EXERCISE 4

A paper published in the Journal of Quality Technology (Cryer e Ryan, 1990, vol.22 no. 3, p. 189) presented the data of a chemical process where the measured variable is a color property. Data are stored in the file `ESE3_es4_dataset.csv`.

1. Identify a model for the measured variable and compute a confidence interval on estimated coefficients.
2. Compute the prediction for the next process outcome.

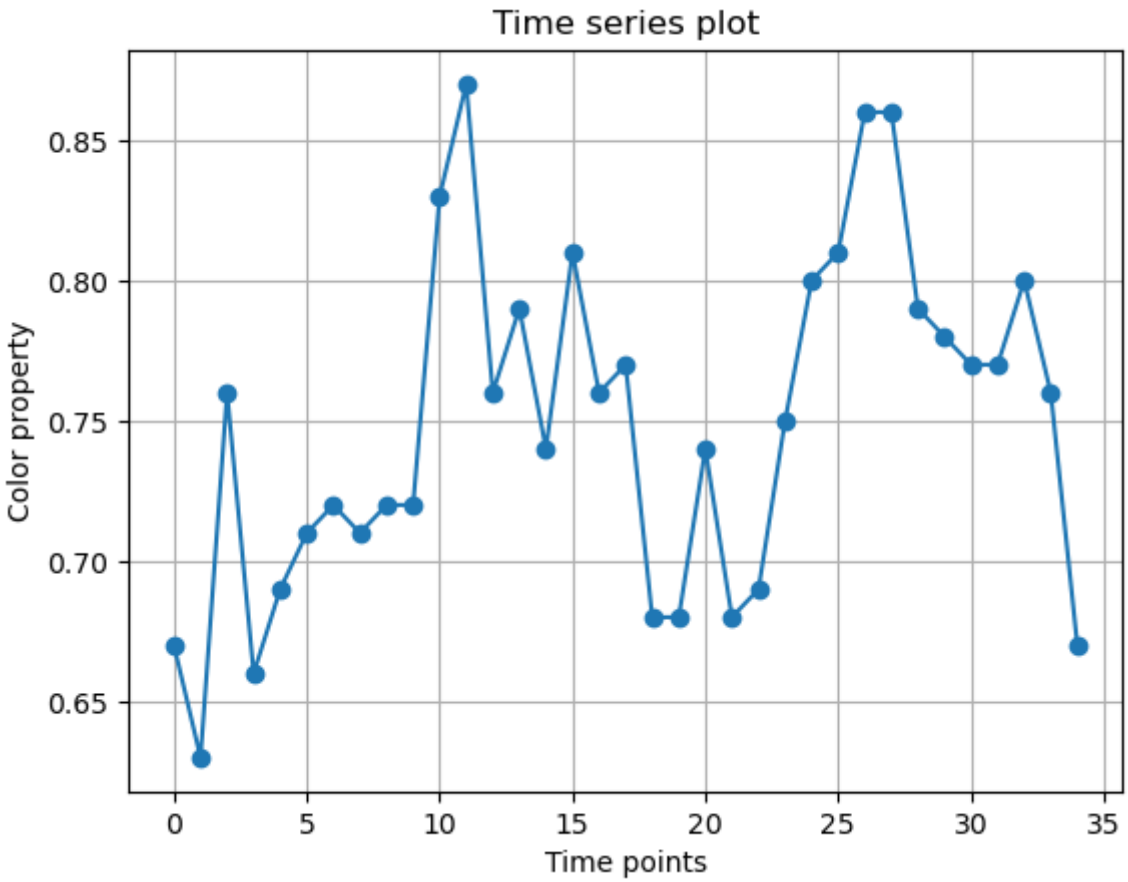
Point 1

Identify a model for the measured variable and compute a confidence interval on estimated coefficients.

> ### Solution
>
> First, we visually inspect the data.

In [ ]:
```
#Import the necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import stats

#Import the dataset
data = pd.read_csv('ESE3_es4_dataset.csv')

# Inspect the dataset
print(data.head())

#Time series plot
plt.plot(data, 'o-')
plt.title('Time series plot')
plt.xlabel('Time points')
plt.ylabel('Color property')
plt.grid()
plt.show()
```

```
    Ex4
0  0.67
1  0.63
2  0.76
3  0.66
4  0.69
```



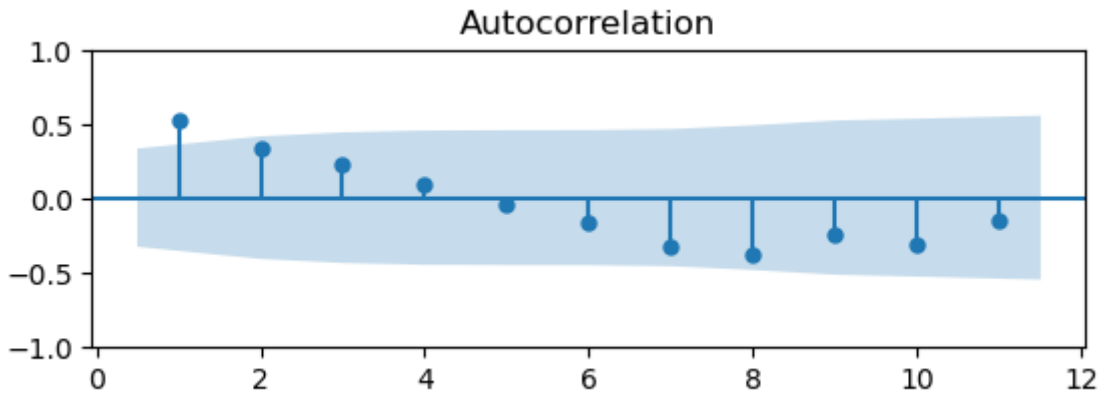
> Now, let's verify randomness: we perform the runs test.

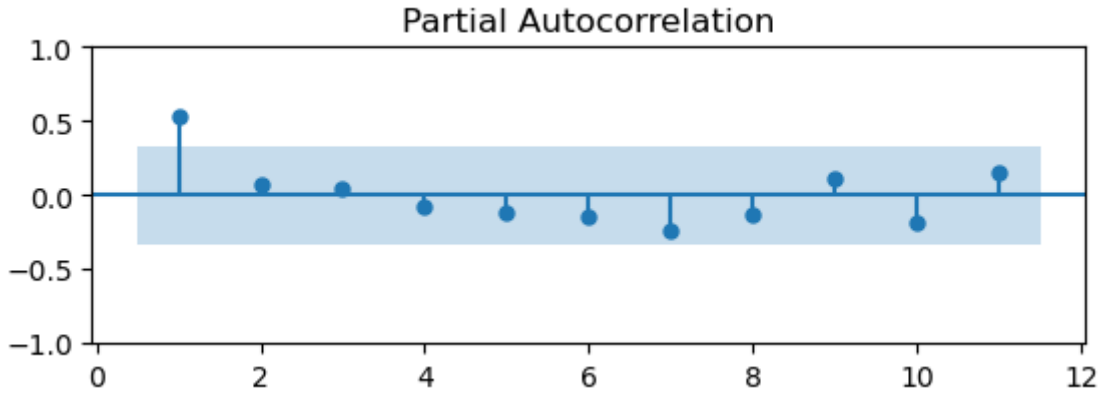
In [ ]:
```
# Import the necessary libraries for the runs test
from statsmodels.sandbox.stats.runs import runstest_1samp

_, pval_runs = runstest_1samp(data['Ex4'], correction=False)
print('Runs test p-value = {:.3f}'.format(pval_runs))

if pval_runs<0.05:
    print('The null hypothesis is rejected: the process is not random')
else:
    print('The null hypothesis is accepted: the process is random')
```

```
Runs test p-value = 0.001
The null hypothesis is rejected: the process is not random
```

> Alwan defines this kind of process as a «meandering process»: a process with successive observations tending to be close together in value (locally).
>
> Let's plot autocorrelation and partial autocorrelation functions.

In [ ]:
```
#ACF and PACF
# Plot the acf and pacf using the statsmodels library
import statsmodels.graphics.tsaplots as sgt

fig, ax = plt.subplots(2, 1)
sgt.plot_acf(data['Ex4'], lags = int(len(data)/3), zero=False, ax=ax[0])
fig.subplots_adjust(hspace=0.5)
sgt.plot_pacf(data['Ex4'], lags = int(len(data)/3), zero=False, ax=ax[1], method =
plt.show()
```





> We can see from the plots that the process seems autocorrelated. Let's test autocorrelation at lag 1 through the Bartlett's test.

In [ ]:
```
from statsmodels.tsa.stattools import acf, pacf

#autocorrelation function
[acf_values, lbq, _] = acf(data, nlags = int(len(data)/3) , qstat=True, fft = Fals
#partial autocorrelation function
pacf_value = pacf(data, nlags = int(len(data)/3))

#Bartlett's test at lag n_lag
lag_test = 1
rk=abs(acf_values[lag_test])
print('Test statistic rk= %f' % rk)

#Critical values
alpha = 0.05
z_alpha2=stats.norm.ppf(1-alpha/2)
print('Rejection region starts at %f' % (z_alpha2/np.sqrt(len(data))))


if rk>z_alpha2/np.sqrt(len(data)):
    print('The null hypothesis is rejected')
else: print('The null hypothesis is accepted')
```

```
Test statistic rk= 0.528209
Rejection region starts at 0.331294
The null hypothesis is rejected
```

> The p-value is less than 0.05, so we can reject the null hypothesis of no autocorrelation at lag 1.

> Positive correlation reflects the tendency of observations close in time to be close in value. Let's visualize this using a scatterplot of variable at time t vs. variable at time $(t-1)$.

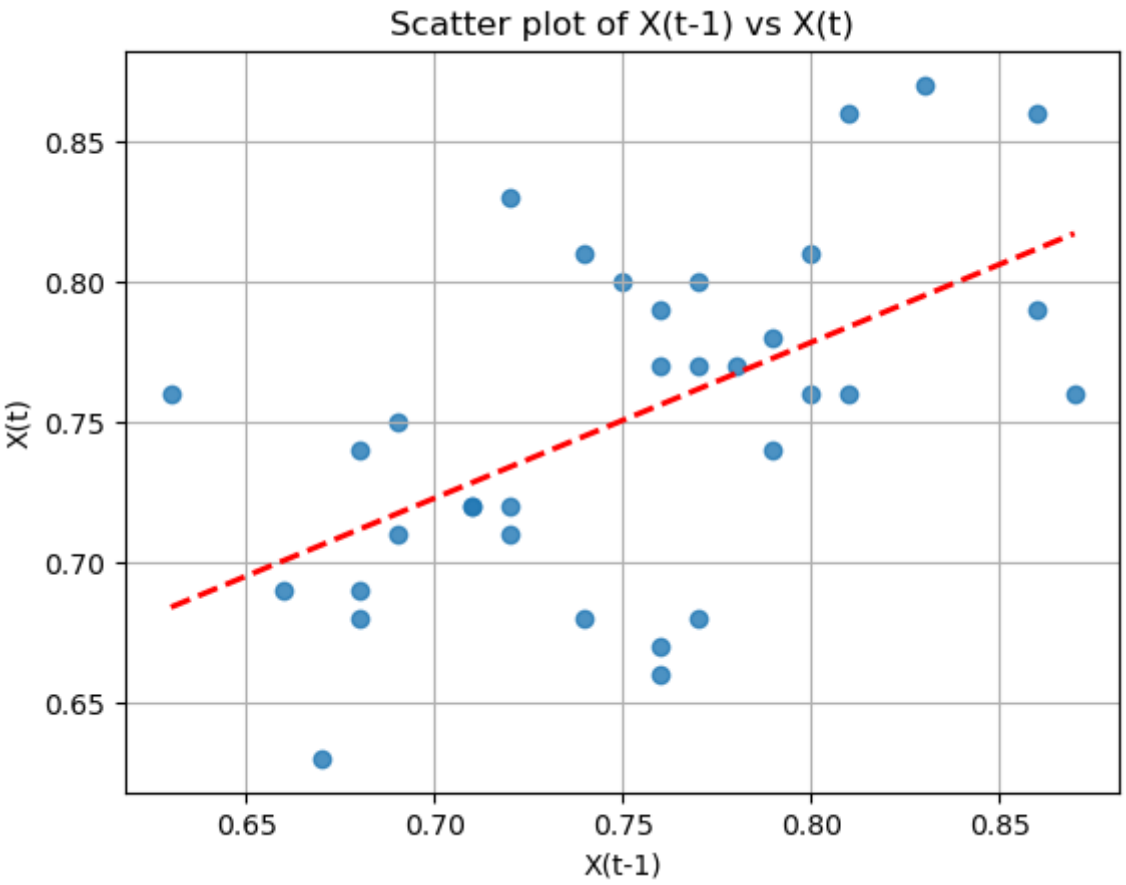
In [ ]:
```
# Calculate the lag1 from data
data['lag1'] = data['Ex4'].shift(1)
```

In [ ]:
```
data.head()
```

Out[ ]:

	Ex4	lag1
0	0.67	NaN
1	0.63	0.67
2	0.76	0.63
3	0.66	0.76
4	0.69	0.66

In [ ]:
```
# Create scatterplot with regression line using seaborn and set axis labels
import seaborn as sns
sns.regplot(x='lag1', y='Ex4', data=data, fit_reg=True, ci=None, line_kws={'color'
plt.title('Scatter plot of X(t-1) vs X(t)')
plt.xlabel('X(t-1)')
plt.ylabel('X(t)')
plt.title('Scatter plot of X(t-1) vs X(t)')
plt.grid()
```



> The most suitable model seems to be AR(1) with positive coefficient (meandering process).

> An AR(1) model can be fitted via linear regression, using *lag1* as regressor.
>
> To do so, we can use the `OLS` function from the `statsmodels` package. The function `OLS` takes as input:
>
> - The time series, `y` (i.e., the variable to be modeled)
> - The regressors, `x` (i.e., the lag1 variable)
>
> After defining the model, we can fit it using the method `fit()`.

In [ ]:
```
#calculate a regression model with constant and lag1
import statsmodels.api as sm

x = data['lag1'][1:]
```

In [ ]:
```
# Add a constant to the model (it is equivalent to adding a column of ones to the d
x = sm.add_constant(data['lag1'][1:])
```

Short reminder about regression

$$\boldsymbol{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} \sim iid\left(\mathbf{0}, \sigma^2\mathbf{I}\right)^{*}$$

$\boldsymbol{y}$ is $nx1$

$\mathbf{X}$ is nxp (p = number of parameters), e.g.: $y = \beta_0 + \beta_1 x + \varepsilon \rightarrow$ p=2

$\boldsymbol{\beta}$ is $px1$

$\boldsymbol{\varepsilon}$ is $nx1$

$$\mathbf{X} = \begin{bmatrix} \mathbf{1} & \boldsymbol{x_{11}} & \cdots & \boldsymbol{x_{1k}} \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{1} & \boldsymbol{x_{n1}} & \cdots & \boldsymbol{x_{nk}} \end{bmatrix} \qquad p = k + 1$$

$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^{\boldsymbol{T}}\mathbf{X})^{-1}\mathbf{X}^{\boldsymbol{T}}\boldsymbol{y}$	Estimated parameters	$\mathrm{E}(\widehat{\boldsymbol{\beta}}) = \boldsymbol{\beta}$ $\mathrm{V}(\widehat{\boldsymbol{\beta}}) = \sigma^2(\mathbf{X}^{\boldsymbol{T}}\mathbf{X})^{-1}$
$\widehat{\boldsymbol{y}} = \mathbf{X}\widehat{\boldsymbol{\beta}}$	Estimated response	
$\widehat{\sigma^2} = MS_E$	Estimated variance	
$\boldsymbol{e} = \boldsymbol{y} - \widehat{\boldsymbol{y}}$	Residuals	

* But to make inference we need normality assumption (NID)

In [ ]:
```
y = data['Ex4'][1:]
model = sm.OLS(y, x).fit()
```

In [ ]:
```
# Print out the statistics
import qda
qda.summary(model)
```

```
REGRESSION EQUATION
-------------------
Ex4 =  + 0.334 const + 0.555 lag1

COEFFICIENTS
------------
 Term   Coef  SE Coef  T-Value  P-Value
const 0.3343   0.1108   3.0171   0.0050
 lag1 0.5549   0.1471   3.7734   0.0007

MODEL SUMMARY
-------------
     S   R-sq  R-sq(adj)
0.0509 0.3079     0.2863

ANALYSIS OF VARIANCE
--------------------
    Source   DF  Adj SS  Adj MS  F-Value  P-Value
Regression  1.0  0.0369  0.0369  14.2383   0.0007
     const  1.0  0.0236  0.0236   9.1029   0.0050
      lag1  1.0  0.0369  0.0369  14.2383   0.0007
     Error 32.0  0.0829  0.0026      NaN      NaN
     Total 33.0  0.1198     NaN      NaN      NaN
```

Test for significance of regression

- $H_0 : \beta_1 = \beta_2 = \beta_k = 0$
- $H_1 : \exists \beta_i \neq 0$

Partition of the variance: $SS_{TOT} = SS_{REG} + SS_E$

Degrees of freedom: $(n-1) = (p-1) + (n-p)$

Definition of the sum of squares:

- $SS_{TOT} = \sum_{i=1}^{n} (y_i - \bar{y_i})^2$
- $SS_{REG} = \sum_{i=1}^{n} (\hat{y}_i - \bar{y_i})^2$
- $SS_E = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$

If H_0 is true:

- $\frac{SS_{REG}}{\sigma^2} \sim \chi^2_{p-1}$
- $\frac{SS_E}{\sigma^2} \sim \chi^2_{n-p}$

then:

$$F_0 = \frac{SS_{REG}/(p-1)}{SS_E/(n-p)} = \frac{MS_{REG}}{MS_E} \sim F_{p-1,n-p}$$

Reject H_0 if: $F_0 > F_{\alpha,p-1,n-p}$

The F-statistic and the corresponding p-value is displayed in the **ANALYSIS OF VARIANCE** table.

Test for individual coefficients

- $H_0 : \beta_i = 0$
- $H_1 : \beta_i \neq 0$

Test statistic: $T_0 = \frac{\hat{\beta}_i}{se(\hat{\beta}_i)} = \frac{\hat{\beta}_i}{\sqrt{\hat{\sigma}^2 C_i}}$

Where C_i is the i-th diagonal element of the $(\mathbf{X}^T\mathbf{X})^{-1}$ matrix.

If H_0 is true: $T_0 \sim t_{n-p}$

Reject H_0 if: $|T_0| > t_{\alpha/2,n-p}$

The t-statistic and the corresponding p-value is displayed in the **COEFFICIENTS** table.

Short reminder about regression

Test for significance of regression

H_0: $\beta_1 = \beta_2 = \ldots = \beta_k = 0$
H_1: $\exists\ \beta_i \neq 0$
(usually we exclude the constant term)

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Regression	1	0,03688	0,036876	14,24	0,001
Ex4lag1	1	0,03688	0,036876	14,24	0,001
Error	32	0,08288	0,002590		
Lack-of-Fit	16	0,03192	0,001995	0,63	0,820
Pure Error	16	0,05096	0,003185		
Total	33	0,11975			

Partition of the variance: $SS_{TOT} = SS_{REG} + SS_E$
Degrees of freedom: $n-1$ $p-1$ $n-p$
E.g., simple regression:

$$SS_{TOT} = \sum_{i=1}^{n}(y_i - \bar{y})^2 \qquad SS_{REG} = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$$

$$SS_E = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

The AR(1) model is: $X_t = 0.334 + 0.555 \cdot X_{t-1} + e_t$

Check on residuals

$\varepsilon \sim NID(\mathbf{0}, \sigma^2\mathbf{I})$

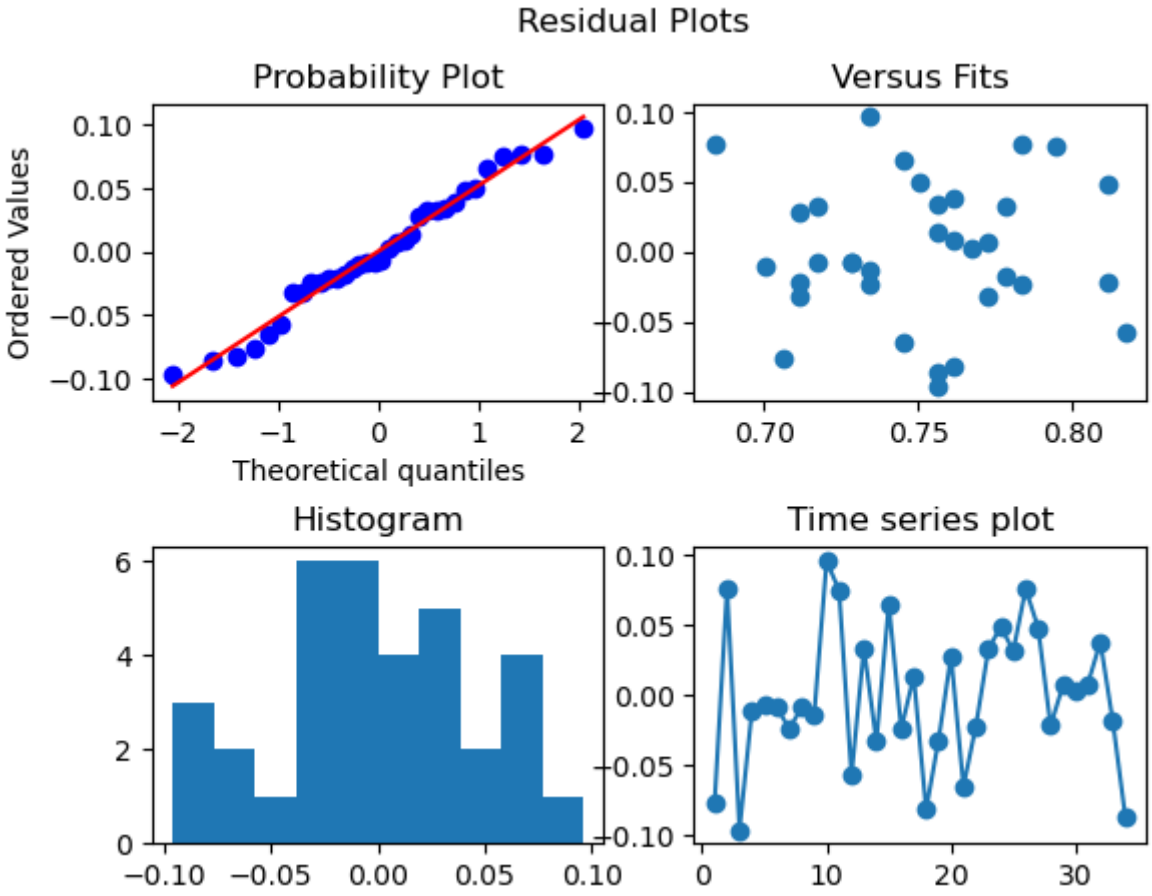
Normal and independently distributed

Let's check residuals. To accept the model, we also need to verify the assumption that residuals are normal and independently distributed.

> First, we check normality of residuals.

In [ ]:
```
#NORMALITY OF RESIDUALS

fig, axs = plt.subplots(2, 2)
fig.suptitle('Residual Plots')

axs[0,0].set_title('Normal probability plot')
stats.probplot(model.resid, dist="norm", plot=axs[0,0])

axs[0,1].set_title('Versus Fits')
axs[0,1].scatter(model.fittedvalues, model.resid)

fig.subplots_adjust(hspace=0.5)

axs[1,0].set_title('Histogram')
axs[1,0].hist(model.resid)

axs[1,1].set_title('Time series plot')
axs[1,1].plot(np.arange(1, len(model.resid)+1), model.resid, 'o-')

_, pval_SW_res = stats.shapiro(model.resid)
print('Shapiro-Wilk test p-value on the residuals = %.3f' % pval_SW_res)
```

```
Shapiro-Wilk test p-value on the residuals = 0.579
```

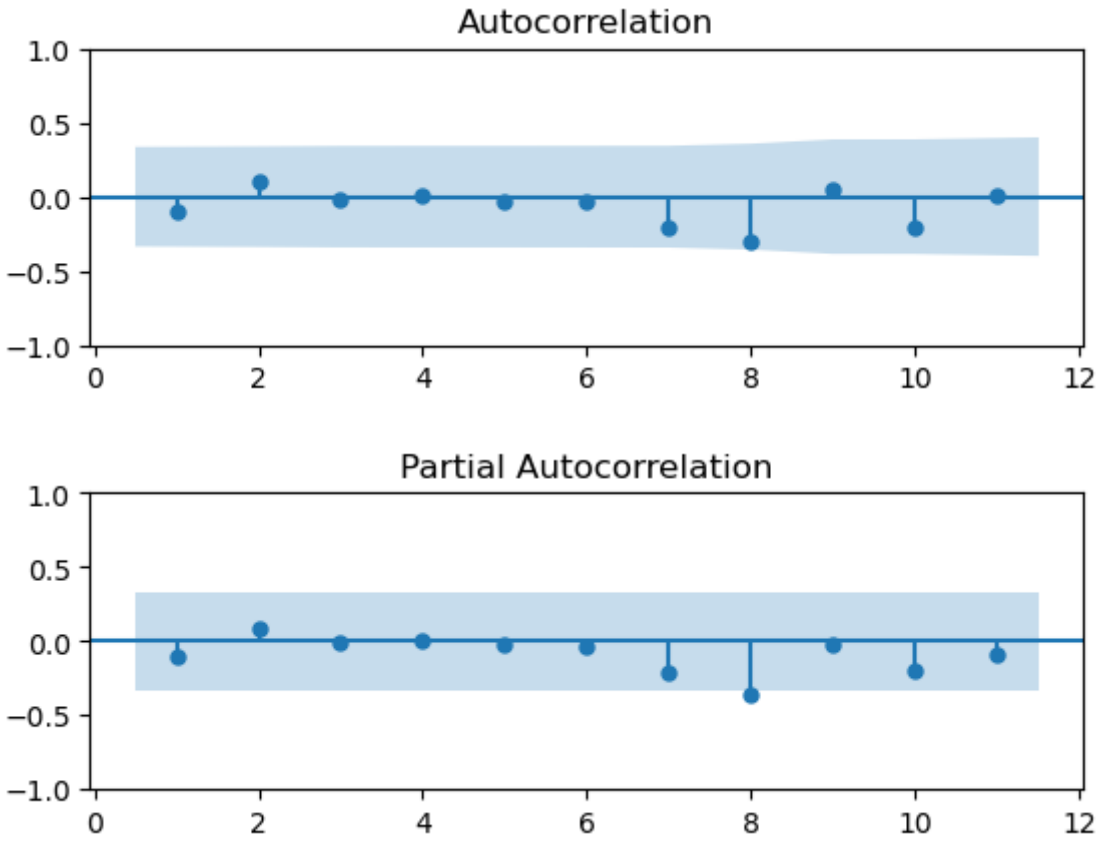


> - The normal probability plot (top left) shows a normal distribution of the residuals.
> - The histogram (bottom left) confirms normality.
> - On top right, residuals versus fits are shown.
> - The bottom right panel shows the time series plot of the residuals.

> Based on the Shapiro-Wilk test, the assumption on normality of residuals is accepted.

In [ ]:
```
#RANDOMNESS OF FESIDUALS
_, pval_runs_res = runstest_1samp(model.resid, correction=False)
print('Runs test p-value on the residuals = {:.3f}'.format(pval_runs_res))
fig, ax = plt.subplots(2, 1)
sgt.plot_acf(model.resid, lags = int(len(data)/3), zero=False, ax=ax[0])
fig.subplots_adjust(hspace=0.5)
sgt.plot_pacf(model.resid, lags = int(len(data)/3), zero=False, ax=ax[1],
              method = 'ywm')
plt.show()
```

```
Runs test p-value on the residuals = 0.742
```



> Residuals do not show autocorrelation. Assumption on randomness of residuals is accepted.

> Let's compute the confidence interval on the AR(1) coefficient

a) Confidence interval on the AR(1) coefficient

$$t_0 = \frac{\hat{\beta}_i - \beta_i}{se(\hat{\beta}_i)} \sim t_{n-p} \quad \text{Student - t with } n\text{-}p \text{ dof}$$

p=n° predictors +1=2
n=34

dof of error term

$$-t_{\alpha/2,n-p} \leq \frac{\hat{\beta}_i - \beta_i}{se(\hat{\beta}_i)} \leq t_{\alpha/2,n-p}$$

$$\hat{\beta}_i - t_{\alpha/2,n-p} se(\hat{\beta}_i) \leq \beta_i \leq \hat{\beta}_i + t_{\alpha/2,n-p} se(\hat{\beta}_i)$$

Inverse Cumulative Distribution Function
Student's t distribution with 32 DF

P(X <= x) x
0.025 -2.03693

$$\alpha = 5\% \Rightarrow t_{0.025,32} = 2.0369$$

a) Confidence interval on the AR(1) c

$$\hat{\beta}_i - t_{\alpha/2,n-p} se(\hat{\beta}_i) \leq \beta_i \leq \hat{\beta}_i + t_{\alpha/2,n-p} se(\hat{\beta}_i)$$

$$t_{0.025,32} = 2.0369$$

$$\hat{\beta}_1 = 0.555$$

$$se(\hat{\beta}_1) = 0.147$$

From regression table

Coefficients

Term	Coef	SE Coef	T-Value	P-Value	VIF
Constant	0,334	0,111	3,02	0,005	
Ex4lag1	0,555	0,147	3,77	0,001	1,00

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
0,0508911	30,79%	28,63%	21,34%

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Regression	1	0,03688	0,036876	14,24	0,001
Ex4lag1	1	0,03688	0,036876	14,24	0,001
Error	32	0,08288	0,002590		
Lack-of-Fit	16	0,03192	0,001995	0,63	0,820
Pure Error	16	0,05096	0,003185		
Total	33	0,11975			

$$0.555 - 2.0369(0.147) \leq \beta_1 \leq 0.555 + 2.0369(0.147)$$
$$0.2553 \leq \beta_1 \leq 0.8545$$

You can access the values of the estimated coefficients and their standard errors from the `RegressionResult` object that is the output of the `fit()` method.

In [ ]:
```
# The RegressionResult object was stored in the variable model
# Get the estimated coefficient
beta1 = model.params['lag1']
print('The estimated coefficient beta1 is %.3f' % beta1)

se_beta1 = model.bse['lag1']
print('The standard error of the estimated coefficient beta1 is %.3f' % se_beta1)

alpha = 0.05
n = len(data)
t_alpha2 = stats.t.ppf(1-alpha/2, n-2)

CI_beta1 = [beta1 - t_alpha2*se_beta1, beta1 + t_alpha2*se_beta1]

print('The confidence interval for beta1 is [%.3f, %.3f]' % (CI_beta1[0], CI_beta1
```

```
The estimated coefficient beta1 is 0.555
The standard error of the estimated coefficient beta1 is 0.147
The confidence interval for beta1 is [0.256, 0.854]
```

> Alternatively, you can use the `conf_int()` method to compute the confidence interval on the coefficients of the model for any specified value of alpha.

In [ ]:
```
# Calculate the confidence interval
CI_beta1 = model.conf_int(alpha=0.05).loc['lag1']
print('The confidence interval for beta1 is [%.3f, %.3f]' % (CI_beta1[0], CI_beta1
```

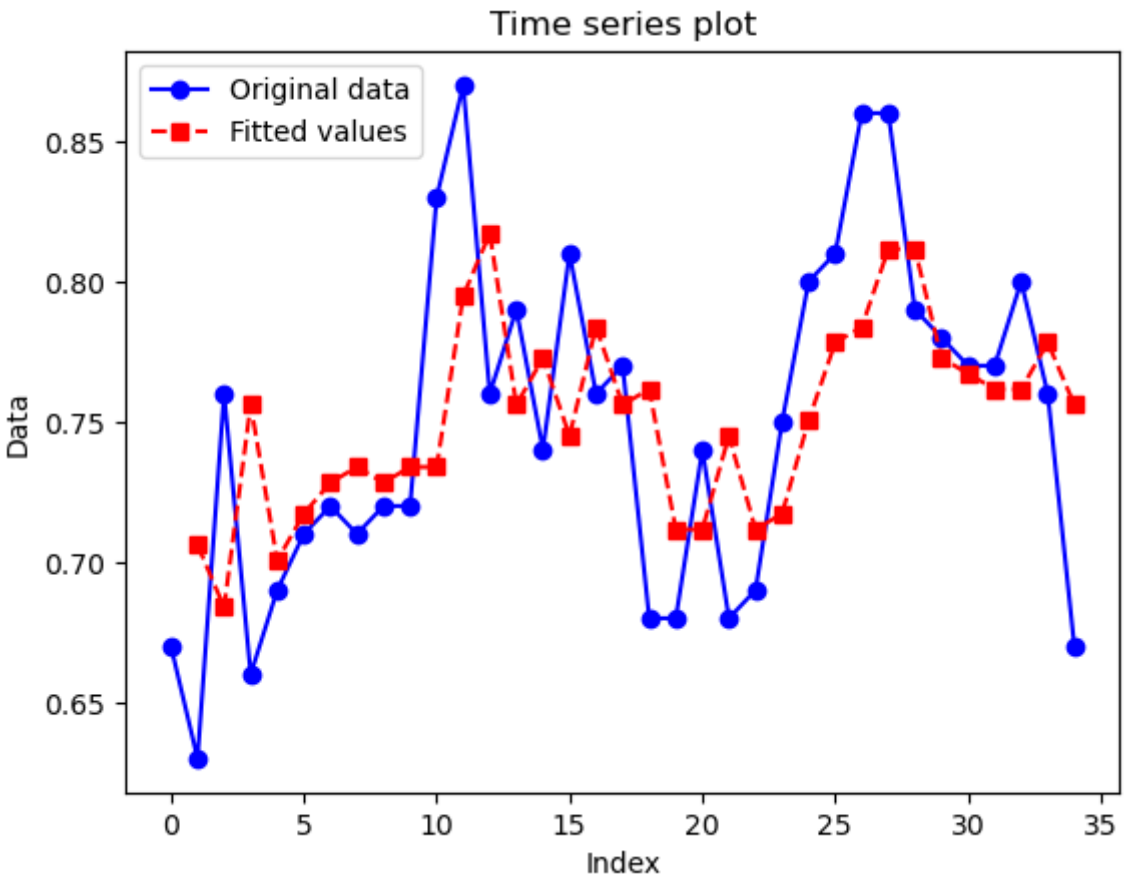
```
The confidence interval for beta1 is [0.255, 0.854]
```

Point 2

Compute the prediction for the next process outcome.

First, we plot the data and the fitted values of the AR(1) model

In [ ]:
```
# plot prediction interval
plt.plot(data['Ex4'], 'o-', color='blue', label='Original data')
plt.plot(model.fittedvalues, 's--', color='red', label='Fitted values')
plt.title('Time series plot')
plt.xlabel('Index')
plt.ylabel('Data')
plt.legend()
plt.show()
```



b) Prediction of next process outcome (36° observation):

$$X_{36} = 0.3343 + 0.5549\, X_{35} + e_{36} = 0.3343 + 0.5549 \cdot 0.67 + e_{36} = 0.7061 + e_{36}$$

$$\hat{X}_{36} = 0.3343 + 0.5549\, X_{35} = 0.3343 + 0.5549 \cdot 0.67 = 0.7061$$

Last observation in the time series (known value)

Let Y be $X_t(response)$ and X be X_{t-1} $(predictor)$
Let X^*=$X_{35} = 0.67$

Confidence interval (mean response)

$$\hat{Y} \pm t_{\alpha/2}, n-p \sqrt{\hat{\sigma}_\varepsilon^2 \left[\frac{1}{n} + \frac{(X^* - \bar{X})^2}{(n-1)S_X^2}\right]}$$

Prediction interval

$$\hat{Y} \pm t_{\alpha/2}, n-p \sqrt{\hat{\sigma}_\varepsilon^2 \left[1 + \frac{1}{n} + \frac{(X^* - \bar{X})^2}{(n-1)S_X^2}\right]}$$

> Let's compute the prediction of the next process outcome (36th observation)

b) Prediction of next process outcome (36° observation):

Let Y be $X_t(response)$ and X be X_{t-1} $(predictor)$
Let X^*=$X_{35} = 0.67$

Confidence interval (mean response)

$$\hat{Y} \pm t_{\alpha/2}, n-p \sqrt{\hat{\sigma}_\varepsilon^2 \left[\frac{1}{n} + \frac{(X^* - \bar{X})^2}{(n-1)S_X^2}\right]}$$

Prediction interval

$$\hat{Y} \pm t_{\alpha/2}, n-p \sqrt{\hat{\sigma}_\varepsilon^2 \left[1 + \frac{1}{n} + \frac{(X^* - \bar{X})^2}{(n-1)S_X^2}\right]}$$

$\hat{Y} = 0.7061$ $\quad \bar{X} = 0.7512$ $\quad S_X^2 = 0.0036$

Descriptive statistics (minitab)
Lag1 time series!

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
0,0508911	30,79%	28,63%	21,34%

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Regression	1	0,03688	0,036876	14,24	0,001
Ex4lag1	1	0,03688	0,036876	14,24	0,001
Error	32	0,08288	0,002590		
Lack-of-Fit	16	0,03192	0,001995	0,63	0,820
Pure Error	16	0,05096	0,003185		
Total	33	0,11975			

$S_Y^2 = \hat{\sigma}_\varepsilon^2 = MS_E = 0.0509\text{^}2 = 0.00259$

$t_{\alpha/2}, n-2 = 2.034$

In [ ]:
```
Xbar = data['lag1'].mean()          # sample mean of the regressor
S2_X = data['lag1'].var()           # sample variance of the regressor

p = len(model.model.exog_names)     # number of regressors
S2_Y = np.var(model.resid, ddof=p)  # sample variance of residuals

alpha = 0.05
t_alpha2 = stats.t.ppf(1-alpha/2, n-2)
```

In [ ]:
```
#predict future outcomes using the regression model
last_lag = data['Ex4'].iloc[-1]
print('X_35 = %.3f' % last_lag)
```

```
#predict the next value
Yhat = model.predict([1,last_lag])
print('Next process outcome = %.3f' % Yhat)
```

```
X_35 = 0.670
Next process outcome = 0.706
```

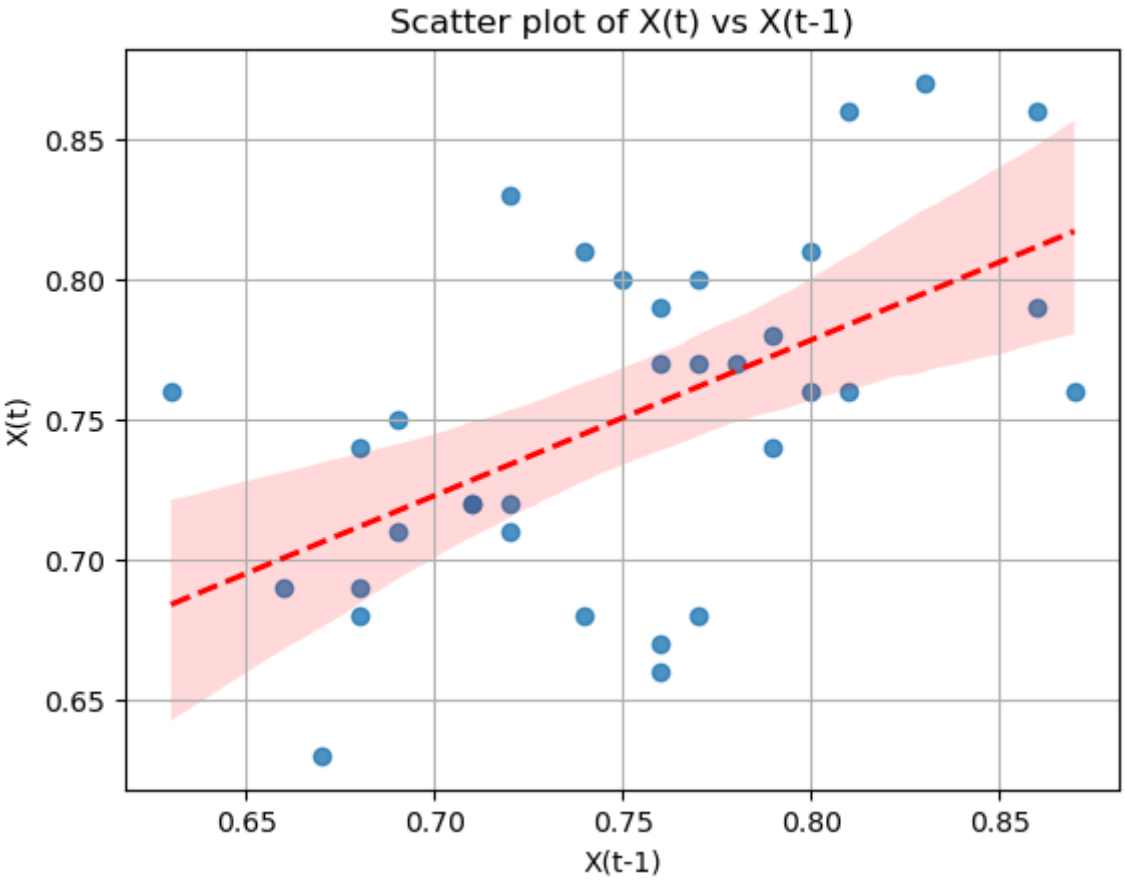
In [ ]:
```
# Calculate the confidence interval
CI = [Yhat - t_alpha2*np.sqrt(S2_Y*(1/n + ((last_lag - Xbar)**2)/((n-1)*S2_X))),
        Yhat + t_alpha2*np.sqrt(S2_Y*(1/n + ((last_lag - Xbar)**2)/((n-1)*S2_X)))]
print('The confidence interval for the mean response is [%.3f, %.3f]' % (CI[0], CI
```

```
The confidence interval for the next value is [0.676, 0.736]
```

In [ ]:
```
# Calculate the PREDICTION interval
PI = [Yhat - t_alpha2*np.sqrt(S2_Y*(1 + 1/n + ((last_lag - Xbar)**2)/((n-1)*S2_X))
        Yhat + t_alpha2*np.sqrt(S2_Y*(1 + 1/n + ((last_lag - Xbar)**2)/((n-1)*S2_X

print('The prediction interval for the next value is [%.5f, %.5f]' % (PI[0], PI[1]
```

```
The confidence interval for the next value is [0.59843, 0.81383]
```

In [ ]:
```
sns.regplot(x='lag1', y='Ex4', data=data, fit_reg=True, ci=95, line_kws={'color':
plt.title('Scatter plot of X(t) vs X(t-1)')
plt.xlabel('X(t-1)')
plt.ylabel('X(t)')
plt.grid()
```



Alternatively, you can use the `get_prediction()` method to compute the prediction for the next process outcome and the associated confidence and prediction intervals.

In [ ]:
```
# compute the prediction interval
prediction_df = model.get_prediction([1,last_lag]).summary_frame(alpha=0.05)
print(prediction_df)
```

```
     mean   mean_se  mean_ci_lower  mean_ci_upper  obs_ci_lower  obs_ci_upper
0  0.7061    0.0148          0.676         0.7363        0.5982        0.8141
```

To visualize the prediction interval, you can evaluate the bounds across the range of the time series using the `get_prediction()` method.

In [ ]:
```
# get the range of values for the regressor
x_range = np.linspace(data['lag1'].min(), data['lag1'].max(), 100)

# add a constant to the regressor
x_range = sm.add_constant(x_range)

# get the prediction interval for each value of the regressor
prediction_df = model.get_prediction(x_range).summary_frame(alpha=0.05)

# plot the data and the intervals
plt.plot(data['lag1'], data['Ex4'], 'o', color='blue', label='Original data')
plt.plot(x_range[:,1], prediction_df['mean'], '--', color='red', label='Fitted valu
plt.fill_between(x_range[:,1], prediction_df['obs_ci_lower'], prediction_df['obs_c
plt.fill_between(x_range[:,1], prediction_df['mean_ci_lower'], prediction_df['mean_
plt.title('Scatter plot of X(t) vs X(t-1)')
plt.xlabel('X(t-1)')
plt.ylabel('X(t)')
plt.show()
```