

# carga\_limpieza

June 13, 2025

## 1 Carga y Limpieza de Datos de Viviendas en Andalucía

Proceso inicial de carga y limpieza de datos del conjunto de datos de viviendas en venta en Andalucía, preparando los datos para análisis posteriores.

Autor: Daniel Carrera Bonilla Trabajo Final de Grado

### 1.1 Cargar Librerías y Datos Crudos

Importación de librerías necesarias para el procesamiento de datos y carga del dataset desde el archivo CSV original.

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import os

[2]: # He añadido esto para poder configurar el entorno correctamente
%matplotlib inline
plt.style.use('seaborn-v0_8-whitegrid')
pd.set_option('display.max_columns', None)

# Definir rutas de archivos donde quiero que se guarden los datos
data_dir = '../data'
raw_data_dir = os.path.join(data_dir, 'processed')
clean_data_dir = os.path.join(data_dir, 'clean')

# Quiero que los datos se guarden en un directorio específico
if not os.path.exists(clean_data_dir):
    os.makedirs(clean_data_dir)

# Cargamos el dataset de Andalucía raw
try:
    file_path = os.path.join(raw_data_dir, 'andalucia_rawv2.csv')
    df_raw = pd.read_csv(file_path)
    print(f"Dataset cargado exitosamente desde: {file_path}")
    print(f"Dimensiones del dataset: {df_raw.shape}")
```

```
except FileNotFoundError:
    print(f"Error: No se pudo encontrar el archivo en {file_path}")
    df_raw = pd.DataFrame()
```

Dataset cargado exitosamente desde: ../data\processed\andalucia\_rawv2.csv  
Dimensiones del dataset: (369240, 9)

```
[3]: # Mostrar las primeras filas del DataFrame para inspección inicial
if not df_raw.empty:
    print("Primeras 5 filas del conjunto de datos:")
    display(df_raw.head())
else:
    print("El DataFrame está vacío porque no se pudo cargar el archivo.")
```

Primeras 5 filas del conjunto de datos:

	Price	Property Type	Size (m2)	Number of Rooms	Number of Bathrooms	\
0	6000.0	countryHouse	32.0	3		1
1	6700.0	flat	28.0	1		1
2	6700.0	flat	28.0	1		1
3	6800.0	flat	26.0	1		1
4	8000.0	flat	84.0	3		1

	Latitude	Longitude	Location	Province
0	36.463557	-5.723404	calle Sainz Andino	2Cádiz
1	36.123324	-5.449942	calle los Barreros	2Cádiz
2	36.124285	-5.448316	calle los Barreros	2Cádiz
3	36.689694	-6.144477	calle Nueva	2Cádiz
4	36.163900	-5.356035	calle Balmes	2Cádiz

## 1.2 Inspección Inicial de Datos Crudos

Análisis preliminar del conjunto de datos: información general, estadísticas descriptivas, dimensiones y tipos de datos. Identificación de posibles problemas como inconsistencias en los tipos de datos o valores que requieren transformación.

```
[4]: # Información general del DataFrame: tipos de datos y valores no nulos
if not df_raw.empty:
    print("Información general del DataFrame:")
    df_raw.info()
else:
    print("El DataFrame está vacío.")
```

Información general del DataFrame:

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

RangeIndex: 369240 entries, 0 to 369239

Data columns (total 9 columns):

#	Column	Non-Null Count	Dtype
0	Price	369240 non-null	float64

```

1  Property Type      369240 non-null  object
2  Size (m2)         369240 non-null  float64
3  Number of Rooms   369240 non-null  int64
4  Number of Bathrooms 369240 non-null  int64
5  Latitude          369240 non-null  float64
6  Longitude         369240 non-null  float64
7  Location          369229 non-null  object
8  Province          369240 non-null  object
dtypes: float64(4), int64(2), object(3)
memory usage: 25.4+ MB

```

```

[5]: # Estadísticas descriptivas
if not df_raw.empty:
    # Estadísticas descriptivas de variables numéricas
    print("\nEstadísticas descriptivas de variables numéricas:")
    display(df_raw.describe())

    # Estadísticas descriptivas de variables categóricas
    print("\nEstadísticas descriptivas de variables categóricas:")
    display(df_raw.describe(include=['object']))
else:
    print("El DataFrame está vacío.")

```

Estadísticas descriptivas de variables numéricas:

	Price	Size (m2)	Number of Rooms	Number of Bathrooms \
count	3.692400e+05	369240.000000	369240.000000	369240.000000
mean	9.339253e+05	421.734140	3.786957	2.446238
std	2.291824e+06	8202.197116	2.222742	2.014015
min	2.500000e+03	9.000000	0.000000	0.000000
25%	6.000000e+04	90.000000	3.000000	1.000000
50%	1.000000e+05	141.000000	3.000000	2.000000
75%	5.945000e+05	288.000000	5.000000	3.000000
max	5.000000e+07	999999.000000	79.000000	46.000000

	Latitude	Longitude
count	369240.000000	369240.000000
mean	37.136370	-4.872701
std	0.535176	1.231564
min	36.010355	-7.493002
25%	36.717090	-5.957531
50%	37.201134	-4.999828
75%	37.412948	-3.780349
max	38.596052	-1.743577

Estadísticas descriptivas de variables categóricas:

Property Type Location Province

count	369240	369229	369240
unique	6	18069	13
top	chalet	Centro	3Jaén
freq	192442	2152	40000

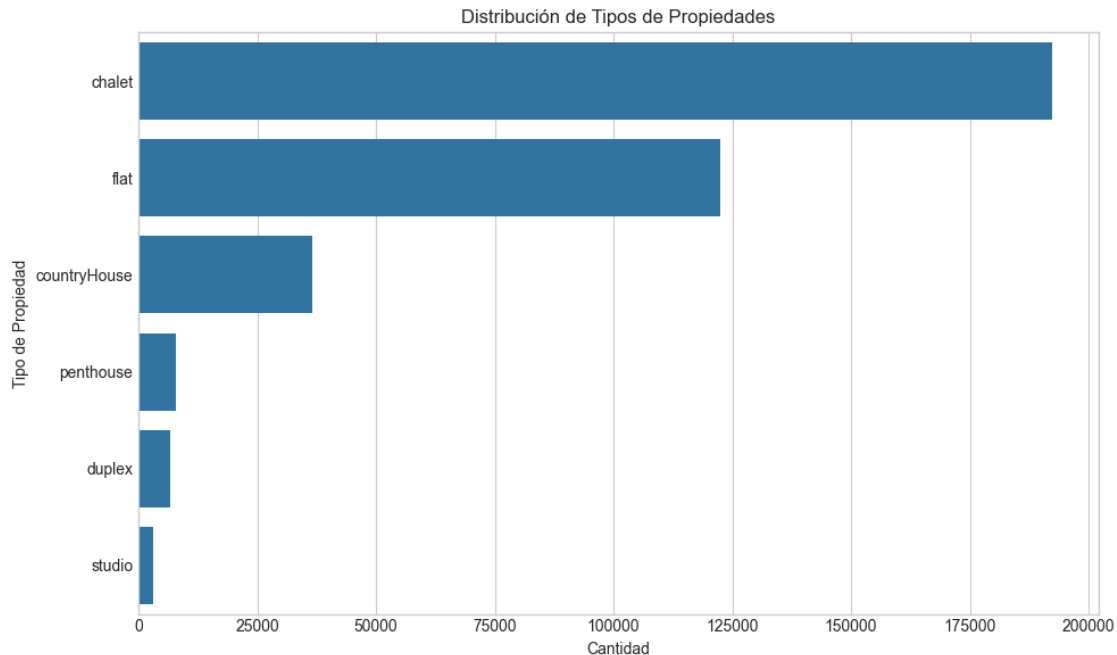
```
[6]: # Verificar los valores únicos en la columna de tipo de propiedad
if not df_raw.empty and 'Property Type' in df_raw.columns:
    property_types = df_raw['Property Type'].value_counts()
    print("Tipos de propiedades y su frecuencia:")
    display(property_types)

    # Visualizar la distribución de tipos de propiedades
    plt.figure(figsize=(10, 6))
    sns.countplot(y=df_raw['Property Type'], order=property_types.index)
    plt.title('Distribución de Tipos de Propiedades')
    plt.xlabel('Cantidad')
    plt.ylabel('Tipo de Propiedad')
    plt.tight_layout()
    plt.show()
else:
    print("El DataFrame está vacío o no contiene la columna 'Property Type'.")
```

Tipos de propiedades y su frecuencia:

Property Type	
chalet	192442
flat	122463
countryHouse	36647
penthouse	7764
duplex	6754
studio	3170

Name: count, dtype: int64



### 1.3 Manejo de Valores Nulos y Duplicados

Detección y tratamiento de valores nulos en el dataset mediante técnicas como imputación o eliminación según corresponda. Identificación y eliminación de registros duplicados para garantizar la calidad de los datos.

```
[7]: # Verificar valores nulos en cada columna
if not df_raw.empty:
    missing_values = df_raw.isnull().sum()
    missing_percentage = (df_raw.isnull().sum() / len(df_raw)) * 100

    missing_info = pd.DataFrame({
        'Valores Nulos': missing_values,
        'Porcentaje (%)': missing_percentage.round(2)
    })

    print("Análisis de valores nulos por columna:")
    display(missing_info[missing_info['Valores Nulos'] > 0])

    if missing_info['Valores Nulos'].sum() == 0:
        print("No se encontraron valores nulos en el dataset.")
    else:
        print("El DataFrame está vacío.")
```

Análisis de valores nulos por columna:

Valores Nulos	Porcentaje (%)
---------------	----------------

Location 11 0.0

```
[8]: # Verificar registros duplicados
if not df_raw.empty:
    duplicates = df_raw.duplicated().sum()
    print(f"Número de registros duplicados: {duplicates}")

    if duplicates > 0:
        # Eliminar duplicados y crear una copia limpia
        df_clean = df_raw.drop_duplicates().reset_index(drop=True)
        print(f"Se eliminaron {duplicates} registros duplicados.")
        print(f"Dimensiones originales: {df_raw.shape}")
        print(f"Dimensiones después de eliminar duplicados: {df_clean.shape}")
    else:
        df_clean = df_raw.copy()
        print("No se encontraron registros duplicados.")
else:
    print("El DataFrame está vacío.")
    df_clean = pd.DataFrame() # DataFrame vacío para evitar errores
```

Número de registros duplicados: 325537

Se eliminaron 325537 registros duplicados.

Dimensiones originales: (369240, 9)

Dimensiones después de eliminar duplicados: (43703, 9)

## 1.4 Limpieza y Transformación de Columnas

Normalización de los nombres de columnas, conversión de tipos de datos, estandarización de unidades y formatos, traducción de columnas al español, y otras transformaciones necesarias para preparar el dataset para análisis posteriores.

```
[9]: # Crear una copia del DataFrame para las transformaciones
if not df_raw.empty:
    # Si ya tenemos df_clean de la celda anterior, usamos ese, si no, creamos
    ↪ una copia
    if 'df_clean' not in locals():
        df_clean = df_raw.copy()

    print("Tipos de datos antes de la transformación:")
    print(df_clean.dtypes)
else:
    print("El DataFrame está vacío.")
```

Tipos de datos antes de la transformación:

Price	float64
Property Type	object
Size (m2)	float64
Number of Rooms	int64
Number of Bathrooms	int64

```
Latitude          float64
Longitude          float64
Location           object
Province           object
dtype: object
```

```
[10]: # Renombrar columnas al español
if not df_clean.empty:
    # Diccionario de mapeo de nombres en inglés a español
    column_mapping = {
        'Price': 'precio',
        'Property Type': 'tipo_propiedad',
        'Size (m2)': 'superficie',
        'Number of Rooms': 'habitaciones',
        'Number of Bathrooms': 'baños',
        'Latitude': 'latitud',
        'Longitude': 'longitud',
        'Location': 'ubicacion'
    }

    # Renombrar columnas
    df_clean = df_clean.rename(columns=column_mapping)

    print("Columnas renombradas al español:")
    print(df_clean.columns.tolist())
else:
    print("El DataFrame está vacío.")
```

Columnas renombradas al español:

```
['precio', 'tipo_propiedad', 'superficie', 'habitaciones', 'baños', 'latitud',
'longitud', 'ubicacion', 'Province']
```

```
[11]: # Traducir y estandarizar valores en la columna tipo_propiedad
if not df_clean.empty and 'tipo_propiedad' in df_clean.columns:
    # Mapeo de tipos de propiedad en inglés a español
    property_type_mapping = {
        'flat': 'piso',
        'chalet': 'chalet',
        'countryHouse': 'casa_rural',
        'duplex': 'duplex',
        'studio': 'estudio',
        'penthouse': 'atico'
    }

    # Reemplazar valores
    df_clean['tipo_propiedad'] = df_clean['tipo_propiedad'].
    ↪map(property_type_mapping).fillna(df_clean['tipo_propiedad'])
```

```

    print("Valores únicos en la columna tipo_propiedad después de la traducción:
↪")
    print(df_clean['tipo_propiedad'].value_counts())
else:
    print("El DataFrame está vacío o no contiene la columna 'tipo_propiedad'.")

```

Valores únicos en la columna tipo\_propiedad después de la traducción:

```

tipo_propiedad
chalet          23072
piso            14253
casa_rural      4347
atico           868
duplex          779
estudio         384
Name: count, dtype: int64

```

```

[12]: # Verificar y tratar valores extremos o inconsistentes en la columna superficie
if not df_clean.empty and 'superficie' in df_clean.columns:
    # Resumen estadístico de la superficie antes de la limpieza
    print("Resumen estadístico de superficie antes de la limpieza:")
    print(df_clean['superficie'].describe())

    # Identificar valores extremos (por ejemplo, propiedades con más de 10,000
↪m²)
    large_properties = df_clean[df_clean['superficie'] > 10000]
    if not large_properties.empty:
        print(f"\nPropiedades con superficie extremadamente grande (>10,000 m²):
↪ {len(large_properties)}")
        display(large_properties[['precio', 'tipo_propiedad', 'superficie',
↪ 'habitaciones', 'ubicacion']])

        # Opción para filtrar o corregir estos valores
        # Aquí podríamos decidir eliminarlos o ajustarlos según el caso
        df_clean = df_clean[df_clean['superficie'] <= 10000]
        print(f"Se filtraron propiedades con superficie > 10,000 m²")

    # Verificar valores negativos o cero en superficie
    invalid_surface = df_clean[(df_clean['superficie'] <= 0)]
    if not invalid_surface.empty:
        print(f"\nPropiedades con superficie inválida (<=0 m²):
↪ {len(invalid_surface)}")
        df_clean = df_clean[df_clean['superficie'] > 0]
        print(f"Se eliminaron propiedades con superficie inválida")

    print("\nResumen estadístico de superficie después de la limpieza:")
    print(df_clean['superficie'].describe())
else:

```



```
print("El DataFrame está vacío o no contiene la columna 'superficie'.")
```

Resumen estadístico de superficie antes de la limpieza:

```
count    43703.000000
mean      460.170798
std       9201.069705
min        9.000000
25%       89.000000
50%      141.000000
75%      300.000000
max     999999.000000
```

Name: superficie, dtype: float64

Propiedades con superficie extremadamente grande (>10,000 m<sup>2</sup>): 69

	precio	tipo_propiedad	superficie	habitaciones	\
565	65000.0	chalet	110000.0	4	
2295	2800000.0	casa_rural	841340.0	1	
2455	2250000.0	chalet	74405.0	5	
2665	1550000.0	chalet	14046.0	10	
2819	1300000.0	casa_rural	13350.0	1	
...	...	...	...	...	
36579	85000.0	chalet	10300.0	1	
40985	75000.0	chalet	133846.0	4	
41802	1500000.0	casa_rural	95000.0	1	
42571	586300.0	casa_rural	311883.0	0	
43532	395000.0	casa_rural	90764.0	0	

	ubicacion
565	calle Maestro Amado, 120
2295	San Martín del Tesorillo
2455	calle Genal
2665	La Granja - La Colina - Los Pastores
2819	Paterna de Rivera
...	...
36579	Casabermeja
40985	calle Carmona, 32
41802	via Verde de la Sierra
42571	VIÑA ALALUZ
43532	camino VIEJO, 1 ,5

[69 rows x 5 columns]

Se filtraron propiedades con superficie > 10,000 m<sup>2</sup>

Resumen estadístico de superficie después de la limpieza:

```
count    43634.000000
mean      268.718440
std       380.934471
```

```

min          9.000000
25%         89.000000
50%        141.000000
75%        299.000000
max       10000.000000
Name: superficie, dtype: float64

```

```

[13]: # Verificar y tratar valores extremos o inconsistentes en la columna precio
if not df_clean.empty and 'precio' in df_clean.columns:
    # Resumen estadístico del precio antes de la limpieza
    print("Resumen estadístico de precio antes de la limpieza:")
    print(df_clean['precio'].describe())

    # Identificar valores extremadamente bajos o altos
    low_price = df_clean[df_clean['precio'] < 1000]
    if not low_price.empty:
        print(f"\nPropiedades con precio extremadamente bajo (<1,000 €):_")
        ↪{len(low_price)}")
        display(low_price[['precio', 'tipo_propiedad', 'superficie',_
        ↪'habitaciones', 'ubicacion']].head())

    high_price = df_clean[df_clean['precio'] > 1000000]
    if not high_price.empty:
        print(f"\nPropiedades con precio extremadamente alto (>1,000,000 €):_")
        ↪{len(high_price)}")
        display(high_price[['precio', 'tipo_propiedad', 'superficie',_
        ↪'habitaciones', 'ubicacion']].head())

    # Opción: filtrar valores extremos
    # df_clean = df_clean[(df_clean['precio'] >= 1000) & (df_clean['precio'] <=_
    ↪1000000)]

    # Otra opción: considerar el rango intercuartílico para detectar outliers
    Q1 = df_clean['precio'].quantile(0.25)
    Q3 = df_clean['precio'].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    print(f"\nLímite inferior para outliers (Q1 - 1.5*IQR): {lower_bound}")
    print(f"\nLímite superior para outliers (Q3 + 1.5*IQR): {upper_bound}")

    outliers = df_clean[(df_clean['precio'] < lower_bound) |_
    ↪(df_clean['precio'] > upper_bound)]
    print(f"\nNúmero de outliers según método IQR: {len(outliers)}")

```

```

    # En este caso, podríamos decidir conservar los outliers para el análisis_
    ↪ exploratorio
    # y tomar decisiones sobre su tratamiento más adelante
    print("\nNo se eliminarán outliers de precio en esta etapa, se considerarán_
    ↪ en el análisis exploratorio.")
else:
    print("El DataFrame está vacío o no contiene la columna 'precio'.")

```

Resumen estadístico de precio antes de la limpieza:

```

count    4.363400e+04
mean     1.002781e+06
std      2.379786e+06
min      2.500000e+03
25%      6.000000e+04
50%      9.900000e+04
75%      6.000000e+05
max      5.000000e+07
Name: precio, dtype: float64

```

Propiedades con precio extremadamente alto (>1,000,000 €): 6902

	precio	tipo_propiedad	superficie	habitaciones	\
1998	18000000.0	chalet	3850.0	18	
1999	14900000.0	chalet	2438.0	7	
2000	14900000.0	chalet	2348.0	7	
2001	14900000.0	chalet	2348.0	7	
2002	14900000.0	chalet	2348.0	7	

	ubicacion
1998	Barrio Zona F
1999	Lugar Sotogrande
2000	calle Maravillas, 823 -828
2001	calle Maravillas
2002	Barrio La Reserva

Límite inferior para outliers (Q1 - 1.5\*IQR): -750000.0

Límite superior para outliers (Q3 + 1.5\*IQR): 1410000.0

Número de outliers según método IQR: 5925

No se eliminarán outliers de precio en esta etapa, se considerarán en el análisis exploratorio.

```

[14]: # Crear variables derivadas útiles
      if not df_clean.empty:
          # Precio por metro cuadrado (€/m²)
          if 'precio' in df_clean.columns and 'superficie' in df_clean.columns:

```

```

df_clean['precio_m2'] = df_clean['precio'] / df_clean['superficie']
print("Se creó la columna precio_m2 (precio por metro cuadrado)")

# Resumen estadístico del precio por metro cuadrado
print("\nEstadísticas de precio por metro cuadrado (€/m²):")
print(df_clean['precio_m2'].describe())

# Eliminar valores extremos o inconsistentes en precio_m2
extreme_price_m2 = df_clean[(df_clean['precio_m2'] > 10000) |
↪(df_clean['precio_m2'] < 10)]
if not extreme_price_m2.empty:
    print(f"\nPropiedades con precio por m² extremo (<10 € o >10,000 €):
↪ {len(extreme_price_m2)}")
    df_clean = df_clean[(df_clean['precio_m2'] >= 10) &
↪(df_clean['precio_m2'] <= 10000)]
    print(f"Se filtraron propiedades con precio por m² extremo")

# Ratio habitaciones/superficie
if 'habitaciones' in df_clean.columns and 'superficie' in df_clean.columns:
    df_clean['densidad_habitaciones'] = df_clean['habitaciones'] /
↪df_clean['superficie']
    print("\nSe creó la columna densidad_habitaciones (ratio habitaciones/
↪superficie)")

print("\nColumnas finales del DataFrame:")
print(df_clean.columns.tolist())

# Mostrar las primeras filas del DataFrame después de las transformaciones
print("\nPrimeras filas del DataFrame después de las transformaciones:")
display(df_clean.head())
else:
    print("El DataFrame está vacío.")

```

Se creó la columna precio\_m2 (precio por metro cuadrado)

Estadísticas de precio por metro cuadrado (€/m²):

```

count    43634.000000
mean      2347.979991
std       3724.745967
min        2.572016
25%       566.666667
50%      1057.647059
75%      2551.544409
max      81893.004115
Name: precio_m2, dtype: float64

```

Propiedades con precio por m² extremo (<10 € o >10,000 €): 1740

Se filtraron propiedades con precio por m<sup>2</sup> extremo

Se creó la columna densidad\_habitaciones (ratio habitaciones/superficie)

Columnas finales del DataFrame:

```
['precio', 'tipo_propiedad', 'superficie', 'habitaciones', 'baños', 'latitud',  
'longitud', 'ubicacion', 'Province', 'precio_m2', 'densidad_habitaciones']
```

Primeras filas del DataFrame después de las transformaciones:

	precio	tipo_propiedad	superficie	habitaciones	baños	latitud	\
0	6000.0	casa_rural	32.0	3	1	36.463557	
1	6700.0	piso	28.0	1	1	36.123324	
2	6700.0	piso	28.0	1	1	36.124285	
3	6800.0	piso	26.0	1	1	36.689694	
4	8000.0	piso	84.0	3	1	36.163900	

	longitud	ubicacion	Province	precio_m2	densidad_habitaciones
0	-5.723404	calle Sainz Andino	2Cádiz	187.500000	0.093750
1	-5.449942	calle los Barreros	2Cádiz	239.285714	0.035714
2	-5.448316	calle los Barreros	2Cádiz	239.285714	0.035714
3	-6.144477	calle Nueva	2Cádiz	261.538462	0.038462
4	-5.356035	calle Balmes	2Cádiz	95.238095	0.035714

## 1.5 Guardar Datos Limpios en CSV

Almacenamiento del DataFrame procesado y limpio en un nuevo archivo CSV para su uso en análisis posteriores y modelado.

```
[15]: # Guardar el DataFrame limpio en un archivo CSV
if not df_clean.empty:
    # Generar nombre de archivo con timestamp para evitar sobreescritura
    timestamp = datetime.now().strftime("%Y%m%d")
    clean_file_path = os.path.join(clean_data_dir,
    ↪f'andalucia_clean_{timestamp}.csv')

    # Guardar el archivo
    df_clean.to_csv(clean_file_path, index=False)
    print(f"Dataset limpio guardado exitosamente en: {clean_file_path}")
    print(f"Dimensiones del dataset limpio: {df_clean.shape}")

    # También guardar una versión sin timestamp para facilitar referencias en
    ↪otros notebooks
    standard_clean_path = os.path.join(clean_data_dir, 'andalucia_clean.csv')
    df_clean.to_csv(standard_clean_path, index=False)
    print(f"Dataset limpio también guardado como: {standard_clean_path}")
else:
    print("El DataFrame está vacío. No se puede guardar.")
```

Dataset limpio guardado exitosamente en:  
../data/clean/andalucia\_clean\_20250528.csv  
Dimensiones del dataset limpio: (41894, 11)  
Dataset limpio también guardado como: ../data/clean/andalucia\_clean.csv