```
import numpy as np
from statsmodels.api import Poisson
import pandas as pd
from pandas.plotting import lag_plot, autocorrelation_plot
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

Parte 1 Exploracion inicial de los datos

```
data =
pd.DataFrame(pd.read_csv("Datasets/dow+jones+index/dow_jones_index.dat
a", header=0, parse_dates=[0], index_col=[2],
infer_datetime_format=True))
```

```
C:\Users\fcmdr\AppData\Local\Temp\ipykernel_5880\2223568683.py:1:
FutureWarning: The argument 'infer_datetime_format' is deprecated and
will be removed in a future version. A strict version of it is now the
default, see https://pandas.pydata.org/pdeps/0004-consistent-to-
datetime-parsing.html. You can safely remove this argument.
  data =
pd.DataFrame(pd.read_csv("Datasets/dow+jones+index/dow_jones_index.dat
a", header=0, parse_dates=[0], index_col=[2],
infer_datetime_format=True))
C:\Users\fcmdr\AppData\Local\Temp\ipykernel_5880\2223568683.py:1:
UserWarning: Could not infer format, so each element will be parsed
individually, falling back to `dateutil`. To ensure parsing is
consistent and as-expected, please specify a format.
  data =
pd.DataFrame(pd.read_csv("Datasets/dow+jones+index/dow_jones_index.dat
a", header=0, parse_dates=[0], index_col=[2],
infer_datetime_format=True))
```

```
df =
data.drop(['quarter','stock','open','close','percent_change_volume_ove
r_last_wk','previous_weeks_volume', 'next_weeks_open',
'next_weeks_close', 'percent_change_next_weeks_price',
'days_to_next_dividend', 'percent_return_next_dividend'], axis=1)

df.groupby(df.index).count().head()
```

	high	low	volume	percent_change_price
date				
1/14/2011	30	30	30	30
1/21/2011	30	30	30	30
1/28/2011	30	30	30	30
1/7/2011	30	30	30	30
2/11/2011	30	30	30	30

```
df.index = pd.to_datetime(df.index)
df.index

DatetimeIndex(['2011-01-07', '2011-01-14', '2011-01-21', '2011-01-28',
               '2011-02-04', '2011-02-11', '2011-02-18', '2011-02-25',
               '2011-03-04', '2011-03-11',
               ...
               '2011-04-21', '2011-04-29', '2011-05-06', '2011-05-13',
               '2011-05-20', '2011-05-27', '2011-06-03', '2011-06-10',
               '2011-06-17', '2011-06-24'],
              dtype='datetime64[ns]', name='date', length=750,
freq=None)

ds = df.index.to_series()

df['MONTH'] = ds.dt.month
df['DAY_OF_WEEK'] = ds.dt.dayofweek
df['DAY'] = ds.dt.day
```

Cambiar los tipos de dato de string a float

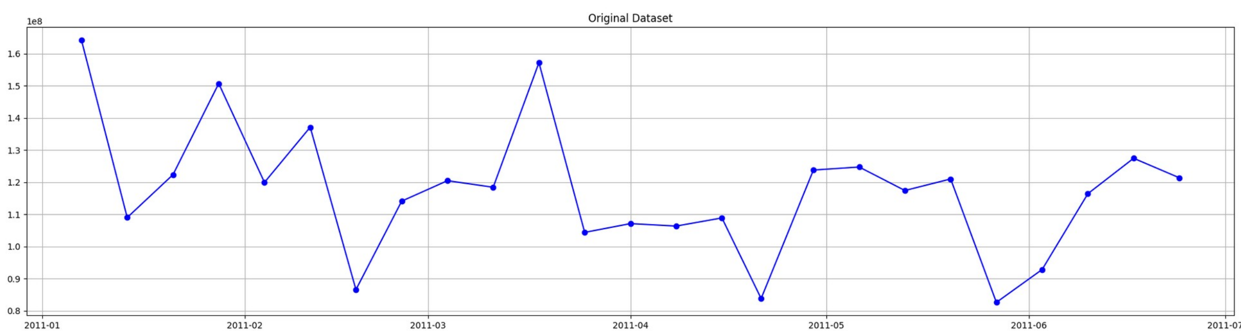
```
df = df.sort_index()
df['high'] = df['high'].replace('[\$,]', '', regex=True)
df['low'] = df['low'].replace('[\$,]', '', regex=True)

df.high = df.high.astype(float)
df.low = df.low.astype(float)

df = df.groupby(df.index).mean()

df.loc['2011']

                 high        low        volume  percent_change_price
MONTH  \
date

2011-01-07  52.394333  50.535000  1.641992e+08              0.533190
1.0
2011-01-14  52.315333  50.572000  1.090246e+08              1.322282
1.0
2011-01-21  52.934333  51.229333  1.223585e+08              0.156960
1.0
2011-01-28  53.713667  51.400333  1.507353e+08             -0.597219
1.0
2011-02-04  53.592333  51.746333  1.199585e+08              2.099038
2.0
2011-02-11  54.679333  52.763000  1.371438e+08              0.922095
2.0
2011-02-18  54.773000  53.369667  8.658673e+07              0.994382
2.0
2011-02-25  54.817667  52.432667  1.141245e+08             -1.331562
```

```
2.0
2011-03-04  54.496333  52.576667  1.204931e+08              -0.174938
3.0
2011-03-11  54.485667  52.070667  1.184469e+08              -1.104409
3.0
2011-03-18  53.392667  50.706000  1.572290e+08              -1.168365
3.0
2011-03-25  54.193667  52.366000  1.044030e+08               1.495959
3.0
2011-04-01  55.040000  53.269667  1.071276e+08               0.831334
4.0
2011-04-08  55.256000  53.895667  1.063614e+08               0.170317
4.0
2011-04-15  55.325333  53.272000  1.088940e+08              -0.714650
4.0
2011-04-21  55.420667  53.016667  8.382196e+07               1.975618
4.0
2011-04-29  56.830667  54.498000  1.238058e+08               2.451867
4.0
2011-05-06  57.190000  54.832333  1.247516e+08              -1.317455
5.0
2011-05-13  56.773333  54.709333  1.174370e+08              -0.372538
5.0
2011-05-20  56.202000  54.218333  1.210239e+08              -0.635752
5.0
2011-05-27  55.176000  53.734000  8.262061e+07               0.769748
5.0
2011-06-03  55.557000  52.807333  9.284393e+07              -3.257622
6.0
2011-06-10  54.100333  52.077333  1.164434e+08              -1.733586
6.0
2011-06-17  53.990333  51.916667  1.274691e+08               0.122466
6.0
2011-06-24  54.099667  51.989000  1.213916e+08              -0.180597
6.0

            DAY_OF_WEEK   DAY
date
2011-01-07          4.0   7.0
2011-01-14          4.0  14.0
2011-01-21          4.0  21.0
2011-01-28          4.0  28.0
2011-02-04          4.0   4.0
2011-02-11          4.0  11.0
2011-02-18          4.0  18.0
2011-02-25          4.0  25.0
2011-03-04          4.0   4.0
2011-03-11          4.0  11.0
2011-03-18          4.0  18.0
```

```
2011-03-25           4.0  25.0
2011-04-01           4.0   1.0
2011-04-08           4.0   8.0
2011-04-15           4.0  15.0
2011-04-21           3.0  21.0
2011-04-29           4.0  29.0
2011-05-06           4.0   6.0
2011-05-13           4.0  13.0
2011-05-20           4.0  20.0
2011-05-27           4.0  27.0
2011-06-03           4.0   3.0
2011-06-10           4.0  10.0
2011-06-17           4.0  17.0
2011-06-24           4.0  24.0
```

Parte 2 Visualización de datos

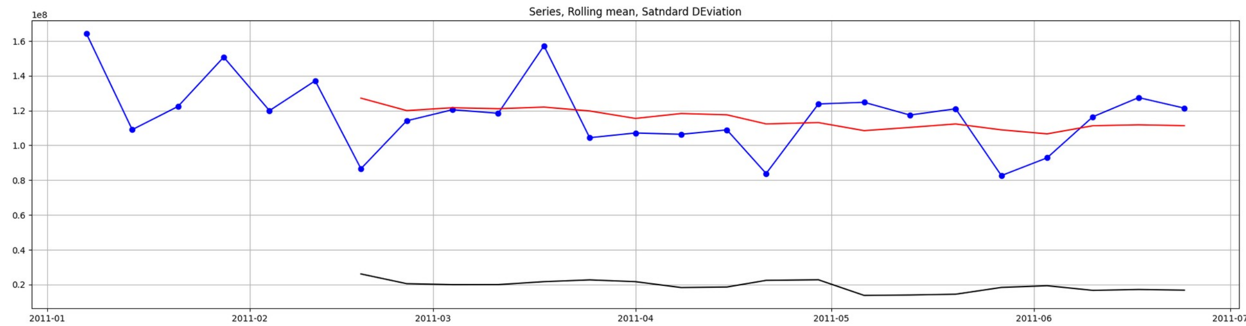
```
fig = plt.figure(figsize = (25,6))
plt.plot(df.index, df['volume'], 'bo-', label='Actual counts')
plt.title('Original Dataset')
plt.grid()
plt.show()
```



Estacionariedad

```
rolling_mean = df.rolling(7).mean()
rolling_std = df.rolling(7).std()

fig = plt.figure(figsize = (25,6))

og = plt.plot(df.index, df['volume'],'bo-', label='Original Data')
roll_mean = plt.plot(rolling_mean.index, rolling_mean['volume'],'r-',
label='Rolling mean')
roll_std = plt.plot(rolling_std.index, rolling_std['volume'],'k-',
label='Rolling std')

plt.title('Series, Rolling mean, Satndard DEviation')
# plt.legend(handles=[og, roll_mean, roll_std], loc="best")
```

```
plt.grid()
plt.show()
```



Parte 3 Prueba de Dicky-Fuller

```
adf = adfuller(df['volume'], maxlag=1)

print("T-Test (Test Statistic):",adf[0],'\n')
print("P-value:",adf[1],"\n")
print("Valores criticos (Critical value):",adf[4])

T-Test (Test Statistic): -5.26849555635069

P-value: 6.361332458733481e-06

Valores criticos (Critical value): {'1%': -3.7377092158564813, '5%': -
2.9922162731481485, '10%': -2.635746736111111}

p_value = adf[1]
t_test = adf[0]
valores_criticos = adf[4]

# Evaluar si la serie es estacionaria en base al umbral
if p_value <= 0.05:
    print('La serie temporal es estacionaria.')
else:
    print('La serie temporal no es estacionaria.')

# Evaluar si la serie es estacionaria basándose en los valores
críticos
if t_test < valores_criticos['1%']:
    print('La serie temporal es estacionaria al 1% de significancia.')
elif t_test < valores_criticos['5%']:
    print('La serie temporal es estacionaria al 5% de significancia.')
elif t_test < valores_criticos['10%']:
    print('La serie temporal es estacionaria al 10% de
significancia.')
else:
    print('La serie temporal no es estacionaria.')
```

```
La serie temporal es estacionaria.
La serie temporal es estacionaria al 1% de significancia.
```

Analizando los resultados se puede concluir que la serie es estacionaria. Esto es debido a que el valor "p-value" es menor a 0.05, rechazando la hipotesis nula por umbral, así mismo, comparando el estadistico de prueba con los valores criticos, se puede determinar que el valor crítico correspondiente al nivel de significancia del 1%, por lo tanto se rechaza la hipotesis nula.

Parte 4 Transformación y diferenciación

```
#Mascara para datos de entrenamiento y muestreo
mask = np.random.rand(len(df)) < 0.8

df_train = df[mask]
df_test = df[~mask]

print("Training data set length: ", len(df_train))
print("Testing data set length: ", len(df_test))
```

```
Training data set length:  18
Testing data set length:  7
```

```
from patsy import dmatrices

expr = """volume ~ DAY + DAY_OF_WEEK + MONTH + high + low +
percent_change_price"""

#Matrices X y Y
y_train, X_train = dmatrices(expr, df_train, return_type =
'dataframe')
y_test, X_test = dmatrices(expr, df_test, return_type='dataframe')

print(X_train.head(10))
print(y_train.head(10))
```

```
            Intercept   DAY  DAY_OF_WEEK  MONTH       high        low
\
date

2011-01-14        1.0  14.0          4.0    1.0  52.315333  50.572000

2011-02-04        1.0   4.0          4.0    2.0  53.592333  51.746333

2011-02-11        1.0  11.0          4.0    2.0  54.679333  52.763000

2011-02-18        1.0  18.0          4.0    2.0  54.773000  53.369667

2011-02-25        1.0  25.0          4.0    2.0  54.817667  52.432667

2011-03-04        1.0   4.0          4.0    3.0  54.496333  52.576667
```

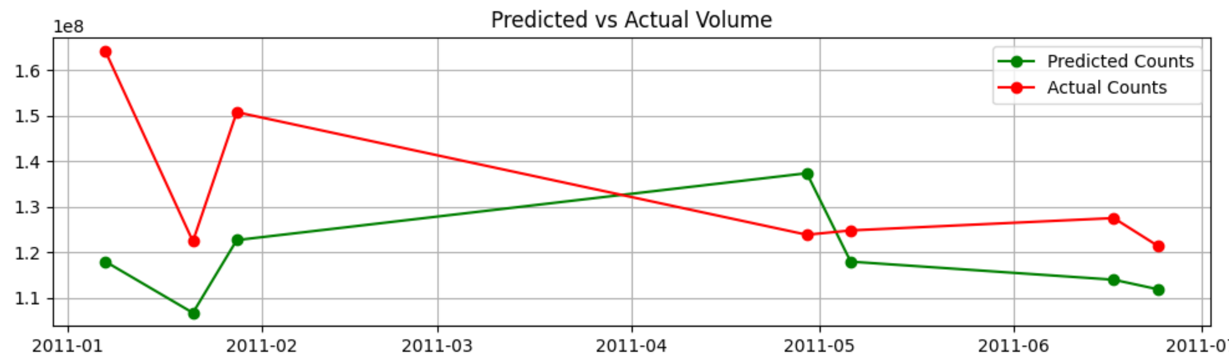
```
2011-03-11        1.0  11.0          4.0    3.0  54.485667  52.070667

2011-03-18        1.0  18.0          4.0    3.0  53.392667  50.706000

2011-03-25        1.0  25.0          4.0    3.0  54.193667  52.366000

2011-04-01        1.0   1.0          4.0    4.0  55.040000  53.269667


            percent_change_price
date
2011-01-14              1.322282
2011-02-04              2.099038
2011-02-11              0.922095
2011-02-18              0.994382
2011-02-25             -1.331562
2011-03-04             -0.174938
2011-03-11             -1.104409
2011-03-18             -1.168365
2011-03-25              1.495959
2011-04-01              0.831334
                  volume
date
2011-01-14  1.090246e+08
2011-02-04  1.199585e+08
2011-02-11  1.371438e+08
2011-02-18  8.658673e+07
2011-02-25  1.141245e+08
2011-03-04  1.204931e+08
2011-03-11  1.184469e+08
2011-03-18  1.572290e+08
2011-03-25  1.044030e+08
2011-04-01  1.071276e+08

poisson_training_results =sm.GLM(y_train, X_train,
family=sm.families.Poisson()).fit()

print(poisson_training_results.summary())

                 Generalized Linear Model Regression Results

======================================================================
========
Dep. Variable:                 volume   No. Observations:
18
Model:                            GLM   Df Residuals:
11
Model Family:                 Poisson   Df Model:
6
Link Function:                    Log   Scale:
```

```
1.0000
Method:                                  IRLS   Log-Likelihood:            -
1.2936e+07
Date:                        Fri, 17 Nov 2023   Deviance:
2.5872e+07
Time:                                11:11:38   Pearson chi2:
2.57e+07
No. Iterations:                            43   Pseudo R-squ. (CS):
1.000
Covariance Type:                    nonrobust

======================================================================
==================
                           coef    std err          z      P>|z|
[0.025      0.975]
----------------------------------------------------------------------
------------------
Intercept               16.1885      0.002   8376.937      0.000
16.185      16.192
DAY                     -0.0020      3e-06   -677.606      0.000
-0.002      -0.002
DAY_OF_WEEK              0.5156      0.000   3034.136      0.000
0.515       0.516
MONTH                   -0.0108   2.59e-05   -417.879      0.000
-0.011      -0.011
high                     0.3184      0.000   2799.024      0.000
0.318       0.319
low                     -0.3236      0.000  -3088.647      0.000
-0.324      -0.323
percent_change_price     0.0516   3.64e-05   1420.170      0.000
0.052       0.052
======================================================================
==================
```

Predicción con los datos de prueba

```
#Make predictions
poisson_predictions = poisson_training_results.get_prediction(X_test)

#Returns dataframe
predictions_summary_frame = poisson_predictions.summary_frame()

predicted_counts = predictions_summary_frame['mean']
actual_counts = y_test['volume']

fig = plt.figure(figsize=(12,3))

predicted, = plt.plot(X_test.index, predicted_counts, 'go-',
label='Predicted Counts')
actual, = plt.plot(X_test.index, actual_counts, 'ro-', label='Actual
```

```
Counts')

plt.title('Predicted vs Actual Volume')
plt.legend(handles=[predicted,actual])
plt.grid()
plt.show()
```



¿Qué información/caracteristicas puede decir de los datos originales?

```
El modelo sugiere que todas las variables son estadísticamente
significativas, con coeficientes asociados a cada una. Asi mismo, el
ajuste del modelo se evalúa mediante R-cuadrado, las iteraciones y las
estadísticas de deviance y chi2. La función de vínculo es logarítmica,
típica en modelos de Poisson. Los intervalos de confianza y la
significancia estadística de los coeficientes proporcionan información
sobre la influencia relativa de cada variable en la cuenta de eventos
representada por "volume".
```

¿Qué pase si se intenta una operación de extrapolación (Forecasting) de los datos con el modelo?

```
A no ser que se realize un ajuste previo, el modelo sugiere que no se
pueden extrapolar los datos originales, esto se debe a que las
condiciones del mundo real pueden cambiar, y el modelo podría no ser
válido para predicciones a largo plazo.
```

Parte 5 - Autocorrelación

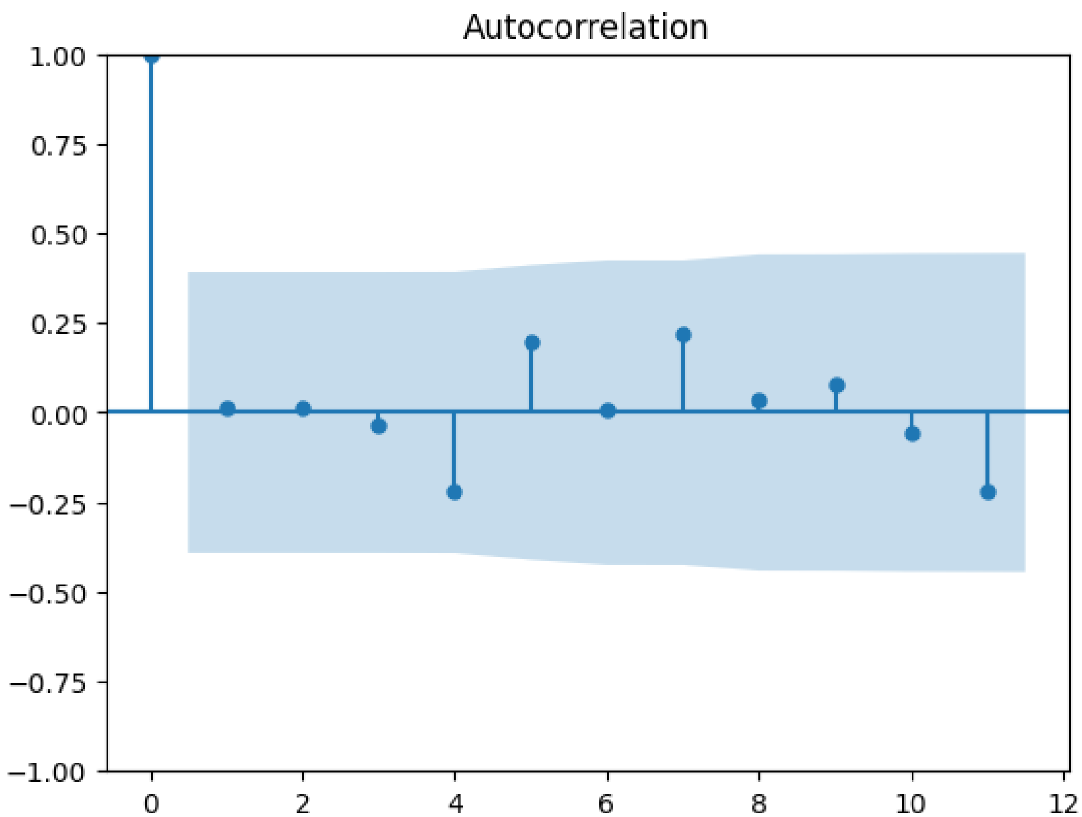
Correlaciones

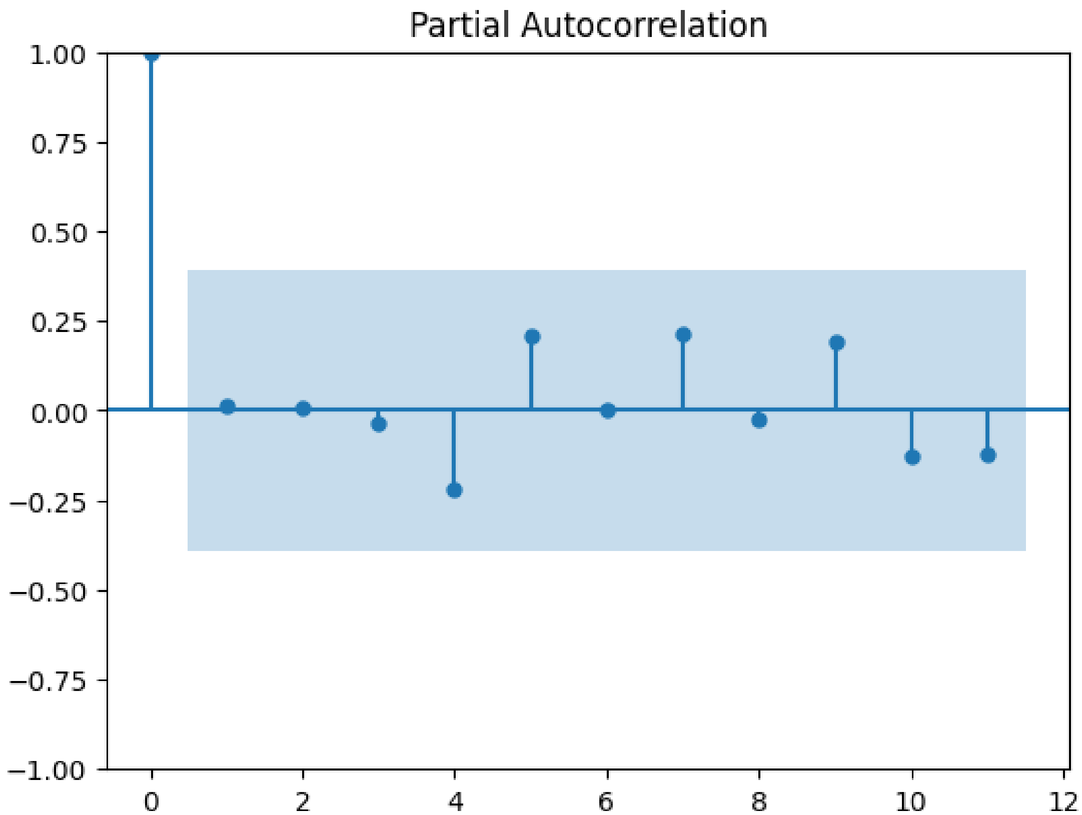
```
autocorrelation_lag1 = df['volume'].autocorr(lag=1)
print('One date Lag:',autocorrelation_lag1)

One date Lag: 0.01341270719330465

autocorrelation_lag2 = df['volume'].autocorr(lag=2)
print('Two date Lag:',autocorrelation_lag1)
autocorrelation_lag3 = df['volume'].autocorr(lag=3)
```

```
print('Three date Lag:',autocorrelation_lag1)
autocorrelation_lag6 = df['volume'].autocorr(lag=6)
print('Six date Lag:',autocorrelation_lag1)
autocorrelation_lag9 = df['volume'].autocorr(lag=9)
print('Nine date Lag:',autocorrelation_lag1)
```

```
Two date Lag: 0.01341270719330465
Three date Lag: 0.01341270719330465
Six date Lag: 0.01341270719330465
Nine date Lag: 0.01341270719330465
```

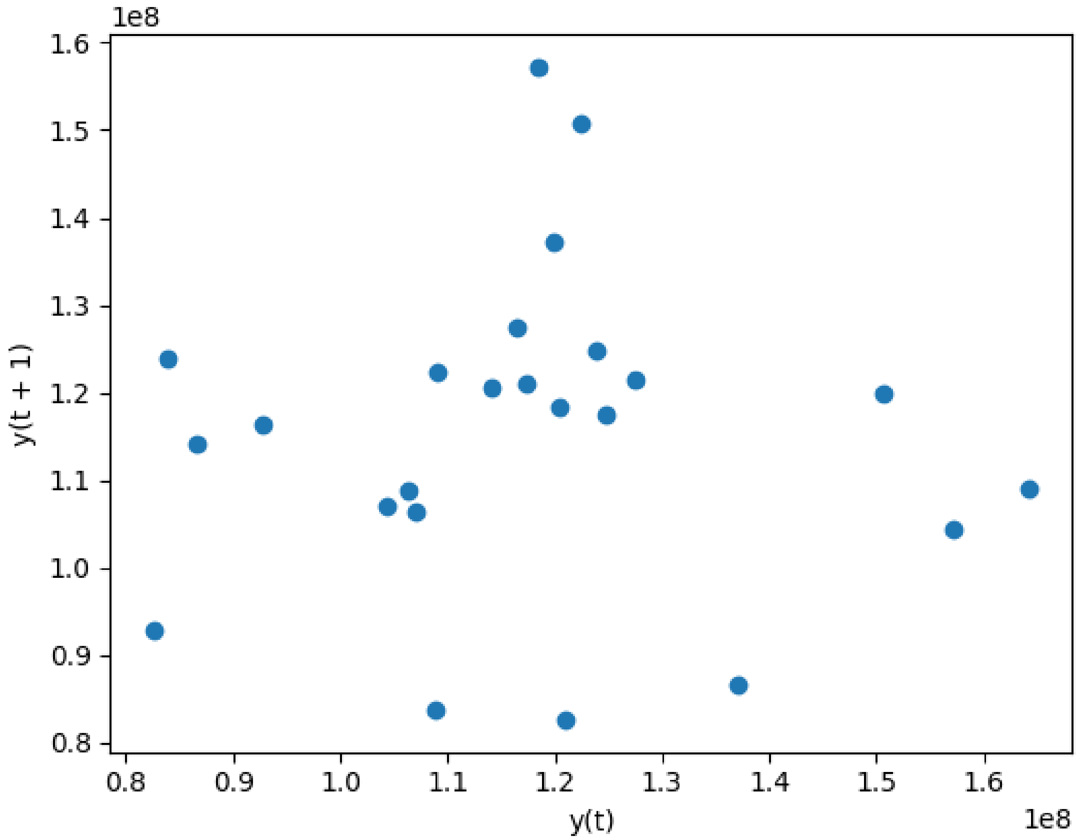
```
plot_acf(df['volume'],lags=11);
plot_pacf(df['volume'],lags=11);
```





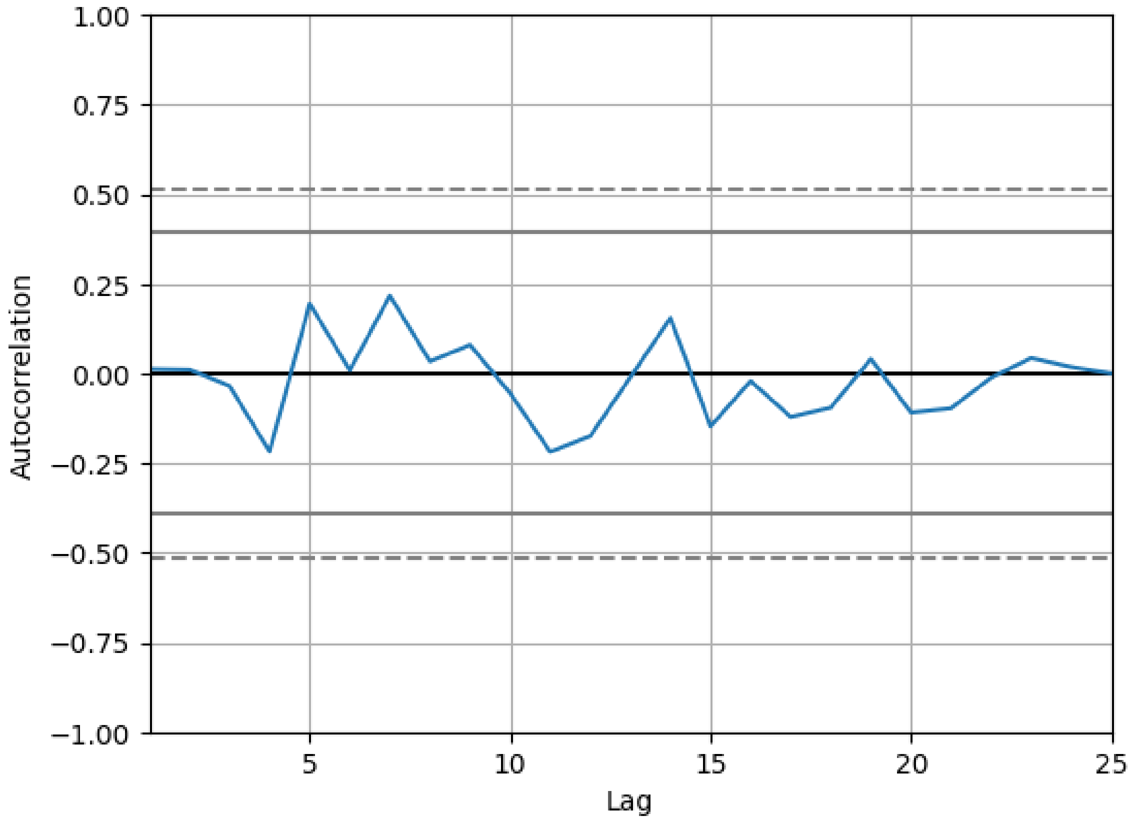
Analizando los datos se puede observar que los datos no se encuentran correlacionados

```
#Comprobación rápida de autocorrelación
lag_plot(df['volume'])
plt.show()
```



Se puede ver que los datos no siguen una tendencia, por lo que podemos decir que los datos no cuentan con correlación

```
autocorrelation_plot(df['volume'])
plt.show()
```



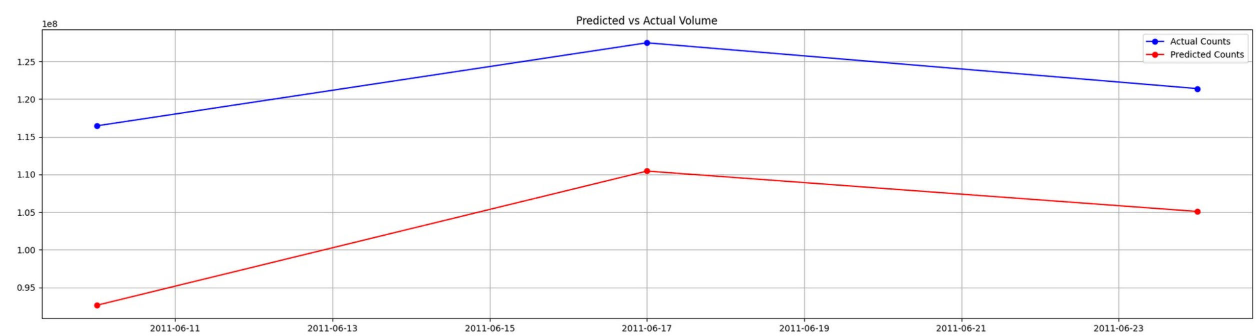
Trazando el coeficiente de correlación para la variable deseada podemos observar que los valores se encuentran adentro del intervalo de confianza

```
#Split
series = df['volume'].copy()

X = series.values

size = int(len(X) * 0.90)
train, test = X[0:size], X[size:len(X)]

print("Test size:", len(test))

ind_train, ind_test = df.index[0:size], df.index[size:len(X)]
```

```
Test size: 3
```

```
from statsmodels.tsa.ar_model import AutoReg

#train autoreg
model = AutoReg(train, lags=10)
model_fit = model.fit()
print("Coefficients:", model_fit.params)
```

```
Coefficients: [-1.84806185e+07 -6.01201520e-01 -4.01596954e-01 -
3.38436130e-01
 -2.86989455e-01 -4.07050422e-02  3.61230341e-01  4.90457343e-01
  4.83198557e-01  8.20572484e-01  5.05180658e-01]

predictions = model_fit.predict(start=len(train),
                                end = len(train)+len(test)-1,
                                dynamic = False)

for i in range(len(predictions)):
    print('Predicted = %f expected = %f' % (predictions[i], test[i]))

Predicted = 92628168.541208 expected = 116443430.933333
Predicted = 110446027.732360 expected = 127469133.966667
Predicted = 105083703.899691 expected = 121391559.133333

fig = plt.figure(figsize=(25,6))

actual, = plt.plot(ind_test, test, 'bo-', label='Actual Counts')
predicted, = plt.plot(ind_test, predictions, 'ro-', label='Predicted
Counts')

plt.title('Predicted vs Actual Volume')
plt.legend(handles=[actual, predicted])
plt.grid()
plt.show()
```




```
train_history = list(train)
predictions = list()

for t in range(len(test)):
    model = AutoReg(train_history, lags = 2)
    model_fit = model.fit()

    y_hat = model_fit.forecast()[0]
    predictions.append(y_hat)

    y_real = test[t]
    train_history.append(y_real)
```

```
    print('Predicted=%f, expected=%f' % (y_hat, y_real))
```

```
Predicted=111362157.233486, expected=116443430.933333
Predicted=114110957.956768, expected=127469133.966667
Predicted=116255934.450556, expected=121391559.133333
```

```
from math import sqrt
from sklearn.metrics import mean_squared_error

rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' %rmse)
```

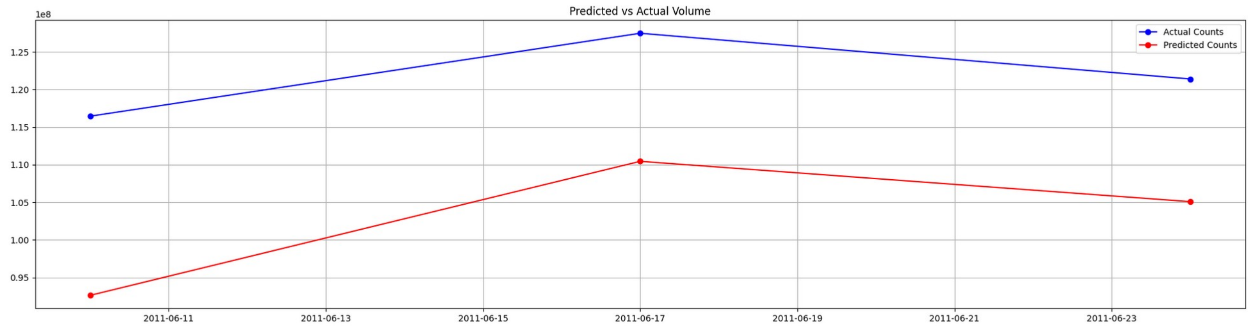
```
Test RMSE: 8768026.186
```

Predicción continua

```
# train autoreg
model = AutoReg(train, lags=10)
model_fit = model.fit()
print("Coefficients:", model_fit.params)

# Ajusta el parámetro 'end' para extender el rango de predicciones a
largo plazo
start = len(train)
end = len(train) + len(test) - 1
predictions = model_fit.predict(start=start, end=end, dynamic=False)

# Imprime las predicciones
for i in range(len(predictions)):
    print('Predicted = %f, Expected = %f' % (predictions[i], test[i]))

# Calcula el error (por ejemplo, el error cuadrático medio)
mse = mean_squared_error(test, predictions)
print("Mean Squared Error:", mse)


fig = plt.figure(figsize=(25,6))
actual, = plt.plot(ind_test, test, 'bo-', label='Actual Counts')
predicted, = plt.plot(ind_test, predictions, 'ro-', label='Predicted
Counts')

plt.title('Predicted vs Actual Volume')
plt.legend(handles=[actual, predicted])
plt.grid()
plt.show()
```

```
Coefficients: [-1.84806185e+07 -6.01201520e-01 -4.01596954e-01 -
3.38436130e-01
 -2.86989455e-01 -4.07050422e-02  3.61230341e-01  4.90457343e-01
  4.83198557e-01  8.20572484e-01  5.05180658e-01]
```

```
Predicted = 92628168.541208, Expected = 116443430.933333
Predicted = 110446027.732360, Expected = 127469133.966667
Predicted = 105083703.899691, Expected = 121391559.133333
Mean Squared Error: 374299670330568.7
```



No hay diferencia entre la predicción a corto plazo y la continua, esto se puede deber a que no hay muchos valores a predecir

Parte 6 Modelo ARIMA

```
from statsmodels.tsa.arima.model import ARIMA

model = ARIMA(series, order=(5,1,0))
model_fit = model.fit()
print('Coefficients:\n%s' % model_fit.params)

Coefficients:
ar.L1     -4.416974e-01
ar.L2     -1.385397e-01
ar.L3     -8.454548e-02
ar.L4     -2.324138e-01
ar.L5     -9.578026e-03
sigma2     3.271982e+14
dtype: float64

C:\Users\fcmdr\AppData\Roaming\Python\Python39\site-packages\
statsmodels\tsa\base\tsa_model.py:473: ValueWarning: A date index has
been provided, but it has no associated frequency information and so
will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\fcmdr\AppData\Roaming\Python\Python39\site-packages\
statsmodels\tsa\base\tsa_model.py:473: ValueWarning: A date index has
been provided, but it has no associated frequency information and so
will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\fcmdr\AppData\Roaming\Python\Python39\site-packages\
statsmodels\tsa\base\tsa_model.py:473: ValueWarning: A date index has
been provided, but it has no associated frequency information and so
```

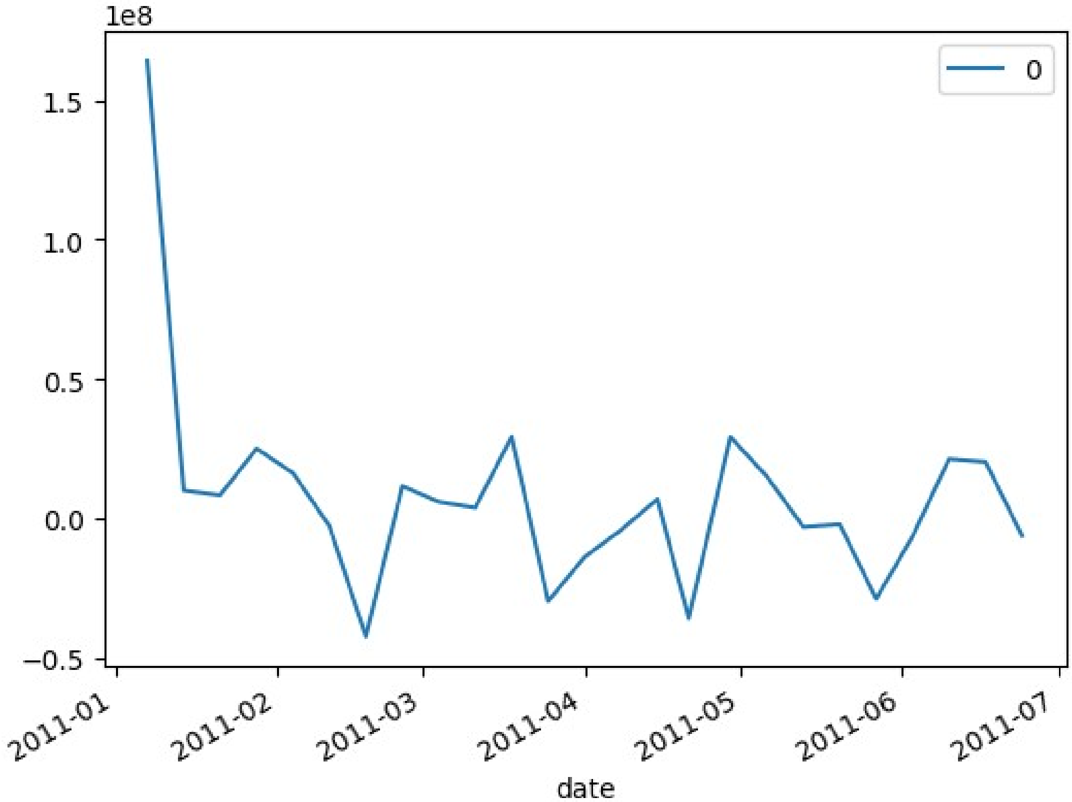
```
will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
```

```
print(model_fit.summary())
```

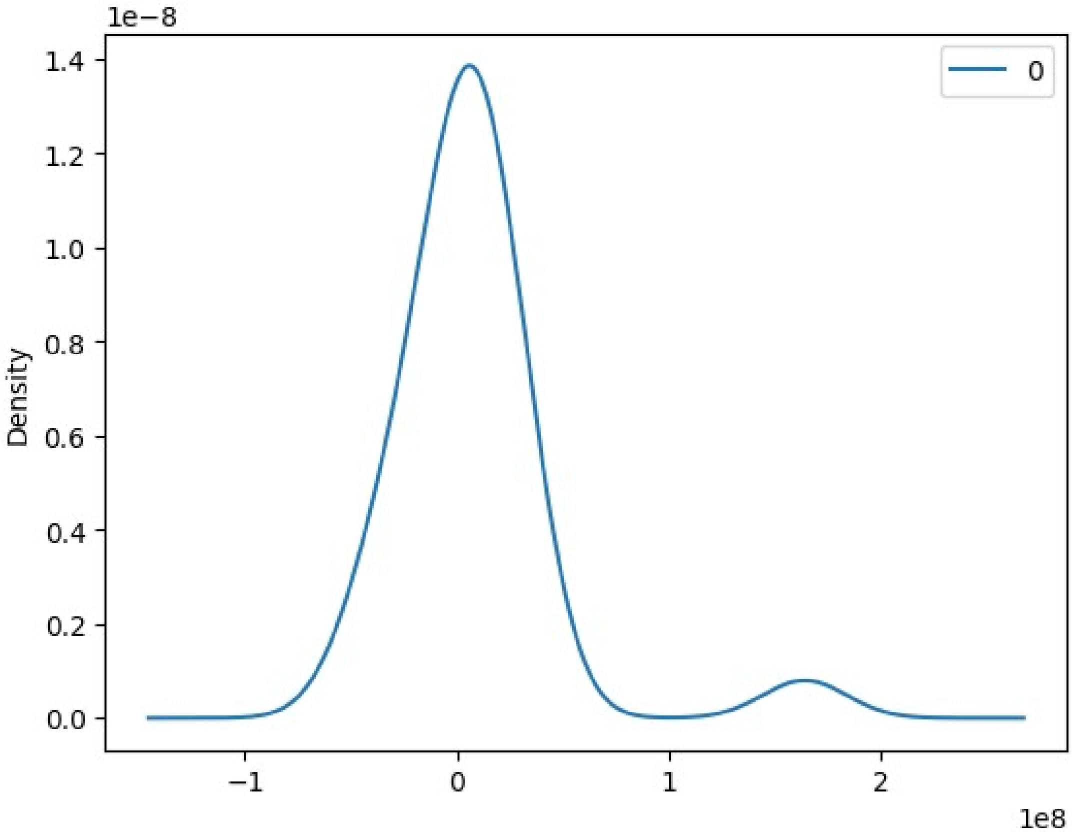
```
                              SARIMAX Results

==============================================================================
========
Dep. Variable:                 volume   No. Observations:
25
Model:                 ARIMA(5, 1, 0)   Log Likelihood
-437.205
Date:                Fri, 17 Nov 2023   AIC
886.410
Time:                        12:15:28   BIC
893.478
Sample:                             0   HQIC
888.285
                                 - 25

Covariance Type:                  opg

==============================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
--------
ar.L1         -0.4417      0.163     -2.703      0.007      -0.762
-0.121
ar.L2         -0.1385      0.227     -0.611      0.541      -0.583
0.306
ar.L3         -0.0845      0.130     -0.651      0.515      -0.339
0.170
ar.L4         -0.2324      0.136     -1.715      0.086      -0.498
0.033
ar.L5         -0.0096      0.137     -0.070      0.944      -0.278
0.259
sigma2      3.272e+14   1.03e-16   3.19e+30      0.000    3.27e+14
3.27e+14
==============================================================================
=============
Ljung-Box (L1) (Q):                   0.08   Jarque-Bera (JB):
1.72
Prob(Q):                              0.78   Prob(JB):
0.42
Heteroskedasticity (H):               0.67   Skew:
-0.63
Prob(H) (two-sided):                  0.59   Kurtosis:
```

```
2.64
===============================================================================
==============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 1.29e+46. Standard errors may be unstable.
```

```
residuals = pd.DataFrame(model_fit.resid)
residuals.plot()
plt.show()
```




```
residuals.plot(kind='kde')
plt.show()
```



Comparación con AR

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

best_alc = float("inf")
best_order = None

for p in range(1,20):
    try:
        model = ARIMA(train, order=(p,0,0))
        model_fit = model.fit()
        aic = model_fit.aic
        print(f"AR({p}): AIC ) {aic:.2f}")

        if aic < best_alc:
            best_alc = aic
            best_order = (p,0,0)

    except Exception as e:
        print(f"Error for AR({p}): {e}")

print(f"\nBest AR order: {best_order} with ALC: {best_alc:.2f}")
```

```
AR(1): AIC ) 811.16
AR(2): AIC ) 812.95
AR(3): AIC ) 814.77
AR(4): AIC ) 816.07
AR(5): AIC ) 816.73
AR(6): AIC ) 818.37
AR(7): AIC ) 819.69
AR(8): AIC ) 821.15
AR(9): AIC ) 821.69
AR(10): AIC ) 823.90
AR(11): AIC ) 69.05
AR(12): AIC ) 28.00
AR(13): AIC ) 6650455044541.46
AR(14): AIC ) 1107904293807.79
AR(15): AIC ) 2541609085586.45
AR(16): AIC ) 718533214510.83
AR(17): AIC ) 358933641129.67
AR(18): AIC ) 197155704356.24
AR(19): AIC ) 164269834228.16

Best AR order: (12, 0, 0) with ALC: 28.00
```

¿En que situaciones cree que seria mejor utilizar un modelo AR o un ARIMA?

```
 Si se esta trabajando con datos que muestran dependencia temporal,
creo que un modelo AR funcionaria perfectamente, no obstante, si los
datos cuentan con tendencia y estacionalidad creo que el modelo ARIMA
seria mas adecuado
```