Grupo 4: S4- CASO PRACTICO-BDD FINAL

Integrantes:

- Ronny Benitez
- Carlos Navas
- Felix Jimenez
- Joceline Salinas

Caso práctico

Modelar a su preferencia (pero justificado datos de la cadena de supermercados de EEUU, Wallmart.

La base con la que trabajaremos este caso práctico contiene información sobre datos históricos de las ventas de Wallmart desde 2010-02-05 hasta 2012-11-01, en el archivo WalmartStoresales. Dentro de este archivo encontrará los siguientes campos:

- Tienda-el número de la tienda.
- Fecha-la semana de ventas Weekly_Sales - ventas para la tienda dada.
- Holiday_Flag: si la semana es una semana especial de vacaciones

  -1 – Semana de vacaciones -0 – Semana no festiva.
- Temperatura - Temperatura el día de la venta.
- Fuel_price -costo del combustible en la región.
- IPC-índice de precios al consumidor vigente.
- Desempleo - tasa de desempleo predominante.
- Eventos festivos.
  - Super bowl: 12 de febrero de 2010, 11 de febrero de 2011, 10 de febrero de 2012, 8 de febrero de 2013
  - Día del Trabajo: 10-sep-10, 9-sep-11, 7-sep-12, 6-sep-13
  - Acción de Gracias: 26-nov-10, 25-nov-11, 23-nov-12, 29-nov-13
  - Navidad: 31-dic-10, 30-dic-11, 28-dic-12, 27-dic-13

In [241…

```
#Librerias EDA
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm

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

In [242…
```
from scipy import stats
```

1. Importe la base de datos a una base en Jupyter Notebook con pandas.

In [243…
```
#Damos permisoaGoogleDriveparaleer nuestrabase dedatos.
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call driv
e.mount("/content/drive", force_remount=True).
```

In [244…
```
#Importamosnuestrabasededatosconel nombrede "df"
filename='/content/drive/MyDrive/Analitica predictiva/Walmart.csv'
datos=pd.read_csv(filename)
datos
```

Out[244]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
0	1	05-02-2010	1643690.90	0	42.31	2.57	211.10	8.11
1	1	12-02-2010	1641957.44	1	38.51	2.55	211.24	8.11
2	1	19-02-2010	1611968.17	0	39.93	2.51	211.29	8.11
3	1	26-02-2010	1409727.59	0	46.63	2.56	211.32	8.11
4	1	05-03-2010	1554806.68	0	46.50	2.62	211.35	8.11
...	...	...	...	...	...	...	...	...
6430	45	28-09-2012	713173.95	0	64.88	4.00	192.01	8.68
6431	45	05-10-2012	733455.07	0	64.89	3.98	192.17	8.67
6432	45	12-10-2012	734464.36	0	54.47	4.00	192.33	8.67
6433	45	19-10-2012	718125.53	0	56.47	3.97	192.33	8.67
6434	45	26-10-2012	760281.43	0	58.85	3.88	192.31	8.67

6435 rows × 8 columns

In [245…
```
datos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Store         6435 non-null   int64  
 1   Date          6435 non-null   object 
 2   Weekly_Sales  6435 non-null   float64
 3   Holiday_Flag  6435 non-null   int64  
 4   Temperature   6435 non-null   float64
 5   Fuel_Price    6435 non-null   float64
 6   CPI           6435 non-null   float64
 7   Unemployment  6435 non-null   float64
dtypes: float64(5), int64(2), object(1)
memory usage: 402.3+ KB
```

Nuestro base de datos contiene 6435 registros no nulos y los tipos de datos incluyen enteros ('int64') para 'Store' y 'Holiday_Flag', y números de punto flotante ('float64') para las otras columnas. La columna 'Date' tiene un tipo de datos 'object', lo que normalmente significa que contiene datos de tipo cadena o categoría. Si 'Date' representa fechas, puede ser útil transformarla a un tipo de datos datetime para facilitar el manejo de actividades relacionadas con fechas.

Uilizamos el siguiente comando para una mejor visualziación de los datos de tipo float64

In [246…
```
pd.options.display.float_format = '{:.2f}'.format
```

Recodificar el nombre de las columnas a español y dejamos en minúsculas para un mejor procesamiento del código

In [247…
```
datos.rename({'Store':'Tienda','Weekly_Sales':'Ventas_semanales','Holiday_Flag':'Va
```

In [248…
```
datos.columns = datos.columns.str.lower()
```

In [249…
```
datos.head()
```

Out[249]:

	tienda	date	ventas_semanales	vacaciones	temperatura	precio_combustible	cpi	desemple
0	1	05-02-2010	1643690.90	0	42.31	2.57	211.10	8.
1	1	12-02-2010	1641957.44	1	38.51	2.55	211.24	8.
2	1	19-02-2010	1611968.17	0	39.93	2.51	211.29	8.
3	1	26-02-2010	1409727.59	0	46.63	2.56	211.32	8.
4	1	05-03-2010	1554806.68	0	46.50	2.62	211.35	8.

In [250…

```
datos.columns
```

Out[250]:

```
Index(['tienda', 'date', 'ventas_semanales', 'vacaciones', 'temperatura',
       'precio_combustible', 'cpi', 'desempleo'],
      dtype='object')
```

In [251…

```
## Corregimos el formato de fecha
datos['date'] = pd.to_datetime(datos['date'], format="%d-%m-%Y")
datos
```

Out[251]:

	tienda	date	ventas_semanales	vacaciones	temperatura	precio_combustible	cpi	dese
0	1	2010-02-05	1643690.90	0	42.31	2.57	211.10	
1	1	2010-02-12	1641957.44	1	38.51	2.55	211.24	
2	1	2010-02-19	1611968.17	0	39.93	2.51	211.29	
3	1	2010-02-26	1409727.59	0	46.63	2.56	211.32	
4	1	2010-03-05	1554806.68	0	46.50	2.62	211.35	
...	...	...	...	...	...	...	...	
6430	45	2012-09-28	713173.95	0	64.88	4.00	192.01	
6431	45	2012-10-05	733455.07	0	64.89	3.98	192.17	
6432	45	2012-10-12	734464.36	0	54.47	4.00	192.33	
6433	45	2012-10-19	718125.53	0	56.47	3.97	192.33	
6434	45	2012-10-26	760281.43	0	58.85	3.88	192.31	

6435 rows × 8 columns

Al tratarse de una data que contiene ventas y fechas, creemos importante crear en nuestro data set variables como , año, mes, semana y día que nos servirán para realizar los análisis descriptivos.

In [252…

```
import calendar

datos = datos.assign(
    anio=datos['date'].dt.year,

  mes=datos['date'].dt.month,
    mes_nombre=datos['date'].dt.month_name(),
    semana=datos['date'].dt.isocalendar().week,
    dia_semana=datos['date'].dt.day_name()
)
datos
```

Out[252]:

	tienda	date	ventas_semanales	vacaciones	temperatura	precio_combustible	cpi	dese
0	1	2010-02-05	1643690.90	0	42.31	2.57	211.10	
1	1	2010-02-12	1641957.44	1	38.51	2.55	211.24	
2	1	2010-02-19	1611968.17	0	39.93	2.51	211.29	
3	1	2010-02-26	1409727.59	0	46.63	2.56	211.32	
4	1	2010-03-05	1554806.68	0	46.50	2.62	211.35	
...	...	...	...	...	...	...	...	...
6430	45	2012-09-28	713173.95	0	64.88	4.00	192.01	
6431	45	2012-10-05	733455.07	0	64.89	3.98	192.17	
6432	45	2012-10-12	734464.36	0	54.47	4.00	192.33	
6433	45	2012-10-19	718125.53	0	56.47	3.97	192.33	
6434	45	2012-10-26	760281.43	0	58.85	3.88	192.31	

6435 rows × 13 columns

2. Obtenga los descriptivos resumen de la base de datos e identifique a las variables numéricas y categóricas. ¿Hay algo que le llame la atención?

In [253...
```
datos['vacaciones'] = datos['vacaciones'].astype(str)
datos['tienda'] = datos['tienda'].astype(str)
```

In [254...
```
tipos_variables = datos.dtypes

# Identificar variables numéricas
variables_numericas = tipos_variables[tipos_variables != 'object'].index.tolist()

# Identificar variables categóricas
variables_categoricas = tipos_variables[tipos_variables == 'object'].index.tolist()

# Mostrar las variables numéricas y categóricas
print("Variables numéricas:", variables_numericas)
print("Variables categóricas:", variables_categoricas)
```

```
Variables numéricas: ['date', 'ventas_semanales', 'temperatura', 'precio_combustib
le', 'cpi', 'desempleo', 'anio', 'mes', 'semana']
Variables categóricas: ['tienda', 'vacaciones', 'mes_nombre', 'dia_semana']
```

In [255...
```
descriptivos = datos.describe()
print(descriptivos)
```

```
       ventas_semanales  temperatura  precio_combustible     cpi  desempleo  \
count           6435.00      6435.00             6435.00 6435.00    6435.00
mean         1046964.88        60.66                3.36  171.58       8.00
std           564366.62        18.44                0.46   39.36       1.88
min           209986.25        -2.06                2.47  126.06       3.88
25%           553350.10        47.46                2.93  131.74       6.89
50%           960746.04        62.67                3.44  182.62       7.87
75%          1420158.66        74.94                3.73  212.74       8.62
max          3818686.45       100.14                4.47  227.23      14.31

         anio     mes  semana
count 6435.00 6435.00 6435.00
mean  2010.97    6.45   25.82
std      0.80    3.24   14.13
min   2010.00    1.00    1.00
25%   2010.00    4.00   14.00
50%   2011.00    6.00   26.00
75%   2012.00    9.00   38.00
max   2012.00   12.00   52.00
```

Observaciones

1. **Número de tiendas:** Hay un total de 45 tiendas, esta variable es categoríca y muestra el ID de cada tienda.

2. **Ventas semanales:** La media de ventas semanales es de aproximadamente 1,046,964.88 ,con una desviación estándar de alrededor de 564,366.62 Esto indica una variabilidad considerable en las ventas semanales entre las tiendas.

3. **Vacaciones:** Esta variable es categórica binaria, y muestra 1 si la semana tiene días festivos y 0 si no.

4. **Temperatura:** La temperatura promedio es de aproximadamente 60.66 grados, con una desviación estándar de alrededor de 18.44 grados. Esto muestra una variabilidad significativa en las condiciones de temperatura entre las tiendas y las semanas.

5. **Precio del combustible:** El precio promedio del combustible es de aproximadamente 3.36 dólares por galón, con una desviación estándar de alrededor de 0.46 dólares por galón. Esto sugiere que el precio del combustible también varía considerablemente entre las tiendas y las semanas.

6. **Índice de Precios al Consumidor (IPC) y Desempleo:** El IPC promedio es de aproximadamente 171.58 y la tasa de desempleo promedio es del 8.00%. Ambos indicadores económicos tienen una desviación estándar significativa, lo que sugiere variabilidad en las condiciones económicas en el área de las tiendas.

3. Evalúe si la base contiene datos perdidos y duplicados.

In [256…

```
datos_perdidos = datos.isnull().sum()
# Mostrar la cantidad de datos perdidos por columna
print(datos_perdidos)
```

```
tienda                0
date                  0
ventas_semanales      0
vacaciones            0
temperatura           0
precio_combustible    0
cpi                   0
desempleo             0
anio                  0
mes                   0
mes_nombre            0
semana                0
dia_semana            0
dtype: int64
```

In [257…
```
### Verificacion de duplicados
datos.duplicated().sum()
```

Out[257]:
```
0
```

Nuestra data no contiene datos nulos o duplicados

4. Evalúe si alguna de las variables contiene datos atípicos (outliers)

5. Grafique las distribuciones de las variables y a priori comente sobre ellas.

Como primer paso procederemos a visualizar las variables de manera independiente. En general, la visualización de las variables es un paso importante antes de la aplicación de la herramienta de regresión.

In [258…
```
var_cuantitativas = datos.select_dtypes('number').columns
var_cualitativas  =datos.select_dtypes('object').columns
```

In [259…
```
datos.groupby('date').count()
```

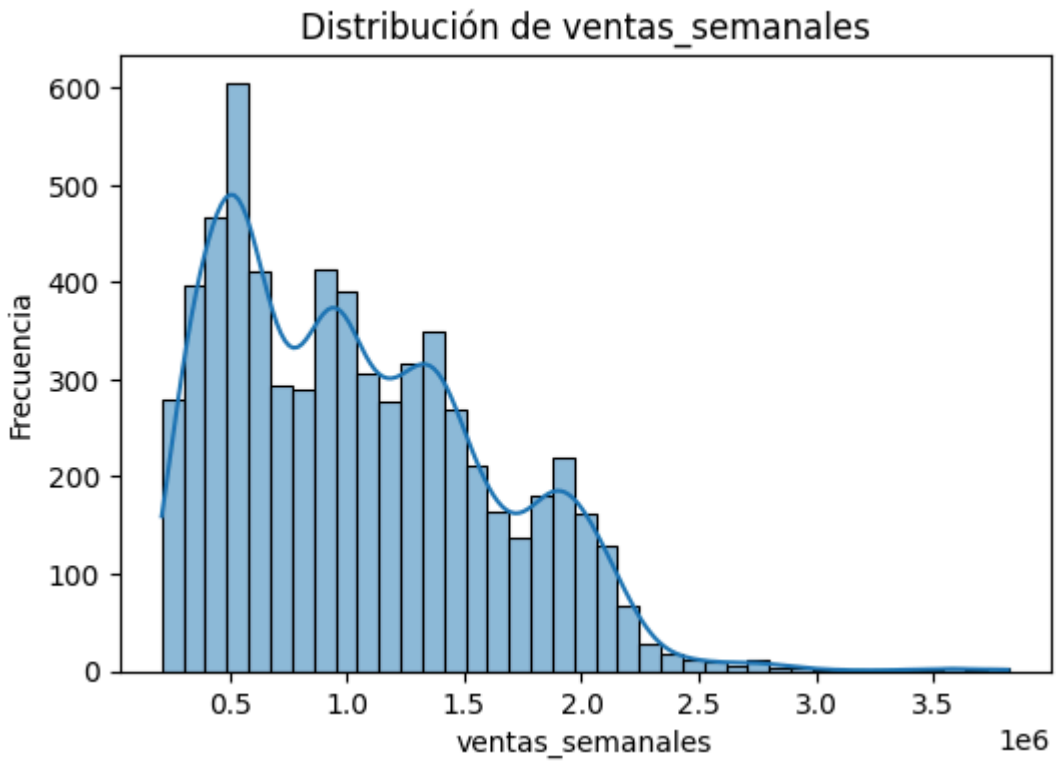
Out[259]:

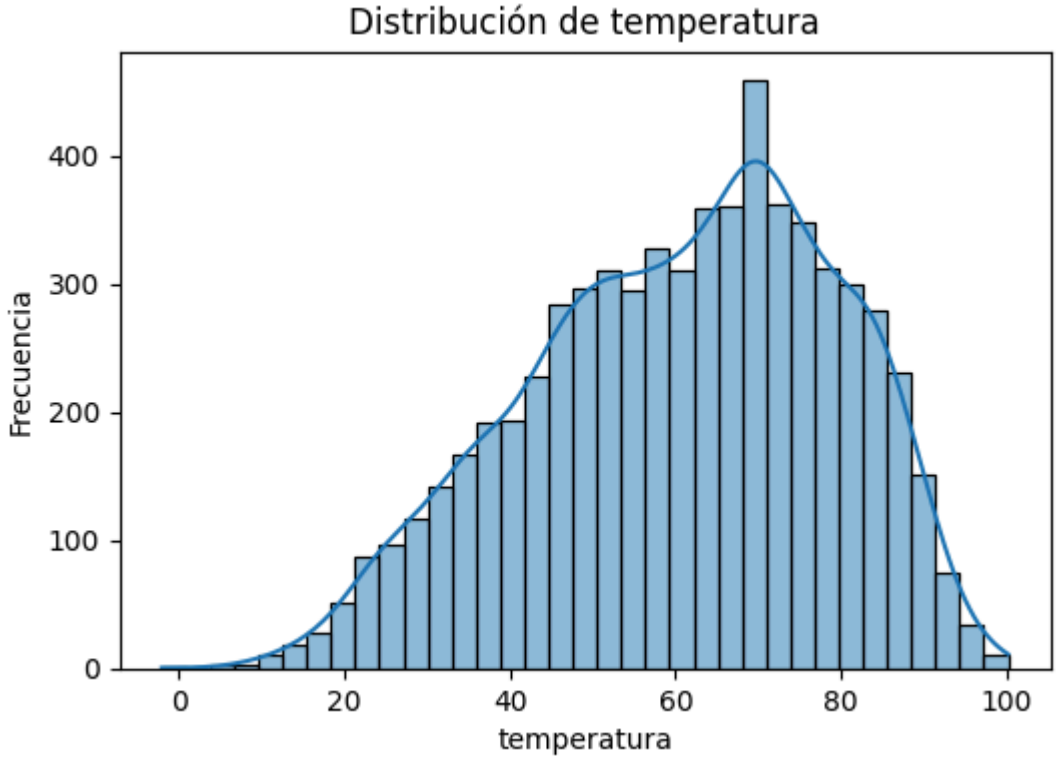
date	tienda	ventas_semanales	vacaciones	temperatura	precio_combustible	cpi	desempleo	a
2010-02-05	45	45	45	45	45	45	45	
2010-02-12	45	45	45	45	45	45	45	
2010-02-19	45	45	45	45	45	45	45	
2010-02-26	45	45	45	45	45	45	45	
2010-03-05	45	45	45	45	45	45	45	
...	...	...	...	...	...	...	...	
2012-09-28	45	45	45	45	45	45	45	
2012-10-05	45	45	45	45	45	45	45	
2012-10-12	45	45	45	45	45	45	45	
2012-10-19	45	45	45	45	45	45	45	
2012-10-26	45	45	45	45	45	45	45	

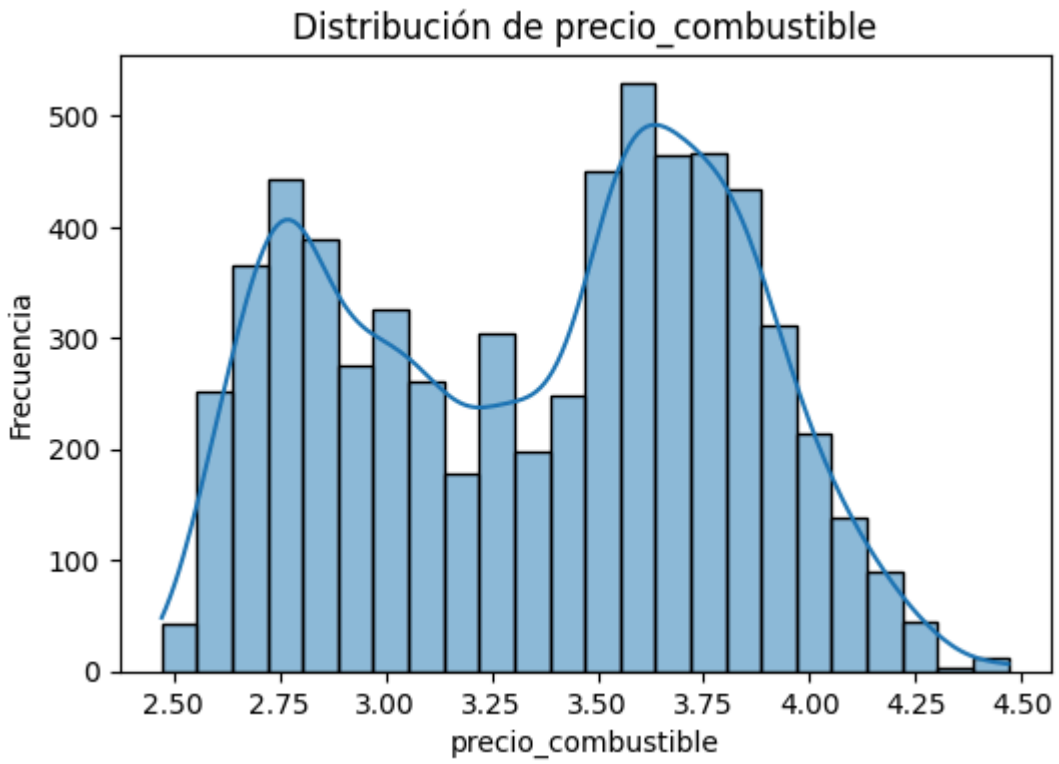
143 rows × 12 columns

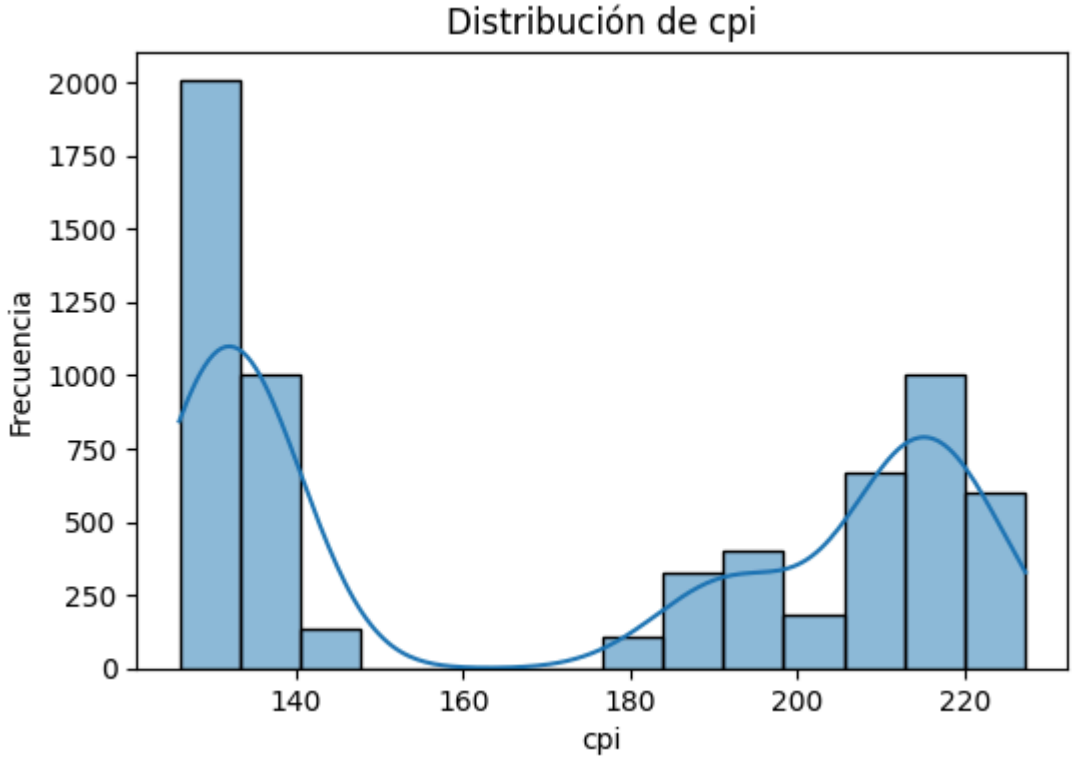
In [260…

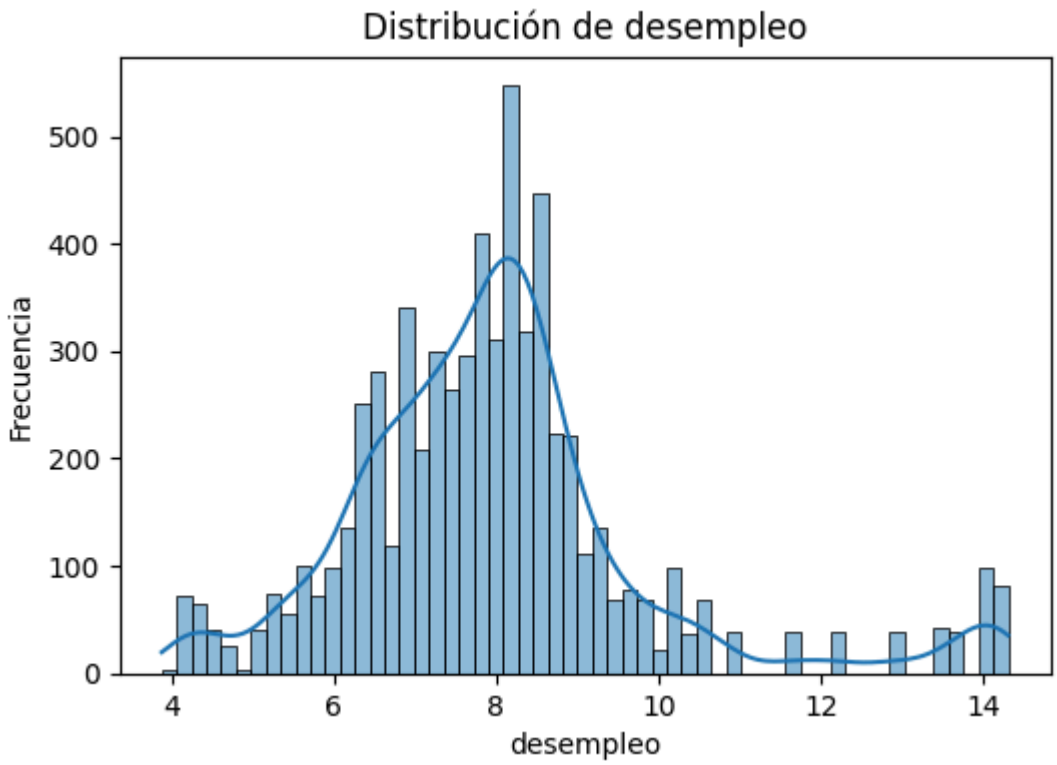
```
columnas_num = ['ventas_semanales', 'temperatura', 'precio_combustible', 'cpi', 'de

# Graficar histogramas y gráficos de densidad para cada variable
for col in columnas_num:
    plt.figure(figsize=(6, 4))
    sns.histplot(data=datos, x=col, kde=True)
    plt.title('Distribución de {}'.format(col))
    plt.xlabel(col)
    plt.ylabel('Frecuencia')
    plt.show()
```


Distribución de ventas_semanales
Frecuencia
0
100
200
300
400
500
600
0.5
1.0
1.5
2.0
2.5
3.0
3.5
ventas_semanales
1e6


Distribución de temperatura
Frecuencia
0
100
200
300
400
0
20
40
60
80
100
temperatura


Distribución de precio_combustible
Frecuencia
precio_combustible
0
100
200
300
400
500
2.50
2.75
3.00
3.25
3.50
3.75
4.00
4.25
4.50


Distribución de cpi
Frecuencia
cpi
0
250
500
750
1000
1250
1500
1750
2000
140
160
180
200
220



Conclusiones:

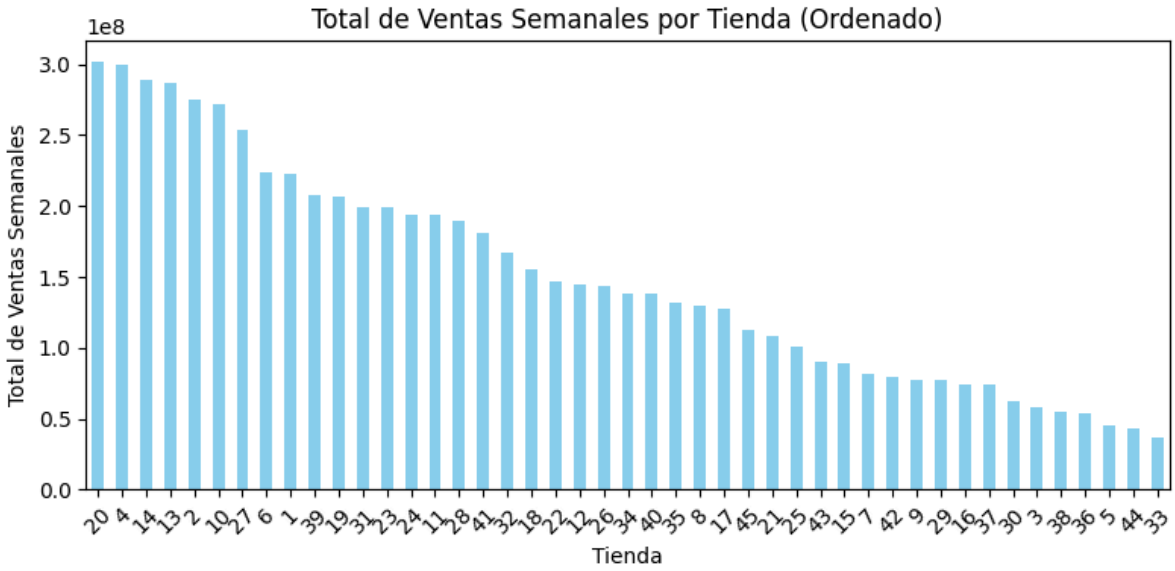
- Las ventas semanales tienen una distribución sesgada hacia la derecha.
- La temperatura y el desempleo tienen una distribución normal.
- El cpi (Índice de Precios al Consumidor) y el precio del combustible tienen una distribución bimodal esto significa que la distribución de los datos tiene dos modas o picos prominentes en lugar de uno solo. Esto sugiere que hay dos valores o rangos de valores que ocurren con mayor frecuencia en los datos.

Continuamos el análisis con nuestras variables categóricas

In [261…

```
import pandas as pd
import matplotlib.pyplot as plt


# y 'ventas_semanales' representa las ventas semanales
# y 'date' contiene las fechas
grptienda = datos.groupby('tienda')['ventas_semanales'].sum()

# Ordenar las tiendas por el total de ventas semanales de mayor a menor
grptienda_ordenado = grptienda.sort_values(ascending=False)

# Generar el gráfico de barras
plt.figure(figsize=(8, 4))
grptienda_ordenado.plot(kind='bar', color='skyblue')

# Personalizar el gráfico
plt.xlabel('Tienda')
plt.ylabel('Total de Ventas Semanales')
plt.title('Total de Ventas Semanales por Tienda (Ordenado)')

plt.xticks(rotation=45)  # Rotar las etiquetas del eje x para mejor legibilidad

plt.tight_layout()  # Ajustar el diseño del gráfico
```

```
# Mostrar el gráfico
plt.show()
```



Podemos observar que existe una variación importante entre las ventas de una tienda respecto de otras, la tienda con más ventas es la 20 y la que menos ventas tiene es la 33

In [262…

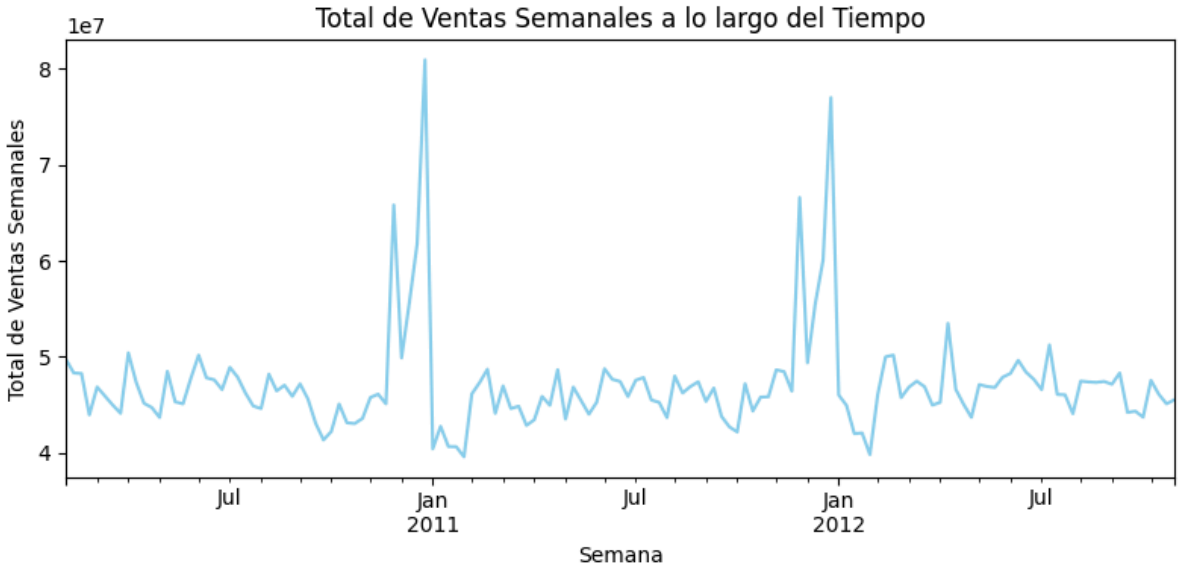
```
import pandas as pd


datos['date'] = pd.to_datetime(datos['date'])

# Agrupar las ventas por año-mes y calcular el total de ventas para cada grupo
ventas_por_año_mes = datos.groupby(datos['date'].dt.to_period('M'))['ventas_semanal

# Convertir la serie resultante a un DataFrame para mostrarlo como tabla
ventas_por_año_mes_df = ventas_por_año_mes.reset_index()
ventas_por_año_mes_df.columns = ['Año-Mes', 'Total de Ventas']

print(ventas_por_año_mes_df)
```

	Año-Mes	Total de Ventas
0	2010-02	190332983.04
1	2010-03	181919802.50
2	2010-04	231412368.05
3	2010-05	186710934.34
4	2010-06	192246172.36
5	2010-07	232580125.98
6	2010-08	187640110.89
7	2010-09	177267896.37
8	2010-10	217161824.02
9	2010-11	202853370.14
10	2010-12	288760532.72
11	2011-01	163703966.83
12	2011-02	186331327.87
13	2011-03	179356448.29
14	2011-04	226526510.97
15	2011-05	181648158.16
16	2011-06	189773385.19
17	2011-07	229911398.87
18	2011-08	188599332.25
19	2011-09	220847738.42
20	2011-10	183261283.15
21	2011-11	210162354.87
22	2011-12	288078102.48
23	2012-01	168894471.66
24	2012-02	192063579.54
25	2012-03	231509650.49
26	2012-04	188920905.95
27	2012-05	188766479.45
28	2012-06	240610329.29
29	2012-07	187509452.40
30	2012-08	236850765.68
31	2012-09	180645544.47
32	2012-10	184361680.42

Podemos observar que existen semanas de algunos meses que tienen más ventas, que se explica por días festivos, ejemplo son los meses de noviembre y diciembre, luego las ventas tienden a caer y estabilizarse.

In [263…

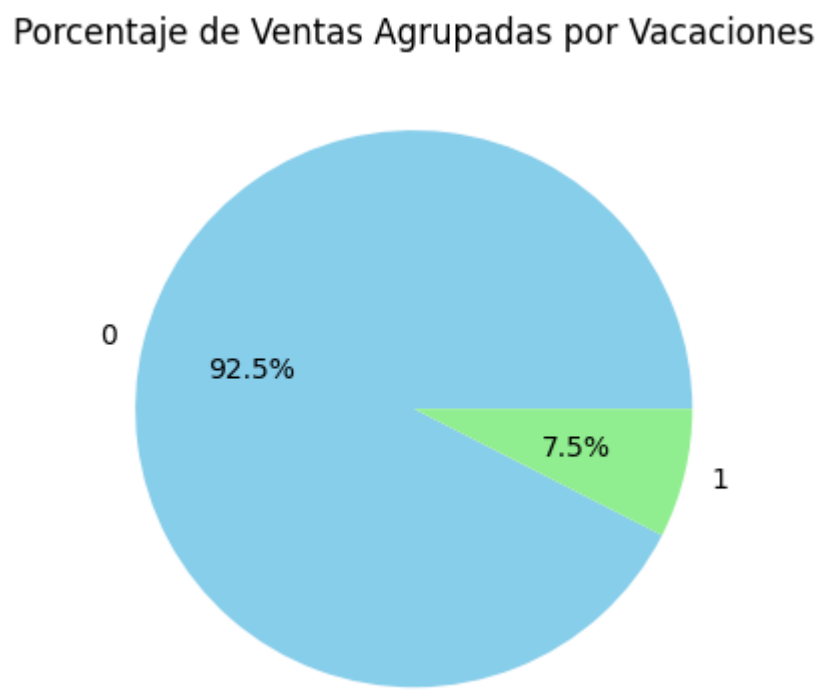
```
import pandas as pd
import matplotlib.pyplot as plt


# Convertir la columna de fechas a tipo datetime si no está en ese formato
datos['date'] = pd.to_datetime(datos['date'])

# Agrupar las ventas por semana y calcular la suma de ventas para cada semana
ventas_por_semana = datos.groupby(datos['date'].dt.to_period('W'))['ventas_semanale

# Generar el gráfico de líneas
plt.figure(figsize=(8, 4))
ventas_por_semana.plot(color='skyblue')

# Personalizar el gráfico
plt.xlabel('Semana')
plt.ylabel('Total de Ventas Semanales')
plt.title('Total de Ventas Semanales a lo largo del Tiempo')

plt.tight_layout()  # Ajustar el diseño del gráfico

# Mostrar el gráfico
plt.show()
```



De la misma forma observamos que existen picos semanales en días festivos, noviembre y diciembre, en donde podemos ver que se puede presentar cierta estacionalidad en los datos.

In [264…

```
import pandas as pd
import matplotlib.pyplot as plt


# y 'ventas_semanales' representa las ventas semanales
# y 'date' contiene las fechas
ventas_por_vacaciones = datos.groupby('vacaciones')['ventas_semanales'].sum()

# Generar el gráfico de pastel
plt.figure(figsize=(4, 4))
plt.pie(ventas_por_vacaciones, labels=ventas_por_vacaciones.index, autopct='%1.1f%%

# Personalizar el gráfico
plt.title('Porcentaje de Ventas Agrupadas por Vacaciones')

plt.tight_layout()  # Ajustar el diseño del gráfico

# Mostrar el gráfico
plt.show()
```



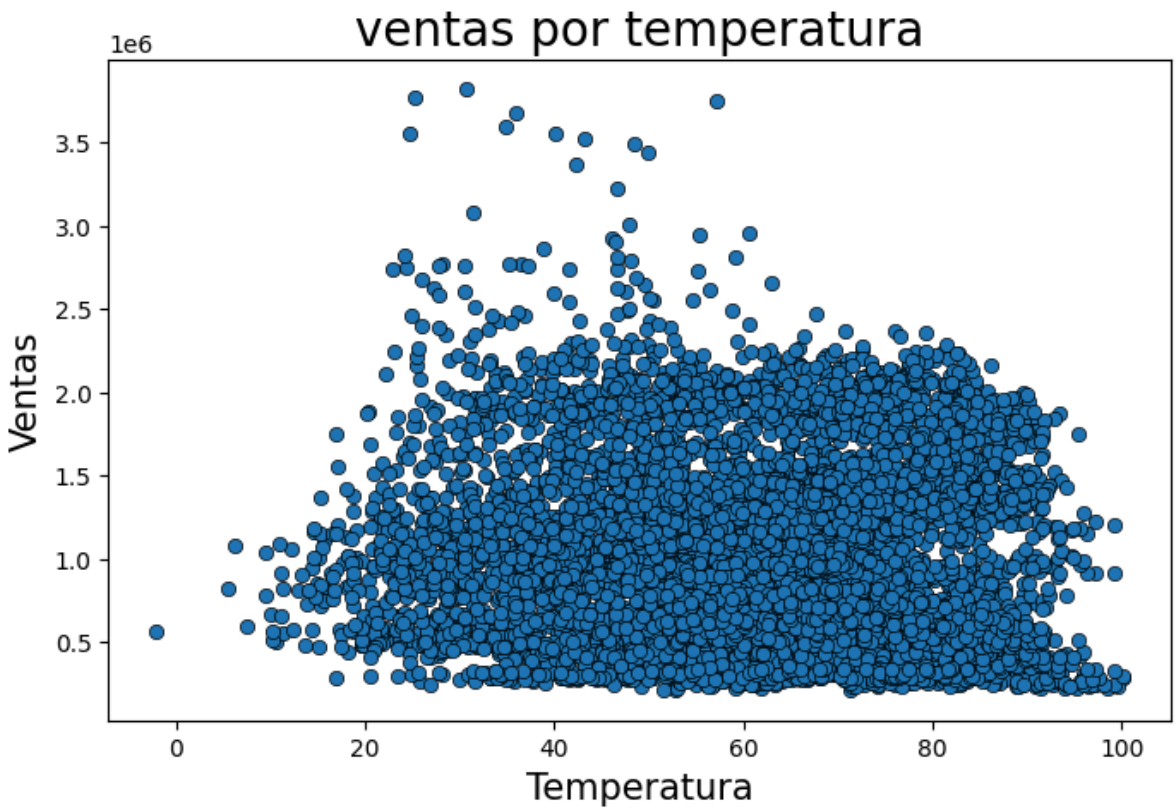
El monto de ventas durante semanas de vacacion representa el 7.5 % del total de las ventas

In [265…
```
import pandas as pd


# y 'ventas_semanales' representa las ventas semanales
# y 'date' contiene las fechas
# 'vacaciones' debe ser un tipo booleano o numérico donde 1 indica que hay vacacion

# Calcular los promedios de ventas para semanas con vacaciones y sin vacaciones
promedio_por_vacaciones = datos.groupby('vacaciones')['ventas_semanales'].mean()

# Mostrar los resultados
print("Promedio de ventas con vacaciones y sin vacaciones:")
print(promedio_por_vacaciones)
```

```
Promedio de ventas con vacaciones y sin vacaciones:
vacaciones
0   1041256.38
1   1122887.89
Name: ventas_semanales, dtype: float64
```

Analizando los promedios de ventas durante las semanas de no vacación vs las de si vacacion, podemos ver que durante los dias festivos se incrementan las ventas en relación a las otras semanas.

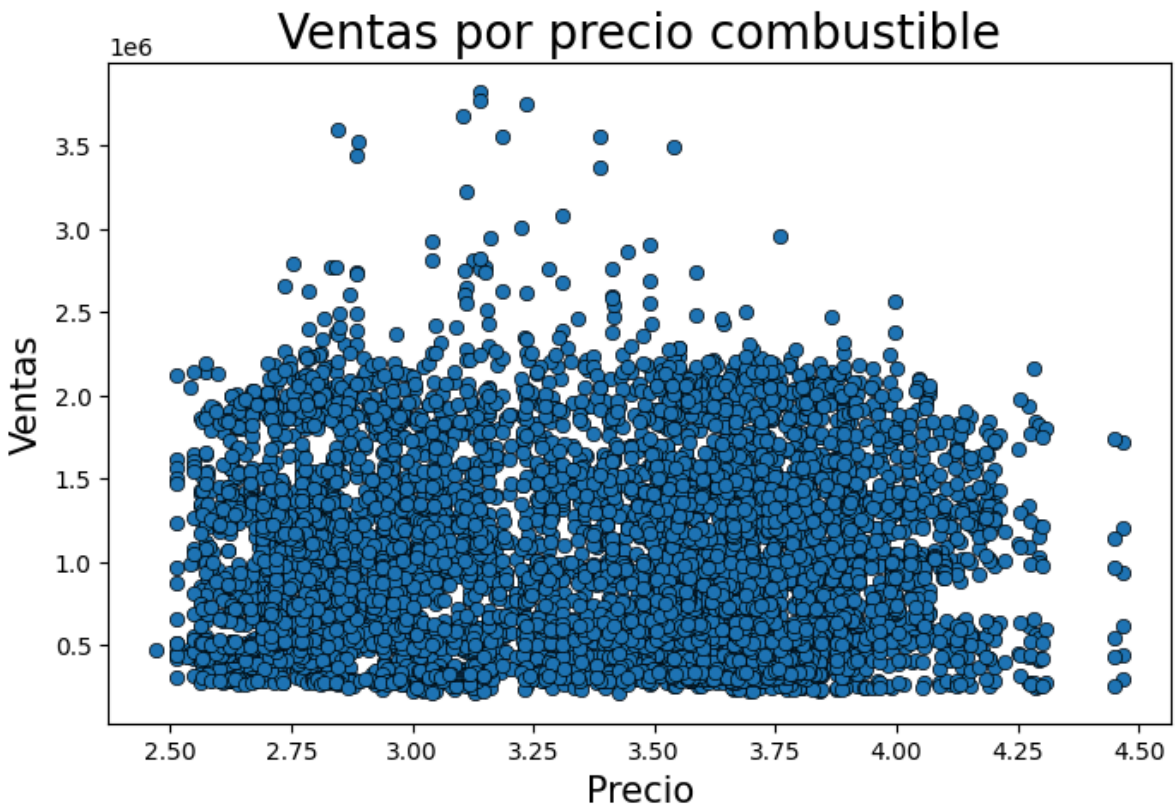
In [266…
```
plt.figure(figsize = (8, 5))
sns.scatterplot(data = datos,
                x = 'temperatura',
                y = 'ventas_semanales',
                edgecolor = "black")


plt.title('ventas por temperatura', size = 20)
plt.xlabel('Temperatura', size = 15)
plt.ylabel('Ventas', size = 15)
plt.show()
```



Se puede observar que las ventas no son afectadas por cambios de tenmperatura

In [267…

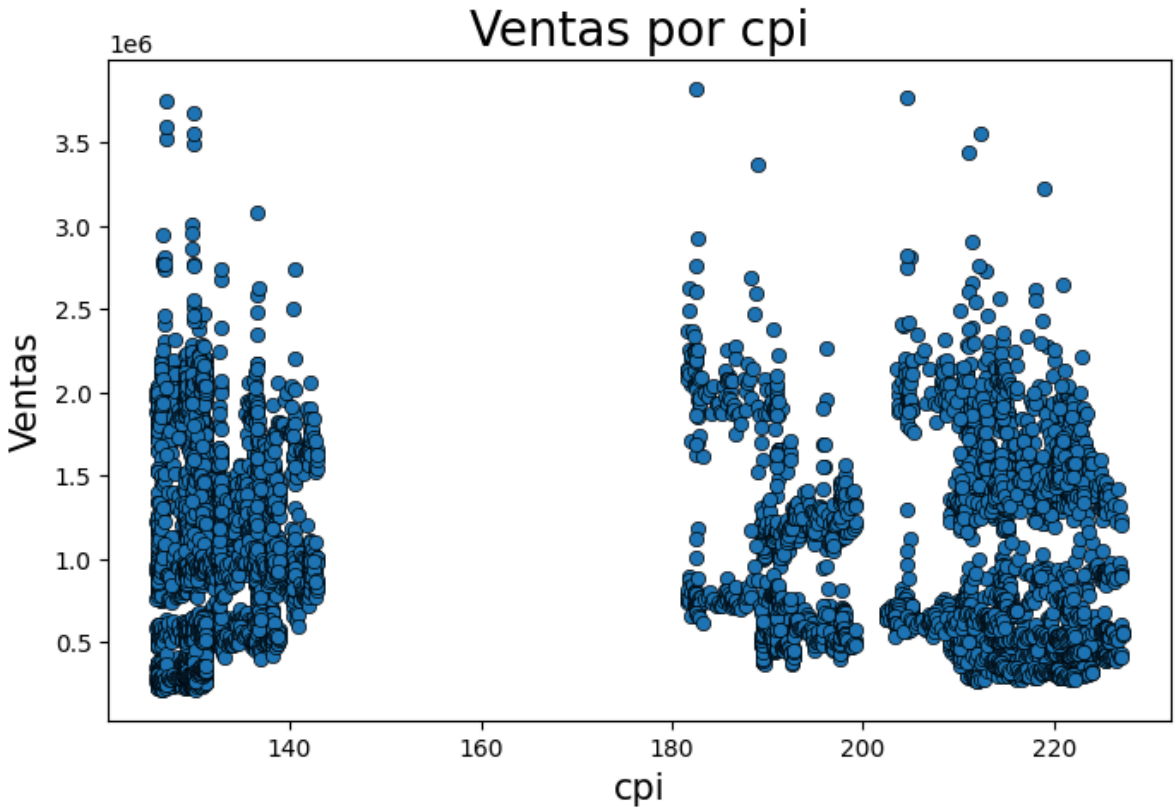
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 5))
sns.scatterplot(data=datos,
                x='precio_combustible',
                y='ventas_semanales',
                edgecolor="black")

plt.title('Ventas por precio combustible', size=20)
plt.xlabel('Precio', size=15)
plt.ylabel('Ventas', size=15)
plt.show()
```



Podemos observar que las ventas no son afectadas por el precio del combustible

In [268…

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 5))
sns.scatterplot(data=datos,
                x='cpi',
                y='ventas_semanales',
                edgecolor="black")

plt.title('Ventas por cpi', size=20)
plt.xlabel('cpi', size=15)
plt.ylabel('Ventas', size=15)
plt.show()
```



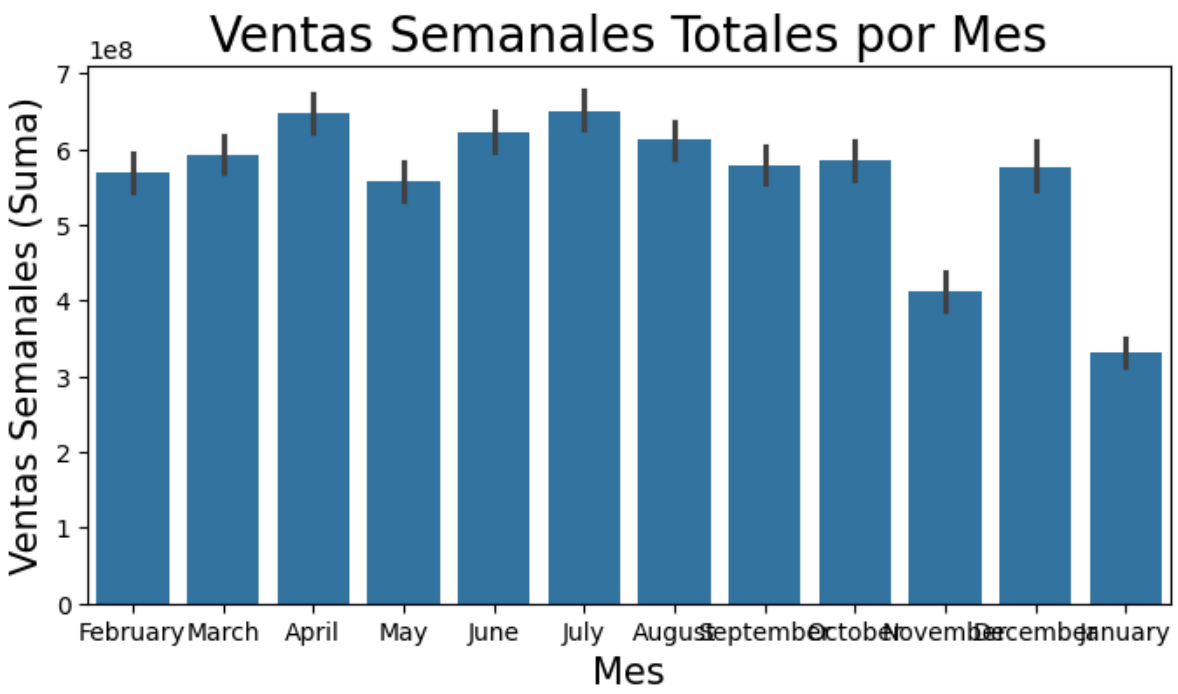
Podemos ver dos grupos, lo que nos da un indicio que el cpi tiene una leve aefctación en las ventas, entre 120 y 150 y otro grupo de 180 a 230

In [269…
```
datos.groupby('mes_nombre')['ventas_semanales'].sum().sort_values(ascending = False
```

Out[269]:
```
mes_nombre
July         650000977.25
April        646859784.97
June         622629886.84
August       613090208.82
March        592785901.28
October      584784787.59
September    578761179.26
December     576838635.20
February     568727890.45
May          557125571.95
November     413015725.01
January      332598438.49
Name: ventas_semanales, dtype: float64
```

In [270…
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Crea el gráfico de barras agrupado por mes_nombre
plt.figure(figsize=(8, 4))
sns.barplot(data=datos,
            x='mes_nombre',
            y='ventas_semanales',
            estimator=np.sum)

# Agrega etiquetas y título
plt.xlabel('Mes', size=15)
plt.ylabel('Ventas Semanales (Suma)', size=15)
plt.title('Ventas Semanales Totales por Mes', size=20)
```

```
# Muestra el gráfico
plt.show()
```



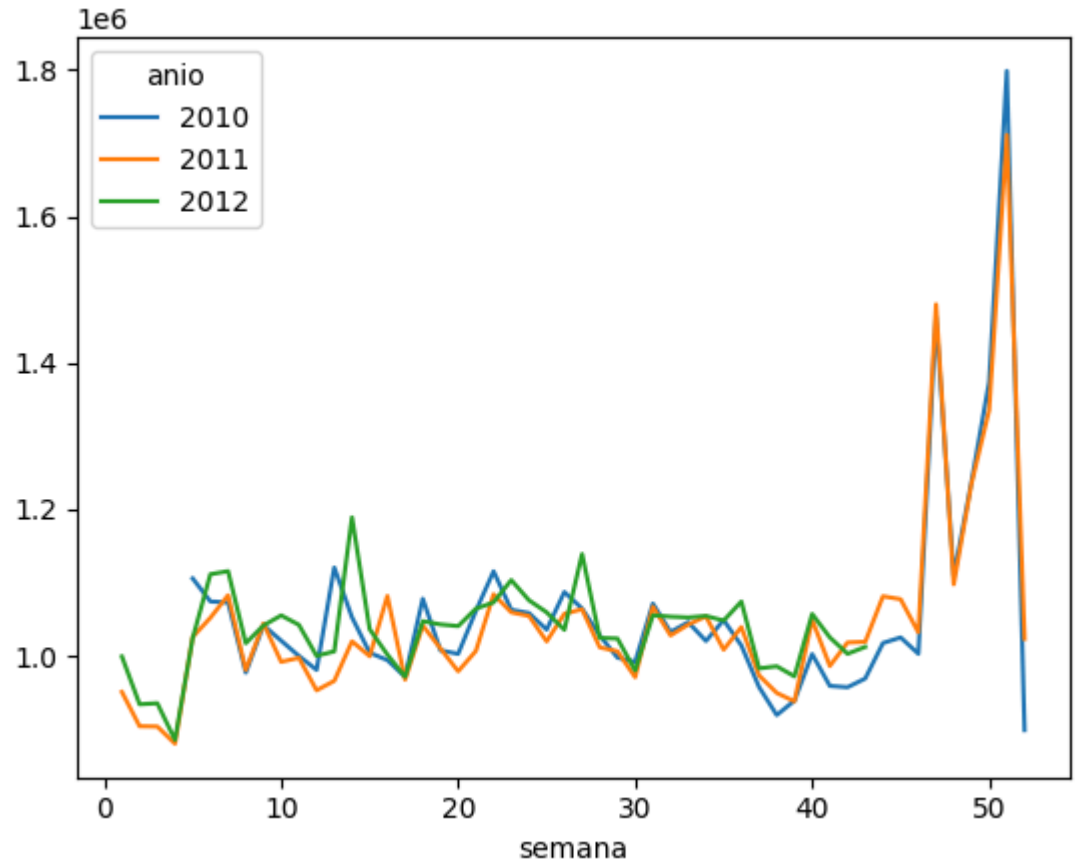
El mes con mayor ventas se presenta en Julio, con 650.000.977 USD, esto se puede explicar porque en el año 2012 la data solo esta hasta el mes de octubre, lo que quiere decir que el mes de julio tiene 3 años, y diciembre 2

In [271…

```
weekly_sales = pd.pivot_table(datos, values = "ventas_semanales", columns = "anio",
weekly_sales.plot()
```

Out[271]:

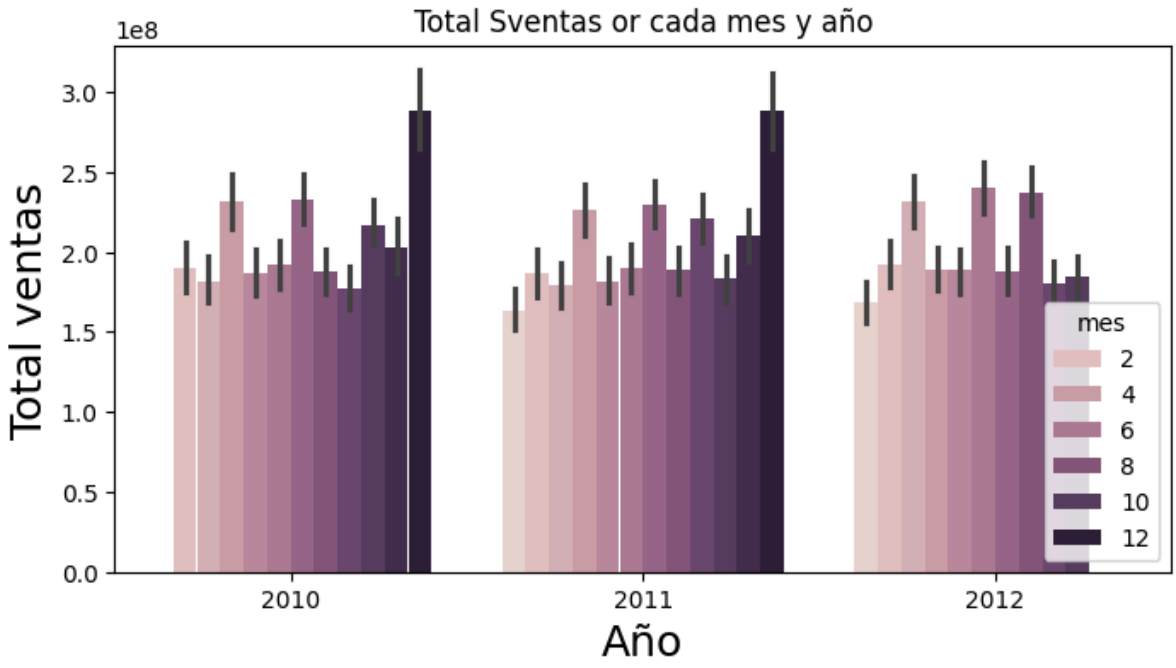
```
<Axes: xlabel='semana'>
```



Evaluando las semanas que tiene el año podemos ver que la semana 51 es la que más ventas presenta con un valor de 157.929.657 USD

In [272…

```
plt.figure(figsize = (8, 4))
sns.barplot(data = datos,
            x = 'anio',
            y = 'ventas_semanales',
            hue = 'mes',
            estimator = np.sum)

# Add labels and title
plt.title('Total Sventas or cada mes y año')
plt.xlabel('Año', size = 18)
plt.ylabel('Total ventas', size = 18)

plt.show()
```



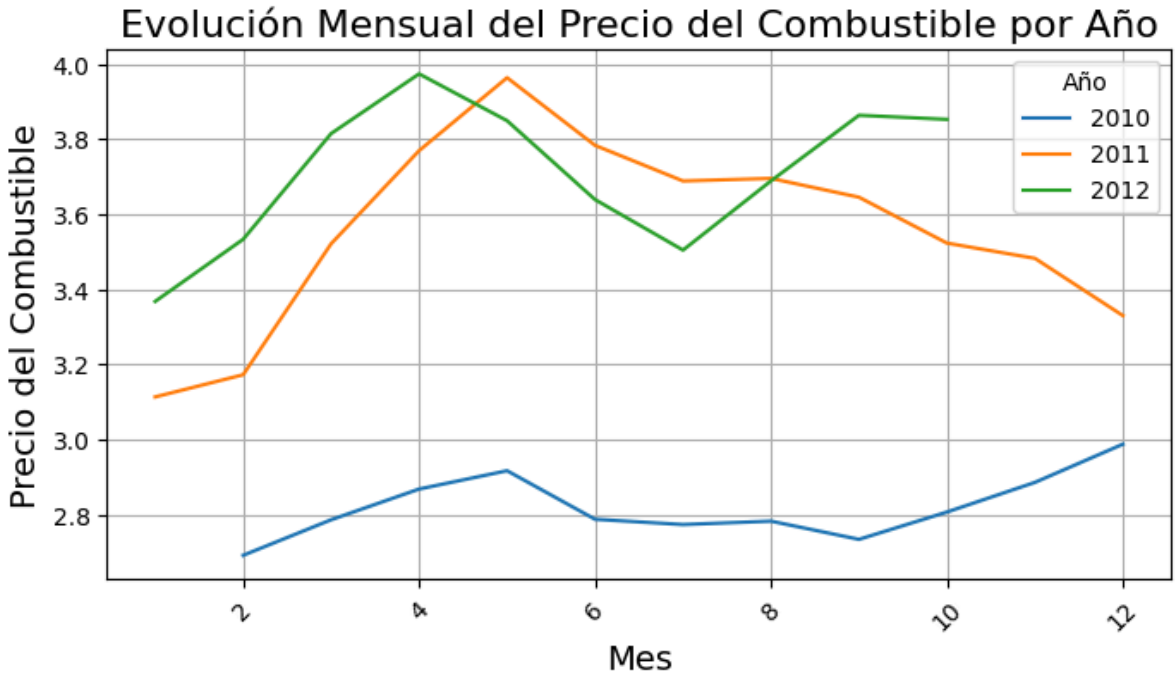
Las ventas totales en diciembre de 2010 y 2011 son las más altas en los tres años, donde:

En 2010, las ventas totales en diciembre fueron las más altas con 288,760,533. En 2011, las ventas totales en diciembre fueron las más altas con 288,760,533. En 2012, las ventas totales en junio fueron las más altas con 240,610,329$.

In [273…

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8  , 4))

# Agrupar los datos por año y mes y calcular el promedio del precio del combustible
precio_combustible_por_mes_y_anio = datos.groupby(['anio', 'mes'])['precio_combusti

# Trazar una línea separada para cada año
for anio in precio_combustible_por_mes_y_anio['anio'].unique():
    datos_anio = precio_combustible_por_mes_y_anio[precio_combustible_por_mes_y_ani
    sns.lineplot(data=datos_anio, x='mes', y='precio_combustible', label=anio)

plt.title('Evolución Mensual del Precio del Combustible por Año', size=16)
plt.xlabel('Mes', size=14)
plt.ylabel('Precio del Combustible', size=14)
```

```
plt.xticks(rotation=45)
plt.legend(title='Año')
plt.grid(True)

plt.show()
```



Podemos observar que el precio del combustible incrementa con el paso del tiempo

Tratamiento de los outliers

Para evaluar si alguna de las variables contiene datos atípicos (outliers) en base a los descriptivos resumen proporcionados, podemos utilizar el método de los valores atípicos basado en los estadísticos de Tukey.

Los estadísticos de Tukey se calculan utilizando el rango intercuartílico (IQR), que es la diferencia entre el percentil 75 (Q3) y el percentil 25 (Q1) de los datos. Generalmente, se considera que un valor es un outlier si está por debajo de Q1 - 1.5 *IQR o por encima de Q3 + 1.5* IQR.

In [274…
```
data_new = datos.copy()
```

Removemos las columnas que representan los mismos datos.

In [275…
```
data_new.drop(['date', 'anio', 'mes', 'dia_semana'], axis = 1, inplace = True)
```

In [276…
```
data_new.dtypes
```

Out[276]:
```
tienda                 object
ventas_semanales      float64
vacaciones             object
temperatura           float64
precio_combustible    float64
cpi                   float64
desempleo             float64
mes_nombre             object
semana                 UInt32
dtype: object
```

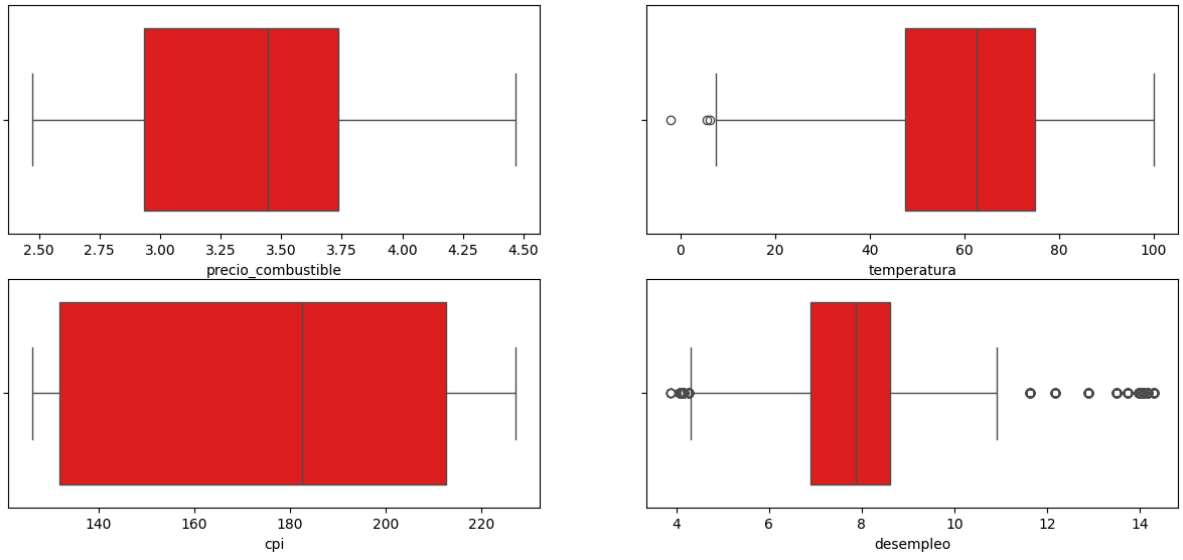
In [277…
```
data_new['tienda'] = data_new['tienda'].astype('object')
data_new['vacaciones'] = data_new['vacaciones'].astype('object')
data_new['semana'] = data_new['semana'].astype('object')
```

In [278…
```
data_new.dtypes
```

Out[278]:
```
tienda                 object
ventas_semanales      float64
vacaciones             object
temperatura           float64
precio_combustible    float64
cpi                   float64
desempleo             float64
mes_nombre             object
semana                 object
dtype: object
```

In [279…
```
cols = ['precio_combustible', 'temperatura', 'cpi', 'desempleo']
plt.figure(figsize=(15 ,10))
for i,col in enumerate(cols):
    print(i, col)
    plt.subplot(3,2,i+1)
    sns.boxplot(data_new, x = col, color = 'red')
plt.show()

print('Number of data rows: ', data_new.shape[0])
```

```
0 precio_combustible
1 temperatura
2 cpi
3 desempleo
```




```
Number of data rows:  6435
```

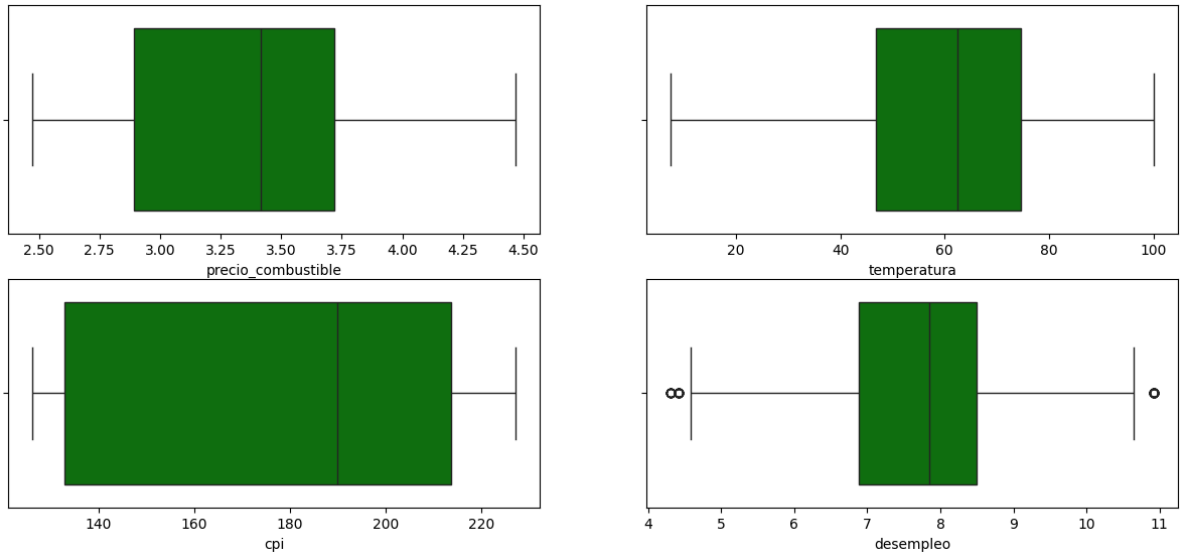
In [280…
```
# Calcular el rango intercuartílico (IQR) para precio_combustible
q1_precio_combustible = data_new['precio_combustible'].quantile(0.25)
q3_precio_combustible = data_new['precio_combustible'].quantile(0.75)
iqr_precio_combustible = q3_precio_combustible - q1_precio_combustible

# Calcular los límites para los valores atípicos de precio_combustible
lower_bound_precio_combustible = q1_precio_combustible - 1.5 * iqr_precio_combustib
upper_bound_precio_combustible = q3_precio_combustible + 1.5 * iqr_precio_combustib

# Eliminar los valores atípicos de precio_combustible
data_new = data_new[(data_new['precio_combustible'] >= lower_bound_precio_combustib

# Repetir para temperatura
```

```
q1_temperatura = data_new['temperatura'].quantile(0.25)
q3_temperatura = data_new['temperatura'].quantile(0.75)
iqr_temperatura = q3_temperatura - q1_temperatura
lower_bound_temperatura = q1_temperatura - 1.5 * iqr_temperatura
upper_bound_temperatura = q3_temperatura + 1.5 * iqr_temperatura
data_new = data_new[(data_new['temperatura'] >= lower_bound_temperatura) & (data_ne

# Repetir para cpi
q1_cpi = data_new['cpi'].quantile(0.25)
q3_cpi = data_new['cpi'].quantile(0.75)
iqr_cpi = q3_cpi - q1_cpi
lower_bound_cpi = q1_cpi - 1.5 * iqr_cpi
upper_bound_cpi = q3_cpi + 1.5 * iqr_cpi
data_new = data_new[(data_new['cpi'] >= lower_bound_cpi) & (data_new['cpi'] <= uppe

# Repetir para desempleo
q1_desempleo = data_new['desempleo'].quantile(0.25)
q3_desempleo = data_new['desempleo'].quantile(0.75)
iqr_desempleo = q3_desempleo - q1_desempleo
lower_bound_desempleo = q1_desempleo - 1.5 * iqr_desempleo
upper_bound_desempleo = q3_desempleo + 1.5 * iqr_desempleo
data_new = data_new[(data_new['desempleo'] >= lower_bound_desempleo) & (data_new['d
```

In [281…

```
cols = ['precio_combustible', 'temperatura', 'cpi', 'desempleo']
plt.figure(figsize=(15 ,10))
for i,col in enumerate(cols):
    print(i, col)
    plt.subplot(3,2,i+1)
    sns.boxplot(data_new, x = col, color = 'g')
plt.show()

print('Number of data rows: ', data_new.shape[0])
```

```
0 precio_combustible
1 temperatura
2 cpi
3 desempleo
```




```
Number of data rows:  5951
```

6. Obtenga las correlaciones entre los datos de corte numérico.

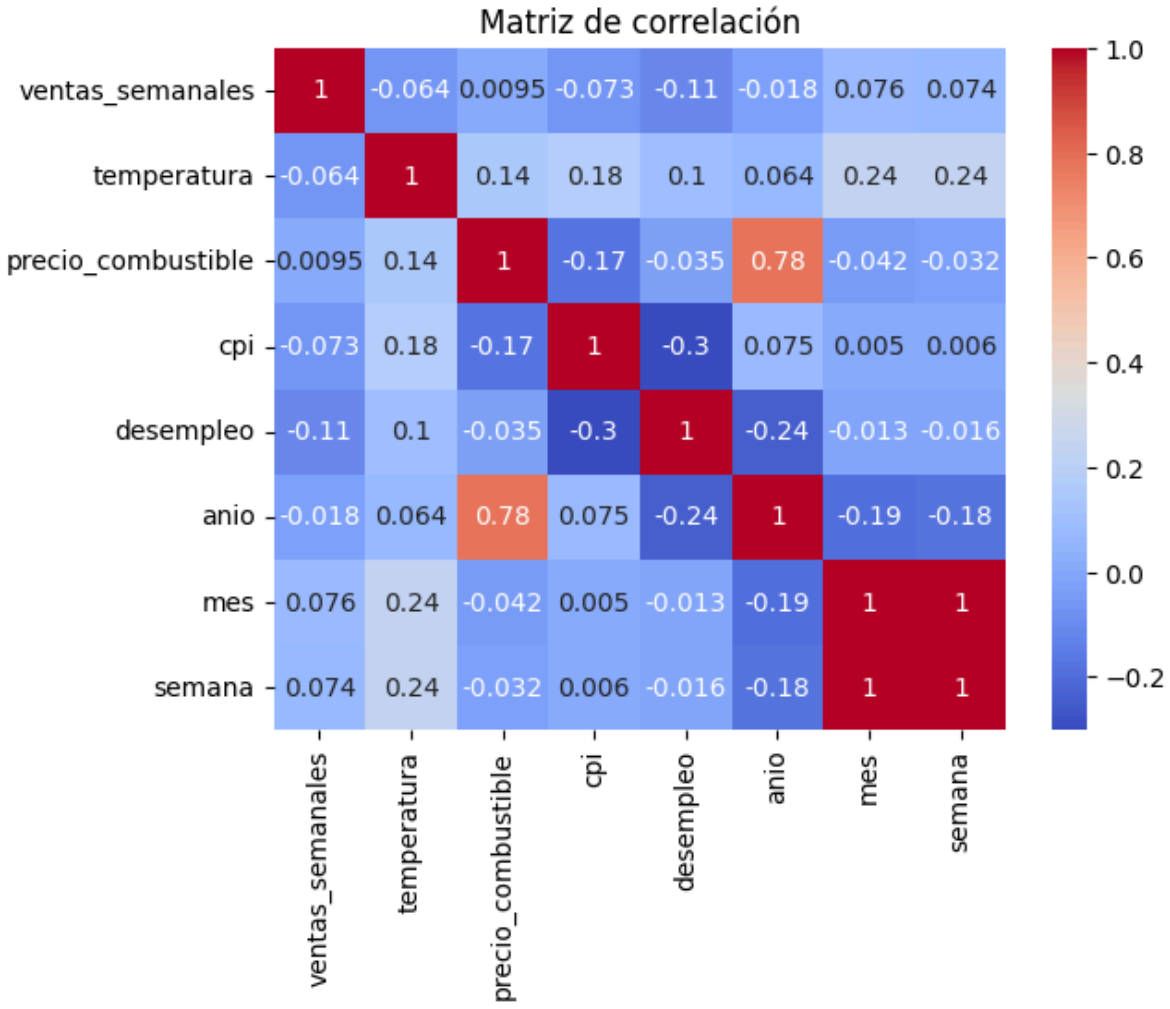
In [282…

```
variables_numericas = datos.select_dtypes(include='number')
correlaciones = variables_numericas.corr()

# Mostrar la matriz de correlación
print(correlaciones)
```

```
                     ventas_semanales  temperatura  precio_combustible   cpi  \
ventas_semanales                 1.00        -0.06                0.01 -0.07
temperatura                     -0.06         1.00                0.14  0.18
precio_combustible               0.01         0.14                1.00 -0.17
cpi                             -0.07         0.18               -0.17  1.00
desempleo                       -0.11         0.10               -0.03 -0.30
anio                            -0.02         0.06                0.78  0.07
mes                              0.08         0.24               -0.04  0.00
semana                           0.07         0.24               -0.03  0.01

                     desempleo  anio   mes  semana
ventas_semanales         -0.11 -0.02  0.08    0.07
temperatura               0.10  0.06  0.24    0.24
precio_combustible       -0.03  0.78 -0.04   -0.03
cpi                      -0.30  0.07  0.00    0.01
desempleo                 1.00 -0.24 -0.01   -0.02
anio                     -0.24  1.00 -0.19   -0.18
mes                      -0.01 -0.19  1.00    1.00
semana                   -0.02 -0.18  1.00    1.00
```

In [283…

```
# Crear mapa de calor de las correlaciones
sns.heatmap(correlaciones, annot=True, cmap='coolwarm')
plt.title("Matriz de correlación")
plt.show()
```



El mapa de calor proporciona una visualización más intuitiva y fácil de interpretar de las correlaciones entre las variables numéricas.

La correlación es una medida de la fuerza de la relación entre dos variables. Puede variar entre -1 y 1. Un valor de 1 indica una correlación positiva perfecta, lo que significa que a

medida que una variable aumenta, la otra también aumenta. Un valor de -1 indica una correlación negativa perfecta, lo que significa que a medida que una variable aumenta, la otra disminuye. Un valor de 0 indica que no hay correlación entre las dos variables.

Conclusiones

1. **Año vs Precio del combustible (0.78):** Esta es la correlación más fuerte en el mapa de calor. Indica una relación significativa entre el año y el precio del combustible, sugiriendo que el precio del combustible tiende a aumentar a lo largo de los años.

2. **Temperatura vs CPI (0.18):** Aunque no es tan fuerte como la correlación anterior, esta relación positiva moderada sugiere que los precios al consumidor tienden a aumentar con el aumento de la temperatura. Esto puede estar relacionado con factores estacionales o climáticos que afectan la demanda y los precios.

3. **CPI vs Desempleo (-0.30):** Esta correlación negativa moderada indica que los precios al consumidor tienden a disminuir cuando la tasa de desempleo aumenta. Esto puede ser resultado de la disminución de la demanda debido a la reducción del poder adquisitivo de los consumidores durante períodos de alto desempleo.

7. Comente que variable escogerán como variable dependiente y que variables introducirán a su modelo.

Para seleccionar la variable dependiente y las variables independientes para un modelo predictivo, es importante considerar el objetivo del análisis y las relaciones potenciales entre las variables. Aquí hay algunas consideraciones para seleccionar las variables:

1. **Variable Dependiente:** La variable que estamos tratando de predecir o explicar en el modelo. En este caso, podríamos seleccionar "ventas_semanales" como nuestra variable dependiente, ya que parece ser la variable principal de interés en el contexto de análisis de ventas.

2. **Variables Independientes:** Las variables que se utilizan para predecir o explicar la variable dependiente. Aquí hay algunas variables potenciales que podrían introducirse en el modelo como variables independientes:

    - **Vacaciones:** Puede tener un impacto en las ventas, ya que los períodos de vacaciones pueden influir en el comportamiento de compra de los consumidores.
    - **Temperatura:** Las condiciones climáticas pueden afectar la demanda de ciertos productos, especialmente en industrias como la moda o la alimentación.
    - **Precio del combustible:** Los cambios en el precio del combustible pueden influir en los costos de transporte y, por lo tanto, en los precios al consumidor y las decisiones de compra.
    - **CPI (Índice de Precios al Consumidor):** El CPI puede ser un indicador de la inflación y puede influir en el poder adquisitivo de los consumidores.
    - **Desempleo:** La tasa de desempleo puede afectar la confianza del consumidor y su capacidad para gastar.
    - **Año y Mes:** Pueden introducirse como variables categóricas para capturar posibles efectos estacionales o tendencias temporales en las ventas.

8. Indique que tipo de modelación realizarán y porqué.

El tipo de modelación a realizar dependerá de la naturaleza de los datos y del objetivo del análisis.Considerando que la variable dependiente es "Ventas_semanales", podemos sugerir un enfoque de modelación de regresión.

La regresión es adecuada cuando queremos analizar la relación entre una variable dependiente continua (en este caso, las ventas semanales) y varias variables independientes (como "Vacaciones", "Temperatura", "Precio_combustible", "CPI" y "Desempleo") que pueden influir en la variable dependiente. El objetivo es encontrar una función o modelo que describa la relación entre estas variables y permita predecir o estimar las ventas semanales en función de los valores de las variables independientes.

Un enfoque común en la regresión es utilizar la regresión lineal, que asume una relación lineal entre las variables independientes y la variable dependiente. Sin embargo, también se pueden explorar otros modelos de regresión más complejos, como la regresión polinómica, la regresión de árbol de decisiones o la regresión de máquinas de soporte vectorial, según las características y los patrones presentes en los datos.

9. Verifique los supuestos, de haber escogido el enfoque econométrico.

In [284…
```
import statsmodels.stats.api as sms
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.compat import lzip
```

In [285…
```
data_new['vacaciones'] = pd.to_numeric(data_new['vacaciones'], errors='coerce')

# Verificar los tipos de datos después de la conversión
print(data_new.dtypes)
```

```
tienda                 object
ventas_semanales      float64
vacaciones              int64
temperatura           float64
precio_combustible    float64
cpi                   float64
desempleo             float64
mes_nombre             object
semana                 object
dtype: object
```

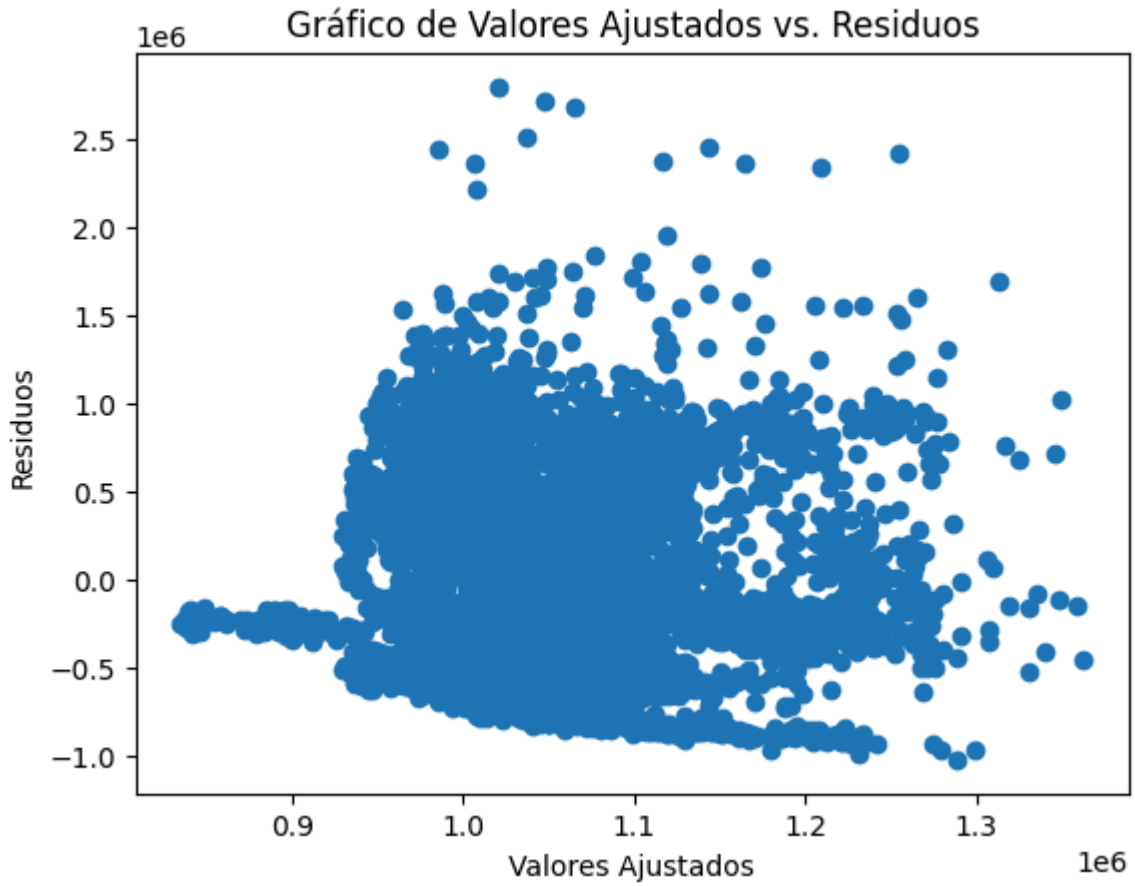
In [286…
```
regresion = ols("ventas_semanales ~ vacaciones + temperatura + precio_combustible +
results = regresion.fit()
```

In [287…
```
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       ventas_semanales   R-squared:                       0.019
Model:                           OLS   Adj. R-squared:                  0.018
Method:                Least Squares   F-statistic:                     23.38
Date:               Tue, 19 Mar 2024   Prob (F-statistic):           2.43e-23
Time:                       04:47:27   Log-Likelihood:                -87280.
No. Observations:               5951   AIC:                         1.746e+05
Df Residuals:                   5945   BIC:                         1.746e+05
Df Model:                          5
Covariance Type:           nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.
975]
--------------------------------------------------------------------------------
Intercept      1.741e+06   9.29e+04     18.741      0.000    1.56e+06    1.92
e+06
vacaciones     7.132e+04   2.92e+04      2.444      0.015    1.41e+04    1.29
e+05
temperatura     -970.8084    423.345     -2.293      0.022   -1800.719    -14
0.898
precio_combustible -8263.4382   1.67e+04     -0.493      0.622   -4.11e+04    2.46
e+04
cpi            -1503.4776    202.937     -7.409      0.000   -1901.307   -110
5.648
desempleo      -4.478e+04   6154.120     -7.276      0.000   -5.68e+04   -3.27
e+04
==============================================================================
Omnibus:                      351.973   Durbin-Watson:                   0.113
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              417.533
Skew:                           0.649   Prob(JB):                     2.16e-91
Kurtosis:                       2.983   Cond. No.                     2.41e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
[2] The condition number is large, 2.41e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [288…

```
X = data_new[['vacaciones', 'temperatura', 'precio_combustible', 'cpi', 'desempleo'
y = data_new['ventas_semanales']

# Añadir una constante al conjunto de variables independientes
X = sm.add_constant(X)

# Ajustar el modelo de regresión lineal
modelo = sm.OLS(y, X).fit()

# Obtener los residuos
residuos = modelo.resid

# Graficar los valores ajustados versus los residuos
plt.scatter(modelo.fittedvalues, residuos)
plt.xlabel('Valores Ajustados')
plt.ylabel('Residuos')
plt.title('Gráfico de Valores Ajustados vs. Residuos')
plt.show()
```



In [289…
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [290…
```
data_new
```

Out[290]:

	tienda	ventas_semanales	vacaciones	temperatura	precio_combustible	cpi	desempleo
0	1	1643690.90	0	42.31	2.57	211.10	8.11
1	1	1641957.44	1	38.51	2.55	211.24	8.11
2	1	1611968.17	0	39.93	2.51	211.29	8.11
3	1	1409727.59	0	46.63	2.56	211.32	8.11
4	1	1554806.68	0	46.50	2.62	211.35	8.11
…	…	…	…	…	…	…	…
6430	45	713173.95	0	64.88	4.00	192.01	8.68
6431	45	733455.07	0	64.89	3.98	192.17	8.67
6432	45	734464.36	0	54.47	4.00	192.33	8.67
6433	45	718125.53	0	56.47	3.97	192.33	8.67
6434	45	760281.43	0	58.85	3.88	192.31	8.67

5951 rows × 9 columns

In [291…
```
df2=data_new[data_new.columns.difference(['ventas_semanales','mes_nombre','semana',
df2
```

Out[291]:

	cpi	desempleo	precio_combustible	temperatura	vacaciones
0	211.10	8.11	2.57	42.31	0
1	211.24	8.11	2.55	38.51	1
2	211.29	8.11	2.51	39.93	0
3	211.32	8.11	2.56	46.63	0
4	211.35	8.11	2.62	46.50	0
...	...	...	...	...	...
6430	192.01	8.68	4.00	64.88	0
6431	192.17	8.67	3.98	64.89	0
6432	192.33	8.67	4.00	54.47	0
6433	192.33	8.67	3.97	56.47	0
6434	192.31	8.67	3.88	58.85	0

5951 rows × 5 columns

In [292…

```
# Creamos el dataframe del VIF
vif_data = pd.DataFrame()
vif_data["feature"] = df2.columns

# Calculamos el VIF por c/variable
vif_data["VIF"] = [variance_inflation_factor(df2.values, i) for i in range(len(df2.

print(vif_data)
```

```
              feature   VIF
0                 cpi 16.63
1          desempleo 23.46
2  precio_combustible 29.06
3         temperatura 13.19
4          vacaciones  1.10
```

Si el VIF es 1, no existe correlacion entre las variables analizadas

Si el VIF es >1 y <5, la correlación es moderada entre entre las variables analizadas, pero no constituye a menudo una transgresión grave para requerir atención

Si el VIF es >5, la correlación es potencialmente severa entre las variables analizadas, por lo que las estimaciones de los coeficientes y los valores p en el resultado de la regresión probablemente no sean confiables.

In [293…

```
regresion = ols("ventas_semanales ~ vacaciones + temperatura + cpi + desempleo", da
results = regresion.fit()
```

In [294…

```
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       ventas_semanales   R-squared:                       0.019
Model:                           OLS   Adj. R-squared:                  0.019
Method:                Least Squares   F-statistic:                     29.17
Date:               Tue, 19 Mar 2024   Prob (F-statistic):           4.72e-24
Time:                       04:47:28   Log-Likelihood:                -87281.
No. Observations:               5951   AIC:                         1.746e+05
Df Residuals:                   5946   BIC:                         1.746e+05
Df Model:                          4
Covariance Type:           nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    1.708e+06   6.52e+04     26.200      0.000    1.58e+06    1.84e+06
vacaciones     7.2e+04   2.91e+04      2.470      0.014    1.49e+04    1.29e+05
temperatura -1010.1452    415.745     -2.430      0.015   -1825.157    -195.133
cpi         -1482.2890    198.329     -7.474      0.000   -1871.087   -1093.492
desempleo    -4.43e+04   6077.860     -7.289      0.000   -5.62e+04   -3.24e+04
==============================================================================
Omnibus:                      352.730   Durbin-Watson:                   0.113
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              418.586
Skew:                           0.650   Prob(JB):                     1.27e-91
Kurtosis:                       2.984   Cond. No.                     1.68e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
[2] The condition number is large, 1.68e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [295…

```
df2=df2[df2.columns.difference(['precio_combustible'])]
df2
```

Out[295]:

	cpi	desempleo	temperatura	vacaciones
0	211.10	8.11	42.31	0
1	211.24	8.11	38.51	1
2	211.29	8.11	39.93	0
3	211.32	8.11	46.63	0
4	211.35	8.11	46.50	0
…	…	…	…	…
6430	192.01	8.68	64.88	0
6431	192.17	8.67	64.89	0
6432	192.33	8.67	54.47	0
6433	192.33	8.67	56.47	0
6434	192.31	8.67	58.85	0

5951 rows × 4 columns

In [296…

```
# Creamos el dataframe del VIF
vif_data = pd.DataFrame()
vif_data["feature"] = df2.columns
```

```
# Calculamos el VIF por c/variable
vif_data["VIF"] = [variance_inflation_factor(df2.values, i) for i in range(len(df2.

print(vif_data)
```

```
       feature   VIF
0          cpi 14.80
1    desempleo 14.40
2  temperatura 12.17
3   vacaciones  1.10
```

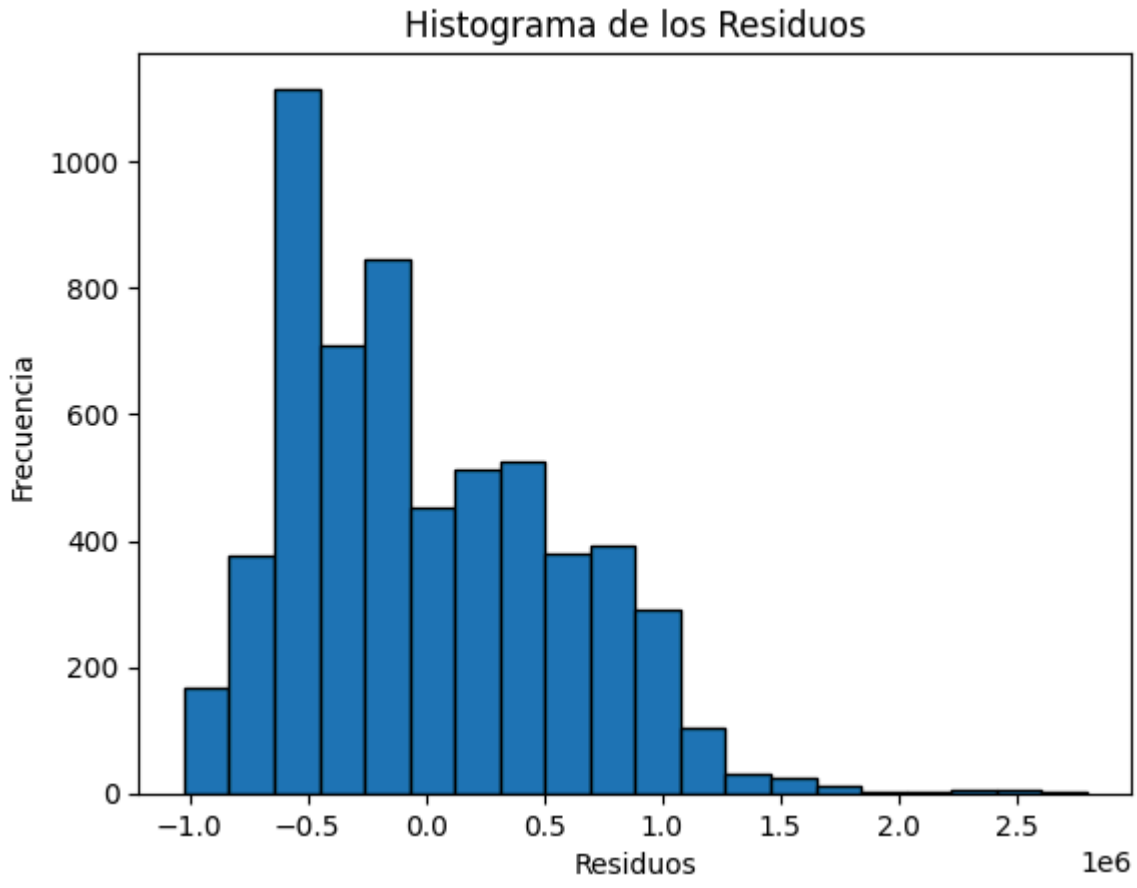
Si la relación es lineal, los residuos deberían estar distribuidos de manera aleatoria y no debería haber un patrón discernible en el gráfico.

El gráfico de valores ajustados vs residuos muestra la relación entre los valores ajustados por un modelo de regresión y los residuos de ese modelo. Los valores ajustados son los valores que el modelo predice para cada punto de datos, mientras que los residuos son la diferencia entre los valores reales y los valores ajustados.

In [297…

```
# Realizar la prueba de Shapiro-Wilk
shapiro_test = stats.shapiro(residuos)
print("Valor p del test de Shapiro-Wilk:", shapiro_test.pvalue)

# Graficar un histograma de los residuos
plt.hist(residuos, bins=20, edgecolor='black')
plt.xlabel('Residuos')
plt.ylabel('Frecuencia')
plt.title('Histograma de los Residuos')
plt.show()
```

```
Valor p del test de Shapiro-Wilk: 2.4607361552929518e-40
```



Si los residuos siguen una distribución normal, el valor p del test de Shapiro-Wilk debería ser mayor que un nivel de significancia elegido y el histograma debería mostrar una forma de

campana simétrica.

In [298…

```
X = data_new[['vacaciones', 'temperatura', 'cpi', 'desempleo']]
y = data_new['ventas_semanales']

# Añadir una constante al conjunto de variables independientes
X = sm.add_constant(X)

# Ajustar el modelo de regresión lineal
modelo = sm.OLS(y, X).fit()

# Hacer predicciones utilizando el modelo ajustado
predicciones = modelo.predict(X)

# Obtener los coeficientes y estadísticas del modelo
coeficientes = modelo.params
resumen_modelo = modelo.summary()

# Imprimir los coeficientes y el resumen del modelo
print("Coeficientes del modelo:")
print(coeficientes)
print("\nResumen del modelo:")
print(resumen_modelo)
```

```
Coeficientes del modelo:
const         1708223.87
vacaciones      71997.99
temperatura     -1010.15
cpi             -1482.29
desempleo      -44302.34
dtype: float64

Resumen del modelo:
                            OLS Regression Results
==============================================================================
Dep. Variable:       ventas_semanales   R-squared:                       0.019
Model:                            OLS   Adj. R-squared:                  0.019
Method:                 Least Squares   F-statistic:                     29.17
Date:                Tue, 19 Mar 2024   Prob (F-statistic):           4.72e-24
Time:                        04:47:28   Log-Likelihood:                -87281.
No. Observations:                5951   AIC:                         1.746e+05
Df Residuals:                    5946   BIC:                         1.746e+05
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        1.708e+06   6.52e+04     26.200      0.000    1.58e+06    1.84e+06
vacaciones     7.2e+04   2.91e+04      2.470      0.014    1.49e+04    1.29e+05
temperatura -1010.1452    415.745     -2.430      0.015   -1825.157    -195.133
cpi         -1482.2890    198.329     -7.474      0.000   -1871.087   -1093.492
desempleo    -4.43e+04   6077.860     -7.289      0.000   -5.62e+04   -3.24e+04
==============================================================================
Omnibus:                      352.730   Durbin-Watson:                   0.113
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              418.586
Skew:                           0.650   Prob(JB):                     1.27e-91
Kurtosis:                       2.984   Cond. No.                     1.68e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
[2] The condition number is large, 1.68e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

CONCLUSION Un R-cuadrado bajo sugiere que el modelo actual puede no ser suficientemente robusto para explicar las variaciones en las ventas semanales y puede requerir ajustes o mejoras adicionales.

10. Obtenga el modelo definitivo, prediga los valores y comente el grado de ajuste del modelo. Justifique con métricas su respuesta.

Para verificar esto probaremos otro método de regresión lineal con una data de prueba y una de entrenamiento.

Transformamos los datos

In [299…

```
!pip install category_encoders
```

```
Requirement already satisfied: category_encoders in /usr/local/lib/python3.10/dist
-packages (2.6.3)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-pac
kages (from category_encoders) (1.25.2)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/d
ist-packages (from category_encoders) (1.2.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-pack
ages (from category_encoders) (1.11.4)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dis
t-packages (from category_encoders) (0.14.1)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-pac
kages (from category_encoders) (1.5.3)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-pack
ages (from category_encoders) (0.5.6)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.1
0/dist-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-pack
ages (from pandas>=1.0.5->category_encoders) (2023.4)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (fro
m patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-pac
kages (from scikit-learn>=0.20.0->category_encoders) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/d
ist-packages (from scikit-learn>=0.20.0->category_encoders) (3.3.0)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-p
ackages (from statsmodels>=0.9.0->category_encoders) (24.0)
```

In [300…

```
num_features = data_new.select_dtypes('number').columns.to_list()
num_features.remove('ventas_semanales')

cat_features = data_new.select_dtypes('object').columns.to_list()

print(f'Variables numericas : {num_features}')
print(f'Variables categoricas: {cat_features}')
```

```
Variables numericas : ['vacaciones', 'temperatura', 'precio_combustible', 'cpi',
'desempleo']
Variables categoricas: ['tienda', 'mes_nombre', 'semana']
```

In [301…

```
var_cuantitativas = data_new.select_dtypes('number').columns
var_cualitativas  =data_new.select_dtypes('object').columns
```

In [302…

```
from sklearn.preprocessing import LabelEncoder
```

In [303…

```
# creating instance of labelencoder
labelencoder = LabelEncoder()
```

In [304…

```
data_new[var_cualitativas] = data_new[var_cualitativas].apply(labelencoder.fit_trar
```

In [305…

```
X = data_new[data_new.columns.difference(['ventas_semanales'])]
y = data_new.ventas_semanales
```

In [306…

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_s
```

In [307…

```
print(X_train.shape,"",type(X_train))
print(y_train.shape,"\t ",type(y_train))
print(X_test.shape,"",type(X_test))
print(y_test.shape,"\t ",type(y_test))
```

```
(4760, 8)  <class 'pandas.core.frame.DataFrame'>
(4760,)          <class 'pandas.core.series.Series'>
(1191, 8)  <class 'pandas.core.frame.DataFrame'>
(1191,)          <class 'pandas.core.series.Series'>
```

In [308…
```
modelo_regresion = LinearRegression()
modelo_regresion.fit(X_train, y_train)
```

Out[308]:
```
▾ LinearRegression
LinearRegression()
```

In [309…
```
predicciones_train = modelo_regresion.predict(X_train)
predicciones_test = modelo_regresion.predict(X_test)
```

Métricas de evaluación

In [310…
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
MSE_train = mean_squared_error(y_train, predicciones_train)
MSE_test = mean_squared_error(y_test, predicciones_test)
print(MSE_train)
print(MSE_test)
```

```
258507436660.1129
269148367069.60245
```

In [311…
```
RMSE_train = np.sqrt(MSE_train)
RMSE_test = np.sqrt(MSE_test)
print(RMSE_train)
print(RMSE_test)
```

```
508436.26607482764
518795.1108767337
```

In [312…
```
MAE_train = mean_absolute_error(y_train, predicciones_train)
MAE_test = mean_absolute_error(y_test, predicciones_test)
print(MAE_train)
print(MAE_test)
```

```
411939.33193182293
419803.465118216
```

In [313…
```
r_square_train = r2_score(y_train, predicciones_train)
r_square_test  = r2_score(y_test, predicciones_test)
print('El R^2 del subconjunto de entrenamiento es:' , r_square_train)
print('El R^2 del subconjunto de prueba es:' , r_square_test)
```

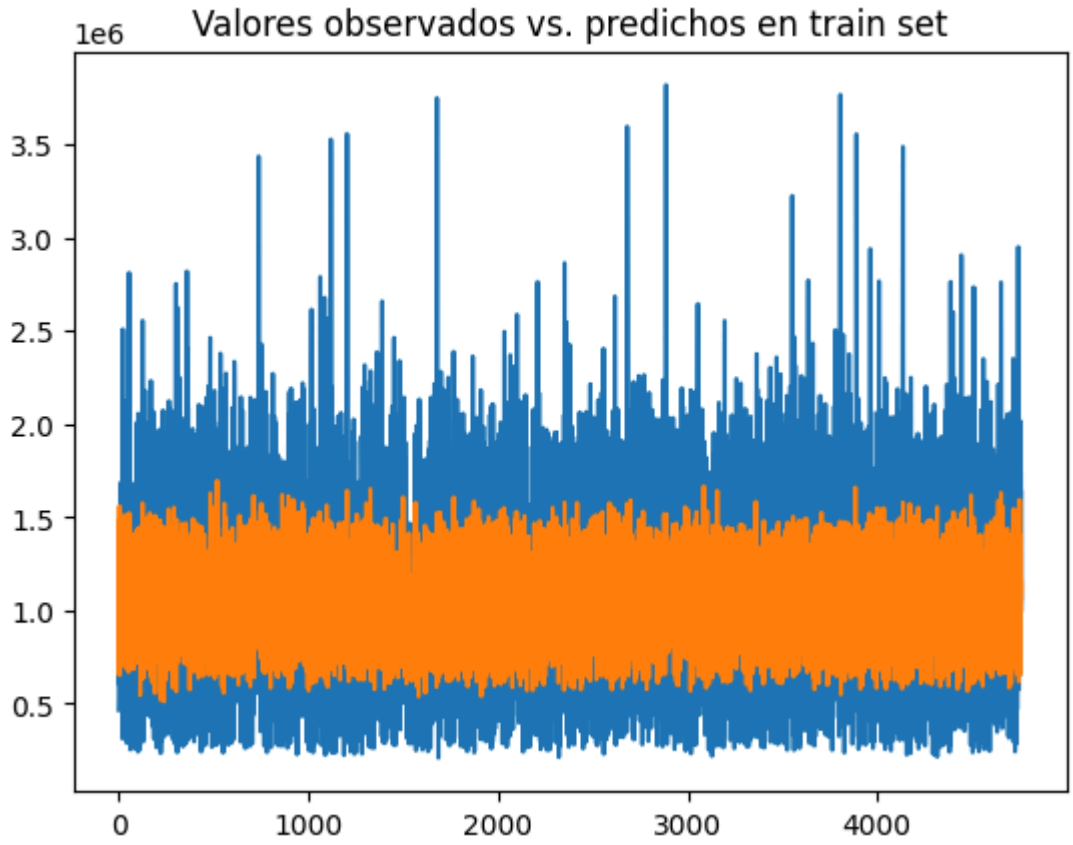
```
El R^2 del subconjunto de entrenamiento es: 0.20458688414643733
El R^2 del subconjunto de prueba es: 0.20208496100200968
```

In [314…
```
# Print the Intercept:
print('intercepto:', modelo_regresion.intercept_)

# Print the Slope:
print('pendiente:', modelo_regresion.coef_)
```

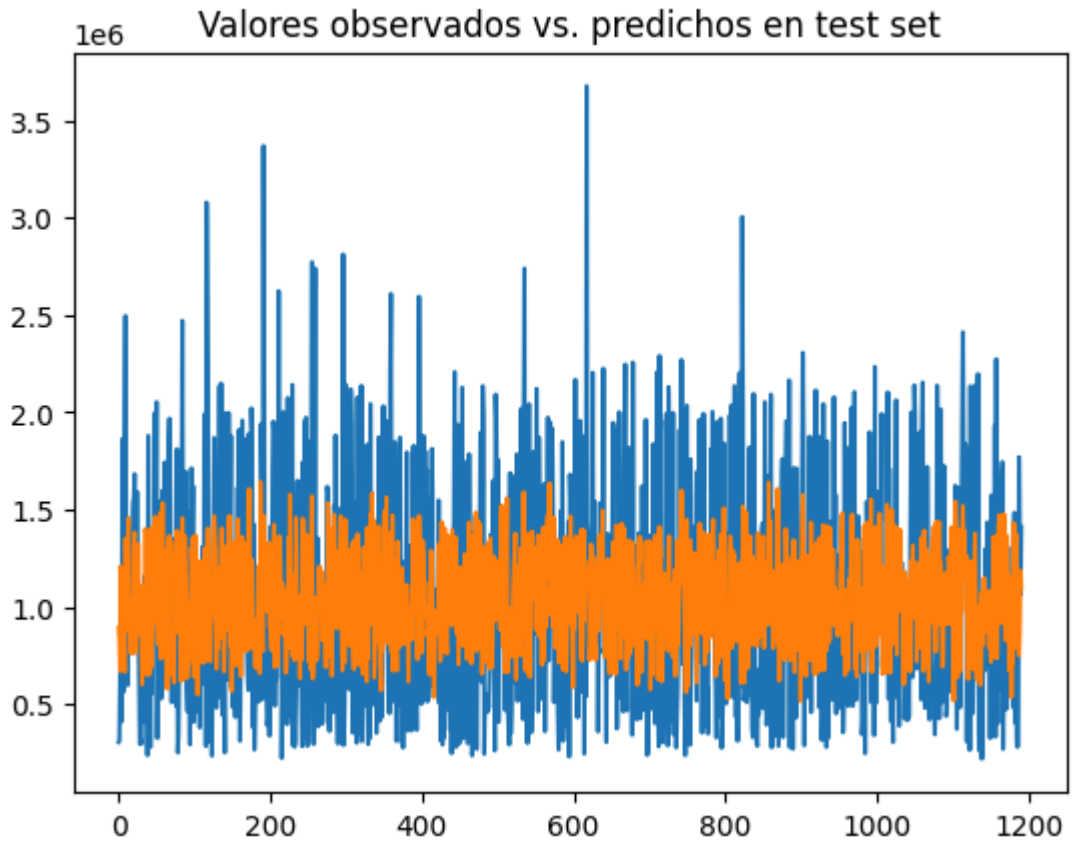
```
intercepto: 2089607.8763574583
pendiente: [  -215.39510495 -53186.31957476 -11007.04101501 -43569.54493494
   3695.06326688  -1090.11578421 -18796.08050671  54477.6015852 ]
```

11. Grafique a los valores predicho de modelo vs los valores reales. ¿Cómo se ven una vez graficados frente a los valores reales? Argumente su respuesta.

In [315…
```
fig, ax = plt.subplots()
ax.plot(y_train.values)
ax.plot(predicciones_train)
plt.title("Valores observados vs. predichos en train set");
```

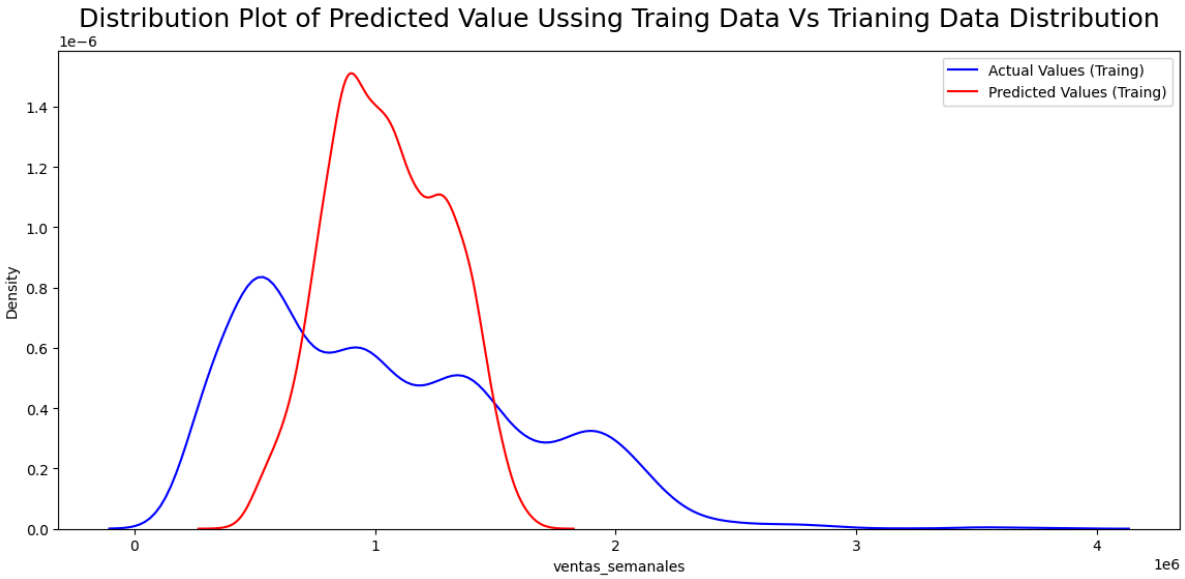


In [316…
```
fig, ax = plt.subplots()
ax.plot(y_test.values)
ax.plot(predicciones_test)
plt.title("Valores observados vs. predichos en test set");
```



In [317…

```
Title = 'Distribution Plot of Predicted Value Ussing Traing Data Vs Trianing Data D
Distribution_Plot(Linear_model, 'Traing',X_train_transformed, y_train, Title)
```



Conclusión: No hay correlaciones lineales entre las variables y la variable objetivo. El modelo parece no estar aprendiendo correctamente de los datos de entrenamiento, por lo que necesitamos aumentar la complejidad de este modelo. Vamos a utilizar características polinomiales para los datos antes de modelar.

12. Concluya sobre su modelo. Para ello, si escogió el enfoque econométrico, interprete coeficientes, por el contrario si escogió el enfoque de machine learning, determine cuáles son las variables que tienen mayor poder explicativo sobre su variable objetivo.

Para nuestro modelo final vamos a probar un modelo de regresión polinómica.

La regresión polinomial es un caso especial de la regresión lineal múltiple en la que se busca obtener la predicción de una variable de respuesta cuantitativa a partir de una variable predictora cuantitativa, donde la relación se modela como una función polinomial de orden o grado 22,23,24

In [318…
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import GridSearchCV

Linear_pipe = Pipeline([('poly_feat', PolynomialFeatures()),
                        ('lin_reg',LinearRegression())])

## Define the parameter grid to search
param_grid = {'poly_feat__degree':[2, 3, 4]}

# Assuming Linear_pipe, X_train_transformed, and y_train are defined
best_estimator = hyperparameter_tunning(Linear_pipe, X_train_transformed, y_train,
```

```
Best parameters: {'poly_feat__degree': 3}

Best score: 0.9580852729481227
```

In [319…
```
## Linear Regression Model After Tunning
poly_reg = best_estimator
```

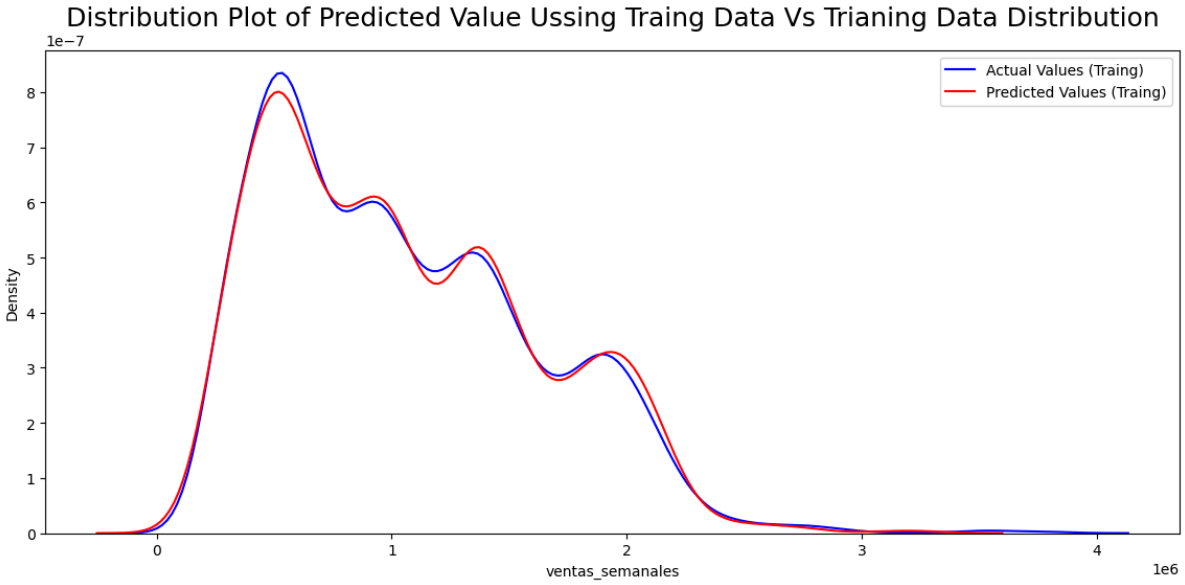
Evaluamos el modelo después del ajuste

In [320…
```
model_evaluation(poly_reg, 'Traing',X_train_transformed, y_train)
```

```
Traing Accuracy:

-> Root Mean Squared Error: 73330.78
-> R-Squere score Training: 98.35 %
```

Graficamos los valores predichos vs los actuales

In [321…
```
Title = 'Distribution Plot of Predicted Value Ussing Traing Data Vs Trianing Data D
Distribution_Plot(poly_reg, 'Traing',X_train_transformed, y_train, Title)
```



Conclusión: Después de aplicar Características Polinomiales a los datos de entrenamiento, el modelo parece haber mejorado más en el aprendizaje de los datos de entrenamiento.

In [322…
```
cross_validation_score(poly_reg,X_train_transformed, y_train)
```

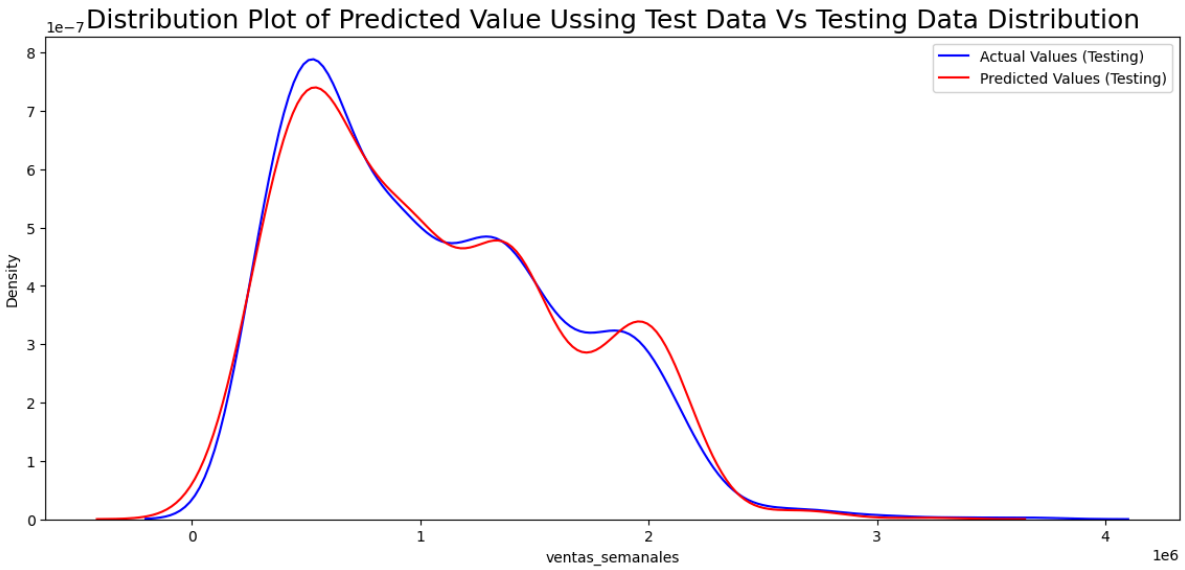
```
Cross Validation Scores: [0.95192441 0.95424793 0.96139568 0.96151762 0.96134071]

Mean of Scores: 95.81 %

Standard Deviation of Scores: 0.004147749248709301
```

Con estos datos, podemos concluir que el modelo de regresión polinómica tiene un buen rendimiento en general, ya que los puntajes de validación cruzada son consistentemente altos, con un promedio del 95.81%. Además, la desviación estándar de los puntajes es baja, lo que indica una baja variabilidad en el rendimiento del modelo en diferentes divisiones de los datos. Esto sugiere que el modelo es robusto y generaliza bien a diferentes subconjuntos de datos.

In [323…
```
model_evaluation(poly_reg,'Testing',X_test_transformed, y_test)
```

```
Testing Accuracy:

-> Root Mean Squared Error: 101361.56
-> R-Squere score Training: 96.95 %
```

In [324…
```
Title = 'Distribution Plot of Predicted Value Ussing Test Data Vs Testing Data Dist
Distribution_Plot(poly_reg, 'Testing',X_test_transformed, y_test, Title)
```



Esto indica que el modelo de regresión polinómica tiene un buen rendimiento en el conjunto de pruebas.

- El error cuadrático medio (RMSE) de las predicciones en el conjunto de pruebas es de aproximadamente 101361.56. El RMSE es una medida de la diferencia entre los valores predichos por el modelo y los valores reales en el conjunto de pruebas. Un RMSE más bajo indica un mejor ajuste del modelo a los datos.
- El puntaje R cuadrado (R^2) del modelo en el conjunto de pruebas es del 96.95%. Este puntaje indica la proporción de la varianza en la variable objetivo que es explicada por

el modelo. Un puntaje R^2 más alto sugiere que el modelo es capaz de explicar una mayor parte de la variabilidad en los datos de prueba. En este caso, un puntaje del 96.95% indica que el modelo explica una gran parte de la variabilidad en los datos de prueba, lo que sugiere un buen ajuste del modelo.

In [325…

```
!jupyter nbconvert --to html 'Grupo4-Deber4_proyecto final.ipynb'
```

```
[NbConvertApp] WARNING | pattern 'Grupo4-Deber4_proyecto final.ipynb' matched no f
iles
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and inclu
de the error message in the cell output (the default behaviour is to abort convers
ion). This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with defa
ult basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_f
ormat=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_f
ormat=notebook --FilesWriter.build_directory= --ClearOutputPreprocessor.enabled=Tr
ue]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateExporte
r.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --TemplateExport
er.exclude_input=True --TemplateExporter.exclude_input_prompt=True]
```

```
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the s
ystem.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only useful fo
r the HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRI
TICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf',
'python', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_name]
--template-file=<Unicode>
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
    Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
    Whether the HTML in Markdown cells and cell outputs should be sanitized.This
    should be set to True by nbviewer or similar tools.
    Default: False
    Equivalent to: [--HTMLExporter.sanitize_html]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
```

```
    overwrite base name use for output files.
                can only be used when converting one notebook at a time.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                    to output to the directory of each notebook. To
recover
                                    previous default behaviour (outputting to the cu
rrent
                                    working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-
slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb --to html

            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'n
otebook', 'pdf', 'python', 'rst', 'script', 'slides', 'webpdf'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX includes
            'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
            'classic'. You can specify the flavor of the format used.

            > jupyter nbconvert --to html --template lab mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve
```

```
                    Multiple notebooks can be given at the command line in a couple of
                    different ways:

                    > jupyter nbconvert notebook*.ipynb
                    > jupyter nbconvert notebook1.ipynb notebook2.ipynb

                    or you can specify the notebooks list in a config file, containing::

                        c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

                    > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```