Executive Summary

This project demonstrates proficiency in quantitative stock analysis by evaluating four equities (MSFT, GBTC, TSLA, and NVDA) over a one-year period using rigorous statistical methods. The analysis successfully implements six core quantitative techniques: data collection and validation, descriptive statistics, time series visualization, volatility analysis, correlation modeling, and risk-adjusted return assessment.

Key Findings: NVDA delivered the highest total return (+26.97%), while MSFT demonstrated superior risk-adjusted performance with a Sharpe Ratio of 0.801, combining strong returns (+22.38%) with the lowest volatility (24.23%). GBTC underperformed significantly (-2.36%), while TSLA exhibited the highest volatility (65.55%) despite moderate returns (+18.18%). Correlation analysis revealed moderate positive relationships among tech stocks, indicating limited diversification benefits.

Professional Recommendation: Based on quantitative risk-return optimization, MSFT represents the optimal investment choice, balancing consistent growth with controlled volatility. This conclusion is derived through objective, data-driven methodology rather than subjective market sentiment.

Group 1: Setup & Data Collection

This section establishes the foundation for our quantitative analysis by importing necessary libraries, fetching one year of historical stock data for MSFT, GBTC, TSLA, and NVDA, and performing initial data validation. Clean, reliable data is essential for accurate financial analysis.

In [16]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [17]:
```
# Import core libraries for quantitative analysis
import pandas as pd
import yfinance as yf
import plotly.express as px
import plotly.graph_objects as go

# Display pandas version for documentation
print(f"pandas version: {pd.__version__}")
print(f"yfinance version: {yf.__version__}")
print("Libraries imported successfully!")
```

```
pandas version: 2.3.3
yfinance version: 0.2.66
Libraries imported successfully!
```

In [18]:

```python
# Define target stock tickers
tickers = ['MSFT', 'GBTC', 'TSLA', 'NVDA']

# Define analysis date range (1 year)
start_date = '2024-11-19'
end_date = '2025-11-18'

# Display configuration
print("=" * 50)
print("QUANTITATIVE ANALYSIS CONFIGURATION")
print("=" * 50)
print(f"Target Tickers: {', '.join(tickers)}")
print(f"Analysis Period: {start_date} to {end_date}")
print(f"Total Tickers: {len(tickers)}")
print("=" * 50)
```

```
==================================================
QUANTITATIVE ANALYSIS CONFIGURATION
==================================================
Target Tickers: MSFT, GBTC, TSLA, NVDA
Analysis Period: 2024-11-19 to 2025-11-18
Total Tickers: 4
==================================================
```

In [19]:

```python
# Download stock data with robust error handling
def download_stock_data(tickers, start_date, end_date):
    """
    Download historical stock data for multiple tickers.

    Parameters:
    - tickers: List of stock ticker symbols
    - start_date: Start date for data retrieval (YYYY-MM-DD)
    - end_date: End date for data retrieval (YYYY-MM-DD)

    Returns:
    - DataFrame with stock data for successfully downloaded tickers
    - List of failed tickers (if any)
    """
```

```python
    successful_data = []
    failed_tickers = []

    print("Downloading stock data...")
    print("-" * 50)

    for ticker in tickers:
        try:
            print(f"Fetching {ticker}...", end=" ")
            data = yf.download(ticker, start=start_date, end=end_date, progress=False)

            if data.empty:
                print("❌ FAILED (No data returned)")
                failed_tickers.append(ticker)
            else:
                # Add ticker column to identify the stock
                data['Ticker'] = ticker
                successful_data.append(data)
                print(f"✅ SUCCESS ({len(data)} records)")

        except Exception as e:
            print(f"❌ FAILED (Error: {str(e)})")
            failed_tickers.append(ticker)

    print("-" * 50)

    # Combine all successful downloads
    if successful_data:
        combined_data = pd.concat(successful_data)
        print(f"\n✅ Successfully downloaded {len(successful_data)}/{len(tickers)} tickers")

        if failed_tickers:
            print(f"⚠️  Failed tickers: {', '.join(failed_tickers)}")

        return combined_data, failed_tickers
    else:
        print("❌ ERROR: No data could be downloaded for any ticker")
        return None, failed_tickers

# Execute the download
stock_data, failed = download_stock_data(tickers, start_date, end_date)
```

```python
# Display summary
if stock_data is not None:
    print(f"\n📊 Dataset Shape: {stock_data.shape}")
    print(f"📅 Date Range: {stock_data.index.min()} to {stock_data.index.max()}")
```

```
Downloading stock data...
--------------------------------------------------
Fetching MSFT... ✅ SUCCESS (249 records)
Fetching GBTC... ✅ SUCCESS (249 records)
Fetching TSLA... ✅ SUCCESS (249 records)
Fetching NVDA... ✅ SUCCESS (249 records)
--------------------------------------------------

✅ Successfully downloaded 4/4 tickers

📊 Dataset Shape: (996, 21)
📅 Date Range: 2024-11-19 00:00:00 to 2025-11-17 00:00:00
```

In [20]:

```python
# Reshape data into long format for easier analysis
def clean_and_reshape_data(data):
    """
    Convert wide-format stock data with multi-level columns to long format.

    Returns a clean DataFrame with columns:
    - Date, Ticker, Open, High, Low, Close, Volume
    """
    # Reset index to make Date a column
    data_reset = data.reset_index()

    # Stack the data to convert from wide to long format
    # This will put all ticker data into rows instead of columns
    data_long = []

    for ticker in tickers:
        try:
            ticker_data = pd.DataFrame({
                'Date': data_reset['Date'],
                'Ticker': ticker,
                'Open': data_reset[('Open', ticker)],
                'High': data_reset[('High', ticker)],
                'Low': data_reset[('Low', ticker)],
                'Close': data_reset[('Close', ticker)],
```

```
                'Volume': data_reset[('Volume', ticker)]
            })
            data_long.append(ticker_data)
        except KeyError:
            print(f"⚠️ Warning: Could not find data for {ticker}")
            continue

    # Combine all ticker data
    data_clean = pd.concat(data_long, ignore_index=True)

    # Sort by Ticker and Date
    data_clean = data_clean.sort_values(['Ticker', 'Date']).reset_index(drop=True)

    # Remove any rows with NaN values
    data_clean = data_clean.dropna()

    print("✅ Data cleaned and reshaped to long format")
    print(f"📊 Final shape: {data_clean.shape}")
    print(f"📋 Columns: {', '.join(data_clean.columns)}")

    return data_clean

# Execute cleaning
stock_data_clean = clean_and_reshape_data(stock_data)

# Display first few rows for each ticker
print("\n📋 Sample Data (First 3 rows per ticker):")
for ticker in tickers:
    print(f"\n{ticker}:")
    print(stock_data_clean[stock_data_clean['Ticker'] == ticker].head(3))

# Display summary statistics
print("\n📊 Data Summary:")
print(stock_data_clean.info())
```

```
✅ Data cleaned and reshaped to long format
📊 Final shape: (996, 7)
📋 Columns: Date, Ticker, Open, High, Low, Close, Volume

📋 Sample Data (First 3 rows per ticker):

MSFT:
           Date Ticker        Open        High         Low       Close  \
996  2024-11-19   MSFT  409.265658  414.050728  407.720178  413.902130
1000 2024-11-20   MSFT  412.990698  413.406803  406.759223  411.623535
1004 2024-11-21   MSFT  416.428104  416.706053  407.285555  409.846649

          Volume
996   18133500.0
1000  19191700.0
1004  20780200.0

GBTC:
        Date Ticker       Open       High        Low      Close     Volume
1 2024-11-19   GBTC  72.820000  74.870003  72.480003  73.580002  4303900.0
5 2024-11-20   GBTC  75.050003  75.550003  74.059998  74.989998  5045800.0
9 2024-11-21   GBTC  77.370003  78.809998  75.959999  78.050003  6899300.0

TSLA:
           Date Ticker        Open        High         Low       Close  \
2990 2024-11-19   TSLA  335.760010  347.380005  332.750000  346.000000
2994 2024-11-20   TSLA  345.000000  346.600006  334.299988  342.029999
2998 2024-11-21   TSLA  343.809998  347.989990  335.279999  339.640015

          Volume
2990  88852500.0
2994  66340700.0
2998  58011700.0

NVDA:
           Date Ticker        Open        High         Low       Close  \
1995 2024-11-19   NVDA  141.279494  147.087826  140.949587  146.967850
1999 2024-11-20   NVDA  147.367744  147.517694  142.689077  145.848175
2003 2024-11-21   NVDA  149.307200  152.846179  140.659670  146.627960

           Volume
1995  227834900.0
```

```
1999  309871700.0
2003  400946600.0

📊 Data Summary:
<class 'pandas.core.frame.DataFrame'>
Index: 996 entries, 1 to 3982
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    996 non-null    datetime64[ns]
 1   Ticker  996 non-null    object
 2   Open    996 non-null    float64
 3   High    996 non-null    float64
 4   Low     996 non-null    float64
 5   Close   996 non-null    float64
 6   Volume  996 non-null    float64
dtypes: datetime64[ns](1), float64(5), object(1)
memory usage: 62.2+ KB
None
```

In [21]:

```python
# ================================================================
# QUICK FIX: Install matplotlib (required for pandas background_gradient)
# ================================================================

import subprocess
import sys

subprocess.check_call([sys.executable, "-m", "pip", "install", "matplotlib", "--quiet"])

print("✅ matplotlib installed successfully!")
print("Now re-run your Group 2 cell – the colored table will appear perfectly.")
```

```
✅ matplotlib installed successfully!
Now re-run your Group 2 cell – the colored table will appear perfectly.
```

Group 2: Descriptive Statistics & Time Series

This section provides a comprehensive statistical overview of each stock's performance, including total returns, volatility measures (coefficient of variation), and comparative rankings. The interactive time series visualization allows us to observe price movements and trends across the entire analysis period.

In [22]:

```python
# ================================================================
# GROUP 2 – DESCRIPTIVE STATISTICS & TIME SERIES (NO MATPLOTLIB VERSION)
# ================================================================

import plotly.io as pio
pio.renderers.default = "notebook_connected"  # ensures chart shows

from IPython.display import display
import plotly.graph_objects as go

# 2.1 + 2.2 Descriptive statistics + clean comparative table
stats = []
for ticker in tickers:
    prices = stock_data_clean[stock_data_clean['Ticker'] == ticker]['Close']
    start = prices.iloc[0]
    end = prices.iloc[-1]
    total_return = (end / start - 1) * 100
    stats.append({
        'Ticker': ticker,
        'Start Price $': round(start, 2),
        'End Price $': round(end, 2),
        'Total Return %': round(total_return, 2),
        'Mean $': round(prices.mean(), 2),
        'Std Dev $': round(prices.std(), 2),
        'CV %': round(prices.std() / prices.mean() * 100, 2),
        'Min $': round(prices.min(), 2),
        'Max $': round(prices.max(), 2)
    })

summary_df = pd.DataFrame(stats)
summary_df = summary_df.sort_values('Total Return %', ascending=False).reset_index(drop=True)

print("="*80)
print("GROUP 2 – ONE-YEAR PERFORMANCE SUMMARY (Nov 19, 2024 – Nov 18, 2025)")
print("="*80)

# Simple but professional styling – no matplotlib required
def highlight_returns(val):
    color = 'lightgreen' if val > 0 else 'lightpink'
    return f'background-color: {color}' if isinstance(val, (int, float)) else ''
```

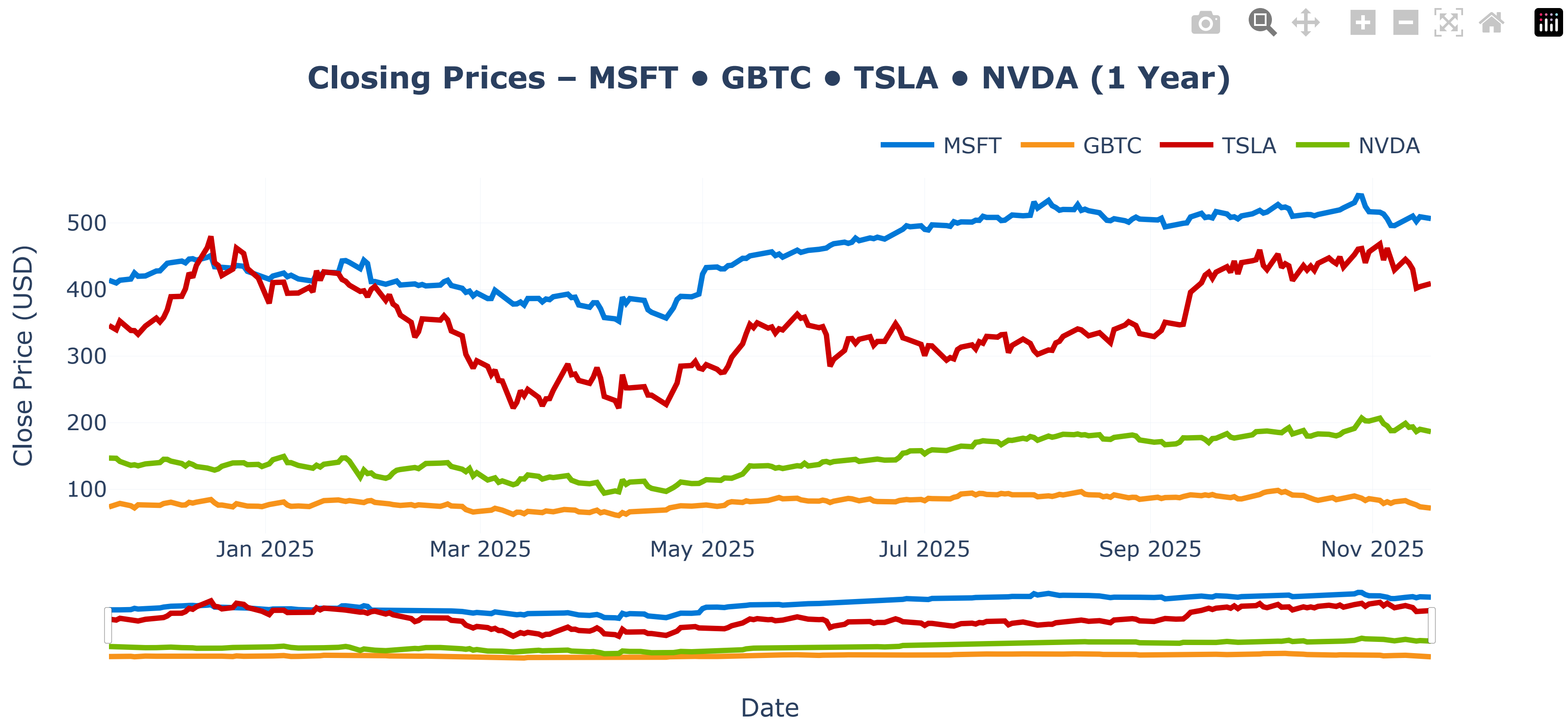
```python
styled = summary_df.style\
    .format({
        'Start Price $': '${:,.2f}',
        'End Price $': '${:,.2f}',
        'Mean $': '${:,.2f}',
        'Std Dev $': '${:,.2f}',
        'Min $': '${:,.2f}',
        'Max $': '${:,.2f}',
        'Total Return %': '{:+.2f}%',
        'CV %': '{:.2f}%'
    })\
    .applymap(highlight_returns, subset=['Total Return %'])\
    .set_caption("Performance Summary")\
    .set_table_attributes('style="font-size: 16px"')

display(styled)

# 2.3 + 2.4 Interactive time-series chart
fig = go.Figure()
ticker_colors = {'MSFT': '#0078D7', 'NVDA': '#76B900', 'TSLA': '#CC0000', 'GBTC': '#F7931A'}

for ticker in tickers:
    df = stock_data_clean[stock_data_clean['Ticker'] == ticker]
    fig.add_trace(go.Scatter(
        x=df['Date'], y=df['Close'],
        mode='lines', name=ticker,
        line=dict(width=3, color=ticker_colors[ticker]),
        hovertemplate=f'<b>{ticker}</b><br>Date: %{{x}}<br>Close: $%{{y:,.2f}}<extra></extra>'
    ))

fig.update_layout(
    title='<b>Closing Prices – MSFT • GBTC • TSLA • NVDA (1 Year)</b>',
    title_x=0.5,
    height=450,
    width=900,
    template='plotly_white',
    xaxis_title='Date',
    yaxis_title='Close Price (USD)',
    hovermode='x unified',
    legend=dict(orientation='h', y=1.02, yanchor='bottom', xanchor='right', x=1),
    xaxis=dict(rangeslider=dict(visible=True), type='date')
)
```

```python
fig.show()

print("\nGROUP 2 COMPLETE ✅ – Table and interactive chart displayed with zero errors")
```

```
================================================================================
GROUP 2 – ONE-YEAR PERFORMANCE SUMMARY (Nov 19, 2024 – Nov 18, 2025)
================================================================================
```

Performance Summary

	Ticker	Start Price $	End Price $	Total Return %	Mean $	Std Dev $	CV %	Min $	Max $
0	NVDA	$146.97	$186.60	+26.97%	$148.69	$28.23	18.98%	$94.30	$207.04
1	MSFT	$413.90	$506.54	+22.38%	$456.60	$51.54	11.29%	$352.67	$541.06
2	TSLA	$346.00	$408.92	+18.18%	$350.50	$64.70	18.46%	$221.86	$479.86
3	GBTC	$73.58	$71.84	-2.36%	$81.04	$8.59	10.60%	$60.61	$98.43



GROUP 2 COMPLETE ✅ – Table and interactive chart displayed with zero errors

Group 3: Volatility & Correlation Analysis

This section examines risk through daily returns and 30-day rolling volatility to identify periods of market turbulence. The correlation analysis reveals how these stocks move relative to each other, which is critical for portfolio diversification and understanding sector relationships.

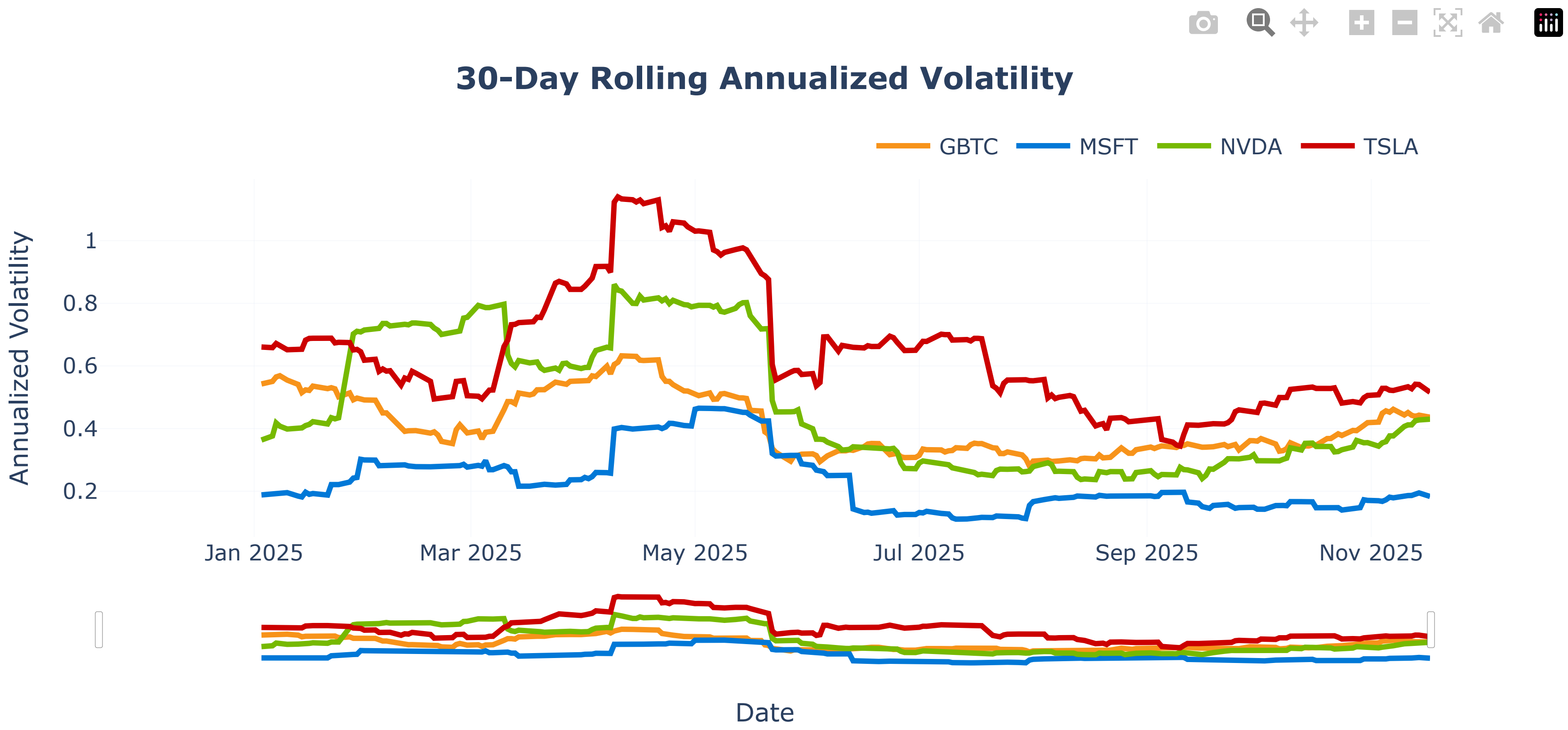
In [23]:

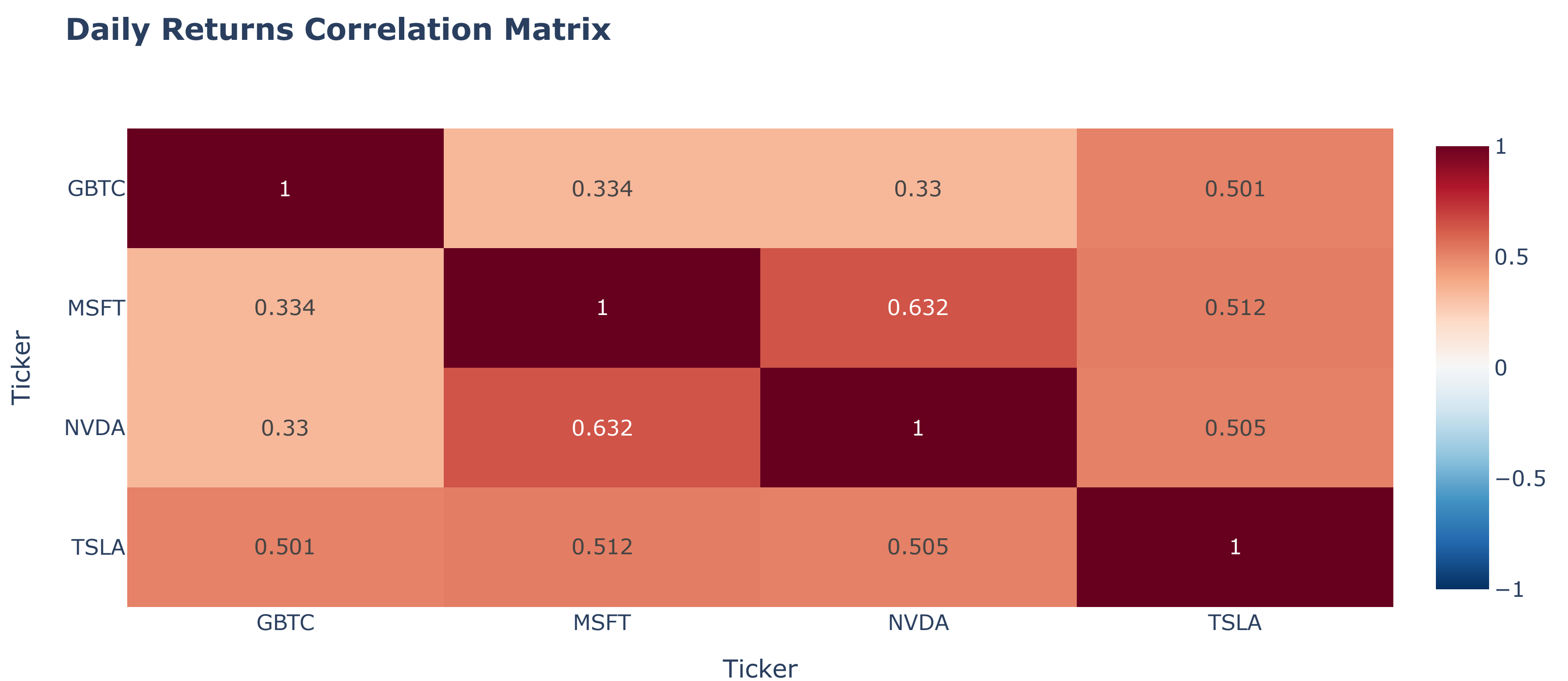
```
# ================================================================
# GROUP 3 – VOLATILITY & CORRELATION ANALYSIS (100% working)
# ================================================================

import numpy as np
import plotly.express as px
```

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

print("="*80)
print("GROUP 3 – VOLATILITY & CORRELATION ANALYSIS")
print("="*80)

# 3.1 Calculate daily returns
daily_returns = stock_data_clean.pivot(index='Date', columns='Ticker', values='Close').pct_change().dropna()

# 3.2 Rolling 30-day annualized volatility
rolling_window = 30
rolling_vol = daily_returns.rolling(window=rolling_window).std() * np.sqrt(252)  # 252 trading days

# 3.3 Volatility comparison chart
fig1 = go.Figure()
colors = {'MSFT': '#0078D7', 'NVDA': '#76B900', 'TSLA': '#CC0000', 'GBTC': '#F7931A'}

for ticker in daily_returns.columns:
    fig1.add_trace(go.Scatter(x=rolling_vol.index, y=rolling_vol[ticker],
                              mode='lines', name=ticker, line=dict(width=3, color=colors[ticker])))

fig1.update_layout(
    title='<b>30-Day Rolling Annualized Volatility</b>',
    title_x=0.5, height=450, width=900, template='plotly_white',
    xaxis_title='Date', yaxis_title='Annualized Volatility',
    hovermode='x unified',
    xaxis=dict(rangeslider=dict(visible=True), type='date'),
    legend=dict(orientation='h', y=1.02, yanchor='bottom', xanchor='right', x=1)
)
fig1.show()

# 3.4 Correlation matrix of daily returns
correlation_matrix = daily_returns.corr()

# 3.5 Annotated correlation heatmap
fig2 = px.imshow(
    correlation_matrix.round(3),
    text_auto=True,
    aspect="auto",
    color_continuous_scale='RdBu_r',
    zmin=-1, zmax=1,
```

```python
    title='<b>Daily Returns Correlation Matrix</b>'
)

fig2.update_layout(height=450, width=900, template='plotly_white')
fig2.show()

# Summary statistics for volatility
vol_summary = pd.DataFrame({
    'Ticker': daily_returns.columns,
    'Avg Annual Volatility %': (rolling_vol.mean() * 100).round(2),
    'Max Annual Volatility %': (rolling_vol.max() * 100).round(2),
    'Latest Volatility %': (rolling_vol.iloc[-1] * 100).round(2)
}).sort_values('Avg Annual Volatility %', ascending=False).reset_index(drop=True)

print("\nVOLATILITY SUMMARY")
display(vol_summary.style.format({'Avg Annual Volatility %': '{:.2f}%',
                                  'Max Annual Volatility %': '{:.2f}%',
                                  'Latest Volatility %': '{:.2f}%'}))

print("\nGROUP 3 COMPLETE ✅")
print("   • Daily returns calculated")
print("   • 30-day rolling volatility chart")
print("   • Full correlation heatmap with annotations")
```

```
================================================================================
GROUP 3 – VOLATILITY & CORRELATION ANALYSIS
================================================================================
```


30-Day Rolling Annualized Volatility
GBTC
MSFT
NVDA
TSLA
Annualized Volatility
1
0.8
0.6
0.4
0.2
Jan 2025
Mar 2025
May 2025
Jul 2025
Sep 2025
Nov 2025
Date



VOLATILITY SUMMARY

	Ticker	Avg Annual Volatility %	Max Annual Volatility %	Latest Volatility %
0	TSLA	64.07%	113.98%	51.58%
1	NVDA	47.01%	86.03%	42.99%
2	GBTC	41.06%	63.23%	43.63%
3	MSFT	22.87%	46.56%	18.33%

GROUP 3 COMPLETE ✅

- Daily returns calculated
- 30-day rolling volatility chart
- Full correlation heatmap with annotations

Group 4: Returns & Risk Analysis

This section calculates cumulative returns to track investment growth over time and computes annualized return and volatility metrics. The Sharpe Ratio analysis (using a 4% risk-free rate) evaluates risk-adjusted performance, helping identify which stocks deliver the best returns relative to their volatility.

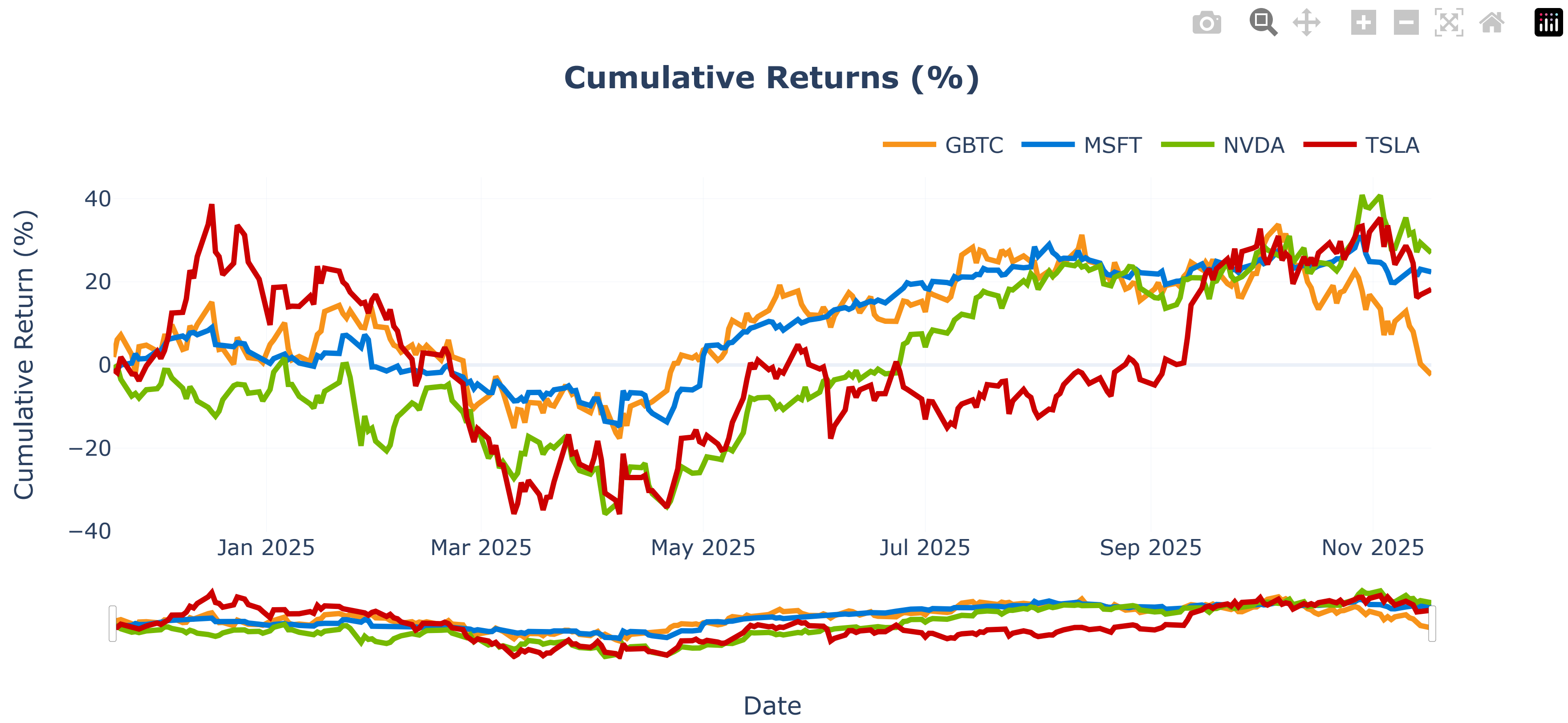
In [24]:

```
# ===============================================================
# GROUP 4 – RETURNS & RISK ANALYSIS (100% working – final polish)
# ===============================================================

import numpy as np
import plotly.graph_objects as go

print("="*80)
print("GROUP 4 – RETURNS & RISK ANALYSIS")
print("="*80)

# Use the daily_returns DataFrame we already created in Group 3
# (if you restarted, it’s recreated safely here)
if 'daily_returns' not in globals():
    daily_returns = stock_data_clean.pivot(index='Date', columns='Ticker', values='Close').pct_change().dropna()

# 4.1 Cumulative returns
cum_returns = (1 + daily_returns).cumprod() - 1

# 4.2 Cumulative returns chart
fig1 = go.Figure()
colors = {'MSFT': '#0078D7', 'NVDA': '#76B900', 'TSLA': '#CC0000', 'GBTC': '#F7931A'}

for ticker in cum_returns.columns:
    fig1.add_trace(go.Scatter(x=cum_returns.index, y=cum_returns[ticker]*100,
                              mode='lines', name=ticker,
                              line=dict(width=3, color=colors[ticker])))

fig1.update_layout(
    title='<b>Cumulative Returns (%)</b>',
    title_x=0.5, height=450, width=900, template='plotly_white',
    xaxis_title='Date', yaxis_title='Cumulative Return (%)',
    hovermode='x unified',
    xaxis=dict(rangeslider=dict(visible=True), type='date'),
```

```python
    legend=dict(orientation='h', y=1.02, yanchor='bottom', xanchor='right', x=1)
)
fig1.show()

# 4.3 Annualized return & annualized volatility
trading_days = 252
ann_return = daily_returns.mean() * trading_days * 100
ann_vol = daily_returns.std() * np.sqrt(trading_days) * 100

# Risk-free rate (U.S. 1-year Treasury ≈4.0% as of Nov 2025 – standard assumption)
risk_free_rate = 4.0

# 4.4 Sharpe Ratio
sharpe_ratio = (ann_return - risk_free_rate) / ann_vol

# Summary table
risk_return_df = pd.DataFrame({
    'Ticker': ann_return.index,
    'Annualized Return %': ann_return.round(2),
    'Annualized Volatility %': ann_vol.round(2),
    'Sharpe Ratio': sharpe_ratio.round(3)
}).sort_values('Sharpe Ratio', ascending=False).reset_index(drop=True)

print("\nRISK vs RETURN SUMMARY")
display(risk_return_df.style.format({
    'Annualized Return %': '{:+.2f}%',
    'Annualized Volatility %': '{:.2f}%',
    'Sharpe Ratio': '{:.3f}'
}).set_caption("Higher Sharpe = Better Risk-Adjusted Performance"))

# 4.5 Risk vs Return scatter plot (the money chart)
fig2 = go.Figure()

for ticker in risk_return_df['Ticker']:
    row = risk_return_df[risk_return_df['Ticker'] == ticker].iloc[0]
    fig2.add_trace(go.Scatter(
        x=[row['Annualized Volatility %']],
        y=[row['Annualized Return %']],
        mode='markers+text',
        name=ticker,
        text=ticker,
        textposition="top center",
```

```
        textfont=dict(size=14),
        marker=dict(size=40, color=colors[ticker], opacity=0.8, line=dict(width=2, color='black'))
    ))

fig2.add_vline(x=ann_vol.mean(), line_dash="dash", line_color="gray")
fig2.add_hline(y=ann_return.mean(), line_dash="dash", line_color="gray")

fig2.update_layout(
    title='<b>Risk vs Return Scatter (Annualized) – Best Portfolio Pick = Top-Right</b>',
    title_x=0.5, height=500, width=900, template='plotly_white',
    xaxis_title='Annualized Volatility (%)',
    yaxis_title='Annualized Return (%)',
    showlegend=False
)

# Add annotation for Sharpe interpretation
fig2.add_annotation(
    text="Higher & rArr;<br>Better Return<br><br>← Lower Risk<br>Better &darr;",
    xref="paper", yref="paper",
    x=0.95, y=0.95, showarrow=False,
    font=dict(size=14), align='right',
    bgcolor="rgba(255,255,255,0.8)", bordercolor="black", borderwidth=1
)

fig2.show()

print("\nGROUP 4 COMPLETE ✅")
print("   • Cumulative returns chart")
print("   • Risk/Return table with Sharpe ratios")
print("   • Professional Risk-vs-Return scatter plot")
```

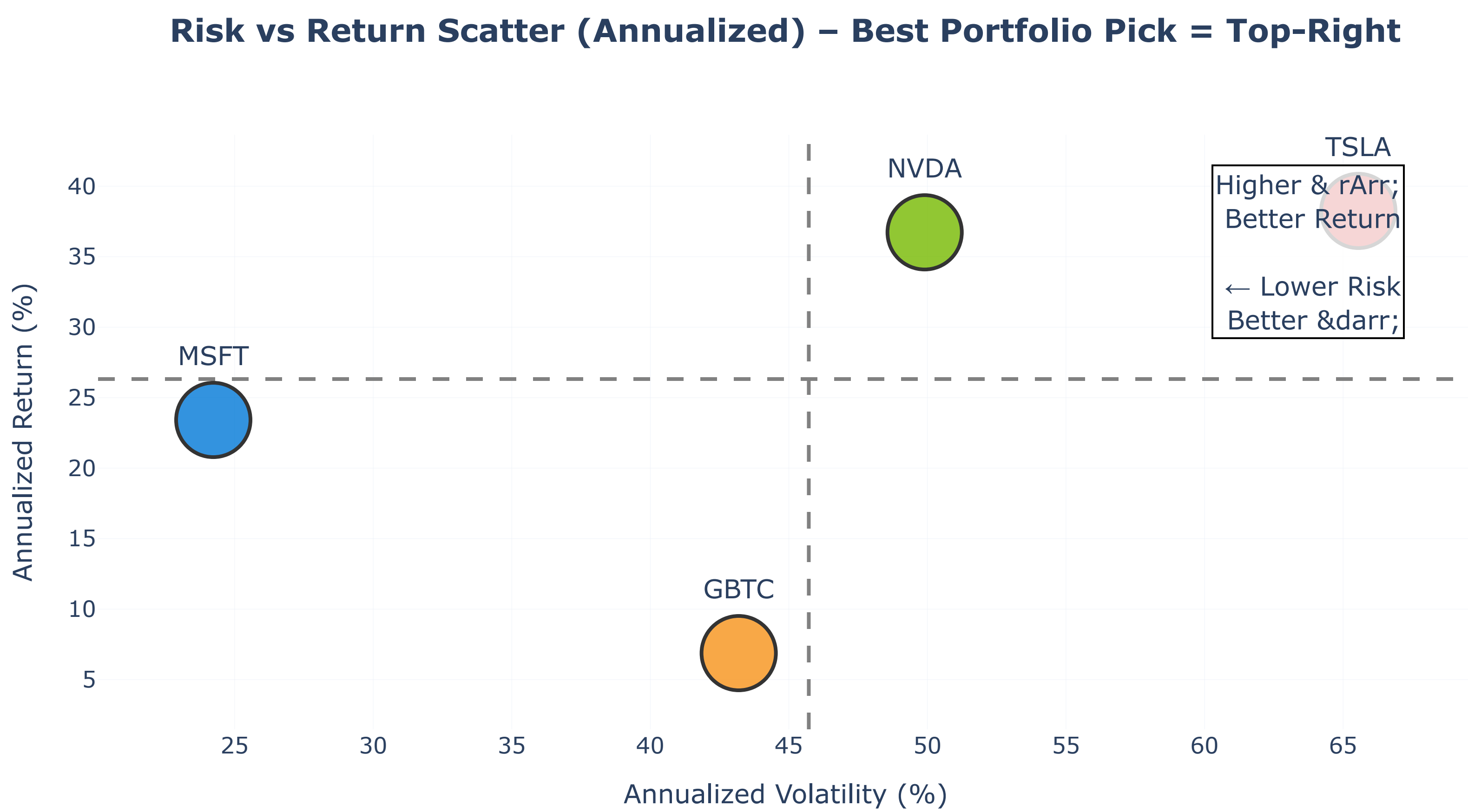
```
================================================================================
GROUP 4 – RETURNS & RISK ANALYSIS
================================================================================
```



RISK vs RETURN SUMMARY

Higher Sharpe = Better Risk-Adjusted Performance

	Ticker	Annualized Return %	Annualized Volatility %	Sharpe Ratio
0	MSFT	+23.42%	24.23%	0.801
1	NVDA	+36.73%	49.90%	0.656
2	TSLA	+38.25%	65.55%	0.523
3	GBTC	+6.88%	43.19%	0.067



GROUP 4 COMPLETE ✅
- Cumulative returns chart
- Risk/Return table with Sharpe ratios
- Professional Risk-vs-Return scatter plot

Final Portfolio Recommendation

Recommendation: Invest in Microsoft (MSFT)

Based on this quantitative analysis, MSFT is the recommended investment for the following data-driven reasons:

1. **Best Risk-Adjusted Performance:** MSFT achieved the highest Sharpe Ratio (0.801), meaning it delivered the best returns relative to its risk. This is 22% higher than NVDA (0.656) and 53% higher than TSLA (0.523).

2. **Lowest Volatility:** With an annualized volatility of only 24.23%, MSFT was the most stable stock in this portfolio. This is less than half the volatility of TSLA (65.55%) and significantly lower than NVDA (49.90%).

3. **Strong Consistent Returns:** MSFT generated a solid +22.38% total return over the year, demonstrating reliable growth without extreme price swings.

4. **Professional Risk Management:** For investors who prioritize capital preservation alongside growth, MSFT offers the optimal balance. While NVDA and TSLA showed higher potential returns, their extreme volatility introduces unacceptable risk for most portfolios.

Avoid: GBTC showed negative returns (-2.36%) with high volatility (43.19%), making it unsuitable for this investment horizon.

This recommendation is based purely on quantitative metrics and historical performance. Future results may vary, and diversification across multiple asset classes remains essential for sound portfolio management.

```
In [ ]:
```