S4- CASO PRACTICO-BDD FINAL

GUIDO YUNGÁN

DESARROLLO

Datos:

De acuerdo a los siguientes campos:

- Store: El número de la tienda.
- Date: Fecha
- Weekly_Sales - Ventas de la semana.
- 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

1. Importe la base de datos y ejecute el script en una base al Jupyter Notebook con pandas.

In [2]:
```
# SE IMPORTA LAS LIBRERIAS
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

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

In [3]:
```
# PARA REGRESION Y VALIDACION DE SUPUESTOS LLAMAMOS A STATSMODEL
import statsmodels.api as sm
import statsmodels.formula.api as sms
from statsmodels.formula.api import ols
from statsmodels.compat import lzip
```

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 [4]:
```
# LLAMAMOS A LA BASE DE DATOS
df=pd.read_csv("Walmart.csv")
df
```

Out[4]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
0	1	05-02-2010	1643690.90	0	42.31	2.572	211.096358	8.106
1	1	12-02-2010	1641957.44	1	38.51	2.548	211.242170	8.106
2	1	19-02-2010	1611968.17	0	39.93	2.514	211.289143	8.106
3	1	26-02-2010	1409727.59	0	46.63	2.561	211.319643	8.106
4	1	05-03-2010	1554806.68	0	46.50	2.625	211.350143	8.106
...	...	...	...	...	...	...	...	...
6430	45	28-09-2012	713173.95	0	64.88	3.997	192.013558	8.684
6431	45	05-10-2012	733455.07	0	64.89	3.985	192.170412	8.667
6432	45	12-10-2012	734464.36	0	54.47	4.000	192.327265	8.667
6433	45	19-10-2012	718125.53	0	56.47	3.969	192.330854	8.667
6434	45	26-10-2012	760281.43	0	58.85	3.882	192.308899	8.667

6435 rows × 8 columns

In [5]:
```
df['Date']= pd.to_datetime(df['Date'])
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df
```

Out[5]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	Year	Month
0	1	2010-05-02	1643690.90	0	42.31	2.572	211.096358	8.106	2010	5
1	1	2010-12-02	1641957.44	1	38.51	2.548	211.242170	8.106	2010	12
2	1	2010-02-19	1611968.17	0	39.93	2.514	211.289143	8.106	2010	2
3	1	2010-02-26	1409727.59	0	46.63	2.561	211.319643	8.106	2010	2
4	1	2010-05-03	1554806.68	0	46.50	2.625	211.350143	8.106	2010	5
...	...	...	...	...	...	...	...	...	...	...
6430	45	2012-09-28	713173.95	0	64.88	3.997	192.013558	8.684	2012	9
6431	45	2012-05-10	733455.07	0	64.89	3.985	192.170412	8.667	2012	5
6432	45	2012-12-10	734464.36	0	54.47	4.000	192.327265	8.667	2012	12
6433	45	2012-10-19	718125.53	0	56.47	3.969	192.330854	8.667	2012	10
6434	45	2012-10-26	760281.43	0	58.85	3.882	192.308899	8.667	2012	10

6435 rows × 10 columns

In [6]:
```
# DESCRIPCION DE VARIABLES NUMERICAS
df.describe()
```

Out[6]:

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	Year	Month
count	6435.000000	6.435000e+03	6435.000000	6435.000000	6435.000000	6435.000000	6435.000000	6435.000000	6435.000000
mean	23.000000	1.046965e+06	0.069930	60.663782	3.358607	171.578394	7.999151	2010.965035	6.475524
std	12.988182	5.643666e+05	0.255049	18.444933	0.459020	39.356712	1.875885	0.797019	3.321797
min	1.000000	2.099862e+05	0.000000	-2.060000	2.472000	126.064000	3.879000	2010.000000	1.000000
25%	12.000000	5.533501e+05	0.000000	47.460000	2.933000	131.735000	6.891000	2010.000000	4.000000
50%	23.000000	9.607460e+05	0.000000	62.670000	3.445000	182.616521	7.874000	2011.000000	6.000000
75%	34.000000	1.420159e+06	0.000000	74.940000	3.735000	212.743293	8.622000	2012.000000	9.000000
max	45.000000	3.818686e+06	1.000000	100.140000	4.468000	227.232807	14.313000	2012.000000	12.000000

In [7]:
```
df.dtypes
```

Out[7]:
```
Store                    int64
Date            datetime64[ns]
Weekly_Sales           float64
Holiday_Flag             int64
Temperature            float64
Fuel_Price             float64
CPI                    float64
Unemployment           float64
Year                     int64
Month                    int64
dtype: object
```

- En el resultado DESCRIPCION DE VARIABLES NUMERICAS, se obervan 7 variables de este tipo, todos con valores positivos. La unica variable con valores en cero es Holiday_Flag (Bandera_Festiva)
- Con respecto a la tabla DESCRIPCION DE VARIABLES CUALITATIVAS, existe solo la variable Date que consta de 6435 registros.

3. Evalúe si la base contiene datos perdidos.

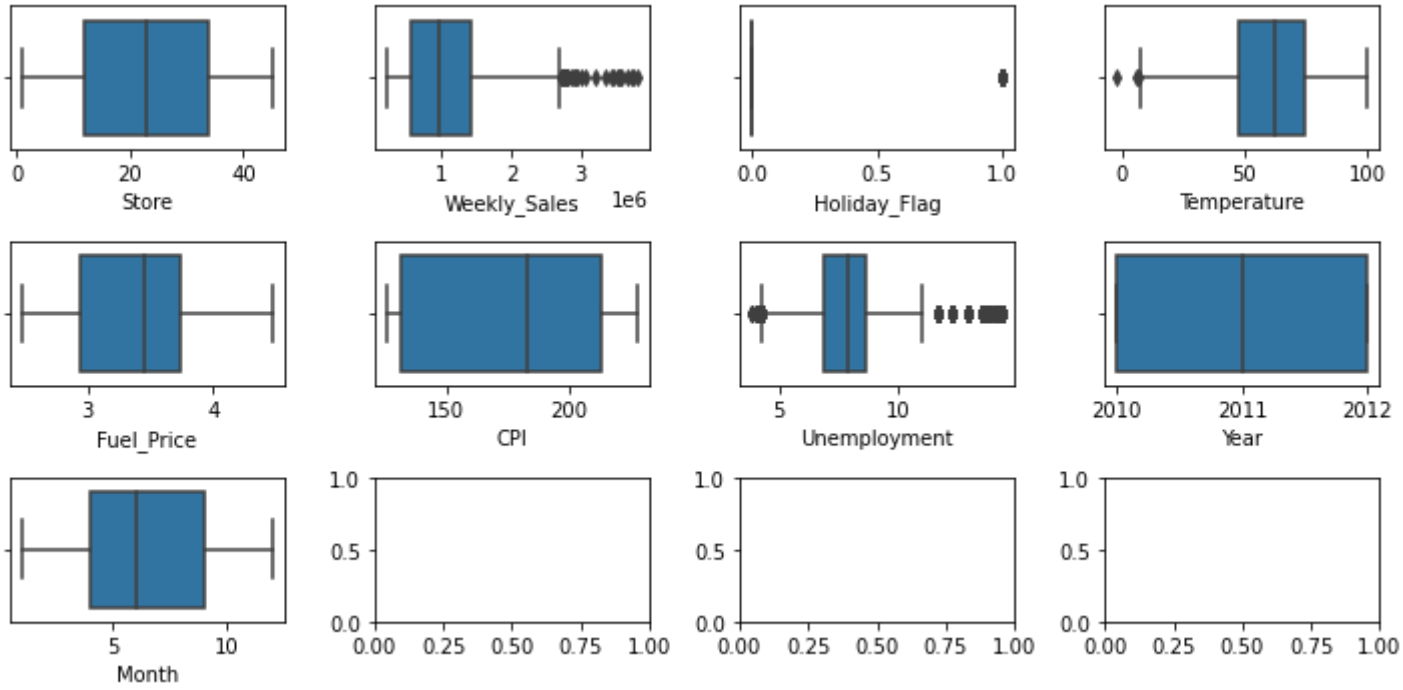
In [8]:
```
# REVISION DE VALORES PERDIDOS
df.isna().sum()
```

Out[8]:
```
Store           0
Date            0
Weekly_Sales    0
Holiday_Flag    0
Temperature     0
Fuel_Price      0
CPI             0
Unemployment    0
Year            0
Month           0
dtype: int64
```

No existen valores perdidos en niguna de las variables, tal y como se verifico en el punto 2.

4. Evalúe si alguna de las variables contiene datos atípicos (outliers). De ser el caso detalle cuáles y qué método estadístico aplicarán para corregir.

In [9]:
```
# VALIDACION DE DATOS ATÍPICOS MEDIANTE GRAFICO DE CAJAS

fig, axs = plt.subplots(3,4, figsize = (10,5))
plt1 = sns.boxplot(df['Store'], ax = axs[0,0])
plt2 = sns.boxplot(df['Weekly_Sales'], ax = axs[0,1])
plt3 = sns.boxplot(df['Holiday_Flag'], ax = axs[0,2])
plt4 = sns.boxplot(df['Temperature'], ax = axs[0,3])
plt5 = sns.boxplot(df['Fuel_Price'], ax = axs[1,0])
plt6 = sns.boxplot(df['CPI'], ax = axs[1,1]) #
plt7 = sns.boxplot(df['Unemployment'], ax = axs[1,2]) #
plt6 = sns.boxplot(df['Year'], ax = axs[1,3]) #
plt7 = sns.boxplot(df['Month'], ax = axs[2,0]) #
plt.tight_layout()
```



5. De ser el caso, detalle cuáles y qué método estadístico aplicarán para corregir.

Con el resultado anterior, se puede visualizar que las variables cuantitativas con más datos atipicos son Weekly_Sales, Temperature y Unemployment. El método estádistico que se utilizara para la remocion de datos atipicos sera el IQR

In [10]:
```
#IQR PARA RETIRO DE DATOS ATIPICOS DE VARIABLE Weekly_Sales
Q1 = df['Weekly_Sales'].quantile(0.25)
Q3 = df['Weekly_Sales'].quantile(0.75)
IQR_Weekly_Sales = Q3 - Q1  #rango intercuartil
print('\nIQR_Weekly_Sales: ',IQR_Weekly_Sales)

# RETIRO DATOS ATIPOS DE VARIABLE EDAD
df = df[~((df['Weekly_Sales'] < (Q1 - 1.5 * IQR_Weekly_Sales)) |(df['Weekly_Sales'] > (Q3 + 1.5 * IQR_Weekly_Sales)))]
print ('\nDatos removidos de la variable Weekly_Sales: ', df.shape)
```

```
IQR_Weekly_Sales:  866808.5549999999

Datos removidos de la variable Weekly_Sales:  (6401, 10)
```

In [11]:
```
#IQR PARA RETIRO DE DATOS ATIPICOS DE VARIABLE Temperature
Q1 = df['Temperature'].quantile(0.25)
Q3 = df['Temperature'].quantile(0.75)
IQR_Temperature = Q3 - Q1  #rango intercuartil
print('\nIQR_Temperature: ',IQR_Temperature)

# RETIRO DATOS ATIPOS DE VARIABLE DURACION
df = df[~((df['Temperature'] < (Q1 - 1.5 * IQR_Temperature)) |(df['Temperature'] > (Q3 + 1.5 * IQR_Temperature)))]
print ('\nDatos removidos de la variable Temperature: ', df.shape)
```

```
IQR_Temperature:  27.340000000000003

Datos removidos de la variable Temperature:  (6398, 10)
```

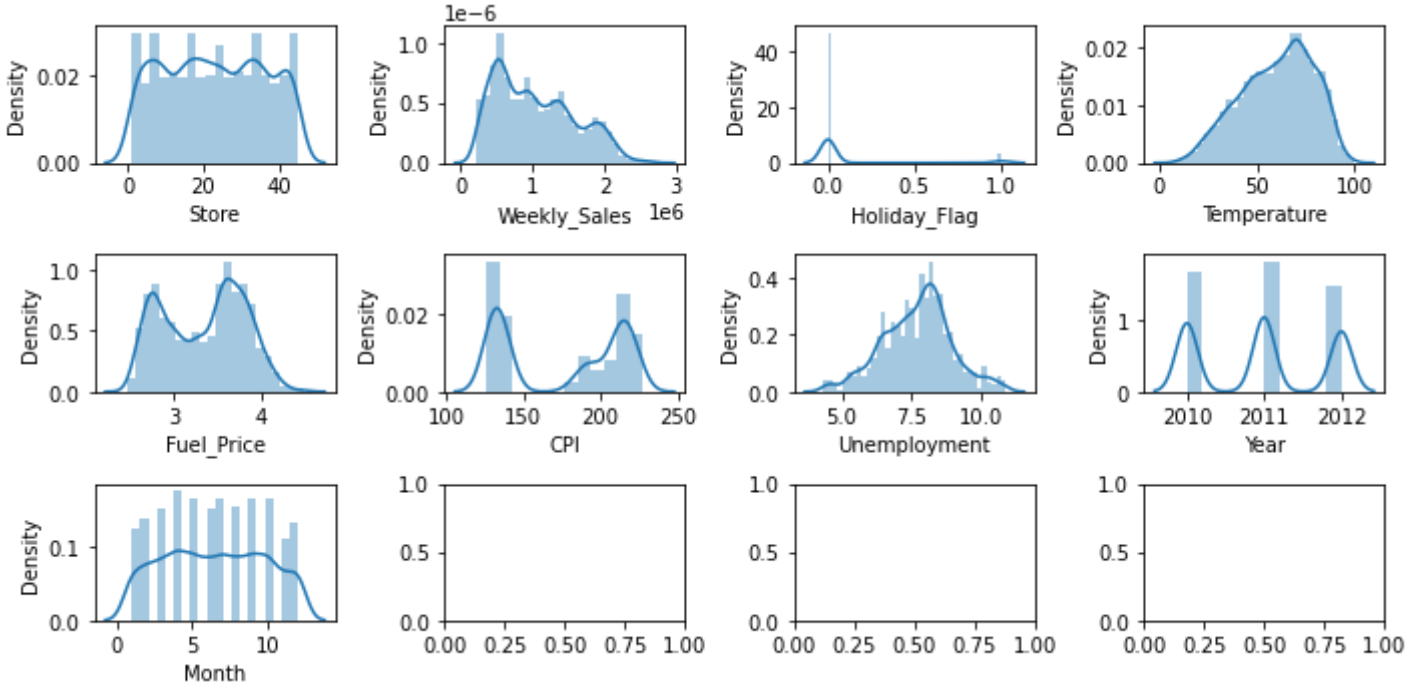
In [12]:
```
#IQR PARA RETIRO DE DATOS ATIPICOS DE VARIABLE Unemployment
Q1 = df['Unemployment'].quantile(0.25)
Q3 = df['Unemployment'].quantile(0.75)
IQR_Unemployment = Q3 - Q1  #rango intercuartil
print('\nIQR_Unemployment: ',IQR_Unemployment)

# RETIRO DATOS ATIPOS DE VARIABLE CAMPAÑA
df = df[~((df['Unemployment'] < (Q1 - 1.5 * IQR_Unemployment)) |(df['Unemployment'] > (Q3 + 1.5 * IQR_Unemployment)))]
df.shape
print ('\nDatos removidos de la variable Unemployment: ', df.shape)
```

```
IQR_Unemployment:  1.7309999999999999

Datos removidos de la variable Unemployment:  (5917, 10)
```

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

In [13]:
```
fig, axs = plt.subplots(3,4, figsize = (10,5))

plt1 = sns.distplot(df['Store'], ax = axs[0,0])
plt2 = sns.distplot(df['Weekly_Sales'], ax = axs[0,1])
plt3 = sns.distplot(df['Holiday_Flag'], ax = axs[0,2])
plt4 = sns.distplot(df['Temperature'], ax = axs[0,3])
plt1 = sns.distplot(df['Fuel_Price'], ax = axs[1,0])
plt2 = sns.distplot(df['CPI'], ax = axs[1,1])
plt3 = sns.distplot(df['Unemployment'], ax = axs[1,2])
plt6 = sns.distplot(df['Year'], ax = axs[1,3]) #
plt7 = sns.distplot(df['Month'], ax = axs[2,0]) #
```

```
plt.tight_layout()
```



Como se observa en las graficas las variables poseen un distribucion multimodal y normal, luego de los ajustes realizados con el IQR

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

In [14]:
```
df.corr().style.background_gradient(cmap='coolwarm')
```

Out[14]:

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	Year	Month
Store	1.000000	-0.319354	0.004289	-0.027045	0.047519	-0.208637	0.309645	-0.007421	0.011490
Weekly_Sales	-0.319354	1.000000	0.024390	-0.041686	0.019664	-0.082977	-0.073092	-0.021170	0.035620
Holiday_Flag	0.004289	0.024390	1.000000	-0.157220	-0.076529	-0.000450	0.011031	-0.053861	0.331851
Temperature	-0.027045	-0.041686	-0.157220	1.000000	0.145157	0.217847	0.024204	0.083390	0.074025
Fuel_Price	0.047519	0.019664	-0.076529	0.145157	1.000000	-0.144515	-0.105214	0.782957	-0.042427
CPI	-0.208637	-0.082977	-0.000450	0.217847	-0.144515	1.000000	-0.219020	0.086930	-0.000550
Unemployment	0.309645	-0.073092	0.011031	0.024204	-0.105214	-0.219020	1.000000	-0.242435	0.005468
Year	-0.007421	-0.021170	-0.053861	0.083390	0.782957	0.086930	-0.242435	1.000000	-0.128867
Month	0.011490	0.035620	0.331851	0.074025	-0.042427	-0.000550	0.005468	-0.128867	1.000000

Se puede observar que las variables con mayor correlación negativa entre : Full_price con Year, Weekly_Sales con Store y una correlacion positiva entre: Unemployment con Store y Month con Holiday_Flag

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

De acuerdo a los graficos y valores de la correlacion en el punto 7 se observa que Year con Fuel Price tienen una fuerte correlacion, sin embargo esto no podria arrojar incoherencias, por lo que se va escoger las siguientes variables:

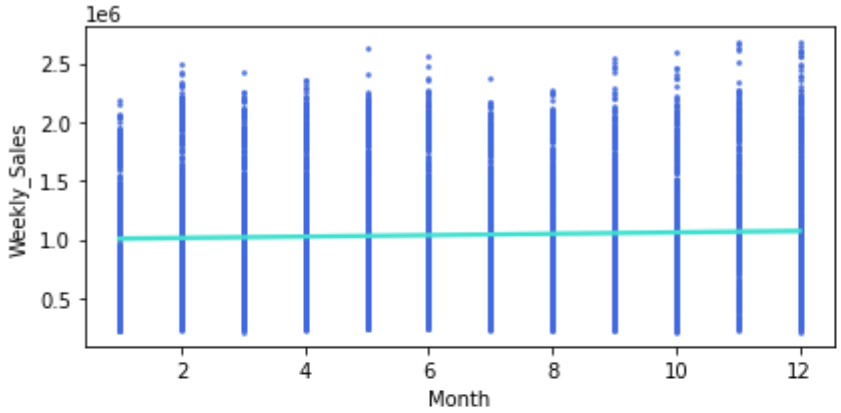
- La variable dependiente escogida es Weekly_Sales (ventas por semana)
- Las variables predictoras son Store, Holyday_Flag, Temperature, Fuel_Price, CPI, Unemployment, Month, en este caso se ha decidido excluir la variable date ya que para el modelo de regresion multiple no es aplicable.

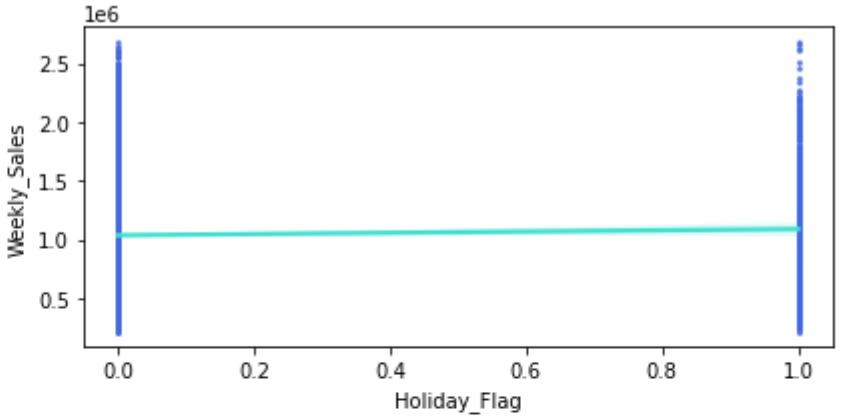
9. Indique que tipo de modelación realizarán y porqué.

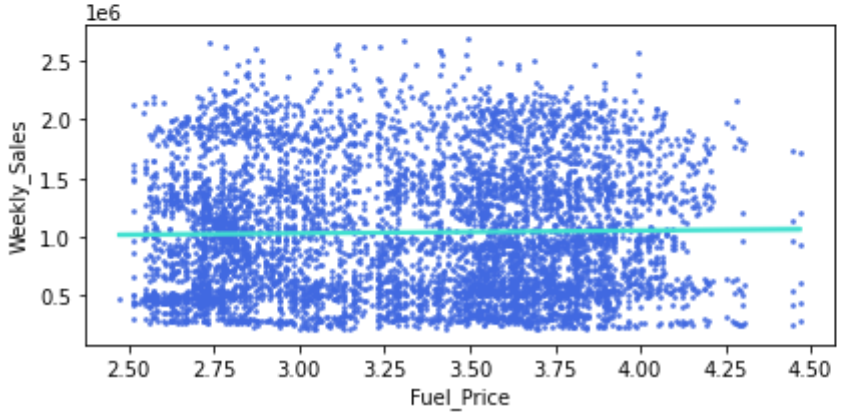
El modelo escogido inicialmente sera el de regresion lineal multiple, esto con el fin de analizar a detalle todas las variables explicativas que poseen baja correlacion entre si, para lo cual se va a excluir la variable Date ya que este modelo no aplica para predictores basados en tiempo

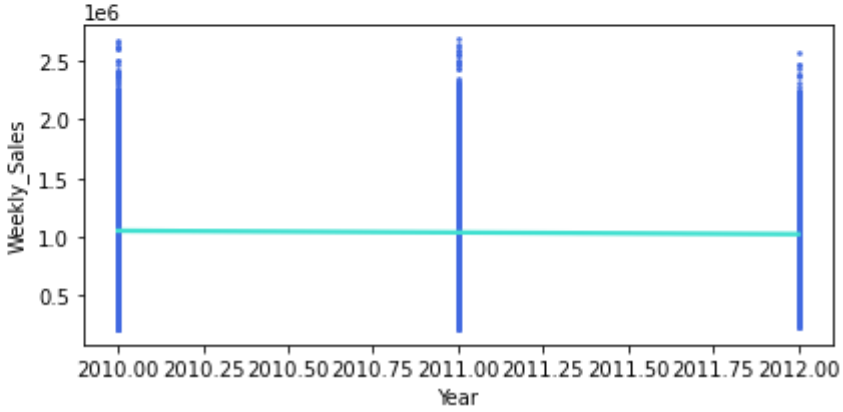
Iniciaremos comparando mediante un grafico de residual como estan las correlaciones de las variables cuantitativas

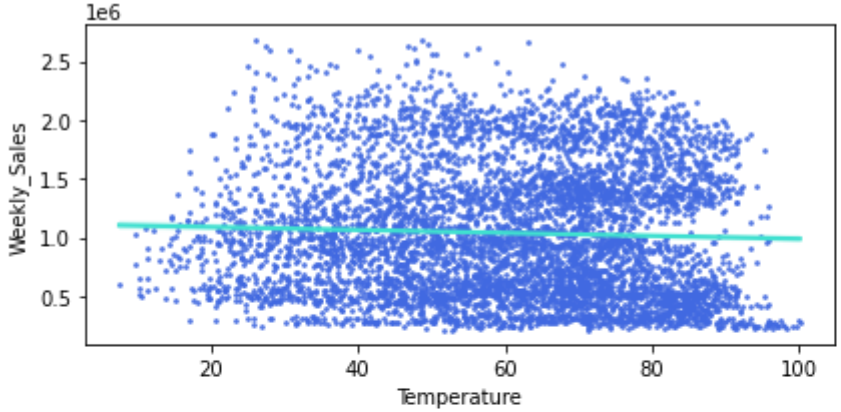
In [17]:
```
# Número de las variables
n = 8
fig = plt.figure(figsize=(12,12))
# Correlaciones en pares
corr = df.corr()
#
cols = corr.nlargest(8, "Weekly_Sales")["Weekly_Sales"].index
# Calculate correlation
for i in np.arange(1,8):
    regline = df[cols[i]]
    ax = fig.add_subplot(4,2,i)
    sns.regplot(x=regline, y=df['Weekly_Sales'], scatter_kws={"color": "royalblue", "s": 3},
                line_kws={"color": "turquoise"})
plt.tight_layout()
plt.show()
```

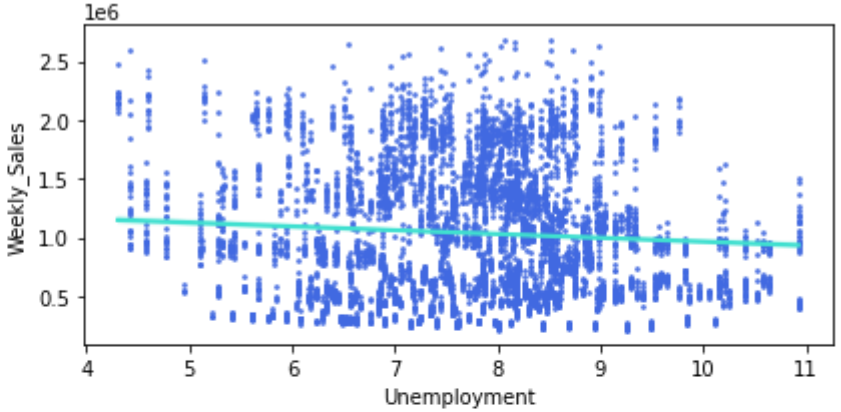

1e6
Weekly_Sales
Month


1e6
Weekly_Sales
Holiday_Flag

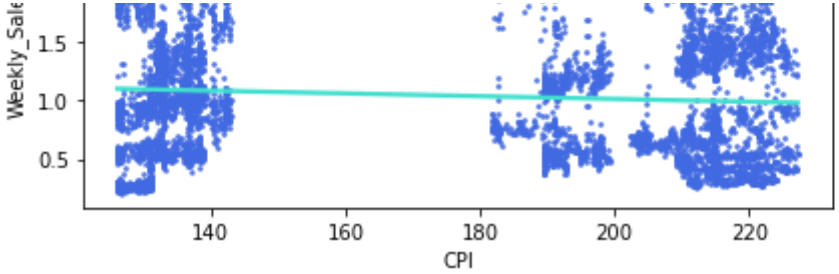

1e6
Weekly_Sales
Fuel_Price


1e6
Weekly_Sales
Year


1e6
Weekly_Sales
Temperature


1e6
Weekly_Sales
Unemployment


1e6



In [18]:
```
#LOGARITMO A LA VARIABLE DEPENDIENTE
log_Weekly_Sales=np.log(df.Weekly_Sales)
df['log_Weekly_Sales']=log_Weekly_Sales
```

In [19]:
```
#APLICAMOS LA REGRESION A TODAS LAS VARIABLES
regresion = ols("log_Weekly_Sales ~ Store + Holiday_Flag + Temperature + Fuel_Price + CPI + Unemployment + Year + Month" , data=
results = regresion.fit()
```

In [20]:
```
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       log_Weekly_Sales   R-squared:                       0.102
Model:                           OLS   Adj. R-squared:                  0.101
Method:                Least Squares   F-statistic:                     83.75
Date:               Tue, 27 Dec 2022   Prob (F-statistic):          7.15e-132
Time:                       09:55:39   Log-Likelihood:                -4916.5
No. Observations:               5917   AIC:                             9851.
Df Residuals:                   5908   BIC:                             9911.
Df Model:                          8
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept       92.5967     32.347      2.863      0.004      29.185     156.008
Store           -0.0139      0.001    -23.538      0.000      -0.015      -0.013
Holiday_Flag    -0.0002      0.031     -0.006      0.996      -0.061       0.060
Temperature     -0.0024      0.000     -5.649      0.000      -0.003      -0.002
Fuel_Price       0.0975      0.028      3.496      0.000       0.043       0.152
CPI             -0.0017      0.000     -7.886      0.000      -0.002      -0.001
Unemployment     0.0145      0.006      2.249      0.025       0.002       0.027
Year            -0.0391      0.016     -2.425      0.015      -0.071      -0.007
Month            0.0062      0.002      2.597      0.009       0.002       0.011
==============================================================================
Omnibus:                      481.857   Durbin-Watson:                   0.062
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              302.019
Skew:                          -0.427   Prob(JB):                     2.61e-66
Kurtosis:                       2.297   Cond. No.                     9.04e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 9.04e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Inicialmente el modelo explica un 10,2% de variabilidad de nuestra variable dependiente por medio de la inclusión de 8 variables explicativas. Tambien se observa que la mayoria de las variables pasan la prueba de confianza del 95 %, sin embargo se debe analizar cuales de las variables son estadisticamente significativas.

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

In [21]:
```
#VALIDACION DE SUPUESTOS
from statsmodels.stats.outliers_influence import variance_inflation_factor
df2=df[df.columns.difference(['Weekly_Sales', 'log_Weekly_Sales', 'Date'])]
df2
```

Out[21]:

	CPI	Fuel_Price	Holiday_Flag	Month	Store	Temperature	Unemployment	Year
0	211.096358	2.572	0	5	1	42.31	8.106	2010
1	211.242170	2.548	1	12	1	38.51	8.106	2010
2	211.289143	2.514	0	2	1	39.93	8.106	2010
3	211.319643	2.561	0	2	1	46.63	8.106	2010
4	211.350143	2.625	0	5	1	46.50	8.106	2010
...	...	...	...	...	...	...	...	...
6430	192.013558	3.997	0	9	45	64.88	8.684	2012
6431	192.170412	3.985	0	5	45	64.89	8.667	2012
6432	192.327265	4.000	0	12	45	54.47	8.667	2012
6433	192.330854	3.969	0	10	45	56.47	8.667	2012
6434	192.308899	3.882	0	10	45	58.85	8.667	2012

5917 rows × 8 columns

In [28]:
```
# 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.columns))]

print(vif_data)
```

```
        feature         VIF
0           CPI   25.057454
1    Fuel_Price   59.515635
2  Holiday_Flag    1.258203
3         Month    5.537761
4         Store    4.584018
5   Temperature   13.553152
6  Unemployment   46.592037
7          Year  165.545544
```

In [29]:
```
# Creamos el dataframe del VIF
vif_data = pd.DataFrame()
vif_data["feature"] = df2.columns.difference([' Holiday_Flag', 'Year'])

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

print(vif_data)
```

```
        feature        VIF
0           CPI  17.095995
1    Fuel_Price  29.622221
2  Holiday_Flag   1.258106
3         Month   5.385998
4         Store   4.583428
5   Temperature  13.551110
6  Unemployment  27.706011
```

In [30]:
```
# Creamos el dataframe del VIF
vif_data = pd.DataFrame()
vif_data["feature"] = df2.columns.difference(['Fuel_Price', 'Year', 'Holiday_Flag'])

# Calculamos el VIF por c/variable
vif_data["VIF"] = [variance_inflation_factor(df2[df2.columns.difference(['Fuel_Price','Year','Holiday_Flag'])].values, i) \
                   for i in range(len(df2[df2.columns.difference(['Fuel_Price','Year', 'Holiday_Flag'])].columns))]

print(vif_data)
```

```
        feature        VIF
0           CPI  15.181578
1         Month   4.664950
2         Store   4.538217
3   Temperature  12.109181
4  Unemployment  20.337504
```

In [31]:
```
# Creamos el dataframe del VIF
vif_data = pd.DataFrame()
vif_data["feature"] = df2.columns.difference(['Fuel_Price', 'Year', 'Unemployment', 'Holiday_Flag'])

# Calculamos el VIF por c/variable
vif_data["VIF"] = [variance_inflation_factor(df2[df2.columns.difference(['Fuel_Price','Year','Unemployment','Holiday_Flag'])].va
                   for i in range(len(df2[df2.columns.difference(['Fuel_Price','Year', 'Unemployment','Holiday_Flag'])].columns)

print(vif_data)
```

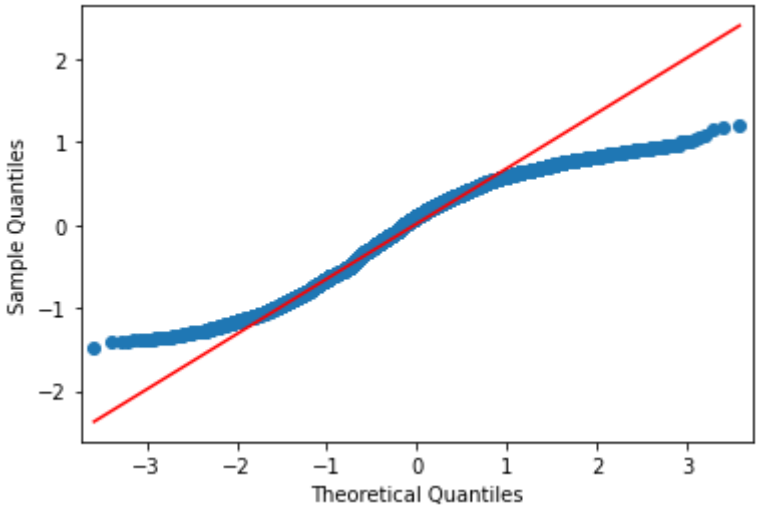
```
       feature        VIF
0          CPI  11.272182
1        Month   4.447210
2        Store   3.404047
3  Temperature  10.877767
```

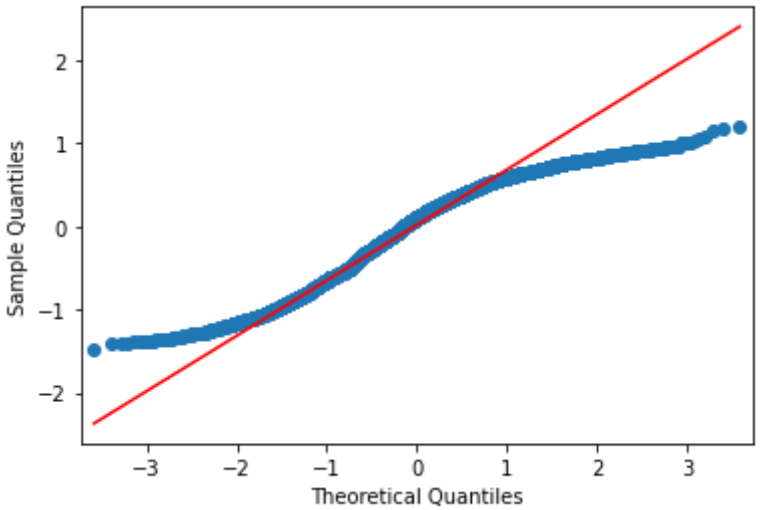
In [32]:
```
# Creamos el dataframe del VIF
vif_data = pd.DataFrame()
vif_data["feature"] = df2.columns.difference(['Fuel_Price', 'Year', 'Unemployment', 'Holiday_Flag', 'CPI'])

# Calculamos el VIF por c/variable
vif_data["VIF"] = [variance_inflation_factor(df2[df2.columns.difference(['Fuel_Price','Year','Unemployment','Holiday_Flag', 'CPI
                   for i in range(len(df2[df2.columns.difference(['Fuel_Price','Year', 'Unemployment','Holiday_Flag', 'CPI'])].c

print(vif_data)
```

```
       feature       VIF
0        Month  4.079343
1        Store  3.304261
2  Temperature  5.127278
```

In [58]:
```
# VALIDACION DE SUPUESTOS POR NORMALIDAD DE RESIDUOS
sm.qqplot(results_3.resid, line='q')
```

Out[58]:





Como se observa existen observaciones por fuera de la linea, por lo que se aplicara la prueba de Jarque-Bera para validar la hipótesis de normalidad en residuos podría ser útil.

In [38]:
```
#IMPORTACION DE SUBMODULO DE STATSMODELS PARA APLICAR PRUEBA ESTADISTICA JARQUE-BERA
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
```

In [39]:
```
nombres = ["Jarque-Bera", "Chi^2 two-tail prob.", "Skew", "Kurtosis"]
jarque_bera = sms.jarque_bera(results_3.resid)
```

```
lzip(nombres, jarque_bera)
```

Out[39]:
```
[('Jarque-Bera', 356.9687578992922),
 ('Chi^2 two-tail prob.', 3.056462813021903e-78),
 ('Skew', -0.46054978946397146),
 ('Kurtosis', 2.2257391000725706)]
```

De acuerdo a los valores observados podemos rechazar la hipotesis nula al 95% de confianza. A continuación se va a contrastar si la media de los residuos de este modelo es 0, o muy cercano a este.

In [40]:
```
#EXTRAER LOS RESIDUOS DEL MODELO Y CALCULAR LA MEDIA DIRECTAMENTE
results_4.resid.mean()
```

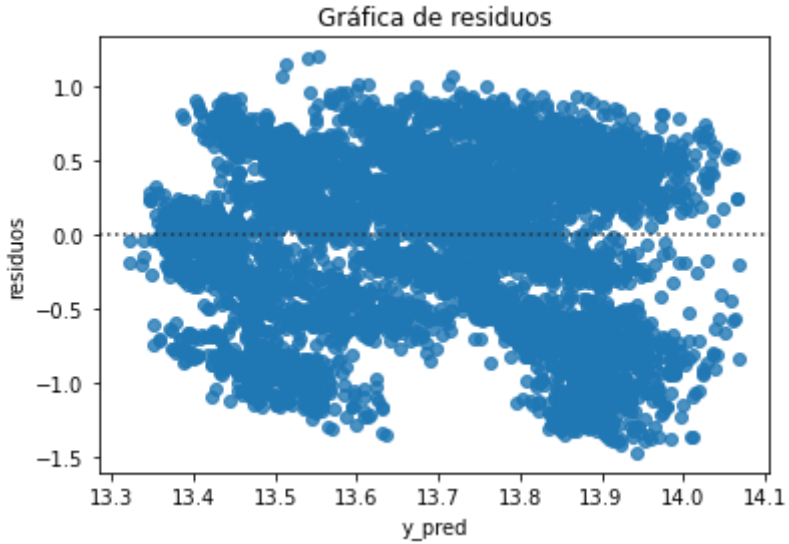
Out[40]:
```
1.4242076806007755e-15
```

La media de los residuos tiende a cero y es muy pequeña por lo que no se estaría violando este supuesto.

In [59]:
```
#VERIFICACION DE SUPUESTO DE HOMOCEDASTICIDAD EN LOS RESIDUOS
y_pred=results_3.predict()
```

In [60]:
```
sns.residplot(y_pred, results_3.resid)
plt.xlabel("y_pred")
plt.ylabel("residuos")
plt.title("Gráfica de residuos")
```

Out[60]:
```
Text(0.5, 1.0, 'Gráfica de residuos')
```



De la gráfica de residuos anterior, podríamos inferir que los residuos no formaron ningún patrón. Por lo tanto, los residuos son independientes entre sí.

In [61]:
```
#BREUSH PAGAN PARA DETERMINAR HETEROCEDASTICIDAD EN UN MODELO DE REGRESION LINEAL
nombres = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-value"]
breuschpagan = sms.het_breuschpagan(results_3.resid, results.model.exog)
lzip(nombres, breuschpagan)
```

Out[61]:
```
[('Lagrange multiplier statistic', 428.1305197121533),
 ('p-value', 1.7874202398675452e-87),
 ('f-value', 57.602825125082845),
 ('f p-value', 7.286398582271713e-91)]
```

Se tiene un p-value menor a 0.05 por lo que no hay evidencia suficiente para descartar heterocedasticidad

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

In [33]:
```
# CON EL VIF MENOR A 5 LLAMAMOS A NUESTRO MODELO DEFINITIVO
regresion_3 = ols("log_Weekly_Sales ~ Month + Store + Temperature" , data=df)
results_3 = regresion_3.fit()
```

In [34]:
```
print(results_3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       log_Weekly_Sales   R-squared:                       0.083
Model:                            OLS   Adj. R-squared:                  0.082
Method:                 Least Squares   F-statistic:                     177.5
Date:                Tue, 27 Dec 2022   Prob (F-statistic):          3.27e-110
Time:                        09:57:23   Log-Likelihood:                -4979.1
No. Observations:                5917   AIC:                             9966.
Df Residuals:                    5913   BIC:                             9993.
Df Model:                           3
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept      14.1060      0.031    455.211      0.000      14.045      14.167
Month           0.0070      0.002      3.153      0.002       0.003       0.011
Store          -0.0122      0.001    -21.961      0.000      -0.013      -0.011
Temperature    -0.0029      0.000     -7.296      0.000      -0.004      -0.002
==============================================================================
Omnibus:                      632.599   Durbin-Watson:                   0.060
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              356.969
Skew:                          -0.461   Prob(JB):                     3.06e-78
Kurtosis:                       2.226   Cond. No.                         286.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [63]:
```
#FORMA AUTOMATICA
df3=df2[df2.columns.difference(['Fuel_Price', 'Year', 'Unemployment', 'CPI', 'log_Weekly_Sales','Holiday_Flag'])]
# df3['intercepto']=1
df3=df3[[ 'Store', 'Temperature', 'Month']]
df3
```

Out[63]:

	Store	Temperature	Month
0	1	42.31	5
1	1	38.51	12
2	1	39.93	2
3	1	46.63	2
4	1	46.50	5
...	...	...	...
6430	45	64.88	9
6431	45	64.89	5
6432	45	54.47	12
6433	45	56.47	10
6434	45	58.85	10

5917 rows × 3 columns

In [64]:
```
results_3.predict()
```

Out[64]:
```
array([14.00573673, 14.0656396 , 13.99171255, ..., 13.48026684,
       13.46049492, 13.4535789 ])
```

In [65]:
```
y_pred=results_3.predict(df3)
y_pred
```

Out[65]:
```
0       14.005737
1       14.065640
2       13.991713
3       13.972243
4       13.993561
          ...
6430    13.429076
6431    13.401127
6432    13.480267
6433    13.460495
6434    13.453579
Length: 5917, dtype: float64
```

In [66]:
```
df.log_Weekly_Sales
```

Out[66]:
```
0       14.312455
1       14.311400
2       14.292966
3       14.158907
4       14.256862
          ...
6430    13.477481
6431    13.505522
6432    13.506897
6433    13.484400
6434    13.541444
Name: log_Weekly_Sales, Length: 5917, dtype: float64
```

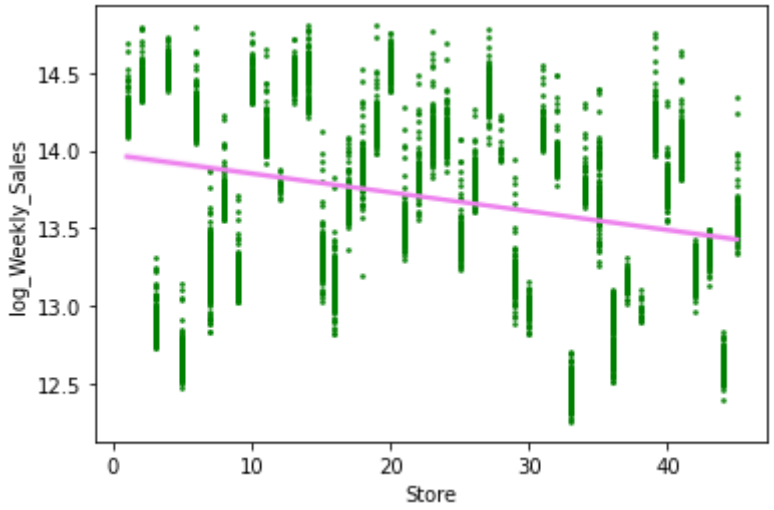
Luego de eliminar 4 variables que se encontraban correlacionadas, se ha pasado de explicar el 10,8 % al 8,3 %. Los valores predecidos y reales estan dentro de un rango aceptable.

12. 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.

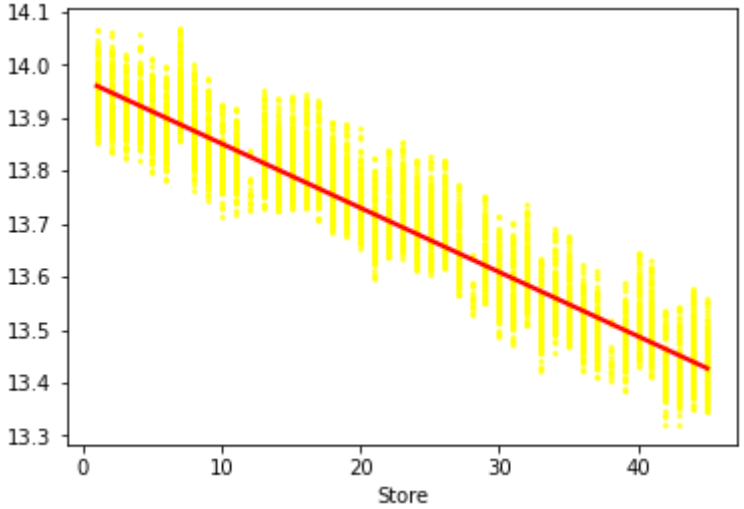
In [67]:
```
# Valores observados en color violeta
sns.regplot(x=df["Store"], y=df['log_Weekly_Sales'], scatter_kws={"color": "green", "s": 3},
                line_kws={"color": "violet"})
```

Out[67]:
```
<AxesSubplot:xlabel='Store', ylabel='log_Weekly_Sales'>
```



In [68]:
```
# Valores predecidos en rojo
sns.regplot(x=df["Store"], y=y_pred, scatter_kws={"color": "yellow", "s": 3},
                line_kws={"color": "red"})
```

Out[68]:
```
<AxesSubplot:xlabel='Store'>
```



EVALUACION DEL MODELO DE REGRESION LINEAL

In [38]:
```
#RECODIFICAMOS LAS VARIABLES CATEGORICAS CON LABEL ENCODER
var_cuantitativas = df.select_dtypes('number').columns
var_cualitativas  =df.select_dtypes('object').columns
from sklearn.preprocessing import LabelEncoder
```

In [39]:
```
labelencoder = LabelEncoder()
```

In [41]:
```
df[var_cualitativas] = df[var_cualitativas].apply(labelencoder.fit_transform)
```

In [75]:
```
X = df[df.columns.difference(['Weekly_Sales'])]
y = df.Weekly_Sales
```

In [52]:
```
from sklearn.model_selection import train_test_split
```

In [53]:
```
X_train , X_test , y_train , y_test = train_test_split(X , y , test_size = 0.20,random_state =123)
```

In [54]:
```
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))
```

```
(4733, 8)  <class 'pandas.core.frame.DataFrame'>
(4733,)          <class 'pandas.core.series.Series'>
(1184, 8)  <class 'pandas.core.frame.DataFrame'>
(1184,)          <class 'pandas.core.series.Series'>
```

In [55]:
```
from sklearn.linear_model import LinearRegression
```

In [56]:
```
modelo_regresion = LinearRegression()
modelo_regresion.fit(X_train, y_train)
```

Out[56]:
```
LinearRegression()
```

In [57]:
```
predicciones_train = modelo_regresion.predict(X_train) # LAS PREDICCIONES SOBRE LA BASE DE ENTRENAMIENTO SERAN MEJORES QUE LAS TE
predicciones_test = modelo_regresion.predict(X_test) # PREDICCIONES SOBRE DATOS QUE NO VIO
```

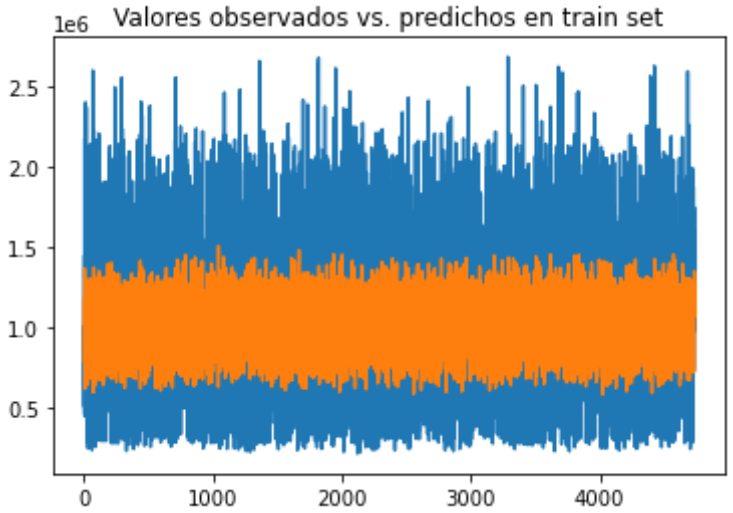
In [58]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

In [59]:
```
from sklearn.metrics import r2_score
```

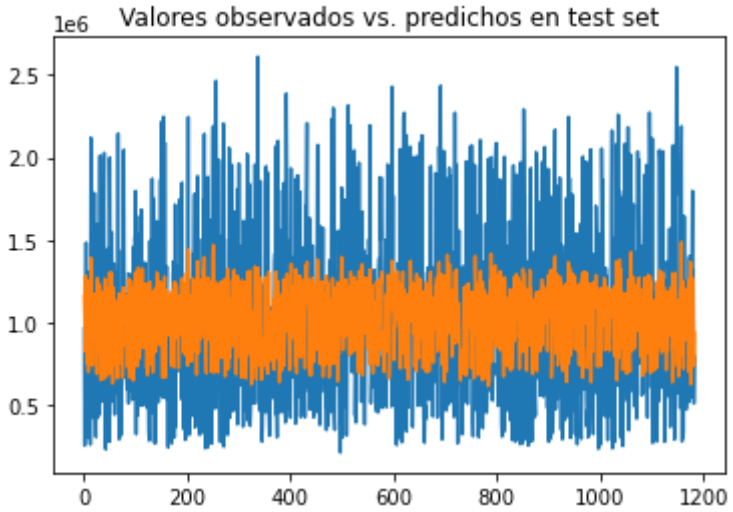
In [60]:
```
r_square_train = r2_score(y_train, predicciones_train)
r_square_test  = r2_score(y_test, predicciones_test)
print('El R^2 del subconjunto de entrenamiento es:' , r_square_train.round(2))
print('El R^2 del subconjunto de prueba es:' , r_square_test.round(2))
```

```
El R^2 del subconjunto de entrenamiento es: 0.13
El R^2 del subconjunto de prueba es: 0.13
```

In [61]:
```
# GRAFICO SOBRE LOS DATOS DE ENTRENAMIENTO
fig, ax = plt.subplots()
ax.plot(y_train.values) # VALORES OBERVADOS
ax.plot(predicciones_train) # VALORES PREDICHOS
plt.title("Valores observados vs. predichos en train set");
```



In [62]:
```
# GRAFICO SOBRE LOS DATOS DE PRUEBA
fig, ax = plt.subplots()
ax.plot(y_test.values)
ax.plot(predicciones_test)
plt.title("Valores observados vs. predichos en test set");
```



In [64]:
```
from sklearn.preprocessing import StandardScaler # ESTANDARIZAMOS LAS VARIABLES
```

In [65]:
```
sc = StandardScaler()
```

In [66]:
```
# ESTANDARIZAMOS SOLO LAS VARIABLES INDEPENDIENTES
X_train_std = sc.fit_transform(X_train) # SOLO SE APLICA EL FIT AL X_TRAIN, SACA LA STD DEL CONJUNTO DE ENTRENAMIENTO
X_test_std = sc.transform(X_test) # SOLO ESTANDARIZA SOBRE EL X_TEST, NO SOBRE EL CONJUNTO DE ENTRENAMIENTO
```

In [67]:
```
# UNA VEZ OBTENIDOS LOS CONJUNTOS DE DATOS ENTRENAMOS EL MODELO
modelo_regresion_std = LinearRegression()
modelo_regresion_std.fit(X_train_std, y_train)
```

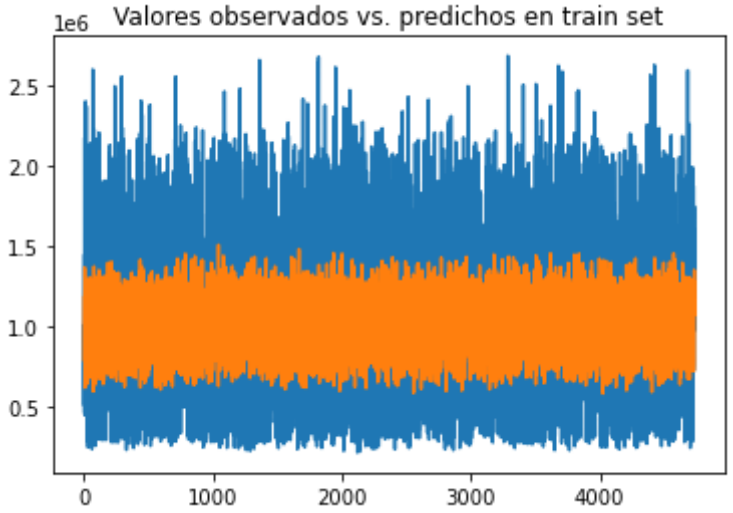
Out[67]:
```
LinearRegression()
```

In [68]:
```
# REALIZAMOS PREDICCIONES
predicciones_train_std = modelo_regresion_std.predict(X_train_std)
predicciones_test_std = modelo_regresion_std.predict(X_test_std)
```

In [69]:
```
# EVALUAMOS EL R CUADRADO
r_square_train_std = r2_score(y_train, predicciones_train_std)
r_square_test_std  = r2_score(y_test, predicciones_test_std)
print('El R^2 del subconjunto de entrenamiento es:' , r_square_train_std.round(2))
print('El R^2 del subconjunto de prueba es:' , r_square_test_std.round(2))
```

```
El R^2 del subconjunto de entrenamiento es: 0.13
El R^2 del subconjunto de prueba es: 0.13
```

In [70]:
```
# GRAFICO SOBRE LOS DATOS DE ENTRENAMIENTO ESTANDARIZADOS
fig, ax = plt.subplots()
ax.plot(y_train.values)
ax.plot(predicciones_train_std)
plt.title("Valores observados vs. predichos en train set");
```



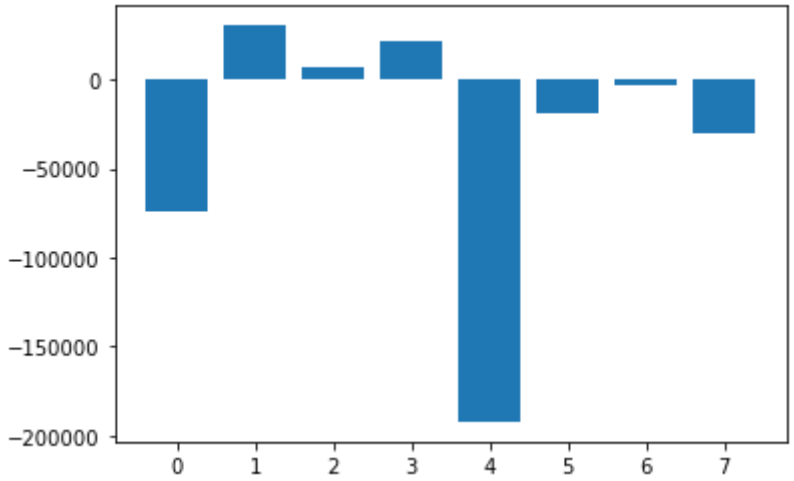
In [71]:
```
importancia = modelo_regresion_std.coef_
```

In [72]:
```
# ITERAR PARA PODER VER LOS COEFICIENTES
for i,v in enumerate(importancia):
    print('Variable explicativa No. %0d, Score: %.5f' % (i,v))
```

```
Variable explicativa No. 0, Score: -73512.17391
Variable explicativa No. 1, Score: 30496.55394
Variable explicativa No. 2, Score: 6999.50011
Variable explicativa No. 3, Score: 21947.74240
Variable explicativa No. 4, Score: -192343.18091
Variable explicativa No. 5, Score: -18946.48215
Variable explicativa No. 6, Score: -3306.86206
Variable explicativa No. 7, Score: -30395.96237
```

In [73]:
```
# Graficar la importancia o "feature importance"
plt.bar([x for x in range(len(importancia))], importancia)
```

```
plt.show()
```



De acuerdo a la grafica las variables con mayor poder explicativo son X1, X2 y X3

FINALMENTE SE ADJUNTA VALIDACION DE MODELO POR DATOS DE PANEL, EN DONDE EL RESULTADO NO ES EL ESPERADO POR LO QUE SE ELIGE EL PRIMER MODELO DE REGRESION LINEAL MULTIPLE

In [ ]:

Modelamiento con datos de panel

Importar los modules requeridos

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

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

In [20]:
```
df=pd.read_csv("Walmart.csv")
df
```

Out[20]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
0	1	05-02-2010	1643690.90	0	42.31	2.572	211.096358	8.106
1	1	12-02-2010	1641957.44	1	38.51	2.548	211.242170	8.106
2	1	19-02-2010	1611968.17	0	39.93	2.514	211.289143	8.106
3	1	26-02-2010	1409727.59	0	46.63	2.561	211.319643	8.106
4	1	05-03-2010	1554806.68	0	46.50	2.625	211.350143	8.106
...	...	...	...	...	...	...	...	...
6430	45	28-09-2012	713173.95	0	64.88	3.997	192.013558	8.684
6431	45	05-10-2012	733455.07	0	64.89	3.985	192.170412	8.667
6432	45	12-10-2012	734464.36	0	54.47	4.000	192.327265	8.667
6433	45	19-10-2012	718125.53	0	56.47	3.969	192.330854	8.667
6434	45	26-10-2012	760281.43	0	58.85	3.882	192.308899	8.667

6435 rows × 8 columns

In [23]:
```
df['Date']= pd.to_datetime(df['Date'])
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df
```

Out[23]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	Year	Month
0	1	2010-05-02	1643690.90	0	42.31	2.572	211.096358	8.106	2010	5
1	1	2010-12-02	1641957.44	1	38.51	2.548	211.242170	8.106	2010	12
2	1	2010-02-19	1611968.17	0	39.93	2.514	211.289143	8.106	2010	2
3	1	2010-02-26	1409727.59	0	46.63	2.561	211.319643	8.106	2010	2
4	1	2010-05-03	1554806.68	0	46.50	2.625	211.350143	8.106	2010	5
...	...	...	...	...	...	...	...	...	...	...
6430	45	2012-09-28	713173.95	0	64.88	3.997	192.013558	8.684	2012	9
6431	45	2012-05-10	733455.07	0	64.89	3.985	192.170412	8.667	2012	5
6432	45	2012-12-10	734464.36	0	54.47	4.000	192.327265	8.667	2012	12
6433	45	2012-10-19	718125.53	0	56.47	3.969	192.330854	8.667	2012	10
6434	45	2012-10-26	760281.43	0	58.85	3.882	192.308899	8.667	2012	10

6435 rows × 10 columns

Para simplificar el análisis limitemos las variables explicativas a 4: el año y estado en donde se recogió la información, el ingreso promedio y la tasa de crimen violento:

In [28]:
```
df=df[['Store', 'Unemployment', 'Temperature', 'CPI', 'Weekly_Sales', 'Year']]
df
```

Out[28]:

	Store	Unemployment	Temperature	CPI	Weekly_Sales	Year
0	1	8.106	42.31	211.096358	1643690.90	2010
1	1	8.106	38.51	211.242170	1641957.44	2010
2	1	8.106	39.93	211.289143	1611968.17	2010
3	1	8.106	46.63	211.319643	1409727.59	2010
4	1	8.106	46.50	211.350143	1554806.68	2010
...	...	...	...	...	...	...
6430	45	8.684	64.88	192.013558	713173.95	2012
6431	45	8.667	64.89	192.170412	733455.07	2012
6432	45	8.667	54.47	192.327265	734464.36	2012
6433	45	8.667	56.47	192.330854	718125.53	2012
6434	45	8.667	58.85	192.308899	760281.43	2012

6435 rows × 6 columns

In [29]:
```
df.describe()
```

Out[29]:

	Store	Unemployment	Temperature	CPI	Weekly_Sales	Year
count	6435.000000	6435.000000	6435.000000	6435.000000	6.435000e+03	6435.000000
mean	23.000000	7.999151	60.663782	171.578394	1.046965e+06	2010.965035
std	12.988182	1.875885	18.444933	39.356712	5.643666e+05	0.797019
min	1.000000	3.879000	-2.060000	126.064000	2.099862e+05	2010.000000
25%	12.000000	6.891000	47.460000	131.735000	5.533501e+05	2010.000000
50%	23.000000	7.874000	62.670000	182.616521	9.607460e+05	2011.000000
75%	34.000000	8.622000	74.940000	212.743293	1.420159e+06	2012.000000
max	45.000000	14.313000	100.140000	227.232807	3.818686e+06	2012.000000

In [30]:
```
df.isna().sum()
```

Out[30]:
```
Store           0
Unemployment    0
Temperature     0
CPI             0
Weekly_Sales    0
Year            0
dtype: int64
```

In [41]:
```
df
```

Out[41]:

		Store	Temperature	CPI	Weekly_Sales	Year
Year	**Unemployment**					
2010	**8.106**	1	42.31	211.096358	1643690.90	2010
	8.106	1	38.51	211.242170	1641957.44	2010
	8.106	1	39.93	211.289143	1611968.17	2010
	8.106	1	46.63	211.319643	1409727.59	2010
	8.106	1	46.50	211.350143	1554806.68	2010
...	**...**	...	...	...	...	...
2012	**8.684**	45	64.88	192.013558	713173.95	2012
	8.667	45	64.89	192.170412	733455.07	2012
	8.667	45	54.47	192.327265	734464.36	2012
	8.667	45	56.47	192.330854	718125.53	2012
	8.667	45	58.85	192.308899	760281.43	2012

6435 rows × 5 columns

Recordemos que en datos de panel, requerimos especificar los índice para nuestra base de datos. En este caso, setiaremos al año y al codificado de estado como referencia.

In [38]:
```
df=df.set_index(['Year', 'Unemployment'])
```

In [39]:
```
Year = df.index.get_level_values('Year').to_list()
df['Year'] = pd.Categorical(Year)
```

Listo, una vez que tenemos nuestra base de datos, nuestro punto de partida en este tipo de análisis siempre será un modelo simple OLS sobre nuestros datos de panel, en el cual, ignoraremos el tiempo y las características individuales, y se enfocará únicamente en las dependencias entre los individuos. Cuando obtengamos el modelo simple, debemos siempre verificar que no haya correlación o endogeneidad en los errores.

Pooled OLS

Iniciaremos con este tipo de modelo como base, ya que si se violan las condiciones especificadas previamente, los modelos de efectos fjos o efectos aleatorios serán más adecuados.

Para ello, importamos al sub-module PooledOLS y a la funcionalidad de Python que nos permite expresar el modelo con notación de fórmula.

In [40]:
```
from linearmodels import PooledOLS
import statsmodels.api as sm
```

In [42]:
```
X = sm.tools.tools.add_constant(df.Store)
y = df.Weekly_Sales
```

In [43]:
```
modelo1 = PooledOLS(y, X)
resultados_pooled_OLS = modelo1.fit(cov_type='clustered', cluster_entity=True)
```

In [44]:
```
# Store values for checking homoskedasticity graphically
predicciones_pooled_OLS = resultados_pooled_OLS.predict().fitted_values
residuos_pooled_OLS = resultados_pooled_OLS.resids
```

In [45]:
```
resultados_pooled_OLS
```

Out[45]:

PooledOLS Estimation Summary

Dep. Variable:	Weekly_Sales	**R-squared:**	0.1124
Estimator:	PooledOLS	**R-squared (Between):**	-0.0017
No. Observations:	6435	**R-squared (Within):**	0.1125
Date:	Mon, Dec 26 2022	**R-squared (Overall):**	0.1124
Time:	11:50:16	**Log-likelihood**	-9.397e+04
Cov. Estimator:	Clustered		
		F-statistic:	815.02
Entities:	3	**P-value**	0.0000
Avg Obs:	2145.0	**Distribution:**	F(1,6433)
Min Obs:	1935.0		
Max Obs:	2340.0	**F-statistic (robust):**	1.541e+04
		P-value	0.0000
Time periods:	349	**Distribution:**	F(1,6433)
Avg Obs:	18.438		
Min Obs:	4.0000		
Max Obs:	78.000		

Parameter Estimates

	Parameter	Std. Err.	T-stat	P-value	Lower CI	Upper CI
const	1.382e+06	8161.4	169.35	0.0000	1.366e+06	1.398e+06
Store	-1.457e+04	117.39	-124.12	0.0000	-1.48e+04	-1.434e+04

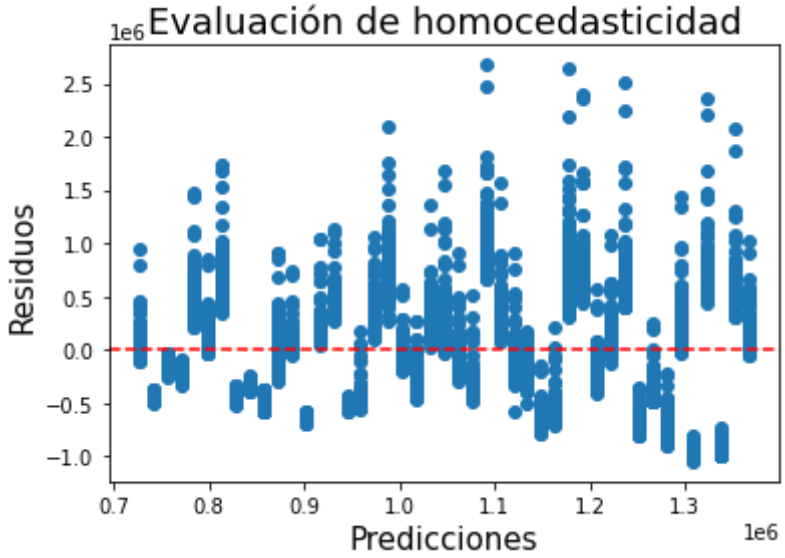
id: 0x226fbf490a0

Verificación de los supuestos de la Regresión Pooled OLS

Homocedasticidad

Para validar este supuesto, primero graficaremos los residuos y validaremos la prueba gráfica con el test estadístico de Breusch-Pagan.

In [46]:
```
fig, ax = plt.subplots()
ax.scatter(predicciones_pooled_OLS, residuos_pooled_OLS)
ax.axhline(0, color = 'r', ls = '--')
ax.set_xlabel('Predicciones', fontsize = 15)
ax.set_ylabel('Residuos', fontsize = 15)
ax.set_title('Evaluación de homocedasticidad', fontsize = 18)
plt.show()
```



De la gráfica, y como los puntos se dispersan, tenemos un indicador de varianza creciente y, por lo tanto, de heteroscedasticidad.

Comprobemos esta intuición gráfcia con el test de Breusch-Pagan:

In [47]:
```
from statsmodels.stats.diagnostic import het_breuschpagan
```

In [48]:
```
pooled_OLS_df = pd.concat([df, residuos_pooled_OLS], axis=1)
pooled_OLS_df = pooled_OLS_df.drop(['Year'], axis = 1).fillna(0)
X_ = sm.tools.tools.add_constant(df['Weekly_Sales']).fillna(0)
```

In [49]:
```
pooled_OLS_df
```

Out[49]:

Year	Unemployment	Store	Temperature	CPI	Weekly_Sales	residual
2010	**8.106**	1	42.31	211.096358	1643690.90	276165.054027
	8.106	1	38.51	211.242170	1641957.44	274431.594027
	8.106	1	39.93	211.289143	1611968.17	244442.324027
	8.106	1	46.63	211.319643	1409727.59	42201.744027
	8.106	1	46.50	211.350143	1554806.68	187280.834027
...	**...**	...	...	...	...	...
2012	**8.684**	45	64.88	192.013558	713173.95	-13229.959151
	8.667	45	64.89	192.170412	733455.07	7051.160849
	8.667	45	54.47	192.327265	734464.36	8060.450849
	8.667	45	56.47	192.330854	718125.53	-8278.379151
	8.667	45	58.85	192.308899	760281.43	33877.520849

6435 rows × 5 columns

In [50]:
```
breusch_pagan = het_breuschpagan(pooled_OLS_df.residual, X_)
labels = ['LM-Stat', 'LM p-val', 'F-Stat', 'F p-val']
print(dict(zip(labels, breusch_pagan)))
```

```
{'LM-Stat': 777.4098587376441, 'LM p-val': 4.402022605165711e-171, 'F-Stat': 883.9589818967331, 'F p-val': 3.9712145580925694e-182}
```

No- autocorrelación

In [51]:
```
from statsmodels.stats.stattools import durbin_watson
```

In [52]:
```
durbin_watson = durbin_watson(pooled_OLS_df.residual)
print(durbin_watson)
```

```
0.12359649509388257
```

Modelo de Efectos Fijos "fixed effects"

In [53]:
```
from linearmodels import PanelOLS
```

In [54]:
```
modelo_fe = PanelOLS(y, X, entity_effects = True)
resultados_fe = modelo_fe.fit()
```

In [55]:
```
resultados_fe
```

Out[55]:

PanelOLS Estimation Summary

Dep. Variable:	Weekly_Sales	**R-squared:**	0.1125
Estimator:	PanelOLS	**R-squared (Between):**	-0.0017
No. Observations:	6435	**R-squared (Within):**	0.1125
Date:	Mon, Dec 26 2022	**R-squared (Overall):**	0.1124
Time:	12:23:04	**Log-likelihood**	-9.397e+04
Cov. Estimator:	Unadjusted		
		F-statistic:	815.08
Entities:	3	**P-value**	0.0000
Avg Obs:	2145.0	**Distribution:**	F(1,6431)
Min Obs:	1935.0		
Max Obs:	2340.0	**F-statistic (robust):**	815.08
		P-value	0.0000
Time periods:	349	**Distribution:**	F(1,6431)
Avg Obs:	18.438		
Min Obs:	4.0000		
Max Obs:	78.000		

Parameter Estimates

	Parameter	Std. Err.	T-stat	P-value	Lower CI	Upper CI
const	1.382e+06	1.348e+04	102.52	0.0000	1.356e+06	1.409e+06
Store	-1.457e+04	510.37	-28.550	0.0000	-1.557e+04	-1.357e+04

F-test for Poolability: 1.2245

P-value: 0.2940

Distribution: F(2,6431)

Included effects: Entity
id: 0x226fcba0310

Modelo de Efectos Aleatorios "random effects"

In [56]:
```
from linearmodels import RandomEffects
```

In [57]:
```
modelo_re = RandomEffects(y, X)
resultados_re = modelo_re.fit()
```

In [58]:
```
resultados_re
```

Out[58]:

RandomEffects Estimation Summary

Dep. Variable:	Weekly_Sales	**R-squared:**	0.1128
Estimator:	RandomEffects	**R-squared (Between):**	-0.0003
No. Observations:	6435	**R-squared (Within):**	0.1125
Date:	Mon, Dec 26 2022	**R-squared (Overall):**	0.1124
Time:	12:23:42	**Log-likelihood**	-9.397e+04
Cov. Estimator:	Unadjusted		
		F-statistic:	818.14
Entities:	3	**P-value**	0.0000
Avg Obs:	2145.0	**Distribution:**	F(1,6433)
Min Obs:	1935.0		
Max Obs:	2340.0	**F-statistic (robust):**	815.21
		P-value	0.0000
Time periods:	349	**Distribution:**	F(1,6433)
Avg Obs:	18.438		
Min Obs:	4.0000		
Max Obs:	78.000		

Parameter Estimates

	Parameter	Std. Err.	T-stat	P-value	Lower CI	Upper CI
const	1.382e+06	1.583e+04	87.319	0.0000	1.351e+06	1.413e+06
Store	-1.457e+04	510.33	-28.552	0.0000	-1.557e+04	-1.357e+04

id: 0x226fcba0460

Test de Haussman

In [59]:
```
import numpy.linalg as la
from scipy import stats
import numpy as np
```

In [60]:
```
def hausman(fe, re):
    b = fe.params
    B = re.params
    v_b = fe.cov
    v_B = re.cov
    df = b[np.abs(b) < 1e8].size
    chi2 = np.dot((b - B).T, la.inv(v_b - v_B).dot(b - B))

    pval = stats.chi2.sf(chi2, df)
    return chi2, df, pval
```

In [61]:
```
hausman = hausman(resultados_fe, resultados_re)
```

In [62]:
```
print('chi-Squared: ' + str(hausman[0]))
print('degrees of freedom: ' + str(hausman[1]))
print('p-Value:' + str(hausman[2]))
```

```
chi-Squared: -0.0009723283814074095
degrees of freedom: 2
p-Value:1.0
```

p-value es 1. Se descarta modelo