In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import nltk
from nltk.corpus import stopwords
#Preprocesamiento
import re, string, nltk
from sklearn.feature_extraction.text import TfidfVectorizer
#experimentos modelos
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
```

In [2]:
```
#!pip install wordcloud
```

In [2]:
```
df=pd.read_excel("datasetv2.xlsx")
```

In [3]:
```
df.sample(10)
```

Out[3]:

	id	message	target
6639	6641	RECARGA HOY desde S/3 y LLEVATE ADICIONAL 50GB...	spam
2533	2535	Corto pero lindo: Sé una buena persona	legit
3604	3606	Estoy organizando una reunión para el equipo d...	spam
8408	8410	Yayaya muy bien	legit
7416	7418	Su envío BCP está detenido, confirme dirección...	spam
2134	2136	Borré el mensaje por accidente. Reenvíalo, por...	legit
5171	5173	Meta del 2023: escuchar musica SIN GASTAR MEGA...	spam
2042	2044	Banco de la Nación: comisiones de cuenta de ah...	legit
4303	4305	Hola, te hemos invitado a XCHAT. ¡Este es nues...	spam
846	847	¿Adónde vamos a comprar? Solo compra cuando ll...	legit

In [4]:
```
df.shape
```

Out[4]:
```
(8457, 3)
```

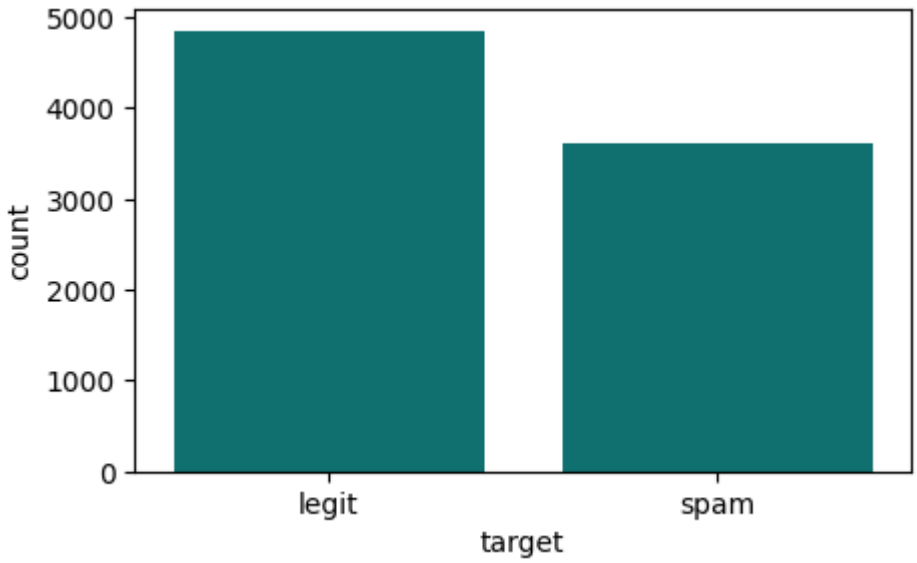
In [5]:
```
df.dtypes
```

Out[5]:
```
id          int64
message    object
target     object
dtype: object
```

In [6]:
```
df.target.value_counts()
```

Out[6]:
```
target
legit    4853
spam     3604
Name: count, dtype: int64
```

In [7]:
```
plt.figure(figsize=(5,3))
sns.barplot(df.target.value_counts(),color="teal")
plt.show()
```



In [8]:
```
df.isnull().sum()
```

Out[8]:
```
id         0
message    0
target     0
dtype: int64
```

In [9]:
```
df=df.drop('id',axis=1)
```

In [10]:
```
df.duplicated().sum()
```

Out[10]:
```
np.int64(181)
```

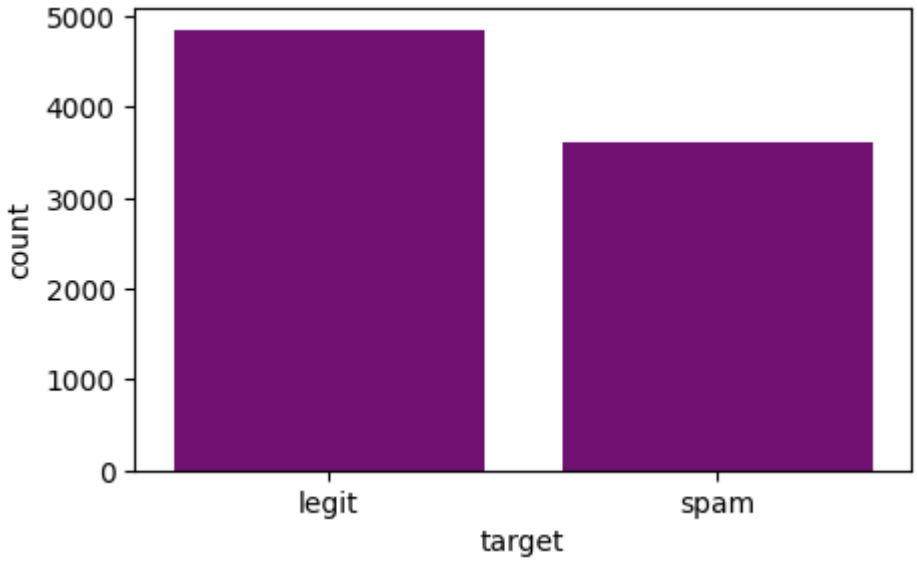
In [11]:
```
df.shape
```

Out[11]:
```
(8457, 2)
```

In [12]:
```
df.target.value_counts()
```

Out[12]:
```
target
legit    4853
spam     3604
Name: count, dtype: int64
```

In [13]:
```
plt.figure(figsize=(5,3))
sns.barplot(df.target.value_counts(),color="purple")
plt.show()
```



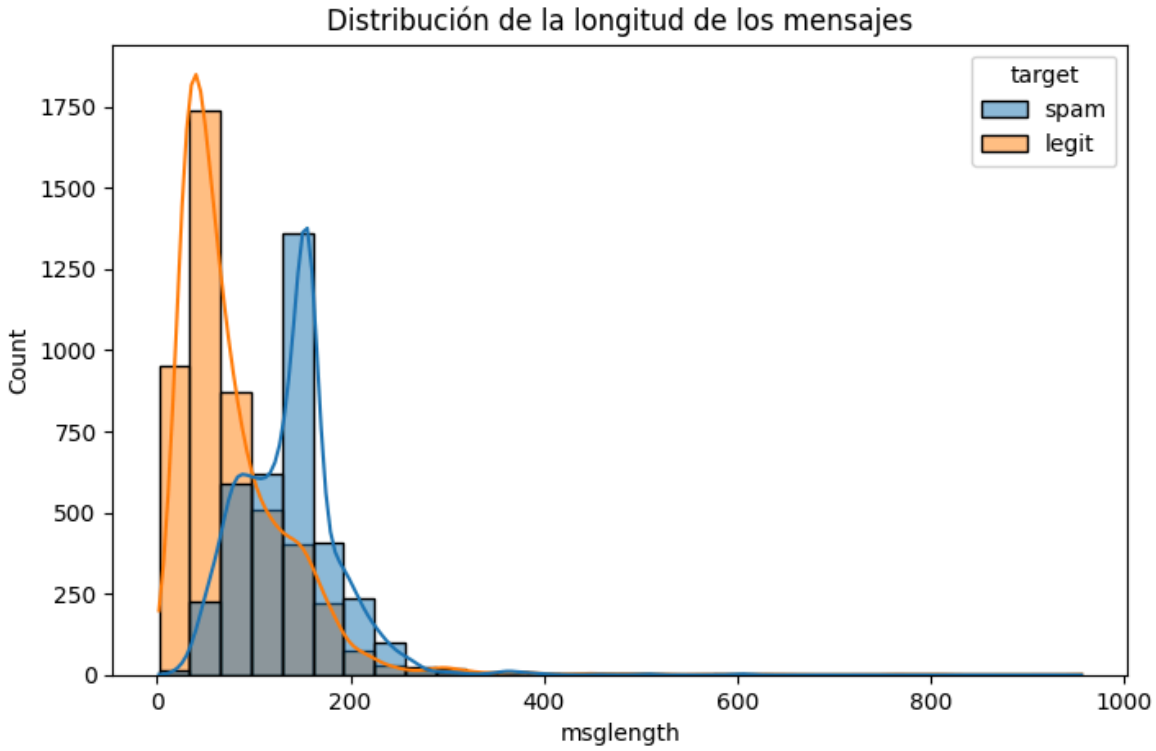
In [14]:
```
df.sample(5)
```

Out[14]:

	message	target
3618	Estudias solo, sin ayuda de nadie. Si no puede...	legit
6916	Se suponía que me despertarías >:(	legit
5604	No sé. Sigues diciendo que no, pero desde que ...	legit
7467	SUBASTA SMS. ¡Has ganado un Nokia 7250i! Esto ...	spam
152	¡¡¡SIN REGALOS!!! ¿Intentas que me tire por un...	legit

In [15]:
```
# Longitud de mensajes
df["message"] = df["message"].astype(str)
df["msglength"] = df["message"].apply(len)
```

In [16]:
```
plt.figure(figsize=(8,5))
sns.histplot(data=df, x="msglength", hue="target", bins=30, kde=True)
plt.title("Distribución de la longitud de los mensajes")
plt.show()
```



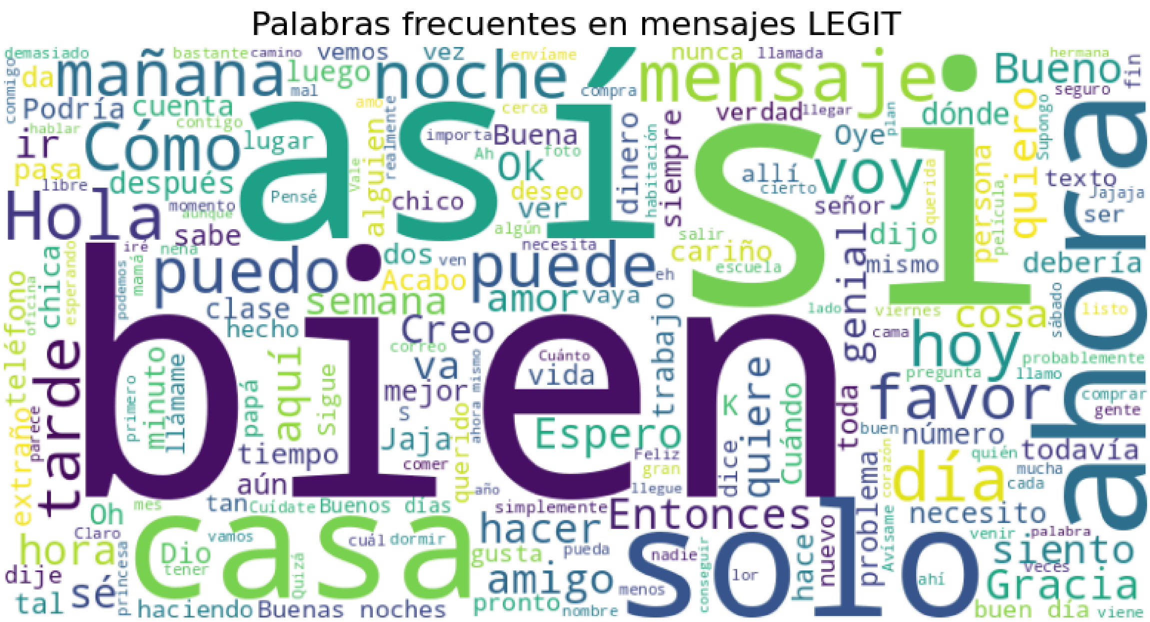
In [17]:
```
# Descargar stopwords
nltk.download("stopwords")
stopwords_es = set(stopwords.words("spanish"))
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\AngelRC2\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [18]:
```
# Función para generar wordcloud
def get_wordcloud(text, title):
    wc = WordCloud(
        width=800,
        height=400,
        stopwords=stopwords_es,
        background_color="white",
        colormap="viridis"
    ).generate(" ".join(text))

    plt.figure(figsize=(10,6))
    plt.imshow(wc, interpolation="bilinear")
    plt.axis("off")
    plt.title(title, fontsize=16)
    plt.show()
```

In [20]:
```
# Wordcloud para spam
spam_text = df[df["target"]=="spam"]["message"]
get_wordcloud(spam_text, "Palabras frecuentes en mensajes SPAM")
```


```
In [22]: # Wordcloud para ham (si hay ejemplos en el dataset)
legittext = df[df["target"]=="legit"]["message"]
get_wordcloud(legittext, "Palabras frecuentes en mensajes LEGIT")
```



Preprocesamiento

1. Pasar todo el texto a minúsculas.
2. Eliminar números, signos de puntuación y caracteres especiales.
3. Quitar stopwords en español.
4. Tokenizar (separar en palabras).
5. Aplicar TF-IDF para transformar el texto en vectores numéricos.

```
In [23]: nltk.download("stopwords")
stopwords_es = set(stopwords.words("spanish"))
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\AngelRC2\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [24]:
```
# --- Función de limpieza ---
def clean_text(text):
    text = text.lower()  # minúsculas
    text = re.sub(r'\d+', '', text)  # eliminar números
    text = text.translate(str.maketrans('', '', string.punctuation))  # quitar p
    tokens = text.split()  # tokenizar
    tokens = [w for w in tokens if w not in stopwords_es]  # quitar stopwords
    return " ".join(tokens)
```

In [25]:
```
# Aplicar limpieza
df["clean_message"] = df["message"].astype(str).apply(clean_text)
```

In [26]:
```
df[["message", "clean_message"]].sample(5)
```

Out[26]:

	message	clean_message
6600	Quizás podría sacar el libro y devolverlo inme...	quizás podría sacar libro devolverlo inmediata...
5238	Mira los videos de Choose Your Babe en sms.shs...	mira videos choose your babe smsshsexnetun fgk...
3350	Está bien, voy a tomar algo o algo. ¿Quieres q...	bien voy tomar ¿quieres vaya buscarte
6501	Q operadora eres ?	q operadora
7412	Su envío Banco de la Nación está detenido, con...	envío banco nación detenido confirme dirección...

In [27]:
```
# --- TF-IDF ---
vectorizer = TfidfVectorizer(max_features=3000)  # limitar a 3000 features
X = vectorizer.fit_transform(df["clean_message"])
y = df["target"]
```

In [28]:
```
print("Shape de X:", X.shape)
```

```
Shape de X: (8457, 3000)
```

Experimento con modelos de ML

In [29]:
```
# Train/Test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

In [30]:
```
# Modelos a comparar
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Extra Trees": ExtraTreesClassifier(random_state=42),
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "SVM": SVC(kernel="linear", random_state=42),
```

```
    "Naive Bayes": MultinomialNB(),
    "SGD": SGDClassifier(random_state=42)
}
```

Entrenar y evaluar

In [31]:
```
results = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    report = classification_report(y_test, y_pred, output_dict=True, zero_divisi
    results[name] = {
        "Accuracy": report["accuracy"],
        "Precision": report["weighted avg"]["precision"],
        "Recall": report["weighted avg"]["recall"],
        "F1-score": report["weighted avg"]["f1-score"]
    }
    print(f"\n📌 Modelo: {name}")
    print(classification_report(y_test, y_pred, zero_division=0))
```

📌 Modelo: Decision Tree

	precision	recall	f1-score	support
legit	0.93	0.92	0.92	971
spam	0.89	0.91	0.90	721
accuracy			0.91	1692
macro avg	0.91	0.91	0.91	1692
weighted avg	0.91	0.91	0.91	1692

📌 Modelo: Random Forest

	precision	recall	f1-score	support
legit	0.94	0.95	0.94	971
spam	0.93	0.91	0.92	721
accuracy			0.93	1692
macro avg	0.93	0.93	0.93	1692
weighted avg	0.93	0.93	0.93	1692

📌 Modelo: Extra Trees

	precision	recall	f1-score	support
legit	0.95	0.94	0.95	971
spam	0.92	0.93	0.93	721
accuracy			0.94	1692
macro avg	0.94	0.94	0.94	1692
weighted avg	0.94	0.94	0.94	1692

📌 Modelo: Logistic Regression

	precision	recall	f1-score	support
legit	0.94	0.96	0.95	971
spam	0.95	0.92	0.93	721
accuracy			0.94	1692
macro avg	0.95	0.94	0.94	1692
weighted avg	0.94	0.94	0.94	1692

📌 Modelo: SVM

	precision	recall	f1-score	support
legit	0.94	0.95	0.95	971
spam	0.93	0.93	0.93	721
accuracy			0.94	1692
macro avg	0.94	0.94	0.94	1692
weighted avg	0.94	0.94	0.94	1692

📌 Modelo: Naive Bayes

	precision	recall	f1-score	support
legit	0.96	0.94	0.95	971
spam	0.92	0.94	0.93	721

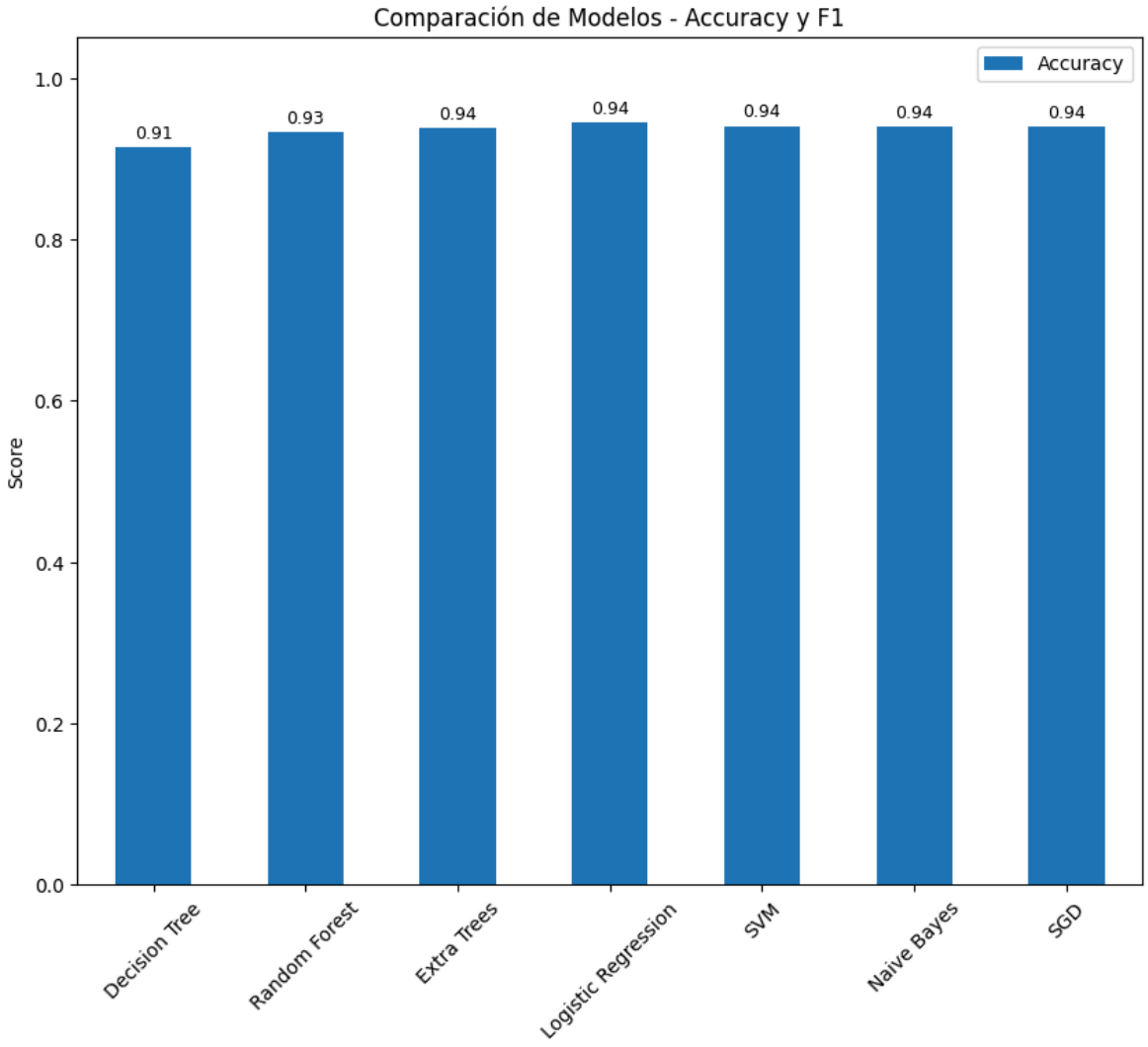
```
    accuracy                           0.94      1692
   macro avg       0.94      0.94      0.94      1692
weighted avg       0.94      0.94      0.94      1692
```

```
📌 Modelo: SGD
              precision    recall  f1-score   support

       legit       0.94      0.95      0.95       971
        spam       0.93      0.92      0.93       721

    accuracy                           0.94      1692
   macro avg       0.94      0.94      0.94      1692
weighted avg       0.94      0.94      0.94      1692
```

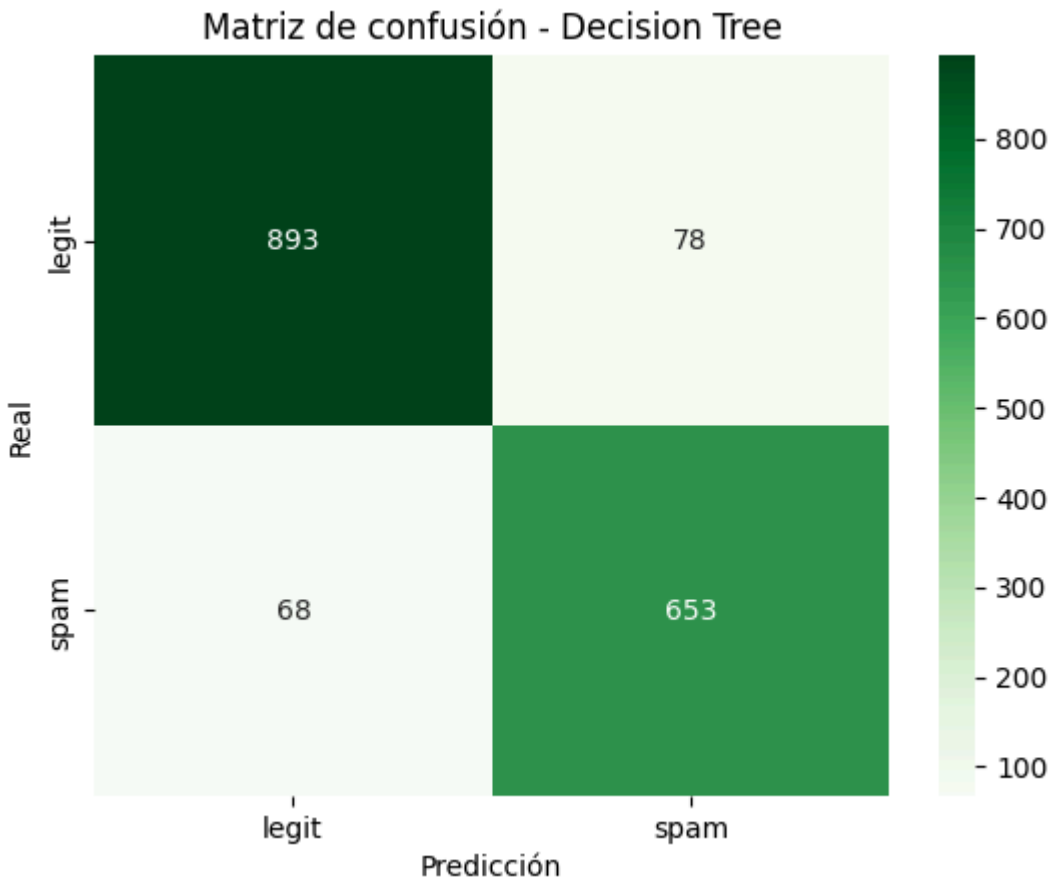
In [32]:
```
# Comparación de resultados
results_df = pd.DataFrame(results).T
print("\n📊 Comparación de Modelos:")
print(results_df)
```

```
📊 Comparación de Modelos:
                     Accuracy  Precision    Recall  F1-score
Decision Tree        0.913712   0.913924  0.913712  0.913785
Random Forest        0.932624   0.932565  0.932624  0.932561
Extra Trees          0.937352   0.937481  0.937352  0.937395
Logistic Regression  0.944444   0.944617  0.944444  0.944301
SVM                  0.940307   0.940263  0.940307  0.940269
Naive Bayes          0.939125   0.939703  0.939125  0.939238
SGD                  0.939125   0.939080  0.939125  0.939086
```

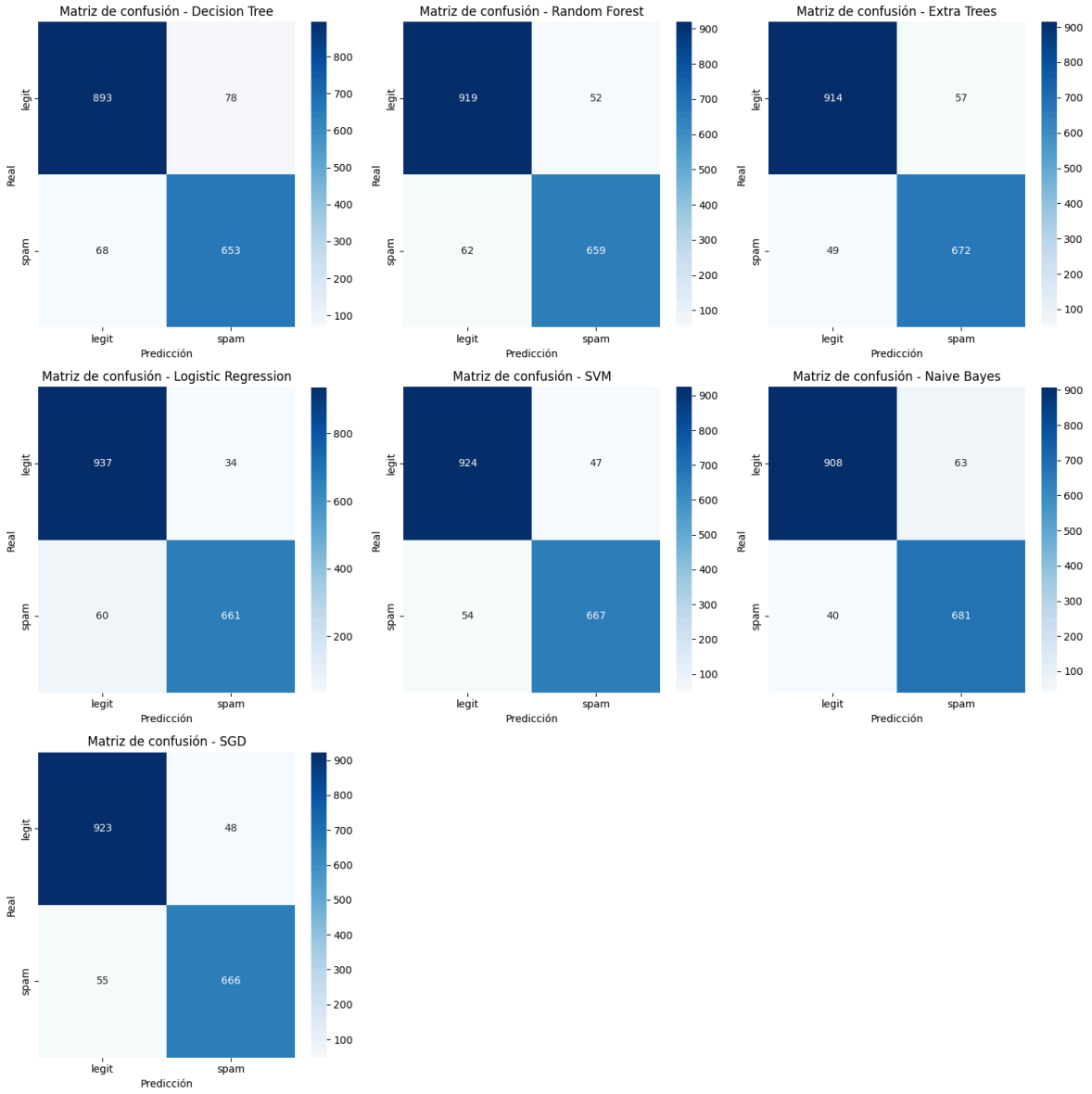
In [33]:
```
# Visualización comparativa con anotaciones de score
ax = results_df[["Accuracy"]].plot(kind="bar", figsize=(10,8))
plt.title("Comparación de Modelos - Accuracy y F1")
plt.ylabel("Score")
plt.xticks(rotation=45)
plt.ylim(0, 1.05)

# Agregar los valores arriba de cada barra
for p in ax.patches:
    ax.annotate(f"{p.get_height():.2f}",
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom', fontsize=9, color="black", xytext=(0,3
                textcoords="offset points")
plt.show()
```



In [34]:
```
# Matriz de confusión de Decision Tree
best_model = DecisionTreeClassifier(random_state=42)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Greens",
            xticklabels=best_model.classes_,
            yticklabels=best_model.classes_)
plt.title("Matriz de confusión - Decision Tree")
plt.xlabel("Predicción")
plt.ylabel("Real")
plt.show()
```



In [37]:
```
# Definir número de filas y columnas para los subplots
n_models = len(models)
n_cols = 3   # número de columnas
n_rows = (n_models + n_cols - 1) // n_cols  # calcular filas necesarias

plt.figure(figsize=(15, 5*n_rows))

for i, (name, model) in enumerate(models.items(), 1):
    # Entrenar modelo
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Matriz de confusión
    cm = confusion_matrix(y_test, y_pred)

    # Graficar en subplot
    plt.subplot(n_rows, n_cols, i)
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=model.classes_, yticklabels=model.classes_)
    plt.title(f"Matriz de confusión - {name}")
    plt.xlabel("Predicción")
    plt.ylabel("Real")

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



MODELO DE ENSAMBLE SVM+RF

VotingClassifier

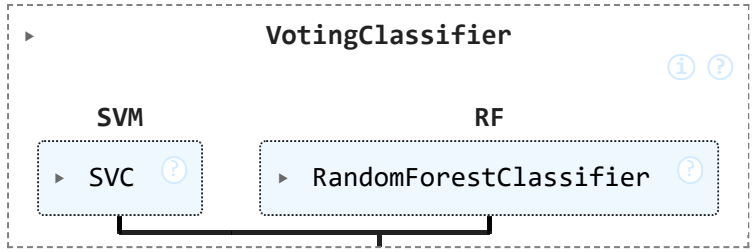
In [38]:
```
from sklearn.ensemble import VotingClassifier
```

In [39]:
```
# Definir modelos base
svm_clf = SVC(kernel="linear", probability=True, random_state=42)
rf_clf = RandomForestClassifier(random_state=42)
```

In [40]:
```
# Ensamble con VotingClassifier
ensemble = VotingClassifier(
    estimators=[("SVM", svm_clf), ("RF", rf_clf)],
    voting="soft"   # "hard" = votación mayoritaria, "soft" = promedio de probab
)
```

In [41]:
```
# Entrenamiento
ensemble.fit(X_train, y_train)
```

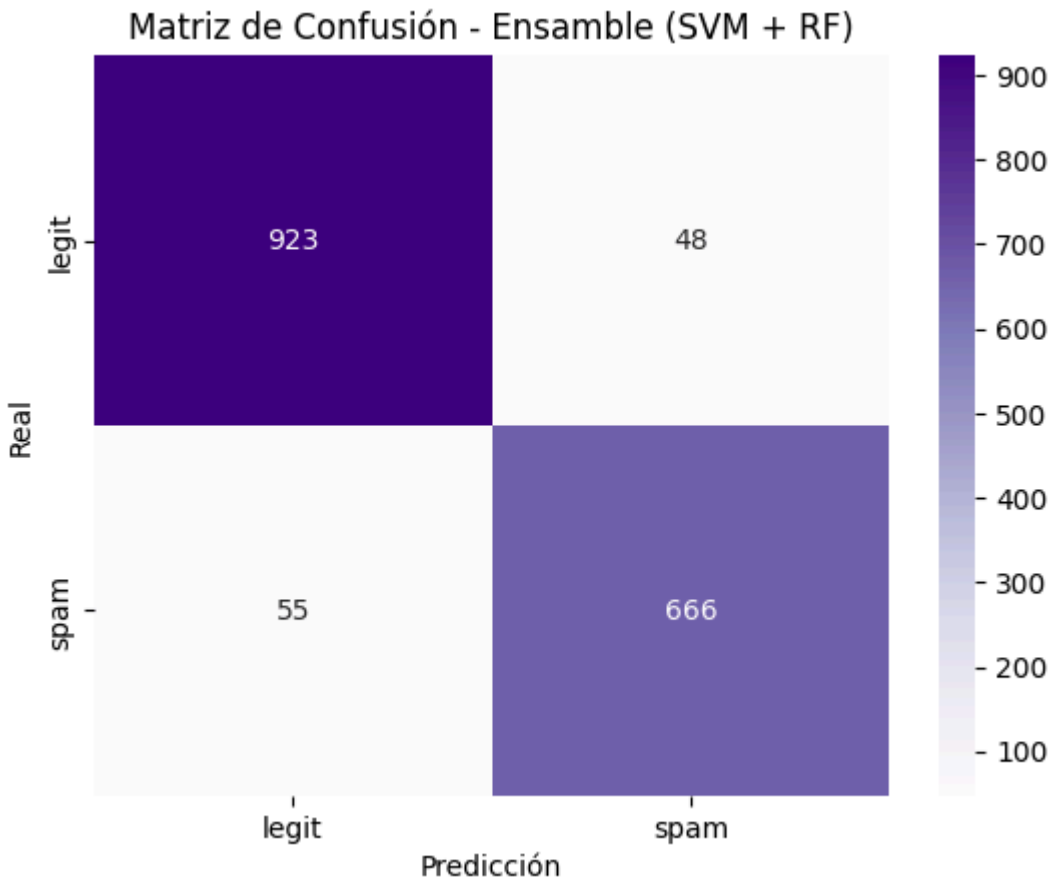
Out[41]:



In [42]:
```
# Evaluación
print("\n📌 Modelo de Ensamble (SVM + Random Forest)")
print(classification_report(y_test, y_pred, zero_division=0))
```

```
📌 Modelo de Ensamble (SVM + Random Forest)
              precision    recall  f1-score   support

       legit       0.94      0.95      0.95       971
        spam       0.93      0.92      0.93       721

    accuracy                           0.94      1692
   macro avg       0.94      0.94      0.94      1692
weighted avg       0.94      0.94      0.94      1692
```

In [43]:
```
# Matriz de confusión
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Purples",
            xticklabels=ensemble.classes_, yticklabels=ensemble.classes_)
plt.title("Matriz de Confusión - Ensamble (SVM + RF)")
plt.xlabel("Predicción")
plt.ylabel("Real")
plt.show()
```



StackingClassifier

In [44]:
```
from sklearn.ensemble import StackingClassifier
```

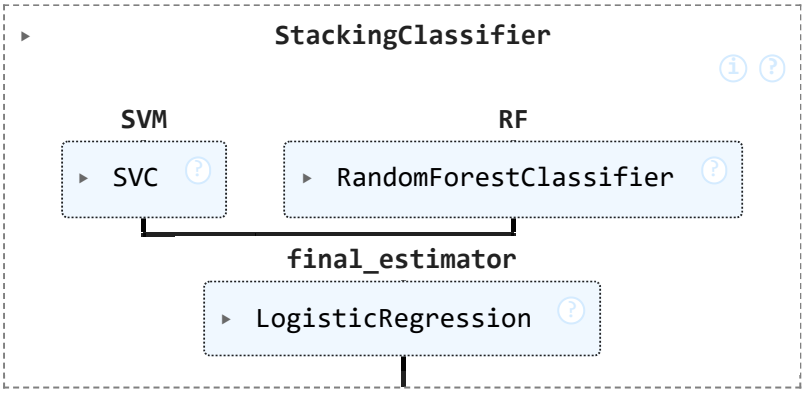
In [45]:
```
# Modelos base
svm_clf = SVC(kernel="linear", probability=True, random_state=42)
rf_clf = RandomForestClassifier(random_state=42)
```

In [46]:
```
# Meta-modelo
meta_model = LogisticRegression(max_iter=1000, random_state=42)
```

In [47]:
```
# StackingClassifier
stacking_clf = StackingClassifier(
    estimators=[("SVM", svm_clf), ("RF", rf_clf)],
    final_estimator=meta_model,
    stack_method="predict_proba",  # usar probabilidades en la combinación
    passthrough=False,  # si True agrega features originales junto a las predicc
    n_jobs=-1
)
```

In [48]:
```
# Entrenamiento
stacking_clf.fit(X_train, y_train)
```

Out[48]:



In [49]:
```
# Evaluación
print("\n📌 Modelo de Ensamble - Stacking (SVM + RF -> Logistic Regression)")
print(classification_report(y_test, y_pred, zero_division=0))
```

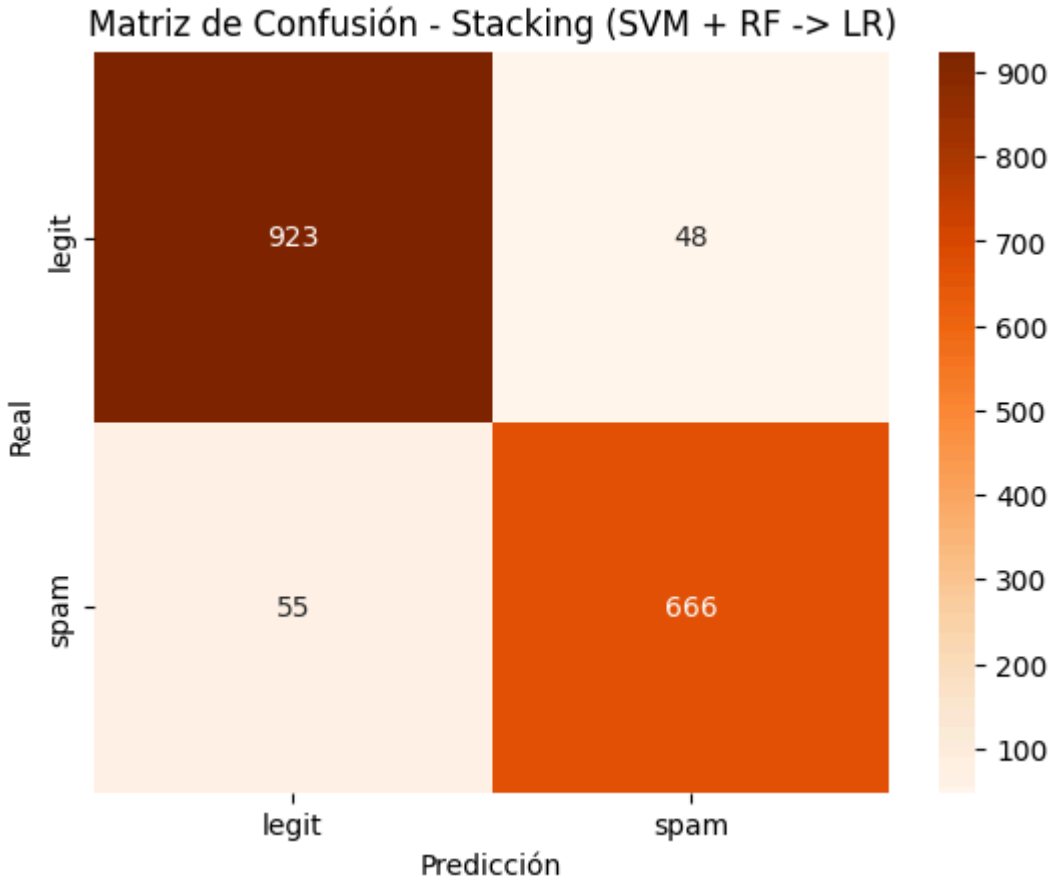
```
📌 Modelo de Ensamble - Stacking (SVM + RF -> Logistic Regression)
              precision    recall  f1-score   support

       legit       0.94      0.95      0.95       971
        spam       0.93      0.92      0.93       721

    accuracy                           0.94      1692
   macro avg       0.94      0.94      0.94      1692
weighted avg       0.94      0.94      0.94      1692
```

In [50]:
```
# Evaluación
print("\n📌 Modelo de Ensamble - Stacking (SVM + RF -> Logistic Regression)")
print(classification_report(y_test, y_pred, zero_division=0))

# Matriz de confusión
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Oranges",
```

```
            xticklabels=stacking_clf.classes_, yticklabels=stacking_clf.classes_
plt.title("Matriz de Confusión - Stacking (SVM + RF -> LR)")
plt.xlabel("Predicción")
plt.ylabel("Real")
plt.show()
```

📌 Modelo de Ensamble - Stacking (SVM + RF -> Logistic Regression)

	precision	recall	f1-score	support
legit	0.94	0.95	0.95	971
spam	0.93	0.92	0.93	721
accuracy			0.94	1692
macro avg	0.94	0.94	0.94	1692
weighted avg	0.94	0.94	0.94	1692



Estrategias para mejorar el modelo:

SMOTE

In [51]:
```
from imblearn.over_sampling import SMOTE
from collections import Counter
```

In [52]:
```
print("🔎 Distribución original:", Counter(y_train))
```

🔎 Distribución original: Counter({'legit': 3882, 'spam': 2883})

In [53]:
```
# Aplicar SMOTE solo al set de entrenamiento
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

```
print("✅ Distribución balanceada con SMOTE:", Counter(y_train_res))
```

```
✅ Distribución balanceada con SMOTE: Counter({'legit': 3882, 'spam': 3882})
```

Voting Classifier (Ensamble Híbrido) combinando:

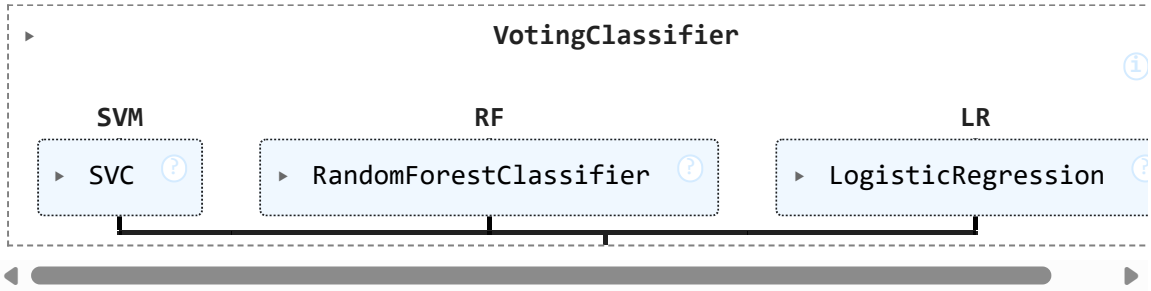
- SVM (lineal o RBF): bueno para separar texto en espacios de alta dimensión.
- Random Forest: robusto ante ruido y capta relaciones no lineales.
- Logistic Regression: modelo base y estable en clasificación binaria.

In [54]:
```
# --- Definición de modelos base ---
svm_clf = SVC(kernel="linear", probability=True, random_state=42)
rf_clf = RandomForestClassifier(n_estimators=200, random_state=42)
lr_clf = LogisticRegression(max_iter=1000, random_state=42)
```

In [55]:
```
# --- Voting Classifier (soft voting para usar probabilidades) ---
voting_clf = VotingClassifier(
    estimators=[("SVM", svm_clf), ("RF", rf_clf), ("LR", lr_clf)],
    voting="soft"
)
```

In [56]:
```
# --- Entrenamiento con dataset balanceado (SMOTE) ---
voting_clf.fit(X_train_res, y_train_res)
```

Out[56]:



In [57]:
```
# --- Predicciones ---
y_pred = voting_clf.predict(X_test)
```

In [58]:
```
# --- Métricas ---
print("🔎 Accuracy:", accuracy_score(y_test, y_pred))
print("\n📊 Reporte de clasificación:\n", classification_report(y_test, y_pred))
print("\n🌀 Matriz de confusión:\n", confusion_matrix(y_test, y_pred))
```

```
🔎 Accuracy: 0.9426713947990544

📊 Reporte de clasificación:
              precision    recall  f1-score   support

       legit       0.96      0.94      0.95       971
        spam       0.92      0.94      0.93       721

    accuracy                           0.94      1692
   macro avg       0.94      0.94      0.94      1692
weighted avg       0.94      0.94      0.94      1692


🌀 Matriz de confusión:
 [[915  56]
 [ 41 680]]
```

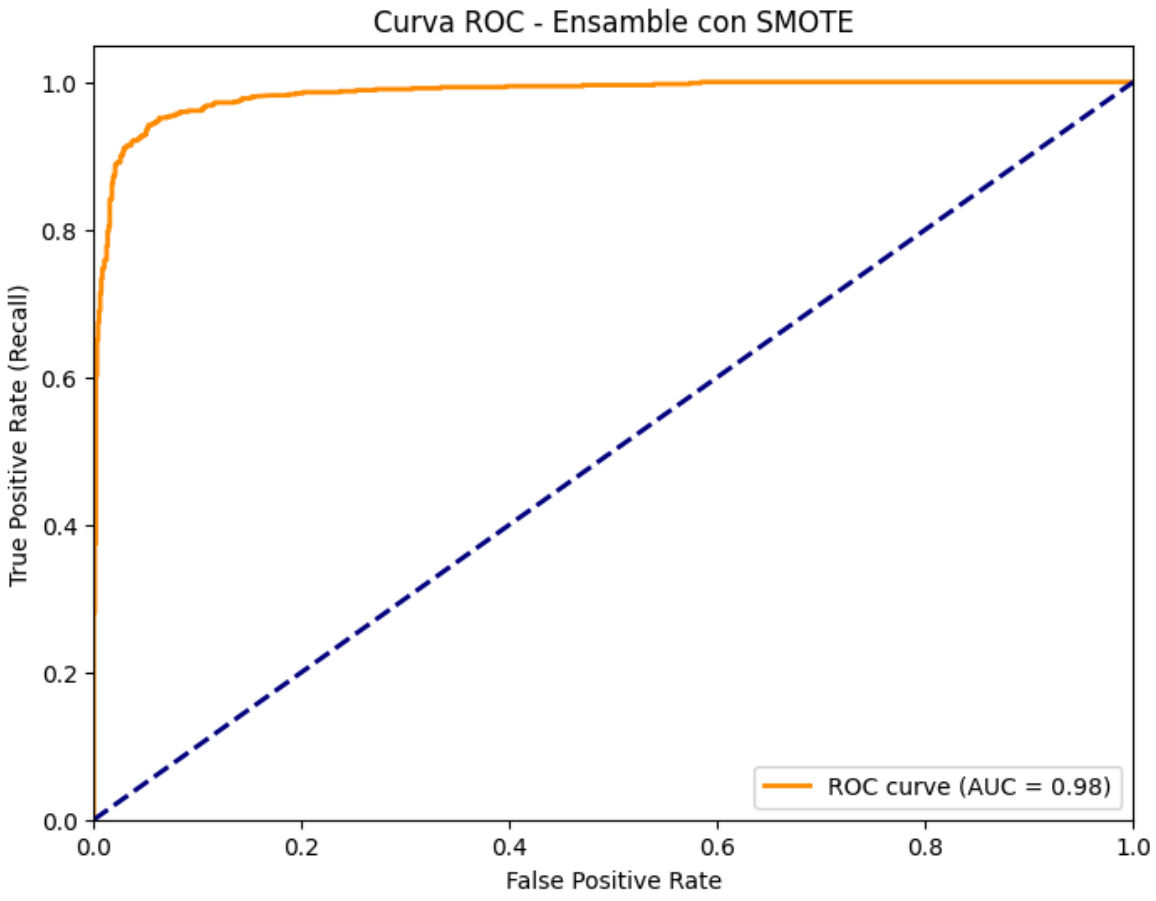
- Matriz de confusión → ver cuántos falsos negativos de spam quedan.
- ROC-AUC → mide capacidad general de separar clases sin importar el threshold.
- PR-AUC (Precisión-Recall) → mucho más útil en casos de desbalance.

In [59]:
```
from sklearn.metrics import roc_auc_score, roc_curve, precision_recall_curve, au
y_proba = voting_clf.predict_proba(X_test)[:,1]
roc_auc = roc_auc_score(y_test, y_proba)
print("ROC-AUC:", roc_auc)
```

```
ROC-AUC: 0.9842556181982056
```

In [60]:
```
# Probabilidades del ensamble (para clase positiva = spam)
y_proba = voting_clf.predict_proba(X_test)[:, 1]
```

In [61]:
```
# --- ROC Curve ---
fpr, tpr, _ = roc_curve(y_test, y_proba, pos_label="spam")
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, color="darkorange", lw=2, label="ROC curve (AUC = %0.2f)" % r
plt.plot([0,1], [0,1], color="navy", lw=2, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate (Recall)")
plt.title("Curva ROC - Ensamble con SMOTE")
plt.legend(loc="lower right")
plt.show()
```



- Eje X (False Positive Rate, FPR) → Proporción de legit que el modelo clasificó como spam (falsos positivos).
- Eje Y (True Positive Rate, Recall, TPR) → Proporción de spam que el modelo detectó correctamente.

1. La curva se eleva rápido hacia arriba (cerca del eje Y) → significa que el modelo logra un alto recall con pocos falsos positivos. (Detecta casi todos los spam sin confundir demasiado los mensajes legítimos).

2. Área bajo la curva (AUC = 0.98)

    - El AUC mide la capacidad general de separación entre clases (0.5 sería azar puro, 1.0 sería perfecto).
    - Un valor de 0.98 indica que el modelo distingue casi perfectamente entre spam y legit.
    - En términos prácticos: si tomas un mensaje spam y uno legit al azar, hay un 98% de probabilidad de que el modelo ordene correctamente cuál es spam y cuál es legit.

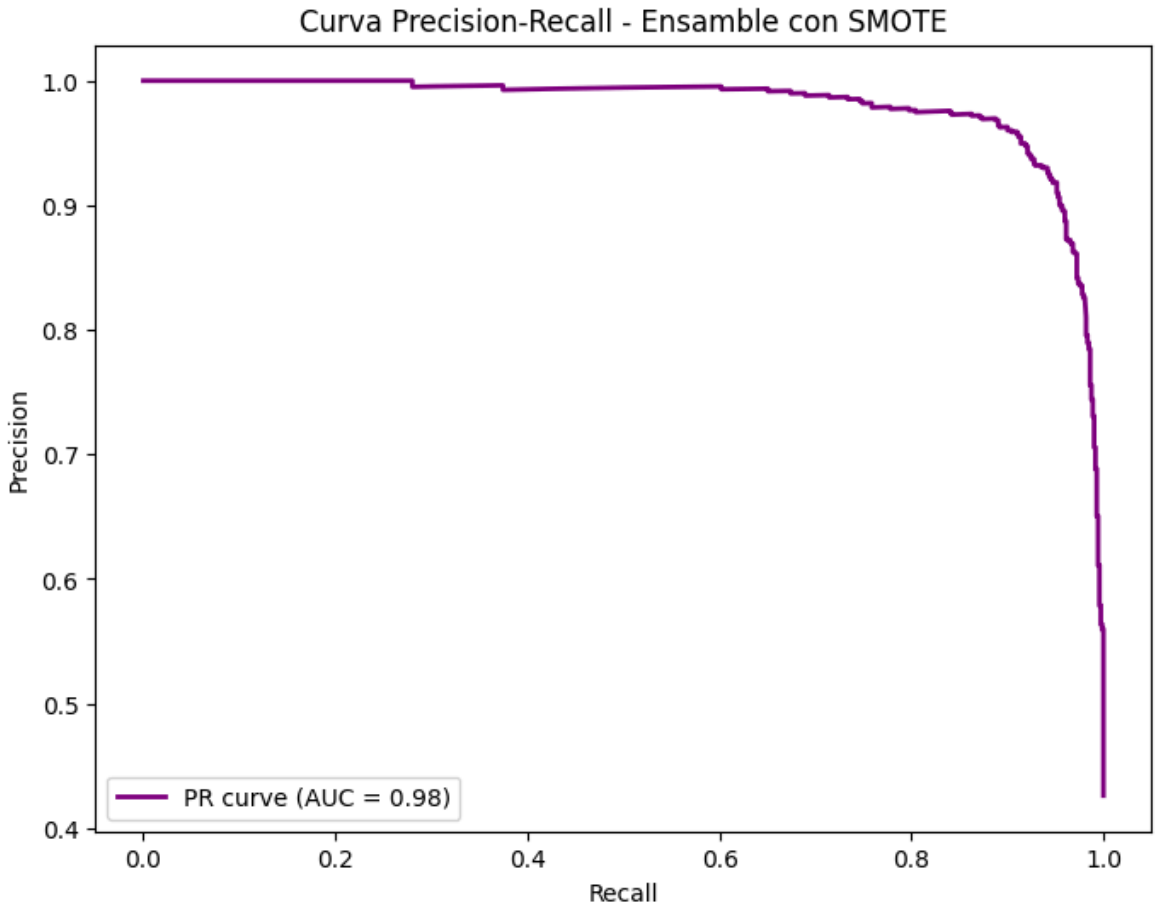
3. La diagonal azul punteada (baseline)

    - Representa un clasificador aleatorio (pura suerte).
    - Tu curva está muy por encima, lo cual confirma que el modelo tiene un rendimiento sobresaliente.

El ensamble con SMOTE logra un modelo muy potente para separar spam de legit. Con un AUC = 0.98, prácticamente roza el desempeño ideal. Significa que puedes ajustar el umbral de decisión según si prefieres minimizar falsos negativos (no dejar escapar spam) o minimizar falsos positivos (no bloquear mensajes legítimos).

In [62]:
```
# --- Precision-Recall Curve ---
precision, recall, _ = precision_recall_curve(y_test, y_proba, pos_label="spam")
pr_auc = auc(recall, precision)
```

In [63]:
```
plt.figure(figsize=(8,6))
plt.plot(recall, precision, color="purple", lw=2, label="PR curve (AUC = %0.2f)"
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Curva Precision-Recall - Ensamble con SMOTE")
plt.legend(loc="lower left")
plt.show()

print("ROC-AUC:", roc_auc)
print("PR-AUC:", pr_auc)
```




```
ROC-AUC: 0.9842556181982056
PR-AUC: 0.9802256006124199
```

In [ ]: