all_auto

January 13, 2025

```
[1]: # use a library to do everything automatically, so that we can compare our␣
      ↪results to

     import numpy as np
     import pandas as pd
     import plotly.express as px
     import plotly.graph_objects as go
     from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
     import statsmodels.tsa.stattools as st
     import statsmodels.api as sm
     import numpy as np
     import pmdarima as pm


     # Load the CSV file
     data = pd.read_csv('../data/EUR_USD_monthly.csv')

     # Extract the time and price series
     time = pd.to_datetime(data.iloc[:, 0])  # Assuming first column is 'time'
     prices = data.iloc[:, 1].replace(',', '', regex=True).astype(float)  # Remove␣
      ↪commas and convert to float


     # Create a DataFrame with the time and prices
     time_series_df = pd.DataFrame({'time': time, 'points': prices})
     time_series_df.set_index('time', inplace=True)

     def plot_series(series):
         # Plot the time series using Plotly
         series.plot()

         plot_acf(series)
         plot_pacf(series)
     plot_series(time_series_df)
```

```python
# Automatically fit the best ARIMA/SARIMA model
model = pm.auto_arima(
    time_series_df,
    seasonal=False,       # Enable seasonal differencing if necessary
    stepwise=False,       # Use stepwise algorithm for faster computation
    trace=True,          # Show the fitting process details
    #error_action='ignore',  # Ignore if the model doesn't fit properly
    #suppress_warnings=True  # Suppress warnings
    max_p=10,              # Set maximum p value to control AR component
    max_q=10,              # Set maximum q value to control MA component
    max_d=1,              # Set maximum differencing order
    max_P=1,              # Set maximum seasonal AR order
    max_Q=1,              # Set maximum seasonal MA order
    max_D=1,
    max_order = 10000
)


# Summary of the best model found
print(model.summary())

# Predict future values (e.g., next 12 periods)
forecast = model.predict(n_periods=12)
print(forecast)
```

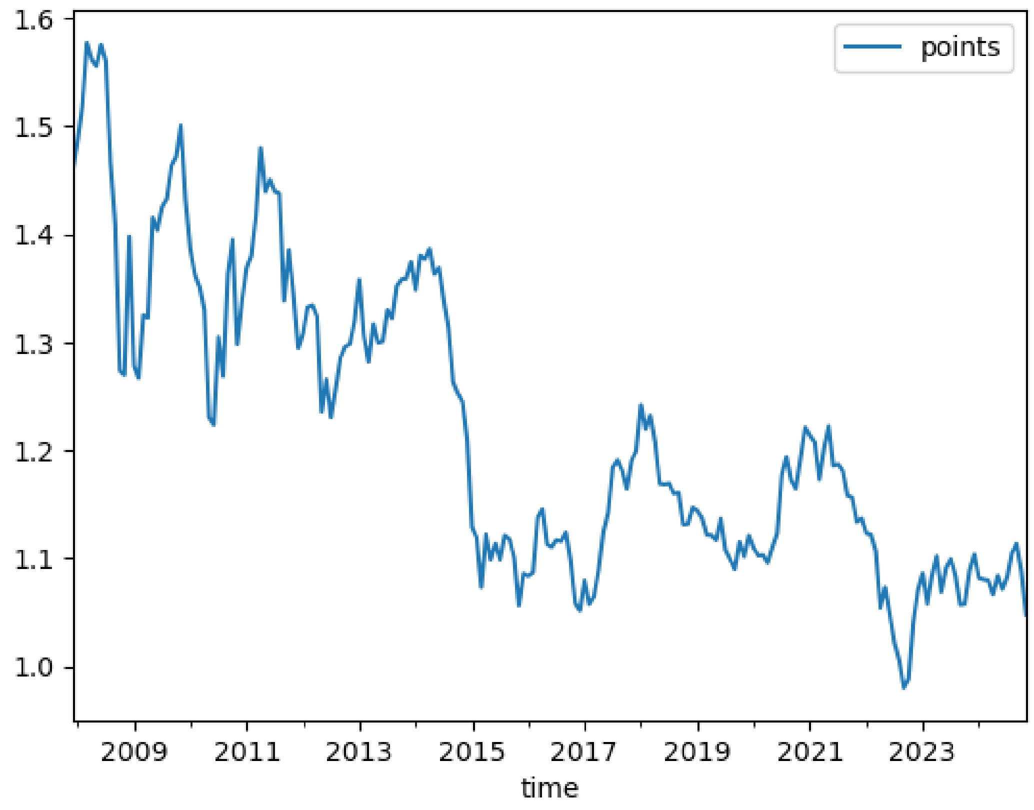
```
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-780.323, Time=0.05 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-778.566, Time=0.09 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-776.664, Time=0.12 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=-777.610, Time=0.20 sec
 ARIMA(0,1,4)(0,0,0)[0] intercept   : AIC=-775.616, Time=0.37 sec
 ARIMA(0,1,5)(0,0,0)[0] intercept   : AIC=-773.677, Time=0.33 sec
 ARIMA(0,1,6)(0,0,0)[0] intercept   : AIC=-774.825, Time=0.49 sec
 ARIMA(0,1,7)(0,0,0)[0] intercept   : AIC=-786.213, Time=0.61 sec
 ARIMA(0,1,8)(0,0,0)[0] intercept   : AIC=-788.144, Time=0.86 sec
 ARIMA(0,1,9)(0,0,0)[0] intercept   : AIC=-788.023, Time=1.14 sec
 ARIMA(0,1,10)(0,0,0)[0] intercept   : AIC=-786.866, Time=1.38 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-778.554, Time=0.08 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-776.576, Time=0.20 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-775.663, Time=0.26 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=-772.678, Time=0.38 sec
 ARIMA(1,1,4)(0,0,0)[0] intercept   : AIC=-773.612, Time=0.82 sec
 ARIMA(1,1,5)(0,0,0)[0] intercept   : AIC=-771.732, Time=0.63 sec
 ARIMA(1,1,6)(0,0,0)[0] intercept   : AIC=-772.217, Time=0.75 sec
 ARIMA(1,1,7)(0,0,0)[0] intercept   : AIC=-789.623, Time=1.25 sec
 ARIMA(1,1,8)(0,0,0)[0] intercept   : AIC=-787.402, Time=1.28 sec
 ARIMA(1,1,9)(0,0,0)[0] intercept   : AIC=-785.811, Time=0.91 sec
 ARIMA(1,1,10)(0,0,0)[0] intercept   : AIC=-784.508, Time=1.16 sec
```

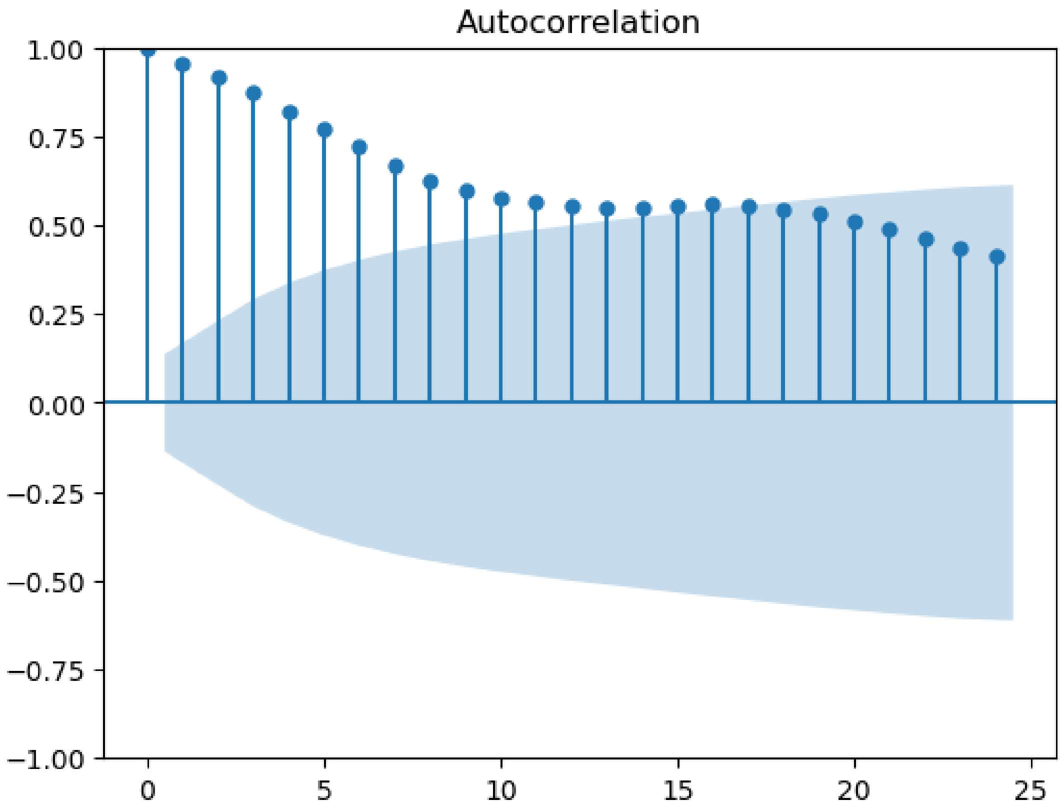
```
ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=-776.734, Time=0.14 sec
ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-776.057, Time=0.39 sec
ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-779.516, Time=0.38 sec
ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=-777.408, Time=0.41 sec
ARIMA(2,1,4)(0,0,0)[0] intercept   : AIC=-776.261, Time=0.61 sec
ARIMA(2,1,5)(0,0,0)[0] intercept   : AIC=-773.019, Time=0.74 sec
ARIMA(2,1,6)(0,0,0)[0] intercept   : AIC=-771.591, Time=0.72 sec
ARIMA(2,1,7)(0,0,0)[0] intercept   : AIC=-788.194, Time=0.89 sec
ARIMA(2,1,8)(0,0,0)[0] intercept   : AIC=-786.177, Time=3.69 sec
ARIMA(2,1,9)(0,0,0)[0] intercept   : AIC=-783.308, Time=1.20 sec
ARIMA(2,1,10)(0,0,0)[0] intercept   : AIC=-783.433, Time=1.09 sec
ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=-778.059, Time=0.15 sec
ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=-776.526, Time=0.38 sec
ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=-776.032, Time=0.60 sec
ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=-775.393, Time=0.78 sec
ARIMA(3,1,4)(0,0,0)[0] intercept   : AIC=-777.290, Time=0.68 sec
ARIMA(3,1,5)(0,0,0)[0] intercept   : AIC=-778.172, Time=0.75 sec
ARIMA(3,1,6)(0,0,0)[0] intercept   : AIC=-774.227, Time=0.65 sec
ARIMA(3,1,7)(0,0,0)[0] intercept   : AIC=-785.189, Time=0.76 sec
ARIMA(3,1,8)(0,0,0)[0] intercept   : AIC=-784.013, Time=1.03 sec
ARIMA(3,1,9)(0,0,0)[0] intercept   : AIC=-780.698, Time=1.02 sec
ARIMA(3,1,10)(0,0,0)[0] intercept   : AIC=-782.129, Time=1.07 sec
ARIMA(4,1,0)(0,0,0)[0] intercept   : AIC=-776.713, Time=0.15 sec
ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=-774.712, Time=0.20 sec
ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=-772.716, Time=0.48 sec
ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=-773.050, Time=0.56 sec
ARIMA(4,1,4)(0,0,0)[0] intercept   : AIC=-775.124, Time=0.59 sec
ARIMA(4,1,5)(0,0,0)[0] intercept   : AIC=inf, Time=0.72 sec
ARIMA(4,1,6)(0,0,0)[0] intercept   : AIC=-772.004, Time=0.86 sec
ARIMA(4,1,7)(0,0,0)[0] intercept   : AIC=-788.061, Time=1.14 sec
ARIMA(4,1,8)(0,0,0)[0] intercept   : AIC=-786.468, Time=1.16 sec
ARIMA(4,1,9)(0,0,0)[0] intercept   : AIC=-783.727, Time=1.09 sec
ARIMA(4,1,10)(0,0,0)[0] intercept   : AIC=-779.023, Time=1.21 sec
ARIMA(5,1,0)(0,0,0)[0] intercept   : AIC=-774.732, Time=0.19 sec
ARIMA(5,1,1)(0,0,0)[0] intercept   : AIC=-772.734, Time=0.23 sec
ARIMA(5,1,2)(0,0,0)[0] intercept   : AIC=-765.585, Time=0.52 sec
ARIMA(5,1,3)(0,0,0)[0] intercept   : AIC=-770.939, Time=0.48 sec
ARIMA(5,1,4)(0,0,0)[0] intercept   : AIC=-773.459, Time=0.59 sec
ARIMA(5,1,5)(0,0,0)[0] intercept   : AIC=-772.102, Time=0.77 sec
ARIMA(5,1,6)(0,0,0)[0] intercept   : AIC=-772.568, Time=0.81 sec
ARIMA(5,1,7)(0,0,0)[0] intercept   : AIC=-784.035, Time=0.82 sec
ARIMA(5,1,8)(0,0,0)[0] intercept   : AIC=-783.259, Time=0.95 sec
ARIMA(5,1,9)(0,0,0)[0] intercept   : AIC=-778.983, Time=1.05 sec
ARIMA(5,1,10)(0,0,0)[0] intercept   : AIC=-776.574, Time=1.14 sec
ARIMA(6,1,0)(0,0,0)[0] intercept   : AIC=-773.346, Time=0.18 sec
ARIMA(6,1,1)(0,0,0)[0] intercept   : AIC=-775.683, Time=0.55 sec
ARIMA(6,1,2)(0,0,0)[0] intercept   : AIC=-771.604, Time=0.59 sec
ARIMA(6,1,3)(0,0,0)[0] intercept   : AIC=-770.539, Time=1.09 sec
ARIMA(6,1,4)(0,0,0)[0] intercept   : AIC=-777.565, Time=1.20 sec
ARIMA(6,1,5)(0,0,0)[0] intercept   : AIC=-763.117, Time=0.79 sec
ARIMA(6,1,6)(0,0,0)[0] intercept   : AIC=-770.514, Time=0.93 sec
ARIMA(6,1,7)(0,0,0)[0] intercept   : AIC=-781.841, Time=0.94 sec
ARIMA(6,1,8)(0,0,0)[0] intercept   : AIC=-780.520, Time=1.12 sec
ARIMA(6,1,9)(0,0,0)[0] intercept   : AIC=-780.710, Time=1.05 sec
ARIMA(6,1,10)(0,0,0)[0] intercept   : AIC=-776.810, Time=1.18 sec
ARIMA(7,1,0)(0,0,0)[0] intercept   : AIC=-780.185, Time=0.57 sec
ARIMA(7,1,1)(0,0,0)[0] intercept   : AIC=-784.702, Time=0.56 sec
ARIMA(7,1,2)(0,0,0)[0] intercept   : AIC=-785.095, Time=0.62 sec
ARIMA(7,1,3)(0,0,0)[0] intercept   : AIC=-778.585, Time=0.71 sec
ARIMA(7,1,4)(0,0,0)[0] intercept   : AIC=-778.842, Time=0.70 sec
ARIMA(7,1,5)(0,0,0)[0] intercept   : AIC=-780.412, Time=0.76 sec
ARIMA(7,1,6)(0,0,0)[0] intercept   : AIC=-782.625, Time=0.87 sec
ARIMA(7,1,7)(0,0,0)[0] intercept   : AIC=-782.239, Time=0.93 sec
ARIMA(7,1,8)(0,0,0)[0] intercept   : AIC=-778.164, Time=1.09 sec
ARIMA(7,1,9)(0,0,0)[0] intercept   : AIC=-778.036, Time=1.10 sec
ARIMA(7,1,10)(0,0,0)[0] intercept   : AIC=-772.525, Time=1.19 sec
ARIMA(8,1,0)(0,0,0)[0] intercept   : AIC=-780.928, Time=0.37 sec
ARIMA(8,1,1)(0,0,0)[0] intercept   : AIC=-783.543, Time=0.70 sec
ARIMA(8,1,2)(0,0,0)[0] intercept   : AIC=-784.090, Time=0.87 sec
ARIMA(8,1,3)(0,0,0)[0] intercept   : AIC=-778.282, Time=0.88 sec
ARIMA(8,1,4)(0,0,0)[0] intercept   : AIC=-778.121, Time=0.92 sec
ARIMA(8,1,5)(0,0,0)[0] intercept   : AIC=-780.449, Time=1.01 sec
ARIMA(8,1,6)(0,0,0)[0] intercept   : AIC=-781.395, Time=1.02 sec
ARIMA(8,1,7)(0,0,0)[0] intercept   : AIC=-779.820, Time=0.99 sec
ARIMA(8,1,8)(0,0,0)[0] intercept   : AIC=-779.167, Time=1.15 sec
ARIMA(8,1,9)(0,0,0)[0] intercept   : AIC=-777.241, Time=1.24 sec
ARIMA(8,1,10)(0,0,0)[0] intercept   : AIC=-776.783, Time=1.42 sec
ARIMA(9,1,0)(0,0,0)[0] intercept   : AIC=-784.241, Time=0.50 sec
ARIMA(9,1,1)(0,0,0)[0] intercept   : AIC=-783.348, Time=0.75 sec
ARIMA(9,1,2)(0,0,0)[0] intercept   : AIC=-781.308, Time=0.80 sec
ARIMA(9,1,3)(0,0,0)[0] intercept   : AIC=-781.580, Time=0.97 sec
ARIMA(9,1,4)(0,0,0)[0] intercept   : AIC=-781.300, Time=0.92 sec
ARIMA(9,1,5)(0,0,0)[0] intercept   : AIC=-783.088, Time=1.02 sec
ARIMA(9,1,6)(0,0,0)[0] intercept   : AIC=-780.519, Time=1.17 sec
ARIMA(9,1,7)(0,0,0)[0] intercept   : AIC=-776.893, Time=1.19 sec
ARIMA(9,1,8)(0,0,0)[0] intercept   : AIC=-776.257, Time=1.29 sec
ARIMA(9,1,9)(0,0,0)[0] intercept   : AIC=-774.141, Time=1.30 sec
ARIMA(9,1,10)(0,0,0)[0] intercept   : AIC=-774.679, Time=1.49 sec
ARIMA(10,1,0)(0,0,0)[0] intercept   : AIC=-783.369, Time=0.54 sec
ARIMA(10,1,1)(0,0,0)[0] intercept   : AIC=-781.589, Time=0.89 sec
ARIMA(10,1,2)(0,0,0)[0] intercept   : AIC=-780.122, Time=0.99 sec
ARIMA(10,1,3)(0,0,0)[0] intercept   : AIC=-781.219, Time=1.06 sec
ARIMA(10,1,4)(0,0,0)[0] intercept   : AIC=-779.766, Time=1.24 sec
ARIMA(10,1,5)(0,0,0)[0] intercept   : AIC=-777.893, Time=1.56 sec
ARIMA(10,1,6)(0,0,0)[0] intercept   : AIC=-776.533, Time=1.64 sec
ARIMA(10,1,7)(0,0,0)[0] intercept   : AIC=-776.063, Time=1.23 sec
```

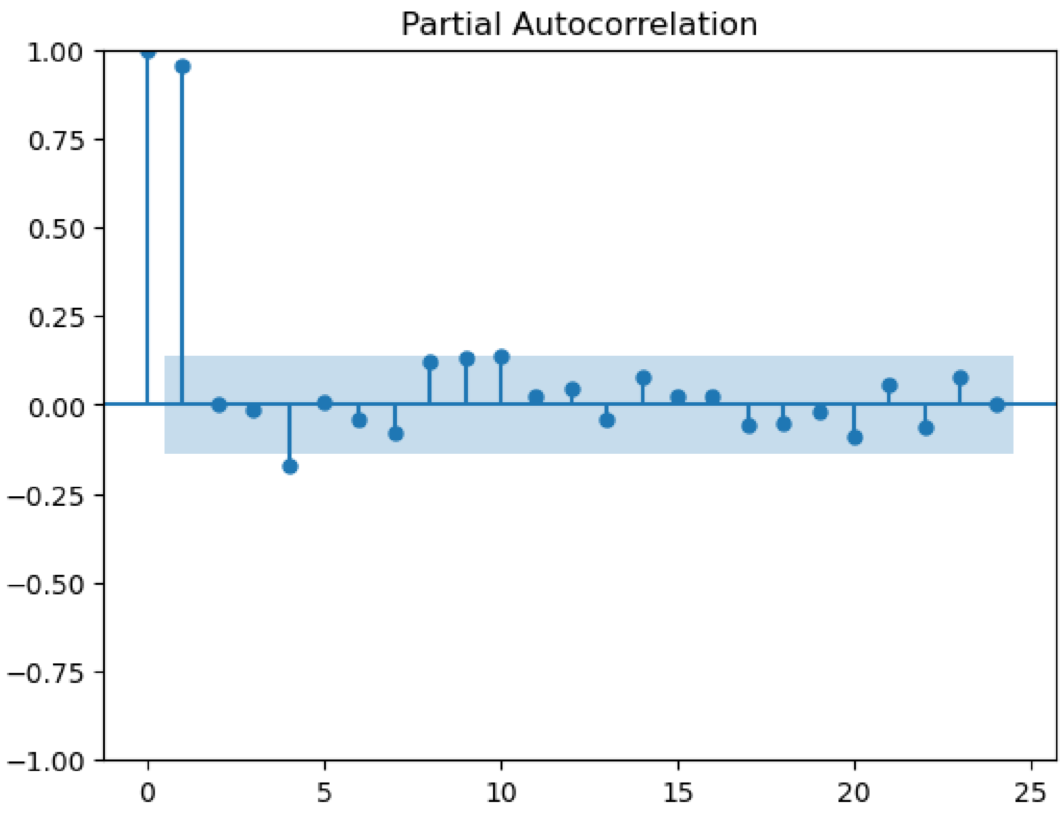
```
 ARIMA(10,1,8)(0,0,0)[0] intercept   : AIC=-774.097, Time=1.40 sec
 ARIMA(10,1,9)(0,0,0)[0] intercept   : AIC=-772.588, Time=1.56 sec
 ARIMA(10,1,10)(0,0,0)[0] intercept   : AIC=-772.297, Time=2.22 sec

Best model:  ARIMA(1,1,7)(0,0,0)[0] intercept
Total fit time: 102.259 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  204
Model:               SARIMAX(1, 1, 7)   Log Likelihood                 404.811
Date:                Mon, 13 Jan 2025   AIC                           -789.623
Time:                        18:44:13   BIC                           -756.491
Sample:                             0   HQIC                          -776.219
                                - 204
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0012      0.000      2.504      0.012       0.000       0.002
ar.L1          0.4156      0.173      2.399      0.016       0.076       0.755
ma.L1         -0.4756      0.158     -3.011      0.003      -0.785      -0.166
ma.L2         -0.0424      0.063     -0.677      0.498      -0.165       0.080
ma.L3          0.1395      0.061      2.269      0.023       0.019       0.260
ma.L4         -0.1904      0.061     -3.124      0.002      -0.310      -0.071
ma.L5         -0.0169      0.059     -0.285      0.775      -0.133       0.099
ma.L6          0.0529      0.077      0.687      0.492      -0.098       0.204
ma.L7         -0.3274      0.068     -4.804      0.000      -0.461      -0.194
sigma2         0.0011   8.76e-05     12.293      0.000       0.001       0.001
================================================================================
===
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
22.66
Prob(Q):                              0.96   Prob(JB):
0.00
Heteroskedasticity (H):               4.02   Skew:
0.00
Prob(H) (two-sided):                  0.00   Kurtosis:
4.64
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
204    1.443172
205    1.438601
206    1.442985
207    1.438519
```

```
208    1.450690
209    1.456158
210    1.459125
211    1.461514
212    1.463662
213    1.465711
214    1.467718
215    1.469708
dtype: float64
```

```
/home/daniels/.conda/envs/adlr/lib/python3.10/site-
packages/statsmodels/tsa/base/tsa_model.py:836: ValueWarning: No supported index
is available. Prediction results will be given with an integer index beginning
at `start`.
  return get_prediction_index(
/home/daniels/.conda/envs/adlr/lib/python3.10/site-
packages/statsmodels/tsa/base/tsa_model.py:836: FutureWarning: No supported
index is available. In the next version, calling this method in a model without
a supported index will result in an exception.
  return get_prediction_index(
```




Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25



```
[ ]:
```

Test stationarity, Augmented Dickey-Fuller unit root test.

```
[2]: #Check stationarity
     # alternative hypothesis is stationarity
     st.adfuller(time_series_df['points'])
```

```
[2]: (-0.8833017572640469,
      0.7934216416235764,
      9,
      194,
      {'1%': -3.4645146202692527,
       '5%': -2.8765564361715534,
       '10%': -2.5747745328940375},
      -717.875322423628)
```

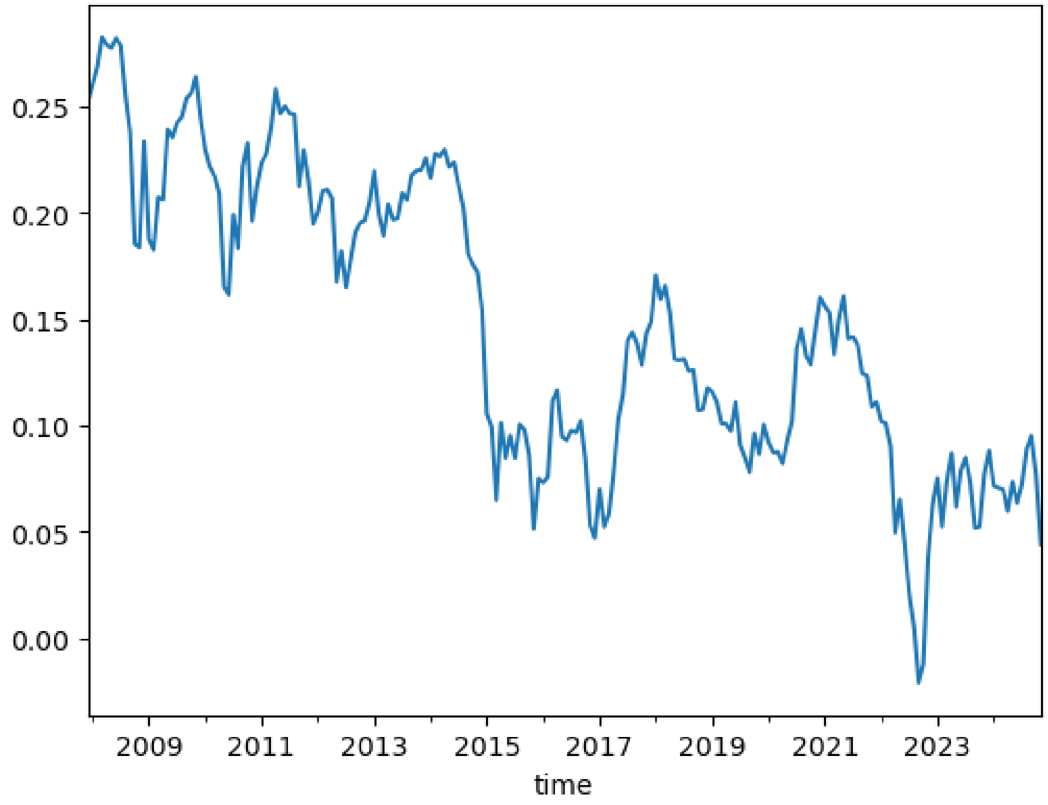
p value $> 0.05 =>$ we can not conclude stationarity

```
[ ]:
```

apply box cox transformation, and use scipy to estimate optimal box cox parameter lambda (todo explain what method scipy uses)

[3]:
```python
from scipy import stats


# Apply Box-Cox transformation and estimate lambda
transformed_prices, lambda_est = stats.boxcox(time_series_df["points"])

# Store the transformed values back into the DataFrame
time_series_df['boxcox_points'] = transformed_prices

time_series_df['boxcox_points'].plot()

print("Optimal lambda: ", lambda_est)
```

```
Optimal lambda:  -2.3018585777936003
```



Differentiate and see when stationarity tests become positive

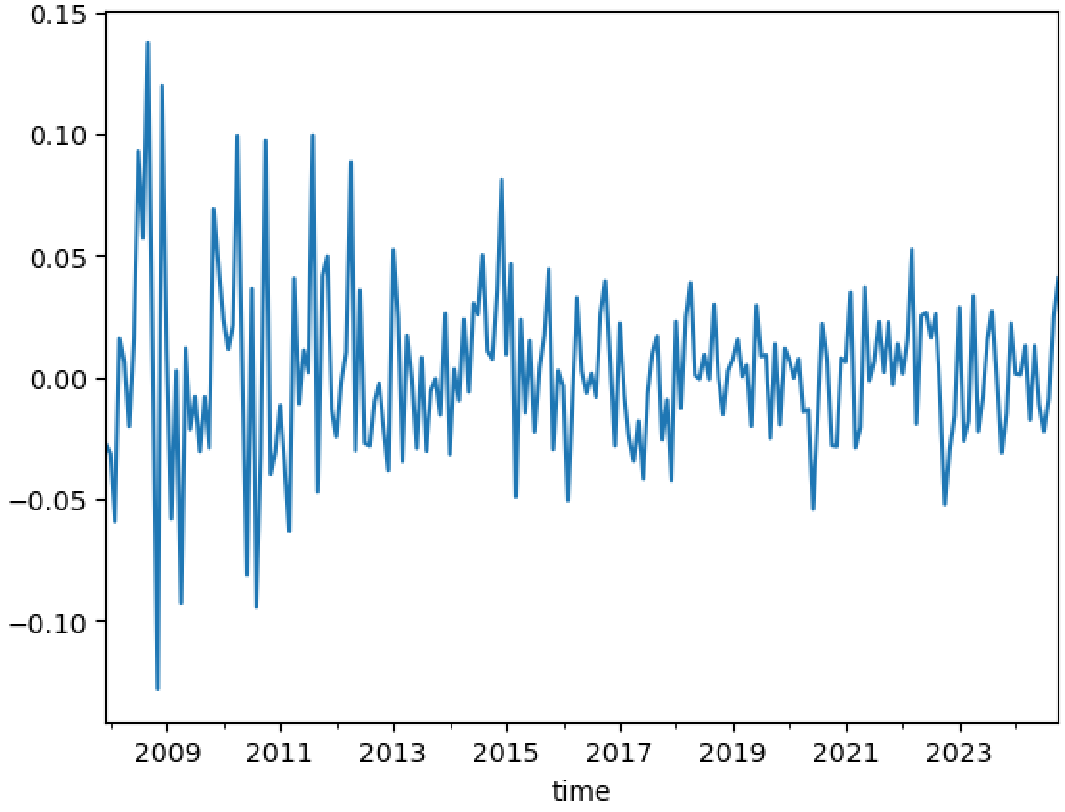
[4]:
```python
alpha = 0.05



# Assuming 'time_series_df' is your DataFrame and 'Prices' is the column
```

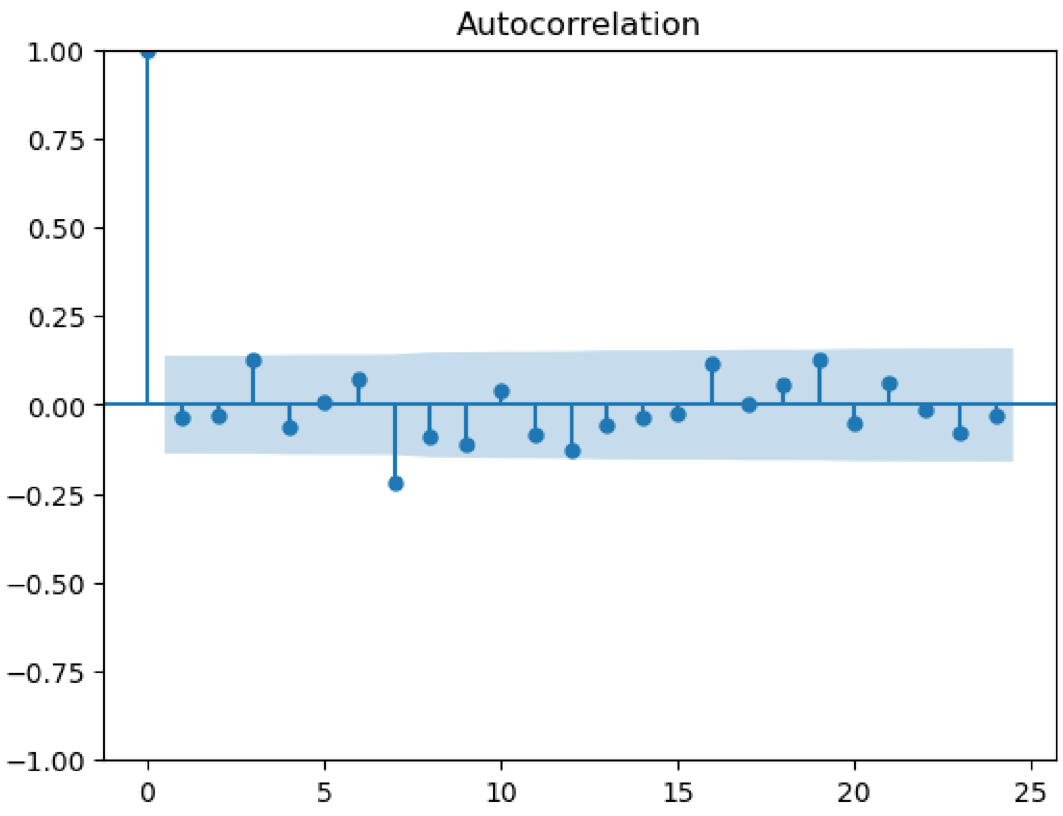
```
current_series = time_series_df['points'] # todo possible bug, try with_: pd.
↪DataFrame(np.random.random(100))#np.exp(time_series_df['boxcox_points'])
differencing_count = 0

# Function to apply ADF test and get p-value
def get_adf_p_value(series):
    adf_result = st.adfuller(series.dropna())
    return adf_result[1]  # Return p-value from ADF test

# Keep iterating until the series is stationary or we reach the max differencing
while get_adf_p_value(current_series) >= alpha:
    differencing_count += 1
    # Apply first-order differencing
    current_series = current_series.diff()

    # Add the differenced series to the DataFrame
    time_series_df[f'diff_prices_{differencing_count}'] = current_series
    print("defferentiation")
    # Print progress with the p-value after differencing
    print(f"After {differencing_count} differencing(s), the ADF p-value is:␣
↪{get_adf_p_value(current_series)}")


# Check if series is stationary after differencing
if get_adf_p_value(current_series) < alpha:
    print(f"Series is stationary after {differencing_count} differencings.")
    print("")
else:
    print("Max differencings reached without achieving stationarity.")
```

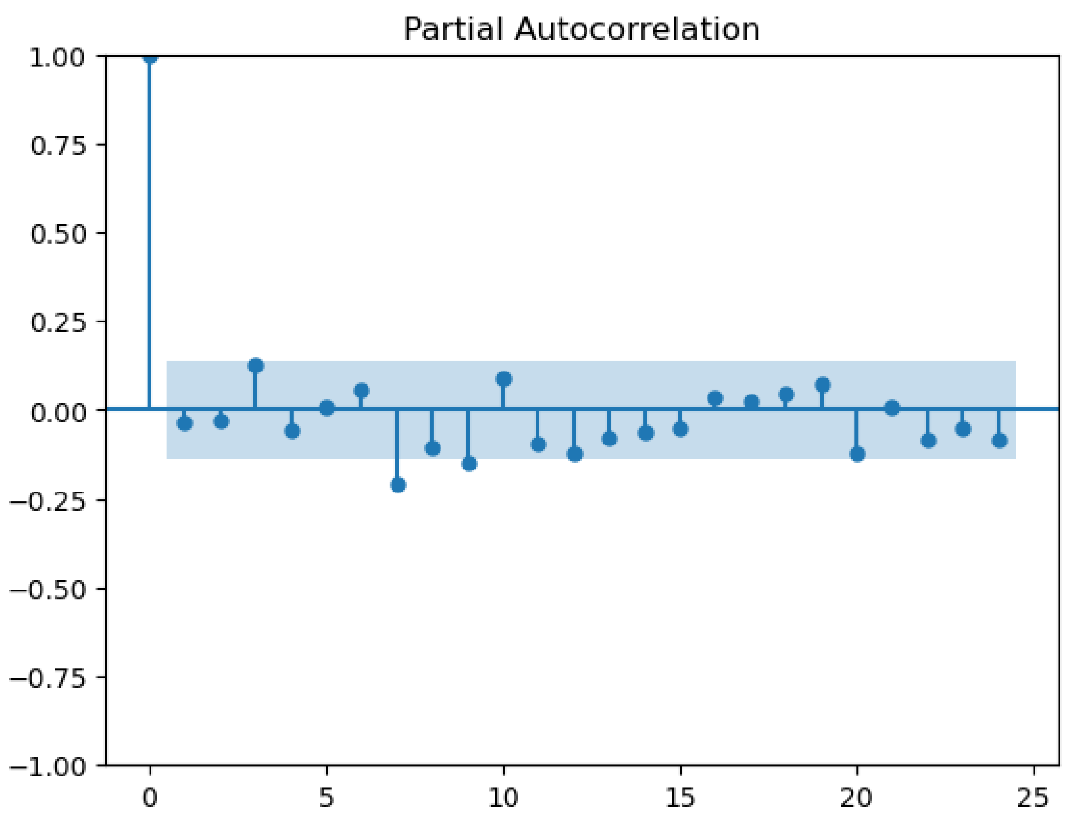
```
defferentiation
After 1 differencing(s), the ADF p-value is: 1.2426670145880777e-08
Series is stationary after 1 differencings.
```

Plot the new time series and acf and pcf

[5]:
```
plot_series(time_series_df[f'diff_prices_{differencing_count}'].dropna())
```


0.15
0.10
0.05
0.00
−0.05
−0.10
2009
2011
2013
2015
2017
2019
2021
2023
time


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25



Print significant auto and partial autocorelations

```
[6]: print(st.acf(time_series_df[f'diff_prices_{differencing_count}'].dropna(),␣
     ↪qstat=True))
```

```
(array([ 1.        , -0.03365732, -0.02826342,  0.12864677, -0.06457575,
        0.00608376,  0.07312028, -0.22007495, -0.09050825, -0.10975887,
        0.04149674, -0.08338499, -0.12708712, -0.05490398, -0.03557711,
       -0.02422668,  0.11440957,  0.00193089,  0.05805922,  0.12820226,
       -0.05282868,  0.06202194, -0.01250253, -0.0767693 ]), array([ 0.23337674,
0.39876442,  3.84240378,  4.71444248,  4.72222158,
        5.85165179, 16.1350117 , 17.88321363, 20.46742119, 20.83871736,
       22.34575616, 25.86475895, 26.52500207, 26.8036972 , 26.9336183 ,
       29.84656821, 29.84740237, 30.60566648, 34.3229306 , 34.95758643,
       35.83715387, 35.87309297, 37.23564319]), array([0.62903125, 0.81923671,
0.27899224, 0.31787164, 0.45071202,
       0.44001183, 0.0239111 , 0.0221193 , 0.01523701, 0.0222467 ,
       0.02182736, 0.01121677, 0.01444139, 0.02041631, 0.02928188,
       0.01881423, 0.02746681, 0.03195811, 0.01682951, 0.02033133,
       0.022804  , 0.03133063, 0.03071251]))
```