Katapult Visual Investigation: Notebook 2 - Creation of English Visualizations

The purpose of this notebook is to read in the preprocessed data from `Katapult_Data_Processing.ipynb`, and to use this data to make the visualizations. In this notebook all images are made in English.

Last execution on December 18, 2025

In [1]:
```
# Import packages

import pandas as pd
import numpy as np
from pathlib import Path
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import matplotlib.transforms as mtransforms
from plotly.subplots import make_subplots
import plotly.graph_objects as go
```

In [2]:
```
# General image settings

mpl.rcParams['svg.fonttype'] = 'none'
plt.rcParams.update({
    "font.family": "Verdana",
    "font.size": 10,
    "text.color": "black",
    "axes.labelcolor": "black",
    "axes.titlecolor": "black",
    "xtick.color": "black",
    "ytick.color": "black"
})
```

In [3]:
```
# Create directories if needed
Path('./data/raw/bruegel/down').mkdir(parents=True, exist_ok=True)
Path('./data/raw/bruegel/ren').mkdir(parents=True, exist_ok=True)
Path('./data/processed/').mkdir(parents=True, exist_ok=True)
Path('./figs/english').mkdir(parents=True, exist_ok=True)
Path('./figs/german').mkdir(parents=True, exist_ok=True)

bruegel_raw_down_dir = Path('./data/raw/bruegel/down')
bruegel_raw_ren_dir = Path('./data/raw/bruegel/ren')
processed_data_dir = Path('./data/processed/')
figs_dir_english = Path('./figs/english/')
figs_dir_german = Path('./figs/german/')

# Load all datasets
eurostat_nrg_ti_gas = pd.read_csv(f'{processed_data_dir}/eurostat_nrg_ti_gas.csv
eurostat_nrg_ti_gasm = pd.read_csv(f'{processed_data_dir}/eurostat_nrg_ti_gasm.c
gas_reg_quart_imp = pd.read_csv(f'{processed_data_dir}/bruegel_gas_reg_quart_imp
```

```
lng_reg_quart_imp = pd.read_csv(f'{processed_data_dir}/bruegel_lng_reg_quart_imp
gas_storage_eu = pd.read_csv(f'{processed_data_dir}/GIE_agsi_gas_storage_eu.csv'
```

Visualization 1: EU Gas Imports from Russia since 01/2021 - Time Series with Events

Load Data and Prepare Timeline

The monthly data CSV is already assigned to a variable. First, all columns that hold a time period are gathered and also assigned to variables for repeated useage. When you open the respective CSV file you will see that there is **gas imports data for all member states and for the EU**. Therefore for each column (a column is one month) we should just sum up all rows which represent member states **OR** the 2 rows which cotain the data of the EU. We sum up the 2 EU rows:

In [4]:
```
# Get respective years or months of Eurostat datasets for later use
eurostat_year_cols = eurostat_nrg_ti_gas.columns[4:].to_list()
eurostat_month_cols = eurostat_nrg_ti_gasm.columns[4:].to_list()

# Just filter for the 2 EU entries
eu_gas_imports_ru_mid = eurostat_nrg_ti_gasm[eurostat_nrg_ti_gasm['import_countr

# Sum pipeline gas (row 0) and LNG gas (row 1) imports for all columns (months)
timeline = eu_gas_imports_ru_mid[eurostat_month_cols].sum().reset_index()
timeline.columns = ['date', 'imports']

# Convert date string to actual date format
timeline['date'] = pd.to_datetime(timeline['date'])
```

Next, some key events are defined, which will be show/annotated at the time series. Each event is a tuple with the following elements:

1. Date of the event
2. Text label of the event
3. y-coordinate for the text
4. x-coordinate for the text
5. Arrow-styling settings

In [5]:
```
events = [
    ('2022-02-24', 'Russia invades Ukraine', 1.02, 50, 'arc3,rad=0.2'),
    ('2022-04-27', 'Russia stops gas supply\nto Poland and Bulgaria', 0.84, 175,
    ('2022-05-18', 'REPowerEU proposal:\nreduce dependency\non Russian fossil fu
    ('2022-06-14', 'Nord Stream 1 gas flows\ncut to 40% of initial flows', 0.67,
    ('2022-07-27', 'Nord Stream 1 gas flows\ncut to 20% of initial flows', 0.35,
    ('2022-08-31', 'Shutdown of Nord Stream 1\npipeline due to maintenance', 0.5
    ('2022-09-26', 'Explosions at the\nNord Stream pipelines', 0.20, -100, 'arc3
    ('2023-02-15', 'Gas price cap\non EU market\ngets activated', 0.32, -10, 'ar
    ('2023-10-08', 'Rupture of\nBalticconnector pipeline', 0.28, 60, 'arc3,rad=0
    ('2024-04-22', 'Balticconnector pipeline\nreturns to service', 0.53, 70, 'ar
    ('2024-12-31', 'Ukraine ends gas transit\nfrom Russia to Europe\n(no extensi
    ('2025-01-31', 'Gas price cap on\nEU market expires', 0.5, 70, 'arc3,rad=0.1
]
```

Before plotting, a helper function is defined. This function returns the import value on the time series closest to a given date. It works as follows:

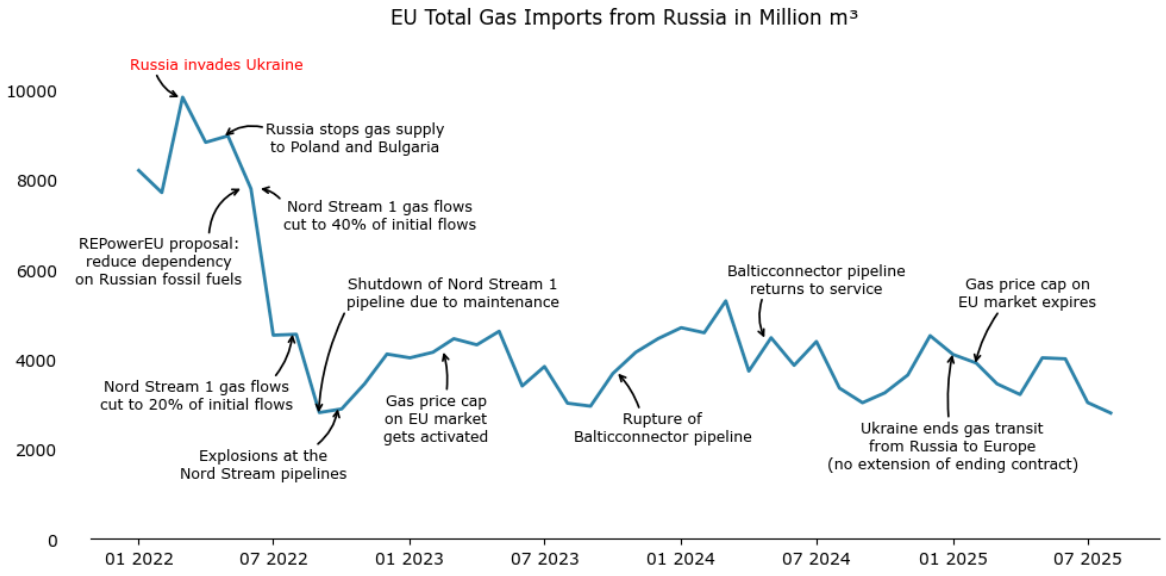
1. df['date'] - target creates a Series of time differences (Timedeltas) between every row's date and the target/event date ()
2. .abs() turns those differences into absolute values, so how far each date is from target, ignoring sign
3. .argmin() finds the index of the smallest difference, so the row whose date is nearest to target
4. df.iloc[idx]['imports'] looks up the imports value at that row and returns it as a float

In [6]:
```
def y_at_date(df, target):
    idx = (df['date'] - target).abs().argmin()
    return float(df.iloc[idx]['imports'])
```

Now, let's plot the timeline with the events:

In [7]:
```
fig_1, ax_1 = plt.subplots(figsize=(10, 5))

# Main series
ax_1.plot(timeline['date'], timeline['imports'], linewidth=2, color='#2E86AB')

# x in data coordinates, y in axes-fraction coordinates, so labels can be placed
text_transform = mtransforms.blended_transform_factory(ax_1.transData, ax_1.tran

for i, (date_str, label, y_lane, dx_days, style) in enumerate(events):
    d = pd.to_datetime(date_str)
    tip_y = y_at_date(timeline, d)

    # Manual horizontal nudge (in days)
    label_x = d + np.timedelta64(dx_days, 'D')

    # Vertical alignment of the text box depending on which lane it's in
    va = 'bottom' if y_lane >= 0.5 else 'top'
    ha = 'left' if i <= 1 else 'center'

    # Build arrowprops
    arrowprops = dict(arrowstyle='->', lw=1.2, color='black', shrinkA=0, shrinkB
    if i == 0:
        arrowprops['relpos'] = (0.15, 0)
    elif i == 1:
        arrowprops['relpos'] = (0, 0.8)
    elif i == 2:
        arrowprops['relpos'] = (0.8, 1)
    elif i == 3:
        arrowprops['relpos'] = (0, 0.9)
    elif i == 4:
        arrowprops['relpos'] = (0.9, 1)
    elif i == 5:
        arrowprops['relpos'] = (0, 0)
    elif i == 6:
        arrowprops['relpos'] = (0.8, 1)
    elif i == 7:
        arrowprops['relpos'] = (0.6, 1)
    elif i == 8:
        arrowprops['relpos'] = (0.4, 0)
```

```
    elif i == 9:
        arrowprops['relpos'] = (0.2, 0)
    elif i == 11:
        arrowprops['relpos'] = (0.3, 0)

    ax_1.annotate(
        label,                                  # Text to display
        xy=(d, tip_y),                          # Arrow tip on the data series (dat
        xycoords='data',                        # Interpret xy in data coordinates
        xytext=(label_x, y_lane),               # Label/text position: x in data, y
        textcoords=text_transform,              # Interpret xytext with blended tra
        ha='center', va=va, fontsize=9,         # Text alignment and size
        color='red' if i == 0 else 'black',     # Text of first event in red color
        arrowprops=arrowprops                   # Arrow styling
    )

# Set titles, axes, layout and general settings
ax_1.set_title('EU Total Gas Imports from Russia in Million m³', pad=35)
ax_1.set_ylim(bottom=0)
ax_1.spines['top'].set_visible(False)
ax_1.spines['right'].set_visible(False)
ax_1.spines['left'].set_visible(False)
ax_1.xaxis.set_major_locator(mdates.MonthLocator(bymonth=[1, 7]))
ax_1.xaxis.set_major_formatter(mdates.DateFormatter('%m %Y'))
ax_1.tick_params(axis='y', which='both', length=0, pad=20)
plt.tight_layout()

# Show image
plt.show()

# Save figure in respective directory
fig_1.savefig(f'{figs_dir_english}/Vis_1_EU_Gas_Imports_Timeline_Events.svg')
```



In [ ]:

Visualization 2: Gas Imports from Russia by EU Countries: Comparison of Time Periods

Data Preparation & Filtering

This section defines an ISO-2 to ISO-3 map, normalizes `EL` to `GR` for GeoJSON matching, and lists the covered ISO-2 countries. It then splits Eurostat columns into two windows: pre-war time with `pre_war_cols` are year-only columns from 1990 to 2021 from `eurostat_nrg_ti_gas`, and `mid_war_cols` are monthly `YYYY-MM` columns from 2022-01 to 2025-08 in `eurostat_nrg_ti_gasm`. This aligns country codes and separates the two time windows for analysis:

In [8]:
```
# ISO codes
iso2_to_iso3 = {
    'AT': 'AUT', 'BE': 'BEL', 'BG': 'BGR', 'HR': 'HRV', 'CY': 'CYP',
    'CZ': 'CZE', 'DE': 'DEU', 'DK': 'DNK', 'EE': 'EST', 'FI': 'FIN',
    'FR': 'FRA', 'EL': 'GRC',
    'HU': 'HUN', 'IE': 'IRL', 'IT': 'ITA',
    'LV': 'LVA', 'LT': 'LTU', 'LU': 'LUX', 'MT': 'MLT', 'NL': 'NLD',
    'PL': 'POL', 'PT': 'PRT', 'RO': 'ROU', 'SK': 'SVK', 'SI': 'SVN',
    'ES': 'ESP', 'SE': 'SWE'
}

# Map EL to GR for GeoJSON matching
iso2_mapping = {k: ('GR' if k == 'EL' else k) for k in iso2_to_iso3.keys()}
eu_countries = list(iso2_to_iso3.keys())

# Define time windows: eurostat_nrg_ti_gasm has years from 1990 to 2023, but we
pre_war_cols = [col for col in eurostat_year_cols if int(col) <= 2021] # Years j
mid_war_cols = eurostat_month_cols.copy() # All months in eurostat_nrg_ti_gasm
```

Calculation of Monthly Averages

The following helper function computes an average over selected columns for a given country and energy product. It filters the input dataframe to import_country == and siec == (default G3000), selects columns, drops missing values, and returns the mean scaled by divide_by (or 0 if no data):

In [9]:
```
def get_avg(df, country, columns, divide_by=1, siec='G3000'):
    vals = df.loc[(df['import_country'] == country) & (df['siec'] == siec), colu
    return vals.mean() / divide_by if len(vals) > 0 else 0
```

Now, we iterate over all countries and call get_avg to compute pre_war (yearly values converted to monthly via divide_by=12) and mid_war (monthly values). Country codes are standardized (country_iso2 via iso2_mapping, country_iso3 via iso2_to_iso3) and change = mid_war - pre_war is added. By comparing the two periods, we can see how much each country's monthly gas imports from Russia have changed in average:

In [10]:
```
fig_2_df = pd.DataFrame([{
    'country_iso2': iso2_mapping[country],
    'country_iso3': iso2_to_iso3[country],
    'pre_war': get_avg(eurostat_nrg_ti_gas, country, pre_war_cols, divide_by=12)
    'mid_war': get_avg(eurostat_nrg_ti_gasm, country, mid_war_cols, divide_by=1)
    }
    for country in eu_countries
])
fig_2_df['change'] = fig_2_df['mid_war'] - fig_2_df['pre_war']
```

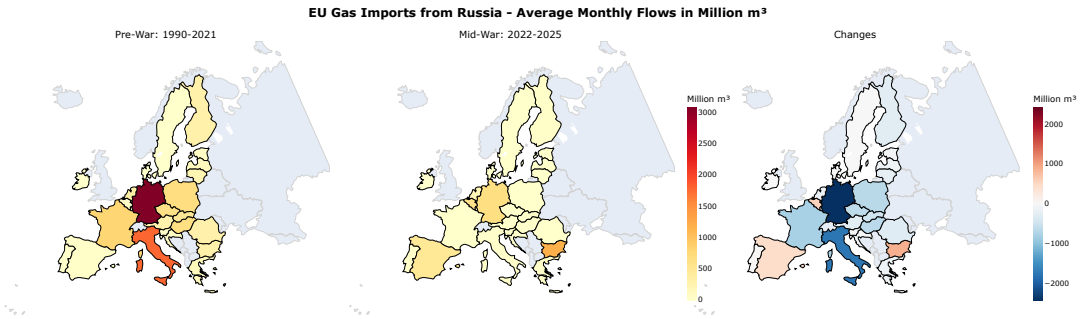
Plotting the Color Map which shows Changes in Russian Gas Imports

Finally, a 1×3 Plotly choropleth subplot is created, which is showing average monthly gas-import flows by European country: pre-war (1990–2021), mid-war (2022–2025), and the difference (mid-war − pre-war). It uses country_iso3 for locations, applies shared geo settings (European scope, centered at 52°N/15°E), and color scales. Each panel has its own colorbar ("Million m^3"), hover shows ISO-2 and the value, and the figure is saved as SVG:

In [11]:
```
# Define plots/subplots: 1 row, 3 columns, each subplot is a separate geo map
fig_2 = make_subplots(
    rows=1, cols=3,
    specs=[[{'type': 'geo'}, {'type': 'geo'}, {'type': 'geo'}]],
    subplot_titles=('Pre-War: 1990-2021', 'Mid-War: 2022-2025', 'Changes'),
    horizontal_spacing=0
)

# Common geographic settings for all three maps (focus on Europe)
geo_settings = dict(
    scope='europe',
    visible=True,
    showcountries=True,
    countrycolor='lightgray',
    bgcolor='white',
    projection_scale=1.3,
    center=dict(lat=52, lon=15)
)

# Build a common color scale for pre- and mid-war time, so comparison is intuiti
# Global min/max across both periods to fix the color range
global_low  = float(fig_2_df[['pre_war','mid_war']].min().min())
global_high = float(fig_2_df[['pre_war','mid_war']].max().max())

# Use ISO3 for locations with Plotly's built-in geography
# Loop over the three subplots: pre-war, mid-war, and the change between them
for i, col, colorscale, title in zip(
    [1, 2, 3],
    ['pre_war', 'mid_war', 'change'],
    ['YlOrRd', 'YlOrRd', 'RdBu'],
    ['Pre-War', 'mid-war', 'Change']
):

    # Define value ranges for the color scales
    zmin = zmax = zmid = None
    if col in ('pre_war', 'mid_war'):         # Same min/max for both periods →
        zmin, zmax = global_low, global_high  # Same color scale for pre- and mi
    else:                                     # For change map: use symmetric ra
        max_change = max(abs(fig_2_df['change'].min()), abs(fig_2_df['change'].m
        zmin, zmax, zmid = -max_change, max_change, 0

    # NEW: custom x-positions for each colorbar (in figure coordinates 0–1)
    colorbar_x_positions = {1: 0.31, 2: 0.70, 3: 1.05}

    fig_2.add_trace(
        go.Choropleth(
            locations=fig_2_df['country_iso3'],  # ISO3 codes for positioning co
            z=fig_2_df[col],                     # Data column for this subplot
```

```
            text=fig_2_df['country_iso2'],       # Shown when hovering over imag
            colorscale=colorscale,
            zmin=zmin, zmax=zmax, zmid=zmid,
            reversescale=(col=='change'),        # Flip RdBu so decreases/increa
            showscale=(i!=1),                    # Hide colorbar for first subpl
            marker_line=dict(color='black', width=0.5),
            colorbar=dict(title='Million m³', x=0.33 + (i-1)*0.335, len=0.7, thi
            hovertemplate=f'<b>%{{text}}</b><br>{title}: %{{z:.2f}}<extra></extr
        ),
        row=1, col=i
    )

# Apply geo settings and configure layout
fig_2.update_geos(geo_settings)
fig_2.update_layout(
    font=dict(family="Verdana", color="black"),
    title=dict(text='EU Gas Imports from Russia - Average Monthly Flows in Milli
               x=0.5, xanchor='center', font=dict(size=20, weight='bold')), heig
               margin=dict(l=10, r=10, t=80, b=10)
)

# Show image
fig_2.show()

# Save figure in respective directory
fig_2.write_image(f'{figs_dir_english}/Vis_2_EU_Gas_Imports_Changes.svg')
```

If you double-click the following image in Jupyter, you will see that it is a markdown cell containing a **static** version of the Plotly figure. This static image is necessary because interactive Plotly visualizations are not included when exporting the notebook to PDF, so the original figure would otherwise not be visible in the exported PDF document:



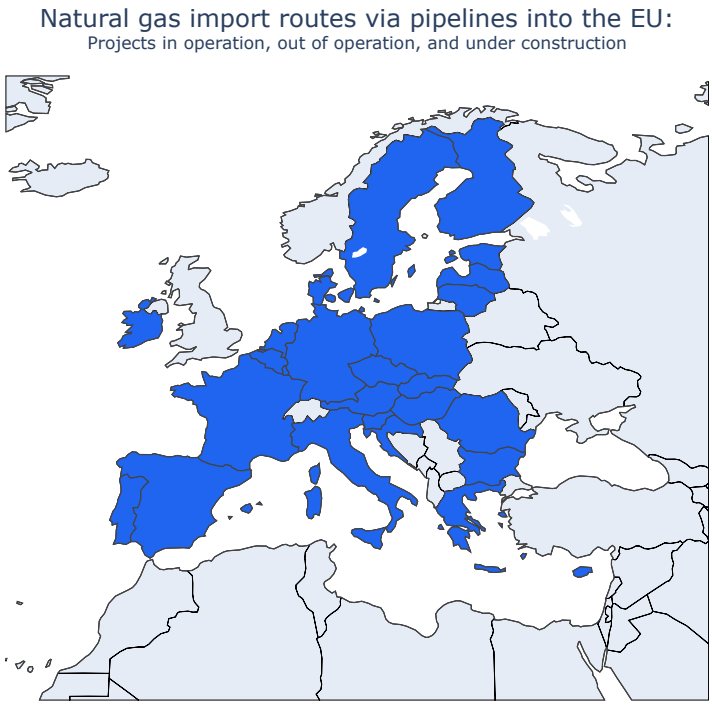
In [ ]:

Visualization 3: Gas Pipelines/Routes into the European Union

For this image, many concepts and code parts from the previous visualization are reused. Teh goal is to plot a plain European map, because other software will be used later, to fill the EU area with the EU flag, the Russian area with Russian flag, and to add arrows to the map which indicate gas pipelines into the EU. Green arrows will indicate pipelines in operation, red arrows indicate pipelines not operating anymore, and purple pipelines are planned projects or pipelines in construction.

In [12]:
```
# EU map with EU-27 marked as blue
# Convert EU country ISO2 codes (eu_countries) to ISO3 by reusing the existing m
eu_iso3 = [iso2_to_iso3[country] for country in eu_countries]

# Create a choropleth where all EU-27 are assigned the same value and colored bl
fig_3 = go.Figure(
    go.Choropleth(
        locations=eu_iso3,   # ISO3 codes for EU-27
        z=[1]*len(eu_iso3),  # Dummy values so all countries are filled
        locationmode="ISO-3",
        colorscale=[[0, "#1F65F0"], [1, "#1F65F0"]],  # Solid blue color for all
        showscale=False
    )
)

# Configure geographic projection and visible region (focus on Europe)
fig_3.update_geos(
    projection_type="natural earth",
    visible=True,
    showcountries=True,
    countrycolor='black',
    countrywidth=0.5,
    bgcolor='white',
    projection_scale=1.3,
    lataxis=dict(range=[17, 80], showgrid=False),   # Limit latitude to Europe r
    lonaxis=dict(range=[-26, 52], showgrid=False),  # Limit longitude to Europe
    showframe=False
)

# Configure layout: title, size, margins, and background
fig_3.update_layout(
    title=dict(text='Natural gas import routes via pipelines into the EU:<br><su
               x=0.5, xanchor='center',font=dict(size=20)),
    height=600, width=900, showlegend=False,
    margin=dict(l=10, r=10, t=80, b=10),
    paper_bgcolor='white',
    plot_bgcolor='white'
)

# Show image
fig_3.show()

# Save figure in respective directory
fig_3.write_image(f'{figs_dir_english}/Vis_3_EU_Gas_Routes_Base.svg')
```

Static image version:



In [ ]:

Visualizations 4.1 and 4.2: Quarterly Pipeline Gas and LNG Imports: Comparison of Time Periods

For the upcoming two visualizations, the DataFrames `gas_reg_quart_imp` and `lng_reg_quart_imp` are used to make bar plots that show the shares of EU gas imports grouped by region of origin. For each group/region there are two bars next to each other: one for the time period **Q1 2019–Q1 2022** and one for **Q2 2022–Q3 2025**. This allows a comparison of how the regional shares have evolved over time, showing from which regions the EU increased or decreased its gas imports pre- and mid-war. `gas_reg_quart_imp` contains pipeline gas imports and `lng_reg_quart_imp` contains LNG imports.

Computation of the regional shares

For each of the two time windows, the helper function `by_window` is applied to the quarterly data:

1. It selects all observations between the given start and end quarter (e.g. from 2019Q1 to 2022Q1).
2. It drops the quarter column and keeps only the numeric columns (the origin regions).
3. It computes, for each region, the **average import volume per quarter** over all quarters in that window (using values.mean()).

This results in one number per region: the average quarterly import for that region in the respective period. The sum over all regions is therefore the **average total import per quarter** in that period.

To obtain the plotted shares, each region's average is divided by this total, so that the bars represent the **share of average quarterly EU gas imports** accounted for by that region in the given time window.

The helper function `by_window` selects all quarters between a start and end period and returns one value per region: either the **average import per quarter** or the **total imports** across all quarters in that window.

In [13]:
```python
def by_window(df, q_start, q_end, *, average=True):

    # Ensure 'quarter' is a quarterly PeriodIndex so comparisons like >= '2019Q1
    if not isinstance(df['quarter'].dtype, pd.PeriodDtype):
        df = df.copy()
        df['quarter'] = pd.PeriodIndex(df['quarter'].astype(str), freq='Q')

    # Select all rows within the requested quarter range (inclusive)
    mask = (df['quarter'] >= pd.Period(q_start, 'Q')) & (df['quarter'] <= pd.Per

    # Drop the quarter column, and keep only numeric region columns
    values = df.loc[mask].drop(columns=['quarter'])

    # Either average per quarter (default) or total across all quarters
    gas_result = values.mean(numeric_only=True) if average else values.sum(numer

    # Sort regions by size (largest first, and return
    return gas_result.sort_values(ascending=False)
```

The helper function `plot_grouped_bars_percent` creates a grouped bar chart. For each region, one bar shows its share in the **pre-war** period and one bar its share in the **mid-war** period. The inputs `pre` and `mid` are Series with one value per region.

In [14]:
```python
def plot_grouped_bars_percent(
    pre, mid, title,
    pre_label='Pre-war (left bars): Q1 2019 - Q1 2022',
    mid_label='Mid-war (right bars): Q2 2022 - Q3 2025',
    y_title='Share of period',
    subtitle=None,
    footer=None,
    highlight_en=("Russia")
):

    # Data preparation
    # Use the union of all region labels from pre and mid
    labels = pre.index.union(mid.index)

    # Reindex so both Series have the same index; missing values become 0
    pre = pre.reindex(labels).fillna(0)
    mid = mid.reindex(labels).fillna(0)

    # Order regions by their pre-period value (fall back to mid if pre is empty)
    order = pre.sort_values(ascending=False).index if pre.sum() else mid.sort_va
    labels = list(order)
    pre = pre.reindex(labels)
    mid = mid.reindex(labels)

    # Base color list (will be overwritten below, kept here for clarity)
```

```
    colors = [('#D52B1E' if l == 'Russia' else "#71797E") for l in labels]

    # Keep values for hover, compute period shares for y
    pre_total = pre.sum()
    mid_total = mid.sum()
    pre_pct = (pre / pre_total) if pre_total else pre*0
    mid_pct = (mid / mid_total) if mid_total else mid*0

    # Show absolute values for hover
    pre_custom  = [[value] for value in pre.values]
    mid_custom = [[value] for value in mid.values]

    fig = go.Figure()

    # Left bars: pre-war shares
    fig.add_trace(go.Bar(
        x=labels, y=pre_pct, name=pre_label,
        marker=dict(color=colors), opacity=0.45,
        customdata=pre_custom,
        hovertemplate='<b>%{x}</b><br>%{customdata[0]:,.0f}<extra></extra>',  #
        # Show percentage labels above the bars
        text=pre_pct,
        texttemplate='%{y:.1%}',
        textposition='outside',
        cliponaxis=False,
        showlegend=False
    ))

    # Right bars: mid-war shares
    fig.add_trace(go.Bar(
        x=labels, y=mid_pct, name=mid_label,
        marker=dict(color=colors), opacity=1.0,
        customdata=mid_custom,
        hovertemplate='<b>%{x}</b><br>%{customdata[0]:,.0f}<extra></extra>',  #
        # Show percentage labels above the bars
        text=mid_pct,
        texttemplate='%{y:.1%}',
        textposition='outside',
        cliponaxis=False,
        showlegend=False
    ))

    # Text-only legend entries (one for each period)
    # Invisible scatter traces are used to get clean legend labels
    fig.add_trace(go.Scatter(
        x=[labels[0]], y=[0],
        mode='lines',
        marker=dict(size=0, color='rgba(0,0,0,0)'),
        name=pre_label,
        hoverinfo='skip',
        showlegend=True,
        legendrank=1,
    ))

    fig.add_trace(go.Scatter(
        x=[labels[0]], y=[0],
        mode='lines',
        marker=dict(size=0, color='rgba(0,0,0,0)'),
        name=mid_label,
        hoverinfo='skip',
```

```
        showlegend=True,
        legendrank=2,
    ))

    # Configure layout
    fig.update_layout(
        font=dict(color='black', family="Verdana", size=13),
        title=dict(text=title, x=0.5, y=0.95, xanchor='center', yanchor='top'),
        barmode='group', bargap=0.25, bargroupgap=0.05,
        autosize=False, width=1000, height=540,
        margin=dict(l=60, r=30, t=120, b=110),
        paper_bgcolor='white', plot_bgcolor='white',
        xaxis=dict(title=''),
        yaxis=dict(title=y_title, tickformat='.0%', dtick=0.1, range=[0, 0.6]),
        legend=dict(orientation='h', x=0.5, xanchor='center', y=-0.15, yanchor='
        uniformtext_minsize=9, uniformtext_mode='hide'
    )

    # Optional subtitle above the main title
    if subtitle:
        fig.add_annotation(
            x=0.477,
            y=1.26,
            xref='paper',
            yref='paper',
            text=subtitle,
            showarrow=False,
            xanchor='center',
            yanchor='top',
            font=dict(size=14),
        )

    # Optional footer annotation below the plot (e.g. totals)
    if footer:
        fig.add_annotation(
            x=0.5, y=-0.3, xref='paper', yref='paper', text=footer,
            showarrow=False, align='center', font=dict(size=13)
        )

    return fig
```

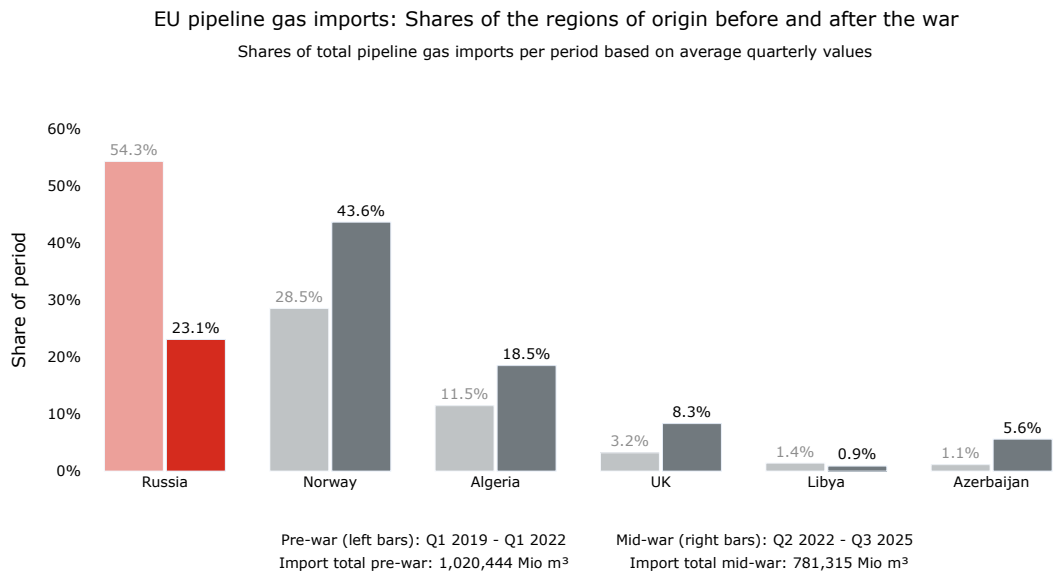
Pipeline gas imports: pre- and mid-war comparison

Let's show pipeline gas imports by region for the two time windows, compute total imports per period, and plot the regional shares using the `plot_grouped_bars_percent` helper.

In [15]:
```
# Pipeline gas: average imports per quarter in each period
gas_pre  = by_window(gas_reg_quart_imp, '2019Q1', '2022Q1')
gas_mid = by_window(gas_reg_quart_imp, '2022Q2', '2025Q3')

# Totals for subtitle/footer: sum across all quarters and regions in each period
gas_tot_pre  = by_window(gas_reg_quart_imp, '2019Q1', '2022Q1', average=False).s
gas_tot_mid = by_window(gas_reg_quart_imp, '2022Q2', '2025Q3', average=False).su

# Short description for the subtitle
gas_subtitle = 'Shares of total pipeline gas imports per period based on average

# Footer text with total import volumes per period (in million m³)
```

```
gas_footer = (
    f'      Import total pre-war: {gas_tot_pre:,.0f} Mio m³'
    f'            Import total mid-war: {gas_tot_mid:,.0f} Mio m³'
)

# Create the grouped bar chart for pipeline gas
fig_4_1 = plot_grouped_bars_percent(
    gas_pre, gas_mid,
    title='EU pipeline gas imports: Shares of the regions of origin before and a
    subtitle=gas_subtitle,
    footer=gas_footer,
    highlight_en=("Russia",)
)

# Show image
fig_4_1.show()

# Save figure in respective directory
fig_4_1.write_image(f'{figs_dir_english}/Vis_4_1_EU_Quart_Pipeline_Gas_Imports.s
```

Static image version:



In [ ]:

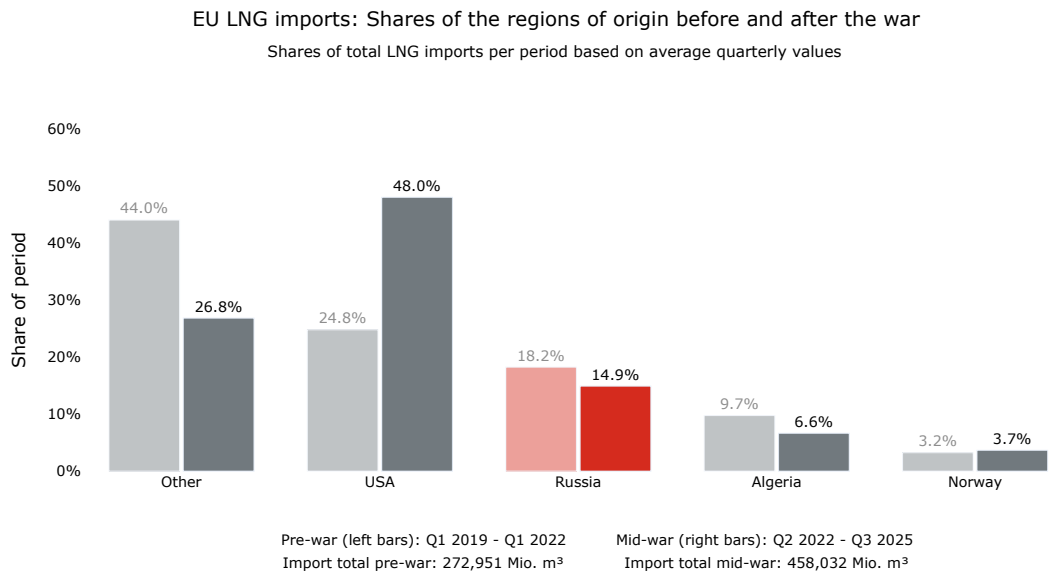
LNG imports: pre- and mid-war comparison

The same calculation as for pipeline gas imports is now repeated for LNG imports. Again, the average quarterly import shares by region are calculated, for the time periods before and after the war.

In [16]:
```
# LNG gas: average imports per quarter in each period
lng_pre  = by_window(lng_reg_quart_imp, '2019Q1', '2022Q1')
lng_mid = by_window(lng_reg_quart_imp, '2022Q2', '2025Q3')

# Totals for subtitle/footer: sum across all quarters and regions in each period
lng_tot_pre  = by_window(lng_reg_quart_imp, '2019Q1', '2022Q1', average=False).s
lng_tot_mid = by_window(lng_reg_quart_imp, '2022Q2', '2025Q3', average=False).su

# Short description for the subtitle
```

```
lng_subtitle = 'Shares of total LNG imports per period based on average quarterl

# Footer text with total import volumes per period (in million m³)
lng_footer = (
    f'       Import total pre-war: {lng_tot_pre:,.0f} Mio. m³'
    f'            Import total mid-war: {lng_tot_mid:,.0f} Mio. m³'
)

# Create the grouped bar chart for LNG imports
fig_4_2 = plot_grouped_bars_percent(
    lng_pre, lng_mid,
    title='EU LNG imports: Shares of the regions of origin before and after the
    subtitle=lng_subtitle,
    footer=lng_footer,
    highlight_en=("Russia",)
)

# Show image
fig_4_2.show()

# Save figure in respective directory
fig_4_2.write_image(f'{figs_dir_english}/Vis_4_2_EU_Quart_LNG_Imports.svg')
```

Static image version:



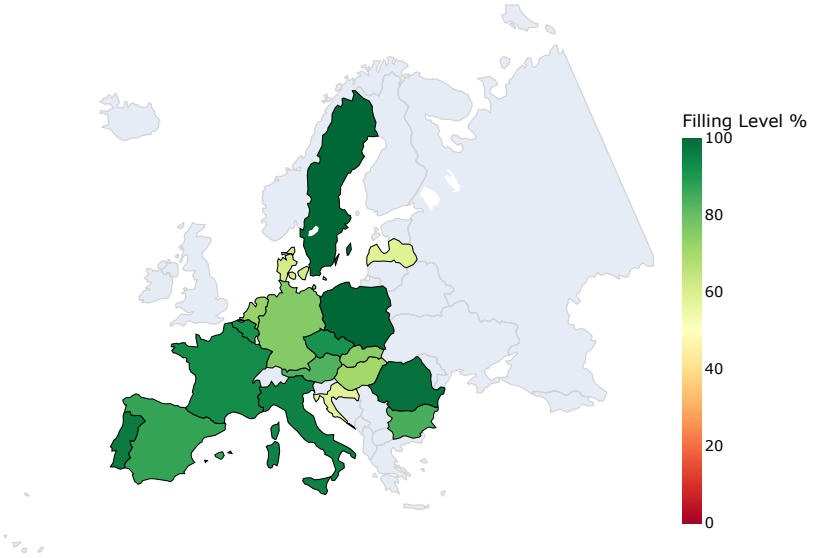
In [ ]:

Visualization 5: Latest Gas Filling Levels of Selected EU Countries

The gas storage filling levels are assigned to the variable `stored in gas_storage_eu` (see begining of the notebook). The following code cell maps these filling levels of selected EU countries on a choropleth map. For visualization 2, iso2_to_iso3 were already used to map two-letter country codes to ISO-3 codes, and this concept is reused here. The filling percentage is shown with a red–yellow–green color scale, focused on Europe, and saved as an SVG file.

In [17]:
```python
# Reuse iso2_to_iso3 from first code cell of Visualization 2 to map countries
gas_storage_eu['iso3'] = gas_storage_eu['country_code'].str.upper().map(iso2_to_

# Create choropleth figure showing gas storage filling levels by country
fig_5 = go.Figure(
    go.Choropleth(
        locations=gas_storage_eu['iso3'],
        z=gas_storage_eu['filling_%'].astype(float),
        text=gas_storage_eu['country_code'],
        locationmode='ISO-3',
        colorscale='RdYlGn',
        zmin=0, zmax=100,
        marker_line=dict(color='black', width=0.5),
        colorbar=dict(title='Filling Level %', len=0.7, thickness=15, x=0.82),
        hovertemplate='<b>%{text}</b><br>Filling Level: %{z:.1f}%<extra></extra>
    )
)

# Adjust geographic projection and focus on Europe
fig_5.update_geos(
    scope='europe',
    visible=True,
    showcountries=True,
    countrycolor='lightgray',
    bgcolor='white',
    projection_scale=1.3,         # Zoom level
    center=dict(lat=52, lon=15),  # Map center (roughly central Europe)
)

# Configure layout
fig_5.update_layout(
    font=dict(family='Verdana', color='black'),
    title=dict(
        text='EU Gas Storage Filling Levels in Percent<br><sup>As of October 25,
        x=0.5, xanchor='center',
        font=dict(size=20)
    ),
    height=600,
    width=900,
    showlegend=False,
    margin=dict(l=10, r=10, t=80, b=10),
    paper_bgcolor='white',
    plot_bgcolor='white'
)

# Show image
fig_5.show()

# Save figure in respective directory
fig_5.write_image(f'{figs_dir_english}/Vis_5_EU_Gas_Storage_Filling_Levels.svg')
```

Static image version:


EU Gas Storage Filling Levels in Percent
As of October 25, 2025
Filling Level %
100
80
60
40
20
0

In [ ]: