

Impacto de la computación en las industrias convencionales

El proposito de esta investigacion es evaluar como los avances en la computacion en la ultima decada han impactado en los negocios convencionales.

Como regla analizaremos industrias que existian mucho antes de la computacion.

Dependencias

```
In [ ]: # Standard packages
import pandas as pd
import numpy as np
import re

# Installed packages
from IPython.display import display
from matplotlib import pyplot as plt
%pip install seaborn
import seaborn as sns
from scipy import stats
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split

# Local packages
# NOTE:
# Evitaremos tener que usar modulos locales externos a
# este notebook de Jupiter para facilitar el uso en la
# plataforma Google Colab.
```

```

Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: seaborn in /home/angrygingy/.local/lib/python3.10/site-packages (0.12.2)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in /home/angrygingy/.local/lib/python3.10/site-packages (from seaborn) (1.23.5)
Requirement already satisfied: pandas>=0.25 in /home/angrygingy/.local/lib/python3.10/site-packages (from seaborn) (1.5.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in /home/angrygingy/.local/lib/python3.10/site-packages (from seaborn) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /home/angrygingy/.local/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.0.7)
Requirement already satisfied: cycler>=0.10 in /home/angrygingy/.local/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /home/angrygingy/.local/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (4.39.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /home/angrygingy/.local/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /home/angrygingy/.local/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (23.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/lib/python3/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (9.0.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/lib/python3/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (2.4.7)
Requirement already satisfied: python-dateutil>=2.7 in /home/angrygingy/.local/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/lib/python3/dist-packages (from pandas>=0.25->seaborn) (2022.1)
Requirement already satisfied: six>=1.5 in /home/angrygingy/.local/lib/python3.10/site-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.1->seaborn) (1.12.0)

```

[notice] A new release of pip is available: 23.1 -> 23.1.2

[notice] To update, run: `pip install --upgrade pip`

Note: you may need to restart the kernel to use updated packages.

Parametros constantes que utilizaremos

```

In [ ]: # Constantes
        COLUMNA_FECHA = 'Año'
        FECHA_MIN = 2001
        FECHA_MAX = 2020
        DATASETS_FOLDER = './content'
        # Utilizaremos esta para indicar si queremos
        # guardar el postprocesado de un dataset,
        # util si el dataset original es muy pesado.
        GUARDAR_DATOS_PROCESADOS = False

        # Global state
        poblacion = None # Usado para normalizar los datos

```

Procesando datos relacionados a la industria de la computacion

Para analizar los avances en la computacion, uniremos varios datasets que representan avances en multiples areas de la computacion, en 1 solo dataframe con los datos mas revelantes agrupado por fecha, mas adelante unificaremos este dataframe resultante con los de otras industrias (unificado en base a la fecha) para analizar si hay correlaciones presentes y si podemos hacer una proyeccion aplicando regresiones.

Una manera muy buena de medir los avances en la computacion es en base a la cantidad de transistores, esta fue incrementandose a lo largo del tiempo siguiendo la Ley de Moore. En este analisis repetitivamente se limpiaran los datos para quedarnos solamente con estos parametros (fecha + cantidad de transistores).

Los datasets utilizamos fueron obtenidos de [wikipedia - cantidad de transistores](https://wikitable2csv.ggor.de) usando wikitable2csv.ggor.de para extraer las tablas de datos.

```
In [ ]: # En esta lista vamos a ir acumulando los diferentes dataframes
# de diferentes areas de la computacion, mas adelante se mergearan
# en uno resultante.
computacion_dfs = []
```

```
In [ ]: def asegurar_tipo_de_dato_fecha(df: pd.DataFrame, columna_fecha:str=COLUMNA_FECHA) -> pd.DataFrame:
    """Aseguramos que el tipo de dato de la fecha es un entero, no de tipo string"""
    df[columna_fecha] = df[columna_fecha].astype(int)
    return df

def mantener_columnas(df:pd.DataFrame, columnas:list) -> pd.DataFrame:
    """Mantiene solo las COLUMNAS especificadas en un DF"""
    return df[columnas]

def fecha_a_anio(df:pd.DataFrame, columna_fecha:str) -> pd.DataFrame:
    """Convierte COLUMNA_FECHA a un entero ANIO y cambia el nombre de la columna a ANIO
    # Convert to year
    fecha_regex = r'\d{4}'
    df[columna_fecha] = df[columna_fecha].astype(str)
    df = df[df[columna_fecha].str.contains(fecha_regex)]
    df[columna_fecha] = df[columna_fecha].apply(lambda x: re.findall(fecha_regex, x)[0])
    # Rename
    if columna_fecha != COLUMNA_FECHA:
        df = df.rename(columns={columna_fecha: COLUMNA_FECHA})
    return df

def remover_commas(df:pd.DataFrame) -> pd.DataFrame:
    """Remueve commas de las columnas"""
    return df.apply(lambda x: x.str.replace(',', ''))

def convertir_a_flotante(df:pd.DataFrame, columnas:list) -> pd.DataFrame:
    """Convierte las columnas especificadas a flotante"""
    for col in columnas:
        df[col] = pd.to_numeric(df[col], errors='coerce')
    return df

def cortar_agregar_anios(df: pd.DataFrame,
```

```

        columna_fecha:str=COLUMNA_FECHA,
        min_anio:int=FECHA_MIN,
        max_anio:int=FECHA_MAX
    ) -> pd.DataFrame:
    """Corta un dataframe DF en los años especificados, agregando los años faltantes
    df = asegurar_tipo_de_dato_fecha(df, columna_fecha)
    # Add missing years
    for year in range(min_anio, max_anio):
        if year not in df[columna_fecha].values:
            df = pd.concat([df, pd.DataFrame.from_records([{'columna_fecha': year}])] * 100)
    df = df.sort_values(by=[columna_fecha]).reset_index(drop=True)
    # Cut
    df = df[(df[columna_fecha] >= min_anio) & (df[columna_fecha] <= max_anio)]
    return df

def mantener_solo_valor_mas_alto(
    df: pd.DataFrame,
    columna_valor:str,
    columna_fecha:str=COLUMNA_FECHA
) -> pd.DataFrame:
    """Mantiene solo el valor mas alto por fecha, util cuando queremos ver el
    avance de una tecnologia, solo nos interesa el valor mas alto"""
    df = df.sort_values(by=[columna_fecha, columna_valor], ascending=False)
    df = df.drop_duplicates(subset=[columna_fecha], keep='first')
    # Back to the original order
    df = df.sort_values(by=[columna_fecha]).reset_index(drop=True)
    return df

def columna_fecha_a_fila(df:pd.DataFrame, pais:str, nueva_columna:str) -> pd.DataFrame:
    """Convierte columnas fecha a fila, filtrando por pais, esto es util para
    analizar la evolucion de una tecnologia por pais
    # Filter rows by country
    df = df[df['Country Name'] == pais]
    # Remove columns that are not a year
    df = df[df.columns[df.columns.str.contains(r'\d{4}')] ]
    # Convert columns to rows
    df = df.melt(id_vars=[], var_name=COLUMNA_FECHA, value_name=nueva_columna)
    return df

def normalizar_con_la_poblacion(df:pd.DataFrame, column:str) -> pd.DataFrame:
    """Normalizamos una COLUMNA de un dataframe DF con la poblacion"""
    return df ## TODO: fix this
    assert poblacion is not None, 'El dataframe Poblacion no fue cargado'
    df[column] = np.divide(df[column], poblacion['US Population']) * 100
    return df

def fusionar_por_fecha(dataframes:list) -> pd.DataFrame:
    """Fusiona los DATAFRAMES por fecha,
    NOTE: todos los dataframes deben tener la misma cantidad de filas"""
    resultado = pd.DataFrame()
    resultado[COLUMNA_FECHA] = dataframes[0][COLUMNA_FECHA].unique()
    resultado = asegurar_tipo_de_dato_fecha(resultado)
    # Merge dataframes
    for df in dataframes:
        df = asegurar_tipo_de_dato_fecha(df)
        resultado = pd.merge(resultado, df, on=COLUMNA_FECHA, how='outer')
    resultado = cortar_agregar_anios(resultado)
    return resultado

```

- [Memoria Flash](#)

Para analizar los avances en la memoria Flash, utilizaremos el dataset relacionado a esta tecnología que fue obtenido de la anteriormente mencionada [wikipedia - cantidad de transistores](#).

```
In [ ]: flash = pd.read_csv(f'{DATASETS_FOLDER}/flash.csv')
#
print('# Muestra del dataset original')
display(flash.tail())
#
flash = mantener_columnas(flash, ['FGMOS transistor count', 'Date of intr
flash = fecha_a_anio(flash, 'Date of introduction')
flash = remover_commas(flash)
flash = convertir_a_flotante(flash, ['FGMOS transistor count'])
flash = cortar_agregar_anios(flash)
flash = mantener_solo_valor_mas_alto(flash, 'FGMOS transistor count')
flash = flash.fillna(method='ffill')
computacion_dfs.append(flash)
#
print('# Muestra del dataset procesado y limpio, solamente nos quedamos c
display(flash.tail())
display(flash.describe())
flash.plot(
    x=COLUMNA_FECHA,
    y='FGMOS transistor count',
    title='cantidad de transistores FGMOS a lo largo del tiempo',
    figsize=(4, 2),
)
plt.show()
```

Muestra del dataset original

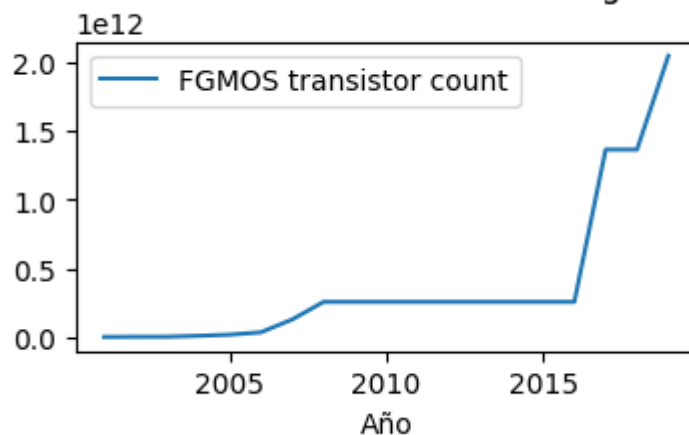
	Chip name	Capacity (bits)	Flash type	FGMOS transistor count	Date of introduction	Manufacturer(s)	Proce
19	KLMCG8GE4A	512 Gb	Stacked 2-bit NAND	256,000,000,000	2011	Samsung	
20	KLUFG8R1EM	4 Tb	Stacked 3-bit V-NAND	1,365,333,333,504	2017	Samsung	
21	eUFS (1 TB)	8 Tb	Stacked 4-bit V-NAND	2,048,000,000,000	2019	Samsung	
22	?	1 Tb	232L TLC NAND die	333,333,333,333	2022	Micron	
23	?	16 Tb	232L package	5,333,333,333,333	2022	Micron	

Muestra del dataset procesado y limpio, solamente nos quedamos con la fecha y cantidad de transistores

	FGMOS transistor count	Año
14	2.560000e+11	2015
15	2.560000e+11	2016
16	1.365333e+12	2017
17	1.365333e+12	2018
18	2.048000e+12	2019

	FGMOS transistor count	Año
count	1.900000e+01	19.000000
mean	3.829278e+11	2010.000000
std	5.649140e+11	5.627314
min	5.368709e+08	2001.000000
25%	2.576980e+10	2005.500000
50%	2.560000e+11	2010.000000
75%	2.560000e+11	2014.500000
max	2.048000e+12	2019.000000

cantidad de transistores FGMOS a lo largo del tiempo



- [FPGA \(matriz de puertas lógicas programable en campo\)](#)

Al igual que la memoria Flash, los datos relacionados al FPGA fueron obtenidos del mismo sitio [wikipedia - cantidad de transistores](#).

A continuacion realizaremos el mismo tratamiento que con la memoria Flash, solamente nos quedaremos con la fecha y la cantidad de transistores.

```
In [ ]: fpga = pd.read_csv(f'{DATASETS_FOLDER}/fpga.csv')
#
print('# Muestra del dataset original')
display(fpga.tail())
#
fpga = mantener_columnas(fpga, ['Transistor count', 'Date of introduction'])
fpga = fecha_a_anio(fpga, 'Date of introduction')
fpga = remover_commas(fpga)
fpga = convertir_a_flotante(fpga, ['Transistor count'])
```

```
fpga = cortar_agregar_anios(fpga)
fpga = fpga.rename(columns={'Transistor count': 'FPGA transistor count'})
fpga = mantener_solo_valor_mas_alto(fpga, 'FPGA transistor count')
fpga = fpga.fillna(method='ffill')
computacion_dfs.append(fpga)
#
print('# Muestra del dataset procesado y limpio')
display(fpga.tail())
display(fpga.describe())
fpga.plot(
    x=COLUMNA_FECHA,
    y='FPGA transistor count',
    title='Cantidad de transistores FPGA a lo largo del tiempo',
    figsize=(4, 2),
)
plt.show()
```

Muestra del dataset original

	FPGA	Transistor count	Date of introduction	Designer	Manufacturer	Process	Area	Tr
11	Virtex-Ultrasc	20,000,000,000	Q1 2015	Xilinx	TSMC	20 nm	NaN	
12	Virtex-Ultrasc+	35,000,000,000	2020	Xilinx	TSMC	16 nm	900 mm2	38
13	Versal VC1902	37,000,000,000	2H 2019	Xilinx	TSMC	7 nm	NaN	
14	Stratix 10 GX 10M	43,300,000,000	Q4 2019	Intel	Intel	14 nm	1,400 mm2	30
15	Versal VP1802	92,000,000,000	2021 ?	Xilinx	TSMC	7 nm	NaN	

Muestra del dataset procesado y limpio

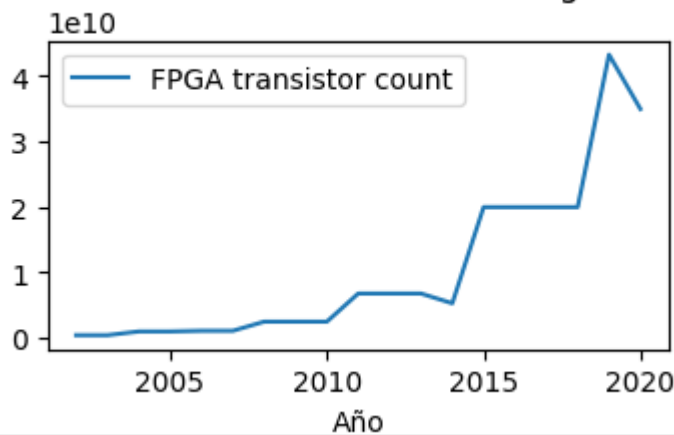
```
/tmp/ipykernel_72182/3403905049.py:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df[columna_fecha] = df[columna_fecha].apply(lambda x: re.findall(fecha
_regex, x)[0])
```

	FPGA transistor count	Año
15	2.000000e+10	2016
16	2.000000e+10	2017
17	2.000000e+10	2018
18	4.330000e+10	2019
19	3.500000e+10	2020

	FPGA transistor count	Año
count	1.900000e+01	20.00000
mean	1.034526e+10	2010.50000
std	1.259800e+10	5.91608
min	4.300000e+08	2001.00000
25%	1.100000e+09	2005.75000
50%	5.300000e+09	2010.50000
75%	2.000000e+10	2015.25000
max	4.330000e+10	2020.00000

Cantidad de transistores FPGA a lo largo del tiempo



- GPU (unidad de procesamiento gráfico)

Los datos fueron obtenidos del mismo sitio que los anteriores [wikipedia - cantidad de transistores](#). A los datos de la GPU le aplicaremos el mismo tratamiento que a los anteriores, solamente nos quedaremos con la fecha y la cantidad de transistores.

```
In [ ]: gpus = pd.read_csv(f'{DATASETS_FOLDER}/gpus.csv')
#
print('# Muestra del dataset original')
display(gpus.tail())
#
gpus = mantener_columnas(gpus, ['Transistor count', 'Year'])
gpus = fecha_a_anio(gpus, 'Year')
gpus = remover_commas(gpus)
gpus = convertir_a_flotante(gpus, ['Transistor count'])
gpus = cortar_agregar_anios(gpus)
gpus = gpus.rename(columns={'Transistor count': 'GPU transistor count'})
gpus = mantener_solo_valor_mas_alto(gpus, 'GPU transistor count')
gpus = gpus.fillna(method='ffill')
computacion_dfs.append(gpus)
#
print('# Muestra del dataset procesado y limpio')
display(gpus.tail())
display(gpus.describe())
gpus.plot(
    x=COLUMNA_FECHA,
    y='GPU transistor count',
```



```
title='cantidad de transistores en GPU a lo largo del tiempo',
figsize=(4, 2),
)
plt.show()
```

Muestra del dataset original

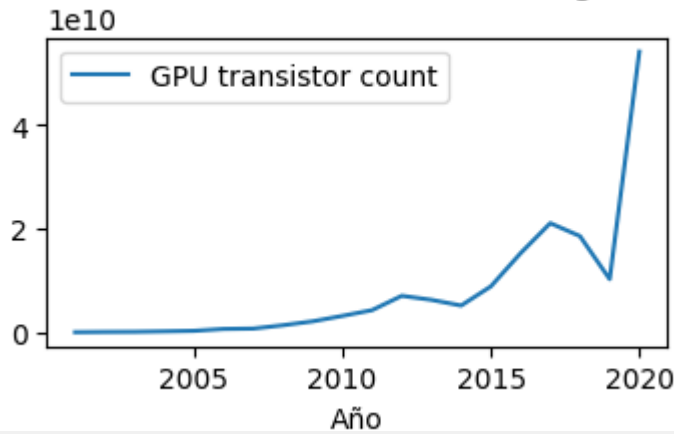
	Processor	Transistor count	Year	Designer(s)	Fab(s)	Process	Area	Tri
158	AD102 Ada Lovelace	76,300,000,000	2022	Nvidia	TSMC	4 nm	608.4 mm²	
159	AD103 Ada Lovelace	45,900,000,000	2022	Nvidia	TSMC	4 nm	378.6 mm²	
160	AD104 Ada Lovelace	35,800,000,000	2022	Nvidia	TSMC	4 nm	294.5 mm²	
161	Navi 31 RDNA3	58,000,000,000	2022	AMD	TSMC	5 nm (GCD) 6 nm (MCD)	531 mm² (MCM) 306 mm² (GCD) 6×37.5 mm² (MCD)	10 13
162	Navi 33 RDNA3	13,300,000,000	2023	AMD	TSMC	6 nm	204 mm²	

Muestra del dataset procesado y limpio

	GPU transistor count	Año
15	1.530000e+10	2016
16	2.110000e+10	2017
17	1.860000e+10	2018
18	1.030000e+10	2019
19	5.420000e+10	2020

	GPU transistor count	Año
count	2.000000e+01	20.00000
mean	8.016336e+09	2010.50000
std	1.260009e+10	5.91608
min	6.000000e+07	2001.00000
25%	5.910000e+08	2005.75000
50%	3.756356e+09	2010.50000
75%	9.250000e+09	2015.25000
max	5.420000e+10	2020.00000

cantidad de transistores en GPU a lo largo del tiempo



- [Microprocesador](#)

Los datos fueron obtenidos del mismo sitio que los anteriores [wikipedia - cantidad de transistores](#). A los datos del microprocesador le aplicaremos el mismo tratamiento que a los anteriores, solamente nos quedaremos con la fecha y la cantidad de transistores.

```
In [ ]: microprocesadores = pd.read_csv(f'{DATASETS_FOLDER}/microprocessors.csv')
#
print('# Muestra del dataset original')
display(microprocesadores.tail())
#
microprocesadores = mantener_columnas(microprocesadores, ['Transistor count'])
microprocesadores = fecha_a_anio(microprocesadores, 'Year')
microprocesadores = remover_commas(microprocesadores)
microprocesadores = microprocesadores[microprocesadores['Transistor count'] > 0]
microprocesadores = convertir_a_flotante(microprocesadores, ['Transistor count'])
microprocesadores = microprocesadores.rename(columns={'Transistor count': 'Transistor count'})
microprocesadores = cortar_agregar_anios(microprocesadores)
microprocesadores = mantener_solo_valor_mas_alto(microprocesadores, 'Microprocessor transistor count')
microprocesadores = microprocesadores.fillna(method='ffill')
computacion_dfs.append(microprocesadores)
#
print('# Muestra del dataset procesado y limpio')
display(microprocesadores.tail())
display(microprocesadores.describe())
microprocesadores.plot(
    x=COLUMNA_FECHA,
    y='Microprocessor transistor count',
    title='Cantidad de transistores en el Microprocesador a lo largo del tiempo',
    figsize=(4, 2),
)
plt.show()

# Muestra del dataset original
```

	Processor	Transistor count	Year	Designer	Process\n(nm)	Area (mm2)	
229	AMD EPYC Genoa (4th gen/9004 series) 13-chip m...	90,000,000,000	2022	AMD	5 nm (CCD)\n6 nm (IOD)	1,263.34 mm²\n12×72.225 (CCD)\n396.64 (IOD)	7
230	Sapphire Rapids quad-chip module (up to 60 cor...	44,000,000,000–48,000,000,000	2023	Intel	Intel 7 (10 nm ESF)	1,600 mm2	27 \n3
231	Apple M2 Pro (12-core 64-bit ARM64 SoC, SIMD, ...	40,000,000,000	2023	Apple	5 nm	?	
232	Apple M2 Max (12-core 64-bit ARM64 SoC, SIMD, ...	67,000,000,000	2023	Apple	5 nm	?	
233	Processor	Transistor count	Year	Designer	Process\n(nm)	Area (mm2)	

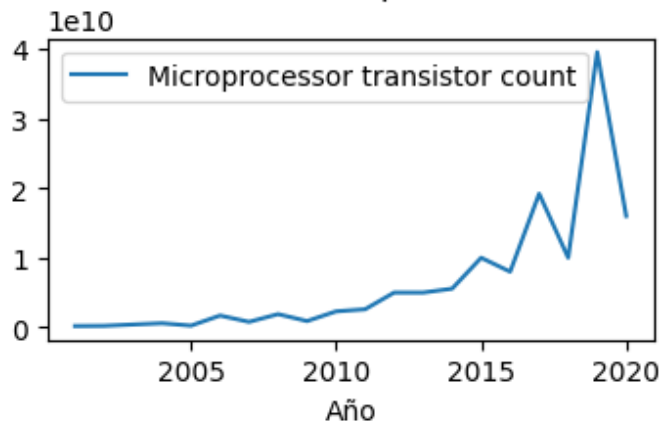
Muestra del dataset procesado y limpio

```
/tmp/ipykernel_72182/3403905049.py:16: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-d  
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
df[columna_fecha] = df[columna_fecha].apply(lambda x: re.findall(fecha  
_regex, x)[0])
```

Microprocessor	transistor count	Año
15	8000000000	2016
16	19200000000	2017
17	10000000000	2018
18	39540000000	2019
19	16000000000	2020

	Microprocessor transistor count	Año
count	2.000000e+01	20.00000
mean	6.507800e+09	2010.50000
std	9.477374e+09	5.91608
min	1.910000e+08	2001.00000
25%	7.397500e+08	2005.75000
50%	2.450000e+09	2010.50000
75%	8.500000e+09	2015.25000
max	3.954000e+10	2020.00000

Cantidad de transistores en el Microprocesador a lo largo del tiempo



- RAM (memoria de acceso aleatorio)

Los datos fueron obtenidos del mismo sitio que los anteriores [wikipedia - cantidad de transistores](#). A los datos de la RAM le aplicaremos el mismo tratamiento que a los anteriores, solamente nos quedaremos con la fecha y la cantidad de transistores.

```
In [ ]: ram = pd.read_csv(f'{DATASETS_FOLDER}/ram.csv')
#
print('# Dataset original')
display(ram.tail())
#
ram = mantener_columnas(ram, ['Date of introduction', 'Transistor count'])
ram = fecha_a_anio(ram, 'Date of introduction')
ram = remover_commas(ram)
ram = convertir_a_flotante(ram, ['Transistor count'])
ram = ram.rename(columns={'Transistor count': 'RAM transistor count'})
ram = cortar_agregar_anios(ram)
ram = mantener_solo_valor_mas_alto(ram, 'RAM transistor count')
ram = ram.fillna(method='ffill')
# MALO: computacion_dfs.append(ram) este dataset tiene un monton Nan valu
#
print('# Dataset limpio')
display(ram.tail())
display(ram.describe())
ram.plot(
    x=COLUMNA_FECHA,
    y='RAM transistor count',
    title='Cantidad de transistores en la RAM a lo largo del tiempo',
```

```
figsize=(4, 2),
)
plt.show()
```

Dataset original

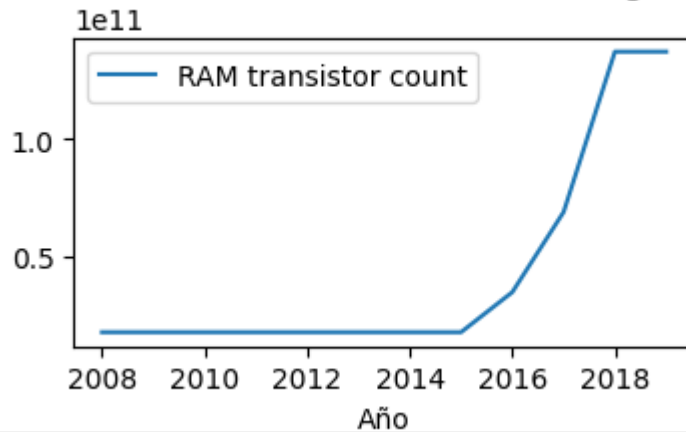
	Chip name	Capacity (bits)	RAM type	Transistor count	Date of introduction	Manufacturer(s)	Process	Area
43	?	16 Gb	SDRAM (DDR3)	17,179,869,184	2008	Samsung	50 nm	?
44	?	32 Gb	SDRAM (HBM2)	34,359,738,368	2016	Samsung	20 nm	?
45	?	64 Gb	SDRAM (HBM2)	68,719,476,736	2017	Samsung	20 nm	?
46	?	128 Gb	SDRAM (DDR4)	137,438,953,472	2018	Samsung	10 nm	?
47	?	?	RRAM (3DSoc)	?	2019	SkyWater Technology	90 nm	?

Dataset limpio

	Año	RAM transistor count
14	2015	1.717987e+10
15	2016	3.435974e+10
16	2017	6.871948e+10
17	2018	1.374390e+11
18	2019	1.374390e+11

	Año	RAM transistor count
count	19.000000	1.200000e+01
mean	2010.000000	4.294967e+10
std	5.627314	4.661933e+10
min	2001.000000	1.717987e+10
25%	2005.500000	1.717987e+10
50%	2010.000000	1.717987e+10
75%	2014.500000	4.294967e+10
max	2019.000000	1.374390e+11

Cantidad de transistores en la RAM a lo largo del tiempo



Luego de ver los datos procesados y limpios de la RAM, se puede llegar a la conclusion de que lo mejor seria descartar este dataset ya que tiene demasiados valores Nan

- [ROM \(memoria de solo lectura\)](#)

Los datos fueron obtendios del mismo sitio que los anteriores [wikipedia - cantidad de transistores](#). A los datos de la ROM le aplicaremos el mismo tratamiento que a los anteriores, solamente nos quedaremos con la fecha y la cantidad de transistores.

```
In [ ]: rom = pd.read_csv(f'{DATASETS_FOLDER}/rom.csv')
#
print('# Muestra del dataset original')
display(rom.tail())
#
rom = mantener_columnas(rom, ['Date of introduction', 'Transistor count'])
rom = fecha_a_anio(rom, 'Date of introduction')
rom = remover_commas(rom)
rom = convertir_a_flotante(rom, ['Transistor count'])
rom = rom.rename(columns={'Transistor count': 'ROM transistor count'})
rom = cortar_agregar_anios(rom)
rom = mantener_solo_valor_mas_alto(rom, 'ROM transistor count')
rom = rom.fillna(method='ffill')
# MALO: computacion_dfs.append(rom) este no tiene suficientes datos valido
#
print('# Muestra del dataset procesado y limpio')
display(rom.tail())
display(rom.describe())
rom.plot(
    x=COLUMNA_FECHA,
    y='ROM transistor count',
    title='Cantidad de transistores en la ROM a lo largo del tiempo',
    figsize=(4, 2),
)
plt.show()

# Muestra del dataset original
```

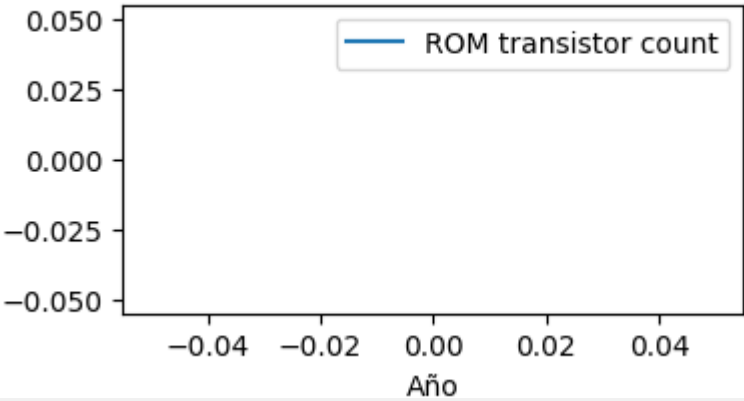
	Chip name	Capacity (bits)	ROM type	Transistor count	Date of introduction	Manufacturer(s)	Process	Area	Ref
17	27512	512 Kb	EPROM (HMOS)	524,288	1984	Intel	?	?	NaN
18	?	1 Mb	EPROM (CMOS)	1,048,576	1984	NEC	1,200 nm	?	NaN
19	?	4 Mb	EPROM (CMOS)	4,194,304	1987	Toshiba	800 nm	?	NaN
20	?	16 Mb	EPROM (CMOS)	16,777,216	1990	NEC	600 nm	?	NaN
21	?	16 Mb	MROM	16,777,216	1995	AKM, Hitachi	?	?	NaN

Muestra del dataset procesado y limpio

	Año	ROM transistor count
14	2015	NaN
15	2016	NaN
16	2017	NaN
17	2018	NaN
18	2019	NaN

	Año	ROM transistor count
count	19.000000	0.0
mean	2010.000000	NaN
std	5.627314	NaN
min	2001.000000	NaN
25%	2005.500000	NaN
50%	2010.000000	NaN
75%	2014.500000	NaN
max	2019.000000	NaN

Cantidad de transistores en la ROM a lo largo del tiempo



Como observamos en el muestreo de datos luego de la limpieza, al igual que con la RAM, vamos a descartar los datos de la ROM ya que faltan demasiados registros desde el 2000 hasta la actualidad.

- [Internet](#)

Datos fueron obtenidos del sitio worldbank.org.

En este dataset proporciona el porcentaje de la poblacion que tiene acceso a internet, una columna por anio y una fila para cada pais, se debera convertir las columnas de anio a filas para poder unificarlo con los demas datasets.

```
In [ ]: internet = pd.read_csv(f'{DATASETS_FOLDER}/theworldbank_internet.csv')
#
print('# Muestra del dataset original')
display(internet.tail())
#
internet = columna_fecha_a_fila(internet, 'United States', 'US internet p
internet = cortar_agregar_anios(internet)
computacion_dfs.append(internet)
#
print('# Muestra del dataset procesado y limpio')
display(internet.tail())
display(internet.describe())
internet.plot(
    x=COLUMNA_FECHA,
    y='US internet percentage',
    title='Porcentaje de la poblacion con internet en los EEUU a lo largo
    figsize=(4, 2),
)
plt.show()
```

Muestra del dataset original

	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	..
261	Kosovo	XKX	Individuals using the Internet (% of population)	IT.NET.USER.ZS	NaN	NaN	NaN	NaN	NaN	NaN	..
262	Yemen, Rep.	YEM	Individuals using the Internet (% of population)	IT.NET.USER.ZS	NaN	NaN	NaN	NaN	NaN	NaN	..
263	South Africa	ZAF	Individuals using the Internet (% of population)	IT.NET.USER.ZS	NaN	NaN	NaN	NaN	NaN	NaN	..
264	Zambia	ZMB	Individuals using the Internet (% of population)	IT.NET.USER.ZS	NaN	NaN	NaN	NaN	NaN	NaN	..
265	Zimbabwe	ZWE	Individuals using the Internet (% of population)	IT.NET.USER.ZS	NaN	NaN	NaN	NaN	NaN	NaN	..

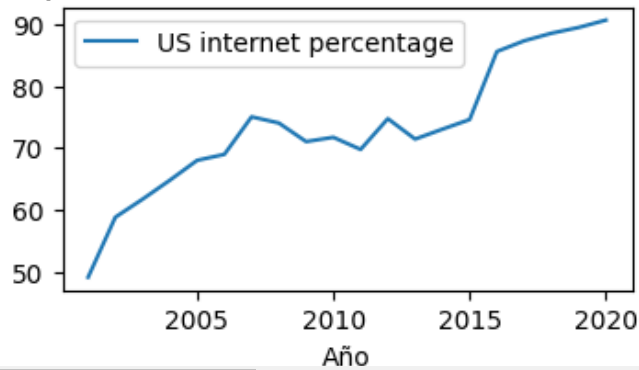
5 rows × 68 columns

Muestra del dataset procesado y limpio

	Año	US internet percentage
56	2016	85.544421
57	2017	87.274889
58	2018	88.498903
59	2019	89.430285
60	2020	90.620470

	Año	US internet percentage
count	20.00000	20.000000
mean	2010.50000	73.383174
std	5.91608	10.809027
min	2001.00000	49.080832
25%	2005.75000	68.690408
50%	2010.50000	72.345000
75%	2015.25000	77.636105
max	2020.00000	90.620470

Porcentaje de la poblacion con internet en los EEUU a lo largo del tiempo



```
In [ ]: ## TODO: agregar dataset relacionado a los telefonos celular.
```

```
In [ ]: ## TODO:
## agregar dataset relacionado al avance de la IA, tengase en cuenta
## que el avance de las redes neuronales se pueden medir con la cantidad
## capas y neuronas utilizadas, mientras que el avance en transformers p
## medirse con la cantidad de parametros utilizados.
## Otra manera de medir el avance podria ser mediante la cantidad de pap
```

```
In [ ]: ## TODO: agregar dataset relacionado a la memoria SSD.
```

Unificamos todos los datasets limpios relacionados a la computacion en un solo dataframe y revisamos si se realizo correctamente

```
In [ ]: computer_advances = fusionar_por_fecha(computacion_dfs)
computer_advances.info()
```

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

```
Int64Index: 20 entries, 0 to 19
```

```
Data columns (total 6 columns):
```

#	Column	Non-Null Count	Dtype
0	Año	20 non-null	int64
1	FGMOS transistor count	19 non-null	float64
2	FPGA transistor count	19 non-null	float64
3	GPU transistor count	20 non-null	float64
4	Microprocessor transistor count	20 non-null	int64
5	US internet percentage	20 non-null	float64

```
dtypes: float64(4), int64(2)
```

```
memory usage: 1.1 KB
```

Revisando la economia de Estados Unidos

Los datos acerca la economia de los Estados Unidos pueden ser de gran ayuda para normalizar o escalar los datos, por ejemplo no es lo mismo el 1% de la poblacion en el 2005 que ese mismo porcentaje en el 2019, tambien puede ayudarnos el GDP para diferenciar si el crecimiento en una industria esta mas relacionada al crecimiento de la economia en general o si es un crecimiento independiente.

- Poblacion

Datos acerca la poblacion fueron obtenidos del sitio worldbank.org.

```
In [ ]: poblacion = pd.read_csv(f'{DATASETS_FOLDER}/theworldbank_population.csv')
#
print('# Muestra del dataset original')
display(poblacion.tail())
#
poblacion = columna_fecha_afila(poblacion, 'United States', 'US Populati
poblacion = cortar_agregar_anios(poblacion)
#
print('# Muestra del dataset procesado y limpio')
display(poblacion.tail())
display(poblacion.describe())
poblacion.plot(
    x=COLUMNA_FECHA,
    y='US Population',
    title='Poblacion de US a lo largo del tiempo',
    figsize=(4, 2),
)
plt.show()
```

Muestra del dataset original

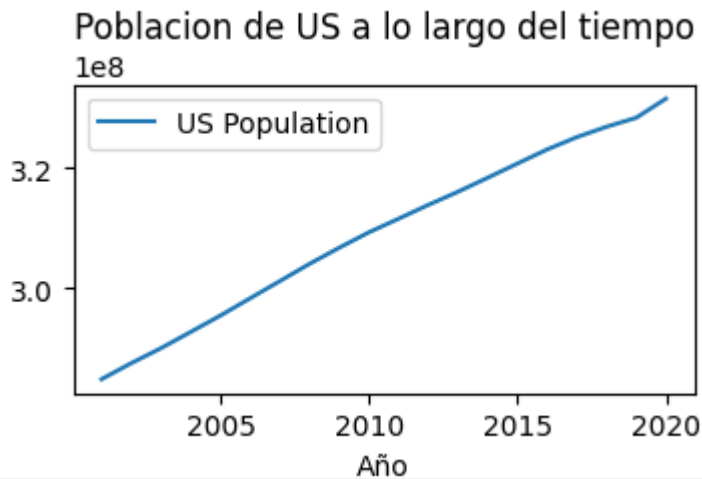
	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	
261	Kosovo	XKX	Population, total	SP.POP.TOTL	947000.0	966000.0	994000.0	102
262	Yemen, Rep.	YEM	Population, total	SP.POP.TOTL	5542459.0	5646668.0	5753386.0	586
263	South Africa	ZAF	Population, total	SP.POP.TOTL	16520441.0	16989464.0	17503133.0	1804
264	Zambia	ZMB	Population, total	SP.POP.TOTL	3119430.0	3219451.0	3323427.0	343
265	Zimbabwe	ZWE	Population, total	SP.POP.TOTL	3806310.0	3925952.0	4049778.0	417

5 rows × 67 columns

Muestra del dataset procesado y limpio

	Año	US Population
56	2016	323071755.0
57	2017	325122128.0
58	2018	326838199.0
59	2019	328329953.0
60	2020	331501080.0

	Año	US Population
count	20.00000	2.000000e+01
mean	2010.50000	3.093169e+08
std	5.91608	1.450577e+07
min	2001.00000	2.849690e+08
25%	2005.75000	2.976641e+08
50%	2010.50000	3.104553e+08
75%	2015.25000	3.213222e+08
max	2020.00000	3.315011e+08



- GDP (producto domestico bruto)

Datos acerca la poblacion y el GDP fueron obtenidos del sitio worldbank.org.

```
In [ ]: gdp = pd.read_csv(f'{DATASETS_FOLDER}/theworlbank_gdp.csv')
#
print('# Muestra del dataset original')
display(gdp.tail())
#
gdp = columna_fecha_a_fila(gdp, 'United States', 'US GDP')
gdp = cortar_agregar_anios(gdp)
gdp = normalizar_con_la_poblacion(gdp, 'US GDP')
#
print('# Muestra del dataset procesado y limpio')
display(gdp.tail())
display(gdp.describe())
gdp.plot(
    x=COLUMNA_FECHA,
    y='US GDP',
    title='US GDP per 100k people over time',
    figsize=(4, 2),
)
plt.show()
```

Muestra del dataset original

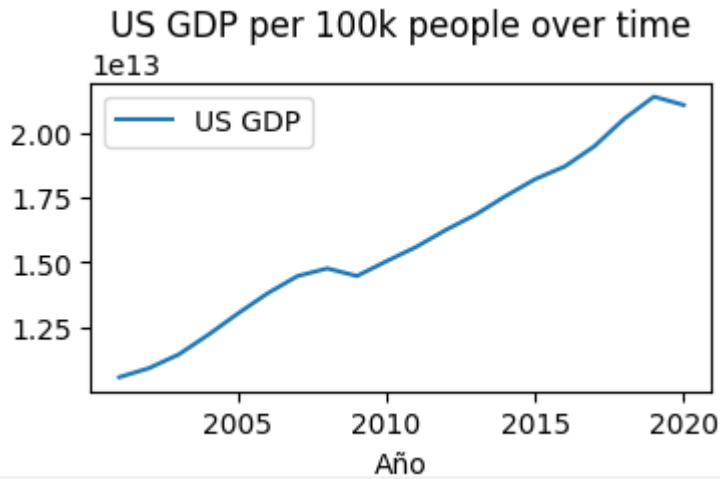
	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962
261	Kosovo	XKX	GDP (current US\$)	NY.GDP.MKTP.CD	NaN	NaN	NaN
262	Yemen, Rep.	YEM	GDP (current US\$)	NY.GDP.MKTP.CD	NaN	NaN	NaN
263	South Africa	ZAF	GDP (current US\$)	NY.GDP.MKTP.CD	8.748597e+09	9.225996e+09	9.813996e+09
264	Zambia	ZMB	GDP (current US\$)	NY.GDP.MKTP.CD	7.130000e+08	6.962857e+08	6.931429e+08
265	Zimbabwe	ZWE	GDP (current US\$)	NY.GDP.MKTP.CD	1.052990e+09	1.096647e+09	1.117602e+09

5 rows × 67 columns

Muestra del dataset procesado y limpio

	Año	US GDP
56	2016	1.869511e+13
57	2017	1.947734e+13
58	2018	2.053306e+13
59	2019	2.138098e+13
60	2020	2.106047e+13

	Año	US GDP
count	20.00000	2.000000e+01
mean	2010.50000	1.582056e+13
std	5.91608	3.345485e+12
min	2001.00000	1.058193e+13
25%	2005.75000	1.362149e+13
50%	2010.50000	1.532435e+13
75%	2015.25000	1.832829e+13
max	2020.00000	2.138098e+13



Unificamos todos los datasets procesados y limpios relacionados a la economia, en uno solo

```
In [ ]: economy = fusionar_por_fecha([gdp, poblacion])
economy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20 entries, 0 to 19
Data columns (total 3 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Año             20 non-null    int64
1   US GDP          20 non-null    float64
2   US Population   20 non-null    float64
dtypes: float64(2), int64(1)
memory usage: 640.0 bytes
```

Procesando datos relacionado a la industria de los bienes raices

Las ventas de bienes raices entre 2001 y 2020 fueron obtenidos de data.gov, este contiene los registros de todas las ventas de bienes raices registradas por el gobierno, notese que el dataset fue nombrado "real-estate-sales-2001-2018" pero en realidad contiene datos hasta el 2020.

Para analizar esta industria con el mayor de los detalles, se procedera a procesar los datos y almacenar estos resultados en multiples columnas como por ejemplo; "ventas", "inversion", "inversion en propiedades de 1 familia", "inversion en propiedades de 2 familia", etc.

Mas informacion acerca la clasificacion de viviendas: [Multifamily residential classification](#).

```
In [ ]: bienes_raices_ventas = pd.read_csv(f'{DATASETS_FOLDER}/real_estate_sales.
#
print('# Muestra del dataset original')
display(bienes_raices_ventas.tail())
#
bienes_raices_ventas = mantener_columnas(bienes_raices_ventas, [
    'Date Recorded', 'Sale Amount', 'Property Type', 'Residential Type',
```

```

    ])
# Guardamos el dataset porque Github no admite archivos mayores a 100MB
if GUARDAR_DATOS_PROCESADOS:
    bienes_raices_ventas.to_csv(f'{DATASETS_FOLDER}/real_estate_sales.csv')
    bienes_raices_ventas = fecha_a_anio(bienes_raices_ventas, 'Date Recorded')
#
print('# Muestra del dataset procesado y limpio')
display(bienes_raices_ventas.tail())
display(bienes_raices_ventas.describe())

```

Muestra del dataset original

```

/tmp/ipykernel_72182/1474315471.py:1: DtypeWarning: Columns (2,3) have mixed
types. Specify dtype option on import or set low_memory=False.
    bienes_raices_ventas = pd.read_csv(f'{DATASETS_FOLDER}/real_estate_sales.csv')

```

	Date Recorded	Sale Amount	Property Type	Residential Type
997208	06/24/2020	53100.0	Single Family	Single Family
997209	11/27/2019	76000.0	Single Family	Single Family
997210	04/27/2020	210000.0	Single Family	Single Family
997211	06/03/2020	280000.0	Single Family	Single Family
997212	12/20/2019	7450000.0	NaN	NaN

```

/tmp/ipykernel_72182/3403905049.py:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```

    df[columna_fecha] = df[columna_fecha].apply(lambda x: re.findall(fecha_regex, x)[0])

```

Muestra del dataset procesado y limpio

	Año	Sale Amount	Property Type	Residential Type
997208	2020	53100.0	Single Family	Single Family
997209	2019	76000.0	Single Family	Single Family
997210	2020	210000.0	Single Family	Single Family
997211	2020	280000.0	Single Family	Single Family
997212	2019	7450000.0	NaN	NaN

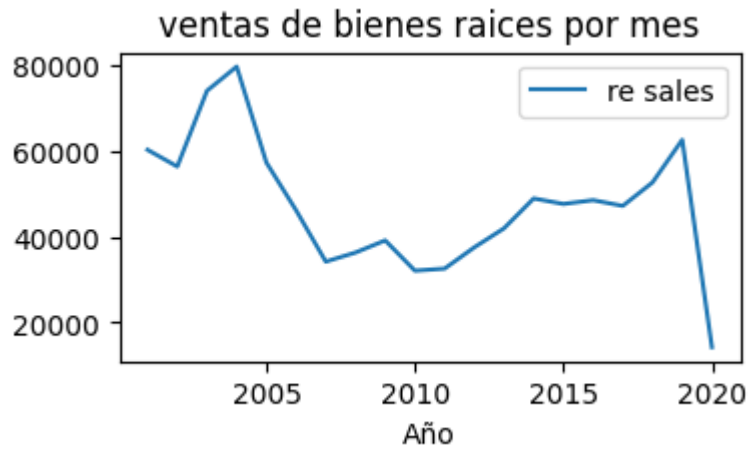
	Sale Amount
count	9.972110e+05
mean	3.911520e+05
std	5.347276e+06
min	0.000000e+00
25%	1.400000e+05
50%	2.250000e+05
75%	3.650000e+05
max	5.000000e+09

- Ventas de bienes raices por mes

```
In [ ]: #
br_ventas_por_anio = pd.DataFrame()
br_ventas_por_anio[COLUMNA_FECHA] = bienes_raices_ventas[COLUMNA_FECHA].u
br_ventas_por_anio = asegurar_tipo_de_dato_fecha(br_ventas_por_anio)
br_ventas_por_anio['re sales'] = bienes_raices_ventas.groupby(COLUMNA_FEC
br_ventas_por_anio = cortar_agregar_anios(br_ventas_por_anio)
br_ventas_por_anio = br_ventas_por_anio.fillna(method='ffill')
br_ventas_por_anio = normalizar_con_la_poblacion(br_ventas_por_anio, 're
#
display(br_ventas_por_anio.tail())
display(br_ventas_por_anio.describe())
br_ventas_por_anio.plot(
    x=COLUMNA_FECHA,
    y='re sales',
    title='ventas de bienes raices por mes',
    figsize=(4, 2)
)
plt.show()
```

	Año	re sales
16	2016	48493
17	2017	47165
18	2018	52622
19	2019	62534
20	2020	14291

	Año	re sales
count	20.00000	20.000000
mean	2010.50000	47438.450000
std	5.91608	15192.165091
min	2001.00000	14291.000000
25%	2005.75000	37211.000000
50%	2010.50000	47388.000000
75%	2015.25000	56550.250000
max	2020.00000	79566.000000



- Inversion en bienes raices por mes

```
In [ ]: #
br_inversion_por_anio = pd.DataFrame()
br_inversion_por_anio[COLUMNA_FECHA] = bienes_raices_ventas[COLUMNA_FECHA]
br_inversion_por_anio = asegurar_tipo_de_dato_fecha(br_inversion_por_anio)
br_inversion_por_anio['re investment'] = bienes_raices_ventas.groupby(COLUMNA_FECHA).sum()['re sales']
br_inversion_por_anio = cortar_agregar_anios(br_inversion_por_anio)
br_inversion_por_anio = normalizar_con_la_poblacion(br_inversion_por_anio)
#
display(br_inversion_por_anio.tail())
display(br_inversion_por_anio.describe())
br_inversion_por_anio.plot(
    x=COLUMNA_FECHA,
    y='re investment',
    title='US real estate total investment per year',
    figsize=(4, 2)
)
plt.show()
```

	Año	re investment
16	2016	1.895052e+10
17	2017	1.844713e+10
18	2018	2.084145e+10
19	2019	2.721690e+10
20	2020	3.178363e+09

	Año	re investment
count	20.00000	2.000000e+01
mean	2010.50000	1.791407e+10
std	5.91608	6.191910e+09
min	2001.00000	3.178363e+09
25%	2005.75000	1.405080e+10
50%	2010.50000	1.782573e+10
75%	2015.25000	2.175421e+10
max	2020.00000	3.021333e+10



- Inversion mensual en bienes raices por cada tipo de propiedad y residencia

```
In [ ]: def obtener_categorias(df:pd.DataFrame, column:str) -> list:
        types = df[column].unique()
        types = [str(x) for x in types]
        return types
def imprimir_categorias(df: pd.DataFrame, column:str) -> None:
    types = obtener_categorias(df, column)
    for i, x in enumerate(types):
        print(f'{i+1}.\t{x}')
print('Tipos de propiedades que tenemos presente en el dataset de bienes
imprimir_categorias(bienes_raices_ventas, 'Property Type')
print()
print('Tipos de residencias que tenemos presente en el dataset de bienes
imprimir_categorias(bienes_raices_ventas, 'Residential Type')
```

Tipos de propiedades que tenemos presente en el dataset de bienes raice

- s:
1. Commercial
 2. Residential
 3. Vacant Land
 4. nan
 5. Apartments
 6. Industrial
 7. Public Utility
 8. Condo
 9. Two Family
 10. Three Family
 11. Single Family
 12. Four Family

Tipos de residencias que tenemos presente en el dataset de bienes raice

- s:
1. nan
 2. Single Family
 3. Condo
 4. Two Family
 5. Three Family
 6. Four Family

Procederemos a agrupar las ventas de bienes raices por mes, tipo de propiedad y residencia en nuevas columnas

```
In [ ]: def agrupar_por_tipo_por_fecha(df:pd.DataFrame, column:str) -> pd.DataFrame:
    resultado = pd.DataFrame()
    resultado[COLUMNA_FECHA] = bienes_raices_ventas[COLUMNA_FECHA].unique
    resultado = resultado.sort_values(by=COLUMNA_FECHA).reset_index(drop=
    #
    resultado = asegurar_tipo_de_dato_fecha(resultado)
    df = asegurar_tipo_de_dato_fecha(df)
    #
    for type in obtener_categorias(df, column):
        new_column_name = f're {type.upper()} investment'
        resultado[new_column_name] = 0
        for year in resultado[COLUMNA_FECHA]:
            sales_amount = df[(df[column] == type) & (df[COLUMNA_FECHA] ==
            resultado.loc[resultado[COLUMNA_FECHA] == year, new_column_na
            resultado = normalizar_con_la_poblacion(resultado, new_column_na
    return resultado

property_types_annual = agrupar_por_tipo_por_fecha(bienes_raices_ventas,
residential_types_annual = agrupar_por_tipo_por_fecha(bienes_raices_venta

# Eliminamos las columnas que hayamos visto que tienen muchos valores en
property_types_annual = mantener_columnas(property_types_annual, [
    COLUMNA_FECHA,
    're CONDO investment',
    're TWO FAMILY investment',
    're THREE FAMILY investment',
    're SINGLE FAMILY investment',
    're FOUR FAMILY investment',
])
property_types_annual = mantener_columnas(property_types_annual, [
    COLUMNA_FECHA,
    're SINGLE FAMILY investment',
```

```
're CONDO investment',
're TWO FAMILY investment',
're THREE FAMILY investment',
're FOUR FAMILY investment',
])

print("Dataset de ventas por tipo de propiedad")
display(property_types_annual)
display(property_types_annual.describe())
print("Dataset de ventas por tipo de residencia")
display(residential_types_annual)
display(residential_types_annual.describe())

# Grafica de ventas por tipo de propiedad
property_types_annual.plot(
    x=COLUMNA_FECHA,
    y=property_types_annual.columns[1:],
    title='Venta de bienes raices por tipo de propiedad a lo largo del ti
)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.show()

# Grafica de ventas por tipo de residencia
residential_types_annual.plot(
    x=COLUMNA_FECHA,
    y=residential_types_annual.columns[1:],
    title='Venta de bienes raices por tipo de residencia a lo largo del t
)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.show()
```

Dataset de ventas por tipo de propiedad

	Año	re SINGLE FAMILY investment	re CONDO investment	re TWO FAMILY investment	re THREE FAMILY investment	re FOUR FAMILY investment
0	1999	0.000000e+00	9.500000e+04	0.000000e+00	0	0
1	2001	2.402711e+06	8.800000e+04	0.000000e+00	0	0
2	2002	0.000000e+00	0.000000e+00	0.000000e+00	0	0
3	2003	1.589000e+05	0.000000e+00	0.000000e+00	0	0
4	2004	1.949900e+06	6.329000e+05	0.000000e+00	0	0
5	2005	0.000000e+00	2.770000e+05	2.640000e+05	0	0
6	2006	3.210213e+09	5.948589e+08	1.603019e+08	86599217	10182700
7	2007	1.455175e+10	2.423580e+09	5.385773e+08	228334313	25293008
8	2008	9.249377e+09	1.571634e+09	3.353331e+08	134461433	23560218
9	2009	8.241461e+09	1.533074e+09	2.929898e+08	115687933	21736964
10	2010	9.933565e+09	1.622946e+09	3.010015e+08	140404872	24317690
11	2011	7.290372e+09	1.231526e+09	2.110223e+08	104523619	21693403
12	2012	9.260353e+09	1.211207e+09	2.299405e+08	91109852	25816807
13	2013	1.028991e+10	1.433486e+09	2.556605e+08	106807382	20942328
14	2014	1.057817e+10	1.787715e+09	3.111695e+08	122285490	35849185
15	2015	1.241745e+10	1.938384e+09	3.668705e+08	149280615	39880680
16	2016	1.049482e+10	3.894486e+09	3.498937e+08	135213185	46846302
17	2017	1.290385e+10	2.071585e+09	4.077366e+08	152915555	34627440
18	2018	1.185046e+10	1.913916e+09	4.677992e+08	201257041	71015671
19	2019	1.216468e+10	2.469696e+09	5.949037e+08	278538608	71852599
20	2020	1.359104e+10	1.732257e+09	4.329061e+08	216103966	202110732
21	2021	0.000000e+00	0.000000e+00	0.000000e+00	0	0

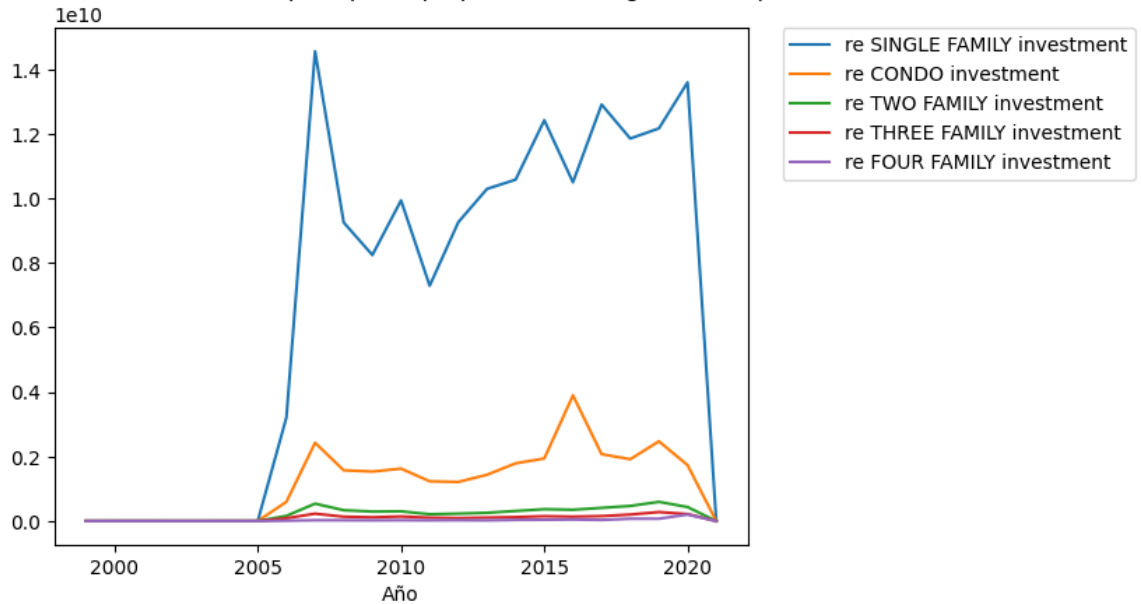
	Año	re SINGLE FAMILY investment	re CONDO investment	re TWO FAMILY investment	re THREE FAMILY investment	re FOUR FAMILY investment
count	22.000000	2.200000e+01	2.200000e+01	2.200000e+01	2.200000e+01	2.200000e+01
mean	2010.454545	7.092363e+09	1.246884e+09	2.389259e+08	1.028874e+08	3.071481e+07
std	6.573593	5.469519e+09	1.062153e+09	1.941756e+08	8.514759e+07	4.391946e+07
min	1999.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	2005.250000	2.063103e+06	3.659750e+05	6.600000e+04	0.000000e+00	0.000000e+00
50%	2010.500000	9.254865e+09	1.483280e+09	2.743252e+08	1.112477e+08	2.264859e+07
75%	2015.750000	1.153239e+10	1.882366e+09	3.626263e+08	1.470617e+08	3.554375e+07
max	2021.000000	1.455175e+10	3.894486e+09	5.949037e+08	2.785386e+08	2.021107e+08

Dataset de ventas por tipo de residencia

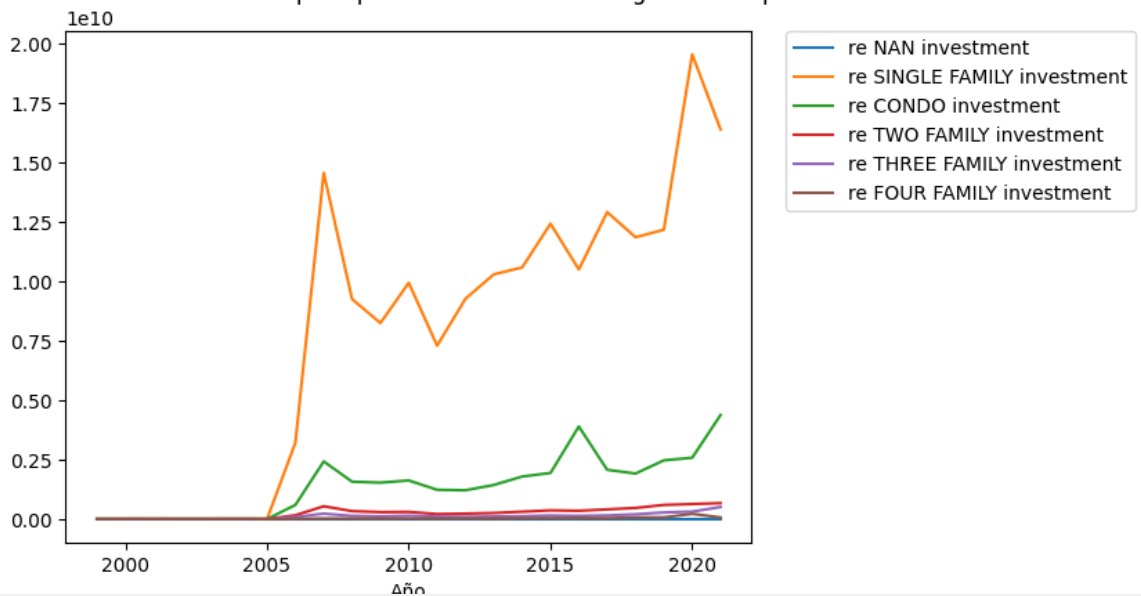
	Año	re NAN investment	re SINGLE FAMILY investment	re CONDO investment	re TWO FAMILY investment	re THREE FAMILY investment	re FOUR FAMILY investment
0	1999	0	0.000000e+00	9.500000e+04	0.000000e+00	0.000000e+00	0
1	2001	0	2.402711e+06	8.800000e+04	0.000000e+00	0.000000e+00	0
2	2002	0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0
3	2003	0	1.589000e+05	0.000000e+00	0.000000e+00	0.000000e+00	0
4	2004	0	1.949900e+06	6.329000e+05	0.000000e+00	0.000000e+00	0
5	2005	0	0.000000e+00	2.770000e+05	2.640000e+05	0.000000e+00	0
6	2006	0	3.210213e+09	5.948589e+08	1.603019e+08	8.659922e+07	10182700
7	2007	0	1.455175e+10	2.423580e+09	5.385773e+08	2.283343e+08	25293008
8	2008	0	9.249377e+09	1.571634e+09	3.353331e+08	1.344614e+08	23560218
9	2009	0	8.241461e+09	1.533074e+09	2.929898e+08	1.156879e+08	21736964
10	2010	0	9.933565e+09	1.622946e+09	3.010015e+08	1.404049e+08	24317690
11	2011	0	7.290372e+09	1.231526e+09	2.110223e+08	1.045236e+08	21693403
12	2012	0	9.260353e+09	1.211207e+09	2.299405e+08	9.110985e+07	25816807
13	2013	0	1.028991e+10	1.433486e+09	2.556605e+08	1.068074e+08	20942328
14	2014	0	1.057817e+10	1.787715e+09	3.111695e+08	1.222855e+08	35849185
15	2015	0	1.241745e+10	1.938384e+09	3.668705e+08	1.492806e+08	39880680
16	2016	0	1.049482e+10	3.894486e+09	3.498937e+08	1.352132e+08	46846302
17	2017	0	1.290385e+10	2.071585e+09	4.077366e+08	1.529156e+08	34627440
18	2018	0	1.185046e+10	1.913916e+09	4.677992e+08	2.012570e+08	71015671
19	2019	0	1.216468e+10	2.469696e+09	5.949037e+08	2.785386e+08	71852599
20	2020	0	1.953491e+10	2.579188e+09	6.332712e+08	3.134679e+08	219821632
21	2021	0	1.638224e+10	4.375425e+09	6.710182e+08	5.073493e+08	68068605

	Año	re NAN investment	re SINGLE FAMILY investment	re CONDO investment	re TWO FAMILY investment	re THREE FAMILY investment	
count	22.000000	22.0	2.200000e+01	2.200000e+01	2.200000e+01	2.200000e+01	2.
mean	2010.454545	0.0	8.107186e+09	1.484264e+09	2.785342e+08	1.303744e+08	3.
std	6.573593	0.0	5.992677e+09	1.238747e+09	2.187378e+08	1.234736e+08	4.
min	1999.000000	0.0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.
25%	2005.250000	0.0	8.043554e+08	1.491894e+08	4.027347e+07	2.164980e+07	2.
50%	2010.500000	0.0	9.596959e+09	1.552354e+09	2.969957e+08	1.189867e+08	2.
75%	2015.750000	0.0	1.208612e+10	2.038284e+09	3.975201e+08	1.520068e+08	3.
max	2021.000000	0.0	1.953491e+10	4.375425e+09	6.710182e+08	5.073493e+08	2.

Venta de bienes raices por tipo de propiedad a lo largo del tiempo



Venta de bienes raices por tipo de residencia a lo largo del tiempo



Las ventas de bienes raices residenciales seran descartadas ya que es muy similar a las ventas de propiedades. Realizar el analisis con estos datos serian como duplicar la informacion.

Procederemos a agrupar los diferentes columnas que obtuvimos del procesamiento y las unificamos en un mismo dataset

```
In [ ]: bienes_raices = fusionar_por_fecha([
        br_ventas_por_anio, property_types_annual, br_inversion_por_anio,
    ])
bienes_raices.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20 entries, 1 to 20
Data columns (total 8 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   Año                                         20 non-null     int64
1   re sales                                   20 non-null     float64
2   re SINGLE FAMILY investment               20 non-null     float64
3   re CONDO investment                       20 non-null     float64
4   re TWO FAMILY investment                  20 non-null     float64
5   re THREE FAMILY investment                20 non-null     int64
6   re FOUR FAMILY investment                 20 non-null     int64
7   re investment                             20 non-null     float64
dtypes: float64(5), int64(3)
memory usage: 1.4 KB
```

Procesando datasets relacionados a la industria de los libros

El dataset a utilizar fue obtenido de [scostap - Goodreads Best Book Ever dataset](#), como los datos fueron scrapeados por el autor, para garantizar que los datos sean correctos, realizaremos una limpieza de outliers.

```
In [ ]: book = pd.read_csv(f'{DATASETS_FOLDER}/books_1.Best_Books_Ever.csv')
#
print('# Muestra del dataset original')
display(book.tail())
#
book = mantener_columnas(book, [
    'bookFormat', 'pages', 'publishDate', 'rating', 'likedPercent', 'price'
])
book = book.dropna()
# Formateamos la fecha
book = book[book['publishDate'].str.match(r'\d{1,2}/\d{1,2}/\d{2}')]
book['publishDate'] = pd.to_datetime(book['publishDate'], format="%m/%d/%y")
book['publishDate'] = book['publishDate'].dt.strftime('%Y-%m-%d')
# Nos aseguramos que estas columnas son numericas y no nulas
for col in ['pages', 'rating', 'likedPercent', 'price']:
    book = book[pd.to_numeric(book[col], errors='coerce').notnull()]
    book[col] = book[col].astype(float)
# Mostramos las columnas finales
print('# Muestra de los datos procesados y limpios')
display(book.tail())
display(book.describe())

# Muestra del dataset original
```


	bookId	title	series	author	rating	description	language	
52473	11492014-fractured	Fractured	Fateful #2	Cheri Schmidt (Goodreads Author)	4.00	The Fateful Trilogy continues with Fractured. ...	English	29400126
52474	11836711-anasazi	Anasazi	Sense of Truth #2	Emma Michaels	4.19	'Anasazi', sequel to 'The Thirteenth Chime' by...	English	99999999
52475	10815662-marked	Marked	Soul Guardians #1	Kim Richardson (Goodreads Author)	3.70	--READERS FAVORITE AWARDS WINNER 2011-- Sixteen...	English	97814610
52476	11330278-wayward-son	Wayward Son	NaN	Tom Pollack (Goodreads Author), John Loftus (G...	3.85	A POWERFUL TREMOR UNEARTH'S AN ANCIENT SECRETBu...	English	97814507
52477	10991547-daughter-of-helaman	Daughter of Helaman	Stripling Warrior #1	Misty Moncur (Goodreads Author)	4.02	Fighting in Helaman's army is Keturah's deepest...	English	97815995

5 rows × 25 columns

Muestra de los datos procesados y limpios

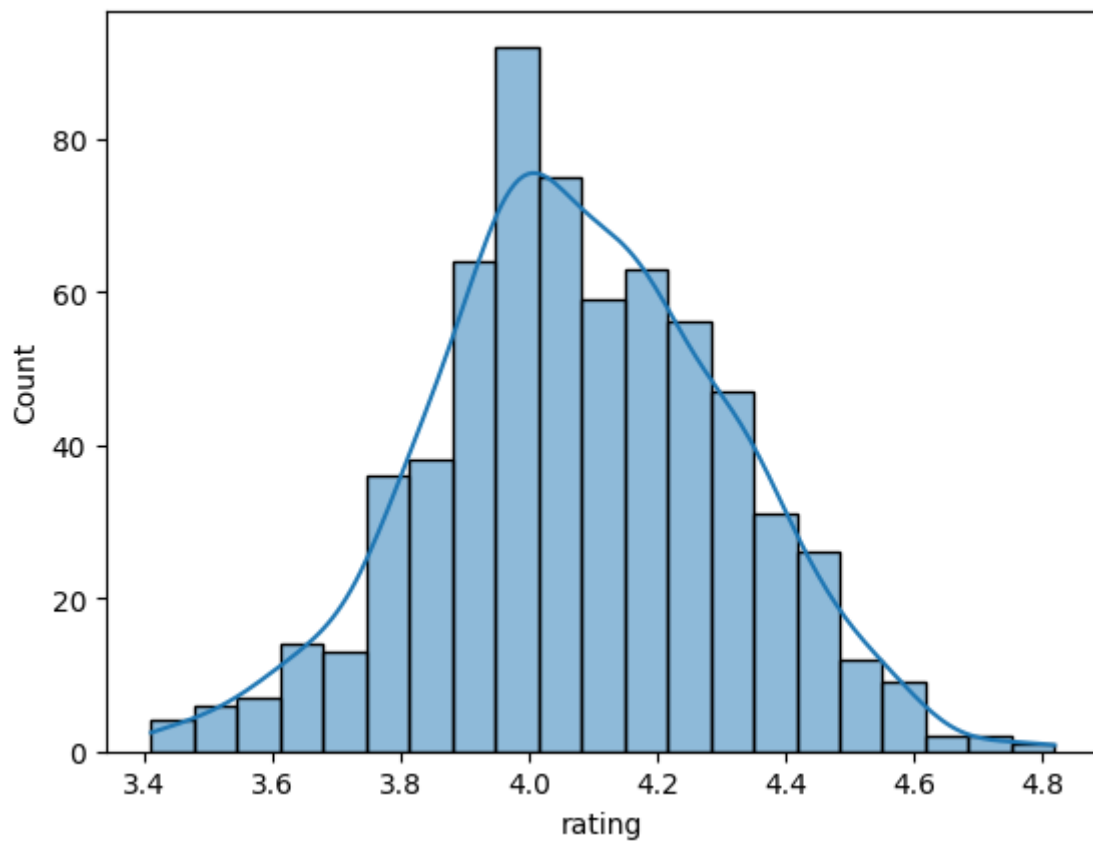
	bookFormat	pages	publishDate	rating	likedPercent	price
814	Mass Market Paperback	357.0	2008-04-29	4.30	97.0	2.86
815	Paperback	515.0	2005-09-22	3.65	86.0	2.86
816	Paperback	416.0	2001-02-01	4.17	95.0	3.55
818	Hardcover	516.0	2014-10-07	4.41	97.0	6.52
819	Hardcover	528.0	2010-04-27	4.40	97.0	6.50

	pages	rating	likedPercent	price
count	657.000000	657.000000	657.000000	657.000000
mean	423.127854	4.076514	92.576865	6.025403
std	289.973231	0.233532	3.742140	7.746120
min	26.000000	3.410000	78.000000	0.850000
25%	272.000000	3.930000	91.000000	2.900000
50%	369.000000	4.060000	93.000000	4.180000
75%	503.000000	4.230000	95.000000	6.270000
max	4100.000000	4.820000	99.000000	110.670000

Veamos la distribucion de los datos para analizar si son correctos e indicados para el analisis

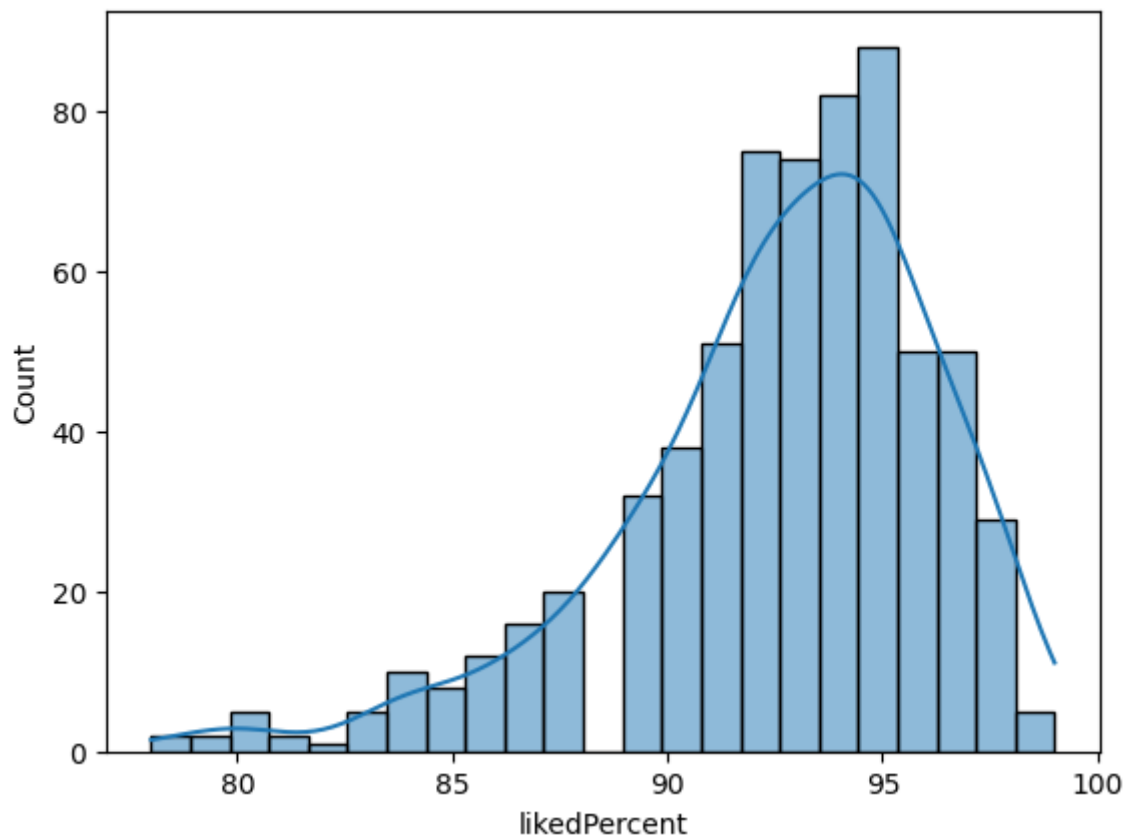
```
In [ ]: # NOTE: sns.histplot solo muestra un grafico por celda.  
sns.histplot(book['rating'], kde=True)
```

```
Out[ ]: <Axes: xlabel='rating', ylabel='Count'>
```



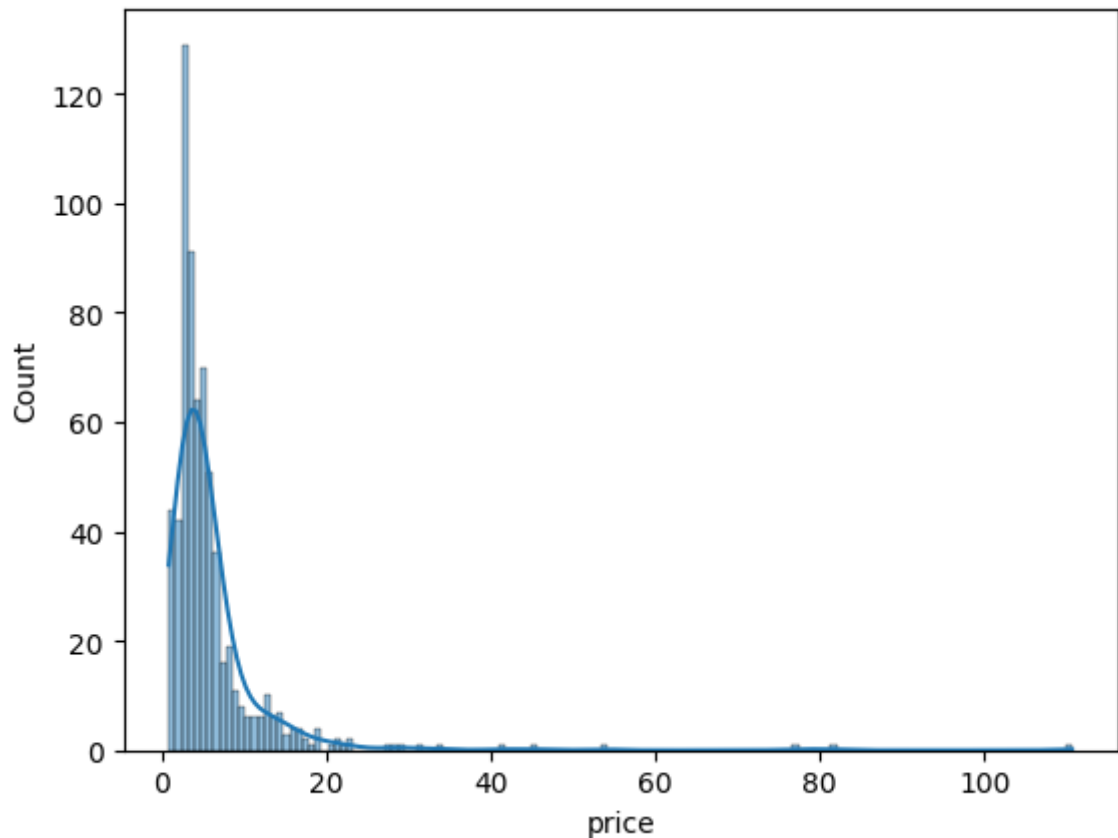
```
In [ ]: sns.histplot(book['likedPercent'], kde=True)
```

```
Out[ ]: <Axes: xlabel='likedPercent', ylabel='Count'>
```



```
In [ ]: sns.histplot(book['price'], kde=True)
```

```
Out[ ]: <Axes: xlabel='price', ylabel='Count'>
```



Podemos ver que tenemos libros con precios muy altos y alejados del "monton", esto puede afectar el analisis, continuemos analizando estos datos...

- Tratando los outliers

```
In [ ]: print('# Outliers')

fig, ax1 = plt.subplots(nrows=1, ncols=1, figsize=(9, 7))
bplot = ax1.boxplot(
    book.select_dtypes(include = ["float64"]),
    vert=True,
    patch_artist=True,
    labels=['pages', 'rating', 'likedPercent', 'price'],
    flierprops=dict(markerfacecolor='r', marker='D')
)
ax1.set_title('Grafico de cajas')

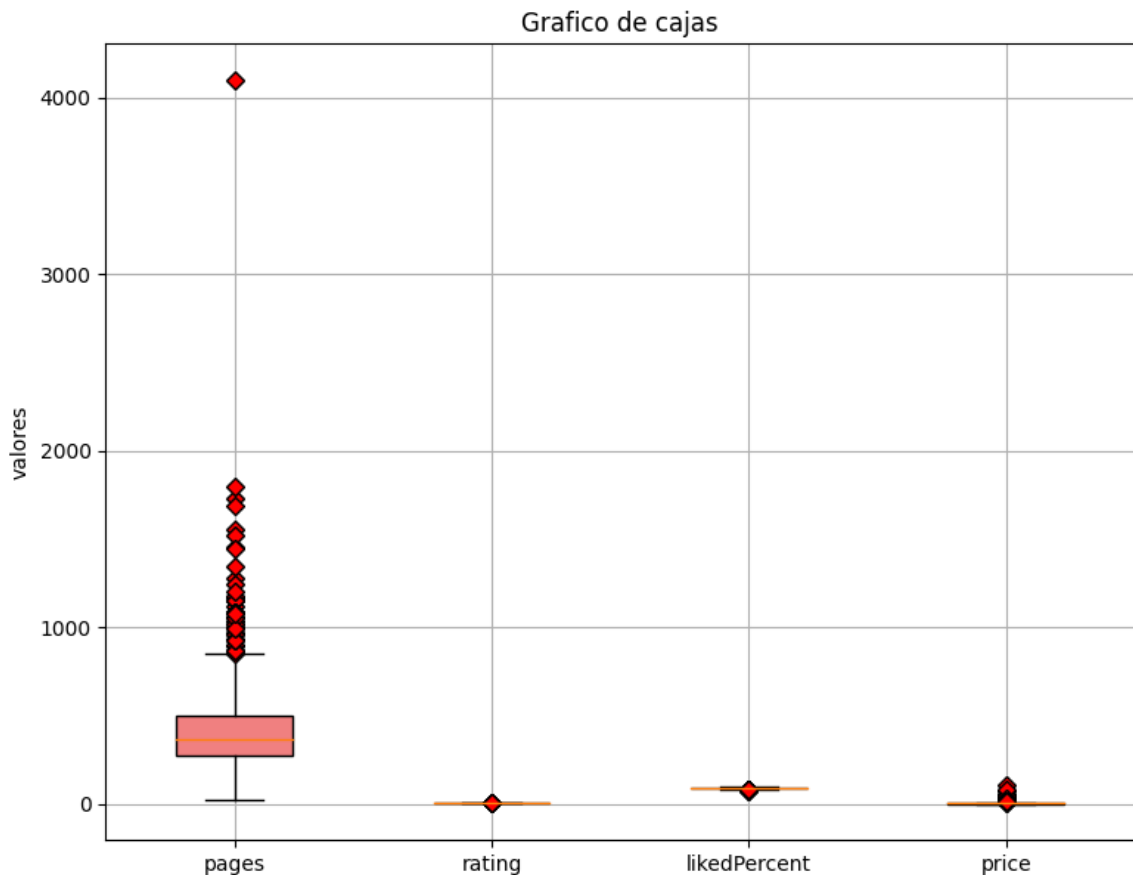
colors = ['lightcoral', 'mediumpurple', 'gold', 'aquamarine']

for patch, color in zip(bplot['boxes'], colors):
    patch.set_facecolor(color)

for ax in [ax1]:
    ax.yaxis.grid(True)
    ax.xaxis.grid(True)
    ax.set_ylabel('valores')

plt.show()
```

Outliers



```
In [ ]: # Removing outliers using Z score

def mostrar_min_max(df:pd.DataFrame, column:str) -> None:
    print(f'\tmin {column}: {df[column].min()}\tmax {column}: {df[column]
```

```

threshold = 3
for column in ['pages', 'rating', 'likedPercent', 'price']:
    print('')
    print(f'Removiendo outliers en la columna {column}, Z scores:')
    z = np.abs(stats.zscore(book[column]))
    display(z)
    print('Antes de remover outliers:')
    mostrar_min_max(book, 'pages')
    book = book[(z < threshold)]
    print('Despues de remover outliers:')
    mostrar_min_max(book, 'pages')

```

Removiendo outliers en la columna pages, Z scores:

```

0      0.169551
1      1.542255
4      0.268754
5      0.444766
6      0.973686

```

...

```

814    0.228222
815    0.317071
816    0.024600
818    0.320522
819    0.361937

```

Name: pages, Length: 657, dtype: float64

Antes de remover outliers:

min pages: 26.0 max pages: 4100.0

Despues de remover outliers:

min pages: 26.0 max pages: 1276.0

Removiendo outliers en la columna rating, Z scores:

```

0      0.134665
1      2.142164
4      0.448313
5      0.682423
6      1.204225

```

...

```

814    0.212702
815    0.512579
816    0.058131
818    0.517169
819    0.572254

```

Name: pages, Length: 648, dtype: float64

Antes de remover outliers:

min pages: 26.0 max pages: 1276.0

Despues de remover outliers:

min pages: 26.0 max pages: 1049.0

Removiendo outliers en la columna likedPercent, Z scores:

```
0      0.068177
1      2.547646
4      0.601600
5      0.870566
6      1.296981
```

```
...
```

```
814    0.157832
815    0.675434
816    0.153324
818    0.680708
819    0.743994
```

Name: pages, Length: 634, dtype: float64

Antes de remover outliers:

min pages: 26.0 max pages: 1049.0

Despues de remover outliers:

min pages: 26.0 max pages: 936.0

Removiendo outliers en la columna price, Z scores:

```
0      0.012310
1      2.858521
4      0.722762
5      1.017948
6      1.360906
```

```
...
```

```
814    0.110705
815    0.803793
816    0.230785
818    0.809581
819    0.879037
```

Name: pages, Length: 623, dtype: float64

Antes de remover outliers:

min pages: 26.0 max pages: 936.0

Despues de remover outliers:

min pages: 26.0 max pages: 870.0

Analizando el impacto de los avances de la computacion en las ventas de bienes raices

A continuacion analizaremos los datos de los avances en computacion y de bienes raices para ver si encontramos alguna correlacion y si logramos poder proyectar alguna regresion.

Ignoraremos las correlaciones que existan entre columnas propias de la computacion o de los bienes raices, es decir por ejemplo: alguna correlacion entre la cantidad de transistores en los Microprocesadores y la cantidad de transistores en las GPU, esto es debido a que solo nos interesa como afecta una industria a otra, no a si misma.

- Analizando correlaciones entre la computacion y bienes raices

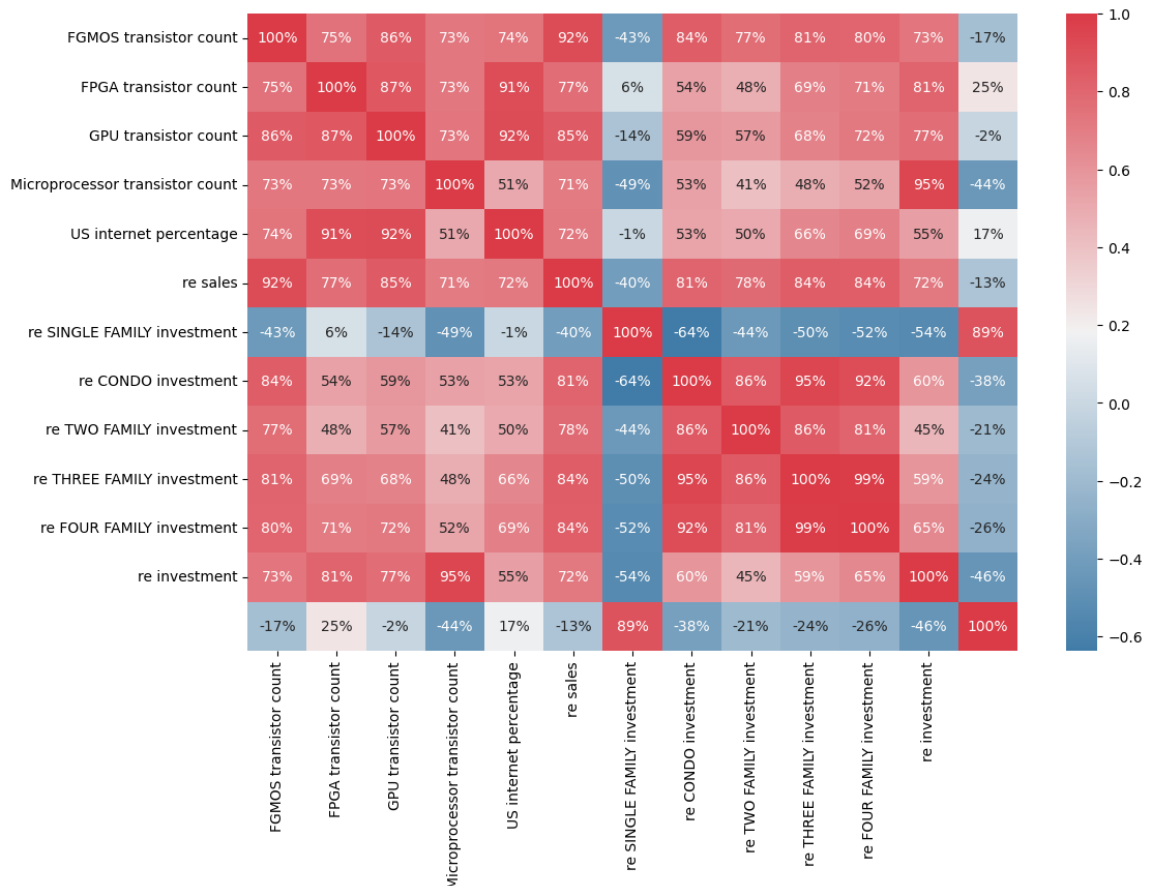
Procederemos a buscar correlaciones entre las columnas relacionadas a la computacion y las columnas relacionadas a los bienes raices, para ver si es posible proyectar alguna regresion.

```
In [ ]: computacion_y_bienes_raices = fusionar_por_fecha([computer_advances, bien
numeric_df = computacion_y_bienes_raices.select_dtypes(include=['float64'])
```

```
columns_to_analyze_correlation = numeric_df.columns[1:]
```

```
In [ ]: corr = computacion_y_bienes_raices.corr()
plt.subplots(figsize=(12,8))
sns.heatmap(
    corr,
    xticklabels=columns_to_analyze_correlation,
    yticklabels=columns_to_analyze_correlation,
    annot=True,
    fmt='.0%',
    cmap=sns.diverging_palette(240, 10, as_cmap=True)
)
```

Out[]: <Axes: >



En la grafica de calor podemos observar que en general, hay una alta correlacion entre las columnas de la computacion y las columnas de los bienes raices, teniendo la mayor correlacion las columnas de "Microprocessor transistor count" (cantidad de transistores en los microprocesadores) y la columna "re investment" (inversion en bienes raices) con un 95%, lo que nos indica que tenemos unas buenas chances de poder proyectar una regresion mas adelante.

Veamos en detalle las correlaciones que existen con las columnas "re sales" (ventas de bienes raices) y "re investment" (inversion en bienes raices), vemos estas en profundidad ya que estas abarcan a todas las demas columnas de bienes raices.

```
In [ ]: print("Variables mas correlacionadas con la inversion total en bienes rai
display(corr['re investment'].abs().sort_values(ascending=False))
print("Variables mas correlacionadas con la cantidad total de ventas en b
display(corr['re sales'].abs().sort_values(ascending=False))
```

```

Variables mas correlacionadas con la inversion total en bienes raices
re investment          1.000000
re sales               0.891257
re FOUR FAMILY investment 0.463065
GPU transistor count   0.441144
re SINGLE FAMILY investment 0.381849
re THREE FAMILY investment 0.263131
FGMOS transistor count 0.245910
re TWO FAMILY investment 0.243688
re CONDO investment    0.205083
Año                   0.174011
Microprocessor transistor count 0.167502
US internet percentage 0.133680
FPGA transistor count  0.018511

```

Name: re investment, dtype: float64

Variables mas correlacionadas con la cantidad total de ventas en bienes raices

```

re sales              1.000000
re investment         0.891257
re SINGLE FAMILY investment 0.635902
re FOUR FAMILY investment 0.539678
re THREE FAMILY investment 0.521715
re TWO FAMILY investment 0.501825
GPU transistor count  0.487257
re CONDO investment   0.435862
Año                  0.428682
US internet percentage 0.399518
FPGA transistor count 0.142902
FGMOS transistor count 0.056822
Microprocessor transistor count 0.008890

```

Name: re sales, dtype: float64

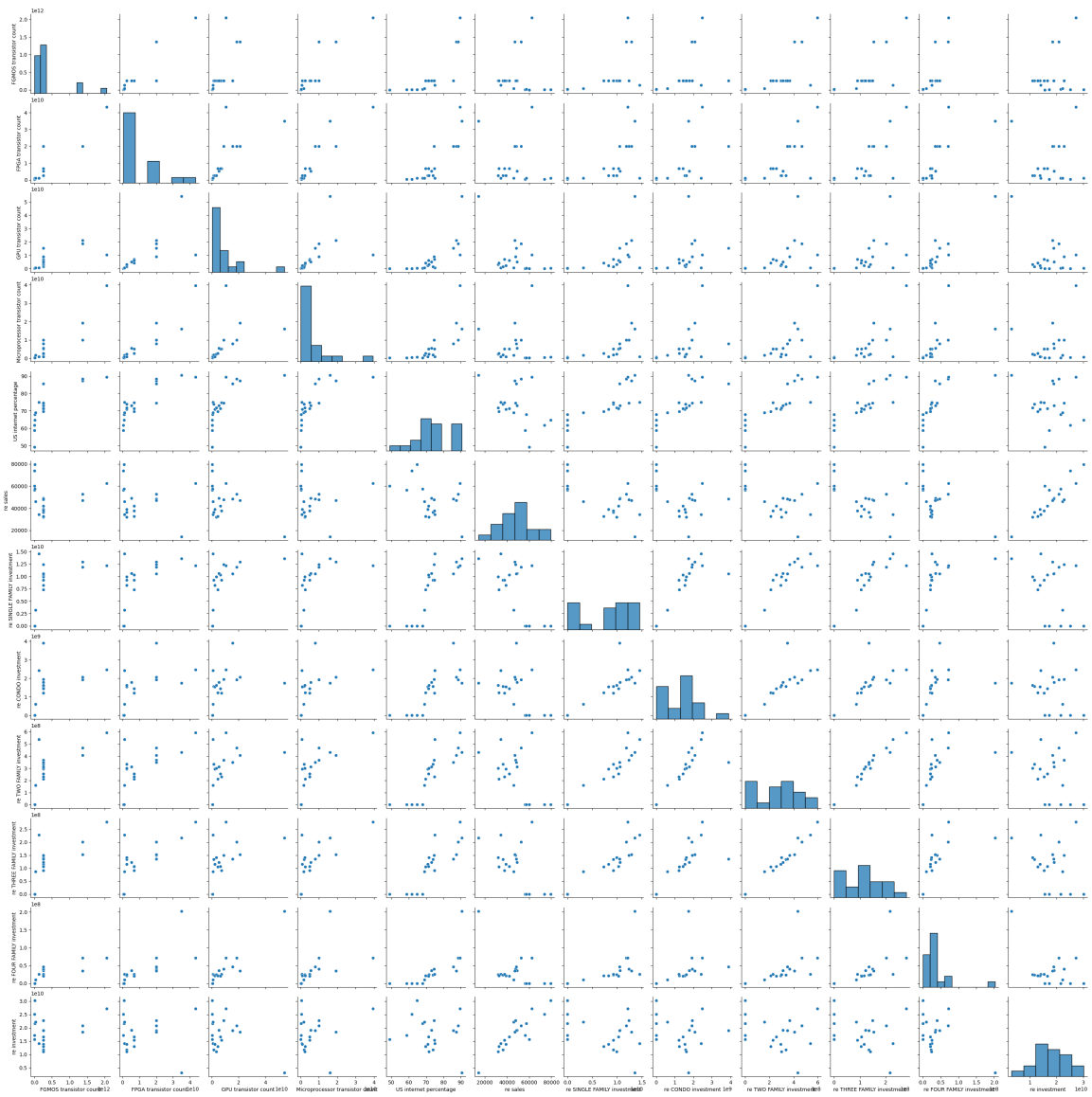
Podemos observar que el avance computacional mas correlacionado con los bienes raices fue el de las GPU, ya que tiene un coeficiente de 0.48 en relacion con las ventas y 0.48 con las inversiones, recordemos que segun el [coeficiente de correlacion de Pearson](#), valores cercanos a 1 (o -1) es una correlacion perfecta y cercanos a 0 es una correlacion nula.

Sin embargo, un coeficiente de 0.48 sigue estando muy alejado del perfecto 1, por lo tanto sigamos buscando correlaciones utilizando una grafica de pares (pairplot).

```

In [ ]: pp = sns.pairplot(
        numeric_df,
        x_vars=columns_to_analyze_correlation,
        y_vars=columns_to_analyze_correlation
    )

```

Podemos observar visualmente que hay correlaciones entre varias columnas, algunas de ellas son:

- "US internet percentage" con "re THREE FAMILY investment".
- "Microprocessor transistor count" con "re SINGLE FAMILY investment".

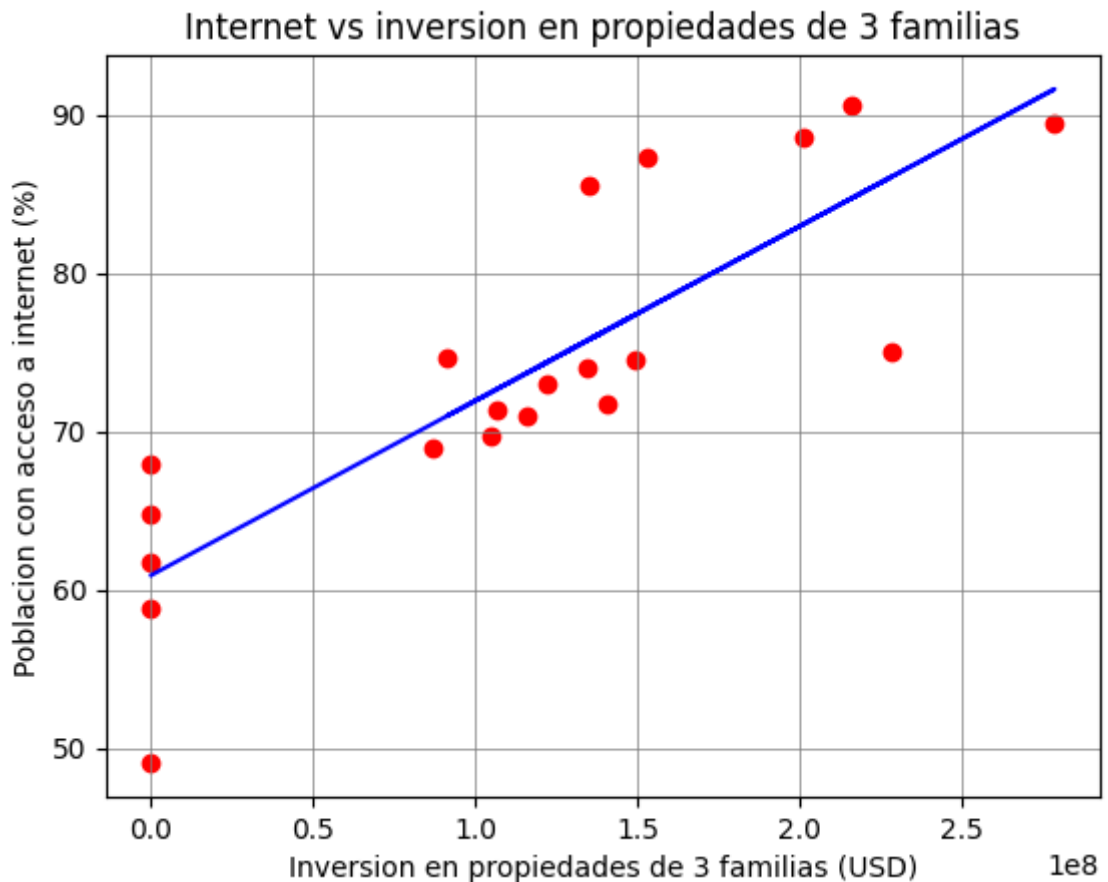
Tambien podemos ver correlaciones muy claras entre columnas de bienes raices pero recordemos que esas correlaciones estan fuera de nuestra investigacion.

Veamos en detalle estas correlaciones encontradas en el pairplot e intentemos hacer algunas proyecciones utilizando regresiones.

- Proyectando regresion lineal de "Porcentaje de la poblacion con acceso a internet" con "Inversion en propiedades de 3 familias"

```
In [ ]: X = computacion_y_bienes_raices['re THREE FAMILY investment'].values.reshape(
y = computacion_y_bienes_raices['US internet percentage'].values.reshape(
#
lin_reg = LinearRegression()
lin_reg.fit(X,y)
#
```

```
plt.scatter(X, y, color = "red")
plt.plot(X, lin_reg.predict(X), color = "blue")
plt.title("Internet vs inversion en propiedades de 3 familias")
plt.xlabel("Inversion en propiedades de 3 familias (USD)")
plt.ylabel("Poblacion con acceso a internet (%)")
plt.grid(color='gray', linestyle='-', linewidth=0.5)
plt.show()
```

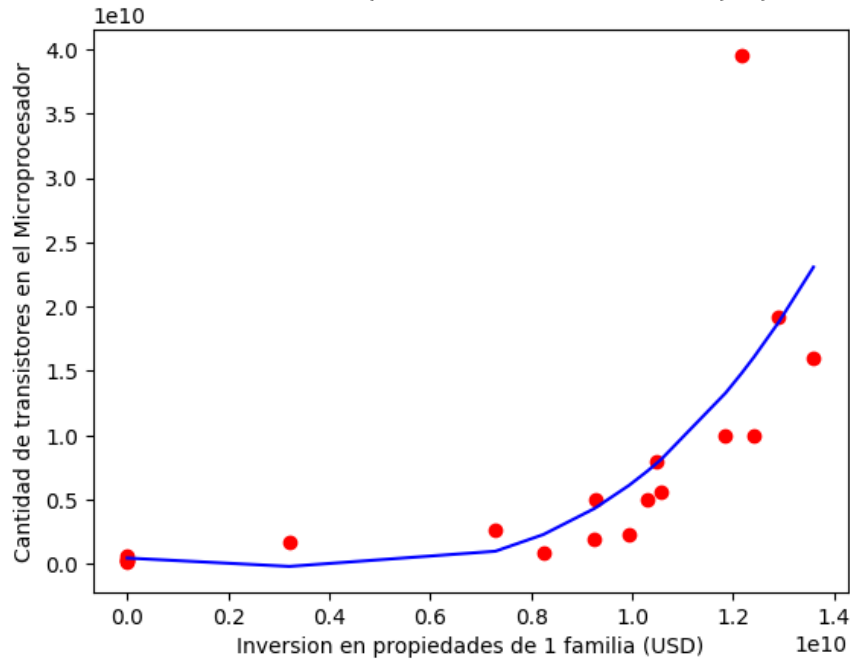


- Proyectando regresion polinomial en "Cantidad de transistores en el Microprocesador" con "Inversion en propiedades de 1 familia"

```
In [ ]: # Removemos este valor (solo es una fila) para tener un mejor grafico mas
computacion_y_bienes_raices = computacion_y_bienes_raices[computacion_y_b
# Ordenamos el eje X para evitar una prediccion polinomial horrible
computacion_y_bienes_raices = computacion_y_bienes_raices.sort_values(by=
#
y = computacion_y_bienes_raices['Microprocessor transistor count'].values
X = computacion_y_bienes_raices['re SINGLE FAMILY investment'].values.res
#
poly_reg=PolynomialFeatures(degree=3)
X_poly=poly_reg.fit_transform(X)
poly_reg.fit(X_poly,y)
lin_reg2=LinearRegression()
lin_reg2.fit(X_poly,y)
#
plt.scatter(X,y,color='red')
plt.plot(X,lin_reg2.predict(poly_reg.fit_transform(X)),color='blue')
plt.title('Cantidad de transistores en el microprocesador vs Inversion en
plt.xlabel('Inversion en propiedades de 1 familia (USD)')
```

```
plt.ylabel('Cantidad de transistores en el Microprocesador')
plt.show()
```

Cantidad de transistores en el microprocesador vs Inversion en propiedades de 1 familia



Hasta ahora hemos podido observar que segun el analisis de las correlaciones, las regresiones lineales y polinomicas, los avances en la computacion (por ejemplo en las GPU, microprocesadores e internet) han tenido un impacto en las ventas de propiedades de bienes raices.

Ahora veamos cuanto han impactado cada una de todos los avances aplicando una regresion lineal multiple y analizando los coeficientes de cada una de las columnas.

- Analizando cuanto fue el impacto en la INVERSION de bienes raices aplicando una regresion lineal multiple

```
In [ ]: df = computacion_y_bienes_raices
df = df.dropna()
#
X = mantener_columnas(
    df,
    df.columns[1:-1].tolist()
)
y = df['re investment'].values
#
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
regressor = LinearRegression()
regressor.fit(X_train, y_train)
df = df.drop(columns=['re investment', COLUMNA_FECHA], axis=1)
df = df.T
df = df.index
coeff_df = pd.DataFrame(regressor.coef_, df, columns=['Coefficient'])
coeff_df
```

Out[]:

	Coefficient
FGMOS transistor count	-2.224972e-02
FPGA transistor count	-6.263645e-02
GPU transistor count	8.760943e-03
Microprocessor transistor count	9.342333e-01
US internet percentage	1.255950e+09
re sales	2.285855e+05
re SINGLE FAMILY investment	4.920637e-01
re CONDO investment	-8.669829e+00
re TWO FAMILY investment	-5.435284e+01
re THREE FAMILY investment	4.441178e+01
re FOUR FAMILY investment	2.343419e+02

Podemos observar que por cada porcentaje de la poblacion con acceso a internet, la inversion en bienes raices aumenta 1.2×10^9 USD, esto hace que el internet sea el avance computacional mas importante para la inversion en bienes raices.

Hagamos el mismo analisis pero con la venta de bienes raices:

- Analizando el impacto en la VENTA de bienes raices aplicando una regresion lineal multiple

```
In [ ]: df = computacion_y_bienes_raices
df = df.dropna()
#
X = mantener_columnas(
    df,
    df.columns[1:-1].tolist()
)
y = df['re sales'].values
#
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
regressor = LinearRegression()
regressor.fit(X_train, y_train)
df = df.drop(columns=['re sales', COLUMNA_FECHA], axis=1)
df = df.T
df = df.index
coeff_df = pd.DataFrame(regressor.coef_, df, columns=['Coefficient'])
coeff_df
```

Out[]:

	Coefficient
FGMOS transistor count	-1.503254e-20
FPGA transistor count	8.963740e-17
GPU transistor count	2.921871e-17
Microprocessor transistor count	-7.050621e-17
US internet percentage	4.339115e-08
re SINGLE FAMILY investment	1.000000e+00
re CONDO investment	-1.716786e-16
re TWO FAMILY investment	-1.225563e-16
re THREE FAMILY investment	6.510148e-17
re FOUR FAMILY investment	-7.962270e-15
re investment	1.723037e-14

Nuevamente, podemos ver que la tecnologia con el mejor coeficiente es internet.

Conclusion

En esta investigacion hemos podido observar que los avances en la computacion y los bienes raices tienen una buena correlacion, hemos podido proyectar regresiones lineales, polinomicas y calcular mediante una regresion lineal multiple cual es el coeficiente con el cual impacta cada avance computacional en las ventas e inversion de bienes raices.

Observamos que el avance en Microprocesadores y GPUs son las que tienen las mejores correlaciones, y que el internet fue el que mas impacto tuvo en las ventas e inversion de bienes raices, esto tiene mucho sentido ya que el internet provoco que se facilite la busqueda, comunicacion entre partes y finalmente la adquisicion de propiedades.

Pero sin embargo se debe tener en cuenta que la industria de bienes raices fue altamente afectada por la burbuja del 2008, por lo que se debe continuar investigando en otras industrias (como la de los libros) para poder llegar a una conclusion mas solida.