Actividad 3: Series de Tiempo

Mario Alberto Castañeda Martínez - A01640152

Victor Hugo Arreola Elenes - A01635682

Luis Manuel Orozco Yáñez - A01707822

Fernando Ojeda Marín - A01639252

Se comienza por importar las librerías necesarias:

```
from google.colab import drive
drive.mount('/content/drive')
%cd "/content/drive/MyDrive/CONCENTRACION AI/data"
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
/content/drive/MyDrive/CONCENTRACION AI/data
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import  scipy.stats as stats
import datetime
```

Exploración inicial de datos:

```
df = pd.read_csv('dow_jones_index.data')
```

```
df.shape
```

```
(750, 16)
```

```
df.head()
```

```
   quarter stock       date    open    high     low   close     volume  \
0        1    AA   1/7/2011  $15.82  $16.72  $15.78  $16.42  239655616

1        1    AA  1/14/2011  $16.71  $16.71  $15.64  $15.97  242963398

2        1    AA  1/21/2011  $16.19  $16.38  $15.60  $15.79  138428495

3        1    AA  1/28/2011  $15.87  $16.63  $15.82  $16.13  151379173

4        1    AA   2/4/2011  $16.18  $17.39  $16.18  $17.14  154387761
```

```
   percent_change_price  percent_change_volume_over_last_wk  \
0               3.79267                                 NaN
1              -4.42849                            1.380223
2              -2.47066                          -43.024959
3               1.63831                            9.355500
4               5.93325                            1.987452

   previous_weeks_volume next_weeks_open next_weeks_close  \
0                    NaN          $16.71           $15.97
1            239655616.0          $16.19           $15.79
2            242963398.0          $15.87           $16.13
3            138428495.0          $16.18           $17.14
4            151379173.0          $17.33           $17.37

   percent_change_next_weeks_price  days_to_next_dividend  \
0                        -4.428490                     26
1                        -2.470660                     19
2                         1.638310                     12
3                         5.933250                      5
4                         0.230814                     97

   percent_return_next_dividend
0                      0.182704
1                      0.187852
2                      0.189994
3                      0.185989
4                      0.175029
```

Se eliminan las columnas que no se utilizarán:

```
df = df.drop(['quarter', 'stock', 'open', 'close',
'percent_change_volume_over_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)
```

Se eliminan los signos $ para que las columnas sean numéricas:

```
df['high'] = pd.to_numeric(df['high'].str.replace('$', ''),
errors='coerce')
df['low'] = pd.to_numeric(df['low'].str.replace('$', ''),
errors='coerce')
```

Debido a que el dataset tiene múltiples muestras de las mismas fechas, lo que se hará para tener fechas únicas, es que se hará un promedio para cada fecha repetida y al final quedará una sola fecha la cual tendrá el promedio de las muestras repetidas.

Se utilizará gruopby en la columna date para hacer un promedio de las muestras repetidas.

```
#df = df.drop_duplicates(subset='date', keep='first')
df = df.groupby('date').mean()

df.head()
```

	high	low	volume	percent_change_price
date				
1/14/2011	52.315333	50.572000	1.090246e+08	1.322282
1/21/2011	52.934333	51.229333	1.223585e+08	0.156960
1/28/2011	53.713667	51.400333	1.507353e+08	-0.597219
1/7/2011	52.394333	50.535000	1.641992e+08	0.533190
2/11/2011	54.679333	52.763000	1.371438e+08	0.922095

Se le da un formato datetime a las fechas del dataset, además de que se agregan al index de éste y se ordenan de forma ascendente:

```
#df['date'] = pd.to_datetime(df['date'], format='%m/%d/%Y')
df.index = pd.to_datetime(df.index, format='%m/%d/%Y')
#df.set_index('date', inplace=True)

df = df.sort_index()

#df['high'] = pd.to_numeric(df['high'].str.replace('$', ''),
errors='coerce')
#df['low'] = pd.to_numeric(df['low'].str.replace('$', ''),
errors='coerce')
```

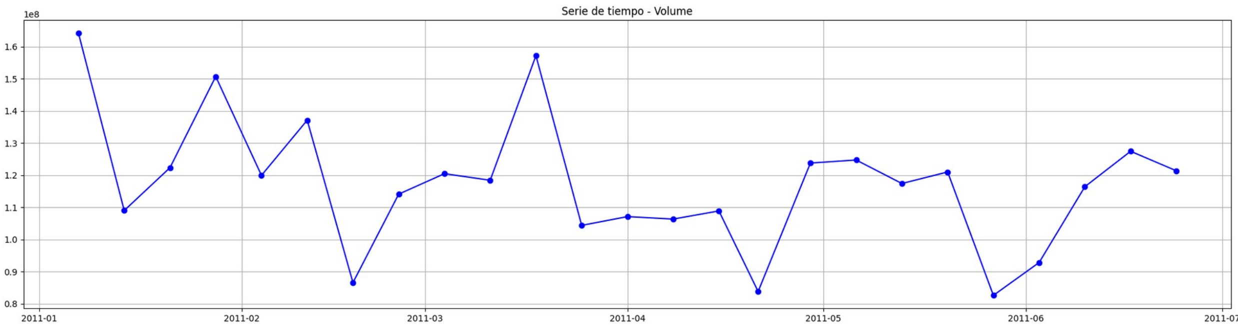
Se observa el dataset sin fechas repetidas, con formato datetime y con sus columnas numéricas:

```
df.head()
```

	high	low	volume	percent_change_price
date				
2011-01-07	52.394333	50.535000	1.641992e+08	0.533190
2011-01-14	52.315333	50.572000	1.090246e+08	1.322282
2011-01-21	52.934333	51.229333	1.223585e+08	0.156960
2011-01-28	53.713667	51.400333	1.507353e+08	-0.597219
2011-02-04	53.592333	51.746333	1.199585e+08	2.099038

Se visualiza una serie de tiempo para la variable 'volume', quitando los valores duplicados del dataset, se queda con un total de 25 datos:

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

plt.plot(df.index, df['volume'], 'bo-', label='volume')
plt.title('Serie de tiempo - Volume')
plt.grid()
plt.show()
```

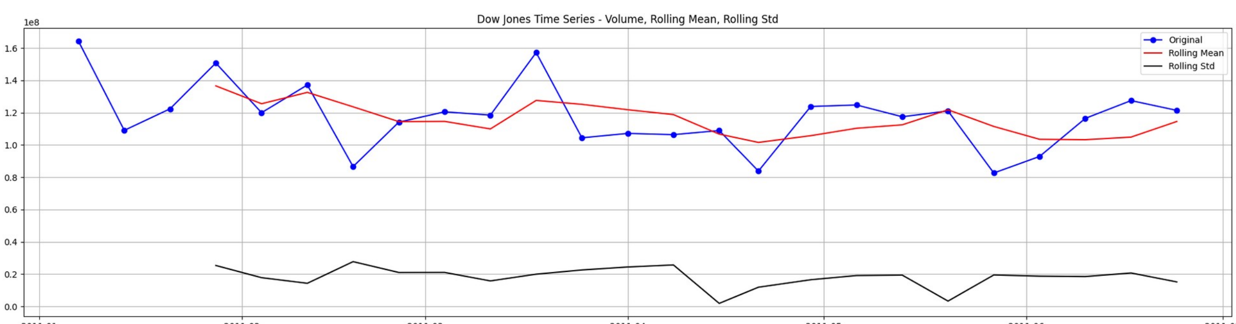


Después de preparar la base de datos y darle un formato adecuado, se procederá realizar una verificación de estacionariedad de forma visual y por dickey-fuller.

Verificación de Estacionariedad (Visual y Dickey-Fuller)

Verificación Visual de Estacionariedad:

```
rolling_mean = df.rolling(4).mean()
rolling_std = df.rolling(4).std()

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

original, =plt.plot(df.index, df['volume'], 'bo-', label='Original')
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('Dow Jones Time Series - Volume, Rolling Mean, Rolling Std')
plt.legend(handles=[original, roll_mean, roll_std], loc='best')
plt.grid()
plt.show()
```



Al visualizar la serie de tiempo de los datos 'volume' en el dataset de Dow Jones, se observa que tanto en la media móvil como la desviación estándar móvil se mantienen constantes en la serie de tiempo, por lo que de forma visual se puede decir que la serie de tiempo es estacionaria.

Prueba Dicky-Fuller para Estacionariedad:

```
from statsmodels.tsa.stattools import adfuller

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

print('T-test (Test Stadistic): ', adf[0], '\n')
print('P-value: ', adf[1], '\n')
print('Valores criticos: ', adf[4])
```

```
T-test (Test Stadistic):  -5.26849555635069

P-value:  6.361332458733481e-06

Valores criticos:  {'1%': -3.7377092158564813, '5%': -
2.9922162731481485, '10%': -2.635746736111111}
```

De acuerdo con la prueba Dicky-Fuller realizada, se observa que el valor de prueba T-Test (-5.26) es menor que todos los valores críticos en todos los porcentajes, por lo que esto significa que la prueba confirma que la serie de tiempo que se está usando es estacionaria.

Cabe destacar que al tener una serie de tiempo estacionaria, no existe la necesidad de hacer una transformación.

Creación de modelo de regresión de Poisson

Creación de conjunto de entrenamiento y de prueba:

```
mask = np.random.rand(len(df)) < 0.8

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

print('Datos de entrenamiento: ', len(df_train))
print('Datos de prueba: ', len(df_test))
```

```
Datos de entrenamiento:  18
Datos de prueba:  7
```

Se genera el modelo de regresión:

```
from patsy import dmatrices

expr = """volume ~ high + low + percent_change_price"""

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())
print(y_train.head())
```

```
            Intercept       high        low  percent_change_price
date
```

```
2011-01-07        1.0  52.394333  50.535000             0.533190
2011-01-14        1.0  52.315333  50.572000             1.322282
2011-01-21        1.0  52.934333  51.229333             0.156960
2011-01-28        1.0  53.713667  51.400333            -0.597219
2011-02-11        1.0  54.679333  52.763000             0.922095
                  volume
date
2011-01-07  1.641992e+08
2011-01-14  1.090246e+08
2011-01-21  1.223585e+08
2011-01-28  1.507353e+08
2011-02-11  1.371438e+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:
14
Model Family:                 Poisson   Df Model:
3
Link Function:                    Log   Scale:
1.0000
Method:                          IRLS   Log-Likelihood:            -
2.5688e+07
Date:                Fri, 10 Nov 2023   Deviance:
5.1376e+07
Time:                        14:10:39   Pearson chi2:
5.01e+07
No. Iterations:                    11   Pseudo R-squ. (CS):
1.000
Covariance Type:            nonrobust

======================================================================
==================
                         coef    std err          z      P>|z|
[0.025      0.975]
----------------------------------------------------------------------
------------------
Intercept             22.8005      0.001   2.42e+04      0.000
22.799      22.802
high                   0.1188   6.23e-05   1908.973      0.000
0.119       0.119
```

```
low                           -0.2039   6.34e-05  -3216.128      0.000
-0.204      -0.204
percent_change_price           0.0158   1.85e-05    854.266      0.000
0.016       0.016
======================================================================
==================
```

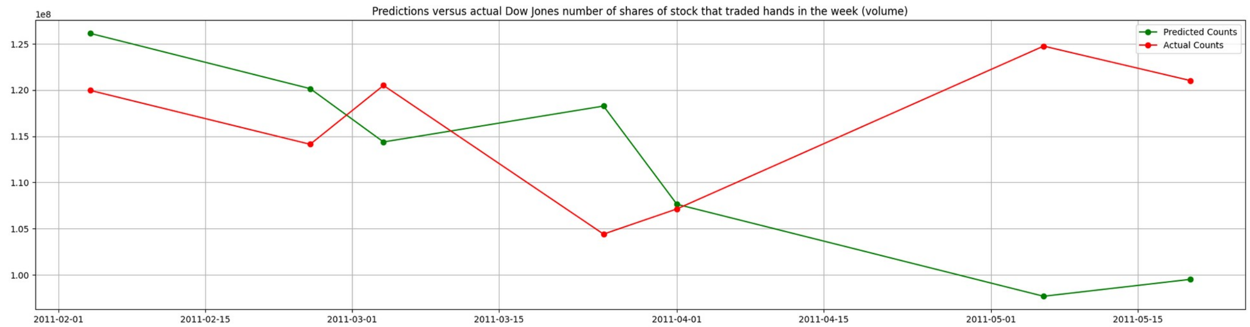
Predicción de datos usando el modelo:

```
poisson_predictions = poisson_training_results.get_prediction(x_test)

predictions_summary = poisson_predictions.summary_frame()
print(predictions_summary)
```

```
                    mean      mean_se  mean_ci_lower  mean_ci_upper
date
2011-02-04  1.261313e+08  5574.601381   1.261204e+08   1.261422e+08
2011-02-25  1.201498e+08  3923.444835   1.201421e+08   1.201575e+08
2011-03-04  1.143761e+08  2783.501565   1.143707e+08   1.143816e+08
2011-03-25  1.182628e+08  3921.705027   1.182551e+08   1.182705e+08
2011-04-01  1.076340e+08  3315.924227   1.076275e+08   1.076405e+08
2011-05-06  9.767610e+07  5646.535466   9.766504e+07   9.768717e+07
2011-05-20  9.950495e+07  4276.700400   9.949656e+07   9.951333e+07
```

Visualización de datos reales y datos predichos por el modelo:

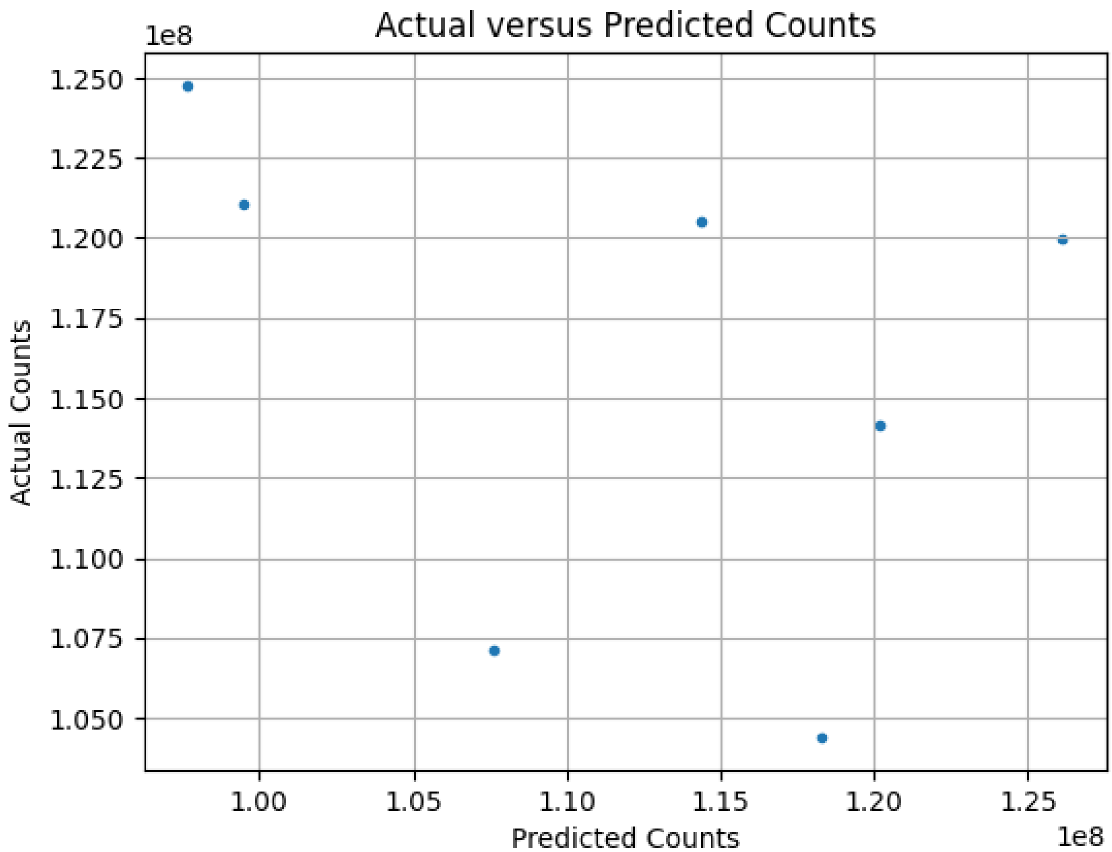
```
predicted_counts = predictions_summary['mean']
actual_counts= y_test['volume']

#Grafica:
fig = plt.figure(figsize=(25,6))

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('Predictions versus actual Dow Jones number of shares of
stock that traded hands in the week (volume)')
plt.legend(handles = [predicted, actual])
plt.grid()
plt.show()
```



En la gráfica anterior, se observa una comparación entre los valores actuales y los predichos por el modelo y se puede describir que en esta predicción de datos sí existe cierta precisión con los datos reales. Se puede observar que en 4 de los puntos mostrados se tiene precisión cercana a los datos reales, mientras que en 3 puntos sí existe un error mayor a comparación de los otros 4.

Gráfica del error del modelo:

```
fig = plt.figure()
plt.scatter(x=predicted_counts, y=actual_counts, marker='.')

plt.title('Actual versus Predicted Counts')
plt.xlabel('Predicted Counts')
plt.ylabel('Actual Counts')
plt.grid()
plt.show()
```



Se observa que en la gráfica de errores, se alcanza a ver una ligera tendencia de los errores, lo que puede significar que el modelo necesita mayor precisión.

Considero que esto puede deberse a que el modelo cuenta con pocos datos de entrenamiento y de prueba. Es posible que con un dataset original con una mayor cantidad de datos, se pueda obtener un modelo con una mayor precisión entre datos reales y predichos.

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

Considero que en caso de que se realice un forecast para los datos, sería necesario considerar la opción de aumentar el número de datos en el dataset para incrementar las posibilidades de una mayor precisión al realizar el pronóstico. De igual forma, sería útil hacer un análisis de autocorrelación y autocorrelación parcial para finalmente probar con un modelo AR y otro ARIMA para ver con qué modelo se tiene mayor precisión al pronosticar.

A partir de la generación del modelo de regresión de Poisson, se puede decir que debido a que existe una cierta precisión en el modelo, en caso de que se realice una predicción, los datos predichos tendrán un cierto grado de precisión, como se menciona en el párrafo anterior, sería necesaria la opción de aumentar o agregar datos para realizar una predicción lo más certera posible.

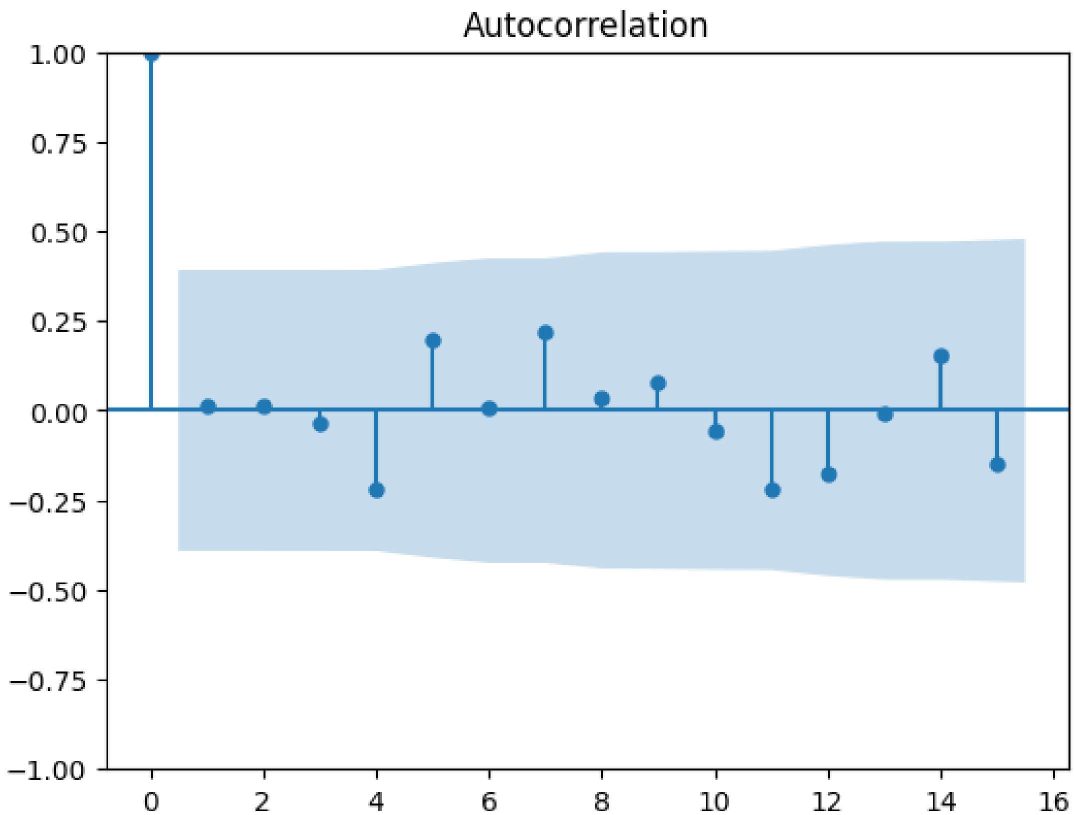
Análisis de Autocorrelación y Autocorrelación Parcial

Función de Autocorrelación

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

autocorr_lag2 = df['volume'].autocorr(lag=2)
print('Two date lag: ', autocorr_lag2)

autocorr_lag3 = df['volume'].autocorr(lag=3)
print('Three date lag: ', autocorr_lag3)

autocorr_lag6 = df['volume'].autocorr(lag=6)
print('Six date lag: ', autocorr_lag6)

autocorr_lag9 = df['volume'].autocorr(lag=9)
print('Nine date lag: ', autocorr_lag9)
```

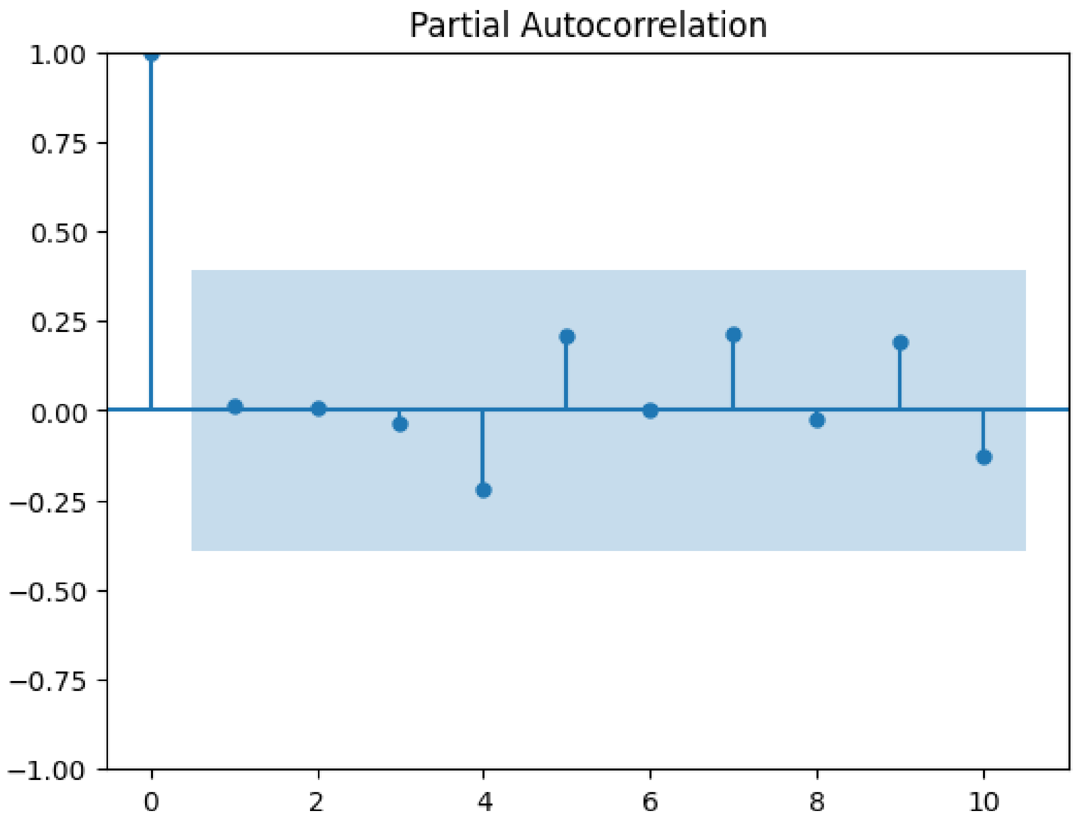
```
One date lag:  0.01341270719330465
Two date lag:  0.009796061559425587
Three date lag:  -0.042923672654262106
Six date lag:  0.046366685046454414
Nine date lag:  0.14693871890490812
```

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(df['volume'],lags=15);
```



Función de autocorrelación parcial

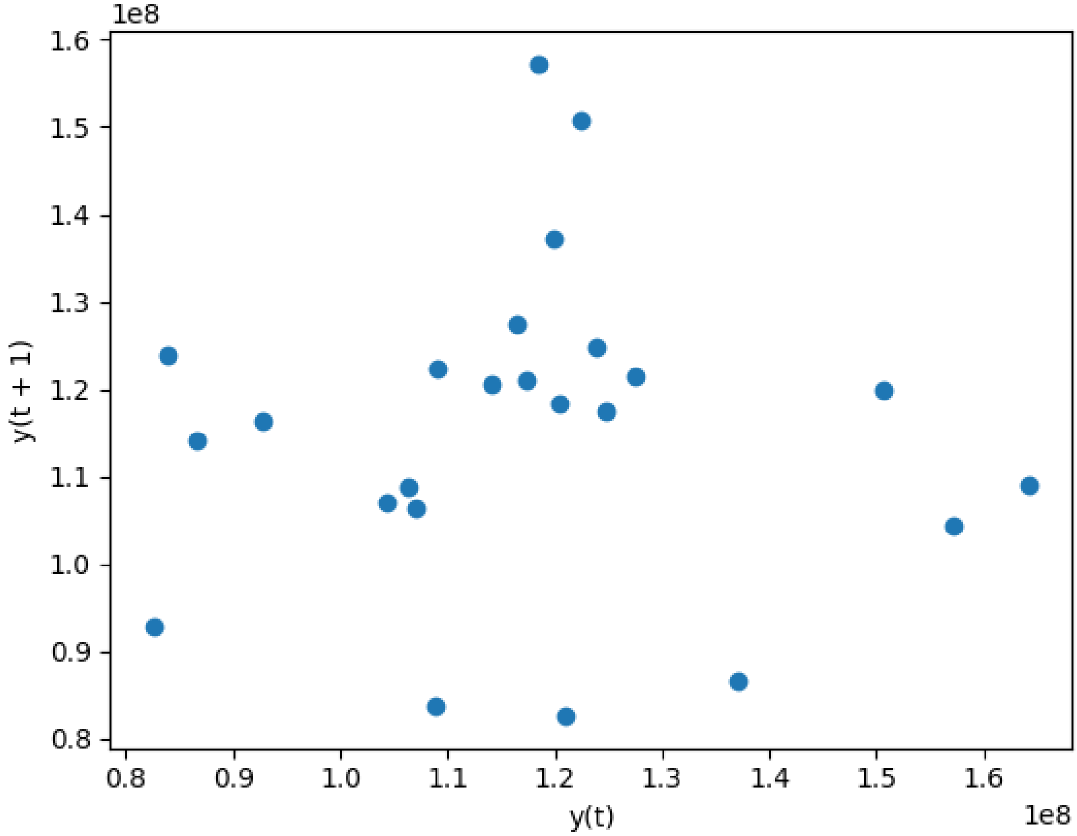
```
plot_pacf(df['volume'],lags=10);
```



No se observan datos que estén muy autocorrelacionados en esta serie de 15 datos.

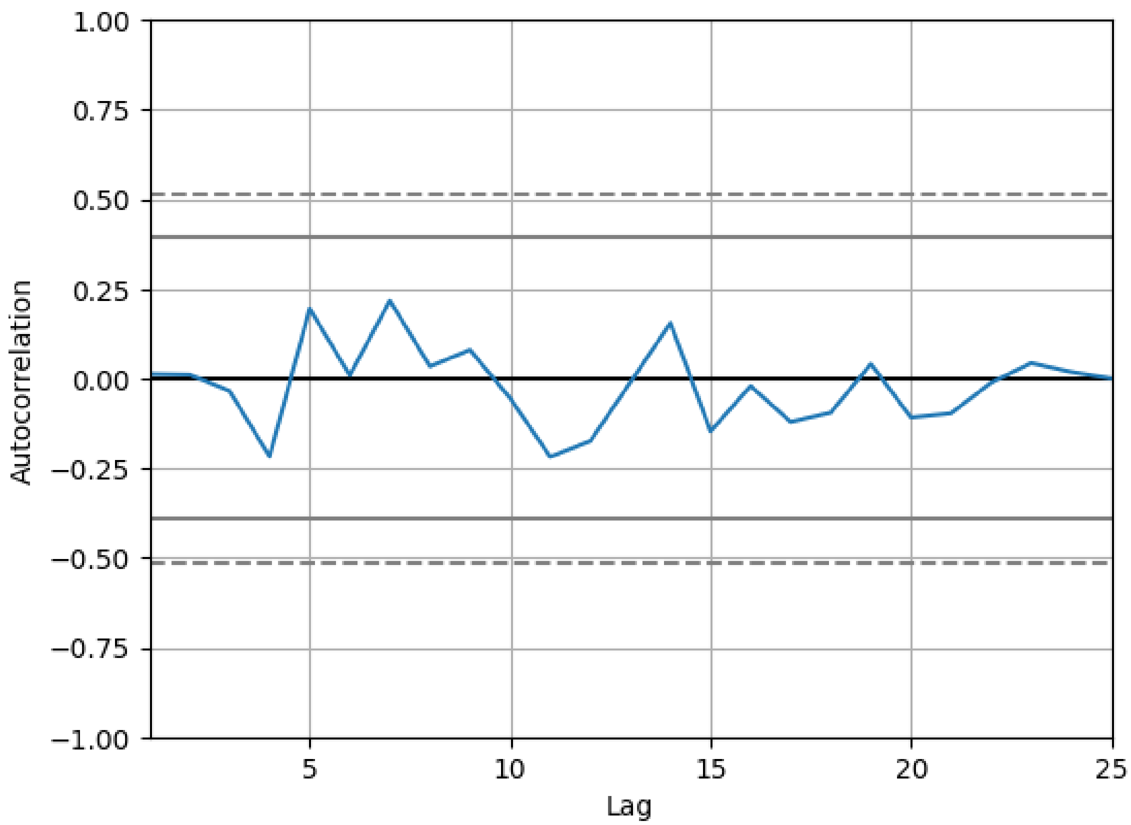
Comprobración de autocorrelación

```
from matplotlib import pyplot
from pandas.plotting import lag_plot

lag_plot(df['volume'])
pyplot.show()
```



No se observa un tendencia en los datos mostrados en el gráfico anterior, por lo que se puede inferir que no existe una clara correlación.

```
from pandas.plotting import autocorrelation_plot

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



Cuando se traza el coeficiente de correlación para cada variable desfasada y se observa que los valores se mantienen dentro de las líneas, las cuales abarcan un intervalo de entre 0.50 y -0.50 del coeficiente de autocorrelación.

Modelos de Autoregresión (AR)

Para probar el modelo de autoregresión se hará un predicción con tinua de los datos, seguido de una predicción a corto-plazo, para finalmente realizar un comparación de ambas predicciones utilizadas:

Predicción a corto-plazo:

```
#se separan los datos de la columna volume para realizar la
predicción:
series = df['volume'].copy()
print(series)
```

```
date
2011-01-07    1.641992e+08
2011-01-14    1.090246e+08
2011-01-21    1.223585e+08
2011-01-28    1.507353e+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
2011-04-08    1.063614e+08
2011-04-15    1.088940e+08
2011-04-21    8.382196e+07
2011-04-29    1.238058e+08
2011-05-06    1.247516e+08
2011-05-13    1.174370e+08
2011-05-20    1.210239e+08
2011-05-27    8.262061e+07
2011-06-03    9.284393e+07
2011-06-10    1.164434e+08
2011-06-17    1.274691e+08
2011-06-24    1.213916e+08
Name: volume, dtype: float64
```

```
#se dividen los datos en entrenamiento y prueba, se obtendrán 7 datos de conjunto de prueba:
x = series.values

size = int(len(x) * 0.75)
train, test  = x[0:size], x[size:len(x)]

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

#split indexes into train and test
ind_train, ind_test = df.index[0:size], df.index[size:len(x)]
```

```
Test size:  7
```

```
# set utiliza AutoReg
from statsmodels.tsa.ar_model import AutoReg

#train autoregr
model = AutoReg(train, lags=8)
model_fit = model.fit()
print('Coefficients: %s' % model_fit.params)
```

```
Coefficients: [-9.08437041e+08 -9.44791131e-02  3.50409799e-01
2.69650803e+00
  1.01598348e+00  5.51058242e-01  1.91059551e+00  2.64987723e+00
 -5.30585276e-01]
```

```
# SE UTILIZA EL MODELO PARA REALIZAR PREDICCIONES CON EL CONJUNTO DE 7 DATOS DE PRUEBA
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=-83658970.751678, expected=117436999.366667
predicted=53887072.023782, expected=121023933.633333
predicted=-1417299.457201, expected=82620614.933333
predicted=-527765972.120937, expected=92843928.300000
predicted=-329131407.178128, expected=116443430.933333
predicted=-535504280.699688, expected=127469133.966667
predicted=-2262995767.337777, expected=121391559.133333
```

Una vez realizada la predicción con los 7 datos de prueba, se procede a calcular el error del modelo:

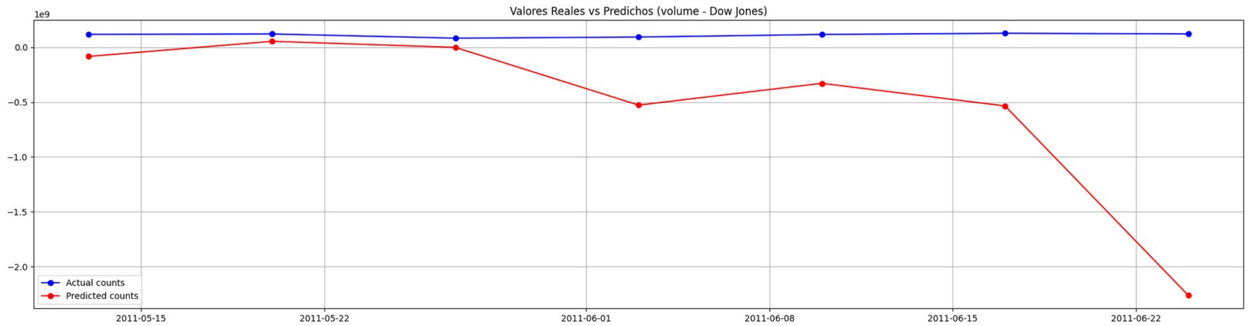
```
from math import sqrt
from sklearn.metrics import mean_squared_error

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

```
Test RMSE: 982746848.270
```

Se obtiene un error cuadrático medio de 982746848.27, un error algo importante en cuanto a magnitud.

Para tener una referencia más visual del modelo, se realiza una gráfica para hacer la comparativa en los 7 datos predichos y reales:

```
#grafica de valores reales contra valores predichos
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('Valores Reales vs Predichos (volume - Dow Jones)')
plt.legend(handles=[actual, predicted])
plt.grid()
plt.show()
```



Se observa que en las predicciones de los 7 datos, las tres primeras tienen un alto grado de parecido con los datos reales, a partir del tercer datos, se eempieza a observar que los datos predichos se empiezan a alejar de los datos reales, a pesar que tengamos una serie estacionaria. Este primer forecast presenta este problema por lo que se probará ahora con una predicción a continua.

Predicción continua:

```
# modelo AutoReg, a medida que disponga de nuevas observaciones:
train_history = list(train)
predictions = list()

#walk-forward validation
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))
```

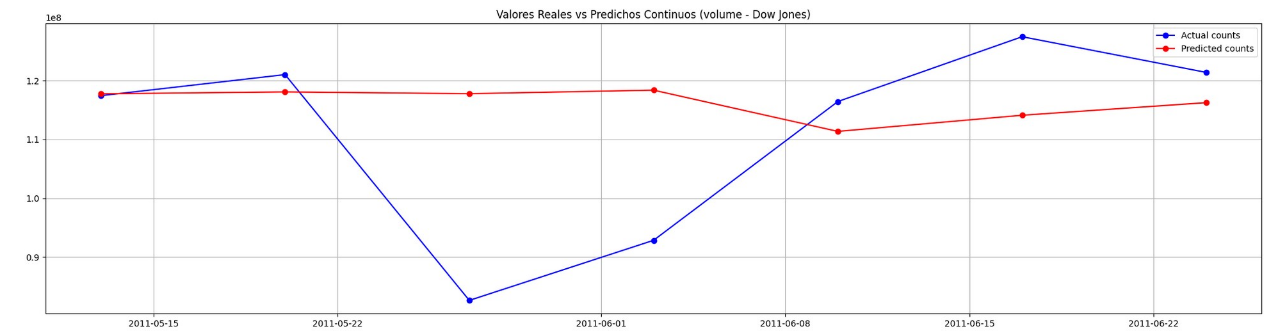
```
predicted=117754241.527644, expected=117436999.366667
predicted=118076413.228416, expected=121023933.633333
predicted=117775352.991099, expected=82620614.933333
predicted=118373431.200936, expected=92843928.300000
predicted=111362157.233486, expected=116443430.933333
predicted=114110957.956768, expected=127469133.966667
predicted=116255934.450556, expected=121391559.133333
```

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

```
Test RMSE: 17431639.932
```

Se obtiene un error cuadrárico medio de 17431639.93 en esta predicción.

```
#grafica de valores reales contra valores predichos en predicción
continua
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('Valores Reales vs Predichos Continuos (volume - Dow
Jones)')
plt.legend(handles=[actual, predicted])
plt.grid()
plt.show()
```



Al visualizar la gráfica de comparación entre los datos reales y los datos predichos por el modelo, se observan diferencias claras con la predicción anterior. En este caso las diferencias entre datos no son tan significativas como en el primer modelo, aquí se aprecia un error menor que en el modelo.

Si se realiza un comparación entre ambos modelos o predicciones, se puede observar que claramente la primera predicción tiene un error mayor al pronosticar, por lo que se concluye que al utilizar AR en estos datos, es mejor aplicar un pronóstico como en el segundo modelo, ya que tiene menor error.

```
x = 982746848.27
y = 17431639.93
if x > y:
  print('El error en el primer modelo es mayor')
else:
  print('El error en el segundo modelo es mayor')
```

```
El error en el primer modelo es mayor
```

El primer modelo tiene un error significativamente mayor al del segundo modelo.

Modelos ARIMA

Se realizará un modelo ARIMA para los datos que son estacionarios:

```python
import warnings
from statsmodels.tsa.arima.model import ARIMA
warnings.filterwarnings('ignore')

best_aic = 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_aic:
      best_aic = 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 AIC: {best_aic:.2f}')
```

```
AR(1): AIC= 664.35
AR(2): AIC= 666.11
AR(3): AIC= 667.89
AR(4): AIC= 669.25
AR(5): AIC= 670.96
AR(6): AIC= 672.33
AR(7): AIC= 674.77
AR(8): AIC= 1145.36
AR(9): AIC= 668212491277.82
AR(10): AIC= 501537043978.41
Error for AR(11): LU decomposition error.
AR(12): AIC= 237761.20
Error for AR(13): LU decomposition error.
AR(14): AIC= 184897529101.23
AR(15): AIC= 11166195871.04
AR(16): AIC= 17265413588.88
AR(17): AIC= 690.79
AR(18): AIC= 692.79
AR(19): AIC= 694.79

Best AR order: (1, 0, 0) with AIC: 664.35
```

Se obtiene que la notación ARIMA ideal en este caso es (1,0,0) con un p = 1, d = 0 y q = 0.

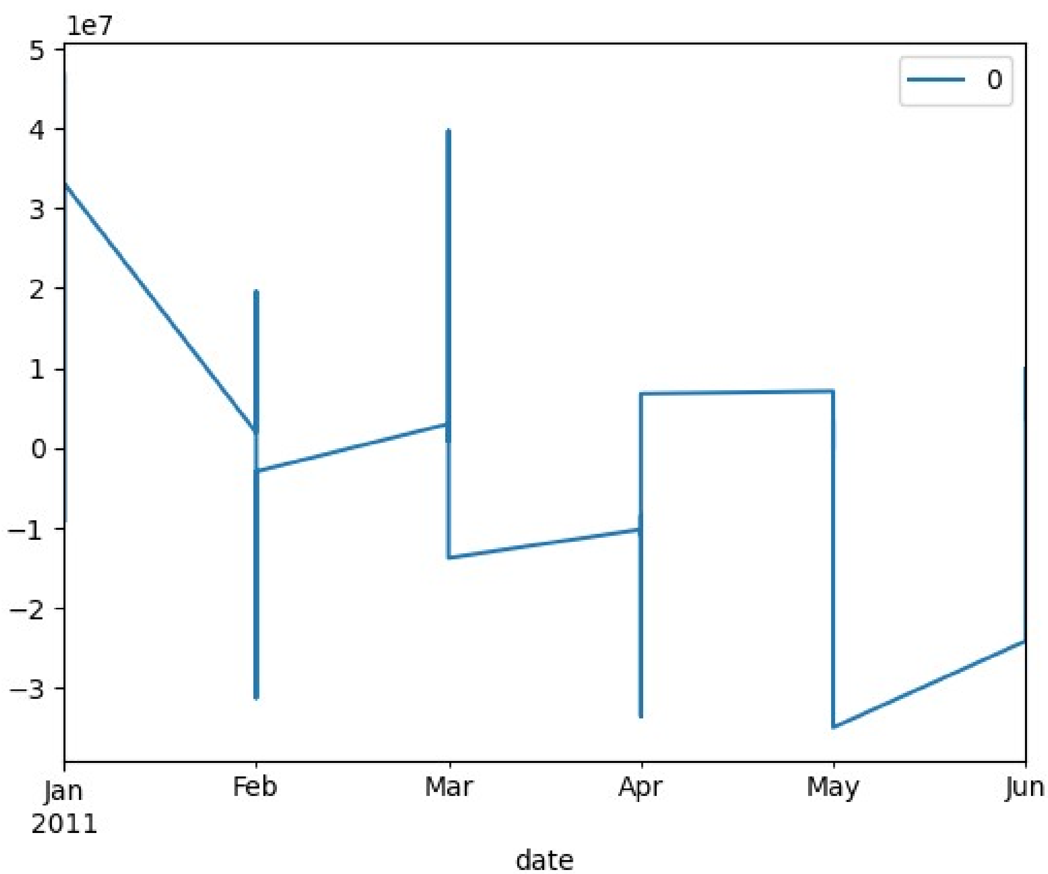
```python
series = df['volume'].copy()
print(series)
```

```
series.index = series.index.to_period('M')
print('\n', series)
```

Se crea el modelo ARIMA (1,0,0)

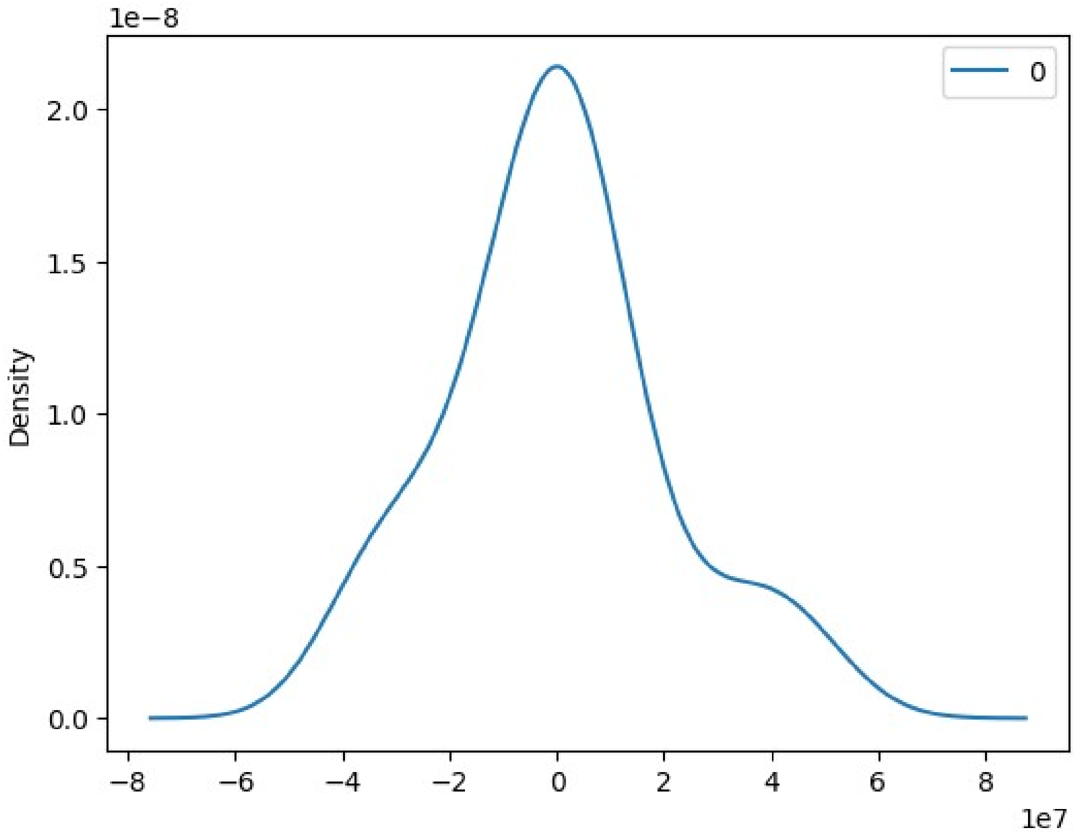
```
model = ARIMA(series, order=(1,0,0))
model_fit = model.fit()
print('Coeffients: \n%s' % model_fit.params)

Coeffients:
const     1.175478e+08
ar.L1     1.541620e-02
sigma2    3.252677e+14
dtype: float64

print(model_fit.summary())

                               SARIMAX Results

======================================================================
========
Dep. Variable:                 volume   No. Observations:
25
Model:                 ARIMA(1, 0, 0)   Log Likelihood
-456.014
Date:                Tue, 14 Nov 2023   AIC
918.029
Time:                        21:32:11   BIC
921.685
Sample:                    01-31-2011   HQIC
919.043
                         - 06-30-2011

Covariance Type:                  opg

======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
const       1.175e+08    3.3e+06     35.598      0.000    1.11e+08
1.24e+08
ar.L1          0.0154      0.242      0.064      0.949      -0.459
0.490
sigma2      3.253e+14      0.040   8.11e+15      0.000    3.25e+14
3.25e+14
======================================================================
=============
Ljung-Box (L1) (Q):                  0.00   Jarque-Bera (JB):
```

```
0.67
Prob(Q):                                  0.99   Prob(JB):
0.72
Heteroskedasticity (H):                   0.42   Skew:
0.39
Prob(H) (two-sided):                      0.24   Kurtosis:
3.19
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
[2] Covariance matrix is singular or near-singular, with condition
number 5.03e+31. Standard errors may be unstable.

#line plot with residuals
from matplotlib import pyplot
residuals = pd.DataFrame(model_fit.resid)
residuals.plot()
pyplot.show()
```



No se observan tendecias de los residuos del modelo.

```
#density plot of residuals
residuals.plot(kind='kde')
pyplot.show()
```



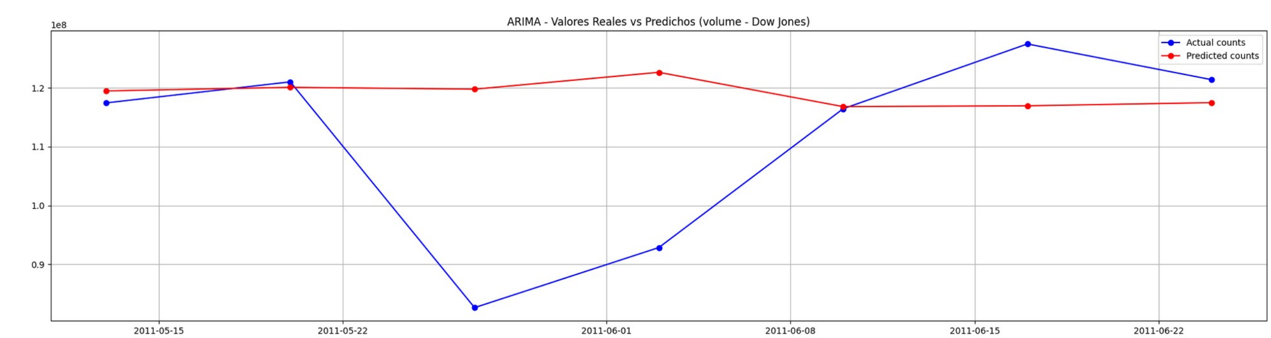
Al realizar una gráfica para medir la distribución de los errores residuales, se puede observar que en su mayoría están centrados a 0, lo que puede indicar que no existe un sesgo en la predicción de los datos.

Modelo ARIMA:

Se hace el modelo ARIMA(1,0,0) y se obtienen datos predichos y reales:

```
history = list(train)
predictions = list()

for t in range(len(test)):
  model = ARIMA(history, order=(1,0,0))
  model_fit = model.fit()

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

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

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

predicted=119476111.774296, expected=117436999.366667
predicted=120092244.549804, expected=121023933.633333
predicted=119776224.755665, expected=82620614.933333
predicted=122639869.508515, expected=92843928.300000
predicted=116807983.896452, expected=116443430.933333
predicted=116945374.731096, expected=127469133.966667
predicted=117484910.135014, expected=121391559.133333

# Error cuadrático medio del modelo ARIMA:
rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)

Test RMSE: 18514503.162

#grafica de valores reales contra valores predichos en predicción
continua
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('ARIMA - Valores Reales vs Predichos (volume - Dow Jones)')
plt.legend(handles=[actual, predicted])
plt.grid()
plt.show()
```



La gráfica muestra que el modelo ARIMA tiene una precisión significativa a excepción de dos datos que tienen un error notable.

De acuerdo con la gráfica anterior, se observa que con el modelo ARIMA generado, se obtiene un resultado muy similar al del modelo AR de predicción contínua. La manera en que los datos se observan en el tiempo en ambos modelos es casi idéntica, justo al igual que los respectivos errores de los modelos, donde el error en el modelo ARIMA es ligeramente mayor que el error del modelo AR. Con esta comparación se puede decir que el modelo AR de predicción contínua es ligeramente mejor que el modelo ARIMA, ya que a pesar de que visualmente los

datos predichos y reales se parecen mucho, sí existe una diferencia en el error donde a pesar de que es pequeña, indica que el modelo AR es mejor para pronosticar los datos

```
# comparación de errores del modelo ARIMA y el modelo AR:
x = 18514503.162
y = 17431639.93
if x > y:
  print('El error en el modelo ARIMA es mayor')
else:
  print('El error en el modelo AR es mayor')
```

```
El error en el modelo ARIMA es mayor
```

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

Se puede considerar que utilizar un modelo de Autoregresión AR es mejor para predecir de forma contínua, en el caso de modelos ARIMA pueden servir de mejor forma para predecir en escenarios con una mayor complejidad(para el ajuste de parámetros) o para datos que requieran alguna diferenciación exista algún tipo de tendencia. Podría decirse que de igual forma, modelos como el de ARIMA y el AR necesitan de más datos para funcionar o predecir de mejor forma.

Para el caso particular de este problema y base de datos (Dow Jones) se puede decir que un modelo AR le puede venir mejor a este tipo de datos, ya que el problema no es especialmente complejo, los datos son estacionarios y no requieren de una aplicación de diferenciación o algo de índole un poco más compleja como lo es en el modelo ARIMA.