_importance)
#pos = np.arange(sorted_idx.shape[0]) + .5
#
#plt.figure(figsize=(8, 18))
#plt.barh(pos, feature_importance[sorted_idx], align='center')
#plt.yticks(pos, X_train_sm_imp.keys()[sorted_idx])
#plt.xlabel('Relative Importance')
#plt.title('Variable Importance')
#plt.show()
```

Entrenamiento ADASYN Logistic Regression

In [44]:
```
#Modelo de Regresion Logistica ADASYN
logreg_model_ada = LogisticRegression(C=1.0,penalty='l2',random_state=1,max_iter=10000)
logreg_model_ada.fit(X_train_ada, y_train_ada)
```

Out[44]:
```
LogisticRegression(max_iter=10000, random_state=1)
```

In [45]:
```
#Prediccion sobre el test
pred_y_log_ada = logreg_model_ada.predict(X_test_ada)
mostrar_resultados(y_test_ada, pred_y_log_ada)
#Recall de attrition de 73%, activos en 90%

#Prediccion sobre el train
pred_y_log_ada_train = logreg_model_ada.predict(X_train_ada)
```



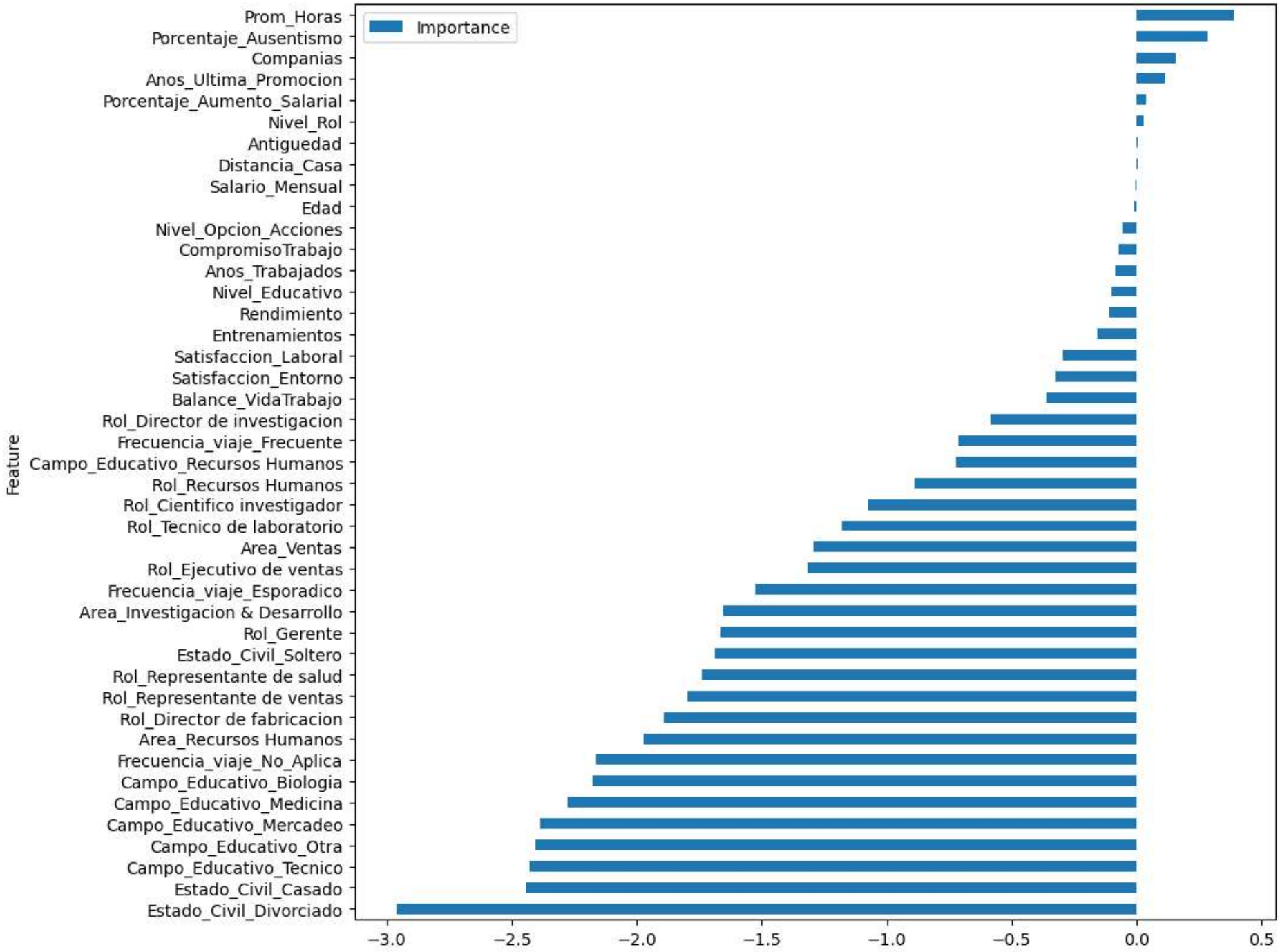
	precision	recall	f1-score	support
0	0.77	0.90	0.83	884
1	0.88	0.73	0.80	890
accuracy			0.82	1774
macro avg	0.83	0.82	0.82	1774
weighted avg	0.83	0.82	0.82	1774

In [46]:
```
metricas_comparacion(y_test_ada, pred_y_log_ada, y_train_ada, pred_y_log_ada_train)
```

```
Accuracy:  0.8167981961668546
AUC test:  0.8170916162489197
AUC train:  0.8267456711374261
Delta AUC (overfitting):  0.009654054888506458
```

In [47]:
```
#Importancia de las variables
coefficients_log_ada = logreg_model_ada.coef_[0]

ft_importance_log_ada = pd.DataFrame({'Feature': X_ada.columns, 'Importance': coefficients_log_ada})
ft_importance_log_ada = ft_importance_log_ada.sort_values('Importance', ascending=True)
ft_importance_log_ada.plot(x='Feature', y='Importance', kind='barh', figsize=(10, 10))
```

Out[47]:
```
<AxesSubplot:ylabel='Feature'>
```



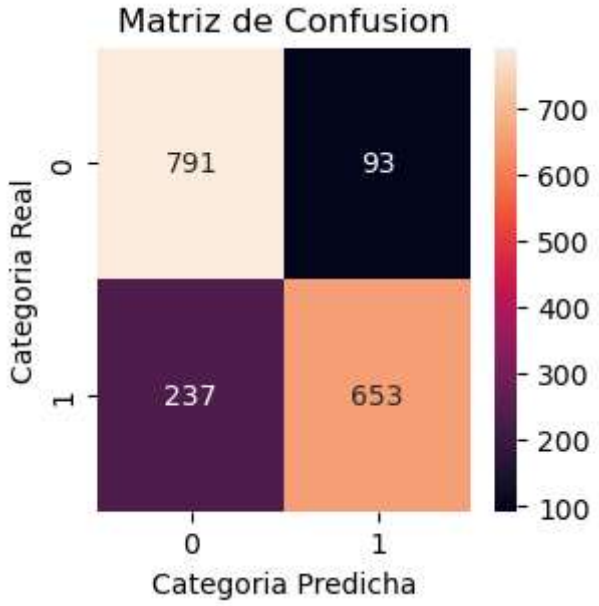
In [48]:
```
#Modelo de Regresion Logistica ADASYN (solo importantes)

#Quitar columnas con menos importancia
```

```
cols_del_imp = ['Porcentaje_Aumento_Salarial','Nivel_Rol','Antiguedad','Distancia_Casa','Salario_Mensual','Edad']

X_train_ada_imp = X_train_ada.loc[:, [name for name in X_train_ada.columns if name not in cols_del_imp]]
X_test_ada_imp = X_test_ada.loc[:, [name for name in X_test_ada.columns if name not in cols_del_imp]]
```

In [49]:
```
#Entrenar de nuevo
logreg_model_ada_imp = LogisticRegression(C=1.0,penalty='l2',random_state=1,max_iter=10000)
logreg_model_ada_imp.fit(X_train_ada_imp, y_train_ada)
```

Out[49]:
```
LogisticRegression(max_iter=10000, random_state=1)
```

In [50]:
```
#Prediccion sobre el test
pred_y_log_ada_imp = logreg_model_ada_imp.predict(X_test_ada_imp)
mostrar_resultados(y_test_ada, pred_y_log_ada_imp)
#Recall de attrition de 77%, activos en 89%

#Prediccion sobre el train
pred_y_log_ada_train_imp = logreg_model_ada_imp.predict(X_train_ada_imp)
```



```
              precision    recall  f1-score   support

           0       0.77      0.89      0.83       884
           1       0.88      0.73      0.80       890

    accuracy                           0.81      1774
   macro avg       0.82      0.81      0.81      1774
weighted avg       0.82      0.81      0.81      1774
```

In [51]:
```
metricas_comparacion(y_test_ada, pred_y_log_ada_imp, y_train_ada, pred_y_log_ada_train_imp)
```

```
Accuracy:  0.8139797068771139
AUC test:  0.8142521226295186
AUC train:  0.8253010567134088
Delta AUC (overfitting):  0.011048934083890183
```

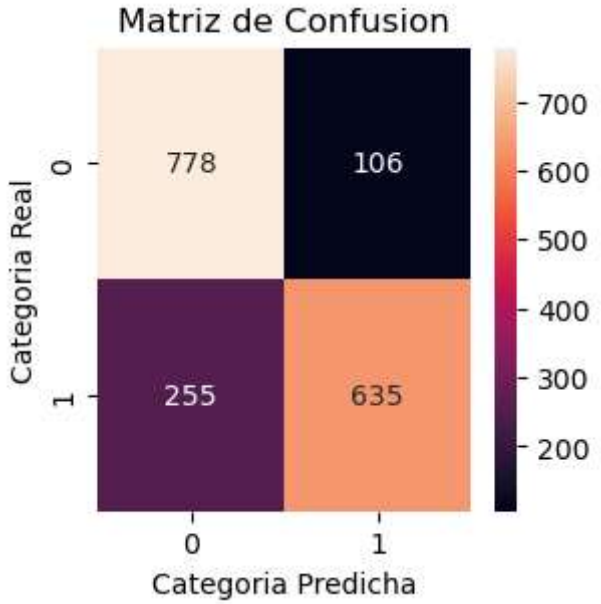
In [39]:
```
#Importancia de las variables
#coefficients_log_ada_imp = logreg_model_ada_imp.coef_[0]

#ft_importance_log_ada_imp = pd.DataFrame({'Feature': X_train_ada_imp.columns, 'Importance': coefficients_log_ada_imp})
#ft_importance_log_ada_imp = ft_importance_log_ada_imp.sort_values('Importance', ascending=True)
#ft_importance_log_ada_imp.plot(x='Feature', y='Importance', kind='barh', figsize=(10, 10))
```

Entrenamiento ADASYN Decision Tree

In [41]:
```
#Arbol de Decision individual ADASYN
dectree_model_ada = DecisionTreeClassifier(max_depth= 5, random_state = 1)
dectree_model_ada.fit(X_train_ada,y_train_ada)
```

Out[41]:
```
DecisionTreeClassifier(max_depth=5, random_state=1)
```

In [42]:
```
#Prediccion sobre el test
pred_y_dectree_ada = dectree_model_ada.predict(X_test_ada)
mostrar_resultados(y_test_ada, pred_y_dectree_ada)
#Recall de attrition de 71%, activos en 88%

#Prediccion sobre el train
pred_y_dectree_ada_train = dectree_model_ada.predict(X_train_ada)
```



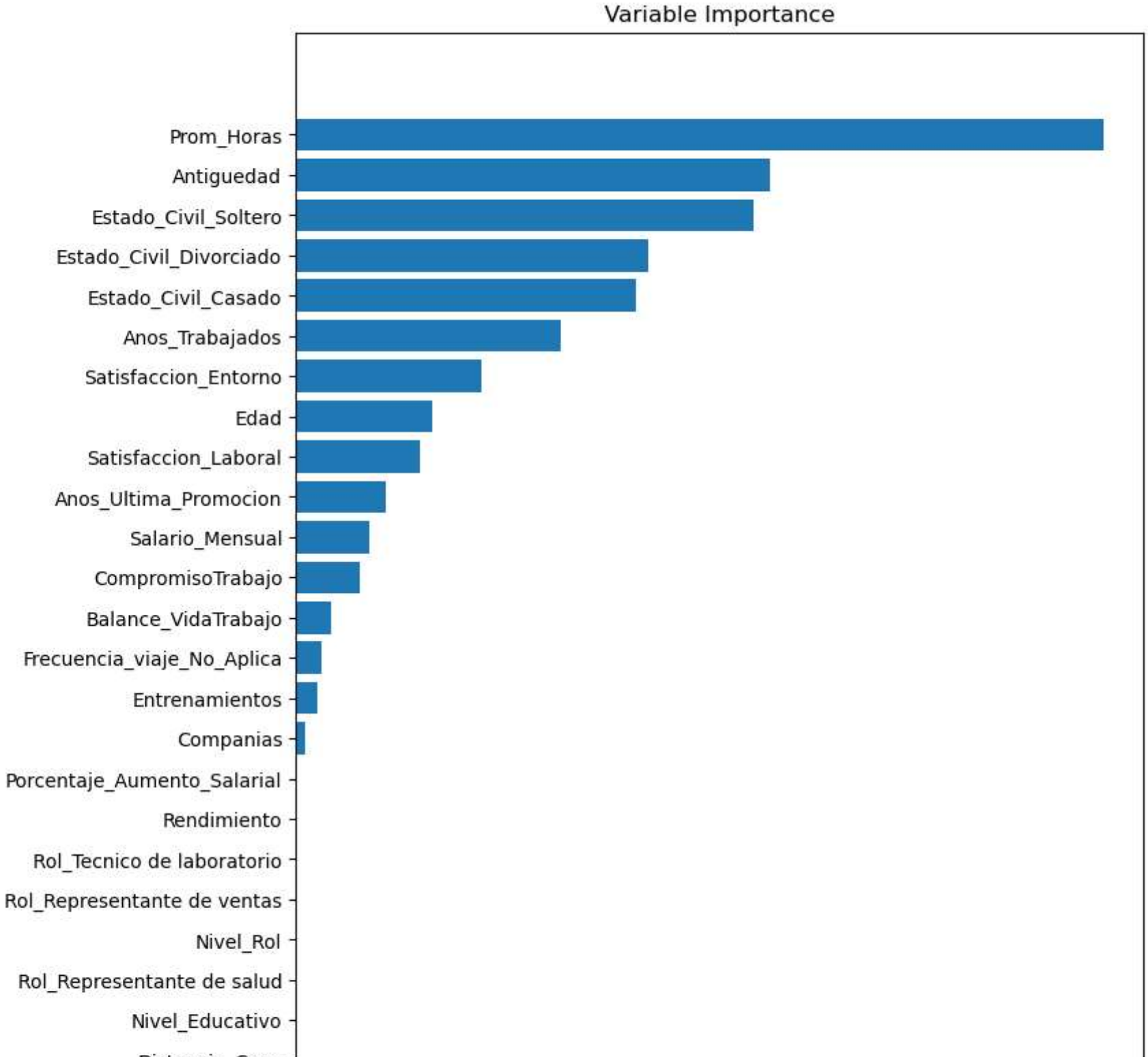
	precision	recall	f1-score	support
0	0.75	0.88	0.81	884
1	0.86	0.71	0.78	890
accuracy			0.80	1774
macro avg	0.81	0.80	0.80	1774
weighted avg	0.81	0.80	0.80	1774

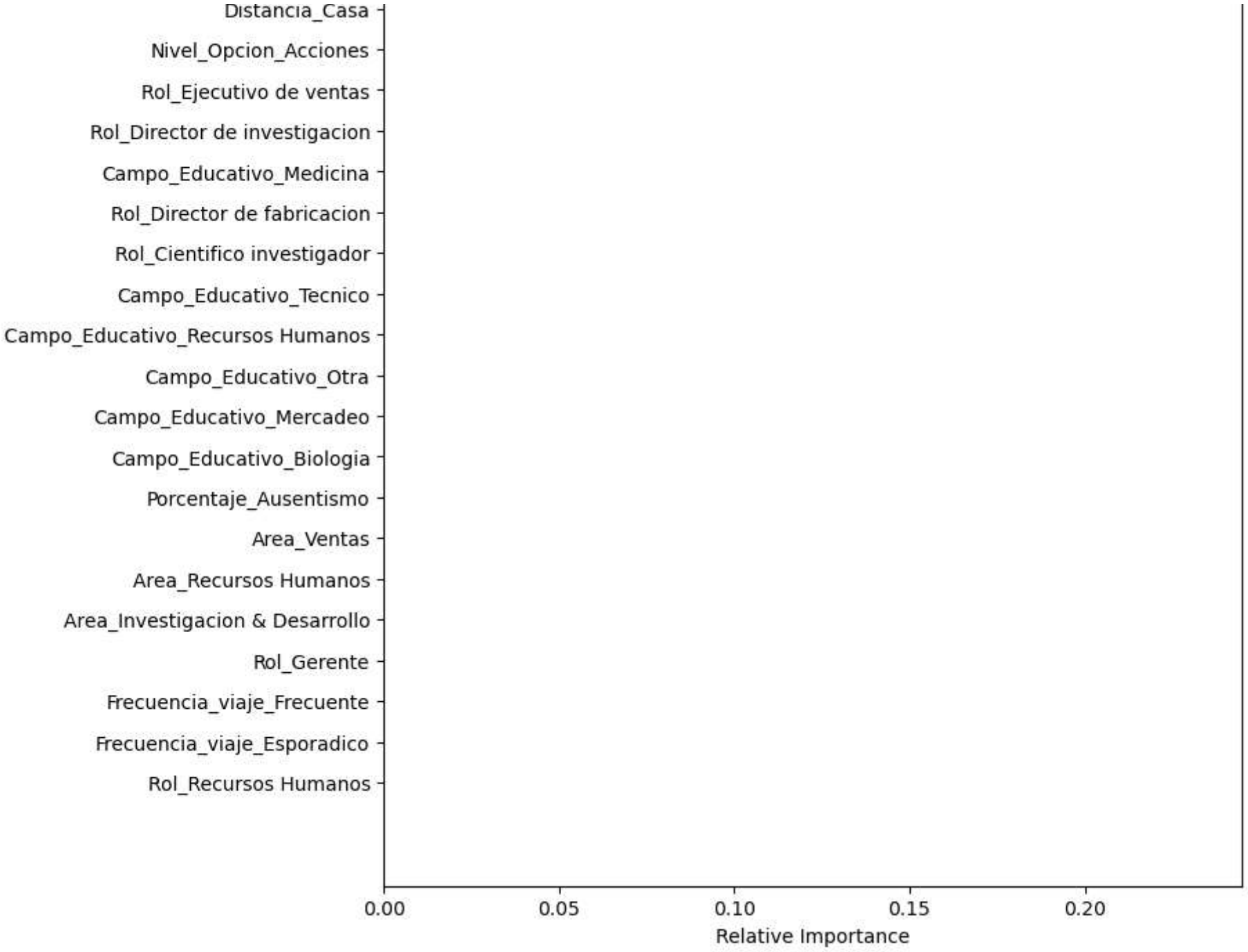
In [43]:
```
metricas_comparacion(y_test_ada, pred_y_dectree_ada, y_train_ada, pred_y_dectree_ada_train)
```

```
Accuracy:  0.7965050732807215
AUC test:  0.7967868219024861
AUC train:  0.8051023120189122
Delta AUC (overfitting):  0.008315490116426183
```

In [44]:
```
#Hallar variables importantes
feature_importance = dectree_model_ada.feature_importances_
#Importancia relativa a la importancia maxima
feature_importance = 100.0 * (feature_importance / 100)
                             #feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5

plt.figure(figsize=(8, 18))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train_sm.keys()[sorted_idx])
```

```
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```


Variable Importance
Prom_Horas
Antiguedad
Estado_Civil_Soltero
Estado_Civil_Divorciado
Estado_Civil_Casado
Anos_Trabajados
Satisfaccion_Entorno
Edad
Satisfaccion_Laboral
Anos_Ultima_Promocion
Salario_Mensual
CompromisoTrabajo
Balance_VidaTrabajo
Frecuencia_viaje_No_Aplica
Entrenamientos
Companias
Porcentaje_Aumento_Salarial
Rendimiento
Rol_Tecnico de laboratorio
Rol_Representante de ventas
Nivel_Rol
Rol_Representante de salud
Nivel_Educativo



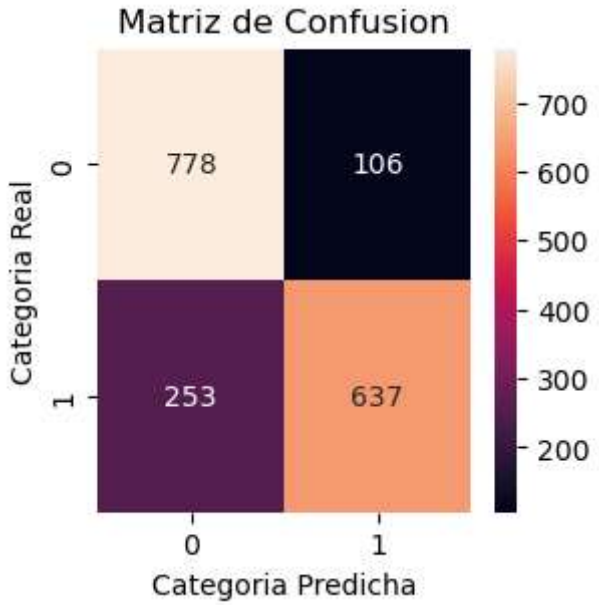
In [45]:
```
#Modelo de Decision Tree ADAYSIN(solo importantes)

#Quitar columnas con menos importancia
```

```
cols_del_imp = [ 'Porcentaje_Aumento_Salarial','Rendimiento','Rol_Tecnico de laboratorio','Rol_Representante de ventas',
                 'Nivel_Rol','Rol_Representante de salud','Nivel_Educativo','Distancia_Casa','Nivel_Opcion_Acciones',
                 'Rol_Ejecutivo de ventas','Rol_Director de investigacion','Campo_Educativo_Medicina',
                 'Rol_Director de fabricacion','Rol_Cientifico investigador','Campo_Educativo_Tecnico',
                 'Campo_Educativo_Recursos Humanos','Campo_Educativo_Otra','Campo_Educativo_Mercadeo',
                 'Campo_Educativo_Biologia','Porcentaje_Ausentismo','Area_Ventas','Area_Recursos Humanos',
                 'Area_Investigacion & Desarrollo','Rol_Gerente','Frecuencia_viaje_Frecuente','Frecuencia_viaje_Esporadico',
                 'Frecuencia_viaje_No_Aplica','Rol_Recursos Humanos']

X_train_sm_imp = X_train_sm.loc[:, [name for name in X_train_sm.columns if name not in cols_del_imp]]
X_test_sm_imp = X_test_sm.loc[:, [name for name in X_test_sm.columns if name not in cols_del_imp]]
```

In [46]:
```
#Arbol de Decision individual ADASYN con variables importantes
dectree_model_ada_imp = DecisionTreeClassifier(max_depth= 5, random_state = 1)
dectree_model_ada_imp.fit(X_train_ada_imp,y_train_ada)
```

Out[46]:
```
DecisionTreeClassifier(max_depth=5, random_state=1)
```

In [47]:
```
#Prediccion sobre el test
pred_y_dectree_ada_imp = dectree_model_ada_imp.predict(X_test_ada_imp)
mostrar_resultados(y_test_ada, pred_y_dectree_ada_imp)
#Recall de attrition de 72%, activos en 88%

#Prediccion sobre el train
pred_y_dectree_ada_train_imp = dectree_model_ada_imp.predict(X_train_ada_imp)
```



```
              precision    recall  f1-score   support

           0       0.75      0.88      0.81       884
           1       0.86      0.72      0.78       890

    accuracy                           0.80      1774
   macro avg       0.81      0.80      0.80      1774
weighted avg       0.81      0.80      0.80      1774
```

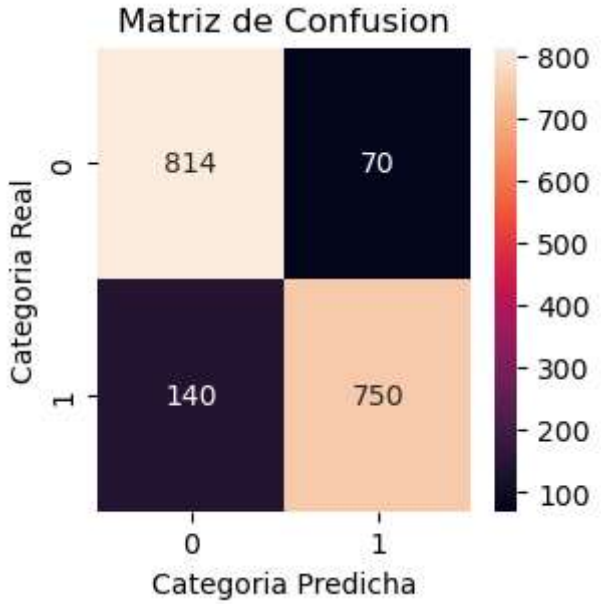
In [48]:
```
metricas_comparacion(y_test_ada, pred_y_dectree_ada_imp, y_train_ada, pred_y_dectree_ada_train_imp)
```

```
Accuracy:  0.7976324689966178
AUC test:  0.7979104174081041
AUC train:  0.8041411432376743
Delta AUC (overfitting):  0.006230725829570183
```

Entrenamiento ADASYN Random Forest

In [32]:
```
#Random Forest con ADASYN
rf_model_ada =RandomForestClassifier(max_depth=5,random_state=1, class_weight = 'balanced')
rf_model_ada.fit(X_train_ada,y_train_ada)

#Prediccion sobre test
y_pred_rf_ada = rf_model_ada.predict(X_test_ada)
```

In [33]:
```
mostrar_resultados(y_test_ada, y_pred_rf_ada)
```



	precision	recall	f1-score	support
0	0.85	0.92	0.89	884
1	0.91	0.84	0.88	890
accuracy			0.88	1774
macro avg	0.88	0.88	0.88	1774
weighted avg	0.88	0.88	0.88	1774

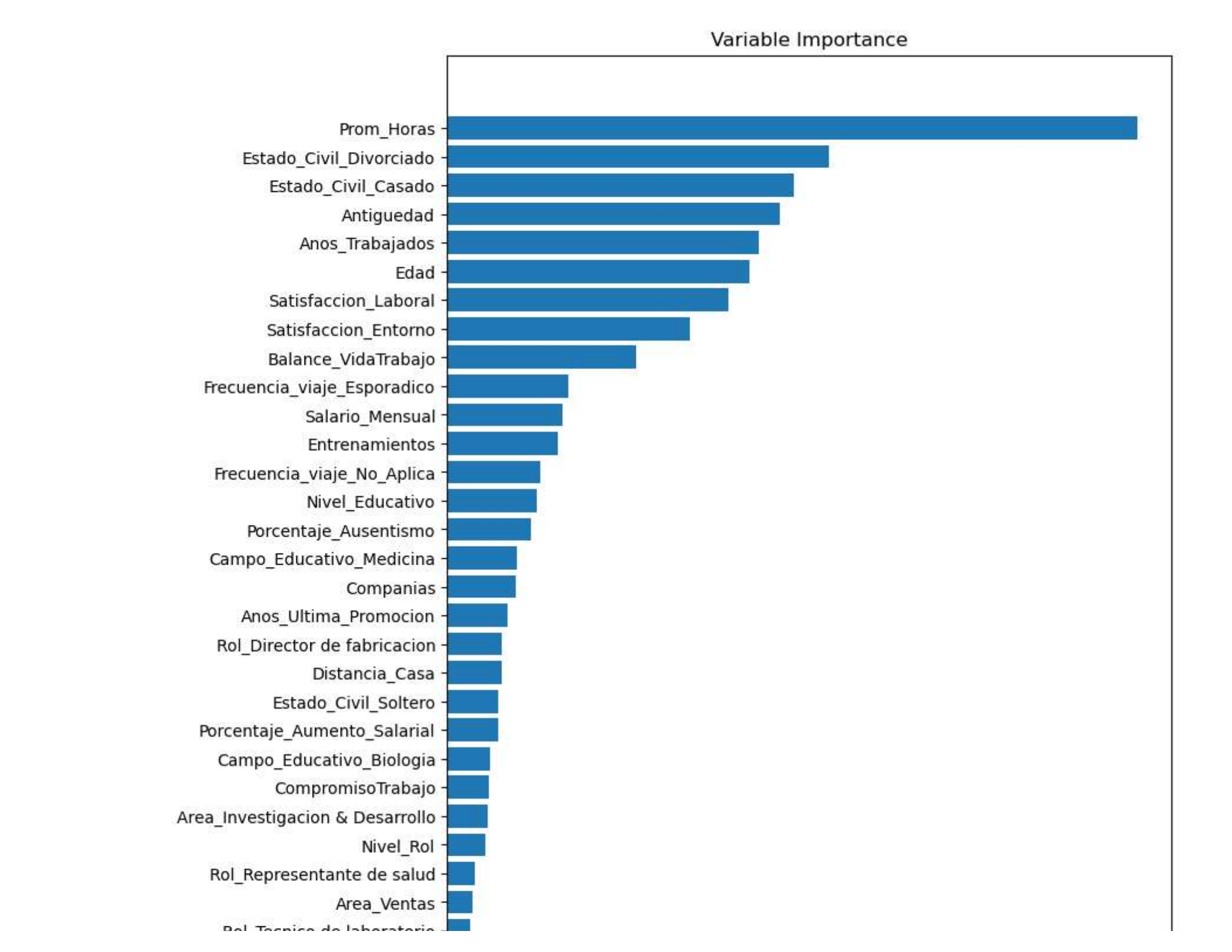
In [34]:
```
#Prediccion sobre el train
y_pred_rf_ada_train = rf_model_ada.predict(X_train_ada)
```

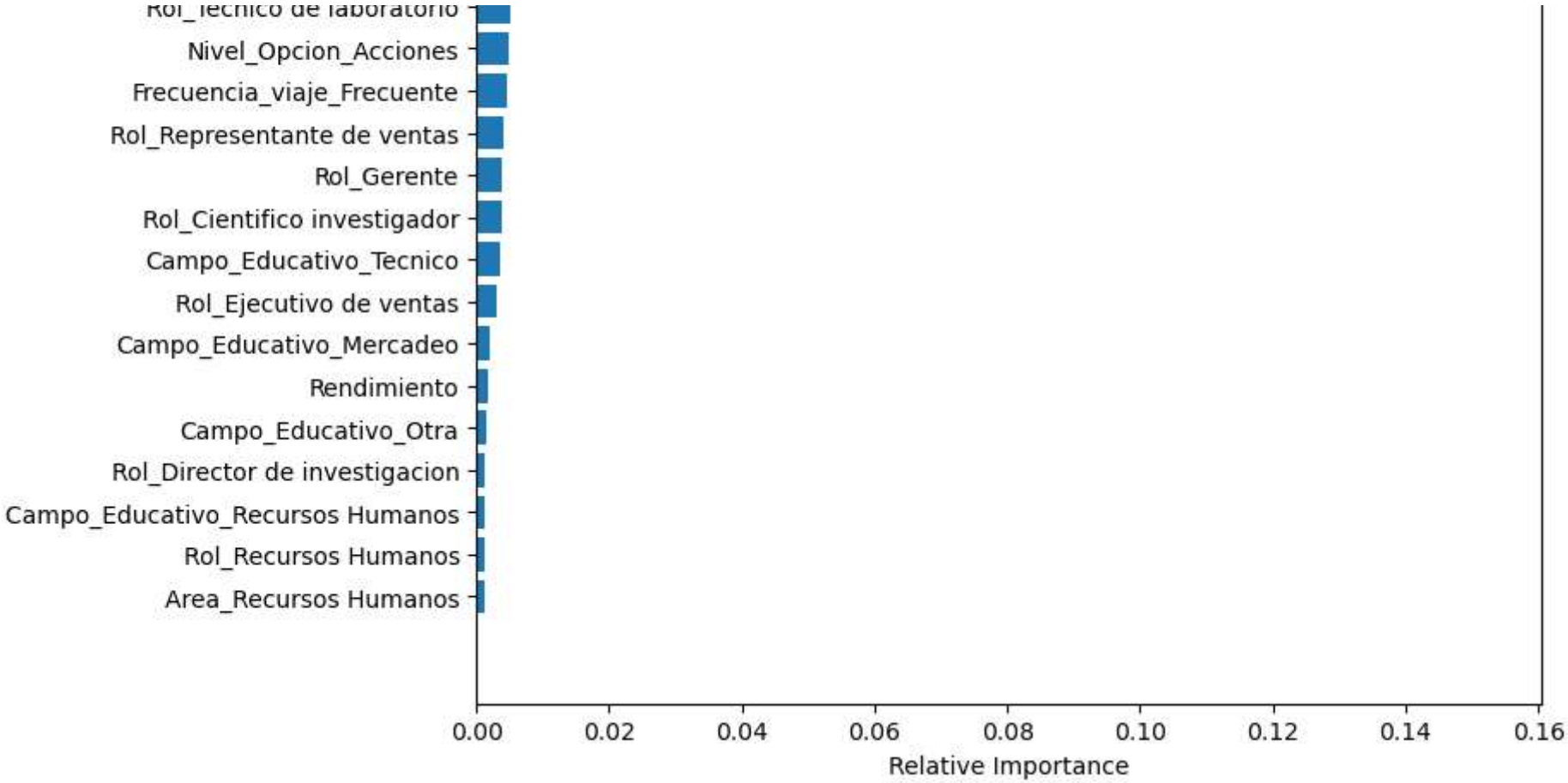
In [35]:
```
metricas_comparacion(y_test_ada, y_pred_rf_ada, y_train_ada, y_pred_rf_ada_train)
```

```
Accuracy:  0.8816234498308907
AUC test:  0.8817555544257463
AUC train:  0.8884471963331888
Delta AUC (overfitting):  0.006691641907442536
```

In [36]:
```
#Importancia variables
feature_importance = rf_model_ada.feature_importances_
#Importancia relativa a la importancia maxima
feature_importance = 100.0 * (feature_importance / 100)
                              #feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
```

```
plt.figure(figsize=(8, 15))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train_ada.keys()[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```


Variable Importance
Prom_Horas
Estado_Civil_Divorciado
Estado_Civil_Casado
Antiguedad
Anos_Trabajados
Edad
Satisfaccion_Laboral
Satisfaccion_Entorno
Balance_VidaTrabajo
Frecuencia_viaje_Esporadico
Salario_Mensual
Entrenamientos
Frecuencia_viaje_No_Aplica
Nivel_Educativo
Porcentaje_Ausentismo
Campo_Educativo_Medicina
Companias
Anos_Ultima_Promocion
Rol_Director de fabricacion
Distancia_Casa
Estado_Civil_Soltero
Porcentaje_Aumento_Salarial
Campo_Educativo_Biologia
CompromisoTrabajo
Area_Investigacion & Desarrollo
Nivel_Rol
Rol_Representante de salud
Area_Ventas

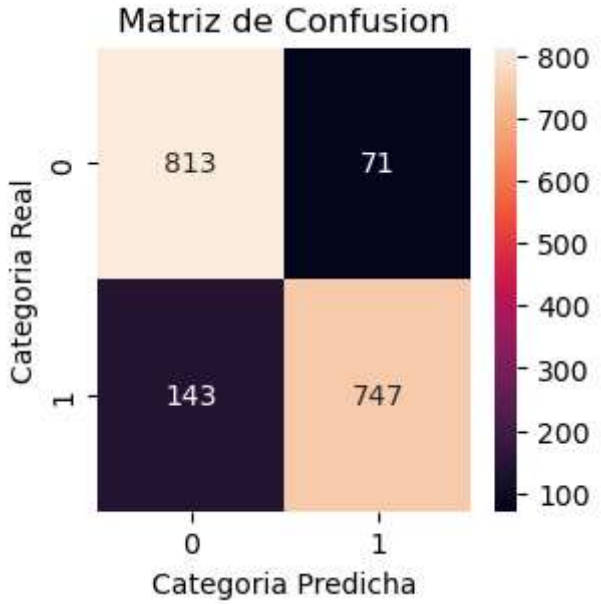


In [37]:

```
#Modelo de Random Forecast ADASYN (solo importantes)
# Segun la grafica las varaibles de rol, campo educativo, compromiso trabajo, nivel de acciones, rendimiento no son importantes
# se deja frecuencia de viaje frecuente porque las demas estan arriba
# se quitan todos los roles y areas para ser consistentes

#Quitar columnas con menos importancia
cols_del_imp = ['Area_Investigacion & Desarrollo','CompromisoTrabajo','Nivel_Opcion_Acciones','Campo_Educativo_Tecnico',
                'Area_Ventas','Campo_Educativo_Biologia','Rol_Ejecutivo de ventas','Rol_Representante de ventas',
                'Campo_Educativo_Mercadeo','Campo_Educativo_Otra','Rol_Gerente','Rol_Cientifico investigador',
                'Rol_Recursos Humanos','Area_Recursos Humanos','Rol_Representante de salud','Rol_Tecnico de laboratorio',
                'Rol_Director de investigacion','Rendimiento','Campo_Educativo_Recursos Humanos',
                'Rol_Director de fabricacion','Campo_Educativo_Medicina']

X_train_ada_imp = X_train_ada.loc[:, [name for name in X_train_ada.columns if name not in cols_del_imp]]
X_test_ada_imp = X_test_ada.loc[:, [name for name in X_test_ada.columns if name not in cols_del_imp]]
```

In [38]:
```
#Entrenar el modelo de nuevo con variables importantes
#Random Forest individual
rf_model_ada_imp =RandomForestClassifier(max_depth=5,random_state=1, class_weight = 'balanced')
rf_model_ada_imp.fit(X_train_ada_imp,y_train_ada)

#Prediccion sobre test
y_pred_rf_ada_imp = rf_model_ada_imp.predict(X_test_ada_imp)
```

In [39]:
```
mostrar_resultados(y_test_ada, y_pred_rf_ada_imp)
```



	precision	recall	f1-score	support
0	0.85	0.92	0.88	884
1	0.91	0.84	0.87	890
accuracy			0.88	1774
macro avg	0.88	0.88	0.88	1774
weighted avg	0.88	0.88	0.88	1774

In [40]:
```
metricas_comparacion(y_test_ada, y_pred_rf_ada_imp, y_train_ada, y_pred_rf_ada_train)
```

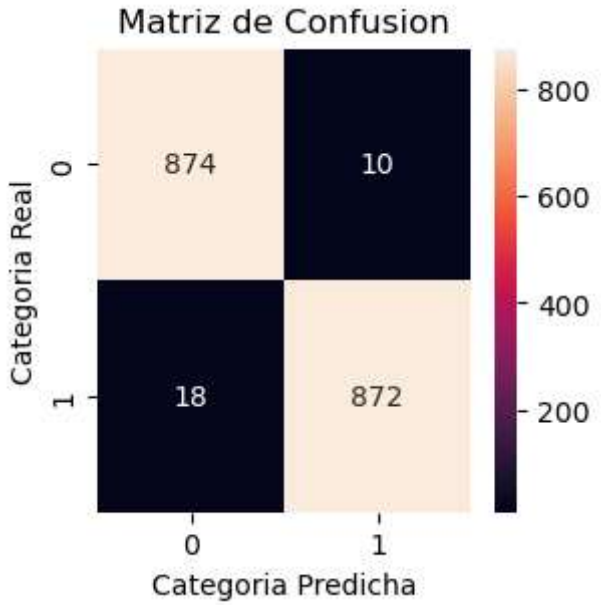
```
Accuracy:  0.8793686583990981
AUC test:  0.8795045503075906
AUC train:  0.8884471963331888
Delta AUC (overfitting):  0.008942646025598244
```

Entrenamiento ADASYN Gradient Boosting

In [52]:
```
#Gradient Boosting individual
gbm_model_ada = GradientBoostingClassifier(max_depth = 5,max_features = 'auto',random_state = 1)
gbm_model_ada.fit(X_train_ada,y_train_ada)
```

Out[52]:
```
GradientBoostingClassifier(max_depth=5, max_features='auto', random_state=1)
```

In [53]:
```
#Prediccion sobre el test
pred_y_gbm_ada = gbm_model_ada.predict(X_test_ada)
mostrar_resultados(y_test_ada, pred_y_gbm_ada)
#Recall de attrition de 98%, activos en 99%

#Prediccion sobre el train
pred_y_gbm_ada_train = gbm_model_ada.predict(X_train_ada)
```



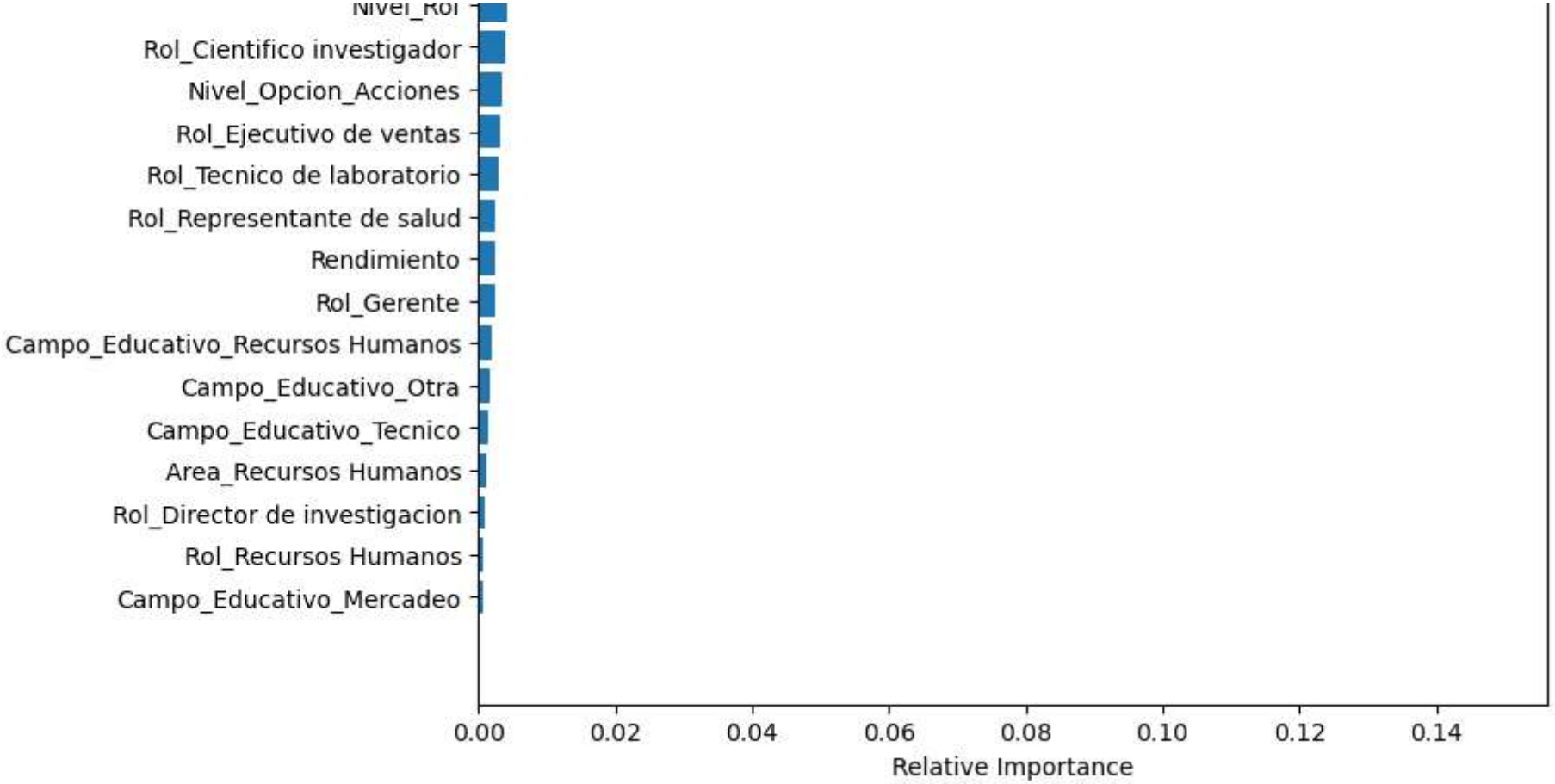
	precision	recall	f1-score	support
0	0.98	0.99	0.98	884
1	0.99	0.98	0.98	890
accuracy			0.98	1774
macro avg	0.98	0.98	0.98	1774
weighted avg	0.98	0.98	0.98	1774

In [54]:
```
metricas_comparacion(y_test_ada, pred_y_gbm_ada, y_train_ada, pred_y_gbm_ada_train)
```

```
Accuracy:  0.9842164599774521
AUC test:  0.9842315318521532
AUC train:  0.9988465974625145
Delta AUC (overfitting):  0.0146150656103613
```

In [60]:
```
feature_importance = gbm_model_ada.feature_importances_
#Importancia relativa a la importancia maxima
feature_importance = 100.0 * (feature_importance / 100)
                              #feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5

plt.figure(figsize=(8, 15))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train_ada.keys()[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```


Variable Importance
Prom_Horas
Porcentaje_Ausentismo
Estado_Civil_Divorciado
Edad
Antiguedad
Anos_Trabajados
Estado_Civil_Casado
Satisfaccion_Laboral
Salario_Mensual
Estado_Civil_Soltero
Satisfaccion_Entorno
Companias
Anos_Ultima_Promocion
Balance_VidaTrabajo
Entrenamientos
Distancia_Casa
Nivel_Educativo
Porcentaje_Aumento_Salarial
Frecuencia_viaje_Esporadico
Frecuencia_viaje_No_Aplica
CompromisoTrabajo
Area_Ventas
Frecuencia_viaje_Frecuente
Campo_Educativo_Biologia
Campo_Educativo_Medicina
Rol_Director de fabricacion
Area_Investigacion & Desarrollo
Rol_Representante de ventas

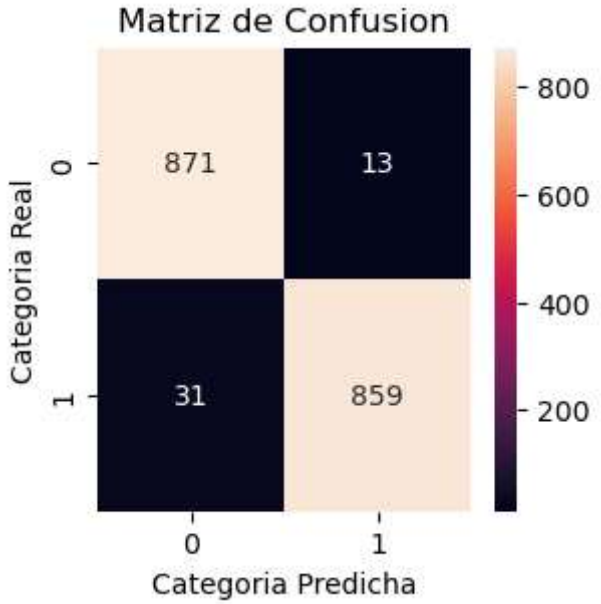


In [56]:
```
#Modelo de Gradient Boosting ADASYN (solo importantes)

#Quitar columnas con menos importancia
cols_del_imp = ['Campo_Educativo_Mercadeo','Rol_Recursos Humanos','Rol_Director de investigacion',
                'Area_Recursos Humanos','Campo_Educativo_Tecnico','Campo_Educativo_Otra','Campo_Educativo_Recursos Humanos',
                'Rol_Gerente','Rendimiento','Rol_Representante de salud','Rol_Tecnico de laboratorio',
                'Rol_Ejecutivo de ventas','Nivel_Opcion_Acciones','Rol_Cientifico investigador','Nivel_Rol',
                'Rol_Representante de ventas','Area_Investigacion & Desarrollo','Rol_Director de fabricacion',
                'Campo_Educativo_Medicina','Campo_Educativo_Biologia']

X_train_ada_imp = X_train_ada.loc[:, [name for name in X_train_ada.columns if name not in cols_del_imp]]
X_test_ada_imp = X_test_ada.loc[:, [name for name in X_test_ada.columns if name not in cols_del_imp]]
```

In [57]:
```
#Entrenar de nuevo
gbm_model_ada_imp = GradientBoostingClassifier(max_depth = 5,max_features = 'auto',random_state = 1)
gbm_model_ada_imp.fit(X_train_ada_imp,y_train_ada)
```

Out[57]:
```
GradientBoostingClassifier(max_depth=5, max_features='auto', random_state=1)
```

In [58]:
```
#Prediccion sobre el test
pred_y_gbm_ada_imp = gbm_model_ada_imp.predict(X_test_ada_imp)
mostrar_resultados(y_test_ada, pred_y_gbm_ada_imp)
#Recall de attrition de 97%, activos en 99%

#Prediccion sobre el train
pred_y_gbm_ada_train_imp = gbm_model_ada_imp.predict(X_train_ada_imp)
```



	precision	recall	f1-score	support
0	0.97	0.99	0.98	884
1	0.99	0.97	0.98	890
accuracy			0.98	1774
macro avg	0.98	0.98	0.98	1774
weighted avg	0.98	0.98	0.98	1774

In [59]:
```
metricas_comparacion(y_test_ada, pred_y_gbm_ada_imp, y_train_ada, pred_y_gbm_ada_train_imp)
```

```
Accuracy:  0.9751972942502819
AUC test:  0.9752313284864508
AUC train:  0.9963899475521882
Delta AUC (overfitting):  0.02115861906573746
```

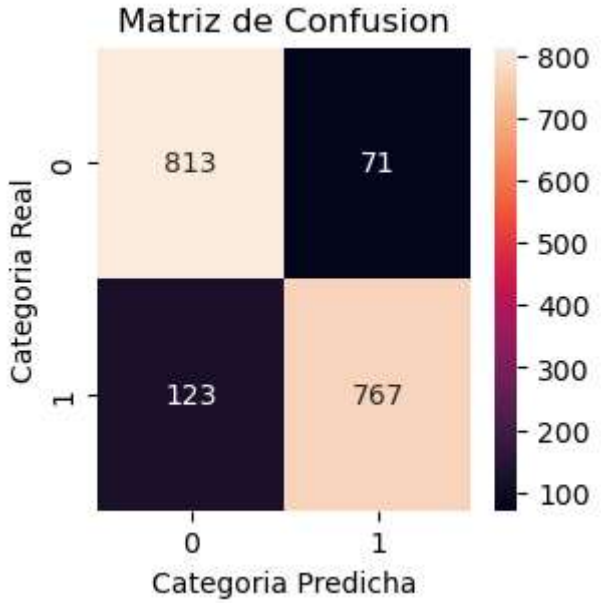
In [62]:
```
#feature_importance = gbm_model_ada_imp.feature_importances_
##Importancia relativa a la importancia maxima
#feature_importance = 100.0 * (feature_importance / 100)
#                             #feature_importance.max())
#sorted_idx = np.argsort(feature_importance)
#pos = np.arange(sorted_idx.shape[0]) + .5
#
#plt.figure(figsize=(8, 18))
#plt.barh(pos, feature_importance[sorted_idx], align='center')
#plt.yticks(pos, X_train_ada_imp.keys()[sorted_idx])
#plt.xlabel('Relative Importance')
#plt.title('Variable Importance')
#plt.show()
```

Búsqueda de mejor semilla sobre modelo elegido: Random Forest (ADASYN) con variables importantes

In [54]:
```
#Almacenamiento inicial de semilla y AUC
mejor_seed = 3996
mejor_auc = 0.890740

#Iterador sobre semillas y predicciones
for i in range(3001,5001):
    #Random Forest individual
    rf_model_ada =RandomForestClassifier(max_depth=5,random_state=i, class_weight = 'balanced')
    rf_model_ada.fit(X_train_ada_imp,y_train_ada)
    #Prediccion sobre test
    y_pred_rf_ada = rf_model_ada.predict(X_test_ada_imp)
    #AUC test
    fpr, tpr, _ = metrics.roc_curve(y_test_ada,  y_pred_rf_ada)
    auc_test = metrics.roc_auc_score(y_test_ada, y_pred_rf_ada)
    if auc_test > mejor_auc:
        mejor_seed = i
        mejor_auc = auc_test

#Imprimir mejores resultados
print("Mejor Seed: ",mejor_seed)
print("Mejor AUC test: ",mejor_auc)

#corriendo hasta 5001
```

```
Mejor Seed:  3996
Mejor AUC test:  0.8907405053637704
```

In [55]:
```
#Resultados Random Forest ADASYN con semilla ganadora
rf_model_ada =RandomForestClassifier(max_depth=5,random_state=3996, class_weight = 'balanced')
rf_model_ada.fit(X_train_ada_imp,y_train_ada)

#Prediccion sobre test
y_pred_rf_ada = rf_model_ada.predict(X_test_ada_imp)

mostrar_resultados(y_test_ada, y_pred_rf_ada)
```



	precision	recall	f1-score	support
0	0.87	0.92	0.89	884
1	0.92	0.86	0.89	890
accuracy			0.89	1774
macro avg	0.89	0.89	0.89	1774
weighted avg	0.89	0.89	0.89	1774

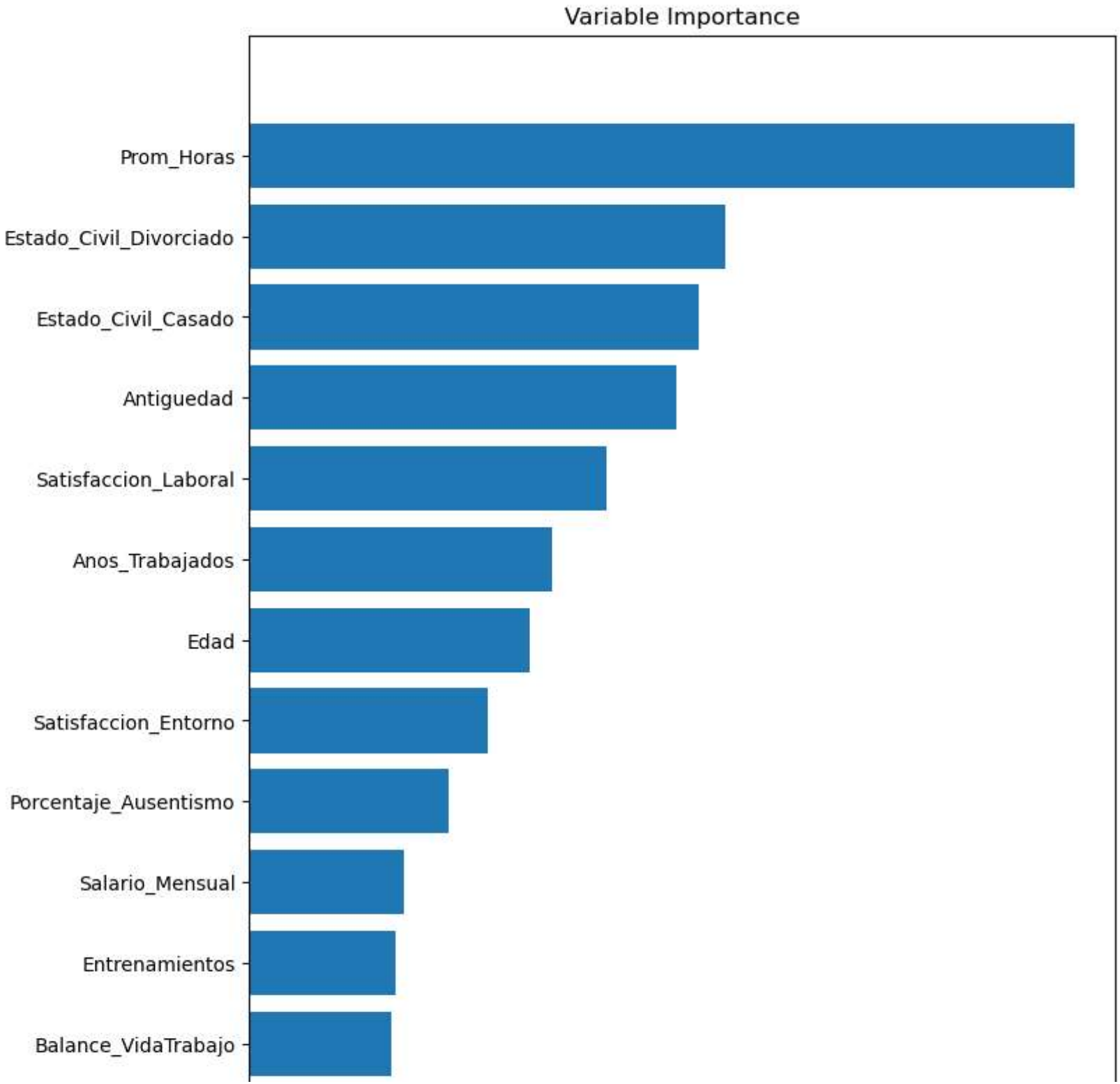
In [58]:
```
#Prediccion sobre el train
pred_y_rf_ada_train = rf_model_ada.predict(X_train_ada_imp)
```

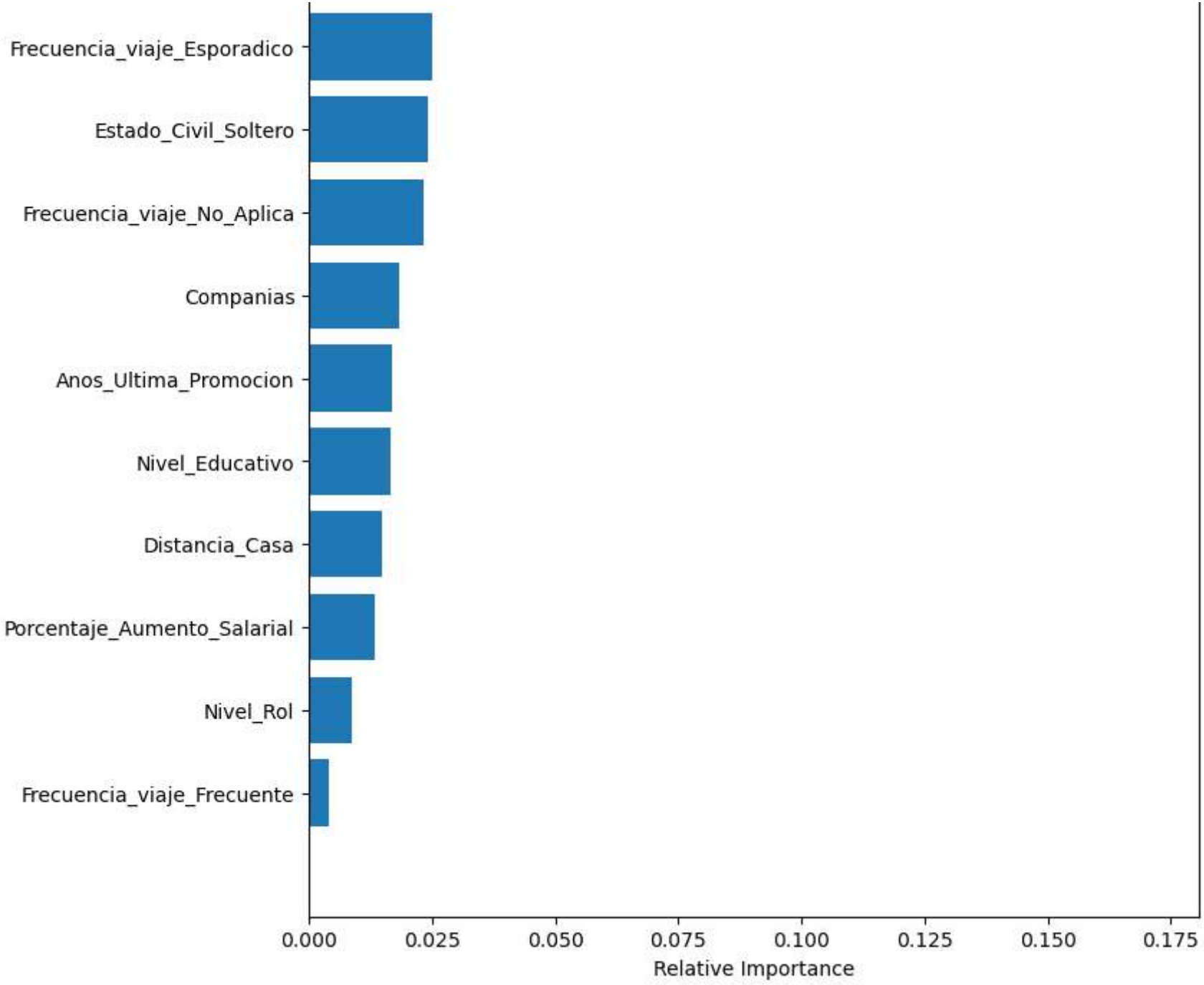
In [59]:
```
metricas_comparacion(y_test_ada, y_pred_rf_ada, y_train_ada, pred_y_rf_ada_train)
```

```
Accuracy:  0.8906426155580609
AUC test:  0.8907405053637704
AUC train:  0.8830928397906496
Delta AUC (overfitting):  0.007647665573120799
```

In [56]:

```python
#Hallar variables importantes
feature_importance = rf_model_ada.feature_importances_
#Importancia relativa a la importancia maxima
feature_importance = 100.0 * (feature_importance / 100)
                              #feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5

plt.figure(figsize=(8, 18))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train_ada_imp.keys()[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```


Variable Importance
Prom_Horas
Estado_Civil_Divorciado
Estado_Civil_Casado
Antiguedad
Satisfaccion_Laboral
Anos_Trabajados
Edad
Satisfaccion_Entorno
Porcentaje_Ausentismo
Salario_Mensual
Entrenamientos
Balance_VidaTrabajo



Predicción sobre base completa para Dashboard

In [60]:
```
#Copia de datos de entrada y llave ID empleado
#De df_raw se sacará la base de dashboard
df_raw = df.copy()
df_raw = df_raw.set_index('ID_empleado')
```

In [61]:
```
df_raw.head()
```

Out[61]:

ID_empleado	Edad	Desercion	Frecuencia_viaje	Area	Distancia_Casa	Nivel_Educativo	Campo_Educativo	Recuento_Empleado	Genero	Nivel_Rol
1	51	No	Esporadico	Ventas	6	2	Biologia	1	Femenino	2
2	31	Si	Frecuente	Investigacion & Desarrollo	10	1	Biologia	1	Femenino	3
3	32	No	Frecuente	Ventas	17	4	Otra	1	Masculino	3
4	38	No	No_Aplica	Recursos Humanos	2	5	Biologia	1	Masculino	3
5	32	No	Esporadico	Ventas	10	1	Medicina	1	Masculino	3

In [62]:
```
#Quitar columnas sobrantes para crear entrada al modelo
#De df_raw_X se sacara entrada al modelo, omitiendo resultado final
cols_del = ['Mayor18','Recuento_Empleado','Horario','Genero','Anos_Lider_Actual','Ausentismo']
df_raw_X = df_raw.loc[:, [name for name in df_raw.columns if name not in cols_del]]

df_raw_X.head()
```

Out[62]:

ID_empleado	Edad	Desercion	Frecuencia_viaje	Area	Distancia_Casa	Nivel_Educativo	Campo_Educativo	Nivel_Rol	Rol	Estado_Civil	Sa
1	51	No	Esporadico	Ventas	6	2	Biologia	2	Representante de salud	Casado	
2	31	Si	Frecuente	Investigacion & Desarrollo	10	1	Biologia	3	Cientifico investigador	Soltero	
3	32	No	Frecuente	Ventas	17	4	Otra	3	Ejecutivo de ventas	Casado	
4	38	No	No_Aplica	Recursos Humanos	2	5	Biologia	3	Recursos Humanos	Casado	
5	32	No	Esporadico	Ventas	10	1	Medicina	3	Ejecutivo de ventas	Soltero	

In [63]:
```
#Dummificar y borrar target redundante
df_raw_X = pd.get_dummies(df_raw_X, columns=['Desercion', 'Frecuencia_viaje', 'Area','Campo_Educativo','Rol',
                                             'Estado_Civil'])
df_raw_X = df_raw_X.drop('Desercion_No', axis=1)
df_raw_X.head()
```

Out[63]:

ID_empleado	Edad	Distancia_Casa	Nivel_Educativo	Nivel_Rol	Salario_Mensual	Companias	Porcentaje_Aumento_Salarial	Nivel_Opcion_Acciones	Anos_Tra
1	51	6	2	2	131.16	1	11	0	
2	31	10	1	3	41.89	0	23	1	
3	32	17	4	3	193.28	1	15	3	
4	38	2	5	3	83.21	3	11	3	
5	32	10	1	3	23.42	4	12	2	

In [64]:
```
#Quitar target principal de la entrada
df_raw_X = df_raw_X.drop('Desercion_Si', axis=1)
df_raw_X.head()
```

Out[64]:

ID_empleado	Edad	Distancia_Casa	Nivel_Educativo	Nivel_Rol	Salario_Mensual	Companias	Porcentaje_Aumento_Salarial	Nivel_Opcion_Acciones	Anos_Tra
1	51	6	2	2	131.16	1	11	0	
2	31	10	1	3	41.89	0	23	1	
3	32	17	4	3	193.28	1	15	3	
4	38	2	5	3	83.21	3	11	3	
5	32	10	1	3	23.42	4	12	2	

In [65]:
```
#Quitar columnas con menos importancia segun analisis previo
cols_del_imp = ['Area_Investigacion & Desarrollo','CompromisoTrabajo','Nivel_Opcion_Acciones','Campo_Educativo_Tecnico',
                'Area_Ventas','Campo_Educativo_Biologia','Rol_Ejecutivo de ventas','Rol_Representante de ventas',
                'Campo_Educativo_Mercadeo','Campo_Educativo_Otra','Rol_Gerente','Rol_Cientifico investigador',
                'Rol_Recursos Humanos','Area_Recursos Humanos','Rol_Representante de salud','Rol_Tecnico de laboratorio',
                'Rol_Director de investigacion','Rendimiento','Campo_Educativo_Recursos Humanos',
                'Rol_Director de fabricacion','Campo_Educativo_Medicina']

df_raw_X = df_raw_X.loc[:, [name for name in df_raw_X.columns if name not in cols_del_imp]]
```

In [66]:
```
df_raw_X.head()
```

Out[66]:

ID_empleado	Edad	Distancia_Casa	Nivel_Educativo	Nivel_Rol	Salario_Mensual	Companias	Porcentaje_Aumento_Salarial	Anos_Trabajados	Entrenamiento
1	51	6	2	2	131.16	1	11	1	
2	31	10	1	3	41.89	0	23	6	
3	32	17	4	3	193.28	1	15	5	
4	38	2	5	3	83.21	3	11	13	
5	32	10	1	3	23.42	4	12	9	

In [67]:
```
#Random Forest individual campeon
rf_model_ada =RandomForestClassifier(max_depth=5,random_state=3996, class_weight = 'balanced')
rf_model_ada.fit(X_train_ada_imp,y_train_ada)
```

Out[67]:
```
RandomForestClassifier(class_weight='balanced', max_depth=5, random_state=3996)
```

In [68]:
```
#Prediccion sobre entrada
y_pred_rf_ada_raw = rf_model_ada.predict(df_raw_X)
```

In [69]:
```
#Convertir prediccion a dataframe con mismo indice de entrada
y_pred_rf_ada_raw_df = pd.DataFrame(data = y_pred_rf_ada_raw, columns = ['y_pred_rf_ada_raw'],
                                    index = df_raw_X.index.copy())

y_pred_rf_ada_raw_df.head()
```

Out[69]:

ID_empleado	y_pred_rf_ada_raw
1	0
2	1
3	0
4	0
5	0

In [70]:
```
#Probabilidades sobre entrada
y_prob_rf_ada_raw = rf_model_ada.predict_proba(df_raw_X)[:,1]
y_prob_rf_ada_raw
```

Out[70]:
```
array([0.3208080212, 0.6122491676, 0.3716451499, ..., 0.1820298632,
       0.3708517273, 0.3106304255])
```

In [71]:
```
#Convertir probabilidades a dataframe con mismo indice de entrada
y_prob_rf_ada_raw_df = pd.DataFrame(data = y_prob_rf_ada_raw, columns = ['y_prob_rf_ada_raw'],
                                    index = df_raw_X.index.copy())

y_prob_rf_ada_raw_df.head()
```

Out[71]:

ID_empleado	y_prob_rf_ada_raw
1	0.320808
2	0.612249
3	0.371645
4	0.195029
5	0.453909

In [72]:
```
#Combinar resultados con entrada cruda
df_out = pd.merge(df_raw, y_pred_rf_ada_raw_df, how = 'left', left_index = True, right_index = True)
df_out = pd.merge(df_out, y_prob_rf_ada_raw_df, how = 'left', left_index = True, right_index = True)
df_out.head()
```

Out[72]:

ID_empleado	Edad	Desercion	Frecuencia_viaje	Area	Distancia_Casa	Nivel_Educativo	Campo_Educativo	Recuento_Empleado	Genero	Nivel_Rol
1	51	No	Esporadico	Ventas	6	2	Biologia	1	Femenino	2
2	31	Si	Frecuente	Investigacion & Desarrollo	10	1	Biologia	1	Femenino	3
3	32	No	Frecuente	Ventas	17	4	Otra	1	Masculino	3
4	38	No	No_Aplica	Recursos Humanos	2	5	Biologia	1	Masculino	3
5	32	No	Esporadico	Ventas	10	1	Medicina	1	Masculino	3

In [73]:
```
#ID empleado nuevamente a columna
df_out.reset_index(inplace=True)
df_out.head()
```

Out[73]:

	ID_empleado	Edad	Desercion	Frecuencia_viaje	Area	Distancia_Casa	Nivel_Educativo	Campo_Educativo	Recuento_Empleado	Genero	Nivel_
0	1	51	No	Esporadico	Ventas	6	2	Biologia	1	Femenino	
1	2	31	Si	Frecuente	Investigacion & Desarrollo	10	1	Biologia	1	Femenino	
2	3	32	No	Frecuente	Ventas	17	4	Otra	1	Masculino	
3	4	38	No	No_Aplica	Recursos Humanos	2	5	Biologia	1	Masculino	
4	5	32	No	Esporadico	Ventas	10	1	Medicina	1	Masculino	

In [74]:

```
#Exportar a csv
ruta = r'C:\Users\Jaime Andres Molina\Documents\Maestria\Proyecto Empresarial\Entrega Semestre 3\all_attrition_prob_v2.csv'
df_out.to_csv(ruta, index = False, sep=';')
```

Calculadora de Riesgo de Deserción para Candidatos

In [1]:
```
#Librerias numericas y graficos
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#Graficos
import plotly as py
import plotly.graph_objs as go

#Manejo de fechas
import datetime as dt

#Modelos de Arboles
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

#Medidas
from sklearn import metrics
from sklearn import model_selection
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score

#Split X - y en conjuntos training y testing
from sklearn.model_selection import train_test_split

#Mostrar todas las columnas de frame
pd.set_option('display.max_columns', None)

#Asignar el formato de los decimales
np.set_printoptions(formatter={'float': lambda x: "{0:0.10f}".format(x)})

#Para datos desbalanceados
from pylab import rcParams

from imblearn.under_sampling import NearMiss
from imblearn.over_sampling import RandomOverSampler
from imblearn.combine import SMOTETomek
```

```
from imblearn.ensemble import BalancedBaggingClassifier
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import ADASYN

from collections import Counter
```

In [2]:
```
#Ejecutar si no esta instalada la libreria tkinter en el equipo
#pip install tk
```

In [3]:
```
#Importar utilidades de tkinter (visualizacion)
import sys
from tkinter import *
from tkinter import ttk
from tkinter import filedialog
import tkinter as tk
#Importar libreria de imagenes
from PIL import ImageTk, Image
import os
```

Inclusión de modelo predictivo

In [4]:
```
#Carga de datos modelo entrenado
ruta = r'C:\Users\Jaime Andres Molina\Documents\Maestria\Proyecto Empresarial\Fuentes\01_Fuentes\all_attrition_modeling.csv'
#CSV
df = pd.read_csv (ruta, sep = ';')
```

In [5]:
```
#Quitar columnas por nombre
cols_del = ['ID_empleado','Mayor18','Recuento_Empleado','Horario','Genero','Anos_Lider_Actual','Ausentismo',
            'Frecuencia_viaje','Porcentaje_Aumento_Salarial','Nivel_Opcion_Acciones','Entrenamientos','Antiguedad',
            'Anos_Ultima_Promocion','Satisfaccion_Entorno','Satisfaccion_Laboral','Balance_VidaTrabajo',
            'CompromisoTrabajo','Rendimiento','Prom_Horas','Porcentaje_Ausentismo']
df_in = df.loc[:, [name for name in df.columns if name not in cols_del]]
```

In [6]:
```
#Dummificar variables seleccionadas
df_in= pd.get_dummies(df_in, columns=['Desercion', 'Area','Campo_Educativo', 'Rol', 'Estado_Civil'])
```

In [7]:
```
#Quitar columnas redundantes
cols_del_in = ['Desercion_No']
df_in = df_in.loc[:, [name for name in df_in.columns if name not in cols_del_in]]
```

In [8]:
```
#Definimos nuestras etiquetas y features
y = df_in['Desercion_Si']
X = df_in.drop('Desercion_Si', axis=1)
```

In [9]:
```
#ADASYN sobre entrenamiento
ada = ADASYN(sampling_strategy='auto', n_neighbors=5, random_state=1)
X_ada, y_ada = ada.fit_resample(X, y)
#Dividir la base en train y test de ADASYN
X_train_ada, X_test_ada, y_train_ada, y_test_ada = train_test_split(X_ada, y_ada, test_size=0.25, random_state=28)
```

In [10]:
```
#Random Forest con ADASYN mejor semilla
rf_model_ada =RandomForestClassifier(max_depth=5,random_state=4455, class_weight = 'balanced')
rf_model_ada.fit(X_train_ada,y_train_ada)
```

Out[10]:
```
RandomForestClassifier(class_weight='balanced', max_depth=5, random_state=4455)
```

Desarrollo de interfaz gráfica

In [11]:
```
#Crear app principal llamada root
root = tk.Tk()
#Se configura un titulo para la ventana
root.title("Calculadora de Probabilidad de Deserción")
#Para evitar que la ventana cambie de tamaño
root.resizable(0, 0)
#Configuracion de tamaño relativo de columnas y filas
#Ultima columna doble de ancho
root.columnconfigure(0, weight=1)
root.columnconfigure(1, weight=1)
root.columnconfigure(2, weight=2)
#Primera y segunda fila doble de ancho
root.rowconfigure(0, weight=2)
root.rowconfigure(1, weight=2)
```

In [12]:
```
#Crear titulo principal en frame con una etiqueta
titulo =Label(root,text="Probabilidad de Deserción para Candidatos",
              font=("Segoe UI",20,"bold"),fg="#005C76",anchor="center",padx=10, pady=10)
#Posicion del titulo
titulo.grid(column=1,row=0,columnspan=2)
```

In [13]:
```
#Insertar logo de farmaceutica
ruta_img = r'C:\Users\Jaime Andres Molina\Documents\Maestria\Proyecto Empresarial\Imagenes\calculadora.png'
img = ImageTk.PhotoImage(Image.open(ruta_img))
logo_empresa = Label(root,image = img,padx=10,pady=20)
logo_empresa.grid(column=0,row=0)
```

In [14]:
```
#Creacion de etiquetas de segmentos generales

#Crear etiqueta Criterio candidato y posicion
```

```
etiqueta_criteriocandidato=Label(root,text="Datos de Candidato",font=("Segoe UI",16,'bold'),fg="#A92E63")
etiqueta_criteriocandidato.grid(column=0,row=3,columnspan=2,padx=10, pady=10)
#Crear etiqueta resultados deserción y posicion
etiqueta_resultadosdesercion=Label(root,text="Resultados deserción",font=("Segoe UI",16,'bold'),fg="#A92E63")
etiqueta_resultadosdesercion.grid(column=2,row=3,padx=10, pady=10)

#Crear etiqueta resultado individual y posicion
etiqueta_individual=Label(root,text="Individual",font=("Segoe UI",14,'bold'),fg="#00A981")
etiqueta_individual.grid(column=2,row=5)
#Crear etiqueta resultado grupal y posicion
etiqueta_grupal=Label(root,text="Grupal",font=("Segoe UI",14,'bold'),fg="#00A981")
etiqueta_grupal.grid(column=2,row=10)
```

In [15]:

```
#Crear etiquetas de entrada de datos

#Candidato entrada numerica
entry_candidato=Label(root, text="N° Candidato",font=("Segoe UI",12),fg="#4D4D4D").grid(row=4, column=0,padx=10,pady=5,
                                                                                       sticky='E')
candidatoE =IntVar()
entry_candidato = Entry(root, textvariable=candidatoE,width=22,font=("Segoe UI",12))
entry_candidato.grid(row=4, column=1,padx=10,pady=5,sticky='W')

#Area entrada lista
entry_Area=Label(root, text="Area",font=("Segoe UI",12),fg="#4D4D4D").grid(row=5, column=0,padx=10,pady=5,sticky='E')
comboArea = ttk.Combobox(state="readonly",values=["Investigacion & Desarrollo", "Recursos Humanos", "Ventas"]
                         ,font=("Segoe UI",12))
comboArea.grid(row=5, column=1,padx=10,pady=5,sticky='W')

#Distancia a Casa entrada numerica
entry_Distancia=Label(root, text="Distancia [km]",font=("Segoe UI",12),fg="#4D4D4D").grid(row=6, column=0,padx=10,pady=5,
                                                                                         sticky='E')
dist_e= IntVar()
entry_Distancia=Entry(root, textvariable=dist_e,width=22,font=("Segoe UI",12))
entry_Distancia.grid(row=6, column=1,padx=10,pady=5,sticky='W')

#Nivel de rol entrada lista
entry_NivelRol=Label(root, text="Nivel Rol",font=("Segoe UI",12),fg="#4D4D4D").grid(row=7, column=0,padx=10,pady=5,
                                                                                   sticky='E')
comboNivelR = ttk.Combobox(state="readonly",values=["1","2","3","4","5"],font=("Segoe UI",12))
comboNivelR.grid(row=7, column=1,padx=10,pady=5,sticky='W')

#Rol entrada lista
entry_Rol=Label(root, text="Rol",font=("Segoe UI",12),fg="#4D4D4D").grid(row=8, column=0,padx=10, pady=5,sticky='E')
comboRol = ttk.Combobox(state="readonly",
                        values=["Cientifico investigador", "Director de fabricacion", "Director de investigacion",
                                "Ejecutivo de ventas","Gerente","Recursos Humanos","Representante de salud",
                                "Representante de ventas","Tecnico de laboratorio"]
```

```
                        ,font=("Segoe UI",12))
comboRol.grid(row=8, column=1,padx=10, pady=5,sticky='W')

#Campo Educativo entrada lista
entry_CampoEducativo=Label(root, text="Campo Educativo",font=("Segoe UI",12),fg="#4D4D4D").grid(row=9, column=0,padx=10,
                                                                                                pady=5,sticky='E')
comboCampoE = ttk.Combobox(state="readonly",
                           values=["Biologia", "Medicina", "Mercadeo","Otra","Recursos Humanos","Tecnico"]
                           ,font=("Segoe UI",12))
comboCampoE.grid(row=9, column=1,padx=10,pady=5,sticky='W')

#Nivel Educativo entrada lista
entry_NivelEducativo=Label(root, text="Nivel Educativo",font=("Segoe UI",12),fg="#4D4D4D").grid(row=10, column=0,padx=10,
                                                                                                pady=5,sticky='E')
comboNivelE = ttk.Combobox(state="readonly",values=["1","2","3","4","5"],font=("Segoe UI",12))
comboNivelE.grid(row=10, column=1,padx=10,pady=5,sticky='W')

#Experiencia entrada numerica
entry_exp =Label(root, text="Experiencia [Años]",font=("Segoe UI",12),fg="#4D4D4D").grid(row=11, column=0,padx=10,pady=5,
                                                                                         sticky='E')
exp_e= IntVar()
entry_exp=Entry(root, textvariable=exp_e,width=22,font=("Segoe UI",12))
entry_exp.grid(row=11, column=1,padx=10,pady=5,sticky='W')

#Compañias entrada numerica
entry_companias=Label(root, text="Compañías",font=("Segoe UI",12),fg="#4D4D4D").grid(row=12, column=0,padx=10,pady=5,
                                                                                     sticky='E')
companias_e= IntVar()
entry_companias=Entry(root, textvariable=companias_e,width=22,font=("Segoe UI",12))
entry_companias.grid(row=12, column=1,padx=10,pady=5,sticky='W')

#Salario entrada numerica
entry_salario=Label(root, text="Salario Mensual",font=("Segoe UI",12),fg="#4D4D4D").grid(row=13, column=0,padx=10,pady=5,
                                                                                        sticky='E')
salario_e= IntVar()
entry_salario=Entry(root, textvariable=salario_e,width=22,font=("Segoe UI",12))
entry_salario.grid(row=13, column=1,padx=10,pady=5,sticky='W')

#Estado Civil entrada lista
entry_EstadoCivil=Label(root, text="Estado Civil",font=("Segoe UI",12),fg="#4D4D4D").grid(row=14, column=0,padx=10,pady=5,
                                                                                          sticky='E')
comboEstadoC = ttk.Combobox(state="readonly",values=["Casado", "Divorciado", "Soltero"],font=("Segoe UI",12))
comboEstadoC.grid(row=14, column=1,padx=10,pady=5,sticky='W')

#Edad entrada numerica
entry_edad=Label(root, text="Edad",font=("Segoe UI",12),fg="#4D4D4D").grid(row=15, column=0,padx=10,pady=5,sticky='E')
edad_e= IntVar()
```

```
entry_edad=Entry(root, textvariable=edad_e,width=22,font=("Segoe UI",12))
entry_edad.grid(row=15, column=1,padx=10,pady=5,sticky='W')
```

In [16]:

```
#Label de Resultado prediccion individual e iconos

ruta_alta = r'C:\Users\Jaime Andres Molina\Documents\Maestria\Proyecto Empresarial\Imagenes\Alerta_Alta.png'
ruta_media = r'C:\Users\Jaime Andres Molina\Documents\Maestria\Proyecto Empresarial\Imagenes\Alerta_Media.png'
ruta_baja = r'C:\Users\Jaime Andres Molina\Documents\Maestria\Proyecto Empresarial\Imagenes\Alerta_Baja.png'
#Guardar iconos
img_alta_original = Image.open(ruta_alta)
img_media_original = Image.open(ruta_media)
img_baja_original = Image.open(ruta_baja)
#Cambiar tamaño en pixeles
resize_alta = img_alta_original.resize((100, 100))
resize_media = img_media_original.resize((100, 100))
resize_baja = img_baja_original.resize((100, 100))
#Guardar en tkinter
img_alta = ImageTk.PhotoImage(resize_alta)
img_media = ImageTk.PhotoImage(resize_media)
img_baja = ImageTk.PhotoImage(resize_baja )

resultado_ind=Label(root, text=" - %",font=("Segoe UI",20),fg="#4D4D4D")
resultado_ind.grid(row=6, column=2,padx=10,pady=10,rowspan=4)

#Tabla de resultados grupales top 5

#Cambiar tamaño en pixeles iconos
resize_alta_grupal = img_alta_original.resize((20,20))
resize_media_grupal = img_media_original.resize((20,20))
resize_baja_grupal = img_baja_original.resize((20,20))
#Guardar en tkinter
img_alta_grupal = ImageTk.PhotoImage(resize_alta_grupal)
img_media_grupal = ImageTk.PhotoImage(resize_media_grupal)
img_baja_grupal = ImageTk.PhotoImage(resize_baja_grupal)

#Widget de tabla
treeview = ttk.Treeview(columns=("Candidato", "Probabilidad"),height=5)
#Nombramiento columnas
treeview.heading('#0',text='')
treeview.heading("Candidato", text="Candidato")
treeview.heading("Probabilidad", text="Probabilidad")
#Ajustar el ancho de las columnas
treeview.column('#0',anchor=CENTER,width=50)
treeview.column("# 1",anchor=CENTER, stretch=NO,width=100)
treeview.column("# 2",anchor=CENTER, stretch=NO,width=100)

#Estilo para las líneas y encabezado
```

```
style = ttk.Style()
style.configure("mystyle.Treeview", highlightthickness=10, bd=10, borderwidth=10, font=('Segoe UI', 10))
style.configure("mystyle.Treeview.Heading", font=('Segoe UI',10,'bold'))
#Cambiar el color de las líneas de la tabla a negro
style.configure("mystyle.Treeview", foreground="black", background="white")
#Aplicar el estilo a la tabla
treeview.configure(style="mystyle.Treeview")
#Ubicacion tabla
treeview.grid(row=11, column=2,padx=10,pady=10,rowspan=5)
```

Funciones

In [17]:

```
#Captura de datos y calculo predicción individual
def enviar_datos():
    #Array para almacenar los datos de entrada
    datos = []
    #Copia de datos de entrada al modelo y crea ID empleado, limpieza de frame entrada
    df_raw = X_train_ada.copy()
    df_raw.insert(loc=0, column='ID_empleado',value=0)
    df_raw.drop(df_raw.index,inplace=True)
    #Almacenamiento datos
    datos.append({
            'ID_empleado': entry_candidato.get(),
            'Area': comboArea.get(),
            'Distancia_Casa': entry_Distancia.get(),
            'Nivel_Rol': comboNivelR.get(),
            'Rol':comboRol.get(),
            'Campo_Educativo': comboCampoE.get(),
            'Nivel_Educativo': comboNivelE.get(),
            'Anos_Trabajados': entry_exp.get(),
            'Companias': entry_companias.get(),
            'Salario_Mensual': entry_salario.get(),
            'Edad': entry_edad.get(),
            'Estado_Civil': comboEstadoC.get()
    })

    #Transformar a dataframe
    df_calculadora = pd.DataFrame(datos,columns=['ID_empleado','Edad','Area','Distancia_Casa','Nivel_Educativo',
                                                 'Campo_Educativo','Nivel_Rol','Rol','Estado_Civil','Salario_Mensual',
                                                 'Companias','Anos_Trabajados'])
    #Garantizar tipos de datos numericos entrada
    df_calculadora['ID_empleado'] = pd.to_numeric(df_calculadora['ID_empleado'].str.replace(',','.'))
    df_calculadora['Edad'] = pd.to_numeric(df_calculadora['Edad'].str.replace(',','.'))
    df_calculadora['Distancia_Casa'] = pd.to_numeric(df_calculadora['Distancia_Casa'].str.replace(',','.'))
    df_calculadora['Nivel_Rol'] = pd.to_numeric(df_calculadora['Nivel_Rol'].str.replace(',','.'))
    df_calculadora['Nivel_Educativo'] = pd.to_numeric(df_calculadora['Nivel_Educativo'].str.replace(',','.'))
```

```
df_calculadora['Anos_Trabajados'] = pd.to_numeric(df_calculadora['Anos_Trabajados'].str.replace(',','.'))
df_calculadora['Companias'] = pd.to_numeric(df_calculadora['Companias'].str.replace(',','.'))
df_calculadora['Salario_Mensual'] = pd.to_numeric(df_calculadora['Salario_Mensual'].str.replace(',','.'))
#Dummificar entrada
df_calculadora = pd.get_dummies(df_calculadora, columns=['Area','Campo_Educativo','Rol','Estado_Civil'])

#--------------Funciones de prediccion
#Inserta datos en frame entrada y rellena dummies
for column in df_calculadora:
    df_raw[column] = df_calculadora[column]
df_raw.fillna(0,inplace=True)
#Set de index y creacion entrada modelo
df_raw = df_raw.set_index('ID_empleado')
df_raw_X = df_raw.copy()
#Prediccion sobre entrada
y_pred_rf_ada_raw = rf_model_ada.predict(df_raw_X)

#Convertir prediccion a dataframe con mismo indice de entrada
y_pred_rf_ada_raw_df = pd.DataFrame(data = y_pred_rf_ada_raw, columns = ['y_pred_rf_ada_raw'],
                                    index = df_raw_X.index.copy())
#Probabilidades sobre entrada
y_prob_rf_ada_raw = rf_model_ada.predict_proba(df_raw_X)[:,1]

#Convertir probabilidades a dataframe con mismo indice de entrada
y_prob_rf_ada_raw_df = pd.DataFrame(data = y_prob_rf_ada_raw, columns = ['y_prob_rf_ada_raw'],
                                    index = df_raw_X.index.copy())
#Combinar resultados con entrada cruda
df_out = pd.merge(df_raw, y_pred_rf_ada_raw_df, how = 'left', left_index = True, right_index = True)
df_out = pd.merge(df_out, y_prob_rf_ada_raw_df, how = 'left', left_index = True, right_index = True)
#ID empleado nuevamente a columna
df_out.reset_index(inplace=True)

#Muestra resultado en label

print((df_out['y_prob_rf_ada_raw'].values[:1][0])*100)
print(str(round((df_out['y_prob_rf_ada_raw'].values[:1][0])*100,2))+'%')

resultado_num = round((df_out['y_prob_rf_ada_raw'].values[:1][0])*100,2)
resultado = str(resultado_num)+'%'

resultado_ind.config(text=resultado)

#Color de label segun resultado
if resultado_num >= 55:
    resultado_ind.config(font=("Segoe UI",20,'bold'),fg="#A92E63",image=img_alta,compound='top')
elif resultado_num >= 30:
    resultado_ind.config(font=("Segoe UI",20,'bold'),fg="#EC9A29",image=img_media,compound='top')
```

```
    else:
        resultado_ind.config(font=("Segoe UI",20,'bold'),fg="#0F8B8D",image=img_baja,compound='top')
```

In [18]:

```
#Para manejo de calculo masivo

#Instancia de dataframe para almacenar resultados grupales
df_out = pd.DataFrame()

def exportar_csv():
    ruta_archivo = filedialog.asksaveasfilename(defaultextension=".csv", filetypes=[("Archivo CSV", "*.csv")])
    if ruta_archivo:
        try:
            #Guardar DataFrame como archivo CSV en la ruta especificada
            df_out.to_csv(ruta_archivo, index=False)
            print("Datos exportados exitosamente a:", ruta_archivo)

        except Exception as e:
            print("Error al exportar los datos:", e)

def limpiar_masivo():
    #Limpiar la tabla de calculo masivo
    for i in treeview.get_children():
        treeview.delete(i)

def calculo_masivo(df_grupal):
    global df_out
    #Copia de datos de entrada al modelo y crea ID empleado, limpieza de frame entrada
    df_raw = X_train_ada.copy()
    df_raw.insert(loc=0, column='ID_empleado',value=0)
    df_raw.drop(df_raw.index,inplace=True)

    #Crear base para exportacion con indice nuevo
    df_grupal_export = df_grupal.set_index('ID_empleado')
    #Dummificar entrada
    df_grupal = pd.get_dummies(df_grupal, columns=['Area','Campo_Educativo','Rol','Estado_Civil'])

    #--------------Funciones de prediccion
    #Inserta datos en frame entrada y rellena dummies
    for column in df_grupal:
        df_raw[column] = df_grupal[column]
    df_raw.fillna(0,inplace=True)
    #Set de index y creacion entrada modelo
    df_raw = df_raw.set_index('ID_empleado')
    df_raw_X = df_raw.copy()
    #Prediccion sobre entrada
    y_pred_rf_ada_raw = rf_model_ada.predict(df_raw_X)
```

```
    #Convertir prediccion a dataframe con mismo indice de entrada
    y_pred_rf_ada_raw_df = pd.DataFrame(data = y_pred_rf_ada_raw, columns = ['y_pred_rf_ada_raw'],
                                        index = df_raw_X.index.copy())
    #Probabilidades sobre entrada
    y_prob_rf_ada_raw = rf_model_ada.predict_proba(df_raw_X)[:,1]

    #Convertir probabilidades a dataframe con mismo indice de entrada
    y_prob_rf_ada_raw_df = pd.DataFrame(data = y_prob_rf_ada_raw, columns = ['y_prob_rf_ada_raw'],
                                        index = df_raw_X.index.copy())
    #Combinar resultados con entrada cruda
    df_out = pd.merge(df_grupal_export, y_pred_rf_ada_raw_df, how = 'left', left_index = True, right_index = True)
    df_out = pd.merge(df_out, y_prob_rf_ada_raw_df, how = 'left', left_index = True, right_index = True)
    #ID empleado nuevamente a columna
    df_out.reset_index(inplace=True)
    df_grupal_export.reset_index(inplace=True)

    #Extraer top 5
    df_out_top = df_out.sort_values('y_prob_rf_ada_raw',ascending = True).head()
    #Mostrar top 5 en pantalla
    #Limpiar tabla si ya hay datos
    limpiar_masivo()
    # Insertar datos en la tabla
    for ind in df_out_top.index:
        if df_out_top['y_prob_rf_ada_raw'][ind]*100 >= 55:
            imagen_registro = img_alta_grupal
        elif df_out_top['y_prob_rf_ada_raw'][ind]*100 >= 30:
            imagen_registro = img_media_grupal
        else:
            imagen_registro = img_baja_grupal

        treeview.insert(parent="",index="end", image=imagen_registro,
                        value=(df_out_top['ID_empleado'][ind],
                               str(round(df_out_top['y_prob_rf_ada_raw'][ind]*100,2))+'%'))
    #Habilita exportacion
    boton_exportar["state"] = "normal"
    boton_exportar.config(bg='teal')

#Captura de datos
def cargarArchivo():
    #Lectura
    ruta_archivo = filedialog.askopenfilename(filetypes=[("Archivo XLSX", "*.xlsx")])
    if ruta_archivo:
        try:
            #Cargar el archivo XLSX en un DataFrame
            df_masivo = pd.read_excel(ruta_archivo)
            calculo_masivo(df_masivo)
```

```
        except Exception as e:
            print("Error al cargar el archivo:", e)
```

Accionables de interfaz

In [19]:
```
#Boton evaluar - funcion enviar_datos
boton_evaluar = Button(root, text="Evaluar",font=("Segoe UI",12,'bold'),fg="white",bg="teal",command=enviar_datos)
boton_evaluar.grid(row=18,column=0,columnspan=2,pady=10)

#Boton de cargue masivo
boton_cargar = tk.Button(root, text="Cargar Formato Grupal", font=("Segoe UI",12,'bold'),fg="white",bg="teal",
                         command=cargarArchivo)
boton_cargar.grid(row=18,column=2,pady=10)

#Boton de exportar masivo
boton_exportar = tk.Button(root, text="Exportar resultados", font=("Segoe UI",12,'bold'),fg="white",bg="#4D4D4D",
                         command=exportar_csv,disabledforeground="black")
boton_exportar.grid(row=19,column=2,pady=10)
boton_exportar["state"] = "disabled"
```

Ejecución de Interfaz

In [20]:
```
#Mostrar la ventana (entra en un loop)
root.mainloop()
```

```
63.343023230030774
63.34%
20.815220158606717
20.82%
Datos exportados exitosamente a: C:/Users/Jaime Andres Molina/Documents/Maestria/Proyecto Empresarial/Fuentes/01_Fuentes/Carga Ca
lculadora/resultados.csv
```