Clasificación Binaria con Regresión Logística

Entender el problema

Nos piden un modelo que prediga si los clientes van a pagar o impagar para poder hacer acciones preventivas en algún caso (clientes ya existentes) o comerciales (créditos preconcedidos).

Parace muy claro, pero ¿qué significa impagar? ¿Es que mañana dejo de pagar una factura?¿Y si la repongo en menos de 30 días?¿Es tener una deuda a los seis meses sin pagar? Esto tenemos que aclararlo, no esperar a deducirlo del datatset (Aunque en estos ejemplos lo hagamos así). En nuestro caso nos contestan que es tener deudas pendientes de más de 2 años. Perfecto. Vamos con ello.

Prepaaración Datos: Primer Vistazo

In [2]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

df = pd.read_csv('data/cs-training.csv')

df.head()
```

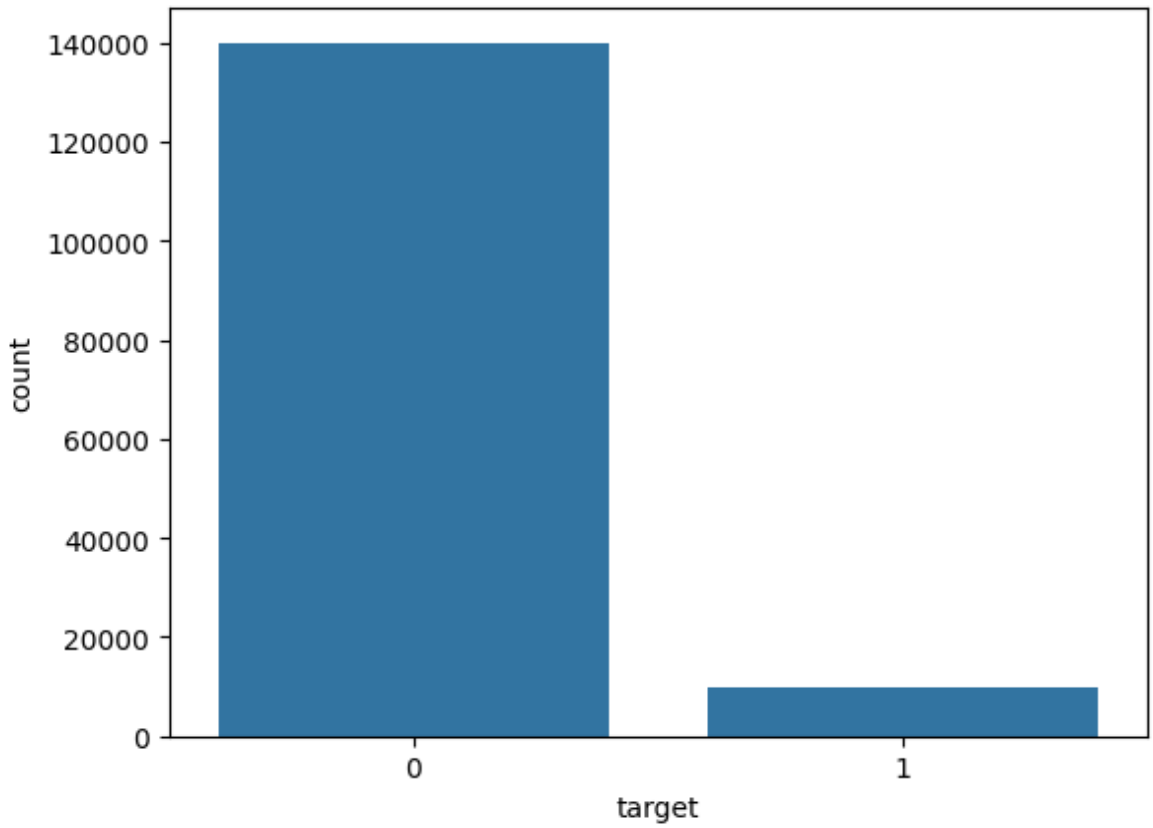
Out[2]:

	Unnamed: 0	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	Numb 59DaysPastD
0	1	1	0.766127	45	
1	2	0	0.957151	40	
2	3	0	0.658180	38	
3	4	0	0.233810	30	
4	5	0	0.907239	49	

Preparación Datos: Limpieza de datos

In [3]:
```
df.rename(columns = {'SeriousDlqin2yrs': 'target'}, inplace=True)
```

In [4]:
```
sns.countplot(data=df, x='target');
```



In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 12 columns):
 #   Column                                Non-Null Count   Dtype  
---  ------                                --------------   -----  
 0   Unnamed: 0                            150000 non-null  int64  
 1   target                                150000 non-null  int64  
 2   RevolvingUtilizationOfUnsecuredLines  150000 non-null  float64
 3   age                                   150000 non-null  int64  
 4   NumberOfTime30-59DaysPastDueNotWorse  150000 non-null  int64  
 5   DebtRatio                             150000 non-null  float64
 6   MonthlyIncome                         120269 non-null  float64
 7   NumberOfOpenCreditLinesAndLoans       150000 non-null  int64  
 8   NumberOfTimes90DaysLate               150000 non-null  int64  
 9   NumberRealEstateLoansOrLines          150000 non-null  int64  
 10  NumberOfTime60-89DaysPastDueNotWorse  150000 non-null  int64  
 11  NumberOfDependents                    146076 non-null  float64
dtypes: float64(4), int64(8)
memory usage: 13.7 MB
```

In [6]:
```
# Nos cargamos columnas inutiles
df.drop(['Unnamed: 0'], axis=1, inplace=True)
```

In [7]:
```
# Nos cargamos duplicados
df.dropna(inplace=True)
```

Preparación datos: Dividimos el dataset

In [8]:
```
from sklearn.model_selection import train_test_split

X = df.drop(['target'], axis=1)
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, rand
```

MiniEDA: Matriz de correlación

Vamos a cargarnos algunas columnas que no estén muy relacionadas con el target (seria más aconsejable descartarlo haciendo un bivariante con test de hipótesis o usando pairplot, aunque hay que tener en cuenta que debe hacerse por partes porque pairplot es muy demandante y este dataset es grande en dato).

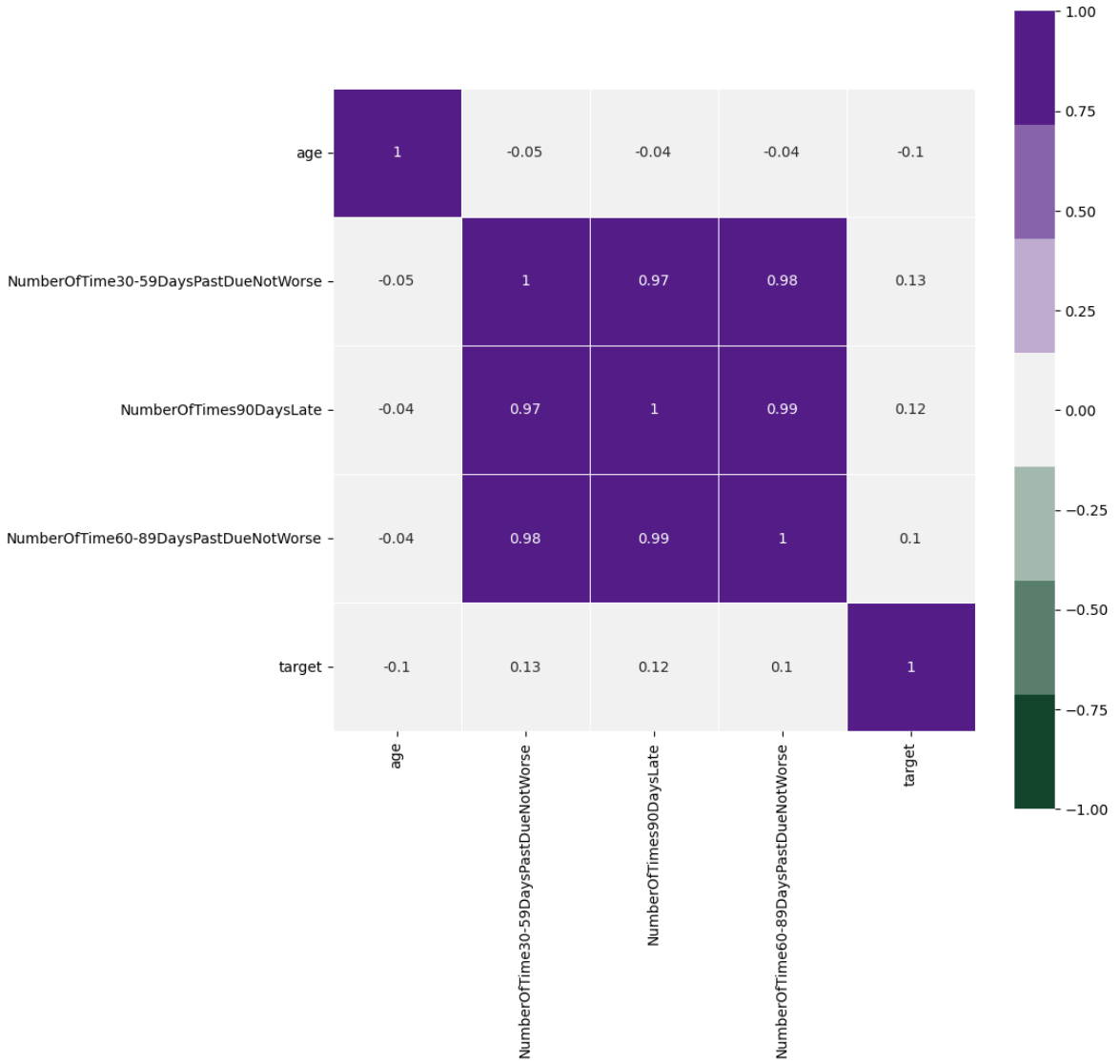
In [9]:
```
df_eda = X_train.copy()
df_eda["target"] = y_train.copy()
```

In [10]:
```
np.abs(df_eda.corr()['target']).sort_values(ascending=False)
```

Out[10]:
```
target                                1.000000
NumberOfTime30-59DaysPastDueNotWorse    0.127577
NumberOfTimes90DaysLate                 0.115376
age                                     0.102266
NumberOfTime60-89DaysPastDueNotWorse    0.098259
NumberOfDependents                      0.046620
NumberOfOpenCreditLinesAndLoans         0.027646
MonthlyIncome                           0.018645
NumberRealEstateLoansOrLines            0.004433
DebtRatio                               0.003192
RevolvingUtilizationOfUnsecuredLines    0.002603
Name: target, dtype: float64
```

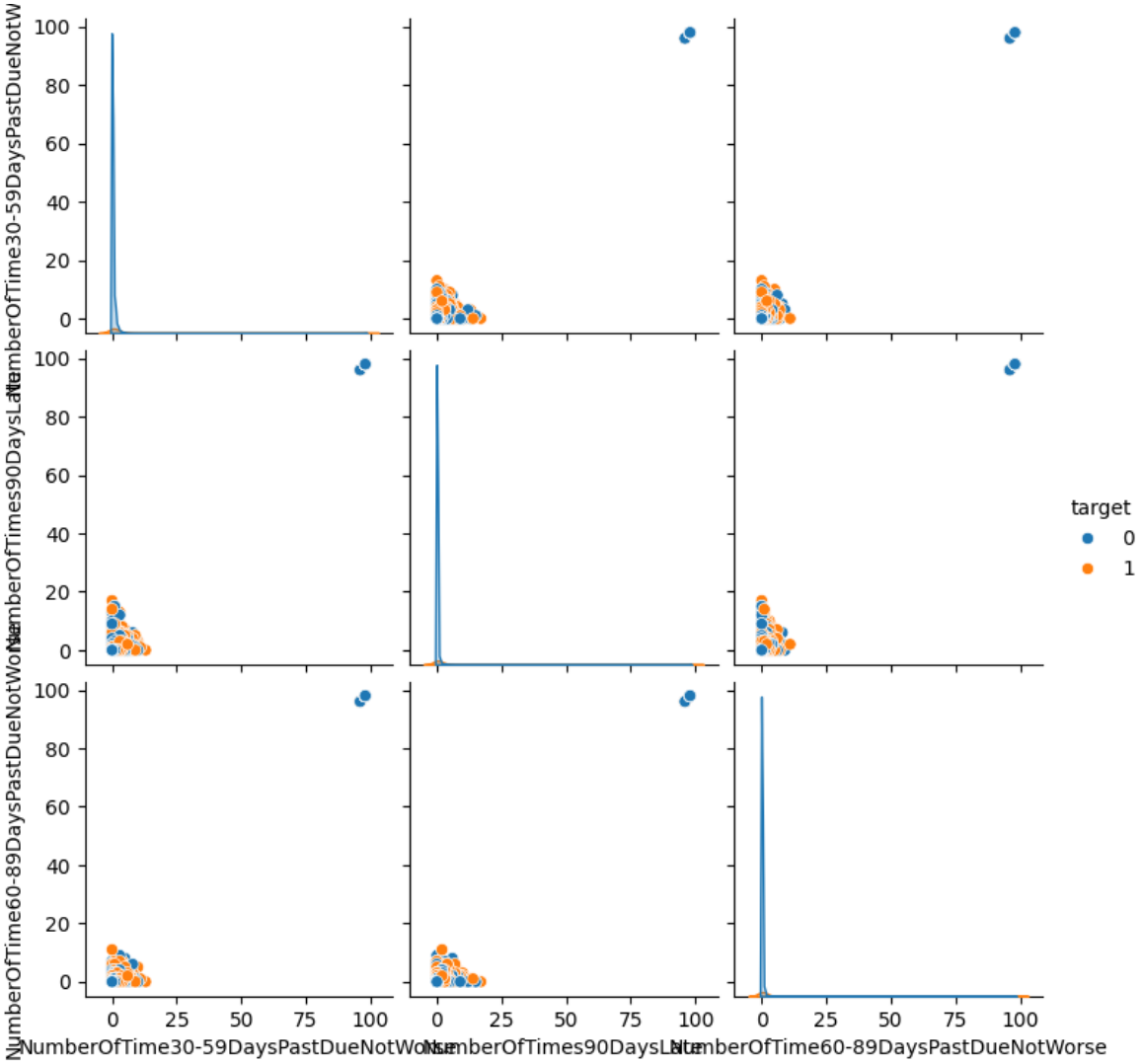
In [11]:
```
corr = np.abs(df_eda.corr()['target']).sort_values(ascending=False)

# Features con menos de 0.1 de correlación vs el target
bad_corr_feat = corr[corr < 0.05].index.values

# Filtramos las columnas buenas
df_eda.drop(columns=bad_corr_feat, inplace=True)
```

In [12]:
```
plt.figure(figsize=(10,10))
sns.heatmap(np.round(df_eda.corr(), 2),
            vmin=-1,
            vmax=1,
            annot=True,
            cmap=sns.diverging_palette(145, 280, s=85, l=25, n=7),
            square=True,
            linewidths=.5);
```



In [13]:
```
sns.pairplot(df_eda[[col for col in df_eda.columns if "NumberOf" in col or col =
```

Out[13]: <seaborn.axisgrid.PairGrid at 0x7efc88ea5f10>



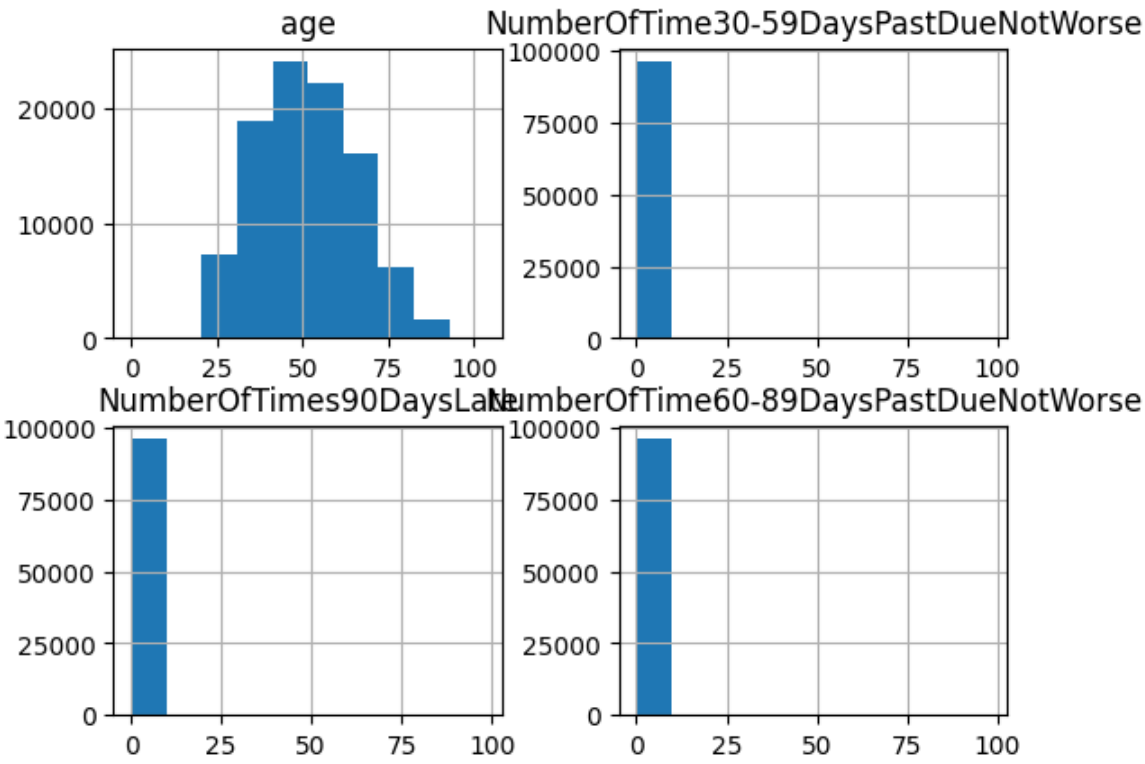
Te invito a usar nuestra librería de funciones gráficas para hacer un bivariante al estilo de las diagonales para las features con la mayor correlación.

In [14]:
```
features_num = [col for col in df_eda.columns if "target" not in col]
X_train = X_train[features_num].copy()
```

Preparación de Features

In [15]:
```
X_train.hist()
```

Out[15]:
```
array([[<Axes: title={'center': 'age'}>,
        <Axes: title={'center': 'NumberOfTime30-59DaysPastDueNotWorse'}>],
       [<Axes: title={'center': 'NumberOfTimes90DaysLate'}>,
        <Axes: title={'center': 'NumberOfTime60-89DaysPastDueNotWorse'}>]],
      dtype=object)
```



In [16]:
```
X_train.describe()
```

Out[16]:

	age	NumberOfTime30-59DaysPastDueNotWorse	NumberOfTimes90DaysLate	Numb 89DaysPastD
count	96215.000000	96215.000000	96215.000000	
mean	51.285174	0.380211	0.210248	
std	14.428079	3.479168	3.444867	
min	0.000000	0.000000	0.000000	
25%	41.000000	0.000000	0.000000	
50%	51.000000	0.000000	0.000000	
75%	61.000000	0.000000	0.000000	
max	103.000000	98.000000	98.000000	

In [17]:
```
X_train[X_train["NumberOfTime30-59DaysPastDueNotWorse"] > 0]
```

Out[17]:

	age	NumberOfTime30-59DaysPastDueNotWorse	NumberOfTimes90DaysLate	NumberOfTin 89DaysPastDueNotV
44650	41	2	1	
27331	39	1	0	
85530	53	1	0	
106214	45	3	0	
106548	47	2	0	
...	...	...	...	
934	45	1	0	
51349	30	3	0	
55095	61	2	0	
46514	78	1	0	
1052	59	6	2	

16182 rows × 4 columns

Podríamos hacer categóricas de las "veces en días vencidos", pero lo vamos a dejar así por espacio, no son variables para escalar necesariamente.

Modelo

In [18]:
```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(max_iter = 5000)

log_reg.fit(X_train, y_train)
```

Out[18]:
```
LogisticRegression(max_iter=5000)
```

Veremos los hiperparámetros (en este caso argumentos al constructor de la clase LogisticRegression) a medida que vayamos usando el modelo. Ahora tratemos "max_iter":

`max_iter` = numero de iteraciones del optimizador (recuerda que el gradiente descendente va probando combinaciones hasta que encuentra una que no mejora la función de minimización después de varios intentos), esta es otra forma de decirle el máximo número de combinaciones, de veces que se aplica el gradiennte descendente en caso de que no se llegue a la condición anterior (denominada tolerancia)

In [19]:
```
print(log_reg.coef_)
print(log_reg.intercept_)
print(log_reg.classes_)
```

```
[[-0.02789161  0.50545547  0.43901075 -0.9050834 ]]
[-1.46237341]
[0 1]
```

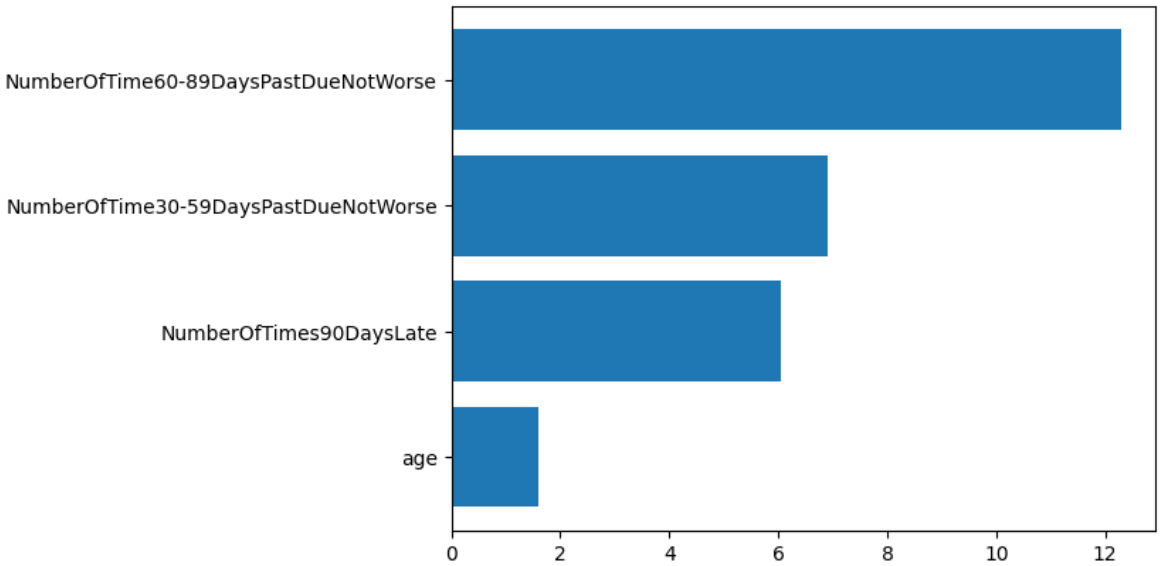
Feature importance

In [20]:
```
intercept = log_reg.intercept_
coefs = log_reg.coef_.ravel()

features = pd.DataFrame(coefs, X_train.columns, columns=['coefficient']).copy()
features['coefficient'] = np.abs(features['coefficient'])

features.sort_values('coefficient', ascending=False).head()
```

Out[20]:

	coefficient
NumberOfTime60-89DaysPastDueNotWorse	0.905083
NumberOfTime30-59DaysPastDueNotWorse	0.505455
NumberOfTimes90DaysLate	0.439011
age	0.027892

In [22]:
```
stdevs = []
for i in X_train.columns:
    stdev = df[i].std()
    stdevs.append(stdev)

features["stdev"] = np.array(stdevs).reshape(-1,1)
features["importance"] = features["coefficient"] * features["stdev"]
features['importance_standarized'] = features['importance'] / y_train.std()

features = features.sort_values('importance_standarized', ascending=True)
plt.barh(features.index, features.importance_standarized);
```



Accuracy

Antes de usar el X_test hay que hacerles las mismas transformaciones que le hayamos hecho al X_train

In [23]:
```
X_test = X_test[X_train.columns].copy()
```

In [24]:
```
from sklearn.metrics import accuracy_score

acc_train = round(accuracy_score(log_reg.predict(X_train), y_train), 3)
acc_test = round(accuracy_score(log_reg.predict(X_test), y_test), 3)

print("Accuracy train:", acc_train)
print("Accuracy test:", acc_test)
```

```
Accuracy train: 0.932
Accuracy test: 0.929
```

¿Es buenos nuestro clasificador?

In [25]:
```
y_train.value_counts(True)
```

Out[25]:
```
target
0    0.93079
1    0.06921
Name: proportion, dtype: float64
```

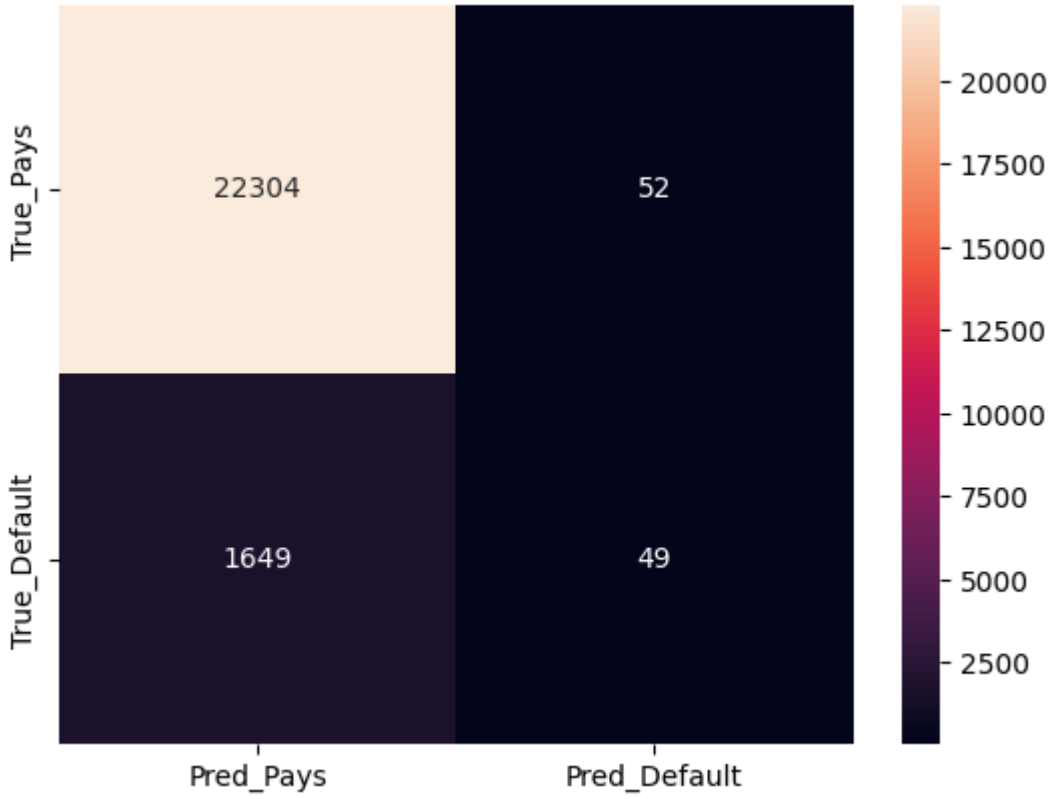
In [26]:
```
y_test.value_counts(True)
```

Out[26]:
```
target
0    0.929409
1    0.070591
Name: proportion, dtype: float64
```

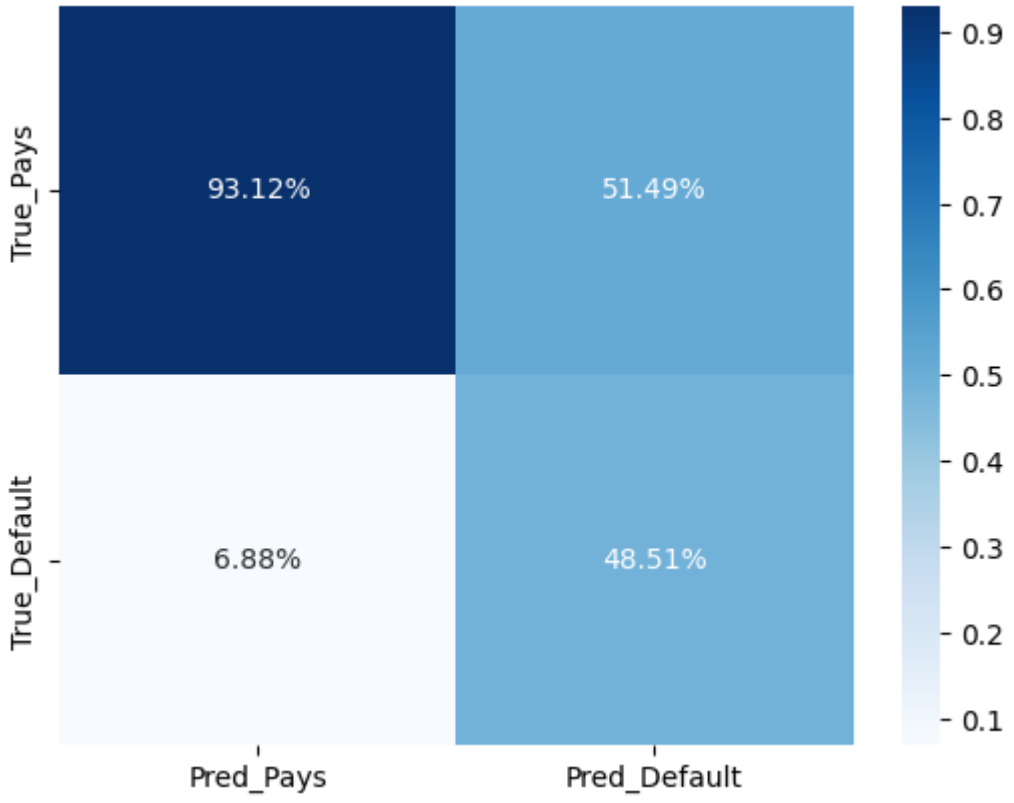
No, no lo es. Tendríamos que tratarlo de alguna forma mejor (Está muy desbalanceado) o ver otras métricas

Confusion matrix

In [27]:
```
from sklearn.metrics import confusion_matrix

c_matrix = confusion_matrix(y_test, log_reg.predict(X_test))

c_matrix_df = pd.DataFrame(c_matrix, columns = ['Pred_Pays', 'Pred_Default'],
                           index = ['True_Pays', 'True_Default'])

sns.heatmap(c_matrix_df, annot=True, fmt='g');
```



In [28]:
```
sns.heatmap(c_matrix_df/np.sum(c_matrix_df), annot=True,
            fmt='.2%', cmap='Blues');
```



Classification report

In [29]:
```
from sklearn.metrics import classification_report
from pprint import pprint
```

```
print(classification_report(y_test, log_reg.predict(X_test)))
```

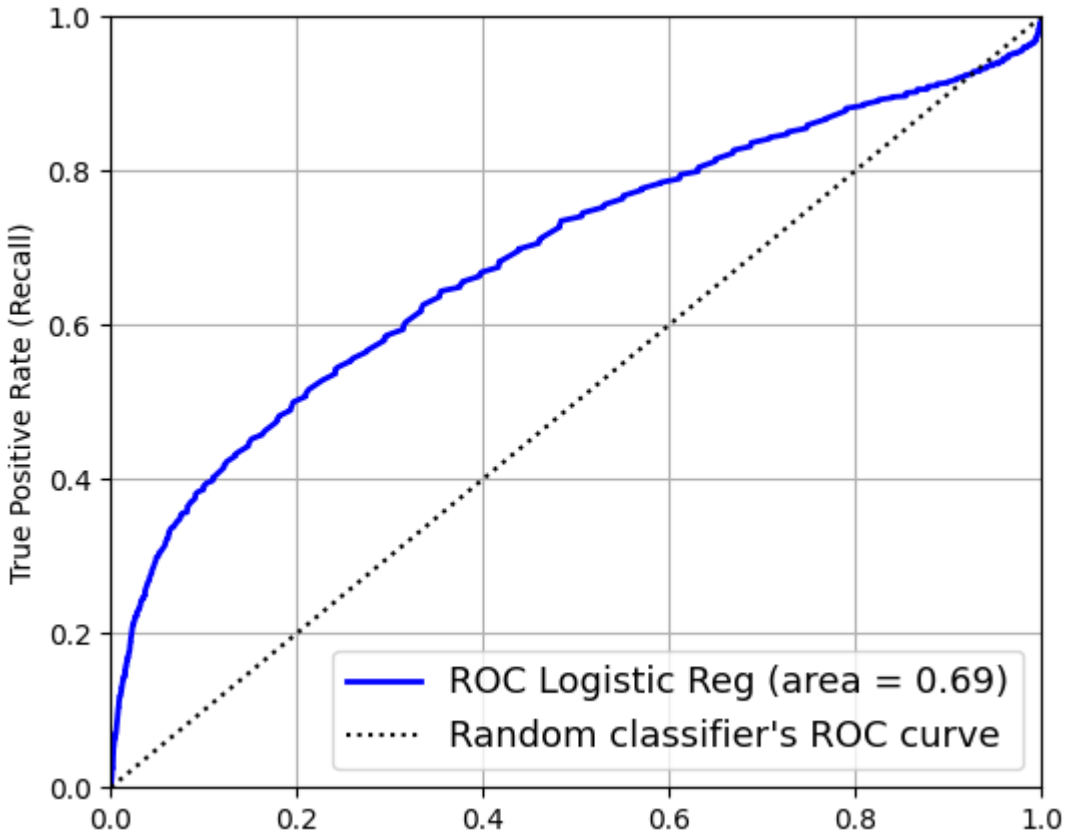
```
              precision    recall  f1-score   support

           0       0.93      1.00      0.96     22356
           1       0.49      0.03      0.05      1698

    accuracy                           0.93     24054
   macro avg       0.71      0.51      0.51     24054
weighted avg       0.90      0.93      0.90     24054
```

ROC Curve

In [30]:
```
### Usando matplotlib
from sklearn.metrics import roc_curve, auc
scores = log_reg.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, scores[:,1])
roc_auc = auc(fpr,tpr)
print("AUROC: %.2f" %(roc_auc))
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, linewidth=2, color= "blue", label=f"ROC Logistic Reg (area =
plt.plot([0, 1], [0, 1], 'k:', label="Random classifier's ROC curve")
plt.ylabel('True Positive Rate (Recall)')
plt.grid()
plt.axis([0, 1, 0, 1])
plt.legend(loc="lower right", fontsize=13)
```

```
AUROC: 0.69
```

Out[30]: <matplotlib.legend.Legend at 0x7efc7d9d8430>



In [ ]: