DCC-analyse med EGARCH-modell

Denne anaylsen undersøker den dynamiske samvariasjonen mellom to finansielle tidsserier ved bruk av EGARCH og DCC. Modellene estimeres basert på daglige strømpriser fra Norge og Tyskland i peiroden 2019 til 2024.

Analysen retter fokus mot mulige strukturelle endringer rundt Russlands invasjon av Ukraina 24. februar 2022, som er markert i figurene.

Kravspesifikasjon

For å sikre reproduserbarhet, anbefales det å benytte samme versjoner av alle biblioteker vedlagt i egen requirements.txt fil.

In [4]:
```python
# --- Importer nødvendige biblioteker ---
import sys
import os
import pandas as pd
import numpy as np
import scipy
import statsmodels
import arch
import matplotlib
import plotly
import openpyxl

# Valgfritt: Importer notebook-spesifikke pakker dersom tilgjengelig
try:
    import notebook
except ImportError:
    notebook = None

try:
    import ipykernel
except ImportError:
    ipykernel = None

# --- Samle versjonsinformasjon ---
def get_versions():
    """Hent versjonsnummer for Python og relevante pakker."""
    versions = {
        "python": f"{sys.version_info.major}.{sys.version_info.minor}.{sys.version_info.micro}",
        "pandas": pd.__version__,
        "numpy": np.__version__,
        "scipy": scipy.__version__,
        "statsmodels": statsmodels.__version__,
        "arch": arch.__version__,
        "matplotlib": matplotlib.__version__,
        "plotly": plotly.__version__,
        "openpyxl": openpyxl.__version__,
    }
    if notebook:
        versions["notebook"] = notebook.__version__
    if ipykernel:
        versions["ipykernel"] = ipykernel.__version__
    return versions

# --- Lagre kravspesifikasjon til requirements.txt ---
def save_requirements(filename="requirements.txt"):
    """Lagre alle pakkeversjoner til en requirements.txt-fil."""
    versions = get_versions()
    with open(filename, "w") as f:
        for package, version in versions.items():
            f.write(f"{package}=={version}\n")

# --- Kjør lagring ---
save_requirements()
```

Importer biblioteker og definer konstanter

Denne seksjonen laster inn nødvendige Python-biblioteker for dataanalyse, statistisk modellering, visualisering og eksport til Excel.
I tillegg defineres konstanter som brukes gjennom hele analysen:

- Filbaner for input og output
- Tidsperiode for analyse
- Navnemapping for figurer

- Datoe for invasjonen
- Fargepalett for visualiseringer

Mapper for lagring av resultater opprettes automatisk dersom de ikke eksisterer.

In [6]:
```python
# --- Importer nødvendige biblioteker ---

# Standardbibliotek
import os
import re
import inspect
from pathlib import Path
from datetime import datetime
from itertools import product

# Tredjepartsbibliotek: Data og analyse
import numpy as np
import pandas as pd
from scipy.optimize import minimize
from scipy.stats import skew, kurtosis, gaussian_kde, probplot

# Tredjepartsbibliotek: Modellering og statistikk
from arch import arch_model
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.stats.diagnostic import het_arch, acorr_ljungbox
from statsmodels.tsa.stattools import adfuller, acf, pacf

# Tredjepartsbibliotek: Visualisering
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from IPython.display import display

# Tredjepartsbibliotek: Excel-eksport
from pandas import ExcelWriter
from openpyxl import Workbook, load_workbook
from openpyxl.utils import get_column_letter
from openpyxl.utils.dataframe import dataframe_to_rows
from openpyxl.styles import Font, Border, Side

# --- Konstanter: Filbaner ---
INPUT_DIR = Path("input/daily_aggregate")
OUTPUT_DIR = Path("output")
EXCEL_DIR = OUTPUT_DIR / "excel"

# Sørg for at output-mapper eksisterer
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)
EXCEL_DIR.mkdir(parents=True, exist_ok=True)

# --- Konstanter: Tidsperiode og parametere ---
YEARS = range(2019, 2025)
ROLLING_WINDOW = 30
FIGURE_EXPORT_SCALE = 25

# --- Konstanter: Navnemapping for figurer ---
NAME_MAP = {
    "GER": "Tyskland",
    "NO2": "Norge"
}

# --- Viktige datoer (for analyse og figurer) ---
BREAK_DATE = "2022-02-24"
BREAK_DATE_LABEL = "24. feb 2022"
BREAK_DATE_DT = pd.to_datetime(BREAK_DATE)

# --- Fargepalett for visualisering ---
COLOR_1 = "#1f77b4"  # blå
COLOR_2 = "#ff7f0e"  # oransje
COLOR_3 = "#2ca02c"  # grønn
COLORS = [COLOR_1, COLOR_2, COLOR_3]
```

In [7]:
```python
# --- Felles layoutinnstillinger for Plotly-figurer ---
common_layout: dict = {
    "template": "plotly_white",
    "font": {
        "family": "Times New Roman",
        "size": 16,
        "color": "black",
    },
    "margin": {
        "l": 80,
        "r": 40,
```

```
        "t": 80,
        "b": 70,
    },
    "legend": {
        "title": {"text": ""},
        "orientation": "h",
        "yanchor": "bottom",
        "y": 1.02,
        "xanchor": "right",
        "x": 1,
    },
    "hovermode": "x unified",
    "xaxis": {
        "showgrid": True,
        "title_font": {
            "size": 16
        },
        "tickfont": {
            "size": 14
        },
    },
    "yaxis": {
        "showgrid": True,
        "title_font": {
            "size": 16
        },
        "tickfont": {
            "size": 14
        },
    },
}
```

Støttefunksjoner

Her defineres nødvendige støttefunksjoner for analyse og presentasjon.

In [9]:
```
def add_break_line(
    fig: go.Figure,
    x: float,
    label: str,
    color: str = "red",
    dash: str = "dash",
    line_width: int = 2
) -> None:
    """
    Legger til en vertikal linje i en Plotly-figur for å indikere et bruddpunkt.

    Parametre:
    ----------
    fig : go.Figure
        Plotly-figur som linjen skal legges til i.
    x : float
        X-posisjon for linjen (f.eks. dato eller tallverdi).
    label : str
        Navn som vises i figurens legend.
    color : str, optional
        Farge på linjen, som Plotly-fargenavn eller hex-kode. Standard er "red".
    dash : str, optional
        Linjetype ("solid", "dash", "dot"). Standard er "dash".
    line_width : int, optional
        Tykkelse på linjen. Standard er 2.

    Returnerer:
    -----------
    None
    """

    # --- Tegn vertikal linje ---
    fig.add_shape(
        type="line",
        x0=x, x1=x,
        y0=0, y1=1,
        xref="x", yref="paper",
        line=dict(color=color, dash=dash, width=line_width),
        layer="above"
    )

    # --- Legg til dummy-trace for legend ---
    fig.add_trace(
        go.Scatter(
            x=[None],
```

```
            y=[None],
            mode="lines",
            line=dict(color=color, dash=dash, width=line_width),
            name=label,
            hoverinfo="skip",
            showlegend=True
        )
    )
```

In [10]:

```
def plot_timeseries(
    data: pd.DataFrame,
    title: str,
    y_title: str,
    filename: str,
    show: bool = True,
    show_break: bool = False
) -> None:
    """
    Lager en linjegraf for én eller flere tidsserier.

    Parametre:
    ----------
    data : pd.DataFrame
        DataFrame med datetime-indeks og én eller flere kolonner.
    title : str
        Tittel på figuren.
    y_title : str
        Y-akse tittel.
    filename : str
        Navn på filen figuren skal lagres som.
    show : bool, optional
        Om figuren skal vises etter lagring. Standard er True.
    show_break : bool, optional
        Om en vertikal bruddlinje skal legges til på BREAK_DATE_DT. Standard er False.

    Returnerer:
    -----------
    None
    """

    # --- Initialiser figur ---
    fig = go.Figure()

    # --- Sjekk om data er tom ---
    if data.empty:
        print("Advarsel: Data er tom – ingen figur genereres.")
        return

    # --- Legg til dataserier ---
    if data.shape[1] == 1:
        col = data.columns[0]
        name = NAME_MAP.get(col, col)

        fig.add_trace(
            go.Scatter(
                x=data.index,
                y=data[col],
                name=name,
                line=dict(color=COLORS[0]),
            )
        )
    else:
        for i, col in enumerate(data.columns):
            name = NAME_MAP.get(col, col)
            fig.add_trace(
                go.Scatter(
                    x=data.index,
                    y=data[col],
                    name=name,
                    line=dict(color=COLORS[i % len(COLORS)]),
                )
            )

    # --- Legg til bruddlinje hvis ønskelig ---
    if show_break:
        add_break_line(fig, BREAK_DATE_DT, BREAK_DATE_LABEL)

    # --- Oppdater layout ---
    fig.update_layout(
        title=title,
        yaxis_title=y_title,
        **common_layout
    )
```

```
    # --- Lagre figur ---
    save_figure(fig, filename)

    # --- Vis figur ---
    if show:
        fig.show()
```

In [11]:

```
def plot_scatter(
    data: pd.DataFrame,
    title: str,
    y_title: str,
    filename: str,
    show: bool = True,
    show_break: bool = False
) -> None:
    """
    Lager et scatter-plot for én eller flere tidsserier.

    Parametre:
    ----------
    data : pd.DataFrame
        DataFrame med datetime-indeks og én eller flere kolonner.
    title : str
        Tittel på figuren.
    y_title : str
        Y-akse tittel.
    filename : str
        Navn på filen figuren skal lagres som.
    show : bool, optional
        Om figuren skal vises etter lagring. Standard er True.
    show_break : bool, optional
        Om dataserien skal deles opp før/etter BREAK_DATE_DT. Standard er False.

    Returnerer:
    -----------
    None
    """

    # --- Initialiser figur ---
    fig = go.Figure()

    # --- Sjekk om data er tom ---
    if data.empty:
        print("Advarsel: Data er tom – ingen figur genereres.")
        return

    # --- Legg til dataserier ---
    if show_break and data.shape[1] == 1:
        col = data.columns[0]
        name = NAME_MAP.get(col, col)

        pre_break = data.loc[data.index < BREAK_DATE_DT, col]
        post_break = data.loc[data.index >= BREAK_DATE_DT, col]

        fig.add_trace(
            go.Scatter(
                x=pre_break.index,
                y=pre_break.values,
                name=f"{name} (før)",
                mode="markers",
                marker=dict(color=COLORS[0]),
            )
        )
        fig.add_trace(
            go.Scatter(
                x=post_break.index,
                y=post_break.values,
                name=f"{name} (etter)",
                mode="markers",
                marker=dict(color=COLORS[1]),
            )
        )
    else:
        for i, col in enumerate(data.columns):
            name = NAME_MAP.get(col, col)
            fig.add_trace(
                go.Scatter(
                    x=data.index,
                    y=data[col],
                    name=name,
                    mode="markers",
                    marker=dict(color=COLORS[i % len(COLORS)]),
```

```
                )
            )

    # --- Legg til bruddlinje hvis ønskelig ---
    if show_break:
        add_break_line(fig, BREAK_DATE_DT, BREAK_DATE_LABEL)

    # --- Oppdater layout ---
    fig.update_layout(
        title=title,
        yaxis_title=y_title,
        **common_layout
    )

    # --- Lagre figur ---
    save_figure(fig, filename)

    # --- Vis figur ---
    if show:
        fig.show()
```

In [12]:

```
def plot_histogram(
    data: pd.DataFrame,
    title: str,
    x_title: str,
    filename: str,
    show: bool = True
) -> None:
    """
    Lager histogram for én eller flere variabler.

    Parametre:
    ----------
    data : pd.DataFrame
        DataFrame med én eller flere kolonner.
    title : str
        Tittel på figuren.
    x_title : str
        X-akse tittel.
    filename : str
        Navn på filen figuren skal lagres som.
    show : bool, optional
        Om figuren skal vises etter lagring. Standard er True.

    Returnerer:
    -----------
    None
    """

    # --- Initialiser figur ---
    fig = go.Figure()

    # --- Sjekk om data er tom ---
    if data.empty:
        print("Advarsel: Data er tom – ingen histogram genereres.")
        return

    # --- Legg til histogramspor ---
    for i, col in enumerate(data.columns):
        name = NAME_MAP.get(col, col)
        fig.add_trace(
            go.Histogram(
                x=data[col],
                name=name,
                marker=dict(color=COLORS[i % len(COLORS)]),
                opacity=0.75
            )
        )

    # --- Oppdater layout ---
    fig.update_layout(
        title=title,
        xaxis_title=x_title,
        barmode="overlay",
        **common_layout
    )

    # --- Lagre figur ---
    save_figure(fig, filename)

    # --- Vis figur ---
    if show:
        fig.show()
```

In [13]:
```python
def plot_rolling_average(
    data: pd.DataFrame,
    title: str,
    y_title: str,
    filename: str,
    window: int = ROLLING_WINDOW,
    show: bool = True,
    show_break: bool = True
) -> None:
    """
    Plotter glidende gjennomsnitt av én eller flere tidsserier.

    Parametre:
    ----------
    data : pd.DataFrame
        DataFrame med datetime-indeks og én eller flere kolonner.
    title : str
        Tittel på figuren.
    y_title : str
        Y-akse tittel.
    filename : str
        Navn på filen figuren skal lagres som.
    window : int, optional
        Lengde på det glidende vinduet. Standard er ROLLING_WINDOW.
    show : bool, optional
        Om figuren skal vises etter lagring. Standard er True.
    show_break : bool, optional
        Om en bruddlinje skal legges til. Standard er True.

    Returnerer:
    -----------
    None
    """

    # --- Sjekk om data er tom ---
    if data.empty:
        print("Advarsel: Data er tom – glidende gjennomsnitt ikke generert.")
        return

    # --- Beregn glidende gjennomsnitt ---
    rolling_data = data.rolling(window=window, min_periods=1).mean()

    # --- Plot glidende gjennomsnitt ---
    plot_timeseries(
        rolling_data,
        title=title,
        y_title=y_title,
        filename=filename,
        show=show,
        show_break=show_break
    )
```

In [14]:
```python
def plot_histogram_comparison(
    series1: pd.Series,
    series2: pd.Series,
    label1: str,
    label2: str,
    title: str,
    xlabel: str,
    filename: str,
    bins: int = 40,
    show: bool = True
) -> None:
    """
    Plotter to distribusjoner med histogram, KDE og gjennomsnittslinjer.
    Legenden viser gjennomsnitt og plasseres under plottet.

    Parametre:
    ----------
    series1, series2 : pd.Series
        Pandas Series med verdier.
    label1, label2 : str
        Navn for dataseriene.
    title : str
        Tittel på figuren.
    xlabel : str
        Navn på x-aksen.
    filename : str
        Navn på filen figuren skal lagres som.
    bins : int, optional
        Antall søyler i histogrammet. Standard er 40.
    show : bool, optional
```

```
    Om figuren skal vises etter lagring. Standard er True.

Returnerer:
-----------
None
"""

# --- Bruk mapping på etiketter ---
label1_mapped = NAME_MAP.get(label1, label1)
label2_mapped = NAME_MAP.get(label2, label2)

# --- Beregn statistikk ---
mean1, mean2 = series1.mean(), series2.mean()
kde1, kde2 = gaussian_kde(series1), gaussian_kde(series2)

# --- Definer x-akse for KDE ---
x_range = np.linspace(
    min(series1.min(), series2.min()),
    max(series1.max(), series2.max()),
    500
)

# --- Definer visningsområde (±3 std) ---
combined = np.concatenate([series1, series2])
mean_comb = combined.mean()
std_comb = combined.std()
x_min, x_max = mean_comb - 3 * std_comb, mean_comb + 3 * std_comb

# --- Initialiser figur ---
fig = go.Figure()

# --- Legg til histogrammer ---
for series, label, color in [
    (series1, label1_mapped, COLOR_1),
    (series2, label2_mapped, COLOR_2)
]:
    fig.add_trace(
        go.Histogram(
            x=series,
            name=label,
            marker_color=color,
            opacity=0.6,
            nbinsx=bins,
            histnorm="probability density"
        )
    )

# --- Legg til KDE-linjer ---
for kde, mean, label, color in [
    (kde1, mean1, label1_mapped, COLOR_1),
    (kde2, mean2, label2_mapped, COLOR_2)
]:
    fig.add_trace(
        go.Scatter(
            x=x_range,
            y=kde(x_range),
            mode="lines",
            name=f"KDE {label} (μ = {mean:.2f})",
            line=dict(color=color)
        )
    )

# --- Legg til vertikale gjennomsnittslinjer ---
for mean, color in [(mean1, COLOR_1), (mean2, COLOR_2)]:
    fig.add_vline(
        x=mean,
        line=dict(color=color, dash="dash")
    )

# --- Sett opp layout ---
layout = common_layout.copy()
layout.update({
    "title": title,
    "xaxis_title": xlabel,
    "yaxis_title": "Tetthet",
    "barmode": "overlay",
    "xaxis": {**common_layout["xaxis"], "range": [x_min, x_max]},
    "legend": dict(
        orientation="h",
        x=0.5,
        y=-0.3,
        xanchor="center",
        yanchor="top",
```

```
            font=dict(size=13)
        ),
        "margin": dict(l=80, r=40, t=80, b=120)
    })

    fig.update_layout(**layout)

    # --- Lagre figur ---
    save_figure(fig, filename)

    # --- Vis figur ---
    if show:
        fig.show()
```

In [15]:

```
def plot_qq(
    data: pd.Series,
    label: str,
    color: str,
    title: str,
    filename: str,
    show: bool = True
) -> None:
    """
    Lager QQ-plott mot normalfordeling.

    Parametre:
    ----------
    data : pd.Series
        Pandas Series med data som skal sammenlignes med normalfordeling.
    label : str
        Navn som vises i figuren.
    color : str
        Farge på datapunktene.
    title : str
        Tittel på figuren.
    filename : str
        Navn på filen figuren skal lagres som.
    show : bool, optional
        Om figuren skal vises etter lagring. Standard er True.

    Returnerer:
    -----------
    None
    """

    # --- Sjekk om data er tom ---
    if data.empty:
        print("Advarsel: Data er tom – QQ-plot ikke generert.")
        return

    # --- Beregn teoretiske og observerte kvantiler ---
    (osm, osr), (slope, intercept, _) = probplot(data, dist="norm")

    line_x = np.array([osm.min(), osm.max()])
    line_y = slope * line_x + intercept

    # --- Initialiser figur ---
    fig = go.Figure()

    # --- Legg til datapunkter ---
    fig.add_trace(
        go.Scatter(
            x=osm,
            y=osr,
            mode="markers",
            name=label,
            marker=dict(color=color)
        )
    )

    # --- Legg til referanselinje ---
    fig.add_trace(
        go.Scatter(
            x=line_x,
            y=line_y,
            mode="lines",
            name="Normal linje",
            line=dict(color="black", dash="dash")
        )
    )

    # --- Oppdater layout ---
    fig.update_layout(
```

```
        title=f"{title} {label}",
        xaxis_title="Teoretiske kvantiler",
        yaxis_title="Observerte verdier",
        showlegend=False,
        **common_layout
    )

    # --- Lagre figur ---
    save_figure(fig, filename)

    # --- Vis figur ---
    if show:
        fig.show()
```

In [16]:

```
def plot_acf_pacf(
    series: pd.Series,
    title_prefix: str = "",
    lags: int = 20,
    show: bool = True
) -> None:
    """
    Lager ACF- og PACF-plot for en gitt tidsserie, med 95 % konfidensintervall.

    Parametre:
    ----------
    series : pd.Series
        Tidsserie som skal analyseres.
    title_prefix : str, optional
        Prefiks som brukes i titler og filnavn. Standard er tom streng.
    lags : int, optional
        Antall lag som skal beregnes. Standard er 20.
    show : bool, optional
        Om figurene skal vises etter lagring. Standard er True.

    Returnerer:
    -----------
    None
    """

    # --- Beregn konfidensintervall ---
    n = len(series.dropna())
    conf_int = 1.96 / np.sqrt(n)

    # --- Beregn ACF og PACF ---
    acf_vals = acf(series, nlags=lags)
    pacf_vals = pacf(series, nlags=lags)
    x_vals = list(range(len(acf_vals)))

    # --- Definer annotasjon ---
    annotation_text = "Streken viser 95 % konfidensintervall for nullhypotesen (ingen autokorrelasjon)"
    annotation = dict(
        text=annotation_text,
        xref="paper",
        yref="paper",
        x=0,
        y=-0.20,
        showarrow=False,
        font=dict(size=12, color="gray"),
        align="left"
    )

    # --- Lag ACF-figur ---
    acf_fig = go.Figure()
    acf_fig.add_trace(
        go.Bar(x=x_vals, y=acf_vals, name="ACF")
    )
    acf_fig.add_shape(
        type="rect",
        x0=-0.5, x1=lags + 0.5,
        y0=-conf_int, y1=conf_int,
        fillcolor="lightblue",
        opacity=0.3,
        layer="below",
        line_width=0
    )
    acf_fig.add_hline(y=conf_int, line=dict(dash="dash", color="blue"), opacity=0.5)
    acf_fig.add_hline(y=-conf_int, line=dict(dash="dash", color="blue"), opacity=0.5)
    acf_fig.update_layout(
        title=f"{title_prefix} ACF",
        xaxis_title="Lag",
        yaxis_title="Autokorrelasjon",
        annotations=[annotation],
        **common_layout
```

```
    )

    # --- Lag PACF-figur ---
    pacf_fig = go.Figure()
    pacf_fig.add_trace(
        go.Bar(x=x_vals, y=pacf_vals, name="PACF")
    )
    pacf_fig.add_shape(
        type="rect",
        x0=-0.5, x1=lags + 0.5,
        y0=-conf_int, y1=conf_int,
        fillcolor="lightblue",
        opacity=0.3,
        layer="below",
        line_width=0
    )
    pacf_fig.add_hline(y=conf_int, line=dict(dash="dash", color="blue"), opacity=0.5)
    pacf_fig.add_hline(y=-conf_int, line=dict(dash="dash", color="blue"), opacity=0.5)
    pacf_fig.update_layout(
        title=f"{title_prefix} PACF",
        xaxis_title="Lag",
        yaxis_title="Partial Autokorrelasjon",
        annotations=[annotation],
        **common_layout
    )

    # --- Lagre figurer ---
    save_figure(acf_fig, f"{title_prefix}_acf")
    save_figure(pacf_fig, f"{title_prefix}_pacf")

    # --- Vis figurer ---
    if show:
        acf_fig.show()
        pacf_fig.show()
```

In [17]:

```
def save_figure(fig: go.Figure, name: str) -> None:
    """
    Lagrer en Plotly-figur som HTML, PDF og PNG i organiserte undermapper.

    Parametre:
    ----------
    fig : go.Figure
        Plotly-figur som skal lagres.
    name : str
        Filnavn uten filtype (brukes som base for alle formater).

    Returnerer:
    -----------
    None
    """

    # --- Definer undermapper og filstier ---
    formats = ["html", "pdf", "png"]
    paths = {
        fmt: os.path.join(OUTPUT_DIR, fmt, f"{name}.{fmt}")
        for fmt in formats
    }

    # --- Opprett mapper om nødvendig ---
    for fmt in formats:
        os.makedirs(os.path.join(OUTPUT_DIR, fmt), exist_ok=True)

    # --- Intern hjelpefunksjon for trygg lagring ---
    def _safe_write(write_func, path: str, label: str) -> None:
        try:
            write_func(path)
        except Exception as e:
            print(f"Kunne ikke lagre {label} for {name}: {e}")

    # --- Lagre som HTML ---
    _safe_write(fig.write_html, paths["html"], "HTML")

    # --- Lagre som PDF og PNG ---
    for fmt in ["pdf", "png"]:
        _safe_write(
            lambda p: fig.write_image(p, scale=FIGURE_EXPORT_SCALE),
            paths[fmt],
            fmt.upper()
        )

    # --- Registrer figur hvis funksjon finnes ---
    try:
        add_figure(name, fig)
```

```
    except NameError:
        pass
```

In [18]:
```
def save_to_excel(
    excel_path: Path = Path("output/excel/data_series.xlsx"),
    **kwargs
) -> None:
    """
    Lagrer ett eller flere objekter til en Excel-fil.

    - DataFrames lagres på egne arkfaner med variabelnavn som arknavn.
    - Andre objekter lagres i et sammendrag i et ark kalt "variables".

    Parametre:
    ----------
    excel_path : Path, optional
        Filsti til Excel-filen. Standard er "output/excel/data_series.xlsx".
    kwargs : key-value
        Navn og objekter som skal lagres.

    Returnerer:
    -----------
    None
    """

    from openpyxl import load_workbook

    # --- Sørg for at mappe eksisterer ---
    excel_path.parent.mkdir(parents=True, exist_ok=True)

    # --- Opprett fil om den ikke eksisterer ---
    if not excel_path.exists():
        with pd.ExcelWriter(excel_path, engine="openpyxl") as writer:
            pd.DataFrame([["Midlertidig ark, kan slettes."]]).to_excel(writer, sheet_name="temp")

    # --- Registrer eksisterende ark ---
    existing_sheets = set()
    if excel_path.exists():
        wb = load_workbook(excel_path)
        existing_sheets = set(wb.sheetnames)

    variables_info = {}

    # --- Lagre objektene ---
    with pd.ExcelWriter(excel_path, mode="a", engine="openpyxl", if_sheet_exists="replace") as writer:
        for var_name, obj in kwargs.items():
            sheet_name = var_name[:31]  # Excel-begrensning på arkfanenavn

            if isinstance(obj, pd.DataFrame):
                try:
                    obj.to_excel(writer, sheet_name=sheet_name, index=False)
                except Exception as e:
                    print(f"Kunne ikke lagre DataFrame '{var_name}': {e}")
            else:
                # Lag et sammendrag for ikke-DataFrame objekter
                summary = str(obj)[:100]
                variables_info[var_name] = {
                    "Variable": var_name,
                    "Type": type(obj).__name__,
                    "Value": summary
                }

        # --- Lagre sammendrag dersom andre objekter finnes ---
        if variables_info:
            var_df = pd.DataFrame(variables_info.values())
            try:
                var_df.to_excel(writer, sheet_name="variables", index=False)
            except Exception as e:
                print(f"Kunne ikke lagre 'variables'-arket: {e}")

    # --- Ferdig ---
    print(f"Alt er lagret i Excel: {excel_path}")
```

In [19]:
```
def save_garch_summaries_txt(
    garch_models: dict,
    txt_path: str = "output/garch_summaries.txt"
) -> None:
    """
    Lagrer sammendrag fra GARCH-modeller til en tekstfil.

    Parametre:
    ----------
    garch_models : dict
```

```
        Ordbok med nøkler (str) og verdier (fitted GARCH-modeller med .summary()-metode).
    txt_path : str, optional
        Filsti for lagring. Standard er "output/garch_summaries.txt".

    Returnerer:
    -----------
    None
    """

    try:
        # --- Sørg for at mappe eksisterer ---
        os.makedirs(os.path.dirname(txt_path), exist_ok=True)

        # --- Skriv sammendrag til tekstfil ---
        with open(txt_path, "w", encoding="utf-8") as f:
            for key, model in garch_models.items():
                f.write(f"{'=' * 40}\n")
                f.write(f"GARCH Model for: {key}\n")
                f.write(f"{'-' * 40}\n")
                f.write(model.summary().as_text())
                f.write("\n\n")

        print(f"GARCH-sammendrag lagret til: {txt_path}")

    except Exception as e:
        print(f"Kunne ikke lagre GARCH-sammendrag: {e}")
```

In [20]:

```
def export_garch_results_to_excel(
    results_dict: dict,
    filename: str = "data.xlsx"
) -> None:
    """
    Eksporterer GARCH-modellresultater til en Excel-fil med formatert tabell på norsk.

    Parametre:
    ----------
    results_dict : dict
        Dictionary med modellresultater, f.eks. {"GER": result1, "NO2": result2}.
    filename : str, optional
        Navn på Excel-filen. Standard er "data.xlsx".

    Returnerer:
    -----------
    None
    """

    # --- Definer filsti ---
    file_path = EXCEL_DIR / filename

    summary_rows = {}
    temp_writer_data = {}

    # --- Behandle hver modell ---
    for code, result in results_dict.items():
        country = NAME_MAP.get(code, code)
        sheet_name = f"EGARCH-modell for {country}"

        # Hent modellresultater
        params = result.params
        stderr = result.std_err
        tvals = result.tvalues
        pvals = result.pvalues
        conf_int = result.conf_int()

        # Formater rader for Excel
        formatted_rows = []
        for param in params.index:
            # Kategoriser parameter
            if param == "mu":
                section = "Gjennomsnittsmodell"
            elif param.startswith("nu"):
                section = "Distribusjon"
            else:
                section = "Volatilitetsmodell"

            coef = params[param]
            se = stderr[param]
            tval = tvals[param]
            pval = pvals[param]
            ci_low, ci_high = conf_int.loc[param]

            formatted_rows.append([
                section,
```

```
                param,
                f"{coef:.4f}",
                f"{se:.4f}",
                f"{tval:.4f}",
                f"{pval:.4f}",
                f"[{ci_low:.4f}, {ci_high:.4f}]"
            ])

        df_formatted = pd.DataFrame(
            formatted_rows,
            columns=[
                "Modellkomponent", "Parameter", "Estimat",
                "Standardfeil", "t-verdi", "p-verdi", "95 % konfidensintervall"
            ]
        )

        temp_writer_data[sheet_name[:31]] = df_formatted

        # Legg til informasjon for sammendrag
        model = result.model
        summary_rows[code] = {
            "Land": country,
            "Volatilitetsmodell": model.volatility.__class__.__name__,
            "Distribusjon": model.distribution.name,
            "AIC": round(result.aic, 4),
            "BIC": round(result.bic, 4),
            "Log-likelihood": round(result.loglikelihood, 4)
        }

    # --- Opprett sammendrag DataFrame ---
    df_summary = pd.DataFrame(summary_rows.values())
    df_summary = df_summary[[
        "Land", "Volatilitetsmodell", "Distribusjon", "AIC", "BIC", "Log-likelihood"
    ]]

    # --- Skriv til Excel ---
    with pd.ExcelWriter(file_path, engine="openpyxl", mode="w") as writer:
        for sheet, df in temp_writer_data.items():
            df.to_excel(writer, sheet_name=sheet, index=False)
        df_summary.to_excel(writer, sheet_name="GARCH-sammendrag", index=False)

    # --- Ferdig ---
    print(f"GARCH-resultater lagret som faner i: {file_path.resolve()}")
```

In [21]:

```
def test_egarch_variants(
    series: pd.Series,
    distributions: list = ["normal", "t", "skewt"]
) -> pd.DataFrame:
    """
    Estimerer EGARCH-modeller for en gitt serie over ulike (p, q)-kombinasjoner og fordelinger.

    Parametre:
    ----------
    series : pd.Series
        Stasjonær inputserie (f.eks. differensierte priser).
    distributions : list, optional
        Liste over fordelinger som skal testes (f.eks. ["t", "skewt"]). Standard er ["normal", "t", "skewt"].

    Returnerer:
    -----------
    pd.DataFrame
        DataFrame med p, q, fordeling, AIC, BIC og log-likelihood, sortert etter AIC.
    """

    from arch import arch_model

    # --- Definer (p, q)-kombinasjoner ---
    p_q_combos = [(1, 1), (1, 2), (2, 1), (2, 2)]

    results = []

    # --- Estimer modeller ---
    for p, q in p_q_combos:
        for dist in distributions:
            try:
                model = arch_model(series, vol="EGARCH", p=p, q=q, dist=dist)
                res = model.fit(disp="off")

                results.append({
                    "p": p,
                    "q": q,
                    "dist": dist,
                    "loglikelihood": res.loglikelihood,
```

```
                    "aic": res.aic,
                    "bic": res.bic
                })

            except Exception as e:
                print(f"Feil med EGARCH({p},{q}) - {dist}: {e}")

    # --- Returner resultater som DataFrame ---
    return pd.DataFrame(results).sort_values("aic").reset_index(drop=True)
```

In [22]:
```
def load_daily_prices(
    years: list[int],
    input_dir: Path,
    columns_to_keep: list[str]
) -> pd.DataFrame:
    """
    Leser inn og samler daglige prisdata for oppgitte år.

    Parametre:
    ----------
    years : list[int]
        Liste over årstall som skal leses inn.
    input_dir : Path
        Mappe hvor CSV-filene ligger.
    columns_to_keep : list[str]
        Liste over kolonner som skal beholdes fra hver fil.

    Returnerer:
    -----------
    pd.DataFrame
        Samlet og renset DataFrame med dato som indeks.
    """

    dataframes = []
    missing_years = []

    # --- Les inn hvert år ---
    for year in years:
        file_path = input_dir / f"daily_aggregate_{year}.csv"

        if file_path.exists():
            df = pd.read_csv(
                file_path,
                delimiter=";",
                decimal=",",
                thousands="."
            )
            df["Delivery Date CET"] = pd.to_datetime(df["Delivery Date CET"])
            dataframes.append(df)
        else:
            missing_years.append(year)

    # --- Sjekk at filer er funnet ---
    if not dataframes:
        raise FileNotFoundError("Ingen CSV-filer ble funnet i input-mappen.")

    if missing_years:
        print(f"Følgende år manglet filer og ble hoppet over: {missing_years}")

    # --- Slå sammen og bearbeid ---
    combined = pd.concat(dataframes, ignore_index=True)

    # Rens kolonnenavn
    combined.columns = [col.replace(" (EUR)", "") for col in combined.columns]

    # Behold kun nødvendige kolonner og fjern NA
    combined = combined[columns_to_keep].dropna()

    # Sett dato som indeks og sorter
    combined.set_index("Delivery Date CET", inplace=True)
    combined.sort_index(inplace=True)

    return combined
```

In [23]:
```
# --- Definer kolonner som skal beholdes ---
columns_to_keep = ["Delivery Date CET", "GER", "NO2"]

# --- Les inn daglige priser ---
daily_prices = load_daily_prices(
    years=YEARS,
    input_dir=INPUT_DIR,
    columns_to_keep=columns_to_keep
)
```

In [24]:

```python
def descriptive_analysis(
    data: pd.DataFrame,
    filnavn: str = "output/excel/data.xlsx",
    prefix: str = None,
    break_date: pd.Timestamp = pd.Timestamp("2022-02-24")
) -> None:
    """
    Utfører deskriptiv analyse og lagrer resultatene i én Excel-fane,
    med separate overskrifter for hele perioden, før og etter bruddet.

    Parametre:
    ----------
    data : pd.DataFrame
        Inndata med datetime-indeks.
    filnavn : str, optional
        Sti til Excel-filen hvor resultatene skal lagres. Standard er "output/excel/data.xlsx".
    prefix : str, optional
        Prefix for navnet på Excel-arket. Hvis None, brukes variabelnavnet automatisk.
    break_date : pd.Timestamp, optional
        Dato for bruddpunktet (default = 24. februar 2022).

    Returnerer:
    -----------
    None
    """

    from openpyxl import load_workbook

    # --- Sett opp prefix om nødvendig ---
    if prefix is None:
        callers_local_vars = inspect.currentframe().f_back.f_locals.items()
        prefix = next((name for name, val in callers_local_vars if val is data), 'data')
        prefix += "_stats"

    # --- Del opp datasettet ---
    parts = {
        "HELE PERIODEN": data,
        "FØR INVASJONEN": data[data.index < break_date],
        "ETTER INVASJONEN": data[data.index >= break_date],
    }

    output_path = Path(filnavn)
    output_path.parent.mkdir(parents=True, exist_ok=True)

    # --- Opprett fil om den ikke eksisterer ---
    if not output_path.exists():
        with pd.ExcelWriter(output_path, engine="openpyxl") as writer:
            pd.DataFrame([["Midlertidig ark, kan slettes."]]).to_excel(writer, sheet_name="temp")

    arkfane = prefix[:31]
    startrow = 0

    # --- Skriv analyser til Excel ---
    with pd.ExcelWriter(output_path, mode="a", engine="openpyxl", if_sheet_exists="overlay") as writer:
        for delnavn, subset in parts.items():
            description = []

            for col in subset.columns:
                serie = subset[col].dropna()
                if serie.empty:
                    continue

                q1 = serie.quantile(0.25)
                q3 = serie.quantile(0.75)

                description.append({
                    "Serie": col,
                    "Minimum": serie.min(),
                    "1. kvartil": q1,
                    "Median": serie.median(),
                    "3. kvartil": q3,
                    "Maksimum": serie.max(),
                    "Gjennomsnitt": serie.mean(),
                    "Standardavvik": serie.std(),
                    "Skjevhet": skew(serie, bias=False),
                    "Kurtosis": kurtosis(serie, fisher=True, bias=False),
                })

                print(f"- Serie: {col:<10} | Periode: {delnavn:<13} | Antall obs: {len(serie)}")

            df_description = pd.DataFrame(description).round(2)
```

```
            # Skriv tittel og data
            pd.DataFrame([[delnavn]]).to_excel(
                writer,
                sheet_name=arkfane,
                startrow=startrow,
                startcol=0,
                index=False,
                header=False
            )

            df_description.to_excel(
                writer,
                sheet_name=arkfane,
                startrow=startrow,
                startcol=1,
                index=False
            )

            startrow += len(df_description) + 3

    # --- Formatering med openpyxl ---
    wb = load_workbook(output_path)
    ws = wb[arkfane]
    bold_font = Font(bold=True)

    # Sett fet skrift på deloverskrifter
    for row in ws.iter_rows(min_row=1, max_row=startrow):
        if row[0].value in parts.keys():
            row[0].font = bold_font

    # Juster kolonnebredder
    for col in ws.columns:
        max_length = 0
        column = col[0].column_letter
        for cell in col:
            if cell.value is not None:
                try:
                    max_length = max(max_length, len(str(cell.value)))
                except Exception:
                    pass
        adjusted_width = max_length + 2
        ws.column_dimensions[column].width = adjusted_width

    # --- Lagre fil ---
    wb.save(output_path)

    print(f"\nDeskriptiv analyse samlet i én fane: '{arkfane}' i filen {filnavn}")
```

DCC-modellen: Fra teori til kode

Vi viser her hvordan variablene i DCC-modellen (slik den er definert i V-Lab-dokumentasjonen) samsvarer med variablene i vårt eget datasett og kode.

Teori og dokumentasjon:
https://vlab.stern.nyu.edu/docs/correlation/GARCH-DCC

Differensiert serie brukt i modellen

Differansen Δs_t brukes som erstatning for log-avkastning:

$$\Delta s_t = s_t - s_{t-1}$$

Kode:

```
transformed_diff = transformed_prices.diff().dropna()
```

Dette tilsvarer modellen:

$$r_t = \mu_t + \varepsilon_t$$

Hvor $\mu_t \approx 0$, og ε_t estimeres videre med GARCH.

EGARCH-estimering per serie

For hver tidsserie i, estimeres residualer og betinget volatilitet:

$$\varepsilon_{i,t} \sim \text{EGARCH}(1, 1)$$

Kode:

```
res = arch_model(...).fit()
res.resid # → ε_{i,t}
res.conditional_volatility # → √{h_{i,t}}
```

Standardiserte residualer

Standardisering gir vektor z_t, som er input til DCC-modellen:

$$z_{i,t} = \frac{\varepsilon_{i,t}}{\sqrt{h_{i,t}}}$$

Kode:

```
standardized_resid[col] = res.resid / res.conditional_volatility
```

Estimering av DCC-parametere

Parametrene α og β estimeres ved å minimere DCC log-likelihood loss:

$$\min_{\alpha,\beta} \sum_t \left(\log \det R_t + z_t^\top R_t^{-1} z_t \right)$$

Kode:

```
opt_result = minimize(dcc_loss, ...)
alpha, beta = opt_result.x
```

Dynamisk kovarians

Den dynamiske kovariansmatrisen Q_t beskriver samvariasjonen mellom de standardiserte residualene over tid. Den beregnes rekursivt som:

$$Q_t = (1 - \alpha - \beta)\bar{Q} + \alpha z_{t-1} z_{t-1}^\top + \beta Q_{t-1}$$

hvor:

α, β er estimert via optimering

$\bar{Q}$ er gjennomsnittlig kovariansmatrise for residualene:

$$\bar{Q} = \frac{1}{T} \sum_{t=1}^{T} z_t z_t^\top$$

Kode:

```
Q_bar = np.cov(standardized_resid.T)
Q_list = [...]      # alle Q_t
```

Dynamisk korrelasjonsmatrise

For å få en gyldig korrelasjonsmatrise, normaliserer vi Q_t til R_t slik:

$$R_t = D_t^{-1} Q_t D_t^{-1}$$

$$D_t = \text{diag}\left(\sqrt{Q_{11,t}}, \sqrt{Q_{22,t}}, \ldots, \sqrt{Q_{nn,t}} \right)$$

Kode:

```
R_list = [...]      # alle R_t
```

Validering og diagnostikk

Loss over tid og samlet DCC-loss benyttes til evaluering:

- **Total loss**:

$$\sum_t \left(\log \det R_t + z_t^\top R_t^{-1} z_t \right)$$

- **Loss per tidssteg** gir innsikt i modellens svakheter i tid.

Kode:

```
loss_values = [...]
total_loss = sum(loss_values)
```

Variabeloversikt

Teoretisk symbol	Kodevariabel
s_t	`transformed_prices`
Δs_t	`transformed_diff`
$\varepsilon_{i,t}$	`res.resid`
$\sqrt{h_{i,t}}$	`res.conditional_volatility`
$z_{i,t}$	`standardized_resid`
Q_t, $\bar{Q}$	`Q_list`, `Q_bar`
R_t	`R_list`, `R_array`
α, β	`alpha`, `beta` (fra `opt_result`)
DCC log-likelihood loss	`loss_values`, `total_loss`

In [26]:
```
# --- Beregn daglige endringer i priser (robust mot null og negative verdier) ---
daily_prices_diff = daily_prices.diff().dropna()

# --- Funksjon som tester EGARCH-varianter ---
def test_egarch_variants(
    series: pd.Series,
    distributions: list = ["normal", "t", "skewt"]
) -> pd.DataFrame:
    """
    Estimerer EGARCH-modeller for en gitt serie.
    """
    p_q_combos = [(1, 1)]  # Eventuelt utvid til flere kombinasjoner
    results = []

    for p, q in p_q_combos:
        for dist in distributions:
            try:
                model = arch_model(series, vol="EGARCH", p=p, q=q, dist=dist)
                res = model.fit(disp="off")
                results.append({
                    "p": p,
                    "q": q,
                    "dist": dist,
                    "loglikelihood": res.loglikelihood,
                    "aic": res.aic,
                    "bic": res.bic
                })
            except Exception as e:
```

```
                print(f"Feil med EGARCH({p},{q}) - {dist}: {e}")

    return pd.DataFrame(results)

# --- Estimer modeller for hver serie (bruker kun t-fordeling) ---
results_ger = test_egarch_variants(daily_prices_diff["GER"])
results_ger["serie"] = "GER"

results_no2 = test_egarch_variants(daily_prices_diff["NO2"])
results_no2["serie"] = "NO2"

# --- Kombiner og sorter etter laveste absolutt AIC ---
df_all = pd.concat([results_ger, results_no2], ignore_index=True)
df_all.rename(columns={"dist": "distribution"}, inplace=True)
df_all["abs_aic"] = df_all["aic"].abs()
df_all_sorted = df_all.sort_values(by="abs_aic").reset_index(drop=True)

# --- Lag kolonne med modellnavn (p,q distribution) ---
df_all_sorted["modell"] = df_all_sorted.apply(
    lambda row: f"({row['p']},{row['q']}) {row['distribution']}",
    axis=1
)

# --- Del opp i separate tabeller for GER og NO2 ---
df_ger = df_all_sorted[df_all_sorted["serie"] == "GER"].reset_index(drop=True).round(2)
df_no2 = df_all_sorted[df_all_sorted["serie"] == "NO2"].reset_index(drop=True).round(2)

# --- Vis tabeller ---
display(df_ger.style.set_caption(f"EGARCH-modeller for {NAME_MAP['GER']} (kun t-fordeling, sortert etter laveste
display(df_no2.style.set_caption(f"EGARCH-modeller for {NAME_MAP['NO2']} (kun t-fordeling, sortert etter laveste

# --- Legg til kolonne med visningsnavn for figurer ---
df_all_sorted["serie_navn"] = df_all_sorted["serie"].map(NAME_MAP)

# --- Plot AIC ---
fig_aic = px.bar(
    df_all_sorted,
    x="modell",
    y="aic",
    color="serie_navn",
    barmode="group",
    title="AIC for EGARCH-modeller med t-fordeling",
    labels={"modell": "Modell (p,q)", "aic": "AIC"},
    hover_data=["p", "q", "distribution", "loglikelihood"]
)
fig_aic.update_layout(title_font_size=18, legend_title_text="Serie", xaxis_tickangle=-45)
fig_aic.show()

# --- Plot BIC ---
fig_bic = px.bar(
    df_all_sorted,
    x="modell",
    y="bic",
    color="serie_navn",
    barmode="group",
    title="BIC for EGARCH-modeller med t-fordeling",
    labels={"modell": "Modell (p,q)", "bic": "BIC"},
    hover_data=["p", "q", "distribution", "loglikelihood"]
)
fig_bic.update_layout(title_font_size=18, legend_title_text="Serie", xaxis_tickangle=-45)
fig_bic.show()

# --- Plot Log-likelihood ---
fig_ll = px.bar(
    df_all_sorted,
    x="modell",
    y="loglikelihood",
    color="serie_navn",
    barmode="group",
    title="Log-likelihood for EGARCH-modeller med t-fordeling",
    labels={"modell": "Modell (p,q)", "loglikelihood": "Log-likelihood"},
    hover_data=["p", "q", "distribution", "aic"]
)
fig_ll.update_layout(title_font_size=18, legend_title_text="Serie", xaxis_tickangle=-45)
fig_ll.show()

# --- Lagre til Excel ---
excel_path = EXCEL_DIR / "data.xlsx"
sheet_name = "EGARCH-varianter"

try:
    # Prøv å legge til hvis filen finnes
    with ExcelWriter(excel_path, engine="openpyxl", mode="a", if_sheet_exists="replace") as writer:
```

```
        df_all_sorted.round(2).to_excel(writer, sheet_name=sheet_name, index=False)
except FileNotFoundError:
    # Hvis filen ikke finnes, lag en ny
    with ExcelWriter(excel_path, engine="openpyxl", mode="w") as writer:
        df_all_sorted.round(2).to_excel(writer, sheet_name=sheet_name, index=False)

print(f"Lagret EGARCH-resultater til '{sheet_name}' i '{excel_path.name}'")
```

EGARCH-modeller for Tyskland (kun t-fordeling, sortert etter laveste |AIC|)

	p	q	distribution	loglikelihood	aic	bic	serie	abs_aic	modell
0	1	1	skewt	-9189.180000	18390.360000	18423.990000	GER	18390.360000	(1,1) skewt
1	1	1	t	-9199.940000	18409.880000	18437.900000	GER	18409.880000	(1,1) t
2	1	1	normal	-9273.050000	18554.100000	18576.510000	GER	18554.100000	(1,1) normal

EGARCH-modeller for Norge (kun t-fordeling, sortert etter laveste |AIC|)

	p	q	distribution	loglikelihood	aic	bic	serie	abs_aic	modell
0	1	1	skewt	-7111.160000	14234.320000	14267.950000	NO2	14234.320000	(1,1) skewt
1	1	1	t	-7113.200000	14236.410000	14264.430000	NO2	14236.410000	(1,1) t
2	1	1	normal	-7268.060000	14544.110000	14566.530000	NO2	14544.110000	(1,1) normal

```
Lagret EGARCH-resultater til 'EGARCH-varianter' i 'data.xlsx'
```

In [27]:

```python
# --- Forbered data ---
daily_prices_diff = daily_prices.diff().dropna()

# --- Forbered datastrukturer ---
standardized_resid = pd.DataFrame(index=daily_prices_diff.index)
egarch_volatility = pd.DataFrame(index=daily_prices_diff.index)
garch_models = {}
adf_results = []

# --- EGARCH-modellering + ADF- og ARCH-tester ---
for col in daily_prices_diff.columns:
    print(f"\n{'='*80}\nModellering av serie: {col}\n{'='*80}")

    orig_series = daily_prices[col].dropna()
    diff_series = daily_prices_diff[col].dropna()

    # ADF-tester
    adf_stat_orig, adf_pval_orig, *_ = adfuller(orig_series)
    adf_stat_diff, adf_pval_diff, *_ = adfuller(diff_series)
```

```
    print(f"\nADF-test FØR differensiering:")
    print(f"Statistikk = {adf_stat_orig:.4f}, p-verdi = {adf_pval_orig:.4f}")
    print("→ Stasjonær" if adf_pval_orig < 0.05 else "→ Ikke-stasjonær")

    print(f"\nADF-test ETTER differensiering:")
    print(f"Statistikk = {adf_stat_diff:.4f}, p-verdi = {adf_pval_diff:.4f}")
    print("→ Stasjonær" if adf_pval_diff < 0.05 else "→ Ikke-stasjonær")

    adf_results.append({
        "Serie": col,
        "ADF-statistikk (før)": adf_stat_orig,
        "p-verdi (før)": adf_pval_orig,
        "Stasjonær (før)": "Ja" if adf_pval_orig < 0.05 else "Nei",
        "ADF-statistikk (etter)": adf_stat_diff,
        "p-verdi (etter)": adf_pval_diff,
        "Stasjonær (etter)": "Ja" if adf_pval_diff < 0.05 else "Nei"
    })

    # ARCH-test før modellering
    arch_stat_pre, arch_pval_pre, *_ = het_arch(diff_series)
    print(f"\nARCH-test før modellering:")
    print(f"LM-statistikk = {arch_stat_pre:.4f}, p-verdi = {arch_pval_pre:.4f}")

    # Estimer EGARCH(1,1)
    model = arch_model(diff_series, vol="EGARCH", p=1, o=1, q=1, dist="t")
    result = model.fit(disp="off")
    garch_models[col] = result

    resid = result.resid
    cond_vol = result.conditional_volatility
    standardized = resid / cond_vol

    standardized_resid[col] = standardized
    egarch_volatility[col] = cond_vol

    # ARCH-test etter modellering
    arch_stat_post, arch_pval_post, *_ = het_arch(standardized.dropna())
    print(f"\nARCH-test etter modellering:")
    print(f"LM-statistikk = {arch_stat_post:.4f}, p-verdi = {arch_pval_post:.4f}")

    # Diagnostikk
    print("\nModelloppsummering:")
    print(result.summary())
    print("\nLjung-Box (lag 10):")
    print(acorr_ljungbox(standardized.dropna(), lags=10, return_df=True))
    print("\nKvadrerte residualer:")
    print(acorr_ljungbox(standardized.dropna()**2, lags=10, return_df=True))

# --- DCC-tapsfunksjon ---
def dcc_loss(params, residuals):
    alpha, beta = params
    if alpha < 0 or beta < 0 or (alpha + beta >= 1):
        return np.inf
    T = residuals.shape[0]
    Q_bar = np.cov(residuals.T)
    Q = Q_bar.copy()
    loss = 0.0
    for t in range(T):
        z_t = residuals.iloc[t].values.reshape(-1, 1)
        Q = (1 - alpha - beta) * Q_bar + alpha * (z_t @ z_t.T) + beta * Q
        D_inv = np.diag(1 / np.sqrt(np.diag(Q)))
        R_t = D_inv @ Q @ D_inv
        sign, logdet = np.linalg.slogdet(R_t)
        if sign <= 0:
            return np.inf
        e_t = residuals.iloc[t].values
        loss += logdet + e_t.T @ np.linalg.inv(R_t) @ e_t
    return loss

# --- Estimer DCC ---
opt_result = minimize(
    dcc_loss,
    [0.01, 0.98],
    args=(standardized_resid.dropna(),),
    method="SLSQP",
    constraints=[
        {"type": "ineq", "fun": lambda x: x[0]},
        {"type": "ineq", "fun": lambda x: x[1]},
        {"type": "ineq", "fun": lambda x: 1.0 - x[0] - x[1]}
    ],
    options={"disp": True}
)
```

```python
alpha, beta = opt_result.x
print(f"\nOptimal DCC-parametere: alpha = {alpha:.4f}, beta = {beta:.4f}")

# --- Beregn dynamiske kovarianser og korrelasjoner ---
T = len(standardized_resid.dropna())
Q_bar = np.cov(standardized_resid.dropna().T)
Q = Q_bar.copy()
R_list, Q_list = [], []

for t in range(T):
    z_t = standardized_resid.dropna().iloc[t].values.reshape(-1, 1)
    Q = (1 - alpha - beta) * Q_bar + alpha * (z_t @ z_t.T) + beta * Q
    D_inv = np.diag(1 / np.sqrt(np.diag(Q)))
    R_t = D_inv @ Q @ D_inv
    Q_list.append(Q.copy())
    R_list.append(R_t)

# --- Lag DataFrames for DCC-resultater ---
dates = standardized_resid.dropna().index
series_names = standardized_resid.columns.tolist()

dcc_covariances = pd.DataFrame({
    f"{series_names[0]}-{series_names[1]}": [Q[0, 1] for Q in Q_list]
}, index=dates)

correlation_data = {}
for i in range(len(series_names)):
    for j in range(i + 1, len(series_names)):
        pair = f"{series_names[i]}-{series_names[j]}"
        correlation_data[pair] = [R[i, j] for R in R_list]

dcc_correlations = pd.DataFrame(correlation_data, index=dates)

# --- Beregn log-likelihood tap ---
loss_values = []
for t, R_t in enumerate(R_list):
    sign, logdet = np.linalg.slogdet(R_t)
    if sign <= 0:
        continue
    e_t = standardized_resid.dropna().iloc[t].values
    quad_form = e_t.T @ np.linalg.inv(R_t) @ e_t
    loss_values.append(logdet + quad_form)
total_loss = sum(loss_values)
print(f"\nTotal DCC log-likelihood loss: {total_loss:.4f}")

# --- Eksportér resultater ---
print("\nStarter eksport...")
adf_df = pd.DataFrame(adf_results).set_index("Serie")
save_garch_summaries_txt(garch_models)
save_to_excel(
    raw_prices=daily_prices,
    differenced_prices=daily_prices_diff,
    volatility=egarch_volatility,
    standardized_resid=standardized_resid,
    dcc_corrs=dcc_correlations,
    dcc_covs=dcc_covariances,
    dcc_loss=total_loss,
    dcc_alpha=alpha,
    dcc_beta=beta,
    adf_results=adf_df
)
export_garch_results_to_excel(garch_models, "data.xlsx")
```

```
================================================================================
Modellering av serie: GER
================================================================================

ADF-test FØR differensiering:
Statistikk = -2.8312, p-verdi = 0.0539
→ Ikke-stasjonær

ADF-test ETTER differensiering:
Statistikk = -12.1556, p-verdi = 0.0000
→ Stasjonær

ARCH-test før modellering:
LM-statistikk = 336.3552, p-verdi = 0.0000

ARCH-test etter modellering:
LM-statistikk = 75.5584, p-verdi = 0.0000

Modelloppsummering:
                        Constant Mean - EGARCH Model Results
================================================================================
```

```
Dep. Variable:                          GER   R-squared:                       0.000
Mean Model:                  Constant Mean   Adj. R-squared:                  0.000
Vol Model:                           EGARCH   Log-Likelihood:               -9149.73
Distribution:      Standardized Student's t   AIC:                           18311.5
Method:                 Maximum Likelihood   BIC:                           18345.1
                                             No. Observations:                2008
Date:                      Sun, Apr 27 2025   Df Residuals:                    2007
Time:                              12:55:17   Df Model:                           1
                              Mean Model
============================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
----------------------------------------------------------------------------
mu            -1.2423      0.296     -4.190  2.786e-05 [ -1.823, -0.661]
                            Volatility Model
============================================================================
                 coef    std err          t      P>|t|    95.0% Conf. Int.
----------------------------------------------------------------------------
omega          0.1221  5.601e-02      2.179  2.930e-02 [1.229e-02,  0.232]
alpha[1]       0.2540  8.271e-02      3.071  2.132e-03 [9.190e-02,  0.416]
gamma[1]      -0.2572  3.585e-02     -7.175  7.221e-13   [ -0.328, -0.187]
beta[1]        0.9840  8.042e-03    122.364      0.000   [  0.968,  1.000]
                            Distribution
============================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
----------------------------------------------------------------------------
nu             5.0980      0.620      8.216  2.103e-16 [  3.882,  6.314]
============================================================================

Covariance estimator: robust

Ljung-Box (lag 10):
       lb_stat     lb_pvalue
1     6.292161  1.212731e-02
2   133.581364  9.844084e-30
3   146.704947  1.353688e-31
4   156.633919  7.704467e-33
5   181.582297  2.457477e-37
6   181.608217  1.545334e-36
7   346.443227  7.112379e-71
8   347.364807  3.306205e-70
9   371.788351  1.419649e-74
10  375.623016  1.440372e-74

Kvadrerte residualer:
      lb_stat     lb_pvalue
1   49.301006  2.195499e-12
2   64.317416  1.080564e-14
3   65.616553  3.702261e-14
4   68.702717  4.263837e-14
5   68.805392  1.816127e-13
6   72.252436  1.410033e-13
7   74.621841  1.711443e-13
8   74.753665  5.525901e-13
9   74.798459  1.731913e-12
10  77.733097  1.394403e-12

================================================================================
Modellering av serie: NO2
================================================================================

ADF-test FØR differensiering:
Statistikk = -3.0718, p-verdi = 0.0287
→ Stasjonær

ADF-test ETTER differensiering:
Statistikk = -10.2669, p-verdi = 0.0000
→ Stasjonær

ARCH-test før modellering:
LM-statistikk = 251.8866, p-verdi = 0.0000

ARCH-test etter modellering:
LM-statistikk = 26.3190, p-verdi = 0.0033

Modelloppsummering:
                      Constant Mean - EGARCH Model Results
==============================================================================
Dep. Variable:                          NO2   R-squared:                       0.000
Mean Model:                  Constant Mean   Adj. R-squared:                  0.000
Vol Model:                           EGARCH   Log-Likelihood:               -7112.86
Distribution:      Standardized Student's t   AIC:                           14237.7
Method:                 Maximum Likelihood   BIC:                           14271.3
                                             No. Observations:                2008
```

```
Date:                       Sun, Apr 27 2025   Df Residuals:                     2007
Time:                              12:55:18   Df Model:                           1
                                 Mean Model
============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
----------------------------------------------------------------------------
mu         -1.9897e-04  1.009e-02 -1.973e-02      0.984 [-1.997e-02,1.957e-02]
                            Volatility Model
============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
----------------------------------------------------------------------------
omega          0.1357  3.618e-02      3.750  1.770e-04    [6.476e-02,  0.207]
alpha[1]       0.6487  8.115e-02      7.994  1.309e-15      [  0.490,  0.808]
gamma[1]      -0.0224  2.592e-02     -0.866      0.387 [-7.325e-02,2.837e-02]
beta[1]        0.9934  4.070e-03    244.073      0.000      [  0.985,  1.001]
                            Distribution
============================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
----------------------------------------------------------------------------
nu             3.1283      0.232     13.461  2.640e-41 [  2.673,  3.584]
============================================================================

Covariance estimator: robust

Ljung-Box (lag 10):
       lb_stat     lb_pvalue
1     0.247659  6.187283e-01
2    48.455725  3.005889e-11
3    48.812728  1.429868e-10
4    48.889501  6.157124e-10
5    60.936986  7.781120e-12
6    62.191164  1.613029e-11
7   130.233369  5.617871e-25
8   131.009884  1.746921e-24
9   140.832073  6.945802e-26
10  140.832143  2.845650e-25

Kvadrerte residualer:
      lb_stat  lb_pvalue
1    2.460354   0.116752
2    2.544411   0.280213
3    5.040744   0.168838
4   13.994631   0.007312
5   15.185811   0.009597
6   19.928191   0.002852
7   20.036526   0.005491
8   20.258736   0.009400
9   23.166989   0.005832
10  25.721975   0.004133
Singular matrix E in LSQ subproblem    (Exit mode 5)
            Current function value: 1870.7046697057328
            Iterations: 7
            Function evaluations: 26
            Gradient evaluations: 7

Optimal DCC-parametere: alpha = 0.1103, beta = 0.8897

Total DCC log-likelihood loss: 1870.7047

Starter eksport...
GARCH-sammendrag lagret til: output/garch_summaries.txt
Alt er lagret i Excel: output\excel\data_series.xlsx
GARCH-resultater lagret som faner i: C:\Users\julia\DDC-Garch\output\excel\data.xlsx
```

In [28]:
```python
# --- Deskriptiv analyse av daglige strømpriser ---
descriptive_analysis(daily_prices)

# --- Deskriptiv analyse av EGARCH-volatilitet ---
descriptive_analysis(egarch_volatility)

# --- Deskriptiv analyse av standardiserte residualer ---
descriptive_analysis(standardized_resid)

# --- Deskriptiv analyse av DCC-korrelasjoner ---
descriptive_analysis(dcc_correlations)

# --- Deskriptiv analyse av DCC-kovarianser ---
descriptive_analysis(dcc_covariances)
```

```
- Serie: GER        | Periode: HELE PERIODEN | Antall obs: 2009
- Serie: NO2        | Periode: HELE PERIODEN | Antall obs: 2009
- Serie: GER        | Periode: FØR INVASJONEN | Antall obs: 967
- Serie: NO2        | Periode: FØR INVASJONEN | Antall obs: 967
- Serie: GER        | Periode: ETTER INVASJONEN | Antall obs: 1042
- Serie: NO2        | Periode: ETTER INVASJONEN | Antall obs: 1042

Deskriptiv analyse samlet i én fane: 'daily_prices_stats' i filen output/excel/data.xlsx
- Serie: GER        | Periode: HELE PERIODEN | Antall obs: 2008
- Serie: NO2        | Periode: HELE PERIODEN | Antall obs: 2008
- Serie: GER        | Periode: FØR INVASJONEN | Antall obs: 966
- Serie: NO2        | Periode: FØR INVASJONEN | Antall obs: 966
- Serie: GER        | Periode: ETTER INVASJONEN | Antall obs: 1042
- Serie: NO2        | Periode: ETTER INVASJONEN | Antall obs: 1042

Deskriptiv analyse samlet i én fane: 'egarch_volatility_stats' i filen output/excel/data.xlsx
- Serie: GER        | Periode: HELE PERIODEN | Antall obs: 2008
- Serie: NO2        | Periode: HELE PERIODEN | Antall obs: 2008
- Serie: GER        | Periode: FØR INVASJONEN | Antall obs: 966
- Serie: NO2        | Periode: FØR INVASJONEN | Antall obs: 966
- Serie: GER        | Periode: ETTER INVASJONEN | Antall obs: 1042
- Serie: NO2        | Periode: ETTER INVASJONEN | Antall obs: 1042

Deskriptiv analyse samlet i én fane: 'standardized_resid_stats' i filen output/excel/data.xlsx
- Serie: GER-NO2    | Periode: HELE PERIODEN | Antall obs: 2008
- Serie: GER-NO2    | Periode: FØR INVASJONEN | Antall obs: 966
- Serie: GER-NO2    | Periode: ETTER INVASJONEN | Antall obs: 1042

Deskriptiv analyse samlet i én fane: 'dcc_correlations_stats' i filen output/excel/data.xlsx
- Serie: GER-NO2    | Periode: HELE PERIODEN | Antall obs: 2008
- Serie: GER-NO2    | Periode: FØR INVASJONEN | Antall obs: 966
- Serie: GER-NO2    | Periode: ETTER INVASJONEN | Antall obs: 1042

Deskriptiv analyse samlet i én fane: 'dcc_covariances_stats' i filen output/excel/data.xlsx
```

Plotting

Daglige strømpriser

In [31]:

```python
# --- Plott tidsserie for daglige strømpriser ---
plot_timeseries(
    data=daily_prices,
    title="Daglige strømpriser",
    y_title="Pris (EUR/MWh)",
    filename="daily_prices",
    show_break=True
)

# --- Plott histogram for daglige strømpriser ---
plot_histogram(
    data=daily_prices,
    title="Daglige strømpriser",
    x_title="Pris (EUR/MWh)",
    filename="daily_prices"
)

# --- Plott scatter-plot for daglige strømpriser ---
plot_scatter(
    data=daily_prices,
    title="Daglige strømpriser",
    y_title="Pris (EUR/MWh)",
    filename="daily_prices",
    show_break=True
)

# --- Plott 7-dagers glidende gjennomsnitt for daglige strømpriser ---
plot_rolling_average(
    data=daily_prices,
    window=7,
    title="Strømpriser (7-dagers glidende gjennomsnitt)",
    y_title="Pris (EUR/MWh)",
    filename="daily_prices_rolling",
    show=True,
    show_break=True
)

# --- Sammenlign strømpriser i Tyskland og Norge med histogram ---
plot_histogram_comparison(
    series1=daily_prices["GER"],
    series2=daily_prices["NO2"],
```

```
    label1="Tyskland",
    label2="Norge",
    title="Sammenligning av strømpriser i Tyskland og Norge",
    xlabel="Pris (EUR/MWh)",
    filename="daily_prices_comparison"
)
```

EGARCH-volatilitet

In [33]:
```python
# --- Plott EGARCH-volatilitet ---
egarch_volatility = pd.DataFrame({
    col: garch_models[col].conditional_volatility
    for col in daily_prices_diff.columns
})

plot_timeseries(
    data=egarch_volatility,
    title="EGARCH-volatilitet",
    y_title="Volatilitet",
    filename="egarch_volatility",
    show_break=True
)

# --- Plott histogram-sammenligning av volatilitet før og etter invasjonen ---
for col in ["GER", "NO2"]:
    plot_histogram_comparison(
        series1=egarch_volatility.loc[egarch_volatility.index < BREAK_DATE, col],
        series2=egarch_volatility.loc[egarch_volatility.index >= BREAK_DATE, col],
        label1="Før 24. feb 2022",
        label2="Etter 24. feb 2022",
        title=f"EGARCH-volatilitet for {NAME_MAP[col]} før og etter invasjonen",
        xlabel="Volatilitet",
        filename=f"egarch_volatility_{col.lower()}_histogram_comparison"
    )
```

EGARCH-residualer

In [35]:
```python
# --- Plott differensierte strømpriser ---
plot_timeseries(
    data=daily_prices_diff,
    title="Differensierte strømpriser",
    y_title="Endring i pris (EUR/MWh)",
    filename="daily_prices_diff",
    show_break=True
)

# --- Plott residualer fra GARCH-modellene ---
residuals_df = pd.DataFrame({
    col: garch_models[col].resid
    for col in daily_prices_diff.columns
})

plot_timeseries(
    data=residuals_df,
    title="Residualer",
    y_title="Residualer",
    filename="residuals",
    show_break=True
)

# --- Plott standardiserte residualer ---
plot_timeseries(
    data=standardized_resid,
    title="Standardiserte residualer",
    y_title="Standardiserte residualer",
    filename="standardized_resid",
    show_break=True
)
```

DCC korrelasjon

In [37]:
```
# --- Plott DCC tidsvarierende korrelasjon ---
plot_timeseries(
    data=dcc_correlations,
    title="DCC tidsvarierende korrelasjon",
    y_title="Korrelasjon",
    filename="dcc_correlations",
    show_break=True
)
```

DCC kovarians

In [39]:
```
# --- Plott DCC tidsvarierende kovarians ---
plot_timeseries(
    data=dcc_covariances,
```

```
    title="DCC tidsvarierende kovarians",
    y_title="Kovarians",
    filename="dcc_covariance",
    show_break=True
)
```

Sammenligning av daglige priser før og etter invasjonen

In [41]:
```
# --- Plott histogram-sammenligning av strømpriser før og etter invasjonen ---
for pair in daily_prices.columns:
    name = NAME_MAP.get(pair, pair)  # Bruk visningsnavn hvis tilgjengelig

    plot_histogram_comparison(
        series1=daily_prices.loc[daily_prices.index < BREAK_DATE, pair],
        series2=daily_prices.loc[daily_prices.index >= BREAK_DATE, pair],
        label1="Før 24. feb 2022",
        label2="Etter 24. feb 2022",
        title=f"Daglige strømpriser i {name} før og etter invasjonen",
        xlabel="Pris (EUR/MWh)",
        filename=f"daily_prices_{pair.replace(' ', '_').replace('-', '_')}_histogram_kde"
    )
```

Sammenligning av DCC-korrelasjon før og etter invasjonen

In [43]:
```
# --- Plott histogram-sammenligning av DCC-korrelasjoner før og etter invasjonen ---
for pair in dcc_correlations.columns:
    plot_histogram_comparison(
        series1=dcc_correlations.loc[dcc_correlations.index < BREAK_DATE, pair],
        series2=dcc_correlations.loc[dcc_correlations.index >= BREAK_DATE, pair],
        label1="Før 24. feb 2022",
        label2="Etter 24. feb 2022",
        title=f"Tidsvarierende korrelasjon før og etter invasjonen",
        xlabel="DCC-korrelasjon",
        filename=f"dcc_correlation_{pair.replace(' ', '_').replace('-', '_')}_histogram_kde"
    )
```

Sammenligning av DCC-kovarians før og etter invasjonen

In [45]:
```python
# --- Plott histogram-sammenligning av DCC-kovarianser før og etter invasjonen ---
for pair in dcc_covariances.columns:
    plot_histogram_comparison(
        series1=dcc_covariances.loc[dcc_covariances.index < BREAK_DATE, pair],
        series2=dcc_covariances.loc[dcc_covariances.index >= BREAK_DATE, pair],
        label1="Før 24. feb 2022",
        label2="Etter 24. feb 2022",
        title=f"Tidsvarierende kovarians før og etter invasjonen",
        xlabel="DCC-kovarians",
        filename=f"dcc_covariance_{pair.replace(' ', '_').replace('-', '_')}_histogram_kde"
    )
```

QQ-plot for residualer og standardiserte residualer

In [47]:

```python
# --- Plott QQ-plot for residualer og standardiserte residualer ---
for col in daily_prices_diff.columns:
    # Sett farger basert på kolonnenavn
    if "GER" in col:
        color = COLOR_1  # Tyskland
    elif "NO2" in col:
        color = COLOR_2  # Norge (NO2)
    else:
        color = COLOR_3  # Fallback-farge

    # Definer datasett (vanlige og standardiserte residualer)
    datasets = [
        (garch_models[col].resid.dropna(), "residualer", color),
        (standardized_resid[col].dropna(), "standardiserte residualer", color)
    ]

    for data, label, color in datasets:
        safe_label = label.replace(" ", "_").lower()
        filename = f"qq_{col.lower()}_{safe_label}"

        plot_qq(
            data=data,
            label=label,
            color=color,
            title=f"{NAME_MAP.get(col, col)} – QQ-plot for",
            filename=filename
        )
```

ACF og PACF

In [49]:
```
# --- Plott ACF og PACF for råpriser (nivådata) ---
plot_acf_pacf(
    series=daily_prices["GER"],
    title_prefix="Tyskland – priser",
    lags=20
)

plot_acf_pacf(
    series=daily_prices["NO2"],
    title_prefix="Norge – priser",
    lags=20
)

# --- Plott ACF og PACF for differensierte priser ---
plot_acf_pacf(
    series=daily_prices_diff["GER"],
    title_prefix="Tyskland – differensierte priser",
    lags=20
)

plot_acf_pacf(
    series=daily_prices_diff["NO2"],
    title_prefix="Norge – differensierte priser",
    lags=20
)

# --- Plott ACF og PACF for EGARCH-residualer ---
plot_acf_pacf(
    series=standardized_resid["GER"],
    title_prefix="Tyskland – EGARCH-residualer",
    lags=20
)

plot_acf_pacf(
    series=standardized_resid["NO2"],
    title_prefix="Norge – EGARCH-residualer",
    lags=20
)
```