Scikit Learn

Preprocesamiento de datos

CertiDevs

Índice de contenidos

1. Preprocesamiento de datos

El **preprocesamiento** en scikit-learn consiste en aplicar técnicas y transformaciones a los datos antes de usarlos para entrenar modelos de aprendizaje automático.

El objetivo del preprocesamiento es mejorar la **calidad** y la **estructura** de los datos, lo que puede resultar en un **mejor rendimiento** del modelo.

Algunas de las tareas comunes de preprocesamiento incluyen la *limpieza de datos*, la *estandarización*, la *normalización*, la *codificación* de variables categóricas, el manejo de *valores faltantes* y la *selección de características*.

Scikit-learn proporciona una serie de módulos y paquetes para ayudar en el preprocesamiento de datos. Algunos de los más importantes incluyen:

- `sklearn.preprocessing`: Este módulo proporciona funciones y clases para transformar datos, como estandarización, normalización, codificación de variables categóricas y escalado de características. Algunas de las clases y funciones más comunes en este módulo incluyen `StandardScaler`, `MinMaxScaler`, `MaxAbsScaler`, `RobustScaler`, `OneHotEncoder`, `LabelEncoder` y `Normalizer`.
- `sklearn.impute`: Este módulo proporciona clases y funciones para manejar valores faltantes en los datos. Incluye métodos como `SimpleImputer`, que permite reemplazar valores faltantes utilizando estadísticas simples como la media, la mediana o el valor más frecuente, y `KNNImputer`, que utiliza la técnica de vecinos más cercanos para estimar los valores faltantes.
- `sklearn.feature_selection`: Este módulo ofrece técnicas para seleccionar las características más relevantes y reducir la dimensionalidad de los datos. Algunas de las clases y funciones incluidas en este módulo son `SelectKBest`, `SelectPercentile`, `RFE` (Recursive Feature Elimination) y `RFECV` (Recursive Feature Elimination with Cross-Validation).
- `sklearn.feature_extraction`: Este módulo proporciona herramientas para extraer características de datos no estructurados, como texto e imágenes. Incluye clases como `CountVectorizer`, `TfidfVectorizer` y `DictVectorizer` para la extracción de características de texto y `FeatureHasher` para la extracción de características de datos categóricos.
- `sklearn.decomposition`: Este módulo incluye técnicas de reducción de dimensionalidad, como `PCA` (Principal Component Analysis), `NMF` (Non-negative Matrix Factorization) y `LDA` (Latent Dirichlet Allocation), que pueden ser útiles para el preprocesamiento y la selección de características.

Es importante tener en cuenta que la elección de las técnicas adecuadas de preprocesamiento de datos dependerá del problema específico y del conjunto de datos en cuestión. Además, siempre es útil experimentar con diferentes técnicas y evaluar su impacto en el rendimiento del modelo. El preprocesamiento de datos es un paso crucial en el proceso de análisis de datos, y comprender y aplicar correctamente estas técnicas puede marcar la diferencia en la calidad y el rendimiento de los modelos de aprendizaje automático.

A continuación se muestran algunas de las técnicas de preprocesamiento de datos.

1.1. Escalado de características

- **Escalado Min-Max**: La normalización Min-Max se utiliza para que todas las características tengan el mismo rango, generalmente entre `0` y `1`.
  - Esto es útil cuando los algoritmos de aprendizaje automático son sensibles a la escala de las características, ya que la diferencia en escala puede afectar negativamente el rendimiento del modelo.
  - Al aplicar esta técnica, nos aseguramos de que todas las características contribuyan equitativamente al proceso de aprendizaje y evitamos que las características con valores más grandes dominen a las características con valores más pequeños.
- **Estandarización**: La estandarización es otra técnica de escalado que transforma las características para que tengan **media** `0` y **desviación estándar** `1`.
  - Al igual que la normalización Min-Max, la estandarización ayuda a los algoritmos de aprendizaje automático a ser menos sensibles a la escala de las características.
  - La estandarización puede ser más apropiada que la normalización Min-Max cuando los datos tienen una distribución normal o cuando la distribución de los datos es desconocida.

1.2. Codificación de variables categóricas

- **Codificación One-Hot**: La codificación One-Hot se utiliza para convertir variables categóricas en características numéricas, lo que facilita su uso en algoritmos de aprendizaje automático. Al aplicar esta técnica, creamos una nueva columna binaria para cada categoría de la variable categórica, de modo que el modelo pueda aprender la importancia de cada categoría por separado. La codificación One-Hot puede aumentar significativamente el número de características en el conjunto de datos, pero también permite a los algoritmos capturar relaciones no lineales entre las variables categóricas y la variable objetivo.
- **Codificación Ordinal**: La codificación ordinal asigna un número entero a cada categoría de una variable categórica según un orden específico. Esta técnica puede ser útil cuando las categorías tienen un orden inherente, como "bajo", "medio" y "alto". Al aplicar la codificación ordinal, preservamos el orden de las categorías en la representación numérica, lo que puede ser útil para ciertos algoritmos de aprendizaje automático que pueden aprovechar esta información adicional.

1.3. Creación de nuevas características

- **Características polinomiales**: La creación de características polinomiales es una técnica que combina características existentes para crear características de mayor grado. Esto puede ayudar a los algoritmos de aprendizaje automático a capturar relaciones no lineales entre las características y la variable objetivo. Al aplicar esta técnica, podemos mejorar el rendimiento del modelo al proporcionar información adicional que no fue capturada por las características originales.
- **Características de ingeniería de dominio**: La ingeniería de características de dominio implica la creación de nuevas características basadas en el conocimiento específico del dominio del problema. Esta técnica puede ser útil para mejorar el rendimiento del modelo al incorporar

información relevante que no fue capturada por las características originales. Al aplicar la ingeniería de características de dominio, podemos personalizar nuestro modelo para adaptarse mejor a las particularidades del problema y, en última instancia, mejorar su capacidad de generalización.

1.4. Imputación de valores faltantes

- **Imputación media/mediana/moda**: La imputación de valores faltantes mediante la media, la mediana o la moda es una técnica común para abordar la ausencia de datos en un conjunto de datos. Al reemplazar los valores faltantes con la media, la mediana o la moda de la característica, estamos conservando la distribución general de los datos y evitamos la eliminación de registros con información faltante. Esta técnica es útil cuando la cantidad de datos faltantes no es demasiado alta y no existe una correlación significativa entre los valores faltantes y otras características.
- **Imputación k-NN**: La imputación k-NN (k-Nearest Neighbors) es una técnica que reemplaza los valores faltantes en función de la similitud con otros registros en el conjunto de datos. Al utilizar esta técnica, estamos considerando la relación entre las características para estimar los valores faltantes de manera más precisa. La imputación k-NN puede ser útil cuando existe una correlación entre los valores faltantes y otras características, y puede proporcionar estimaciones más precisas que la imputación media/mediana/moda en ciertos casos.

1.5. Eliminación de outliers

- **IQR (Rango Intercuartil)**: La eliminación de outliers basada en el IQR identifica los outliers en función de su posición en la distribución de los datos. Los registros que se encuentran fuera del rango intercuartil (por debajo del cuartil inferior o por encima del cuartil superior) se consideran outliers y se eliminan del conjunto de datos. Esta técnica es útil para eliminar registros extremos de manera más robusta que el Z-Score, ya que es menos sensible a las fluctuaciones en la media y la desviación estándar.
- **Z-Score**: La eliminación de outliers basada en el Z-Score es una técnica que identifica los outliers en función de su desviación estándar respecto a la media. Los registros que tienen un Z-Score mayor o menor que un umbral determinado se consideran outliers y se eliminan del conjunto de datos. Esta técnica es útil para eliminar registros extremos que pueden afectar negativamente el rendimiento del modelo, especialmente en algoritmos sensibles a outliers.

1.6. Agrupación (Binning)

- **Agrupación**: La agrupación, también conocida como binning, es una técnica que transforma variables continuas en variables discretas al dividir el rango de la variable en intervalos o "bins". La agrupación puede ser útil para simplificar el análisis de los datos al reducir la cantidad de ruido en la distribución y resaltar patrones subyacentes. También puede ser útil para algoritmos de aprendizaje automático que funcionan mejor con características discretas o cuando se desea reducir la dimensionalidad del conjunto de datos.

1.7. Tratamiento de características temporales

- **Extracción de características temporales**: La extracción de características temporales implica la creación de nuevas características a partir de datos de fecha y hora. Algunos ejemplos incluyen la extracción de la hora del día, el día de la semana, el mes o el año a partir de una marca de tiempo. Esta técnica puede ser útil para capturar patrones estacionales o tendencias temporales en los datos que pueden ser relevantes para el problema en cuestión.

1.8. Reducción de dimensionalidad

- **Análisis de componentes principales (PCA)**: El PCA es una técnica de reducción de dimensionalidad que transforma las características originales en un nuevo espacio de características ortogonales, donde las nuevas características, llamadas componentes principales, son combinaciones lineales de las características originales. El PCA puede ser útil para reducir la dimensionalidad del conjunto de datos y eliminar la multicolinealidad entre las características, lo que puede mejorar el rendimiento y la interpretabilidad del modelo de aprendizaje automático.
- **Selección de características**: La selección de características es un enfoque para eliminar características irrelevantes o redundantes del conjunto de datos. Esto puede ser útil para mejorar la eficiencia computacional del modelo, reducir el ruido en los datos y mejorar la interpretabilidad del modelo. Existen varias técnicas de selección de características, como la selección univariante, la eliminación recursiva de características y la selección basada en modelos, cada una con sus propios criterios y métodos para determinar qué características son las más relevantes para el problema en cuestión.

2. Ejemplo 1 (datos ficticios)

Dado que los conjuntos de datos de Scikit-learn no tienen valores faltantes ni atípicos, vamos a utilizar un conjunto de **datos ficticio** para ilustrar cómo **limpiar datos** en Python.

A continuación, se muestra cómo generar un conjunto de datos con **valores faltantes** y **atípicos** utilizando NumPy y Pandas.

```
import numpy as np
import pandas as pd

# Crear un conjunto de datos ficticio con valores faltantes y atípicos
np.random.seed(42)
data = np.random.normal(50, 10, size=(100, 4))
data = np.where(data < 0, np.nan, data)
data[np.random.randint(0, 100, 10), np.random.randint(0, 4, 10)] = np.nan
data[0, 0] = 200

data_df = pd.DataFrame(data, columns=["A", "B", "C", "D"])
```

2.1. Identificación y manejo de datos faltantes

Primero, **identificamos datos faltantes** y luego decidimos cómo manejarlos.

Existen diferentes enfoques, como **eliminar filas** con valores faltantes, **reemplazarlos** con la **media**, la **mediana** o la **moda**, o utilizar técnicas avanzadas como la **imputación KNN**.

```
# Identificar datos faltantes
print(data_df.isnull().sum())

# Eliminar filas con datos faltantes
data_df_dropna = data_df.dropna()

# Rellenar datos faltantes con la media
data_df_fillna_mean = data_df.fillna(data_df.mean())

# Rellenar datos faltantes con la mediana
data_df_fillna_median = data_df.fillna(data_df.median())

# Imputación KNN para datos faltantes
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=3)
data_df_knn_imputed = pd.DataFrame(imputer.fit_transform(data_df), columns=data_df
.columns)
```

2.2. Identificación de valores atípicos

Valores atípicos son valores extremos que difieren significativamente del resto de los datos.

Existen diferentes enfoques para identificar y tratar valores atípicos, como el método IQR, el método Z-score y el método de percentiles.

2.2.1. Método IQR

El coeficiente 1.5 en el método IQR (Rango intercuartil) es un factor de escala que se utiliza para determinar los límites inferiores y superiores que identifican valores atípicos en un conjunto de datos. Este coeficiente se utiliza comúnmente en estadísticas para establecer un criterio razonable para definir valores atípicos.

```
# Método IQR para identificar valores atípicos
Q1 = data_df.quantile(0.25)
Q3 = data_df.quantile(0.75)
IQR = Q3 - Q1

outliers = (data_df < (Q1 - 1.5 * IQR)) | (data_df > (Q3 + 1.5 * IQR))
print("Valores atípicos (método IQR):\n", outliers.any(axis=1))
```

El valor 1.5 se utiliza para ampliar el rango intercuartil y, por lo tanto, proporcionar límites menos estrictos para identificar valores atípicos. En términos generales, un valor más alto daría como resultado menos valores atípicos detectados, mientras que un valor más bajo resultaría en más valores atípicos detectados.

2.2.2. Método Z-score

El **método Z-score**, también conocido como desviación estándar, es una técnica estadística que se utiliza para identificar valores atípicos en un conjunto de datos.

El Z-score es una medida que describe la posición de un valor en relación con la media y la desviación estándar de un conjunto de datos.

Un Z-score alto indica que el valor está más lejos de la media, mientras que un Z-score bajo indica que el valor está más cerca de la media.

```
# Método Z-score para identificar valores atípicos
from scipy import stats

z_scores = np.abs(stats.zscore(data_df))
outliers = z_scores > 3
print("Valores atípicos (método Z-score):\n", outliers.any(axis=1))
```

Se utiliza la función `stats.zscore()` de SciPy para calcular los Z-scores de un DataFrame llamado data_df. Luego, se calcula el valor absoluto de los Z-scores y se comparan con un umbral (en este caso, `3`) para identificar los valores atípicos. Un umbral de `3` implica que cualquier valor con un Z-score de más de 3 desviaciones estándar por encima o por debajo de la media se considera un valor atípico.

2.2.3. Método de percentiles

El método de **percentiles** es otra técnica estadística para identificar valores atípicos en un conjunto de datos. Consiste en calcular los percentiles de los datos y establecer límites para los valores atípicos.

Un **valor atípico** se considera cualquier valor que esté por debajo o por encima de los límites definidos por los percentiles.

```
# Método de percentiles para identificar valores atípicos
lower_bound = data_df.quantile(0.01)
upper_bound = data_df.quantile(0.99)

outliers = (data_df < lower_bound) | (data_df > upper_bound)
print("Valores atípicos (método de percentiles):\los):\n", outliers.any(axis=1))
```

Se calculan los percentiles 1% y 99% del DataFrame data_df utilizando la función `quantile()`. Luego, se identifican los valores atípicos como aquellos que están por debajo del límite inferior (percentil 1%) o por encima del límite superior (percentil 99%). Al elegir percentiles más cercanos al centro del rango (como 5% y 95%), se considerarán más valores como atípicos, mientras que al elegir percentiles más alejados del centro (como 0.1% y 99.9%), se considerarán menos valores como atípicos.

2.3. Manejo de valores atípicos

Una vez identificados los **valores atípicos**, podemos tratarlos de varias maneras, como eliminarlos, reemplazarlos con la media, la mediana o la moda, o utilizar técnicas más avanzadas como la transformación Box-Cox o el suavizado exponencial.

```
# Eliminar valores atípicos (usando el método IQR)
data_df_no_outliers_iqr = data_df[~outliers.any(axis=1)]

# Reemplazar valores atípicos con la mediana (usando el método IQR)
data_df_replace_outliers_iqr = data_df.copy()
data_df_replace_outliers_iqr[outliers] = data_df_replace_outliers_iqr.median()

# Transformación Box-Cox para reducir el efecto de valores atípicos
from scipy.stats import boxcox

# Asegurarse de que todos los valores sean estrictamente positivos
data_df_positive = data_df - data_df.min().min() + 1

data_df_boxcox = data_df_positive.apply(lambda x: pd.Series(boxcox(x.dropna())[0]),
axis=0)

# Suavizado exponencial para reducir el efecto de valores atípicos
data_df_exp_smoothed = data_df.ewm(alpha=0.1).mean()
```

Es importante notar que la elección del método para tratar valores faltantes y atípicos dependerá del problema específico y del conocimiento del dominio.

En algunos casos, eliminar valores atípicos o reemplazarlos con la media, mediana o moda puede ser apropiado, mientras que en otros casos, las técnicas más avanzadas como la transformación Box-Cox o el suavizado exponencial pueden ser más adecuadas.

3. Ejemplo 2 (titanic)

```
import seaborn as sns
import pandas as pd
import numpy as np
```

```
# Cargar el conjunto de datos 'titanic'
data_df = sns.load_dataset('titanic')
print(data_df.head())
```

3.1. Identificación y manejo de datos faltantes

```
# Identificar datos faltantes
print(data_df.isnull().sum())

# Eliminar filas con datos faltantes
data_df_dropna = data_df.dropna()

# Rellenar datos faltantes con la media (para columnas numéricas)
data_df_fillna_mean = data_df.fillna(data_df.mean())

# Rellenar datos faltantes con la mediana (para columnas numéricas)
data_df_fillna_median = data_df.fillna(data_df.median())

# Rellenar datos faltantes con la moda (para columnas categóricas)
data_df_fillna_mode = data_df.copy()
for column in data_df_fillna_mode.columns:
    if data_df_fillna_mode[column].dtype == 'object':
        data_df_fillna_mode[column] = data_df_fillna_mode[column].fillna
(data_df_fillna_mode[column].mode()[0])
```

3.2. Identificación y manejo de valores atípicos

Utilizaremos el método IQR para identificar y manejar valores atípicos en las columnas numéricas del conjunto de datos.

```
# Seleccionar columnas numéricas
numeric_columns = data_df.select_dtypes(include=np.number).columns

# Método IQR para identificar valores atípicos
Q1 = data_df[numeric_columns].quantile(0.25)
Q3 = data_df[numeric_columns].quantile(0.75)
IQR = Q3 - Q1

outliers = (data_df[numeric_columns] < (Q1 - 1.5 * IQR)) | (data_df[numeric_columns] >
(Q3 + 1.5 * IQR))
print("Valores atípicos (método IQR):\n", outliers.any(axis=1))

# Eliminar valores atípicos (usando el método IQR)
data_df_no_outliers_iqr = data_df[~outliers.any(axis=1)]

# Reemplazar valores atípicos con la mediana (usando el método IQR)
data_df_replace_outliers_iqr = data_df.copy()
```

```
data_df_replace_outliers_iqr[outliers] = data_df_replace_outliers_iqr.median()
```

Este es un ejemplo de cómo limpiar datos utilizando el conjunto de datos 'titanic' de Seaborn.

La elección del método para tratar valores faltantes y atípicos dependerá del problema específico y del conocimiento del dominio.

3.3. Escalado de características

El **escalado de características** es el proceso de cambiar la escala de los valores de las características para que estén en el mismo rango.

Esto es útil en algoritmos que son sensibles a la escala de las características, como el algoritmo de k-means o el algoritmo de descenso de gradiente.

Existen varios **métodos de escalado de características**, como la normalización min-max y la estandarización.

3.3.1. MinMaxScaler

Normalización Min-Max: Escala las características al rango [0, 1] utilizando la fórmula (x - min) / (max - min).

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
data_df_normalized = pd.DataFrame(scaler.fit_transform(data_df), columns=data_df
.columns)

# Ejemplo 1
age_normalized = data_df_normalized['age']
print(age_normalized.head())

# Ejemplo 2
fare_normalized = data_df_normalized['fare']
print(fare_normalized.head())
```

3.3.2. StandardScaler

Estandarización: Escala las características para que tengan media 0 y desviación estándar 1 utilizando la fórmula (x - mean) / std.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
data_df_standardized = pd.DataFrame(scaler.fit_transform(data_df), columns=data_df
.columns)
```

```
# Ejemplo 1
age_standardized = data_df_standardized['age']
print(age_standardized.head())

# Ejemplo 2
fare_standardized = data_df_standardized['fare']
print(fare_standardized.head())
```

La principal diferencia entre `MinMaxScaler` y `StandardScaler` es cómo ajustan la escala de las características. MinMaxScaler ajusta los valores de las características a un rango específico, generalmente [0, 1], mientras que StandardScaler ajusta la distribución de las características para que tengan una media de 0 y una desviación estándar de 1.

3.4. Codificación de variables categóricas

Las variables categóricas son aquellas que tienen un número limitado de categorías o grupos. Los algoritmos de aprendizaje automático generalmente requieren que las variables categóricas se conviertan en una representación numérica. Hay varios métodos para codificar variables categóricas, como la codificación one-hot y la codificación ordinal.

Codificación One-Hot: Crea nuevas características binarias para cada categoría de la variable categórica.

```
# One-Hot Encoding con Pandas
data_df_one_hot = pd.get_dummies(data_df, columns=['sex'])

# Ejemplo 1
print(data_df_one_hot.head())

# Ejemplo 2
data_df_one_hot_embarked = pd.get_dummies(data_df, columns=['embarked'])
print(data_df_one_hot_embarked.head())
```

Codificación Ordinal: Asigna un número entero a cada categoría de la variable categórica según un orden específico.

```
from sklearn.preprocessing import OrdinalEncoder

encoder = OrdinalEncoder()
data_df_ordinal = data_df.copy()
data_df_ordinal[['embarked']] = encoder.fit_transform(data_df_ordinal[['embarked']])

# Ejemplo 1
print(data_df_ordinal[['embarked']].head())

# Ejemplo 2
```

```
data_df_ordinal_class = data_df_ordinal.copy()
data_df_ordinal_class[['class']] = encoder.fit_transform(data_df_ordinal_class
[['class']])
print(data_df_ordinal_class[['class']].head())
```

3.5. Creación de nuevas características

La **creación de nuevas características** implica combinar o modificar las características existentes para obtener información adicional que puede ser útil para los algoritmos de aprendizaje automático.

Estas nuevas características pueden mejorar el rendimiento del modelo al proporcionar información relevante no capturada por las características originales.

- **Características polinomiales**: Combina las características existentes para crear características polinomiales de mayor grado.

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, include_bias=False)
data_df_poly = pd.DataFrame(poly.fit_transform(data_df), columns=poly
.get_feature_names(data_df.columns))

# Ejemplo 1
age_fare_interaction = data_df_poly['age fare']
print(age_fare_interaction.head())

# Ejemplo 2
fare_squared = data_df_poly['fare^2']
print(fare_squared.head())
```

Características de ingeniería de dominio: Crea nuevas características basadas en el conocimiento específico del dominio.

```
# Ejemplo 1: Crear una nueva característica 'family_size' sumando 'sibsp' y 'parch'
data_df_domain1 = data_df.copy()
data_df_domain1['family_size'] = data_df_domain1['sibsp'] + data_df_domain1['parch']
print(data_df_domain1[['family_size']].head())

# Ejemplo 2: Crear una nueva característica 'is_alone' basada en 'family_size'
data_df_domain2 = data_df_domain1.copy()
data_df_domain2['is_alone'] = data_df_domain2['family_size'] == 0
print(data_df_domain2[['is_alone']].head())
```

Estos son algunos ejemplos de transformaciones de datos que pueden realizarse en un conjunto de datos.

La elección de las transformaciones adecuadas dependerá del problema específico y del conocimiento del dominio.

Es importante experimentar con diferentes transformaciones y evaluar su impacto en el rendimiento del modelo.

4. Ejemplo 3 (Heart Disease UCI)

Vamos a trabajar con el **conjunto de datos "Heart Disease UCI"** disponible en el repositorio de Machine Learning de la UCI.

Este conjunto de datos contiene información sobre pacientes y varios atributos médicos, junto con la presencia o ausencia de enfermedades del corazón.

Puedes encontrar el conjunto de datos aquí: https://archive.ics.uci.edu/ml/datasets/Heart+Disease

Empezaremos aplicando las técnicas de preprocesamiento de datos en el orden en que las mencionamos anteriormente:

- Carga y exploración de datos
- Limpieza de datos
- Transformación de datos
- Binning
- Extracción de características temporales (N/A en este caso)
- PCA y selección de características

4.1. Cargar datos

Primero, necesitaremos cargar las bibliotecas necesarias e importar el conjunto de datos:

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer, KNNImputer
from sklearn.preprocessing import StandardScaler, MinMaxScaler, KBinsDiscretizer
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, chi2

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/heart-
disease/processed.cleveland.data"

column_names = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target']

df = pd.read_csv(url, header=None, names=column_names, na_values="?")
```

4.2. Exploración de datos

```
print(df.head())
print(df.info())
print(df.describe())
```

4.3. Limpieza de datos

```
# Reemplazar valores faltantes con la mediana
imputer = SimpleImputer(strategy='median')
df['ca'] = imputer.fit_transform(df['ca'].values.reshape(-1, 1))
df['thal'] = imputer.fit_transform(df['thal'].values.reshape(-1, 1))

# Alternativa: Imputación K-NN
knn_imputer = KNNImputer(n_neighbors=3)
df[['ca', 'thal']] = knn_imputer.fit_transform(df[['ca', 'thal']])
```

4.4. Transformación de datos

```
# Escalar características numéricas
scaler = StandardScaler()
df[['age', 'trestbps', 'chol', 'thalach', 'oldpeak']] = scaler.fit_transform(df[[
'age', 'trestbps', 'chol', 'thalach', 'oldpeak']])

# Alternativa: MinMaxScaler
minmax_scaler = MinMaxScaler()
df[['age', 'trestbps', 'chol', 'thalach', 'oldpeak']] = minmax_scaler.fit_transform(
df[['age', 'trestbps', 'chol', 'thalach', 'oldpeak']])
```

4.5. Binning

```
# Binning para la característica 'age'
bins = KBinsDiscretizer(n_bins=4, encode='ordinal', strategy='uniform')
df['age'] = bins.fit_transform(df['age'].values.reshape(-1, 1))
```

4.6. PCA y selección de características

```
# Reducción de dimensionalidad con PCA
pca = PCA(n_components=8)
df_pca = pca.fit_transform(df.drop('target', axis=1))
```

4.7. Selección de características utilizando SelectKBest

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

kbest = SelectKBest(chi2, k=8)
X_kbest = kbest.fit_transform(X, y)
```

Después de aplicar todas estas técnicas de preprocesamiento de datos, el conjunto de datos está listo para ser utilizado en un algoritmo de aprendizaje automático.

No todas las técnicas son aplicables o útiles en todos los conjuntos de datos. Por lo tanto, es importante experimentar con diferentes enfoques y evaluar su impacto en el rendimiento del modelo para determinar las mejores técnicas para cada situación específica.