UNIVERSIDAD AMERICANA

Preprocesamiento y Transformación de Datos

Inteligencia de Negocios

Entrega:

- Lester Alejandro Rodriguez Cuevas

Docente: Arlen Jeannette Lopez

16 de Septiembre

Managua, Nicaragua

El presente preprocesamiento de datos, se realizó en un archivo jupyter .ipynb, se utilizó un archivo de tipo .csv con el nombre de 'train', donde se tiene toda la data e información de ventas de casas en un sector.

Declaración de librerías

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Este bloque de código importa las bibliotecas más comunes para el análisis de datos y la visualización en Python:

- pandas para manipulación de datos,
- numpy para cálculos numéricos,
- matplotlib y seaborn para visualización de datos, ya sea mediante gráficos simples o gráficos más complejos y atractivos.

Estos paquetes forman parte del ecosistema de ciencia de datos de Python, que te permite manejar datos, transformarlos y visualizarlos.

Carga de Datos

```
data = pd.read_csv('train.csv')
print('Primeras filas del dataset')
print(data.head())
```

Carga el archivo **train.csv** en un DataFrame llamado data.

Imprime un mensaje indicando que se mostrarán las primeras filas del conjunto de datos.

Utiliza **data.head()** para mostrar las primeras 5 filas del archivo CSV cargado, lo cual es útil para verificar la estructura y contenido de los datos.

Exploración Inicial

```
# Ver el tamaño del dataset
print('Tamaño del dataset')
print(data.shape)

# Obtener información sobre los tipos de datos
print('Tipos de datos')
print(data.info())

# Describir las variables numéricas
print('Descripción de las variables numéricas')
print(data.describe())
```

```
Primeras filas del dataset
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0   1          60       RL         65.0     8450   Pave   NaN      Reg
1   2          20       RL         80.0     9600   Pave   NaN      Reg
2   3          60       RL         68.0    11250   Pave   NaN      IR1
3   4          70       RL         60.0     9550   Pave   NaN      IR1
4   5          60       RL         84.0    14260   Pave   NaN      IR1

  LandContour Utilities  ... PoolArea PoolQC Fence MiscFeature MiscVal MoSold  \
0         Lvl    AllPub  ...        0    NaN   NaN         NaN       0      2
1         Lvl    AllPub  ...        0    NaN   NaN         NaN       0      5
2         Lvl    AllPub  ...        0    NaN   NaN         NaN       0      9
3         Lvl    AllPub  ...        0    NaN   NaN         NaN       0      2
4         Lvl    AllPub  ...        0    NaN   NaN         NaN       0     12

  YrSold  SaleType  SaleCondition  SalePrice
0   2008        WD         Normal     208500
1   2007        WD         Normal     181500
2   2008        WD         Normal     223500
3   2006        WD        Abnorml     140000
4   2008        WD         Normal     250000
```

```
[5 rows x 81 columns]
Tamaño del dataset
(1460, 81)
Tipos de datos
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
```

```
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
None
Descripción de las variables numéricas
                Id   MSSubClass  LotFrontage        LotArea  OverallQual  \
count  1460.000000  1460.000000  1201.000000    1460.000000  1460.000000
mean    730.500000    56.897260    70.049958   10516.828082     6.099315
std     421.610009    42.300571    24.284752    9981.264932     1.382997
min       1.000000    20.000000    21.000000    1300.000000     1.000000
25%     365.750000    20.000000    59.000000    7553.500000     5.000000
50%     730.500000    50.000000    69.000000    9478.500000     6.000000
75%    1095.250000    70.000000    80.000000   11601.500000     7.000000
max    1460.000000   190.000000   313.000000  215245.000000    10.000000

       OverallCond    YearBuilt  YearRemodAdd   MasVnrArea   BsmtFinSF1  ...  \
count  1460.000000  1460.000000   1460.000000  1452.000000  1460.000000  ...
mean      5.575342  1971.267808   1984.865753   103.685262   443.639726  ...
std       1.112799    30.202904     20.645407   181.066207   456.098091  ...
min       1.000000  1872.000000   1950.000000     0.000000     0.000000  ...
25%       5.000000  1954.000000   1967.000000     0.000000     0.000000  ...
50%       5.000000  1973.000000   1994.000000     0.000000   383.500000  ...
75%       6.000000  2000.000000   2004.000000   166.000000   712.250000  ...
max       9.000000  2010.000000   2010.000000  1600.000000  5644.000000  ...
```

```
          PoolArea       MiscVal       MoSold       YrSold      SalePrice
count  1460.000000   1460.000000  1460.000000  1460.000000    1460.000000
mean      2.758904     43.489041     6.321918  2007.815753  180921.195890
std      40.177307    496.123024     2.703626     1.328095   79442.502883
min       0.000000      0.000000     1.000000  2006.000000   34900.000000
25%       0.000000      0.000000     5.000000  2007.000000  129975.000000
50%       0.000000      0.000000     6.000000  2008.000000  163000.000000
75%       0.000000      0.000000     8.000000  2009.000000  214000.000000
max     738.000000  15500.000000    12.000000  2010.000000  755000.000000

[8 rows x 38 columns]
```

Este código realiza un análisis exploratorio básico:

1. **Tamaño del dataset**: Muestra cuántas filas y columnas tiene el DataFrame. Este Data frame cuenta con 1460 registros y 81 columnas
2. **Tipos de datos**: Proporciona detalles sobre los tipos de datos de cada columna y el número de valores no nulos. Vemos que hay datos de tipo entero, tipo objeto y tipo float
3. **Descripción de variables numéricas**: Resume las estadísticas clave (como media, mediana, máximo, etc.) para las columnas numéricas del DataFrame.

Identificación de variables para el análisis

```
#lotarea,neighborhood, overallqual, overallcond, yearbuilt, yearremodadd, grlivarea,garagecars,yr sold, saleprice
important_columns = ['LotArea','Neighborhood','OverallQual','OverallCond','YearBuilt','YearRemodAdd','GrLivArea','GarageCars','YrSold','SalePrice']
print(data[important_columns].head())
```

LotArea: Representa el **área total del terreno** en pies cuadrados de la propiedad.Es una métrica importante ya que el tamaño del lote puede influir significativamente en el precio de la propiedad (SalePrice).

Neighborhood: Se refiere al **vecindario** en el que se encuentra la propiedad. Es una variable categórica que identifica el área o subdivisión en la que se ubica la vivienda.El vecindario puede influir en el valor de la propiedad, ya que las ubicaciones más deseables o prestigiosas suelen tener precios más altos.

OverallQual (Overall Quality): Mide la **calidad general** de los materiales y la mano de obra de la casa, en una escala ordinal de 1 a 10, donde 1 es la peor calidad y 10 la mejor. Una mayor calidad de construcción generalmente está asociada con un precio más alto.

OverallCond (Overall Condition): Se refiere a la **condición general** de la casa, también en una escala de 1 a 10, donde 1 indica una condición muy pobre y 10 es excelente. La condición física de la casa influye en su valor; por ejemplo, casas en mal estado pueden venderse por menos, incluso si tienen otras características valiosas.

YearBuilt: Año en que la casa fue **construida**. Las casas más antiguas pueden requerir más mantenimiento, lo que podría afectar su valor, mientras que las casas más nuevas pueden tener características modernas que las hacen más valiosas.

YearRemodAdd (Year Remodeled or Added): Año en que la casa fue **remodelada o ampliada** por última vez. Las remodelaciones recientes pueden aumentar el valor de una casa, especialmente si han modernizado aspectos clave como la cocina, los baños o las áreas comunes.

GrLivArea (Ground Living Area): Se refiere a la **superficie habitable sobre el nivel del suelo** (en pies cuadrados). Cuanta más área habitable tenga una casa, mayor será su valor, ya que los compradores suelen preferir casas con más espacio.

GarageCars: Representa el **número de plazas para autos en el garaje** de la casa. Tener más espacio en el garaje puede incrementar el valor de una propiedad, ya que es una característica conveniente, especialmente en zonas suburbanas.

9. YrSold (Year Sold): Año en que la casa fue **vendida**. Esto puede ser útil para ver cómo ha cambiado el mercado inmobiliario con el tiempo o para analizar tendencias de precios en función del año de venta.

10. SalePrice: Precio de venta de la propiedad, en dólares. Es la variable objetivo o dependiente que generalmente se intenta predecir en los análisis de precios de viviendas. La mayoría de los análisis giran en torno a entender qué características (como el tamaño del lote, la calidad, la ubicación, etc.) afectan más al precio de venta.

```
# Obtener información sobre los tipos de datos
print('Información sobre los tipos de datos')
print(data[important_columns].info())

#Describir las variables numéricas
print('Describe variables numericas')
print(data[important_columns].describe())
#Proporcion de valores nulos
print('Proporcion de valores nulos')
print(data.isnull().sum()/data.shape[0])
```

Acá realizamos de nuevo una exploración de datos, pero esta vez para los datos o columnas importantes que seleccionamos, donde **important_columns** es el arreglo de las columnas que seleccionamos, de este modo hacemos una exploración más específica con las datos que vamos a trabajar.

```
   LotArea Neighborhood  OverallQual  OverallCond  YearBuilt  YearRemodAdd  \
0     8450      CollgCr            7            5       2003          2003
1     9600      Veenker            6            8       1976          1976
2    11250      CollgCr            7            5       2001          2002
3     9550      Crawfor            7            5       1915          1970
4    14260      NoRidge            8            5       2000          2000

   GrLivArea  GarageCars  YrSold  SalePrice
0       1710           2    2008     208500
1       1262           2    2007     181500
2       1786           2    2008     223500
3       1717           3    2006     140000
4       2198           3    2008     250000
Información sobre los tipos de datos
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   LotArea       1460 non-null   int64
 1   Neighborhood  1460 non-null   object
 2   OverallQual   1460 non-null   int64
 3   OverallCond   1460 non-null   int64
 4   YearBuilt     1460 non-null   int64
 5   YearRemodAdd  1460 non-null   int64
...
SaleType        0.000000
SaleCondition   0.000000
SalePrice       0.000000
Length: 81, dtype: float64
```

```
missing_values = data[important_columns].isnull().sum()
print(missing_values[missing_values >= 0])
print('Las columnas no tienen valores nulos')
```

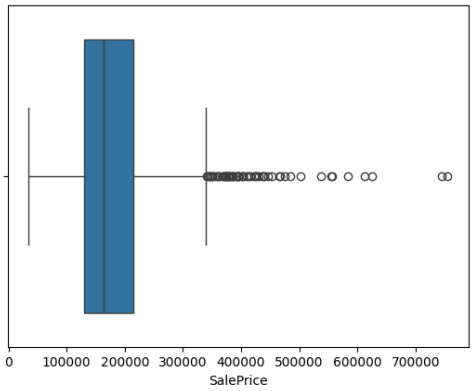
```
[185] ✓ 0.0s

LotArea         0
Neighborhood    0
OverallQual     0
OverallCond     0
YearBuilt       0
YearRemodAdd    0
GrLivArea       0
GarageCars      0
YrSold          0
SalePrice       0
dtype: int64
Las columnas no tienen valores nulos
```

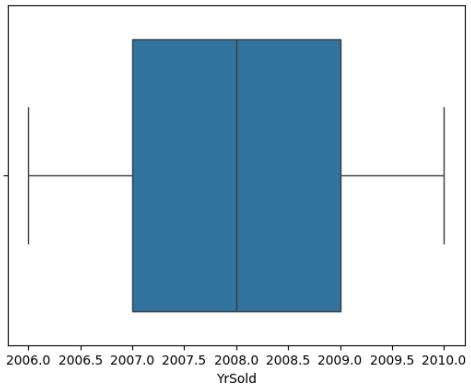
Acá observamos que las columnas importante no tienen valores nulos

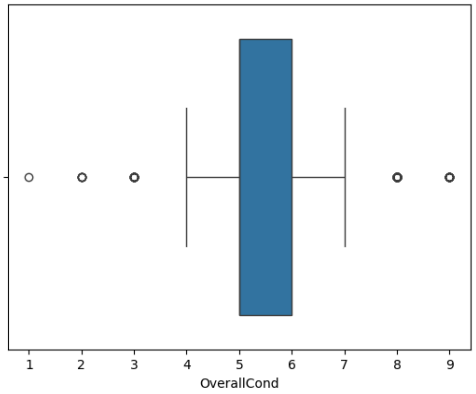
```
print('Distribución de la variable SalePrice')
sns.boxplot(x=data[important_columns]['SalePrice'])
plt.show()
```

```
✓ 0.2s
```

Seguido de eso, realizamos gráficos de distribución para todas las columnas numéricas, esto para ver cómo estaba el comportamiento de los outliers en cada una de ellas, eso nos iba a servir para saber a cuales columnas debíamos tratar sus debidos outliers, aquí adjunto ejemplos de los gráficos que realizamos. (No adjunto todos los gráficos ya que más adelante en la comparación los adjuntare todos):







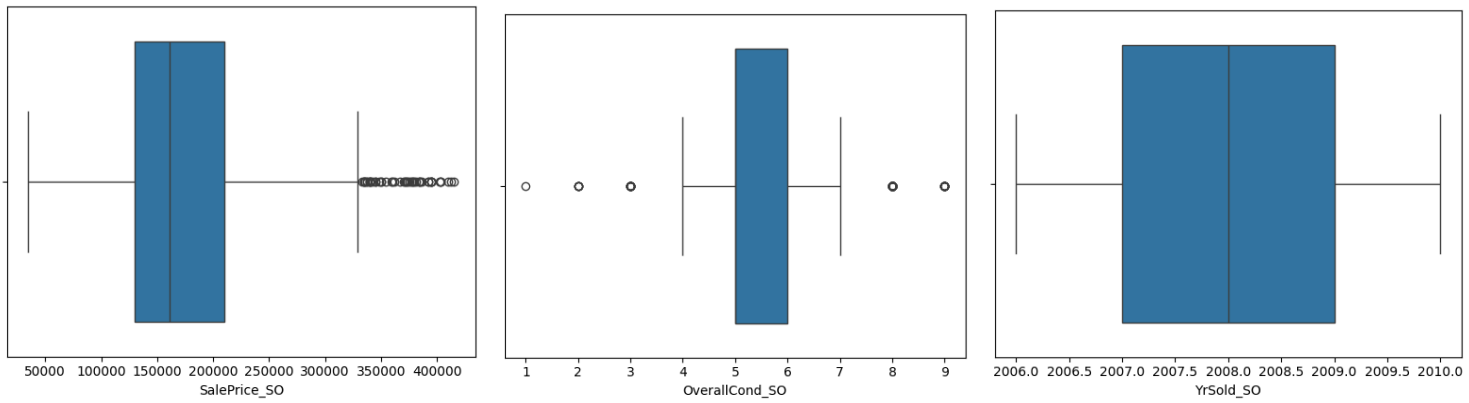
Ejemplos en los cuales vemos outliers abundantes del lado derecho, grafico sin outliers y uno que posee outliers a ambos lados.

Tratamiento de Outliers

```python
from scipy import stats

# Filtrar y conservar los valores sin outliers basados en Z-score
data['SalePrice_SO'] = data['SalePrice'][(np.abs(stats.zscore(data['SalePrice'])) < 3)]
data['LotArea_SO'] = data['LotArea'][(np.abs(stats.zscore(data['LotArea'])) < 5)]
data['OverallQual_SO'] = data['OverallQual'][(np.abs(stats.zscore(data['OverallQual'])) < 3)]
data['OverallCond_SO'] = data['OverallCond'][(np.abs(stats.zscore(data['OverallCond'])) < 5)]
data['YearBuilt_SO'] = data['YearBuilt'][(np.abs(stats.zscore(data['YearBuilt'])) < 3)]
data['YearRemodAdd_SO'] = data['YearRemodAdd'][(np.abs(stats.zscore(data['YearRemodAdd'])) < 3)]
data['GrLivArea_SO'] = data['GrLivArea'][(np.abs(stats.zscore(data['GrLivArea'])) < 3)]
data['GarageCars_SO'] = data['GarageCars'][(np.abs(stats.zscore(data['GarageCars'])) < 3)]
data['YrSold_SO'] = data['YrSold'][(np.abs(stats.zscore(data['YrSold'])) < 3)]
```

Z-score es útil para detectar outliers porque estandariza los valores de una columna para que puedas ver cuán alejados están de la media. Si un valor tiene un Z-score muy alto o muy bajo, es probable que sea un valor atípico en relación con el resto de los datos.

En el código se están limpiando varias columnas de datos eliminando aquellos valores que están demasiado lejos del promedio en términos de desviaciones estándar. Esto te ayuda a quedarte con un conjunto de datos más "normalizado", sin la influencia de valores extremos. En dos casos como el OverallCondition y el LotArea, ya que intente mantener el rango de 3, sin embargo observaba que tenía muchos outliers aun asi que amplíe el límite.



Transformación de Variables

```python
#lotarea,neighborhood, overallqual, overallcond, yearbuilt, yearremodadd, grlivarea,garag

scaler = StandardScaler()
min_max_scaler = MinMaxScaler()

data['SalePrice_scaled'] = scaler.fit_transform(data[['SalePrice']])
data['SalePrice_normalized'] = min_max_scaler.fit_transform(data[['SalePrice']])
print(data[['SalePrice', 'SalePrice_scaled', 'SalePrice_normalized']].head())

data['LotArea_scaled'] = scaler.fit_transform(data[['LotArea']])
data['LotArea_normalized'] = min_max_scaler.fit_transform(data[['LotArea']])
print(data[['LotArea', 'LotArea_scaled', 'LotArea_normalized']].head())

data['OverallQual_scaled'] = scaler.fit_transform(data[['OverallQual']])
data['OverallQual_normalized'] = min_max_scaler.fit_transform(data[['OverallQual']])
print(data[['OverallQual', 'OverallQual_scaled', 'OverallQual_normalized']].head())

data['OverallCond_scaled'] = scaler.fit_transform(data[['OverallCond']])
data['OverallCond_normalized'] = min_max_scaler.fit_transform(data[['OverallCond']])
print(data[['OverallCond', 'OverallCond_scaled', 'OverallCond_normalized']].head())

data['YearBuilt_scaled'] = scaler.fit_transform(data[['YearBuilt']])
data['YearBuilt_normalized'] = min_max_scaler.fit_transform(data[['YearBuilt']])
print(data[['YearBuilt', 'YearBuilt_scaled', 'YearBuilt_normalized']].head())

data['YearRemodAdd_scaled'] = scaler.fit_transform(data[['YearRemodAdd']])
data['YearRemodAdd_normalized'] = min_max_scaler.fit_transform(data[['YearRemodAdd']])
print(data[['YearRemodAdd', 'YearRemodAdd_scaled', 'YearRemodAdd_normalized']].head())

data['GrLivArea_scaled'] = scaler.fit_transform(data[['GrLivArea']])
data['GrLivArea_normalized'] = min_max_scaler.fit_transform(data[['GrLivArea']])
print(data[['GrLivArea', 'GrLivArea_scaled', 'GrLivArea_normalized']].head())
```

```
   SalePrice  SalePrice_scaled  SalePrice_normalized
0     208500          0.347273              0.241078
1     181500          0.007288              0.203583
2     223500          0.536154              0.261908
3     140000         -0.515281              0.145952
4     250000          0.869843              0.298709
   LotArea  LotArea_scaled  LotArea_normalized
0     8450       -0.207142            0.033420
1     9600       -0.091886            0.038795
2    11250        0.073480            0.046507
3     9550       -0.096897            0.038561
4    14260        0.375148            0.060576
   OverallQual  OverallQual_scaled  OverallQual_normalized
0            7            0.651479                0.666667
1            6           -0.071836                0.555556
2            7            0.651479                0.666667
3            7            0.651479                0.666667
4            8            1.374795                0.777778
   OverallCond  OverallCond_scaled  OverallCond_normalized
0            5           -0.517200                   0.500
1            8            2.179628                   0.875
2            5           -0.517200                   0.500
3            5           -0.517200                   0.500
4            5           -0.517200                   0.500
   YearBuilt  YearBuilt_scaled  YearBuilt_normalized
...
1     2007       -0.614439              0.25
2     2008        0.138777              0.50
3     2006       -1.367655              0.00
4     2008        0.138777              0.50
```

Escalado con StandardScaler: Cada columna es transformada para que sus valores tengan una distribución con media 0 y desviación estándar 1. El resultado se guarda en una nueva columna con la terminacion _scaled.

Normalización con MinMaxScaler: Cada columna es transformada para que sus valores se ubiquen entre 0 y 1. El resultado se guarda en una nueva columna con la terminacion _normalized.
Columnas procesadas:
SalePrice, LotArea, OverallQual, OverallCond, YearBuilt, YearRemodAdd, GrLivArea, GarageCars, YrSold.
Cada paso imprime las primeras filas de las columnas originales, escaladas y normalizadas para ver el resultado.
Normalización es buena cuando las variables tienen diferentes escalas y no siguen una distribución normal.
La estandarización es preferible cuando los datos siguen una distribución normal y se usan algoritmos que se benefician de la simetría y la homogeneidad de las variables.

Análisis de asimetría

```
#lotarea,neighborhood, overallqual, overallcond, yearbuilt, yearre

skewnessSale = stats.skew(data['SalePrice_normalized'])
print('Skewness SalePrice:', skewnessSale)

skewnessLot = stats.skew(data['LotArea_normalized'])
print('Skewness LotArea:', skewnessLot)

skewnessQual = stats.skew(data['OverallQual_normalized'])
print('Skewness OverallQual:', skewnessQual)

skewnessCond = stats.skew(data['OverallCond_normalized'])
print('Skewness OverallCond:', skewnessCond)

skewnessBuilt = stats.skew(data['YearBuilt_normalized'])
print('Skewness YearBuilt:', skewnessBuilt)

skewnessRemod = stats.skew(data['YearRemodAdd_normalized'])
print('Skewness YearRemodAdd:', skewnessRemod)

skewnessGrLiv = stats.skew(data['GrLivArea_normalized'])
print('Skewness GrLivArea:', skewnessGrLiv)

skewnessGarage = stats.skew(data['GarageCars_normalized'])
print('Skewness GarageCars:', skewnessGarage)

skewnessSold = stats.skew(data['YrSold_normalized'])
print('Skewness YrSold:', skewnessSold)
```

```
Skewness SalePrice: 1.8809407460340357
Skewness LotArea: 12.195142125084478
Skewness OverallQual: 0.2167209765258635
Skewness OverallCond: 0.6923552135520978
Skewness YearBuilt: -0.6128307242029024
Skewness YearRemodAdd: -0.5030444967598083
Skewness GrLivArea: 1.3651559547734349
Skewness GarageCars: -0.34219689543081294
Skewness YrSold: 0.09616957961803625
```

Este código calcula y muestra la **asimetría** (skewness) de varias columnas normalizadas del dataset usando la función stats.skew(). La asimetría mide la falta de simetría en la distribución de los datos: un valor cercano a 0 indica una distribución simétrica, valores positivos indican una cola más larga hacia la derecha (asimetría positiva), y valores negativos una cola más larga hacia la izquierda (asimetría negativa). Aquí se calcula la asimetría para las columnas normalizadas SalePrice, LotArea, OverallQual, OverallCond, YearBuilt, YearRemodAdd, GrLivArea, GarageCars, y YrSold para entender cómo están distribuidos los datos después de la normalización. Esto lo hacemos para decidir cuáles columnas son las que requieren transformaciones Logarítmicas.

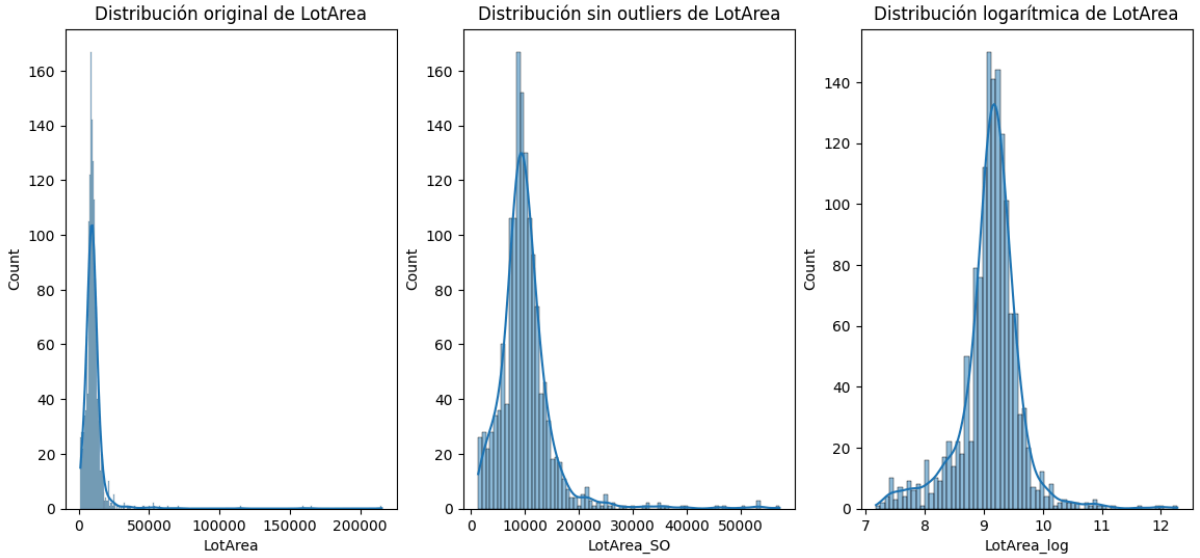
Transformaciones Logarítmicas

Las variables a las que les hacemos transformaciones son aquellas que su asimetria es menor a -0.5 y mayor a 0.5, en este caso seleccionamos SalePrice, LotArea, OverallCond, YearBuilt y GrLivArea.
Presentaré un ejemplo de la estructura que se siguió para hacer la transformación y representarla gráficamente.

```python
# Aplicar la transformación logarítmica a la columna 'LoArea'
data['LotArea_log'] = np.log1p(data['LotArea'])  # log1p para manejar ceros

# Verificar que la transformación se haya aplicado correctamente
print(data[['LotArea', 'LotArea_SO', 'LotArea_log']].head())

# Graficar la distribución antes y después de la transformación
def plot_distribution(original, no_outliers ,transformed, column):
    plt.figure(figsize=(14, 6))

    # Gráfico de la variable original
    plt.subplot(1, 3, 1)
    sns.histplot(original, kde=True)
    plt.title(f'Distribución original de {column}')

    #Gráfico de la variable sin outliers
    plt.subplot(1, 3, 2)
    sns.histplot(no_outliers, kde=True)
    plt.title(f'Distribución sin outliers de {column}')


    # Gráfico de la variable transformada
    plt.subplot(1, 3, 3)
    sns.histplot(transformed, kde=True)
    plt.title(f'Distribución logarítmica de {column}')

    plt.show()

# Mostrar el gráfico para 'LotArea'
plot_distribution(data['LotArea'], data['LotArea_SO'], data['LotArea_log'], 'LotArea')
```

```python
# Evaluar el skewness después de la transformación
skewness_log = stats.skew(data['LotArea_log'].dropna())
print(f'Skewness de LotArea después de la transformación: {skewness_log}')
```
1.2s

Este código aplica una transformación logarítmica a la columna LotArea para reducir su asimetría (skewness), utilizando np.log1p() para manejar posibles valores cero. Luego, se comparan las distribuciones originales, sin outliers, y transformadas con gráficos de histograma utilizando seaborn y matplotlib. Finalmente, se calcula y muestra la asimetría de la columna transformada para verificar si la distribución mejoró después de la transformación.
Representación gráfica de la columna LotArea, para poder observar el cambio que se produce.



Skewness de LotArea después de la transformación: -0.13726327193353463

Comparación y Diferencias con el gráfico

La distribución original de LotArea presenta un fuerte sesgo positivo, con una cola muy larga hacia la derecha debido a la presencia de outliers relacionados con lotes muy grandes. Al eliminar estos valores extremos en la distribución sin outliers, se reduce significativamente la longitud de la cola, haciendo que la distribución sea más compacta, aunque todavía mantiene un ligero sesgo. Por otro lado, la distribución logarítmica, obtenida tras aplicar una transformación logarítmica, normaliza los datos, volviéndolos mucho más simétricos y acercándolos a una distribución normal, lo cual es particularmente útil para
modelos estadísticos y de machine learning.

Creación de nuevas variables

```python
#creacion de columna nueva
data['Price_per_Unit'] = data['SalePrice'] / data['LotArea']

# Verificar que la columna se haya creado correctamente
print(data[['SalePrice', 'LotArea', 'Price_per_Unit']].head())

#grlivarea,garagecars
data['GrLivArea_per_Car'] = data['GrLivArea_SO'] / data['GarageCars_SO']
print(data[['GrLivArea_SO', 'GarageCars_SO', 'GrLivArea_per_Car']].head())
```

✓ 0.0s

```
   SalePrice  LotArea  Price_per_Unit
0     208500     8450       24.674556
1     181500     9600       18.906250
2     223500    11250       19.866667
3     140000     9550       14.659686
4     250000    14260       17.531557
   GrLivArea_SO  GarageCars_SO  GrLivArea_per_Car
0        1710.0              2         855.000000
1        1262.0              2         631.000000
2        1786.0              2         893.000000
3        1717.0              3         572.333333
4        2198.0              3         732.666667
```

Price_per_Unit: Calcula el precio por unidad de área dividiendo el valor de SalePrice entre LotArea. Luego, imprime las primeras filas de las columnas involucradas (SalePrice, LotArea, y Price_per_Unit) para verificar que la columna se haya creado correctamente.
GrLivArea_per_Car: Calcula el área habitable por cada automóvil dividiendo GrLivArea_SO entre GarageCars_SO. También imprime las primeras filas para confirmar la creación de esta nueva columna.

Codificación de variables Categóricas

La única variable categórica que seleccionamos fue Neighborhood, la cual posee 24 diferentes vecindarios registrados, en los cuales cada casa puede ser parte de 1

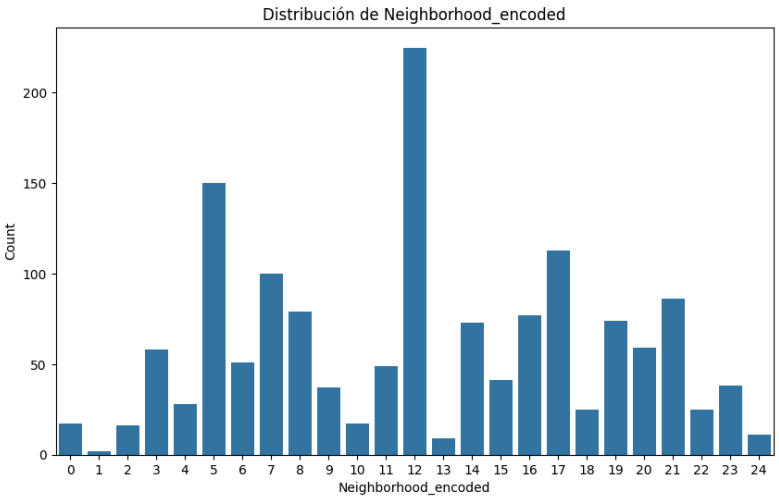
```python
from sklearn.preprocessing import LabelEncoder

# Crear el codificador
label_encoder = LabelEncoder()

# Aplicar Label Encoding a la columna 'Neighborhood'
data['Neighborhood_encoded'] = label_encoder.fit_transform(data['Neighborhood'])

# Verificar que la columna se haya codificado correctamente
print(data[['Neighborhood', 'Neighborhood_encoded']].head())
```

✓ 0.0s

```
  Neighborhood  Neighborhood_encoded
0      CollgCr                    5
1      Veenker                   24
2      CollgCr                    5
3      Crawfor                    6
4      NoRidge                   15
```

```python
plt.figure(figsize=(10, 6))

sns.countplot(x='Neighborhood_encoded', data=data)

# Título y etiquetas
plt.title('Distribución de Neighborhood_encoded')
plt.xlabel('Neighborhood_encoded')
plt.ylabel('Count')

# Mostrar el gráfico
plt.show()
label_encoder = LabelEncoder()
data['Neighborhood_encoded'] = label_encoder.fit_transform(data['Neighborhood'])

# Crear un DataFrame para mostrar la correspondencia
mapping = pd.DataFrame({'Neighborhood': data['Neighborhood'], 'Neighborhood_encoded': data['Neighborhood_encoded']})

# Eliminar duplicados para mostrar solo las correspondencias únicas
mapping_unique = mapping.drop_duplicates().sort_values(by='Neighborhood_encoded')

# Mostrar la relación entre los nombres originales y los valores codificados
print(mapping_unique)
```

Este código primero visualiza la distribución de la columna Neighborhood_encoded mediante un gráfico de barras utilizando sns.countplot. Luego, utiliza LabelEncoder para convertir los valores categóricos de la columna Neighborhood en valores numéricos y los guarda en una nueva columna llamada Neighborhood_encoded. Después, crea un DataFrame que muestra la correspondencia entre los valores originales y los codificados, eliminando duplicados para mostrar solo las correspondencias únicas, y finalmente imprime este DataFrame para ver la relación entre vecindarios y sus códigos.



	Neighborhood	Neighborhood_encoded
219	Blmngtn	0
599	Blueste	1
225	BrDale	2
9	BrkSide	3
69	ClearCr	4
0	CollgCr	5
3	Crawfor	6
39	Edwards	7
50	Gilbert	8
21	IDOTRR	9
23	MeadowV	10
5	Mitchel	11
14	NAmes	12
126	NPkVill	13
7	NWAmes	14
4	NoRidge	15
11	NridgHt	16
8	OldTown	17
267	SWISU	18
10	Sawyer	19
18	SawyerW	20
6	Somerst	21
58	StoneBr	22
41	Timber	23
1	Veenker	24

Aca podemos observar la variable Neighborhood que fue la que codificamos por etiquetas numéricas, en la grafica podemos observar la distribución de los datos y en la tabla podemos observar la correspondencia de los datos originales con los codificados

Datos Originales:

- Valores sin escalado ni normalización: Las columnas como SalePrice, LotArea, OverallQual, entre otras, contenían valores en su escala original. Algunas podían tener una alta varianza o distribución sesgada.
- Outliers presentes: Los valores extremos en columnas como LotArea, SalePrice, y otras, no habían sido eliminados ni ajustados.
- Columnas categóricas sin codificar: La columna Neighborhood, entre otras, contenía valores categóricos sin transformación, lo que dificultaba su uso en modelos de machine learning.

Datos Preprocesados:

- Escalado y normalización: Se aplicaron tanto el StandardScaler como el MinMaxScaler para las columnas seleccionadas, transformando los datos a una distribución estandarizada o entre un rango de 0 y 1, respectivamente. Esto es clave para algoritmos sensibles a la escala de los datos.
- Eliminación de outliers: Usando el Z-score, se han eliminado los valores extremos en varias columnas, creando versiones de las mismas sin outliers (sufijo _SO).
- Transformación logarítmica: Se aplicó una transformación logarítmica a columnas como LotArea para reducir la asimetría y mejorar la normalidad en la distribución de los datos.
- Creación de nuevas columnas: Se añadieron nuevas variables, como Price_per_Unit (relación entre SalePrice y LotArea), y GrLivArea_per_Car (relación entre GrLivArea_SO y GarageCars_SO), que ofrecen nuevas perspectivas y relaciones dentro de los datos.
- Codificación de etiquetas: Se convirtió la columna categórica Neighborhood en valores numéricos mediante LabelEncoder, permitiendo su uso en algoritmos que requieren entradas numéricas.