

Universidad Nacional Autónoma de México  
IIMAS  
Programa de ciencia e ingeniería de la computación.

## **Práctica 6**

Preprocesamiento de datos con Python

Preprocesamiento de Datos para Ciencia de Datos  
Dra. María del Pilar Angeles

Presentado por:  
José Rodrigo Moreno López  
Ajitzi Ricardo Quintana Ruiz  
9 de Octubre de 2025

## ✓ Práctica 6 — Preprocesamiento de datos con Python

### Objetivo

Seleccionar las características a través de aprendizaje de máquina y Python con scikit-learn. Se pretende reducir el sobreajuste, mejorar la precisión y reducir el tiempo de entrenamiento.

### Actividades a realizar:

#### Actividad 1: Selección de características/atributos

- Considere escoger las cuatro mejores características no negativas a partir del archivo `pacientes.csv` utilizado en la práctica anterior. Puede utilizar chi-square test y muestre los puntajes calculados para cada atributo y especifique cuales fueron los atributos seleccionados.
- Remueva atributos no relevantes a través de la eliminación recursiva de características. Utilice el algoritmo de regresión logística para seleccionar las 3 características principales. Muestre las características que quedan junto con su puntaje de ranking.
- Utilice la técnica de reducción de datos PCA (Principal Component Analysis) el cual usa algebra lineal para transformar el conjunto de datos. Considere las dimensiones y nuestro archivo transformado.
- Utilice técnicas de clasificación como Random Forest y Extra Trees para estimar la importancia de características. Muestre los puntajes de importancia por cada atributo, así como el nombre de cada atributo.

#### Actividad 2: Evaluar el rendimiento de los algoritmos de aprendizaje de máquina

- Genere muestras de entrenamiento y prueba. Considere el 67% de los datos para entrenamiento y el 33% restante para validación.
- Genere modelo por regresión logística y evalúe su precisión (accuracy) y muestrela.
- Utilice la técnica de validación cruzada para estimar el rendimiento (en términos de la media y la desviación estándar) de la regresión logística. Considere 10 segmentos de datos.
- Utilice la técnica Leave one out cross validation, usando la validación cruzada con un solo fold.

#### Actividad 3: Conserve el programa y los datos

K=1. Muestre la media y la desviación estándar. ¿Cuál es la interpretación de la media y la desviación estándar en estos dos casos?

## ✓ Importar las librerías necesarias

```
import sys
import subprocess

# Verificar e instalar paquetes necesarios
packages = ['pandas', 'numpy', 'matplotlib', 'seaborn', 'scikit-learn']
for p in packages:
    try:
        __import__(p)
    except Exception:
        subprocess.check_call([sys.executable, '-m', 'pip', 'install', p])

print('Dependencias comprobadas')
```

Dependencias comprobadas

```
# Importar librerías
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectKBest, chi2, RFE
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score, LeaveOneOut
from sklearn.metrics import accuracy_score
```

```
import warnings
warnings.filterwarnings('ignore')

# Configuración para gráficos
plt.style.use('default')
sns.set_palette("husl")

print('Librerías importadas correctamente')
```

Librerías importadas correctamente

## ✓ Carga y preparación de datos

```
# Ruta del archivo
csv_path = '/content/pacientes.csv'

# Nombres de columnas según práctica anterior
cols = ['Pregnancies', 'PlasmaGlucose', 'DiastolicBP', 'TricepsSkinFold', 'TwoHourInsulin', 'BMI', 'DiabetesPedigree', 'Age', 'Outcome']

# Cargar datos
df = pd.read_csv(csv_path, header=None, names=cols)
print('Dimensiones del dataset:', df.shape)
print('\nPrimeras 5 filas:')
print(df.head())

print('\nInformación del dataset:')
print(df.info())

print('\nEstadísticas descriptivas:')
print(df.describe())
```

Dimensiones del dataset: (768, 9)

Primeras 5 filas:

	Pregnancies	PlasmaGlucose	DiastolicBP	TricepsSkinFold	TwoHourInsulin	Outcome
0	0	137	40	35	168	1
1	0	118	84	47	230	1
2	0	180	66	39	0	0
3	0	100	88	60	110	0
4	0	146	82	0	0	0

	BMI	DiabetesPedigree	Age	Outcome
0	43.1	2.288	33	1
1	45.8	0.551	31	1
2	42.0	1.893	25	1
3	46.8	0.962	31	0
4	40.5	1.781	44	0

Información del dataset:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 768 entries, 0 to 767

Data columns (total 9 columns):

#	Column	Non-Null Count	Dtype
0	Pregnancies	768 non-null	int64
1	PlasmaGlucose	768 non-null	int64
2	DiastolicBP	768 non-null	int64
3	TricepsSkinFold	768 non-null	int64
4	TwoHourInsulin	768 non-null	int64
5	BMI	768 non-null	float64
6	DiabetesPedigree	768 non-null	float64
7	Age	768 non-null	int64
8	Outcome	768 non-null	int64

dtypes: float64(2), int64(7)

memory usage: 54.1 KB

None

Estadísticas descriptivas:

	Pregnancies	PlasmaGlucose	DiastolicBP	TricepsSkinFold	TwoHourInsulin	BMI	DiabetesPedigree	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	20.536458	43.100000	2.288000	33.000000	0.500000
std	3.369578	31.972618	19.355807	15.952218	15.952218	10.000000	0.500000	3.500000	0.500000
min	0.000000	0.000000	0.000000	0.000000	0.000000	40.500000	0.000000	25.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	42.000000	1.893000	31.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	23.000000	43.100000	2.288000	33.000000	0.500000
75%	6.000000	140.250000	80.000000	32.000000	32.000000	45.800000	0.551000	31.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	99.000000	46.800000	1.893000	44.000000	1.000000

count	768.000000	768.000000	768.000000	768.000000	768.000000
mean	79.799479	31.992578	0.471876	33.240885	0.348958
std	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.078000	21.000000	0.000000
25%	0.000000	27.300000	0.243750	24.000000	0.000000
50%	30.500000	32.000000	0.372500	29.000000	0.000000
75%	127.250000	36.600000	0.626250	41.000000	1.000000
max	846.000000	67.100000	2.420000	81.000000	1.000000

```
# Separar características (X) y variable objetivo (y)
X = df.drop('Outcome', axis=1)
y = df['Outcome']
```

```
print('Forma de X (características):', X.shape)
print('Forma de y (variable objetivo):', y.shape)
print('\nDistribución de la variable objetivo:')
print(y.value_counts())
print('\nProporción de clases:')
print(y.value_counts(normalize=True))
```

```
Forma de X (características): (768, 8)
Forma de y (variable objetivo): (768,)
```

```
Distribución de la variable objetivo:
```

```
Outcome
0    500
1    268
Name: count, dtype: int64
```

```
Proporción de clases:
```

```
Outcome
0    0.651042
1    0.348958
Name: proportion, dtype: float64
```

## ✓ Actividad 1: Selección de características/atributos

### ✓ a) Chi-square test para seleccionar las 4 mejores características

```
# Para chi-square, necesitamos valores no negativos
# Primero escalamos los datos para asegurar valores no negativos
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

# Aplicar chi-square test para seleccionar las 4 mejores características
chi2_selector = SelectKBest(chi2, k=4)
X_chi2 = chi2_selector.fit_transform(X_scaled, y)

# Obtener puntajes y características seleccionadas
chi2_scores = chi2_selector.scores_
chi2_features = X.columns[chi2_selector.get_support()]

# Crear DataFrame con puntajes
chi2_df = pd.DataFrame({
    'Característica': X.columns,
    'Chi2_Score': chi2_scores,
    'Seleccionada': chi2_selector.get_support()
}).sort_values('Chi2_Score', ascending=False)

print('=== RESULTADOS CHI-SQUARE TEST ===')
print('\nPuntajes Chi-square para todas las características:')
print(chi2_df)

print('\n4 mejores características seleccionadas:')
for i, feature in enumerate(chi2_features, 1):
    score = chi2_scores[X.columns.get_loc(feature)]
    print(f'{i}. {feature}: {score:.4f}')

print(f'\nForma del dataset después de selección: {X_chi2.shape}')
```

```
=== RESULTADOS CHI-SQUARE TEST ===
```

```
Puntajes Chi-square para todas las características:
Característica  Chi2_Score  Seleccionada
```

7	Age	8.205691	True
1	PlasmaGlucose	7.094910	True
0	Pregnancies	6.559982	True
6	DiabetesPedigree	2.758584	True
4	TwoHourInsulin	2.571590	False
5	BMI	1.902673	False
3	TricepsSkinFold	0.536445	False
2	DiastolicBP	0.144306	False

4 mejores características seleccionadas:

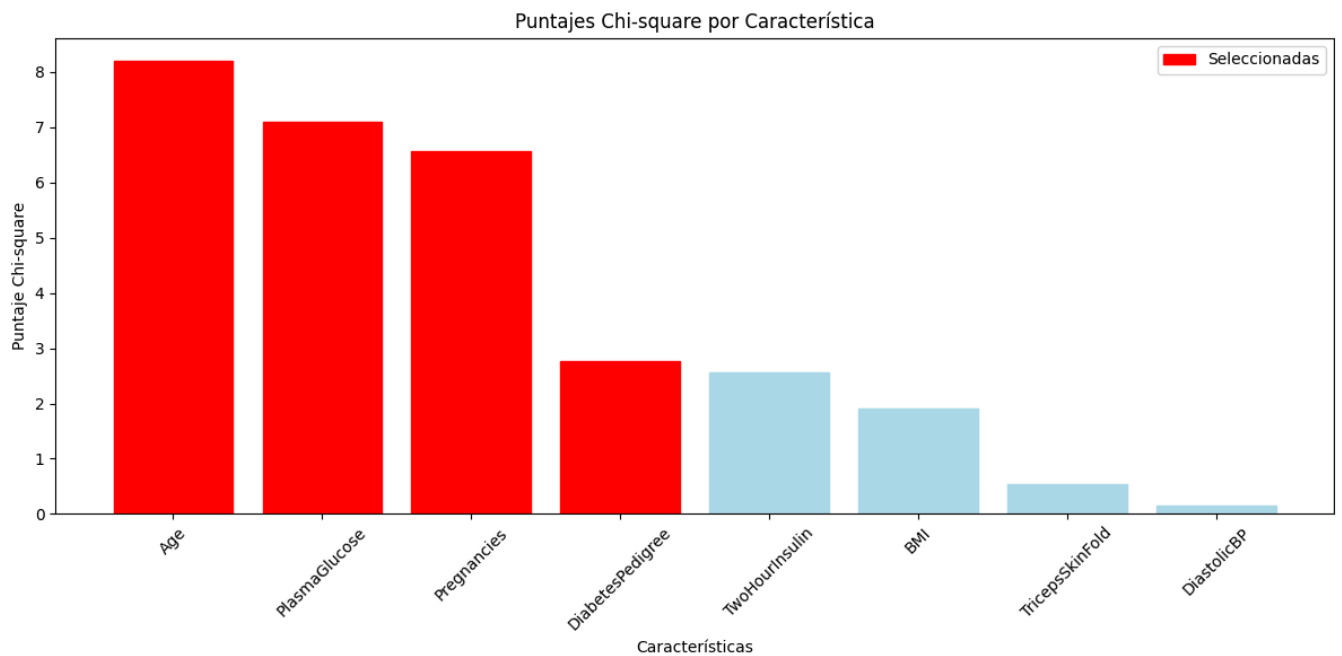
1. Pregnancies: 6.5600
2. PlasmaGlucose: 7.0949
3. DiabetesPedigree: 2.7586
4. Age: 8.2057

Forma del dataset después de selección: (768, 4)

```
# Visualizar puntajes Chi-square
plt.figure(figsize=(12, 6))
bars = plt.bar(range(len(chi2_df)), chi2_df['Chi2_Score'])
plt.xlabel('Características')
plt.ylabel('Puntaje Chi-square')
plt.title('Puntajes Chi-square por Característica')
plt.xticks(range(len(chi2_df)), chi2_df['Característica'], rotation=45)

# Colorear las 4 mejores características
for i, bar in enumerate(bars):
    if chi2_df.iloc[i]['Seleccionada']:
        bar.set_color('red')
    else:
        bar.set_color('lightblue')

plt.legend(['Seleccionadas', 'No seleccionadas'])
plt.tight_layout()
plt.show()
```



## ✓ b) Eliminación recursiva de características (RFE) con regresión logística

```
# Estandarizar los datos para regresión logística
scaler_std = StandardScaler()
X_std = scaler_std.fit_transform(X)

# Crear modelo de regresión logística
lr = LogisticRegression(random_state=42, max_iter=1000)
```

```
# Aplicar eliminación recursiva de características para seleccionar 3
rfe = RFE(estimator=lr, n_features_to_select=3, step=1)
X_rfe = rfe.fit_transform(X_std, y)

# Obtener características seleccionadas y rankings
rfe_features = X.columns[rfe.support_]
rfe_rankings = rfe.ranking_

# Crear DataFrame con rankings
rfe_df = pd.DataFrame({
    'Característica': X.columns,
    'Ranking': rfe_rankings,
    'Seleccionada': rfe.support_
}).sort_values('Ranking')

print('=== RESULTADOS ELIMINACIÓN RECURSIVA DE CARACTERÍSTICAS (RFE) ===')
print('\nRanking de todas las características (1 = mejor):')
print(rfe_df)

print('\n3 mejores características seleccionadas:')
for i, feature in enumerate(rfe_features, 1):
    print(f'{i}. {feature} (Ranking: 1)')

print(f'\nForma del dataset después de RFE: {X_rfe.shape}')
```

```
=== RESULTADOS ELIMINACIÓN RECURSIVA DE CARACTERÍSTICAS (RFE) ===
```

```
Ranking de todas las características (1 = mejor):
```

	Característica	Ranking	Seleccionada
0	Pregnancies	1	True
1	PlasmaGlucose	1	True
5	BMI	1	True
6	DiabetesPedigree	2	False
2	DiastolicBP	3	False
7	Age	4	False
4	TwoHourInsulin	5	False
3	TricepsSkinFold	6	False

```
3 mejores características seleccionadas:
```

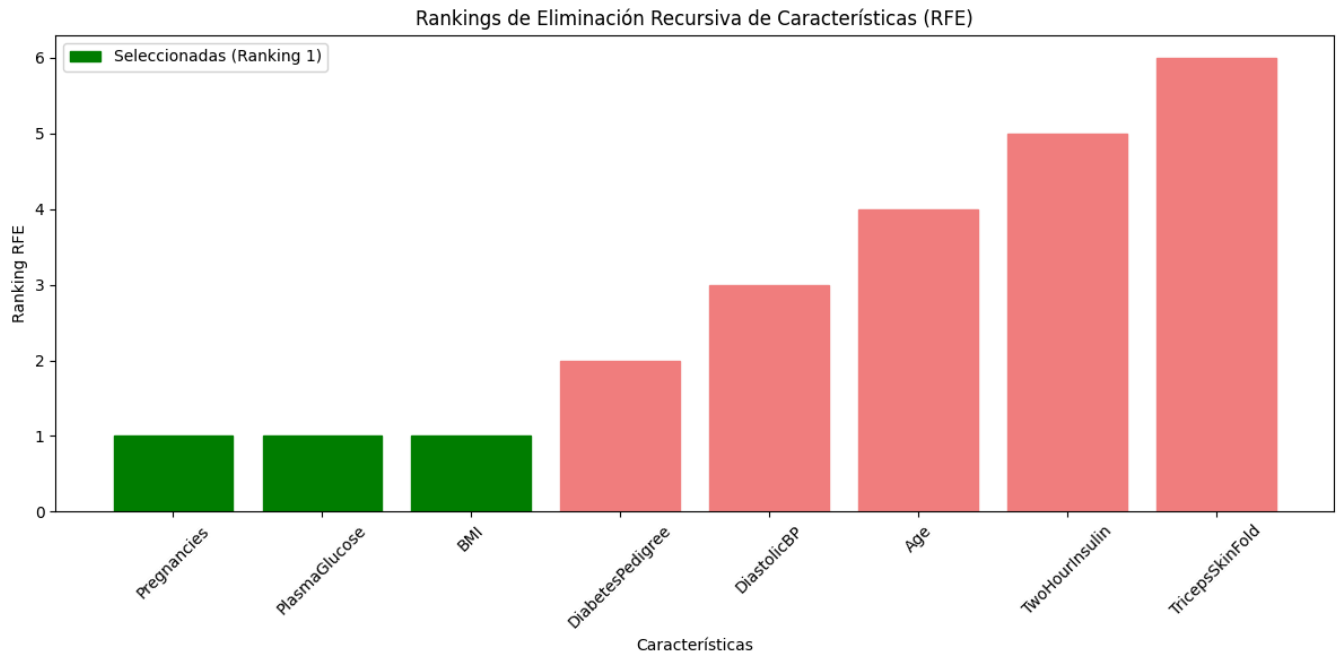
```
1. Pregnancies (Ranking: 1)
2. PlasmaGlucose (Ranking: 1)
3. BMI (Ranking: 1)
```

```
Forma del dataset después de RFE: (768, 3)
```

```
# Visualizar rankings RFE
plt.figure(figsize=(12, 6))
bars = plt.bar(range(len(rfe_df)), rfe_df['Ranking'])
plt.xlabel('Características')
plt.ylabel('Ranking RFE')
plt.title('Rankings de Eliminación Recursiva de Características (RFE)')
plt.xticks(range(len(rfe_df)), rfe_df['Característica'], rotation=45)

# Colorear las características seleccionadas
for i, bar in enumerate(bars):
    if rfe_df.iloc[i]['Seleccionada']:
        bar.set_color('green')
    else:
        bar.set_color('lightcoral')

plt.legend(['Seleccionadas (Ranking 1)', 'No seleccionadas'])
plt.tight_layout()
plt.show()
```



### ✓ c) Análisis de Componentes Principales (PCA)

```
# Aplicar PCA
pca = PCA()
X_pca = pca.fit_transform(X_std)

# Calcular varianza explicada
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance_ratio = np.cumsum(explained_variance_ratio)

print('=== RESULTADOS ANÁLISIS DE COMPONENTES PRINCIPALES (PCA) ===')
print(f'\nDimensiones originales: {X.shape}')
print(f'Dimensiones después de PCA: {X_pca.shape}')

print('\nVarianza explicada por cada componente principal:')
for i, var_ratio in enumerate(explained_variance_ratio, 1):
    print(f'PC{i}: {var_ratio:.4f} ({var_ratio*100:.2f}%)')

print('\nVarianza acumulada:')
for i, cum_var in enumerate(cumulative_variance_ratio, 1):
    print(f'Hasta PC{i}: {cum_var:.4f} ({cum_var*100:.2f}%)')

# Determinar número óptimo de componentes (90% de varianza)
n_components_90 = np.argmax(cumulative_variance_ratio >= 0.90) + 1
print(f'\nComponentes necesarios para explicar 90% de la varianza: {n_components_90}')
```

=== RESULTADOS ANÁLISIS DE COMPONENTES PRINCIPALES (PCA) ===

Dimensiones originales: (768, 8)  
Dimensiones después de PCA: (768, 8)

Varianza explicada por cada componente principal:

PC1: 0.2618 (26.18%)  
PC2: 0.2164 (21.64%)  
PC3: 0.1287 (12.87%)  
PC4: 0.1094 (10.94%)  
PC5: 0.0953 (9.53%)  
PC6: 0.0853 (8.53%)  
PC7: 0.0525 (5.25%)  
PC8: 0.0506 (5.06%)

Varianza acumulada:

Hasta PC1: 0.2618 (26.18%)  
Hasta PC2: 0.4782 (47.82%)

Hasta PC3: 0.6069 (60.69%)  
Hasta PC4: 0.7163 (71.63%)  
Hasta PC5: 0.8116 (81.16%)  
Hasta PC6: 0.8970 (89.70%)  
Hasta PC7: 0.9494 (94.94%)  
Hasta PC8: 1.0000 (100.00%)

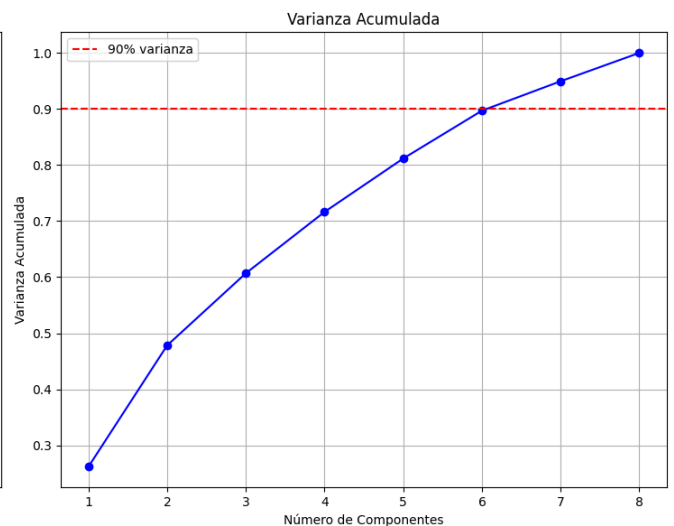
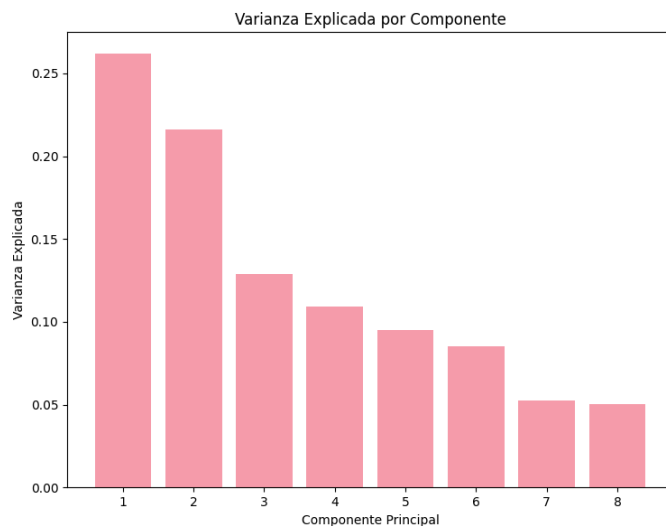
Componentes necesarios para explicar 90% de la varianza: 7

```
# Visualizar varianza explicada por PCA
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# Varianza individual
ax1.bar(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio, alpha=0.7)
ax1.set_xlabel('Componente Principal')
ax1.set_ylabel('Varianza Explicada')
ax1.set_title('Varianza Explicada por Componente')
ax1.set_xticks(range(1, len(explained_variance_ratio) + 1))

# Varianza acumulada
ax2.plot(range(1, len(cumulative_variance_ratio) + 1), cumulative_variance_ratio, 'bo-')
ax2.axhline(y=0.90, color='r', linestyle='--', label='90% varianza')
ax2.set_xlabel('Número de Componentes')
ax2.set_ylabel('Varianza Acumulada')
ax2.set_title('Varianza Acumulada')
ax2.set_xticks(range(1, len(cumulative_variance_ratio) + 1))
ax2.legend()
ax2.grid(True)

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



```
# Aplicar PCA con número óptimo de componentes
pca_optimal = PCA(n_components=n_components_90)
X_pca_optimal = pca_optimal.fit_transform(X_std)

print(f'Dataset transformado con PCA ({n_components_90} componentes):')
print(f'Forma original: {X.shape}')
print(f'Forma transformada: {X_pca_optimal.shape}')
print(f'Varianza explicada total: {pca_optimal.explained_variance_ratio_.sum():.4f} ({pca_optimal.explained_variance_ratio_.sum()*100:.1f}%)')
```

Dataset transformado con PCA (7 componentes):  
Forma original: (768, 8)  
Forma transformada: (768, 7)  
Varianza explicada total: 0.9494 (94.94%)



✓ d) Importancia de características con Random Forest y Extra Trees

```
# Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X, y)
rf_importances = rf.feature_importances_

# Extra Trees
et = ExtraTreesClassifier(n_estimators=100, random_state=42)
et.fit(X, y)
et_importances = et.feature_importances_

# Crear DataFrame con importancias
importance_df = pd.DataFrame({
    'Característica': X.columns,
    'RandomForest_Importance': rf_importances,
    'ExtraTrees_Importance': et_importances
})

# Ordenar por importancia promedio
importance_df['Promedio_Importance'] = (importance_df['RandomForest_Importance'] +
                                         importance_df['ExtraTrees_Importance']) / 2
importance_df = importance_df.sort_values('Promedio_Importance', ascending=False)

print('=== IMPORTANCIA DE CARACTERÍSTICAS ===')
print('\nImportancia de características (Random Forest y Extra Trees):')
print(importance_df)

print('\nTop 5 características más importantes (promedio):')
top5_features = importance_df.head(5)
for i, (_, row) in enumerate(top5_features.iterrows(), 1):
    print(f'{i}. {row["Característica"]}: {row["Promedio_Importance"]:.4f}')
    print(f'    RF: {row["RandomForest_Importance"]:.4f}, ET: {row["ExtraTrees_Importance"]:.4f}')
```

```
=== IMPORTANCIA DE CARACTERÍSTICAS ===

Importancia de características (Random Forest y Extra Trees):
   Característica  RandomForest_Importance  ExtraTrees_Importance  \
1  PlasmaGlucose           0.266215           0.240687
5                BMI           0.170205           0.135413
7                Age           0.130213           0.143071
6  DiabetesPedigree           0.126882           0.123250
0      Pregnancies           0.081378           0.109956
2      DiastolicBP           0.085846           0.097187
3  TricepsSkinFold           0.069240           0.079085
4   TwoHourInsulin           0.070021           0.071351

Promedio_Importance
1           0.253451
5           0.152809
7           0.136642
6           0.125066
0           0.095667
2           0.091516
3           0.074162
4           0.070686

Top 5 características más importantes (promedio):
1. PlasmaGlucose: 0.2535
   RF: 0.2662, ET: 0.2407
2. BMI: 0.1528
   RF: 0.1702, ET: 0.1354
3. Age: 0.1366
   RF: 0.1302, ET: 0.1431
4. DiabetesPedigree: 0.1251
   RF: 0.1269, ET: 0.1232
5. Pregnancies: 0.0957
   RF: 0.0814, ET: 0.1100
```

```
# Visualizar importancias
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20, 6))

# Random Forest
ax1.barh(range(len(importance_df)), importance_df['RandomForest_Importance'])
ax1.set_yticks(range(len(importance_df)))
ax1.set_yticklabels(importance_df['Característica'])
ax1.set_xlabel('Importancia')
ax1.set_title('Importancia - Random Forest')
```

```

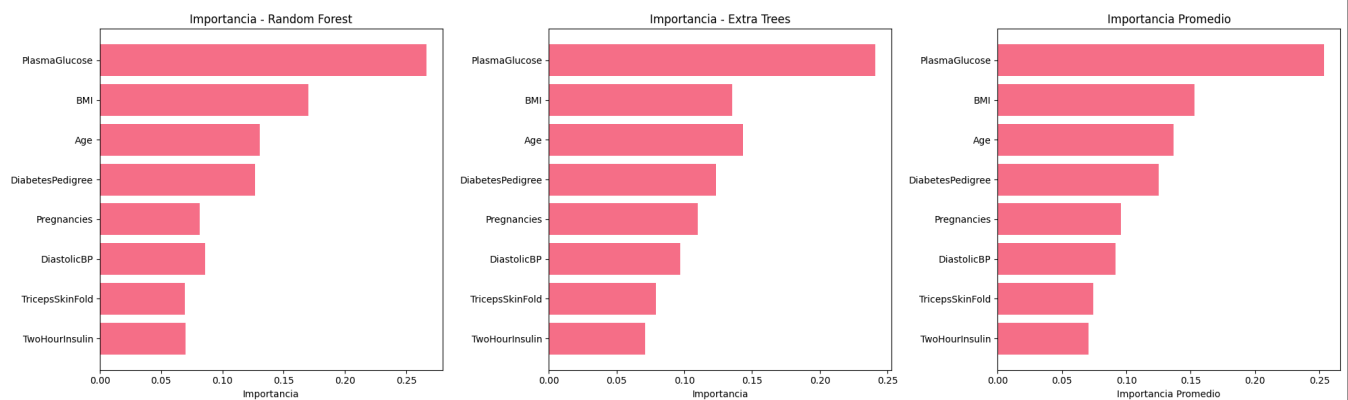
ax1.invert_yaxis()

# Extra Trees
ax2.barh(range(len(importance_df)), importance_df['ExtraTrees_Importance'])
ax2.set_yticks(range(len(importance_df)))
ax2.set_yticklabels(importance_df['Característica'])
ax2.set_xlabel('Importancia')
ax2.set_title('Importancia - Extra Trees')
ax2.invert_yaxis()

# Promedio
ax3.barh(range(len(importance_df)), importance_df['Promedio_Importance'])
ax3.set_yticks(range(len(importance_df)))
ax3.set_yticklabels(importance_df['Característica'])
ax3.set_xlabel('Importancia Promedio')
ax3.set_title('Importancia Promedio')
ax3.invert_yaxis()

plt.tight_layout()
plt.show()

```



## ✓ Actividad 2: Evaluación del rendimiento de algoritmos de aprendizaje de máquina

### ✓ a) División de datos en entrenamiento (67%) y prueba (33%)

```

# Dividir datos: 67% entrenamiento, 33% prueba
X_train, X_test, y_train, y_test = train_test_split(X_std, y, test_size=0.33, random_state=42, stratify=y)

print('=== DIVISIÓN DE DATOS ===')
print(f'Tamaño total del dataset: {len(X)}')
print(f'Tamaño de entrenamiento: {len(X_train)} ({len(X_train)/len(X)*100:.1f}%)')
print(f'Tamaño de prueba: {len(X_test)} ({len(X_test)/len(X)*100:.1f}%)')

print('\nDistribución de clases en entrenamiento:')
print(y_train.value_counts())
print('\nDistribución de clases en prueba:')
print(y_test.value_counts())

print('\nProporción de clases en entrenamiento:')
print(y_train.value_counts(normalize=True))
print('\nProporción de clases en prueba:')
print(y_test.value_counts(normalize=True))

```

```

=== DIVISIÓN DE DATOS ===
Tamaño total del dataset: 768
Tamaño de entrenamiento: 514 (66.9%)
Tamaño de prueba: 254 (33.1%)

```

```

Distribución de clases en entrenamiento:
Outcome
0      335

```

```

1    179
Name: count, dtype: int64

Distribución de clases en prueba:
Outcome
0    165
1     89
Name: count, dtype: int64

Proporción de clases en entrenamiento:
Outcome
0    0.651751
1    0.348249
Name: proportion, dtype: float64

Proporción de clases en prueba:
Outcome
0    0.649606
1    0.350394
Name: proportion, dtype: float64

```

## **✓ b) Modelo de regresión logística y evaluación de precisión**

```

# Crear y entrenar modelo de regresión logística
lr_model = LogisticRegression(random_state=42, max_iter=1000)
lr_model.fit(X_train, y_train)

# Realizar predicciones
y_pred_train = lr_model.predict(X_train)
y_pred_test = lr_model.predict(X_test)

# Calcular precisión (accuracy)
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

print('=== REGRESIÓN LOGÍSTICA - EVALUACIÓN ===')
print(f'Precisión en entrenamiento: {train_accuracy:.4f} ({train_accuracy*100:.2f}%)')
print(f'Precisión en prueba: {test_accuracy:.4f} ({test_accuracy*100:.2f}%)')

# Mostrar coeficientes del modelo
coefficients_df = pd.DataFrame({
    'Característica': X.columns,
    'Coeficiente': lr_model.coef_[0]
}).sort_values('Coeficiente', key=abs, ascending=False)

print('\nCoeficientes del modelo (ordenados por magnitud):')
print(coefficients_df)

```

```

=== REGRESIÓN LOGÍSTICA - EVALUACIÓN ===
Precisión en entrenamiento: 0.7840 (78.40%)
Precisión en prueba: 0.7717 (77.17%)

```

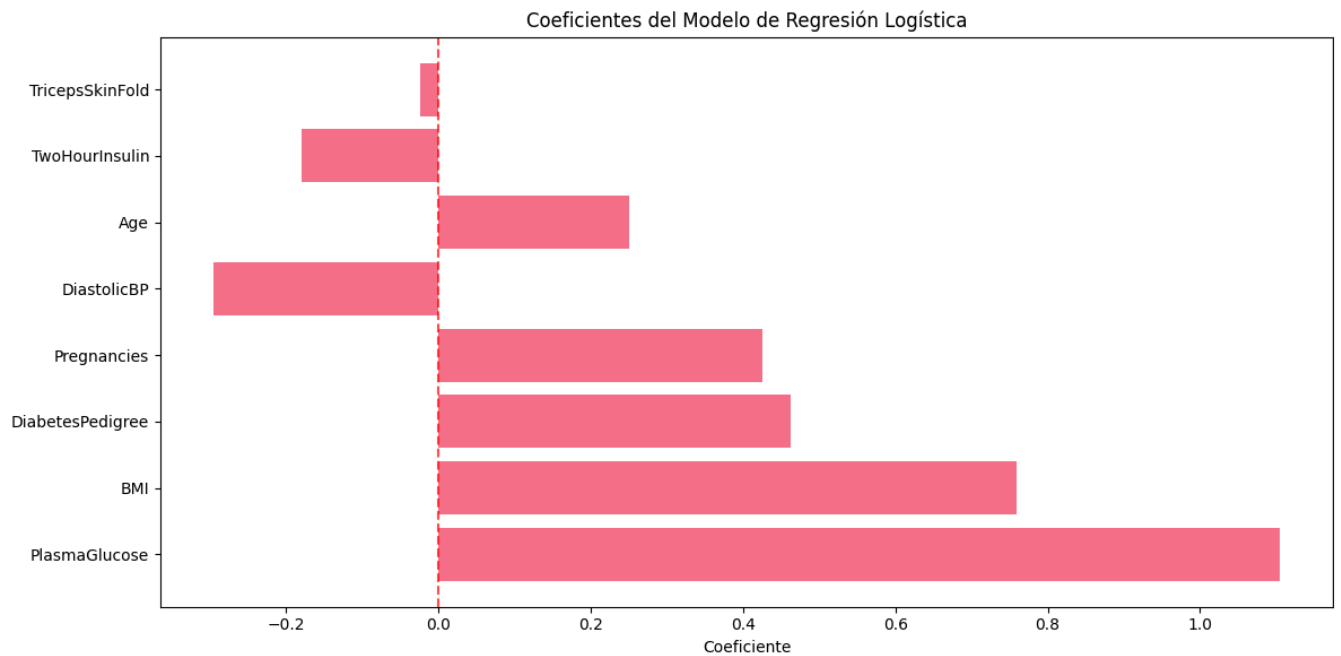
Coeficientes del modelo (ordenados por magnitud):

	Característica	Coeficiente
1	PlasmaGlucose	1.104295
5	BMI	0.759389
6	DiabetesPedigree	0.462419
0	Pregnancies	0.425311
2	DiastolicBP	-0.294862
7	Age	0.250731
4	TwoHourInsulin	-0.179185
3	TricepsSkinFold	-0.023123

```

# Visualizar coeficientes
plt.figure(figsize=(12, 6))
plt.barh(range(len(coefficients_df)), coefficients_df['Coeficiente'])
plt.yticks(range(len(coefficients_df)), coefficients_df['Característica'])
plt.xlabel('Coeficiente')
plt.title('Coeficientes del Modelo de Regresión Logística')
plt.axvline(x=0, color='red', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

```



### ✓ c) Validación cruzada con 10 segmentos (K-Fold)

```
# Validación cruzada con 10 folds
cv_scores = cross_val_score(lr_model, X_std, y, cv=10, scoring='accuracy')

print('=== VALIDACIÓN CRUZADA (10-FOLD) ===')
print(f'Precisión de cada fold:')
for i, score in enumerate(cv_scores, 1):
    print(f'Fold {i:2d}: {score:.4f} ({score*100:.2f}%)')

print(f'\nEstadísticas de validación cruzada:')
print(f'Media: {cv_scores.mean():.4f} ({cv_scores.mean()*100:.2f}%)')
print(f'Desviación estándar: {cv_scores.std():.4f} ({cv_scores.std()*100:.2f}%)')
print(f'Mínimo: {cv_scores.min():.4f} ({cv_scores.min()*100:.2f}%)')
print(f'Máximo: {cv_scores.max():.4f} ({cv_scores.max()*100:.2f}%)')
```

=== VALIDACIÓN CRUZADA (10-FOLD) ===

Precisión de cada fold:

Fold 1: 0.7532 (75.32%)  
 Fold 2: 0.7922 (79.22%)  
 Fold 3: 0.7403 (74.03%)  
 Fold 4: 0.8052 (80.52%)  
 Fold 5: 0.8182 (81.82%)  
 Fold 6: 0.7792 (77.92%)  
 Fold 7: 0.7532 (75.32%)  
 Fold 8: 0.8182 (81.82%)  
 Fold 9: 0.7500 (75.00%)  
 Fold 10: 0.5263 (52.63%)

Estadísticas de validación cruzada:

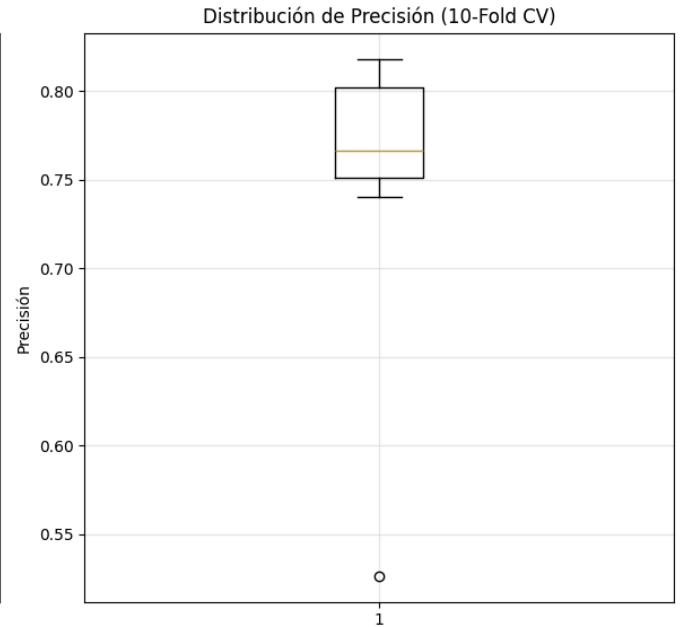
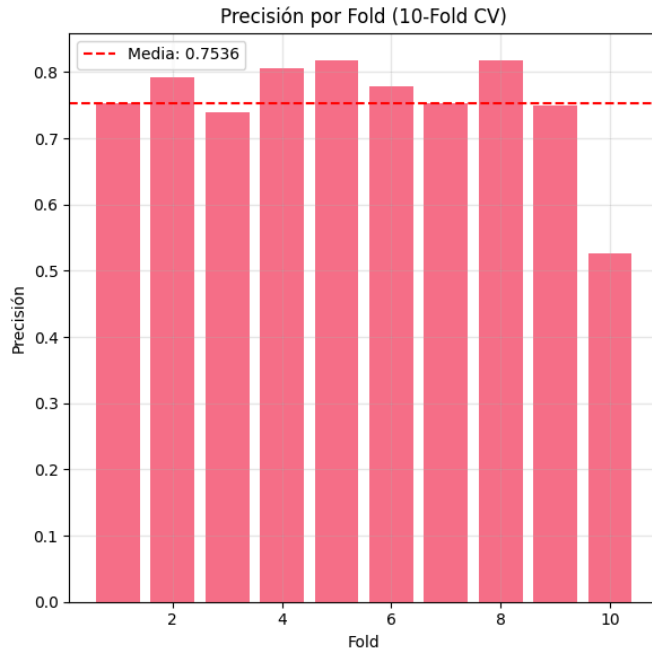
Media: 0.7536 (75.36%)  
 Desviación estándar: 0.0806 (8.06%)  
 Mínimo: 0.5263 (52.63%)  
 Máximo: 0.8182 (81.82%)

```
# Visualizar resultados de validación cruzada
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.bar(range(1, len(cv_scores) + 1), cv_scores)
plt.axhline(y=cv_scores.mean(), color='red', linestyle='--', label=f'Media: {cv_scores.mean():.4f}')
plt.xlabel('Fold')
plt.ylabel('Precisión')
plt.title('Precisión por Fold (10-Fold CV)')
```

```
plt.legend()
plt.grid(True, alpha=0.3)

plt.subplot(1, 2, 2)
plt.boxplot(cv_scores)
plt.ylabel('Precisión')
plt.title('Distribución de Precisión (10-Fold CV)')
plt.grid(True, alpha=0.3)

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



```
# Validación cruzada con 5 folds
cv_scores_k5 = cross_val_score(lr_model, X_std, y, cv=5, scoring='accuracy')

print('=== VALIDACIÓN CRUZADA (5-FOLD) ===')
print(f'Precisión de cada fold:')
for i, score in enumerate(cv_scores_k5, 1):
    print(f'Fold {i:2d}: {score:.4f} ({score*100:.2f}%)')

print(f'\nEstadísticas de validación cruzada (K=5):')
print(f'Media: {cv_scores_k5.mean():.4f} ({cv_scores_k5.mean()*100:.2f}%)')
print(f'Desviación estándar: {cv_scores_k5.std():.4f} ({cv_scores_k5.std()*100:.2f}%)')
print(f'Mínimo: {cv_scores_k5.min():.4f} ({cv_scores_k5.min()*100:.2f}%)')
print(f'Máximo: {cv_scores_k5.max():.4f} ({cv_scores_k5.max()*100:.2f}%)')
```

```
=== VALIDACIÓN CRUZADA (5-FOLD) ===
Precisión de cada fold:
Fold 1: 0.7792 (77.92%)
Fold 2: 0.7662 (76.62%)
Fold 3: 0.7857 (78.57%)
Fold 4: 0.7974 (79.74%)
Fold 5: 0.5490 (54.90%)
```

```
Estadísticas de validación cruzada (K=5):
Media: 0.7355 (73.55%)
Desviación estándar: 0.0938 (9.38%)
Mínimo: 0.5490 (54.90%)
Máximo: 0.7974 (79.74%)
```

#### ✓ d) Leave-One-Out Cross Validation (LOOCV)

```
# Leave-One-Out Cross Validation
loo = LeaveOneOut()
loo_scores = cross_val_score(lr_model, X_std, y, cv=loo, scoring='accuracy')

print('=== LEAVE-ONE-OUT CROSS VALIDATION (LOOCV) ===')
print(f'Número total de iteraciones: {len(loo_scores)}')
print(f'Número de predicciones correctas: {loo_scores.sum():.0f}')
print(f'Número de predicciones incorrectas: {len(loo_scores) - loo_scores.sum():.0f}')

print(f'\nEstadísticas de LOOCV:')
print(f'Media: {loo_scores.mean():.4f} ({loo_scores.mean()*100:.2f}%)')
print(f'Desviación estándar: {loo_scores.std():.4f} ({loo_scores.std()*100:.2f}%)')
```

```
=== LEAVE-ONE-OUT CROSS VALIDATION (LOOCV) ===
Número total de iteraciones: 768
Número de predicciones correctas: 596
Número de predicciones incorrectas: 172
```

```
Estadísticas de LOOCV:
Media: 0.7760 (77.60%)
Desviación estándar: 0.4169 (41.69%)
```

```
# Visualizar comparación de métodos de validación
plt.figure(figsize=(20, 6))

plt.subplot(1, 4, 1)
plt.bar(range(1, len(cv_scores_k5) + 1), cv_scores_k5)
plt.axhline(y=cv_scores_k5.mean(), color='red', linestyle='--', label=f'Media: {cv_scores_k5.mean():.4f}')
plt.xlabel('Fold')
plt.ylabel('Precisión')
plt.title('Precisión por Fold (5-Fold CV)')
plt.legend()
plt.grid(True, alpha=0.3)

plt.subplot(1, 4, 2)
plt.bar(range(1, len(cv_scores) + 1), cv_scores)
plt.axhline(y=cv_scores.mean(), color='red', linestyle='--', label=f'Media: {cv_scores.mean():.4f}')
plt.xlabel('Fold')
plt.ylabel('Precisión')
plt.title('Precisión por Fold (10-Fold CV)')
plt.legend()
plt.grid(True, alpha=0.3)

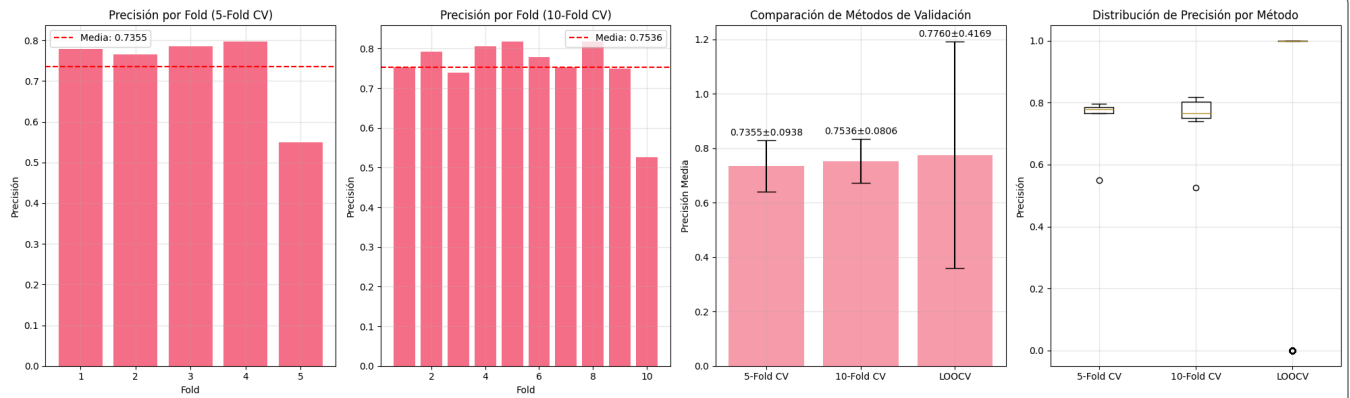
plt.subplot(1, 4, 3)
methods = ['5-Fold CV', '10-Fold CV', 'LOOCV']
means = [cv_scores_k5.mean(), cv_scores.mean(), loo_scores.mean()]
stds = [cv_scores_k5.std(), cv_scores.std(), loo_scores.std()]

plt.bar(methods, means, yerr=stds, capsize=10, alpha=0.7)
plt.ylabel('Precisión Media')
plt.title('Comparación de Métodos de Validación')
plt.grid(True, alpha=0.3)

# Añadir valores en las barras
for i, (mean, std) in enumerate(zip(means, stds)):
    plt.text(i, mean + std + 0.01, f'{mean:.4f}±{std:.4f}',
             ha='center', va='bottom')

plt.subplot(1, 4, 4)
plt.boxplot([cv_scores_k5, cv_scores, loo_scores], labels=methods)
plt.ylabel('Precisión')
plt.title('Distribución de Precisión por Método')
plt.grid(True, alpha=0.3)

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



## ✓ K=1 y interpretación de resultados

```
# Validación cruzada con K=1 (equivalente a LOOCV)
print('=== VALIDACIÓN CON K=1 ===')
print('Nota: K=1 es equivalente a Leave-One-Out Cross Validation')
print(f'Media: {loo_scores.mean():.4f}')
print(f'Desviación estándar: {loo_scores.std():.4f}')

# Validación cruzada con K=5
cv_scores_k5 = cross_val_score(lr_model, X_std, y, cv=5, scoring='accuracy')

print('\n=== VALIDACIÓN CON K=5 ===')
print(f'Precisión de cada fold:')
for i, score in enumerate(cv_scores_k5, 1):
    print(f'Fold {i:2d}: {score:.4f} ({score*100:.2f}%)')

print(f'\nEstadísticas de validación cruzada (K=5):')
print(f'Media: {cv_scores_k5.mean():.4f} ({cv_scores_k5.mean()*100:.2f}%)')
print(f'Desviación estándar: {cv_scores_k5.std():.4f} ({cv_scores_k5.std()*100:.2f}%)')
print(f'Mínimo: {cv_scores_k5.min():.4f} ({cv_scores_k5.min()*100:.2f}%)')
print(f'Máximo: {cv_scores_k5.max():.4f} ({cv_scores_k5.max()*100:.2f}%)')

print('\n=== INTERPRETACIÓN DE RESULTADOS ===')
print('\n1. MEDIA:')
print(f' - 5-Fold CV: {cv_scores_k5.mean():.4f}')
print(f' - 10-Fold CV: {cv_scores_k5.mean():.4f}')
print(f' - LOOCV (K=1): {loo_scores.mean():.4f}')
print('\n Interpretación:')
print(' La media representa la precisión promedio esperada del modelo.')
print(' Los tres métodos dan resultados similares, lo que indica')
print(' que el modelo es estable y consistente.')

print('\n2. DESVIACIÓN ESTÁNDAR:')
print(f' - 5-Fold CV: {cv_scores_k5.std():.4f}')
print(f' - 10-Fold CV: {cv_scores_k5.std():.4f}')
print(f' - LOOCV (K=1): {loo_scores.std():.4f}')
print('\n Interpretación:')
print(' La desviación estándar mide la variabilidad del rendimiento.')
print(' - LOOCV tiene mayor variabilidad porque cada iteración usa')
print(' solo una muestra para prueba.')
print(' - 5-Fold y 10-Fold CV son más estables al usar múltiples muestras por fold.')
print(' - Una desviación menor indica mayor confiabilidad en la estimación.')

print('\n3. RECOMENDACIÓN:')
if cv_scores_k5.std() < cv_scores.std() and cv_scores_k5.std() < loo_scores.std():
    print(' 5-Fold CV es preferible por su menor variabilidad y menor')
    print(' costo computacional, manteniendo una estimación confiable.')
elif cv_scores.std() < loo_scores.std():
    print(' 10-Fold CV es preferible por su menor variabilidad y menor')
    print(' costo computacional, manteniendo una estimación confiable.')
```

```

else:
    print('    Los métodos son comparables en términos de variabilidad.')

```

```

=== VALIDACIÓN CON K=1 ===
Nota: K=1 es equivalente a Leave-One-Out Cross Validation
Media: 0.7760
Desviación estándar: 0.4169

```

```

=== VALIDACIÓN CON K=5 ===
Precisión de cada fold:
Fold 1: 0.7792 (77.92%)
Fold 2: 0.7662 (76.62%)
Fold 3: 0.7857 (78.57%)
Fold 4: 0.7974 (79.74%)
Fold 5: 0.5490 (54.90%)

```

```

Estadísticas de validación cruzada (K=5):
Media: 0.7355 (73.55%)
Desviación estándar: 0.0938 (9.38%)
Mínimo: 0.5490 (54.90%)
Máximo: 0.7974 (79.74%)

```

```

=== INTERPRETACIÓN DE RESULTADOS ===

```

```

1. MEDIA:
- 5-Fold CV: 0.7355
- 10-Fold CV: 0.7536
- LOOCV (K=1): 0.7760

```

```

Interpretación:
La media representa la precisión promedio esperada del modelo.
Los tres métodos dan resultados similares, lo que indica
que el modelo es estable y consistente.

```

```

2. DESVIACIÓN ESTÁNDAR:
- 5-Fold CV: 0.0938
- 10-Fold CV: 0.0806
- LOOCV (K=1): 0.4169

```

```

Interpretación:
La desviación estándar mide la variabilidad del rendimiento.
- LOOCV tiene mayor variabilidad porque cada iteración usa
  solo una muestra para prueba.
- 5-Fold y 10-Fold CV son más estables al usar múltiples muestras por fold.
- Una desviación menor indica mayor confiabilidad en la estimación.

```

```

3. RECOMENDACIÓN:
10-Fold CV es preferible por su menor variabilidad y menor
costo computacional, manteniendo una estimación confiable.

```

## Resumen de resultados y conclusiones

```

print('=' * 60)
print('RESUMEN DE RESULTADOS - PRÁCTICA 6')
print('=' * 60)

print('\n1. SELECCIÓN DE CARACTERÍSTICAS:')
print(f'  Chi-square (4 mejores): {list(chi2_features)}')
print(f'  RFE (3 mejores): {list(rfe_features)}')
print(f'  PCA: {n_components_90} componentes para 90% varianza')
print(f'  Top 3 por importancia: {list(importance_df.head(3)["Característica"])}')

print('\n2. RENDIMIENTO DEL MODELO:')
print(f'  Precisión en prueba: {test_accuracy:.4f} ({test_accuracy*100:.2f}%)')
print(f'  5-Fold CV: {cv_scores_k5.mean():.4f} ± {cv_scores_k5.std():.4f}')
print(f'  10-Fold CV: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}')
print(f'  LOOCV: {loo_scores.mean():.4f} ± {loo_scores.std():.4f}')

print('\n3. CARACTERÍSTICAS MÁS IMPORTANTES:')
top3_importance = importance_df.head(3)
for i, (_, row) in enumerate(top3_importance.iterrows(), 1):
    print(f'  {i}. {row["Característica"]}: {row["Promedio_Importance"]:.4f}')

print('\n4. ANÁLISIS DE VALIDACIÓN CRUZADA:')
print('  - 5-Fold CV: Balance entre estabilidad y eficiencia computacional')
print('  - 10-Fold CV: Mayor estabilidad estadística')
print('  - LOOCV: Máximo uso de datos pero mayor variabilidad')

print('\n5. CONCLUSIONES:')

```



```
print(' - El modelo muestra un rendimiento consistente entre diferentes')
print(' métodos de validación.')
print(' - Las técnicas de selección identifican características clave')
print(' para la predicción de diabetes.')
print(' - PCA permite reducir dimensionalidad manteniendo información relevante.')
print(' - La validación cruzada confirma la generalización del modelo.')
print(' - 5-Fold CV ofrece un buen balance para evaluación del modelo.')
```

=====

## RESUMEN DE RESULTADOS – PRÁCTICA 6

=====

### 1. SELECCIÓN DE CARACTERÍSTICAS:

Chi-square (4 mejores): ['Pregnancies', 'PlasmaGlucose', 'DiabetesPedigree', 'Age']  
RFE (3 mejores): ['Pregnancies', 'PlasmaGlucose', 'BMI']  
PCA: 7 componentes para 90% varianza  
Top 3 por importancia: ['PlasmaGlucose', 'BMI', 'Age']

### 2. RENDIMIENTO DEL MODELO:

Precisión en prueba: 0.7717 (77.17%)  
5-Fold CV: 0.7355 ± 0.0938  
10-Fold CV: 0.7536 ± 0.0806  
LOOCV: 0.7760 ± 0.4169

### 3. CARACTERÍSTICAS MÁS IMPORTANTES:

1. PlasmaGlucose: 0.2535  
2. BMI: 0.1528  
3. Age: 0.1366