In [4]:
```
# Import all required libraries

#Suppress all warnings
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import os
import plotly.io as pio

pio.templates.default = "plotly_white"

from pathlib import Path
from tqdm.autonotebook import tqdm
import warnings
from src.utils.general import LogTime
import plotly.express as px

pio.templates.default = "plotly_white"

import matplotlib.pyplot as plt
import joblib
import random
import IPython.display
from IPython.display import display, HTML

import plotly.graph_objects as go
from plotly.subplots import make_subplots

from statsmodels.tsa.stattools import acf


from stationary_utils import check_unit_root
from stationary_utils import check_trend, check_deterministic_trend
from stationary_utils import check_seasonality
from stationary_utils import check_heteroscedastisticity

from target_transformations import AdditiveDifferencingTransformer, MultiplicativeDifferencingTransformer
from target_transformations import DetrendingTransformer
from target_transformations import DeseasonalizingTransformer
from target_transformations import LogTransformer
from target_transformations import BoxCoxTransformer
from target_transformations import AutoStationaryTransformer

np.random.seed(42)
tqdm.pandas()
```

In [5]:
```
pip install pymannkendall
```

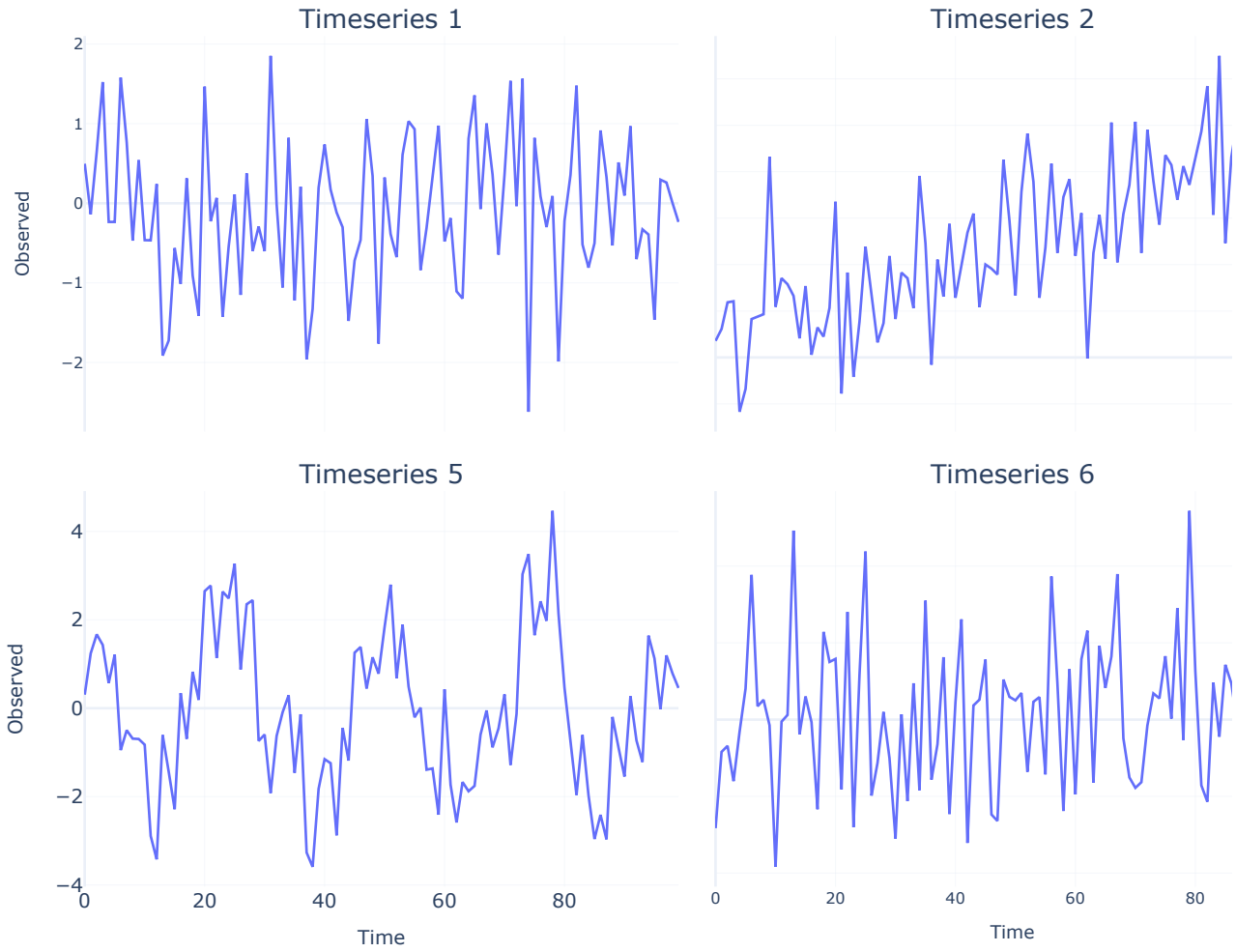
```
Requirement already satisfied: pymannkendall in c:\users\ridhi\onedrive\documents\python\envs\python39\lib\site-packages (1.4.3)
Requirement already satisfied: numpy in c:\users\ridhi\onedrive\documents\python\envs\python39\lib\site-packages (from pymannkendall)
(1.26.3)
Requirement already satisfied: scipy in c:\users\ridhi\onedrive\documents\python\envs\python39\lib\site-packages (from pymannkendall)
(1.12.0)
Note: you may need to restart the kernel to use updated packages.
```

1. GENERATING SYNTHETIC TIME SERIES

In [6]:
```
length = 100
index = pd.date_range(start="2021-11-03", periods=length)
# White Noise
y_random = pd.Series(np.random.randn(length), index=index)
# White Noise
y_random_2 = pd.Series(np.random.randn(length), index=index)
# White Noise+Trend
_y_random = pd.Series(np.random.randn(length), index=index)
t = np.arange(len(_y_random))
y_trend = _y_random+t*_y_random.mean()*0.8
# Heteroscedastic
_y_random = pd.Series(np.random.randn(length), index=index)
t = np.arange(len(_y_random))
y_hetero = (_y_random*t)
# White Noise + Seasonal
_y_random = pd.Series(np.random.randn(length), index=index)
t = np.arange(len(_y_random))
y_seasonal = (_y_random+1.9*np.cos((2*np.pi*t)/(length/4)))
# Unit root
_y_random = pd.Series(np.random.randn(length), index=index)
y_unit_root = _y_random.cumsum()

plot_df = pd.DataFrame({"Time":np.arange(100), "Timeseries 1": y_random, "Timeseries 2": y_trend, "Timeseries 4": y_unit_root, "Times
```

2. PLOT GENERATED TIME SERIES

In [7]:
```
fig = px.line(pd.melt(plot_df, id_vars="Time", value_name="Observed"), x="Time", y="Observed", facet_col="variable", facet_col_wrap=3
fig.update_yaxes(matches=None)
fig.for_each_annotation(lambda a: a.update(text=a.text.split("=")[-1], font=dict(size=16)))
fig.update_layout(
                  autosize=False,
                  width=1600,
                  height=800,
                  yaxis=dict(
                      titlefont=dict(size=15),
                      tickfont=dict(size=15),
                  ),
                  xaxis=dict(
                      titlefont=dict(size=15),
                      tickfont=dict(size=15),
                  )
)
fig.update_annotations(font_size=20)
fig.show()
```



3. CHECK AND HANDLE UNIT ROOTSPlotting Autoregressive series with different ¢

In [8]:
```
import random
import numpy as np

def generate_autoregressive_series(length, phi):
    x_start = random.uniform(-1, 1)
    y = []
    for i in range(length):
        t = x_start * phi + random.uniform(-1, 1)
        y.append(t)
        x_start = t
    return np.array(y)
```

In [9]:
```
fig = make_subplots(
    rows=3, cols=1,  # Assuming you want 3 rows and 1 column of subplots
```

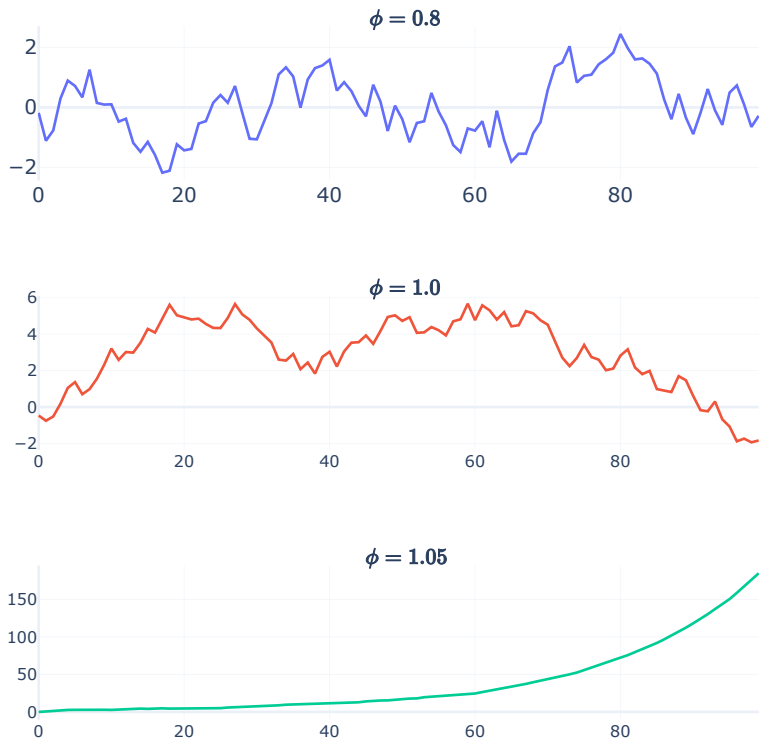
```
    subplot_titles=("$\phi=0.8$", "$\phi=1.0$", "$\phi=1.05$")
)
```

In [10]:
```
fig.append_trace(
    go.Scatter(
        x=np.arange(length),
        y=generate_autoregressive_series(length, phi=0.8),
    ),
    row=1, col=1,
)
```

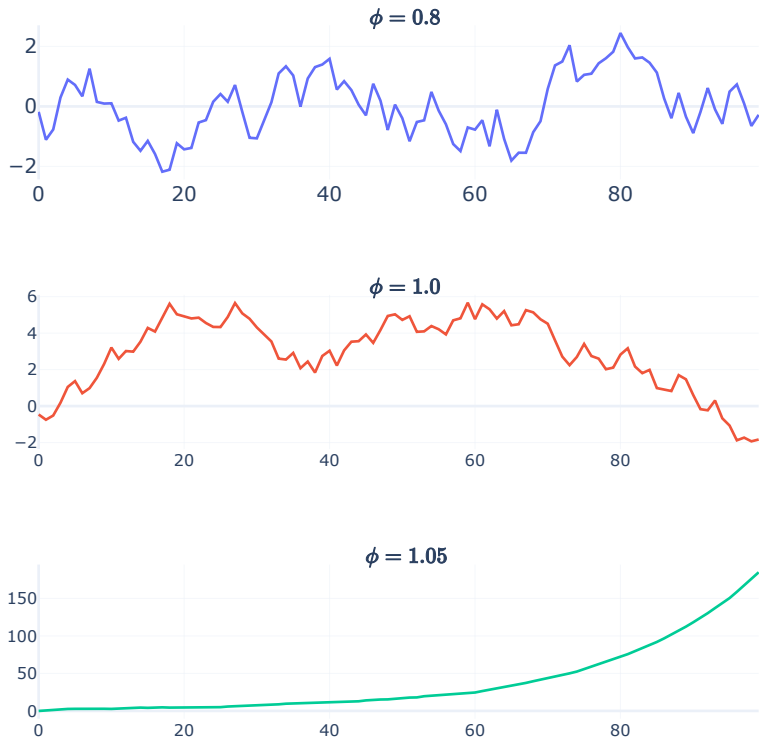
In [11]:
```
fig.append_trace(
    go.Scatter(
        x=np.arange(length),
        y=generate_autoregressive_series(length, phi=1.0),
    ),
    row=2, col=1,
)
```

In [12]:
```
fig.append_trace(
    go.Scatter(x=np.arange(length), y=generate_autoregressive_series(length, phi=1.05)),
    row=3, col=1,
)
```

In [13]:
```
fig.update_layout(
    height=700,
    width=700,
    showlegend=False,
    yaxis=dict(
        titlefont=dict(size=15),
        tickfont=dict(size=15),
    ),
    xaxis=dict(
        titlefont=dict(size=15),
        tickfont=dict(size=15),
    ),
)
```



In [14]:
```
fig.show()
```



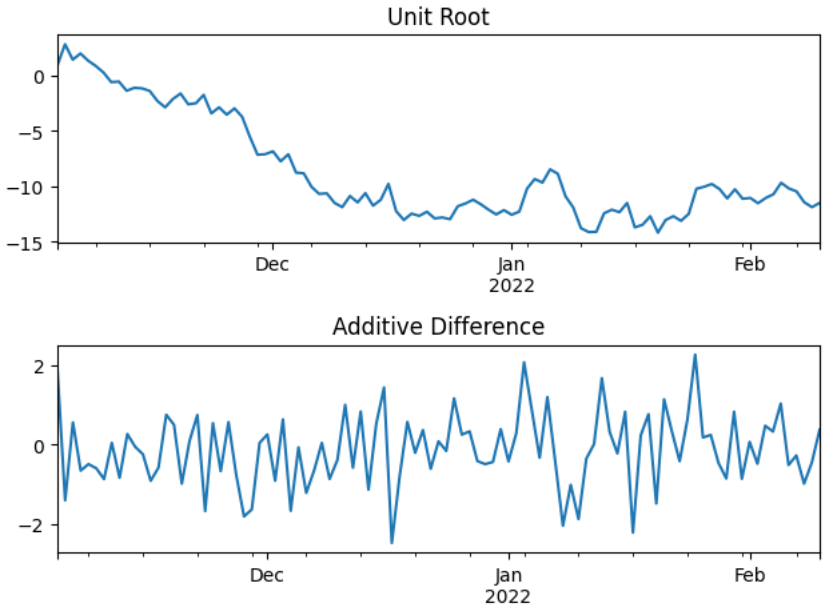
DETECTING UNIT ROOT

In [15]:
```
res = check_unit_root(y_unit_root, confidence=0.05)

print(f"Stationary: {res.stationary} | p-value: {res.results[1]}")
```

```
Stationary: False | p-value: 0.3008085997474035
```

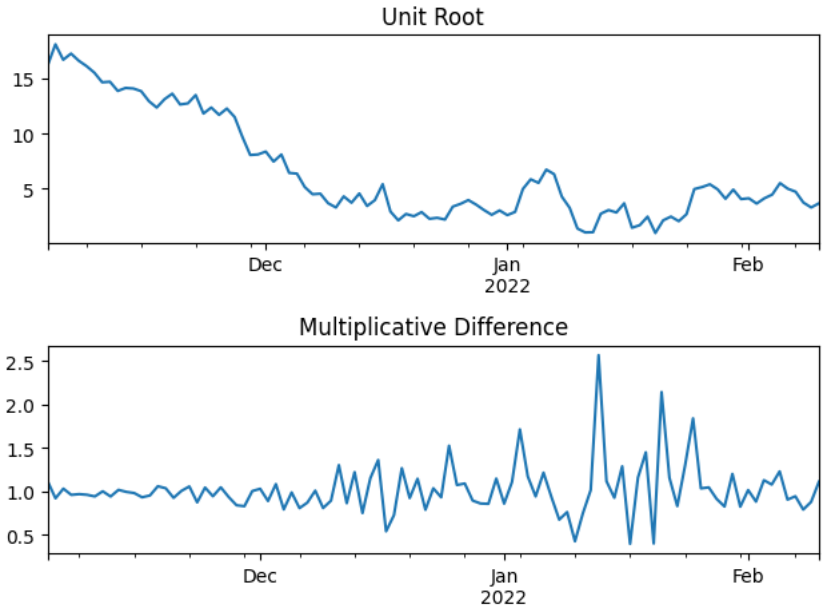
DEFFERENCING

In [16]:
```
diff_transformer = AdditiveDifferencingTransformer()
# [1:]: because differencing reduces the length of the time series by one
y_diff = diff_transformer.fit_transform(y_unit_root)[1:]
fig, axs = plt.subplots(2)
y_unit_root.plot(title="Unit Root", ax=axs[0])
y_diff.plot(title="Additive Difference", ax=axs[1])
plt.tight_layout()
plt.show()
check_unit_root(y_diff)
```



Out[16]: ADF_Test(stationary=True, results=(-10.811906709806532, 1.8992882415174895e-19, 0, 98, {'1%': -3.4989097606014496, '5%': -2.891516256916761, '10%': -2.5827604414827157}, 235.7179734900839))

In [17]:
```
_y_unit_root = y_unit_root + np.abs(y_unit_root.min())+1
diff_transformer = MultiplicativeDifferencingTransformer()
# [1:] because differencing reduces the length of the time series by one
y_diff = diff_transformer.fit_transform(_y_unit_root)[1:]
fig, axs = plt.subplots(2)
_y_unit_root.plot(title="Unit Root", ax=axs[0])
y_diff.plot(title="Multiplicative Difference",ax=axs[1])
plt.tight_layout()
plt.show()
check_unit_root(y_diff)
```



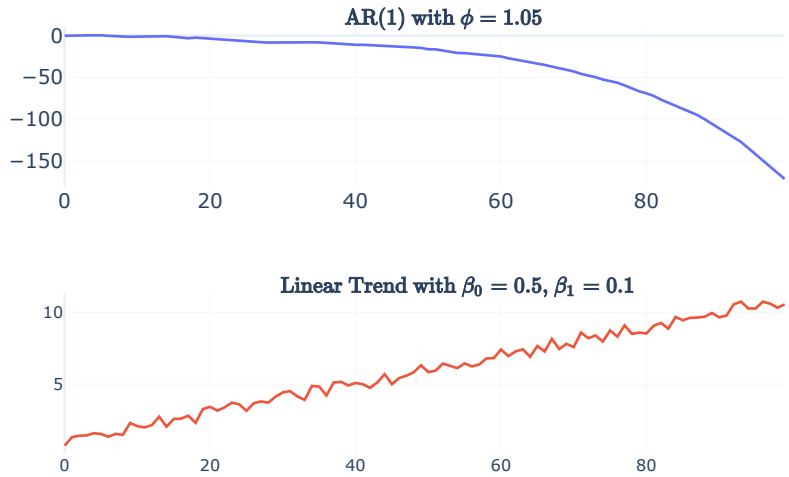
Out[17]: ADF_Test(stationary=True, results=(-5.871214438548429, 3.2369218692824975e-07, 3, 95, {'1%': -3.5011373281819504, '5%': -2.8924800524857854, '10%': -2.5832749307479226}, 48.71998024926833))

4. DECTECT AND HANDLE TRENDS

In [18]:
```
ar_series = generate_autoregressive_series(length, phi=1.05)

beta_0 = 0.5
beta_1 = 0.1
x = np.arange(length)
y_trend = beta_0 + beta_1*x +np.random.rand(length)
```

In [19]:
```
fig = make_subplots(
    rows=2, cols=1, subplot_titles=("$ \\text{AR(1) with }\phi=1.05$", "$ \\text{Linear Trend with }\\beta_0 = 0.5\\text{, }\\beta_1=(
)
```

```
fig.append_trace(
    go.Scatter(
        x=np.arange(length),
        y=ar_series,
    ),
    row=1,
    col=1,
)

fig.append_trace(
    go.Scatter(
        x=np.arange(length),
        y=y_trend,
    ),
    row=2,
    col=1,
)

fig.update_layout(
    height=500,
    width=700,
    showlegend=False,
    yaxis=dict(
        titlefont=dict(size=15),
        tickfont=dict(size=15),
    ),
    xaxis=dict(
        titlefont=dict(size=15),
        tickfont=dict(size=15),
    ),
),
fig.show()
```



In [20]:
```
res = check_deterministic_trend(ar_series)
print(f"Stationary: {res.adf_res.stationary} | Deterministic Trend: {res.deterministic_trend}")
```

```
Stationary: False | Deterministic Trend: False
```

In [21]:
```
res = check_deterministic_trend(y_trend)
print(f"Stationary: {res.adf_res.stationary} | Deterministic Trend: {res.deterministic_trend}")
```

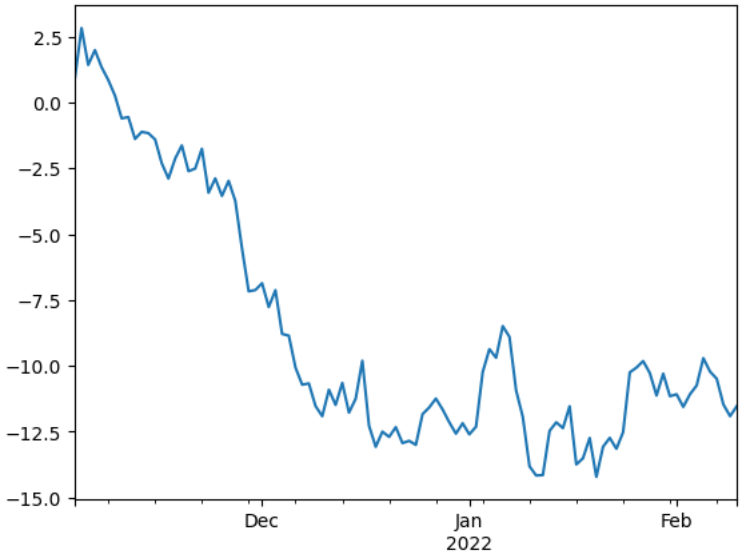
```
Stationary: False | Deterministic Trend: True
```

DETECTING TREND

In [22]:
```
pip install pymannkendall
```

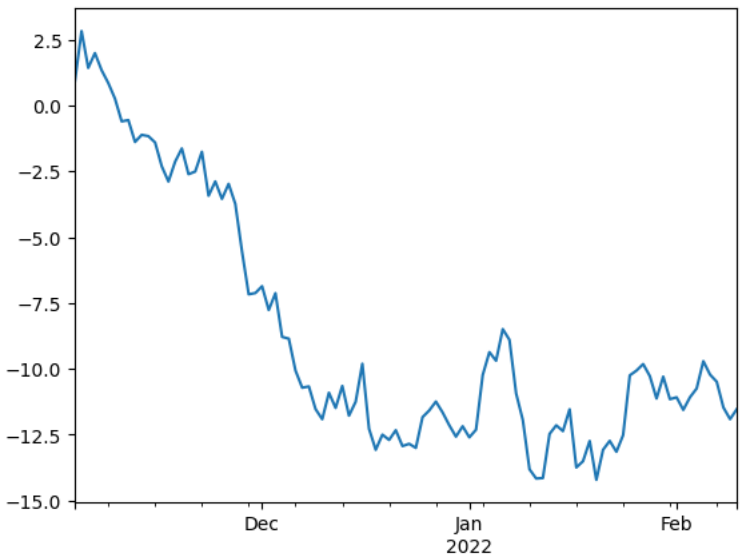
```
Requirement already satisfied: pymannkendall in c:\users\ridhi\onedrive\documents\python\envs\python39\lib\site-packages (1.4.3)
Requirement already satisfied: numpy in c:\users\ridhi\onedrive\documents\python\envs\python39\lib\site-packages (from pymannkendall)
(1.26.3)
Requirement already satisfied: scipy in c:\users\ridhi\onedrive\documents\python\envs\python39\lib\site-packages (from pymannkendall)
(1.12.0)
Note: you may need to restart the kernel to use updated packages.
```

In [41]:
```
y_unit_root.plot()
plt.show()
kendall_tau_res = check_trend(y_unit_root, confidence=0.05)
mann_kendall_res = check_trend(y_unit_root, confidence=0.05, mann_kendall=True)
print(f"Kendall's Tau: Trend: {kendall_tau_res.trend} | Direction: {kendall_tau_res.direction} | Deterministic: {kendall_tau_res.dete
print(f"Mann-Kendall's Tau: Trend: {mann_kendall_res.trend} | Direction: {mann_kendall_res.direction} | Deterministic: {mann_kendall_
```




```
Kendall's Tau: Trend: True | Direction: decreasing | Deterministic: False
Mann-Kendall's Tau: Trend: True | Direction: decreasing | Deterministic: False
```

In [42]:
```
y_unit_root.plot()
plt.show()
kendall_tau_res = check_trend(y_unit_root, confidence=0.05)
mann_kendall_res = check_trend(y_unit_root, confidence=0.05, mann_kendall=True)
print(f"Kendall's Tau: Trend: {kendall_tau_res.trend} | Direction: {kendall_tau_res.direction} | Deterministic: {kendall_tau_res.dete
print(f"Mann-Kendall's Tau: Trend: {mann_kendall_res.trend} | Direction: {mann_kendall_res.direction} | Deterministic: {mann_kendall_
```

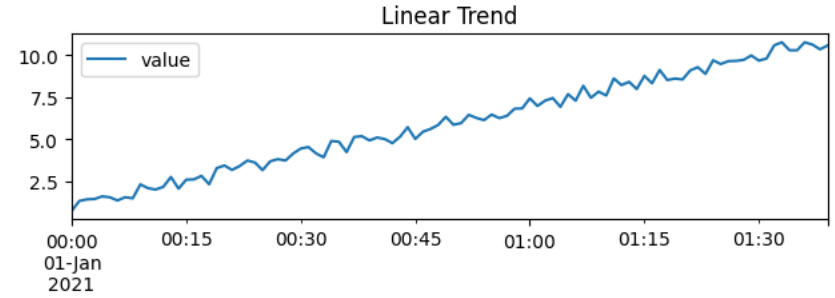


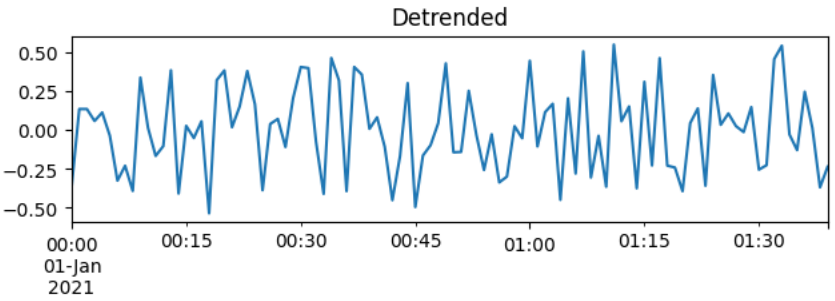

```
Kendall's Tau: Trend: True | Direction: decreasing | Deterministic: False
Mann-Kendall's Tau: Trend: True | Direction: decreasing | Deterministic: False
```

DeTrending

In [28]:
```
# Create DateTimeIndex, 100 minutes long to add to y_trend
m_range = pd.date_range("2021-01-01", periods=100, freq="T")

# Convert array to TimeDate indexed time series
y_trend_df = pd.DataFrame(y_trend, columns=["value"], index=m_range)


detrending_transformer = DetrendingTransformer()
y_diff = detrending_transformer.fit_transform(y_trend_df)
fig, axs = plt.subplots(2)
y_trend_df.plot(title="Linear Trend",ax=axs[0])
y_diff.plot(title="Detrended",ax=axs[1])
plt.tight_layout()
```

```
plt.show()
kendall_tau_res = check_trend(y_diff, confidence=0.05)
mann_kendall_tau_res = check_trend(y_diff, confidence=0.05, mann_kendall=True)
print(f"Kendalls Tau: Trend: {kendall_tau_res.trend}")
print(f"Mann-Kendalls Tau: Trend: {mann_kendall_tau_res.trend}")
```



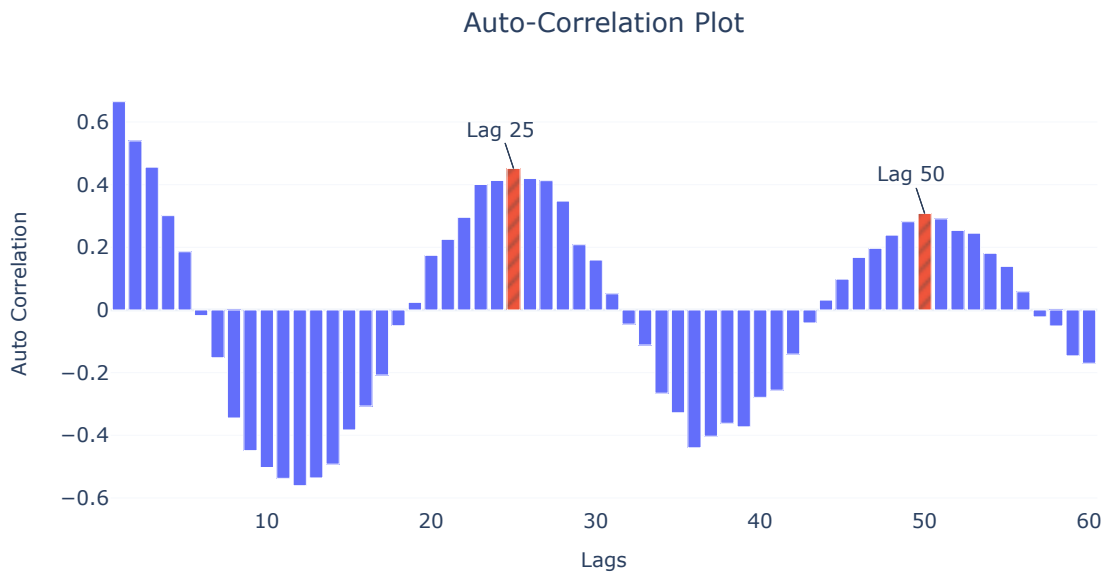



```
Kendalls Tau: Trend: False
Mann-Kendalls Tau: Trend: False
```

5. DETECT AND HANDLE SEASONALITYDetecting SeasonalityPlotting the ACF

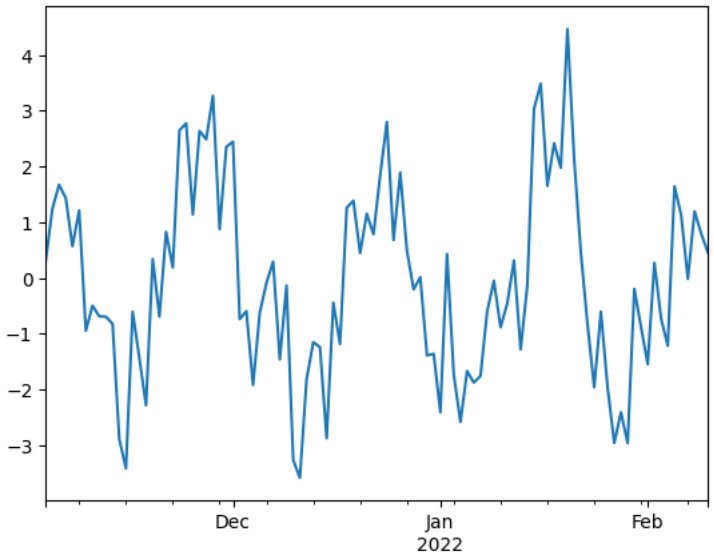
In [29]:
```
r = acf(y_seasonal, nlags=60, fft=False)
r = r[1:]
plot_df = pd.DataFrame(dict(x=np.arange(len(r))+1, y=r))
plot_df['seasonal_lag'] = False
plot_df.loc[plot_df["x"].isin([25,50]), "seasonal_lag"] = True
```

In [30]:
```
fig = px.bar(plot_df, x="x", y="y", pattern_shape="seasonal_lag", color="seasonal_lag", title="Auto-Correlation Plot")
fig.add_annotation(x=25, y=r[24], text="Lag 25")
fig.add_annotation(x=50, y=r[49], text="Lag 50")
fig.update_layout(
    showlegend=False,
    autosize=False,
    width=900,
    height=500,
    title={
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'},
    titlefont={
        "size":20
    },
    yaxis=dict(
        title_text="Auto Correlation",
        titlefont=dict(size=15),
        tickfont=dict(size=15),
    ),
    xaxis=dict(
        title_text="Lags",
        titlefont=dict(size=15),
        tickfont=dict(size=15),
    )
)
fig.update_annotations(font_size=15)
fig.show()
```



Detecting Statically

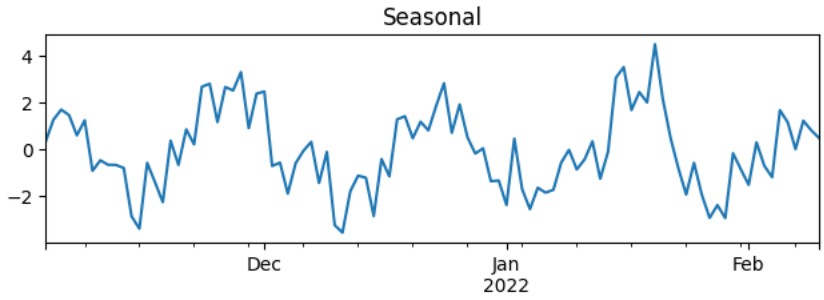
In [31]:
```
y_seasonal.plot()
plt.show()
seasonality_res = check_seasonality(y_seasonal, max_lag=30, seasonal_period=25, confidence=0.05)
print(f"Seasonality Test for 25th lag: {seasonality_res.seasonal}")
seasonality_id_res = check_seasonality(y_seasonal, max_lag=60, confidence=0.05)
print(f"Seasonality identified for: {seasonality_res.seasonal_periods}")
```

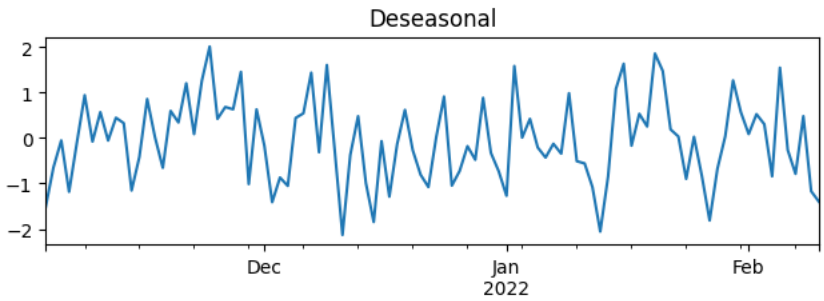



```
Seasonality Test for 25th lag: True
Seasonality identified for: 25
```

De-seasonalizing

In [44]:
```
deseasonalizing_transformer = DeseasonalizingTransformer(seasonality_extraction="period_averages", seasonal_period=25)
y_deseasonalized = deseasonalizing_transformer.fit_transform(y_seasonal, freq="1D")

fid, axs = plt.subplots(2)
y_seasonal.plot(title="Seasonal", ax=axs[0])
y_deseasonalized.plot(title="Deseasonal", ax=axs[1])
plt.tight_layout()
plt.show()

seasonality_res = check_seasonality(y_deseasonalized, seasonal_period=25, max_lag=26, confidence=0.05)
# mann_kendall_res = check_trend(y_diff, confidence=0.05, mann_kendall=True)
print(f"Seasonality at: {seasonality_res.seasonal_periods}: {seasonality_res.seasonal}")
```

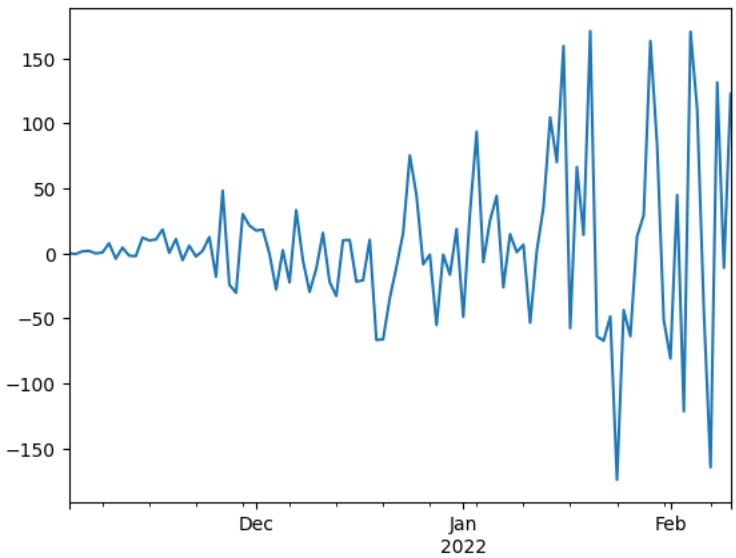





```
Seasonality at: 25: False
```

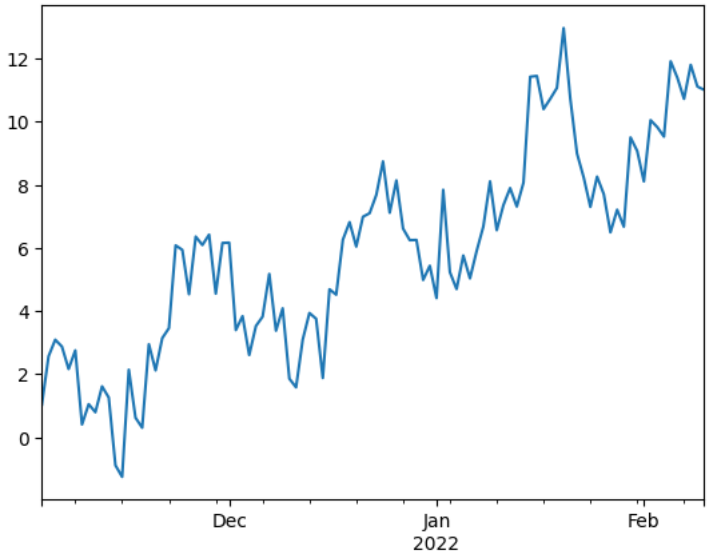
*6. DETECTING AND HANDLING HETEROSCEDASTICITY*Detecting Heteroscedasticity

In [47]:
```
y_hetero.plot()
plt.show()
hetero_res = check_heteroscedastisticity(y_hetero, confidence=0.05)
print(f"White Test for Heteroscedasticity: {hetero_res.heteroscedastic} with a p-value of {hetero_res.lm_p_value}")
```



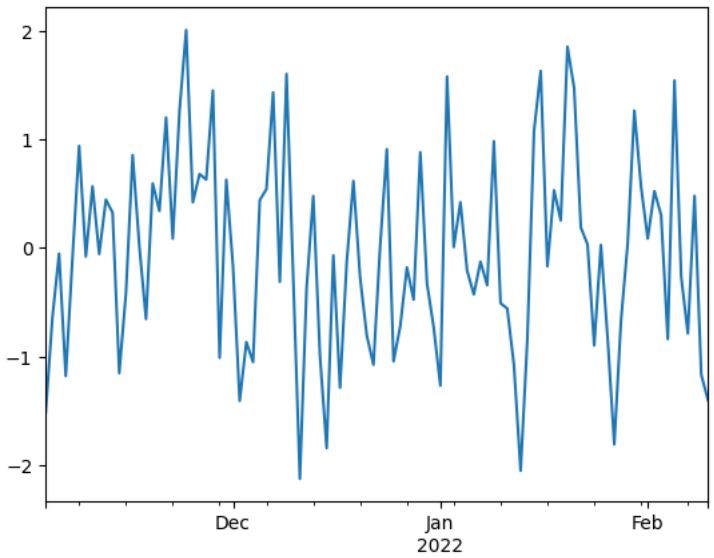

```
White Test for Heteroscedasticity: True with a p-value of 2.8244911172875107e-08
```

In [48]:
```
y_new = y_trend + y_seasonal
y_new.plot()
plt.show()
hetero_res = check_heteroscedastisticity(y_hetero, confidence=0.05)
print(f"White Test for Heteroscedasticity: {hetero_res.heteroscedastic} with a p-value of {hetero_res.lm_p_value}")
```




```
White Test for Heteroscedasticity: True with a p-value of 2.8244911172875107e-08
```

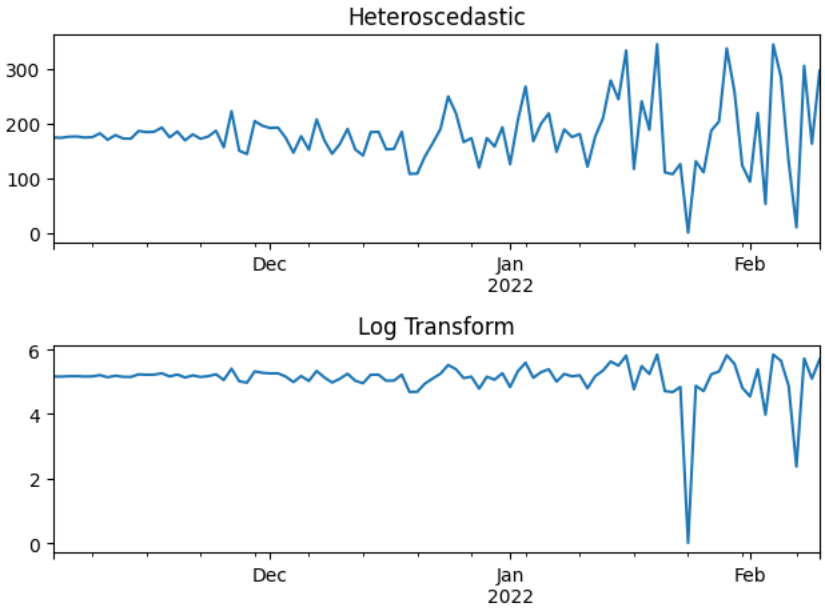
In [49]:
```
y_new = y_trend + y_seasonal
deseasonalizing_transformer = DeseasonalizingTransformer(seasonality_extraction="period_averages", seasonal_period=25)
y_deseasonalized = deseasonalizing_transformer.fit_transform(y_seasonal, freq="1D")
y_deseasonalized.plot()
plt.show()
hetero_res = check_heteroscedastisticity(y_hetero, confidence=0.05)
print(f"White Test for Heteroscedasticity: {hetero_res.heteroscedastic} with a p-value of {hetero_res.lm_p_value}")
```




```
White Test for Heteroscedasticity: True with a p-value of 2.8244911172875107e-08
```

Log Transforms

In [50]:
```
#shifting the series into positive domain
_y_hetero = y_hetero-y_hetero.min()
log_transformer = LogTransformer(add_one=True)
y_log = log_transformer.fit_transform(_y_hetero)
fig, axs = plt.subplots(2)
_y_hetero.plot(title="Heteroscedastic",ax=axs[0])
y_log.plot(title="Log Transform",ax=axs[1])
plt.tight_layout()
plt.show()
hetero_res = check_heteroscedastisticity(y_log, confidence=0.05)
# mann_kendall_res = check_trend(y_diff, confidence=0.05, mann_kendall=True)
print(f"White Test for Heteroscedasticity: {hetero_res.heteroscedastic} with a p-value of {hetero_res.lm_p_value}")
```

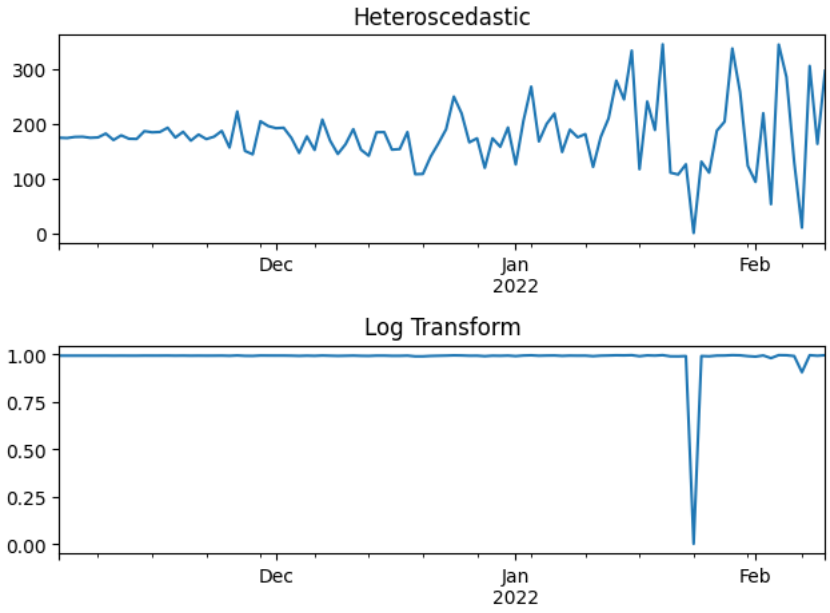



```
White Test for Heteroscedasticity: False with a p-value of 0.10395122689084639
```

Box-Cox TransformsOptimizing for Alpha with Guerrero's method

In [51]:
```
#shifting the series into positive domain
_y_hetero = y_hetero-y_hetero.min()
#Arbitarily divided the data into sub-series of length 25
boxcox_transformer = BoxCoxTransformer(seasonal_period=25, add_one=True, optimization="guerrero")
y_boxcox = boxcox_transformer.fit_transform(_y_hetero)
print(f"Optimal Lambda: {boxcox_transformer.boxcox_lambda}")
fig, axs = plt.subplots(2)
_y_hetero.plot(title="Heteroscedastic",ax=axs[0])
y_boxcox.plot(title="Log Transform",ax=axs[1])
plt.tight_layout()
plt.show()
hetero_res = check_heteroscedastisticity(y_boxcox, confidence=0.05)
# mann_kendall_res = check_trend(y_diff, confidence=0.05, mann_kendall=True)
print(f"White Test for Heteroscedasticity: {hetero_res.heteroscedastic} with a p-value of {hetero_res.lm_p_value}")
```

```
Optimal Lambda: -0.9999965176267451
```



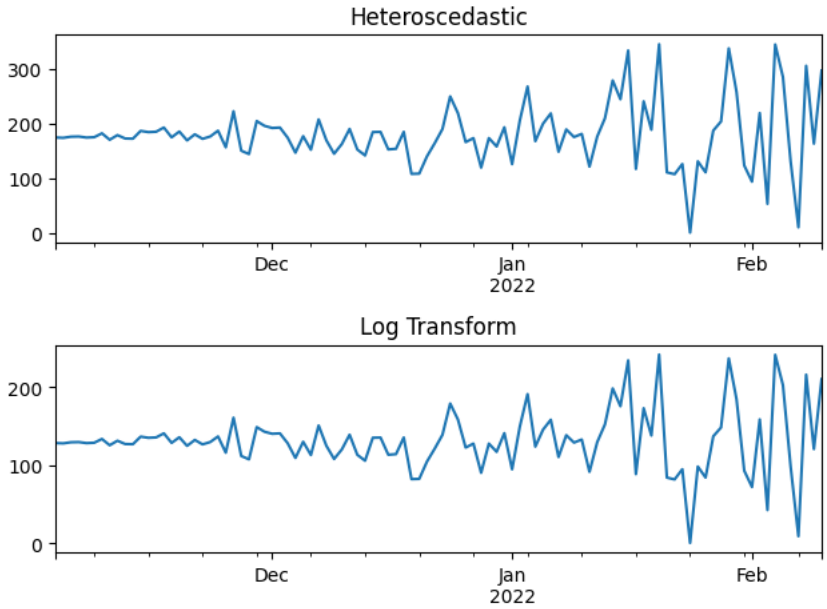

```
White Test for Heteroscedasticity: False with a p-value of 0.4821845477666262
```

Optimizing for Alpha with Loglikelihood

In [52]:
```
#shifting the series into positive domain
_y_hetero = y_hetero-y_hetero.min()
boxcox_transformer = BoxCoxTransformer(add_one=True, optimization="loglikelihood")
y_boxcox = boxcox_transformer.fit_transform(_y_hetero)
print(f"Optimal Lambda: {boxcox_transformer.boxcox_lambda}")
fig, axs = plt.subplots(2)
_y_hetero.plot(title="Heteroscedastic",ax=axs[0])
```

```
y_boxcox.plot(title="Log Transform",ax=axs[1])
plt.tight_layout()
plt.show()
hetero_res = check_heteroscedastisticity(y_boxcox, confidence=0.05)
# mann_kendall_res = check_trend(y_diff, confidence=0.05, mann_kendall=True)
print(f"White Test for Heteroscedasticity: {hetero_res.heteroscedastic} with a p-value of {hetero_res.lm_p_value}")
```

```
Optimal Lambda: 0.9262204095985213
```

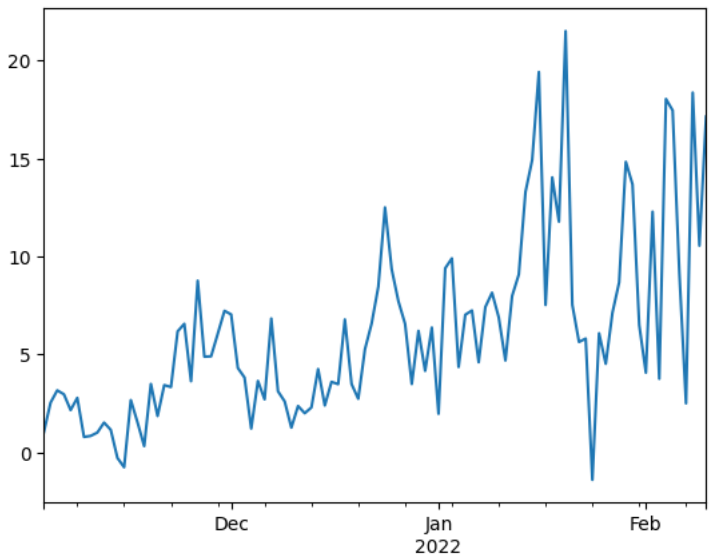



```
White Test for Heteroscedasticity: True with a p-value of 5.2392090667507145e-08
```

6./7. IMPLEMENT AUTOSTATIONARY TRANSFORMER

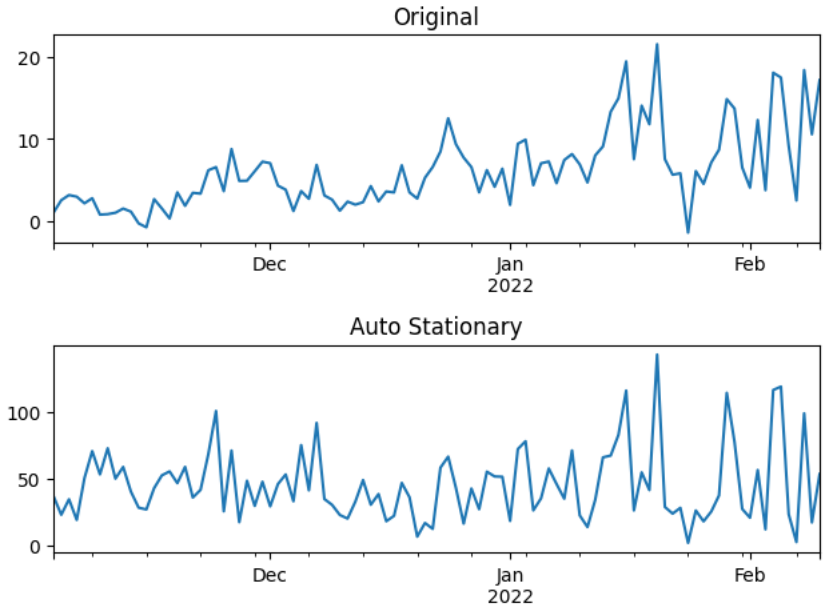
In [39]:
```
y_final = y_trend+y_seasonal+0.05*y_hetero
y_final.plot()
```

Out[39]: <Axes: >



In [40]:
```
auto_stationary = AutoStationaryTransformer(seasonal_period=25)
y_stat = auto_stationary.fit_transform(y_final)
print(f"Transformations applied: {[p.__class__.__name__ for p in auto_stationary._pipeline]}")
fig, axs = plt.subplots(2)
y_final.plot(title="Original",ax=axs[0])
y_stat.plot(title="Auto Stationary",ax=axs[1])
plt.tight_layout()
plt.show()
unit_root = check_unit_root(y_stat,confidence=0.05)
print(f"Unit Root: {unit_root.stationary} with a p-value of {unit_root.results[1]}")
y_inv = auto_stationary.inverse_transform(y_stat)
print(f"Inverse == Original @ precision of 2 decimal points: {np.all(y_inv.round(2)==y_final.round(2))}")
```

```
Transformations applied: ['DetrendingTransformer', 'DeseasonalizingTransformer', 'AddMTransformer', 'BoxCoxTransformer']
```




```
Unit Root: True with a p-value of 0.0009940530355200788
Inverse == Original @ precision of 2 decimal points: True
```

In [ ]: