Tech GC EDA - Understand your Data

The goal of this tutorial is to provide you with a comprehensive understanding of different techniques used to explore and analyze time-series data, which is crucial for building effective forecasting models.

We will cover a wide range of EDA techniques, including:

- **Missing Value Analysis:** Understanding the extent and pattern of missing data in the dataset.
- **Correlation Analysis:** Exploring relationships between different features and target variables.
- **Distribution Analysis:** Visualizing the distribution of key variables to understand their behavior.
- **Time-Series Decomposition:** Breaking down time-series data into trend, seasonality, and residual components.
- **Frequency Analysis:** Using Fourier Transform to analyze the frequency components of the data.
- **Wavelet Decomposition:** A more advanced technique to analyze time-series data at different scales.

By the end of this tutorial, you will have a solid foundation in EDA techniques that can be applied to any forecasting problem

In [3]:
```
#importing relevant libraries

import pandas as pd
import numpy as np
import polars as pl
from matplotlib import pyplot as plt
import seaborn as sns
from pathlib import Path
```

Let's quickly peak through the feature, targets, and also the training set quickly to understand broad features of the data we're working with

In [4]:
```
# Load dataset (assuming Parquet file)
df = []
for i in range(12):
    df.append(pd.read_parquet(f"./train_data/train_{i}.parquet"))
```

In [5]:
```
train_file = pd.concat(df, ignore_index=True)
train_file.head()
```

Out[5]:

	date_id	time_id	symbol_id	weight	feature_00	feature_01	feature_02	feature_03	f
0	0	0	1	3.889038	NaN	NaN	NaN	NaN	
1	0	0	7	1.370613	NaN	NaN	NaN	NaN	
2	0	0	9	2.285698	NaN	NaN	NaN	NaN	
3	0	0	10	0.690606	NaN	NaN	NaN	NaN	
4	0	0	14	0.440570	NaN	NaN	NaN	NaN	

5 rows × 92 columns

The training data has 4 types of columns:

1. {date_id}, {time_id}, {symbol_id} - it becomes your primary key going into the data.
2. weight: this column defines the relative contribution of each return on your forecast, which will primarily become a part of the loss, so we can calculate a weighted loss, as not all decisions are equally weighted.
3. features: The known input features, which will be available at the test time.
4. responder: Along with our actual column to be predicted: target_1, we have addiitonal information to select the correlated features as additional info available for the predictor which is also called: constant_dynamic_features.

Missing Data Analysis

For this purpose, we use a library called: missingo, which is great for looking at the missing nature of the tabular data including time-series.

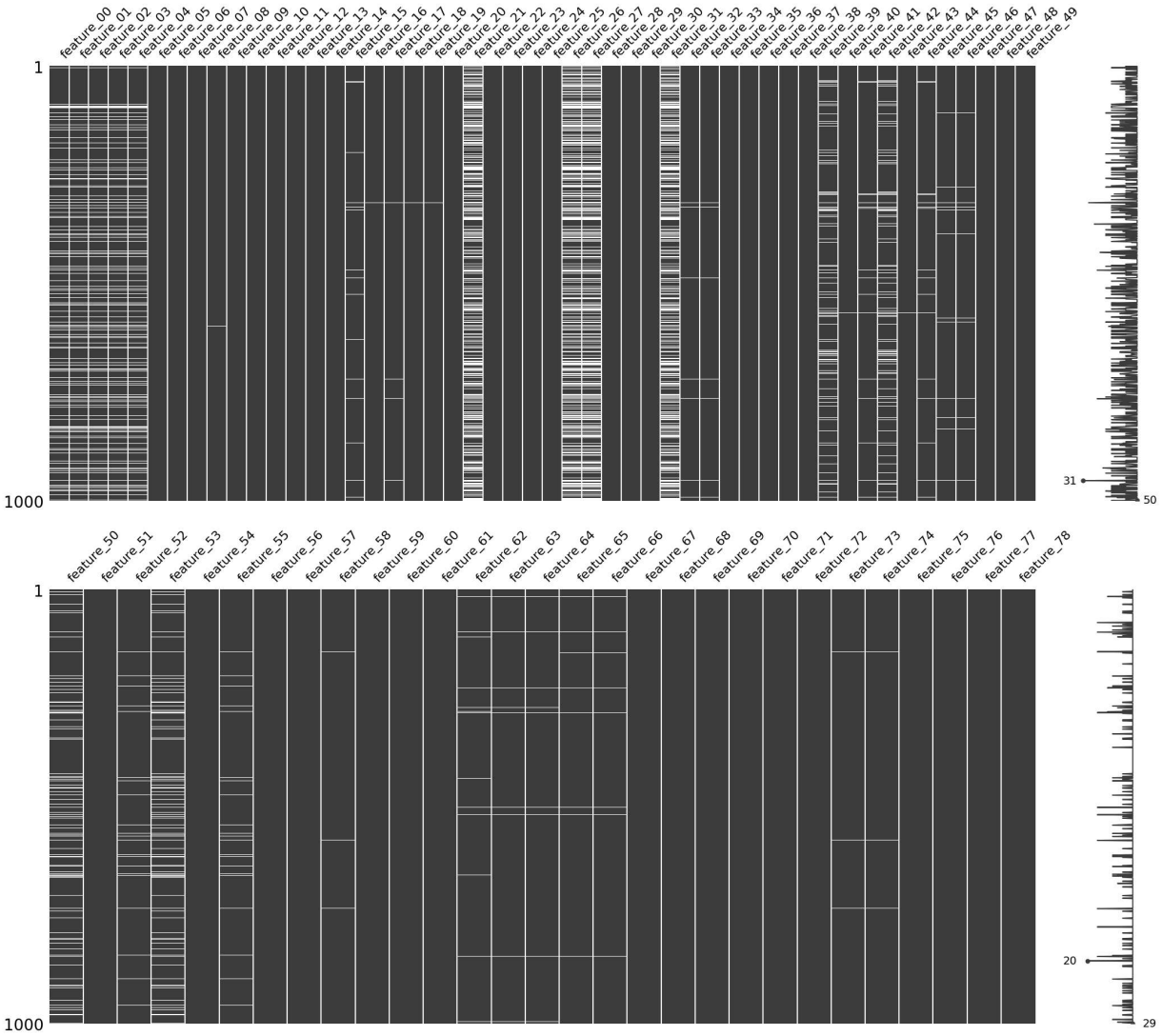
When dealing with tabular data, missing data has a lot of information, that can indicate a lot about the nature of the underlying data. It may also indicate which features contribute to a higher degree towards target.

In the forecasting, these features along with any missing data need to be tackled with the right instruments, and treating missing data is a vast field, and treating missing values requires a separate discussion of its own, which we will discuss in future

The missingno matrix provides a visual representation of missing values in the dataset. Each white line represents a missing value, while the black lines indicate non-missing data. This visualization helps us quickly identify which features have missing data and whether there are any patterns in the missingness (e.g., if certain features are missing together).

In [6]:
```
import missingno
feature_columns = [f"feature_{feature_num:02d}" for feature_num in range(79)]
```

```
missingno.matrix(train_file[feature_columns[0:50]].sample(1000))
missingno.matrix(train_file[feature_columns[50:]].sample(1000))
plt.show()
```

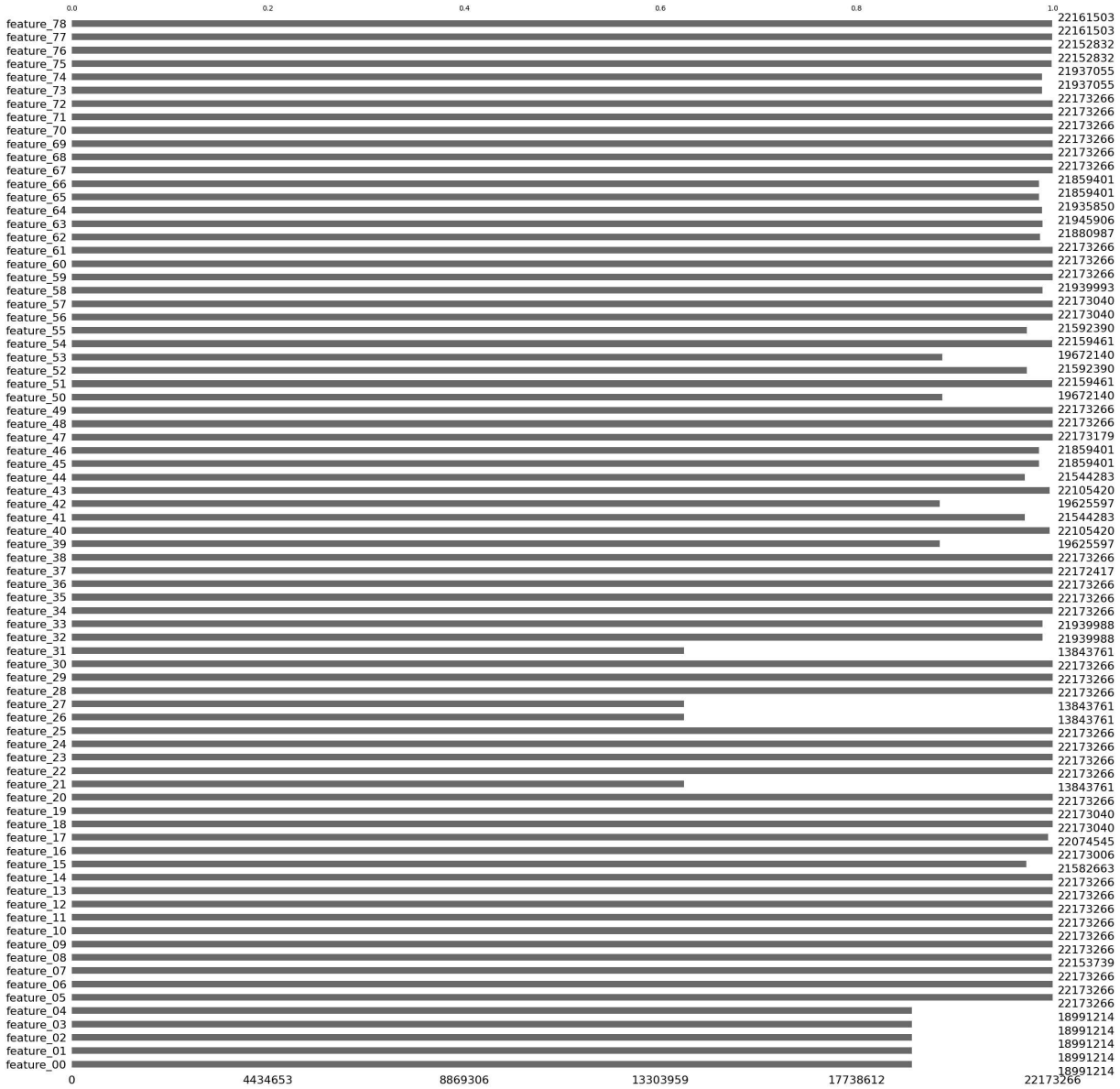


We can see feature_21, feature_26, and others including some from the start are completely missing, and are NAs. We can remove these, as these can not be filled, nor can be tackled at the training time.

Understanding the pattern of missing data is crucial for deciding how to handle it (e.g., imputation, deletion). If missingness is random, simple imputation techniques may suffice. However, if missingness is systematic, more sophisticated methods may be required.

In [7]:
```
missingno.bar(train_file[feature_columns])
```

Out[7]: <Axes: >

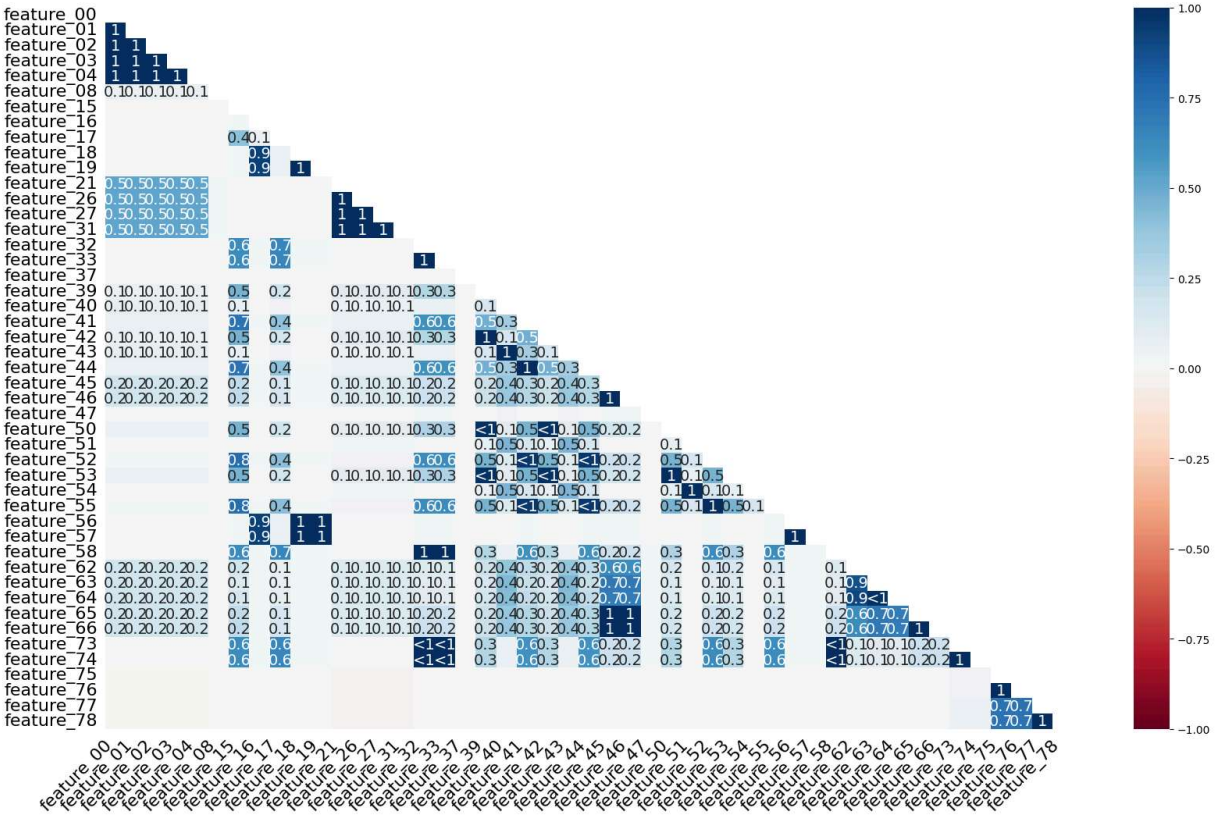


The bar plot shows the total number of non-missing values for each feature. Features with shorter bars have more missing data.

This plot helps prioritize which features may need more attention during data preprocessing. Features with a high percentage of missing values might need to be dropped or imputed carefully.

In [8]:
```
missingno.heatmap(train_file[feature_columns])
```

Out[8]: <Axes: >

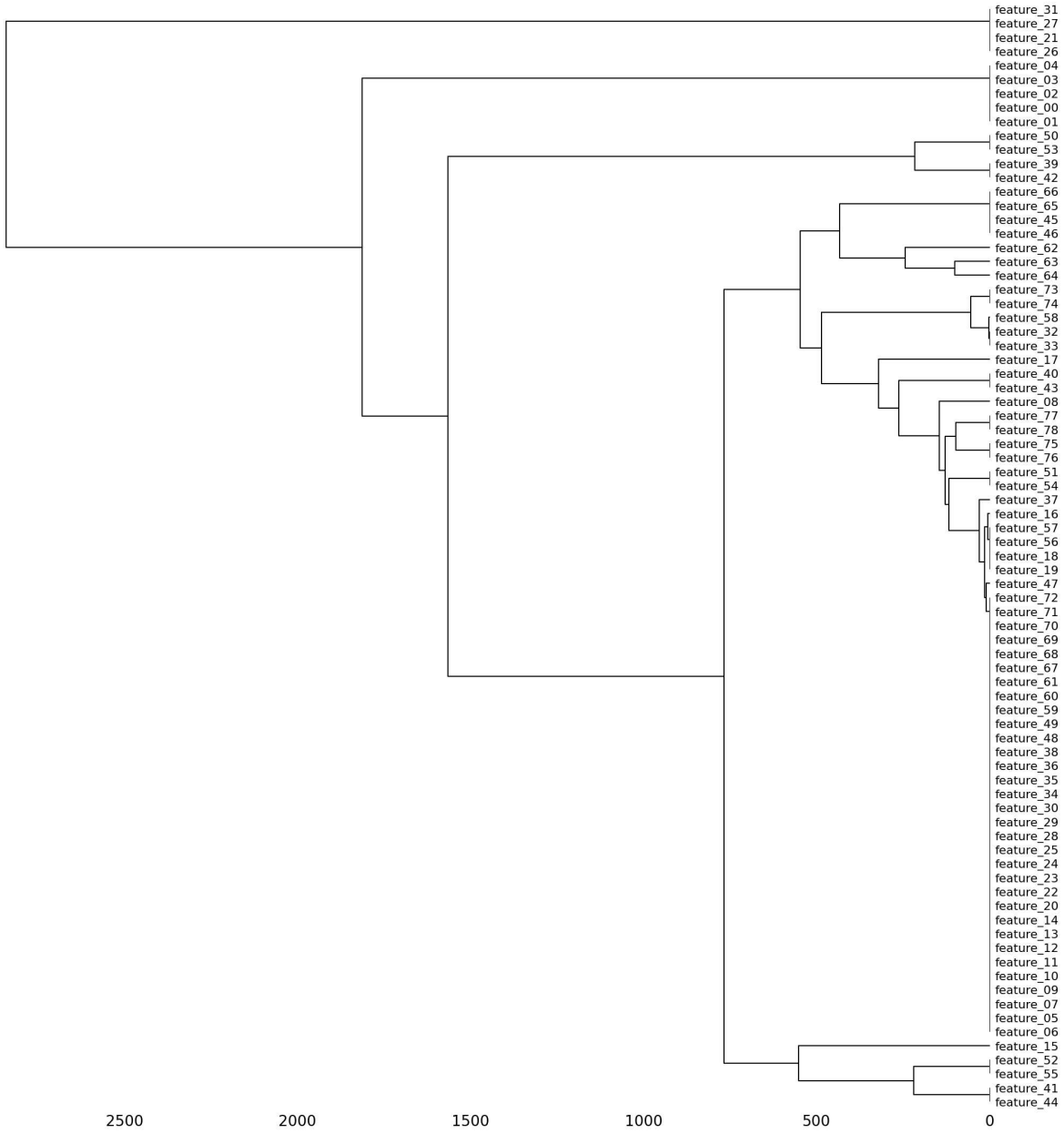


The heatmap shows the correlation of missingness between features. A high correlation (close to 1 or -1) indicates that missingness in one feature is strongly related to missingness in another.

This can help identify if missingness in certain features is related, which could indicate a systematic issue (e.g., data collection errors).

In [9]:
```
missingno.dendrogram(train_file[feature_columns])
```

Out[9]: <Axes: >

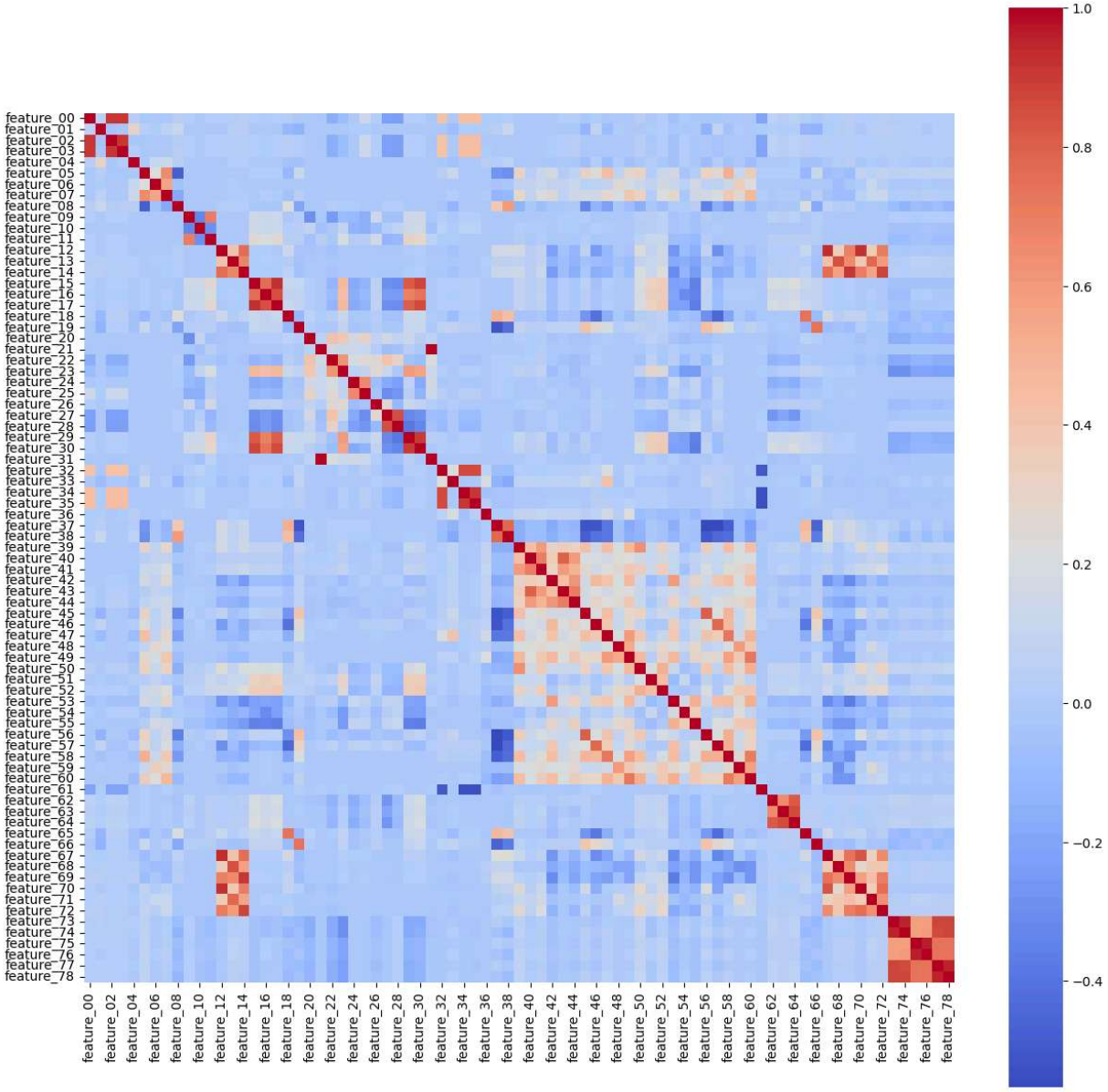


The dendrogram clusters features based on the similarity of their missingness patterns. Features that cluster together have similar patterns of missing data.

This can help identify groups of features that might be missing together, which could be useful for feature engineering or imputation strategies.

Correlation Analysis

In [11]:
```
feature_correlation = train_file[feature_columns].corr().fillna(0)
plt.figure(figsize = (15, 15))
sns.heatmap(feature_correlation, square = True, cmap = 'coolwarm')
plt.show()
```



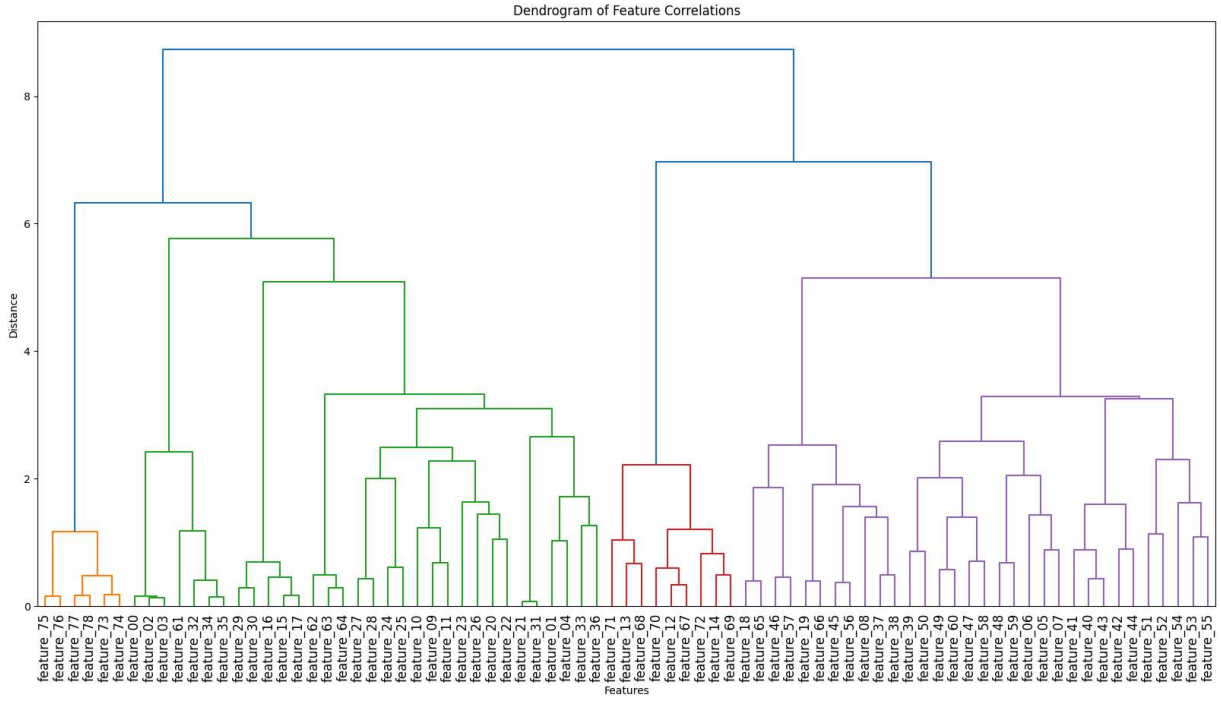
The heatmap shows the pairwise correlation between features. High positive or negative values indicate strong relationships between features.

Understanding feature correlations is essential for feature selection and to avoid multicollinearity in predictive models. Features with high correlation might be redundant and can be removed to simplify the model.

In [12]:
```
from scipy.cluster.hierarchy import dendrogram, linkage

distance_matrix = 1 - np.abs(feature_correlation)
linkage_matrix = linkage(distance_matrix, method = 'ward')

plt.figure(figsize=(20, 10))
dendrogram(
    linkage_matrix,
    labels=feature_correlation.columns,
    leaf_rotation=90,
    leaf_font_size=12
```

```
)
plt.title("Dendrogram of Feature Correlations")
plt.xlabel("Features")
plt.ylabel("Distance")
plt.show()
```

```
C:\Users\daksh\AppData\Local\Temp\ipykernel_13992\953951188.py:4: ClusterWarning: Th
e symmetric non-negative hollow observation matrix looks suspiciously like an uncond
ensed distance matrix
  linkage_matrix = linkage(distance_matrix, method = 'ward')
```

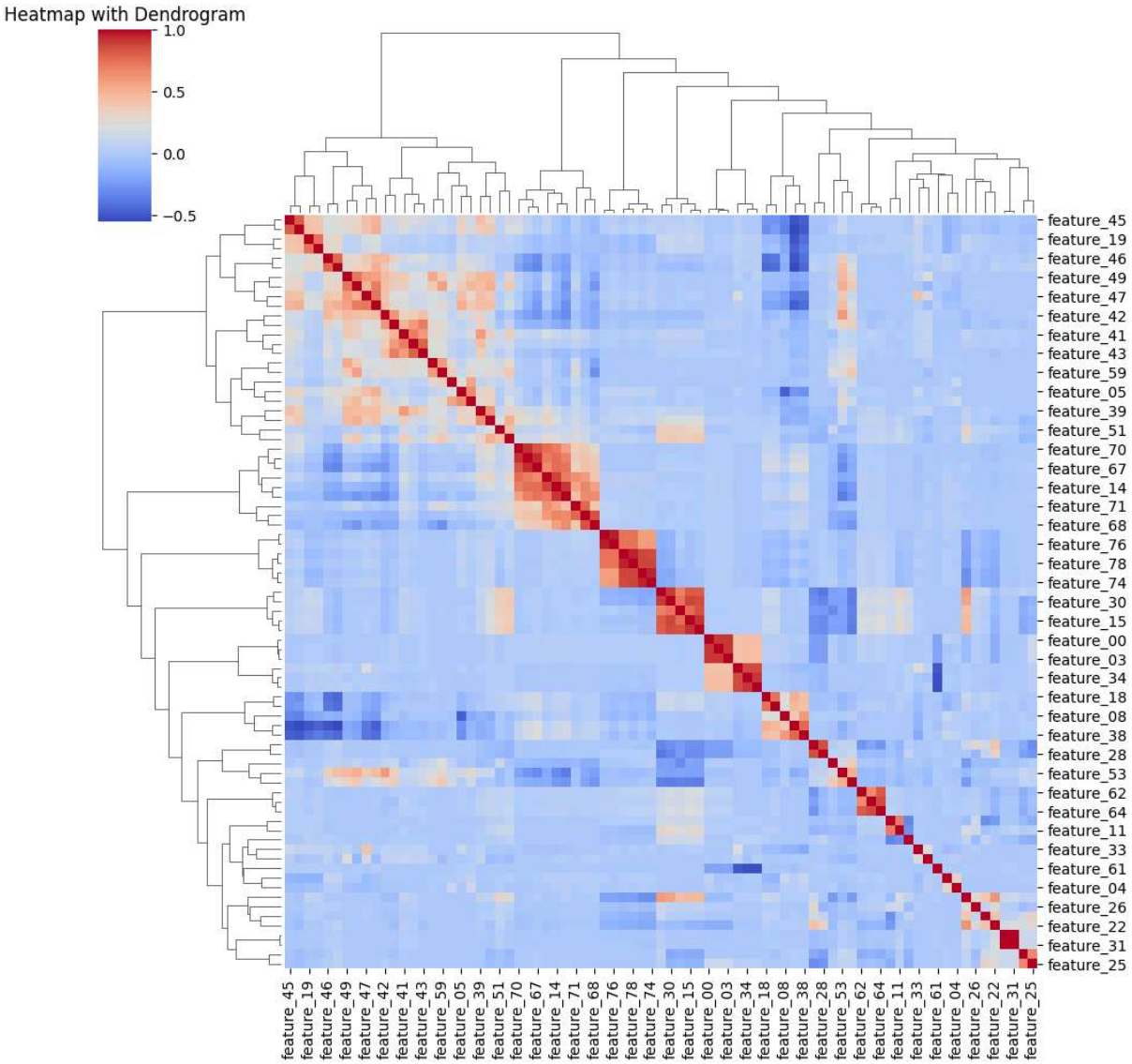


The dendrogram clusters features based on their correlation. Features that are close together in the dendrogram are highly correlated.

This visualization helps in identifying groups of correlated features, which can be useful for dimensionality reduction or feature engineering.

In [13]:
```
plt.figure(figsize = (20, 20))
sns.clustermap(feature_correlation, cmap='coolwarm', annot=False, method='ward')
plt.title("Heatmap with Dendrogram")
plt.show()
```

```
<Figure size 2000x2000 with 0 Axes>
```

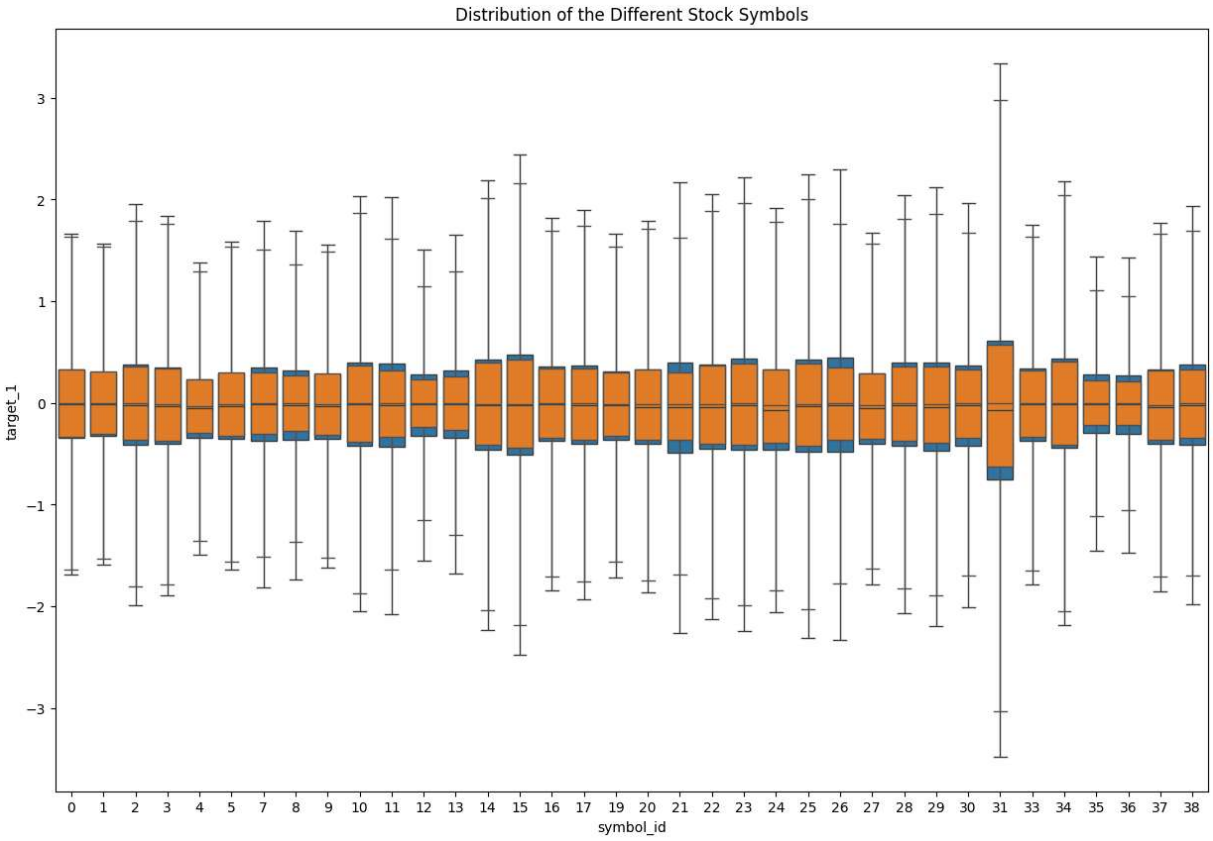


The clustermap combines the heatmap and dendrogram, showing both the correlation values and the hierarchical clustering of features.

This provides a comprehensive view of feature relationships, helping to identify clusters of features that behave similarly.

Target Relationship & Distribution

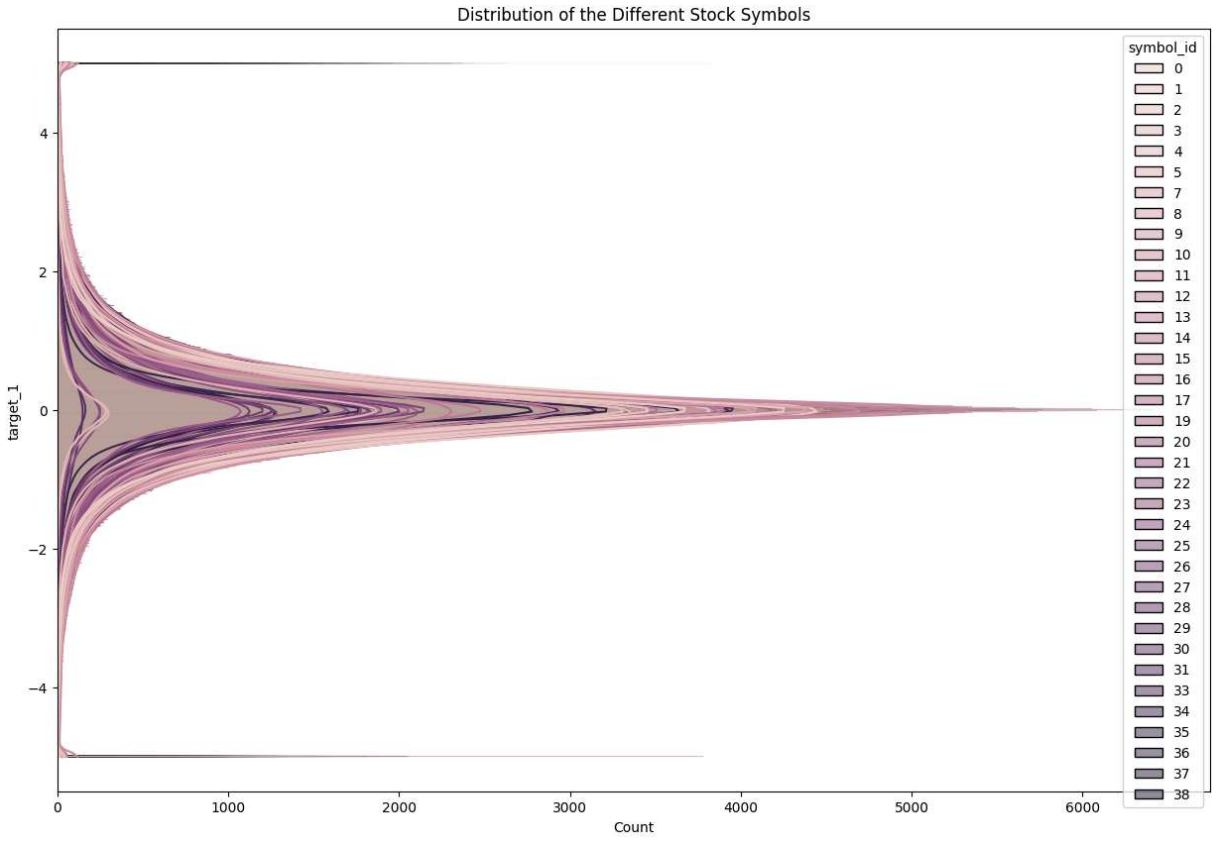
In [14]:
```
plt.figure(figsize = (15, 10))
sns.boxplot(train_file, x = 'symbol_id', y = 'target_1', whis = 2.0, showfliers = F
sns.boxplot(train_file, x = 'symbol_id', y = 'target_2', whis = 2.0, showfliers = F
plt.title('Distribution of the Different Stock Symbols')
plt.show()
```



The boxplot shows the distribution of the target variable (target_1) across different stock symbols. The box represents the interquartile range (IQR), and the whiskers show the range of the data

This helps identify if there are significant differences in the target variable across different symbols. If certain symbols have consistently higher or lower values, this could be important for modeling.

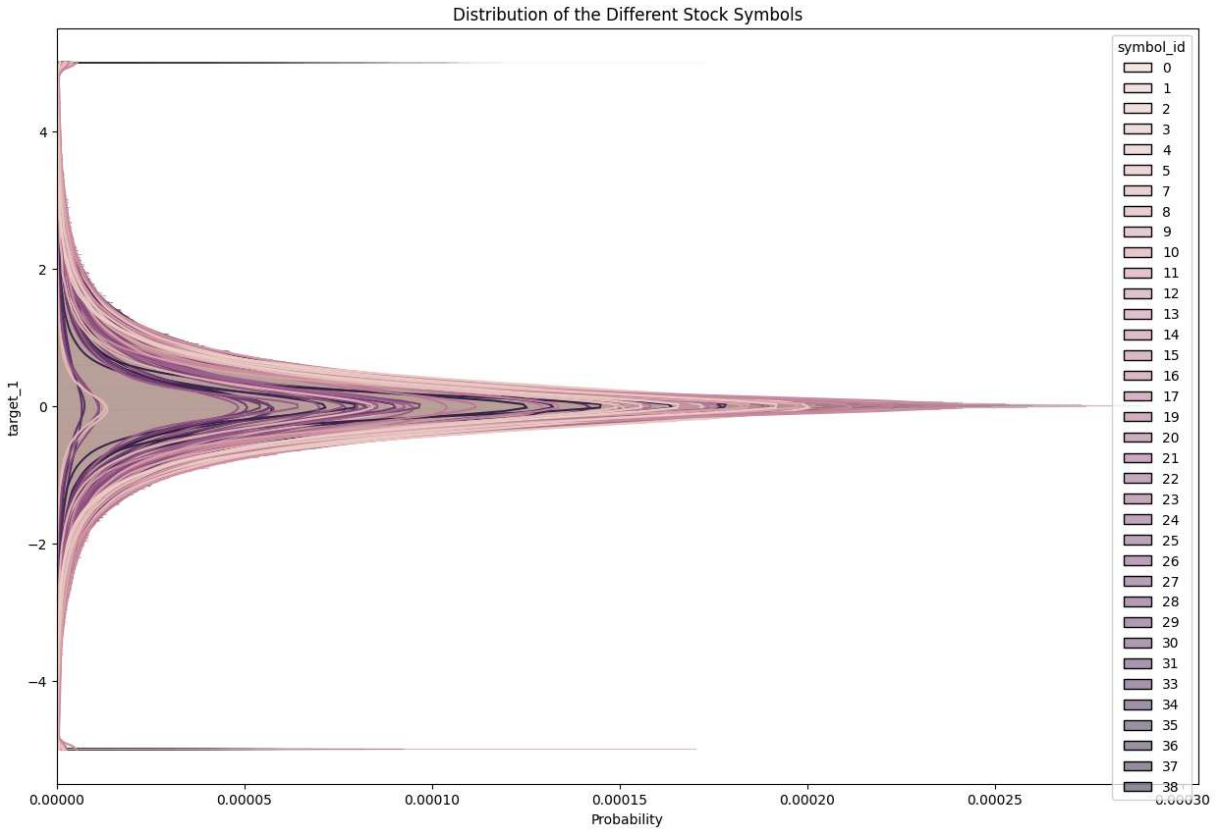
In [15]:
```
import warnings
warnings.simplefilter("ignore")

plt.figure(figsize = (15, 10))
sns.histplot(train_file,y = 'target_1', kde = True, hue = 'symbol_id', color = 'gre
sns.histplot(train_file,y = 'target_2', kde = True, hue = 'symbol_id', color = 'gre
plt.title('Distribution of the Different Stock Symbols')
plt.show()
```



The histogram shows the distribution of target_1 for each symbol, with a kernel density estimate (KDE) overlay.

This visualization helps understand the shape of the distribution (e.g., normal, skewed) and whether there are differences in the distribution across symbols.

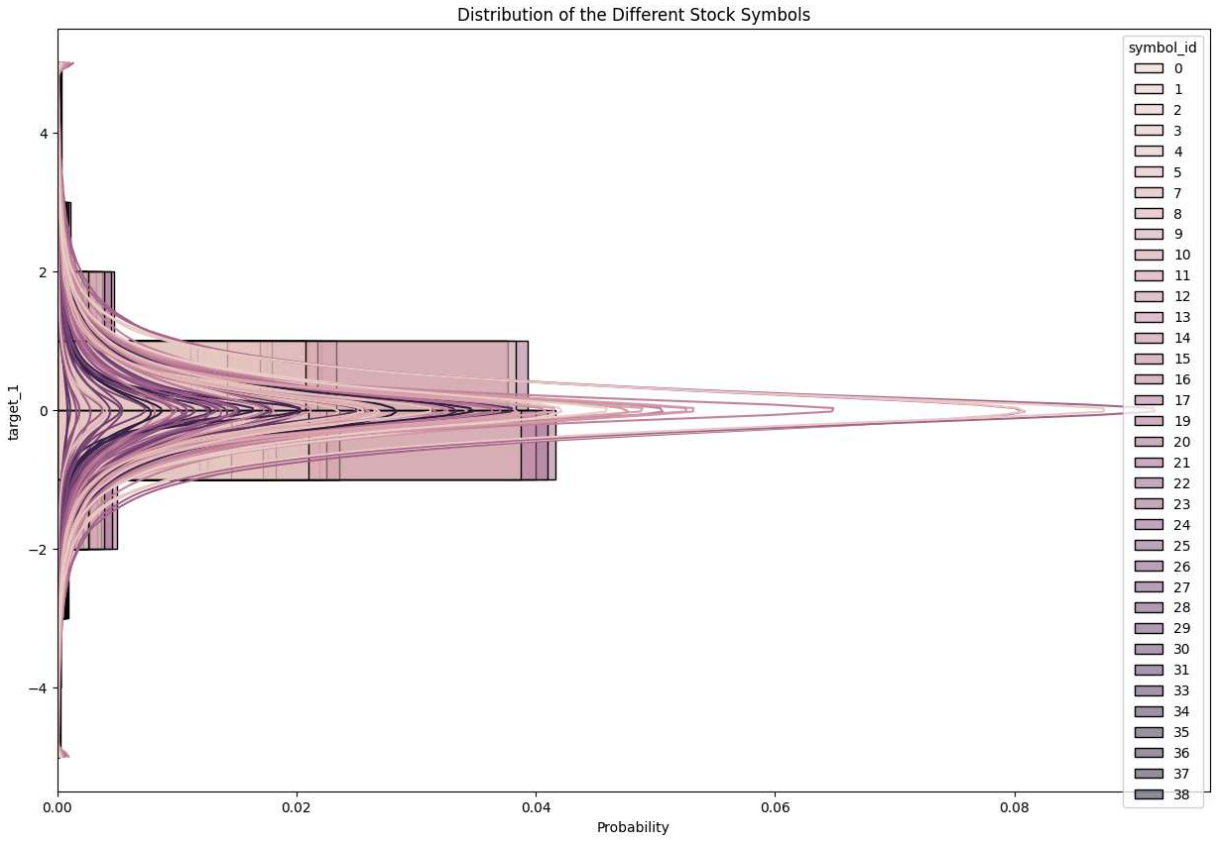
In [16]:
```
plt.figure(figsize = (15, 10))
sns.histplot(train_file,y = 'target_1', kde = True, hue = 'symbol_id', stat = 'prob
sns.histplot(train_file,y = 'target_2', kde = True, hue = 'symbol_id', stat = 'prob
plt.title('Distribution of the Different Stock Symbols')
plt.show()
```



This histogram is weighted by the weight column, which might represent the importance or volume of each data point.

Weighting the histogram can provide a more accurate representation of the distribution, especially if some data points are more significant than others.

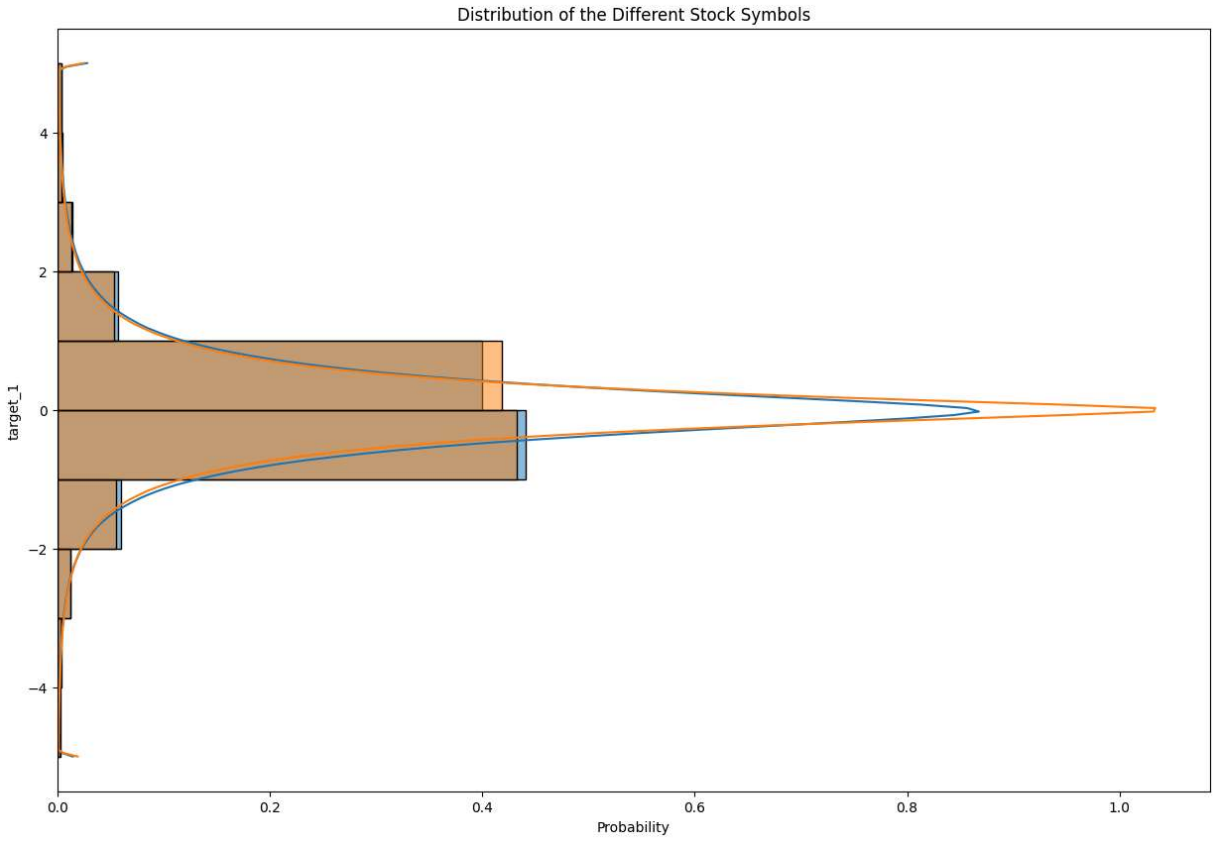
In [17]:
```
import warnings
warnings.simplefilter('ignore')

plt.figure(figsize = (15, 10))
sns.histplot(train_file,y = 'target_1', kde = True, hue = 'symbol_id', stat = 'prob
sns.histplot(train_file,y = 'target_2', kde = True, hue = 'symbol_id', stat = 'prob
plt.title('Distribution of the Different Stock Symbols')
plt.show()
```



This heatmap shows the correlation of target_1 across different symbols.

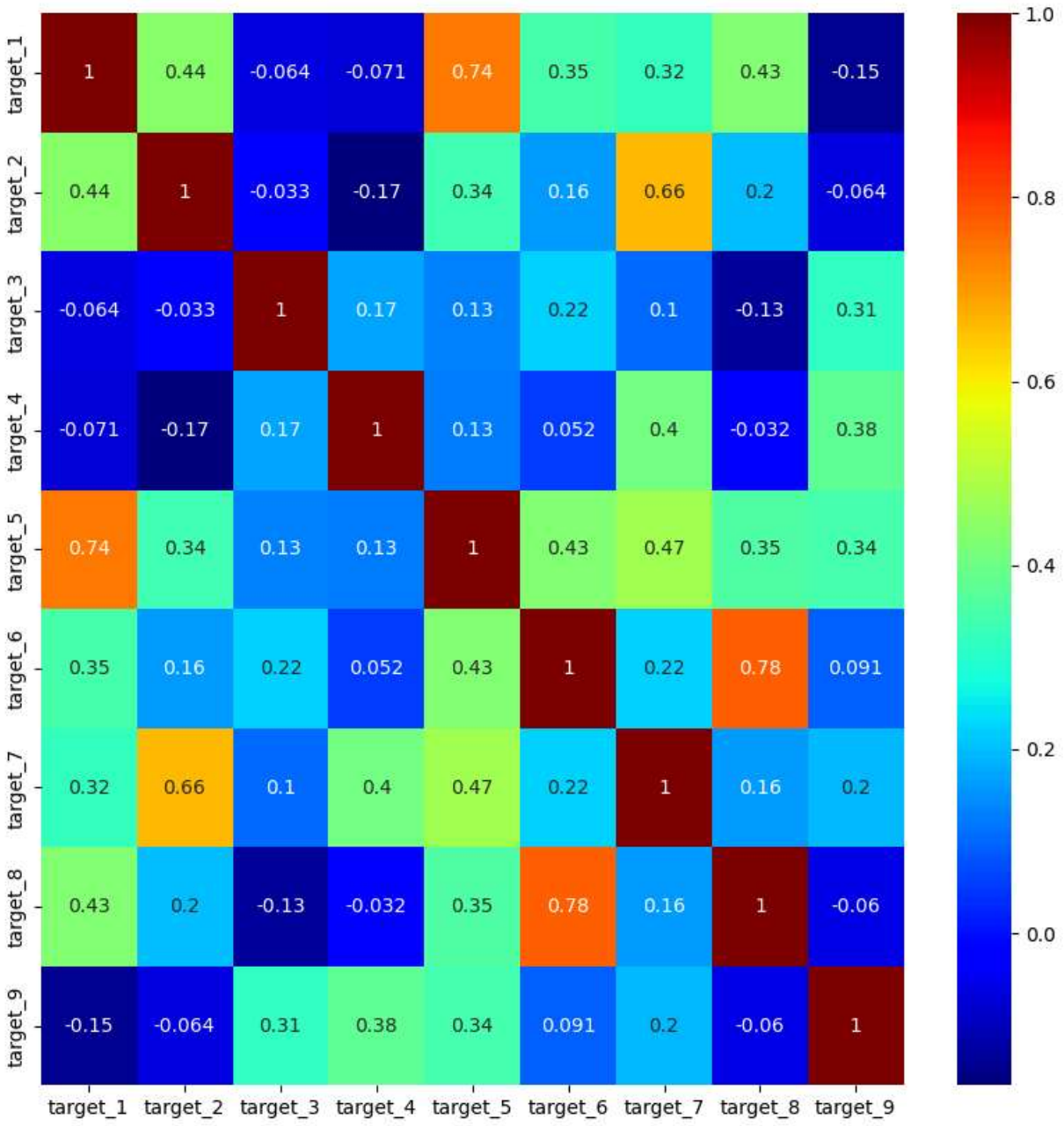
This can help identify if certain symbols move together, which could be useful for portfolio management or risk assessment.

In [18]:
```
plt.figure(figsize = (15, 10))
sns.histplot(train_file,y = 'target_1', kde = True, stat = 'probability', weights =
sns.histplot(train_file,y = 'target_2', kde = True, stat = 'probability', weights =
plt.title('Distribution of the Different Stock Symbols')
plt.show()
```



Lastly, lets quickly look at the most probable returns when averaged by weights, and normalized.

In [19]:
```
target_column = [f'target_{target_num}' for target_num in range(1,10)]
target_corr = train_file[target_column].corr()

plt.figure(figsize = (10,10))
sns.heatmap(target_corr, annot = True, cmap = 'jet')
plt.show()
```

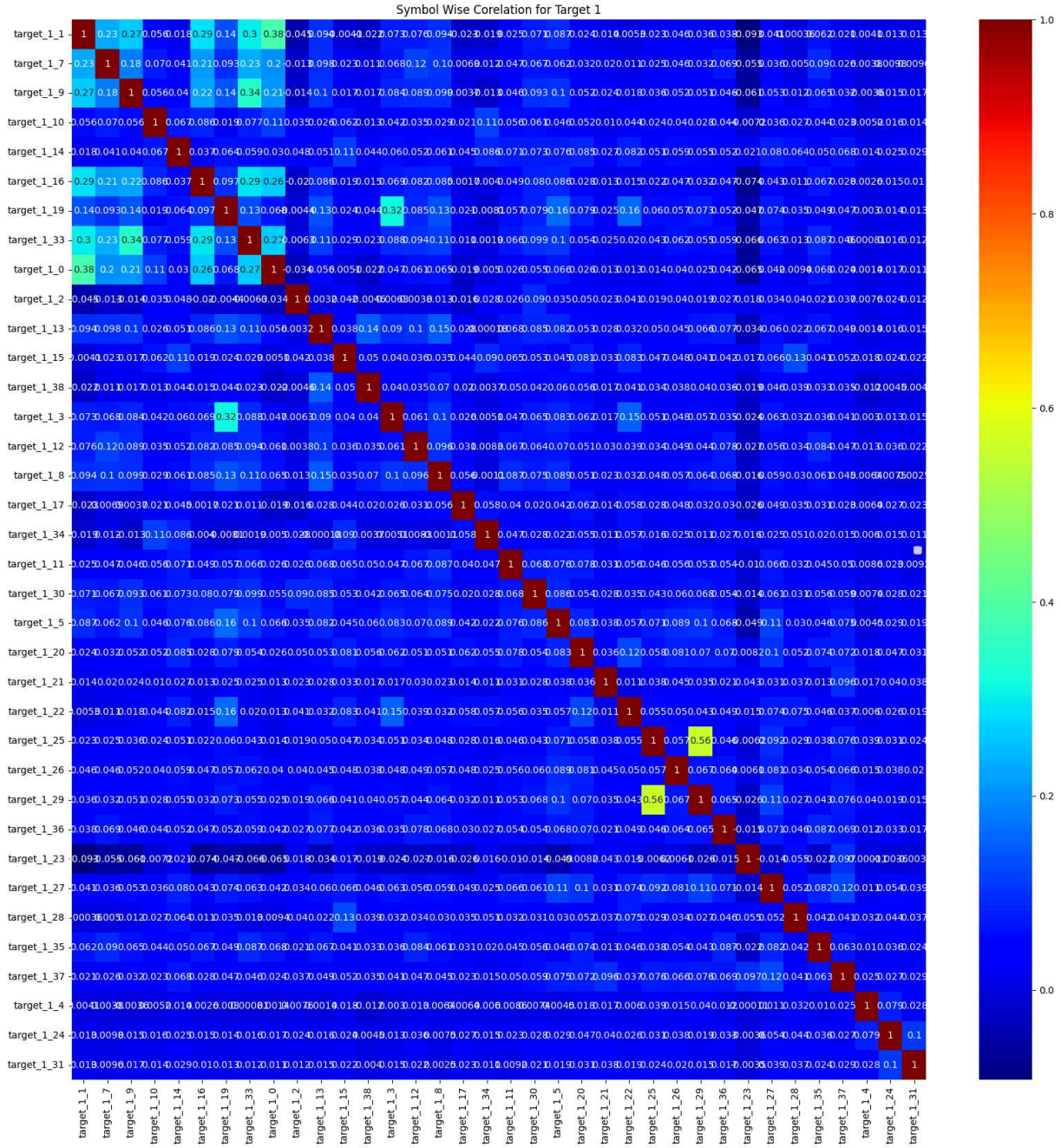


The heatmap shows the correlation between different responder variables.

Understanding the relationships between different responders can help in feature engineering or in understanding the underlying structure of the data.

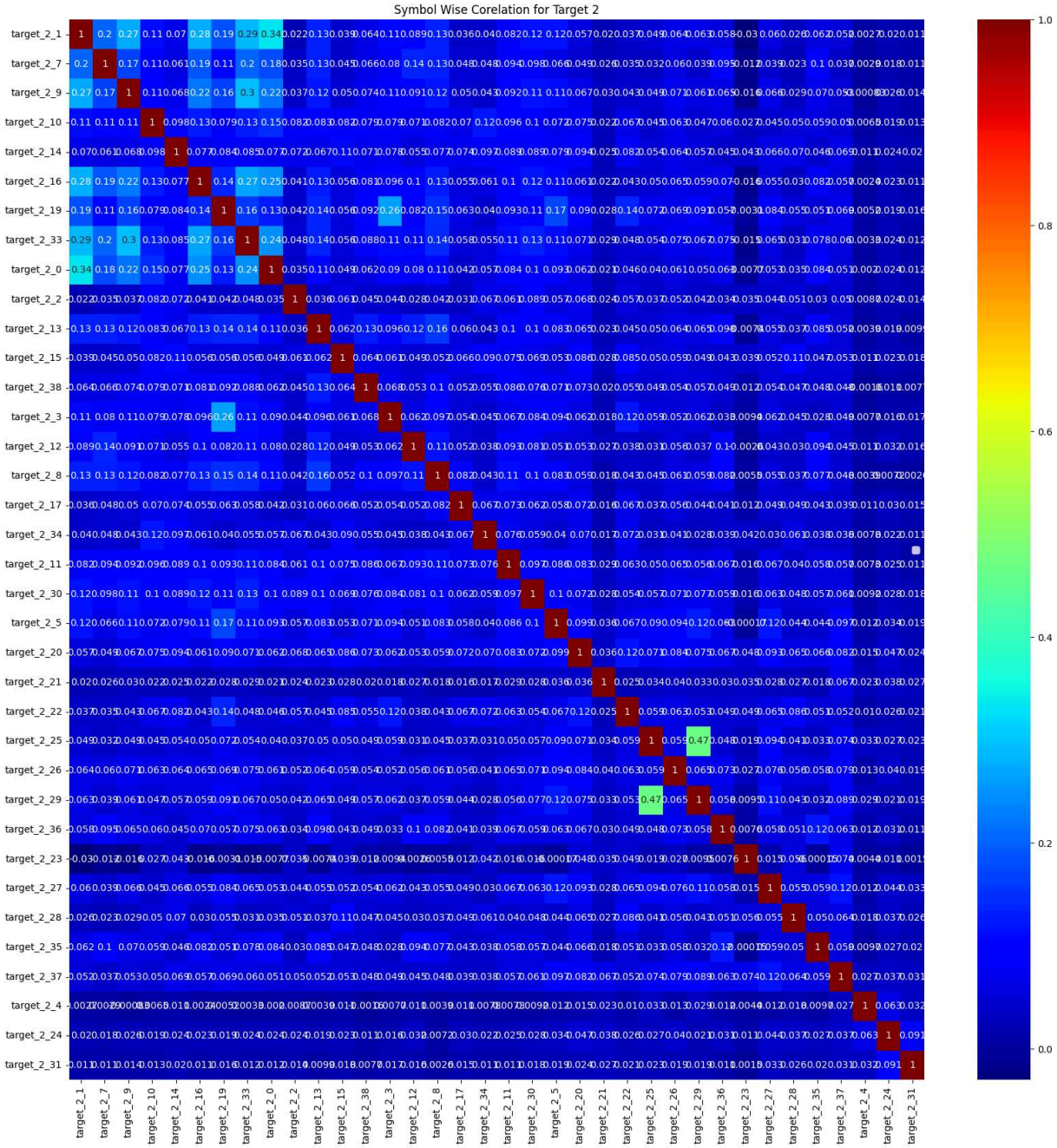
In [20]:
```
symbol_pivot = train_file.pivot(index = ['date_id','time_id'], columns = ['symbol_i
symbol_pivot.columns = [f"{first}_{second}" for first, second in symbol_pivot.colum
symbol_columns = [f"target_1_{symbol}" for symbol in train_file['symbol_id'].unique
symbol_corr = symbol_pivot[symbol_columns].fillna(0).corr()
plt.figure(figsize = (20, 20))
sns.heatmap(symbol_corr, cmap = 'jet', annot=True)
plt.title('Symbol Wise Corelation for Target 1')
plt.legend()
```

Out[20]:
```
<matplotlib.legend.Legend at 0x1eb53be2e90>
```



In [21]:
```
symbol_pivot = train_file.pivot(index = ['date_id','time_id'], columns = ['symbol_i
symbol_pivot.columns = [f"{first}_{second}" for first, second in symbol_pivot.colum
symbol_columns = [f"target_2_{symbol}" for symbol in train_file['symbol_id'].unique
symbol_corr = symbol_pivot[symbol_columns].fillna(0).corr()
plt.figure(figsize = (20, 20))
sns.heatmap(symbol_corr, cmap = 'jet', annot=True)
plt.title('Symbol Wise Corelation for Target 2')
plt.legend()
```

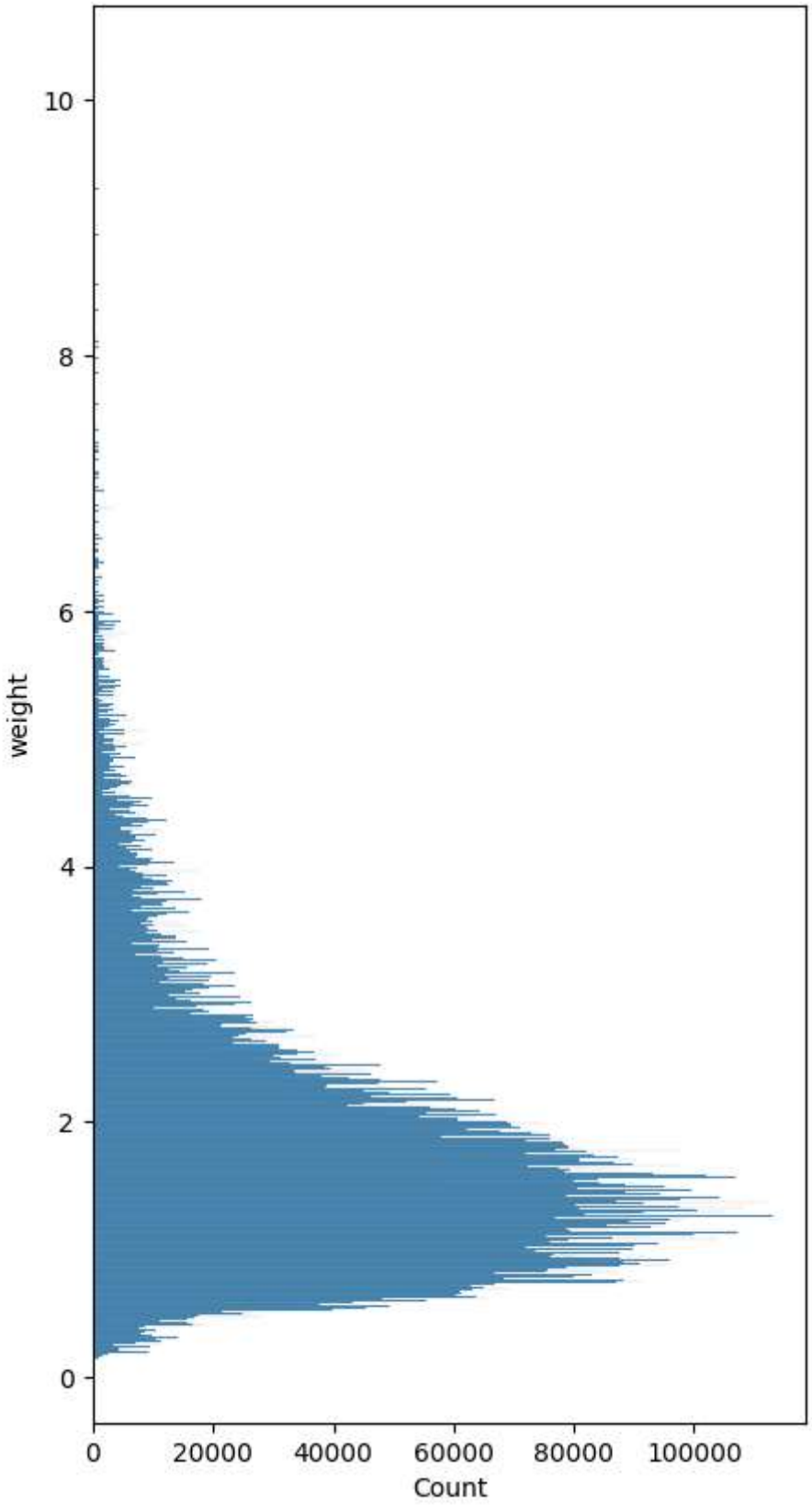
Out[21]: <matplotlib.legend.Legend at 0x1ed86430cd0>



This heatmap shows the correlation of target_1 across different symbols.

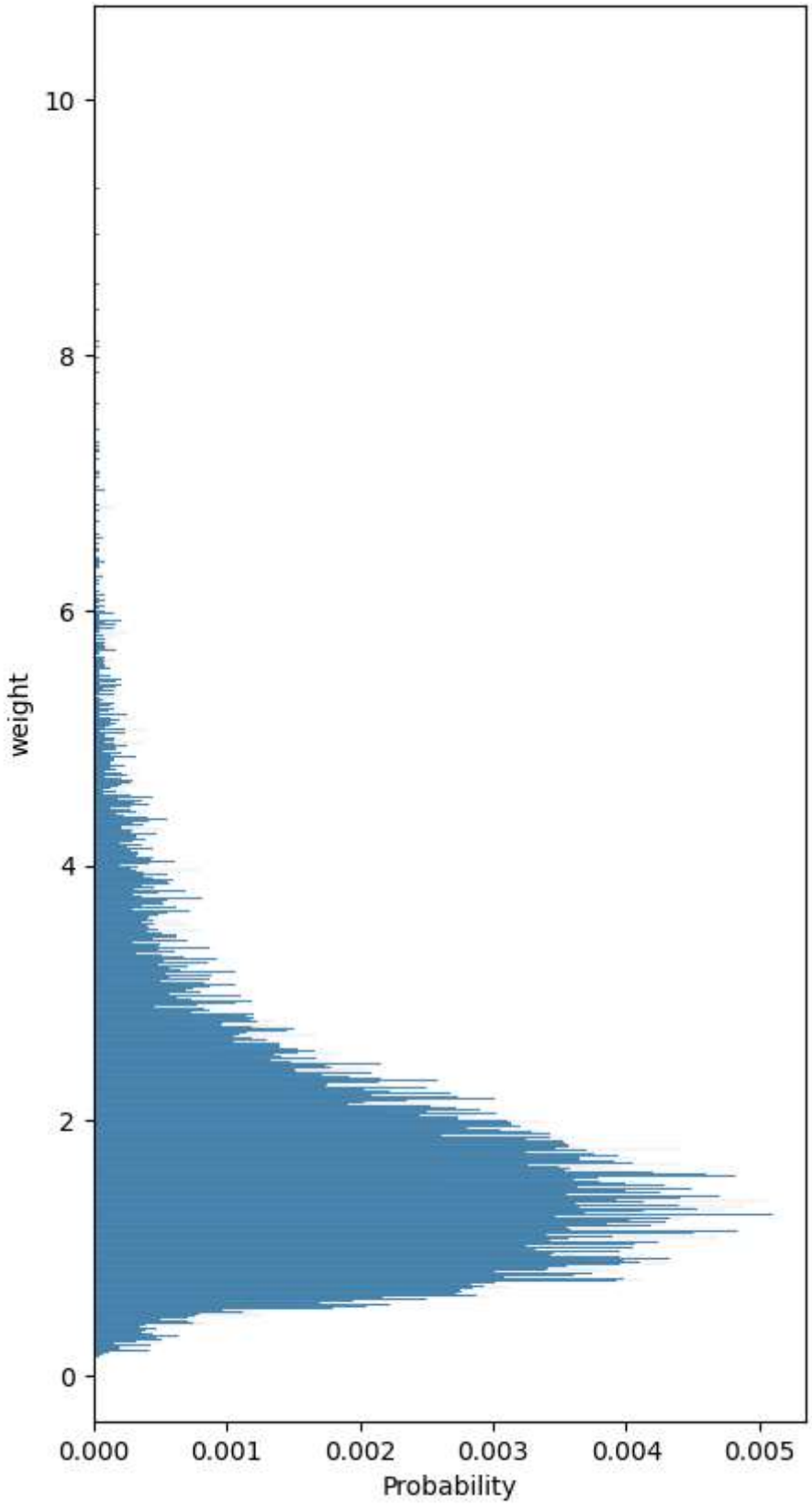
This can help identify if certain symbols move together, which could be useful for portfolio management or risk assessment.

In [22]:
```
plt.figure(figsize = (5,10))
sns.histplot(train_file, y = 'weight')
plt.show()
```



Let's quickly analyze the weight component to understand its distribution.

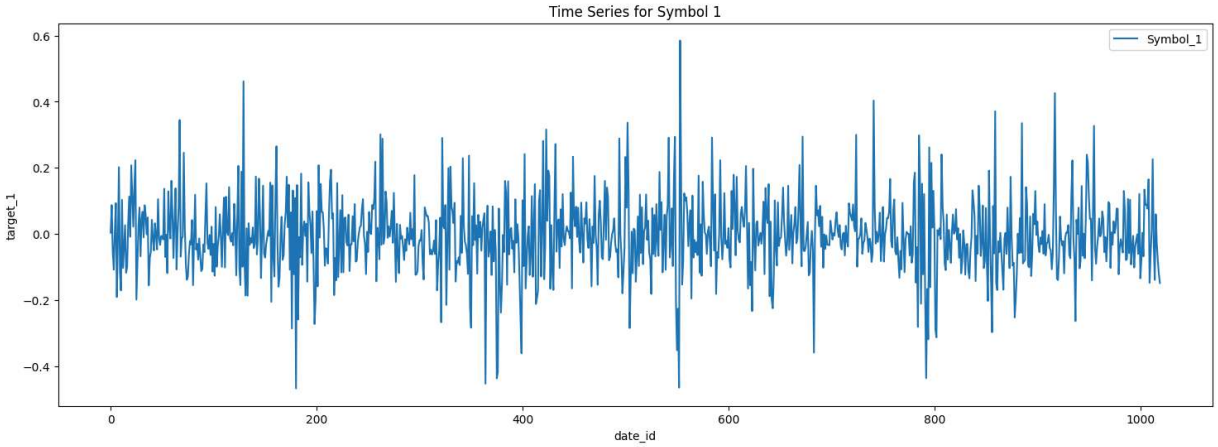
In [23]:
```
plt.figure(figsize = (5,10))
sns.histplot(train_file, y = 'weight', stat = 'probability')
plt.show()
```

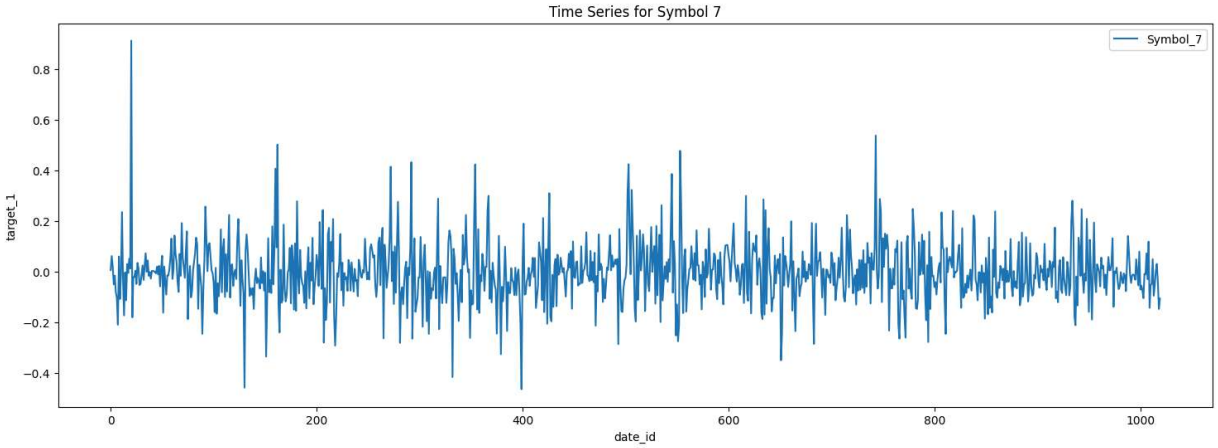


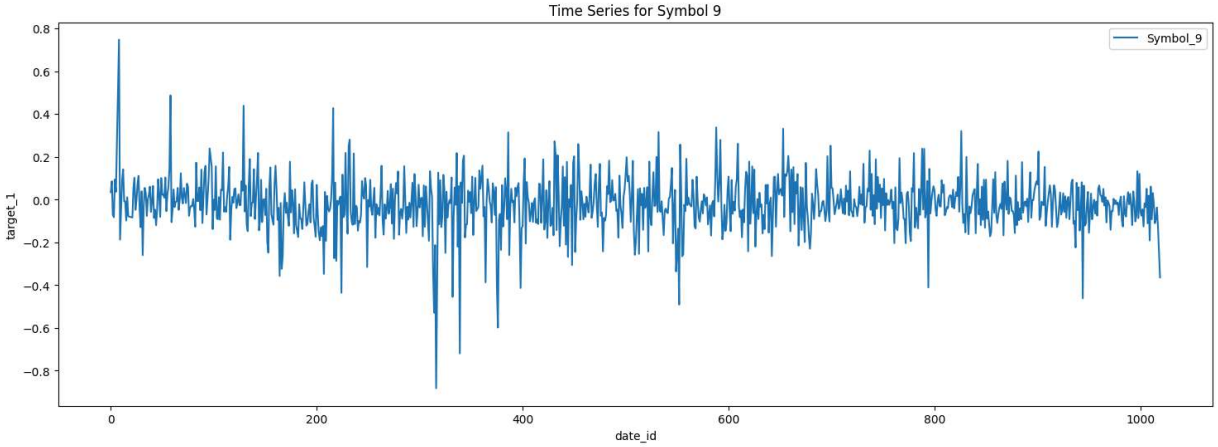
Also a quick preview of the weight component as a measure of standard probability.

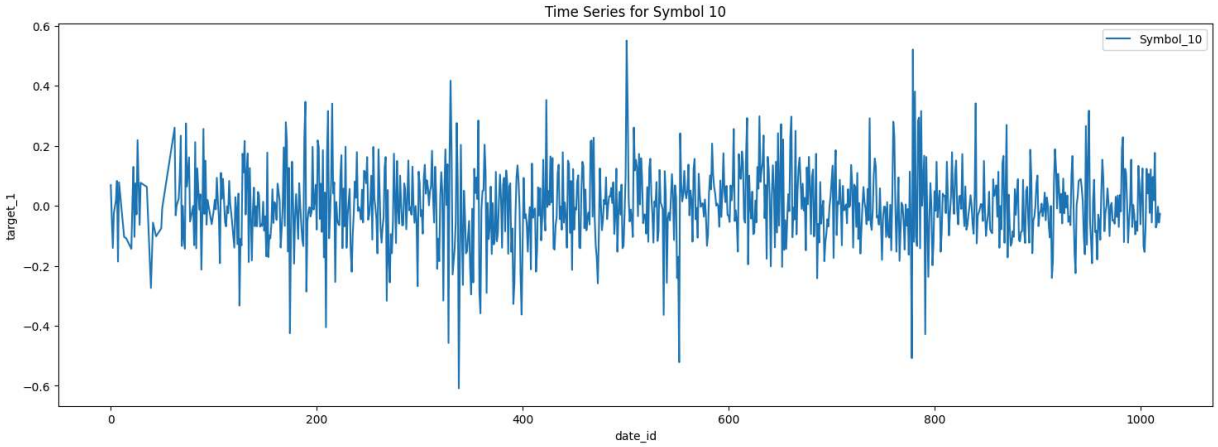
Time Series Analysis

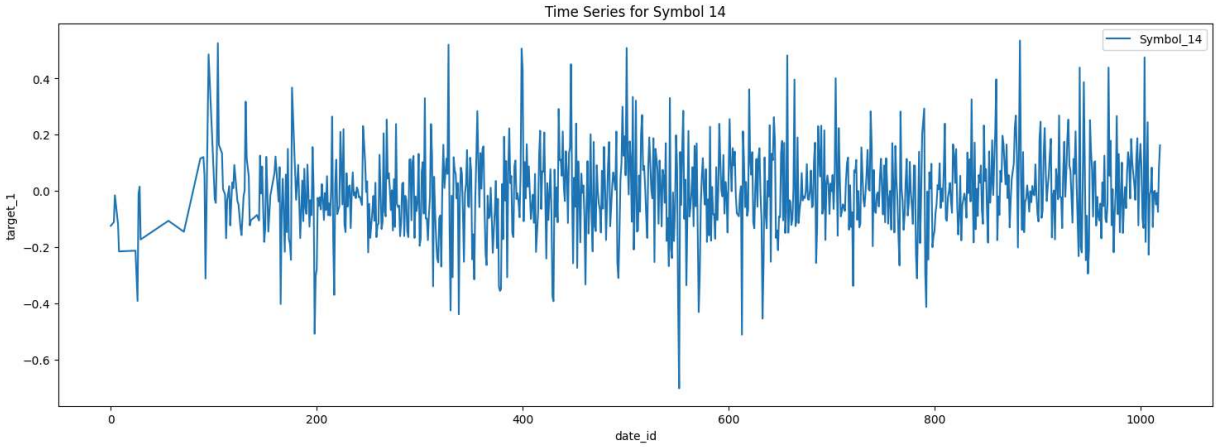
In [24]:
```
daily_avg = train_file.groupby(['date_id','symbol_id'])['target_1'].mean().reset_in

for symbol_id in train_file['symbol_id'].unique():
    symbol_data = daily_avg[daily_avg['symbol_id']==symbol_id]
    plt.figure(figsize=(18,6))
    plt.plot(symbol_data['date_id'], symbol_data['target_1'], label = f"Symbol_{sym
    plt.title(f"Time Series for Symbol {symbol_id}")
    plt.xlabel("date_id")
    plt.ylabel("target_1")
    plt.legend()
    plt.show()
```

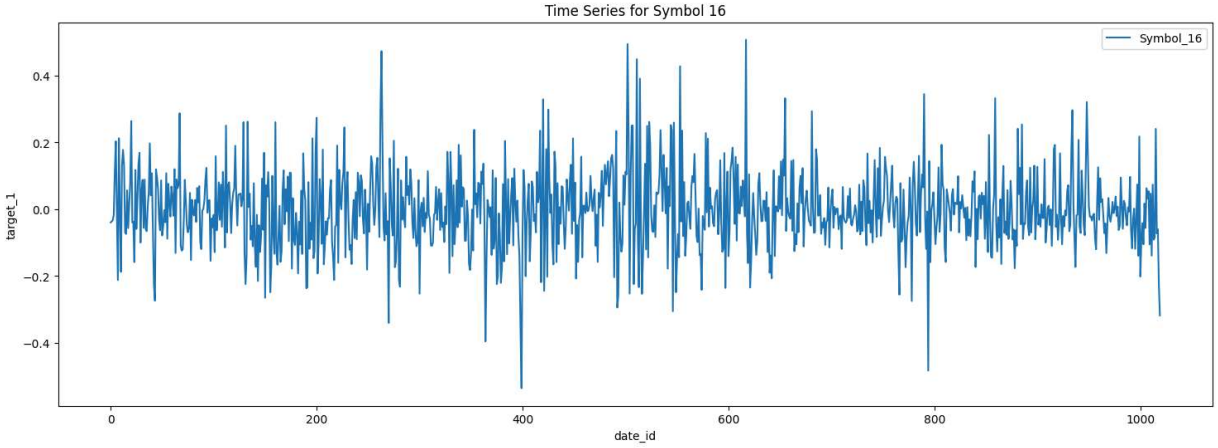


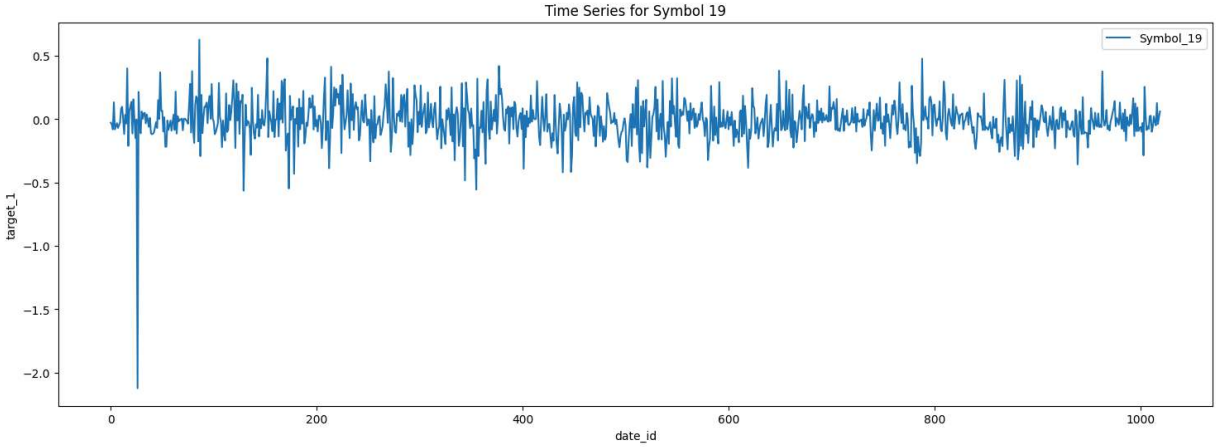


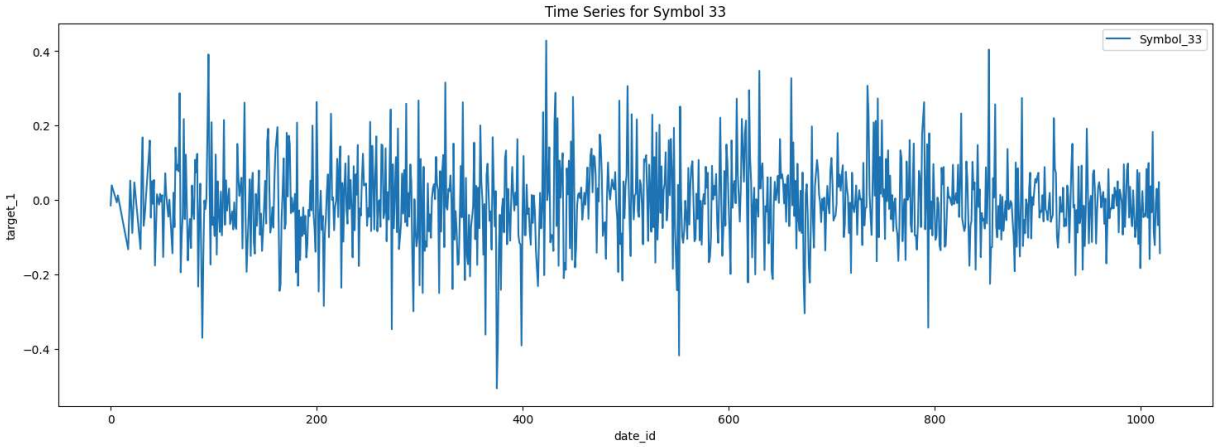


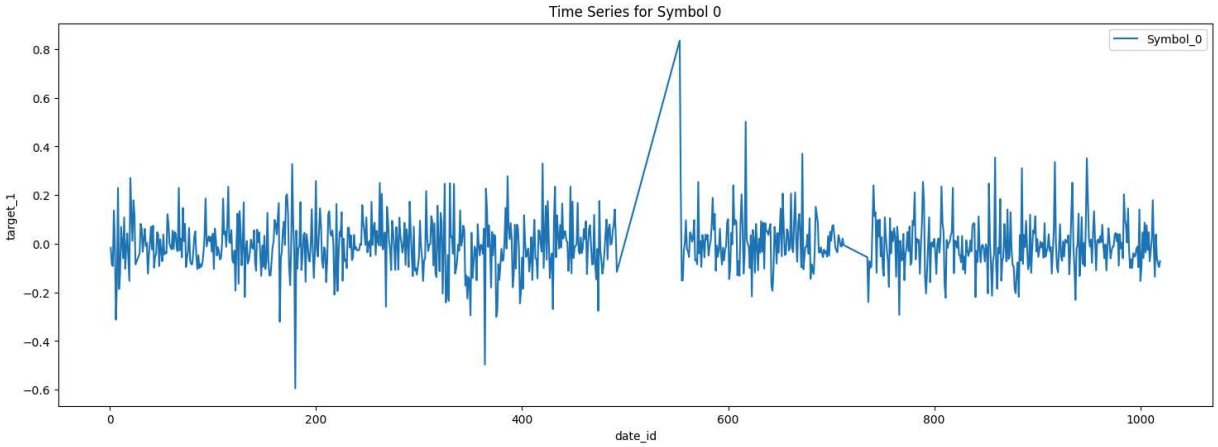

Time Series for Symbol 10
Symbol_10
target_1
date_id

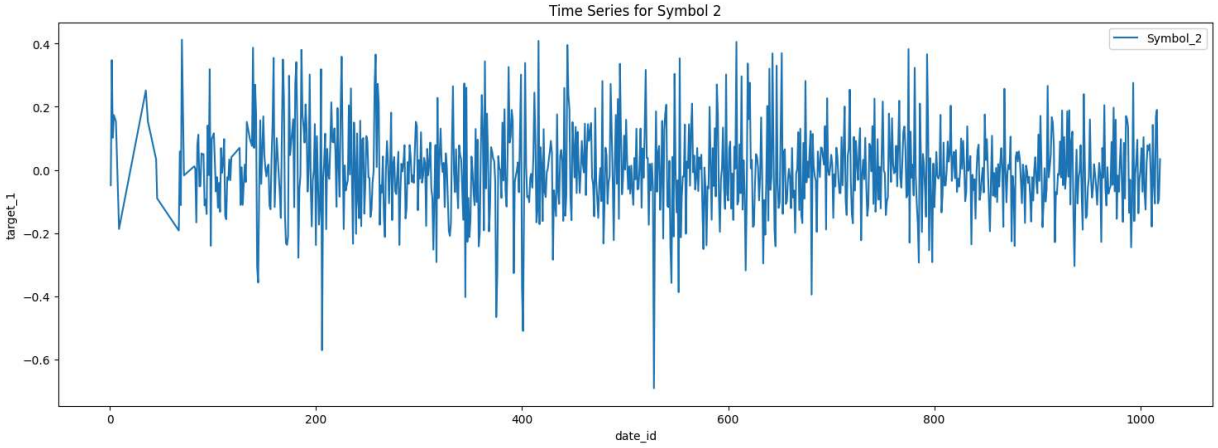

Time Series for Symbol 14
Symbol_14
target_1
date_id

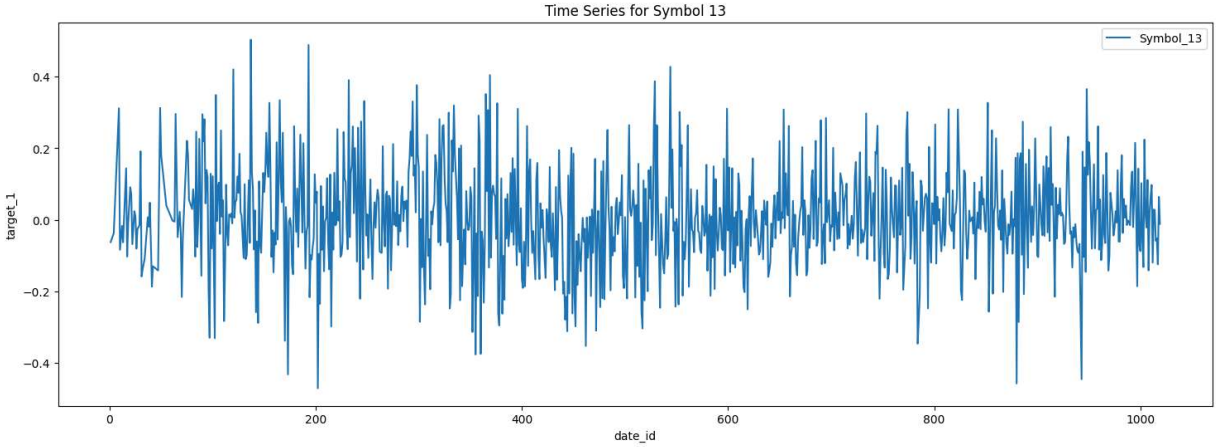

Time Series for Symbol 16
Symbol_16
target_1
date_id

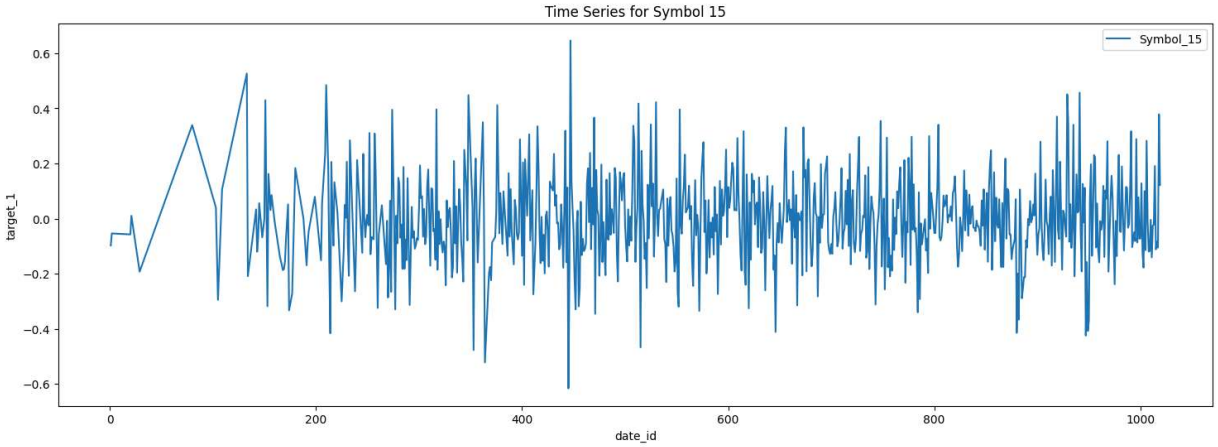

Time Series for Symbol 19
Symbol_19
target_1
date_id

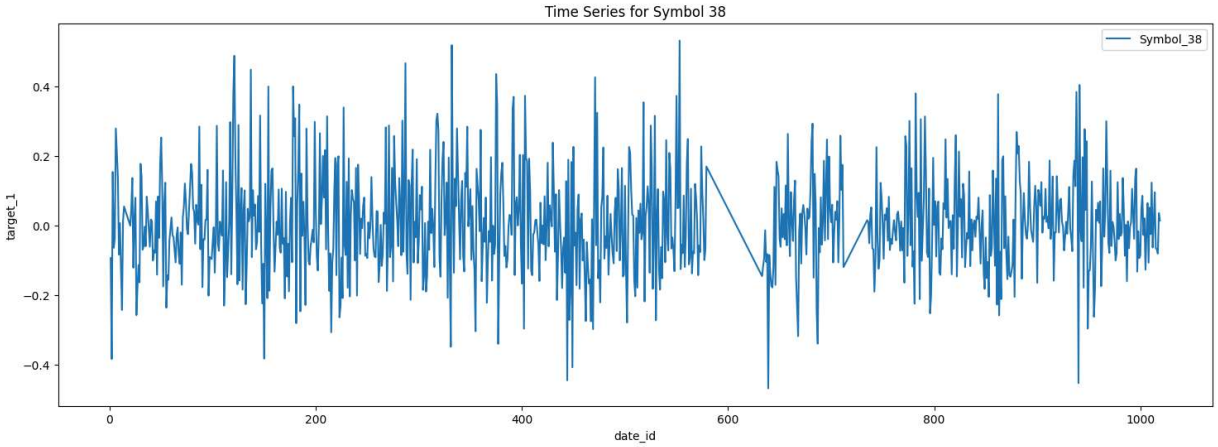

Time Series for Symbol 33
Symbol_33
target_1
date_id

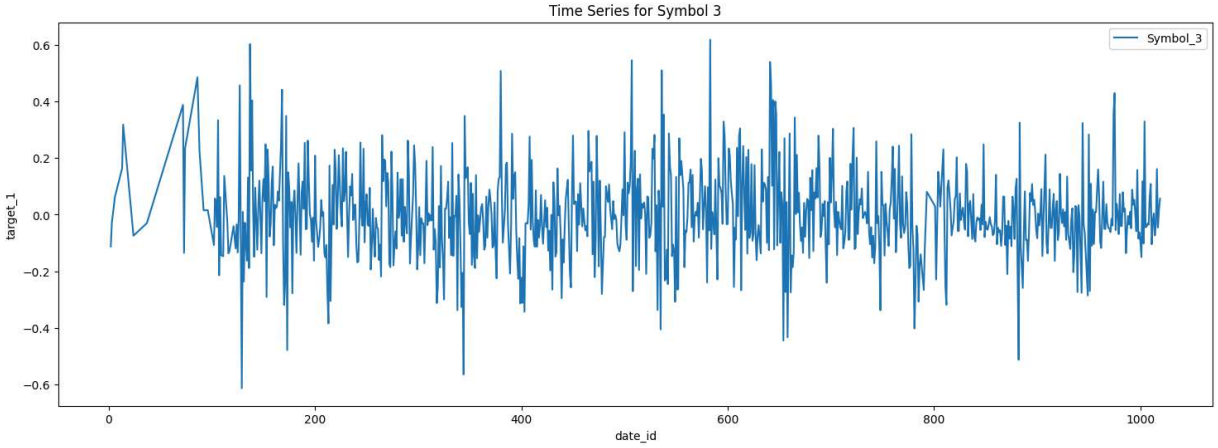

Time Series for Symbol 0
Symbol_0
target_1
date_id

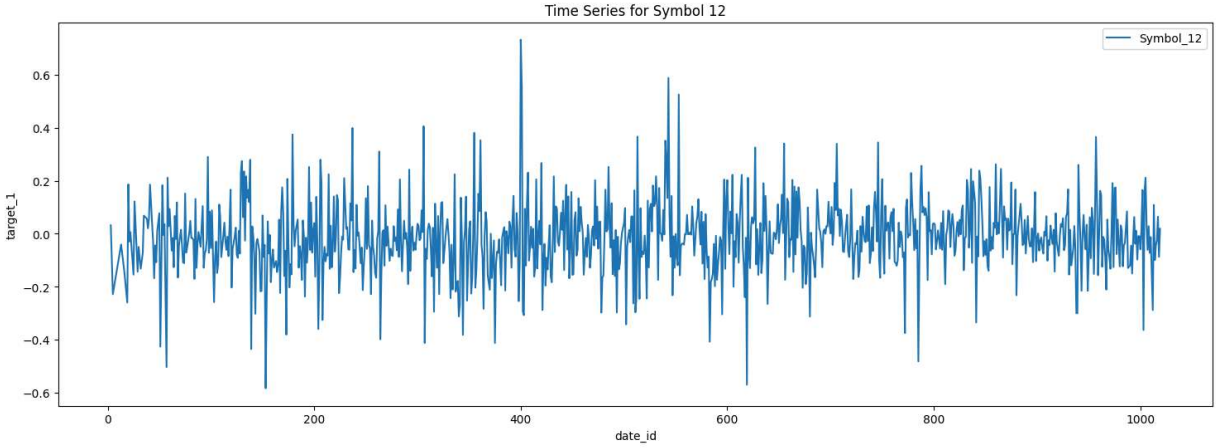

Time Series for Symbol 2
Symbol_2
target_1
date_id

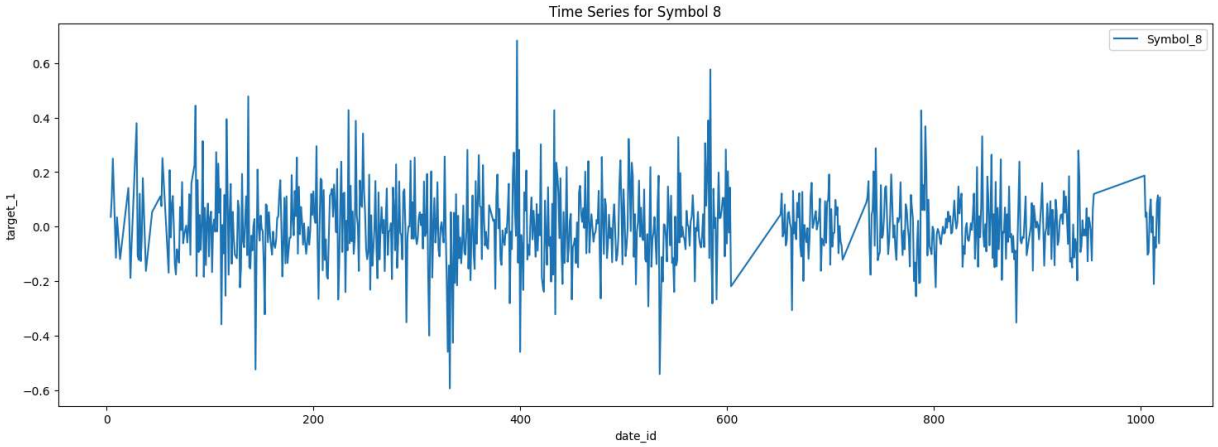

Time Series for Symbol 13
Symbol_13
target_1
date_id

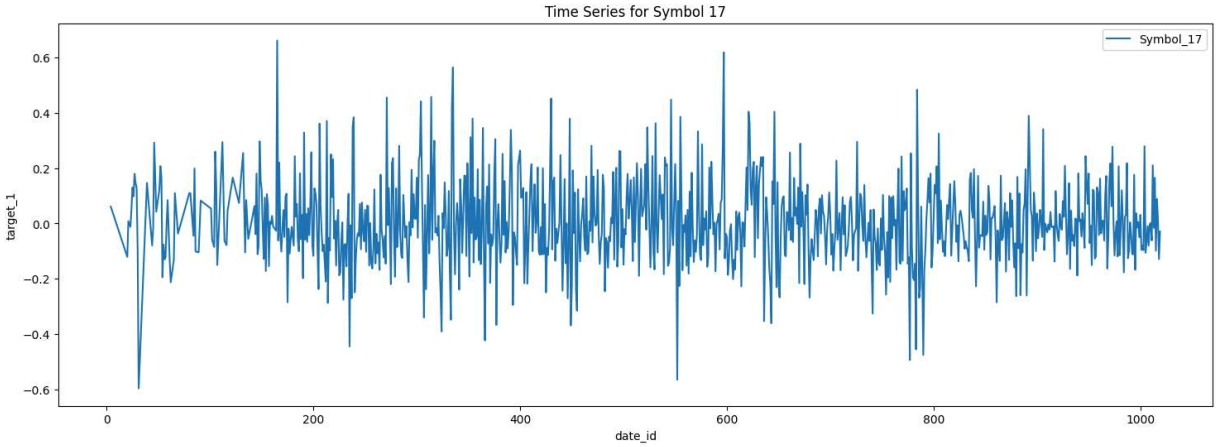

Time Series for Symbol 15
Symbol_15
target_1
date_id

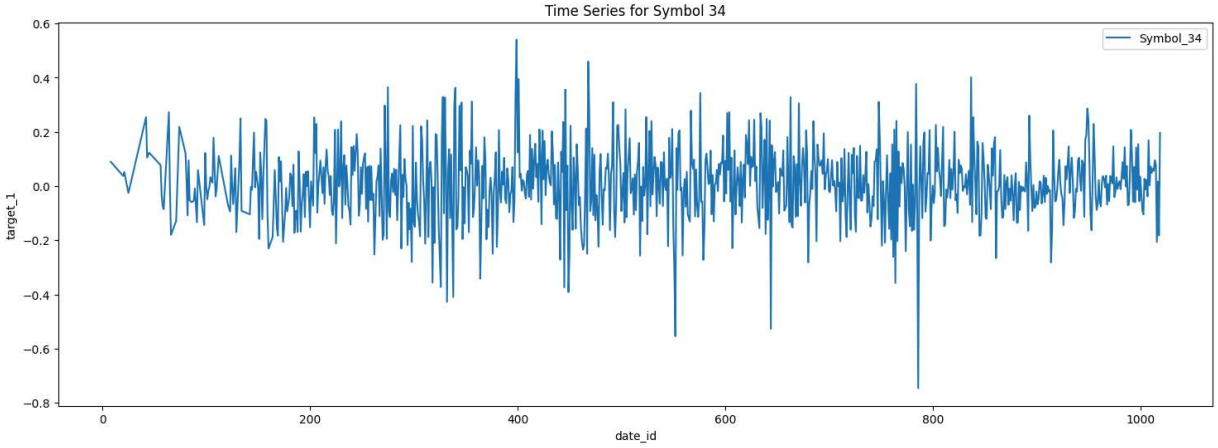

Time Series for Symbol 38
Symbol_38
target_1
date_id

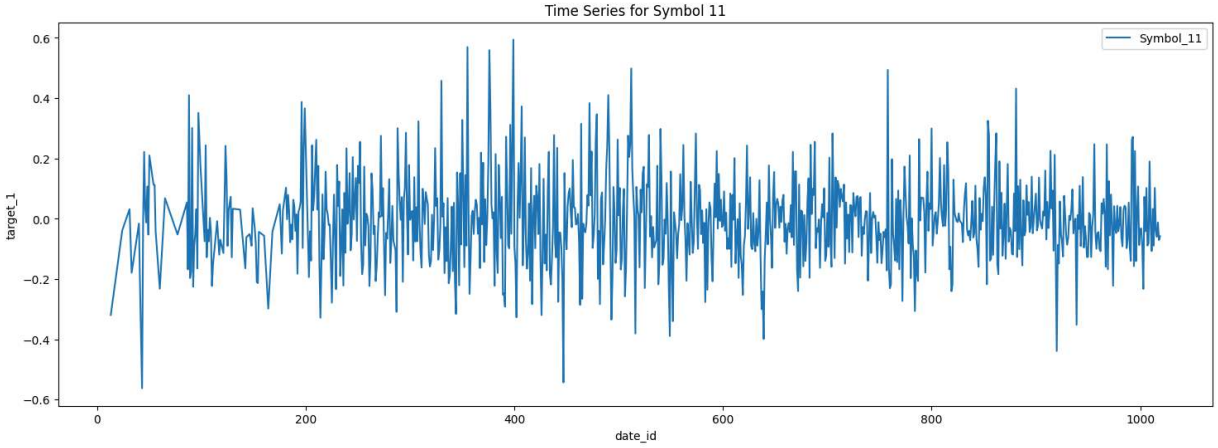

Time Series for Symbol 3
Symbol_3
target_1
date_id

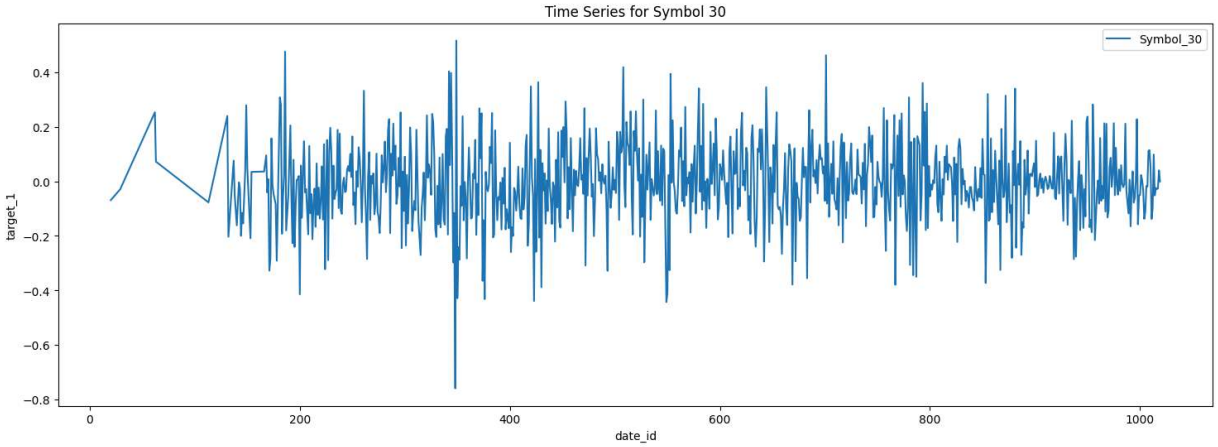

Time Series for Symbol 12
Symbol_12
target_1
date_id

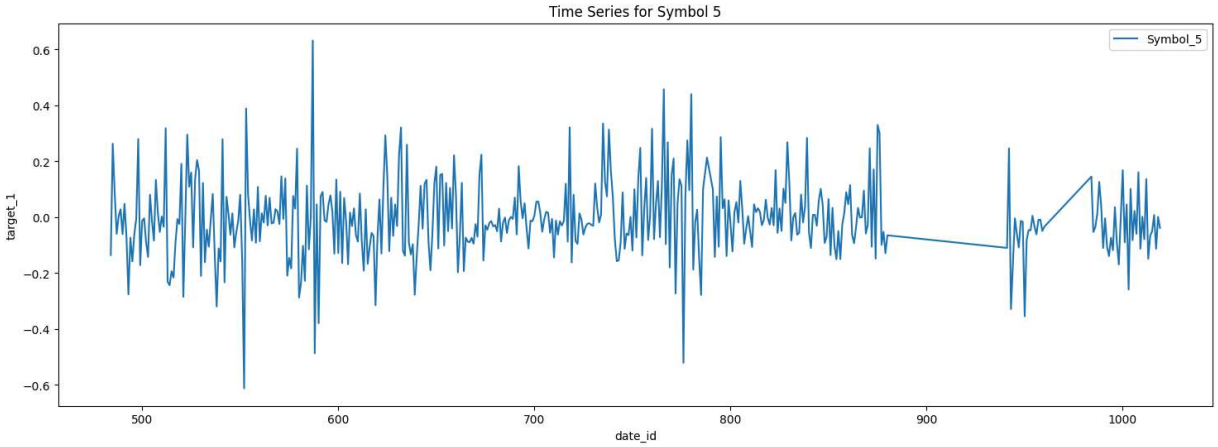

Time Series for Symbol 8
Symbol_8
target_1
date_id

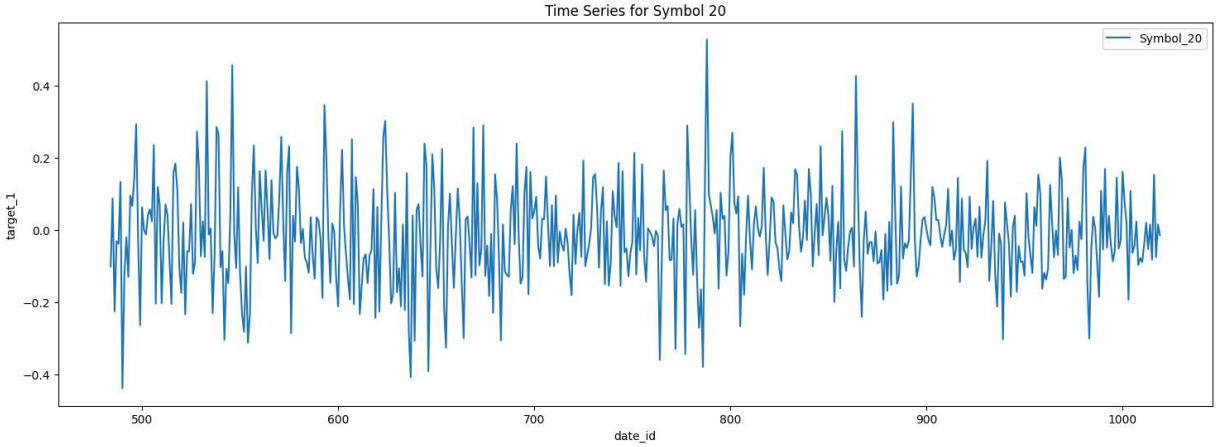

Time Series for Symbol 17
Symbol_17
target_1
date_id

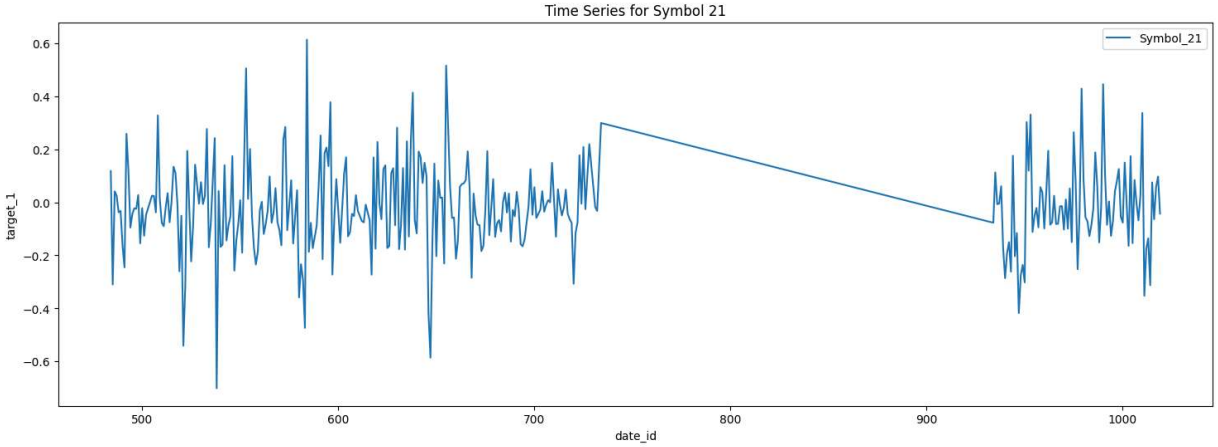

Time Series for Symbol 34
Symbol_34
target_1
date_id

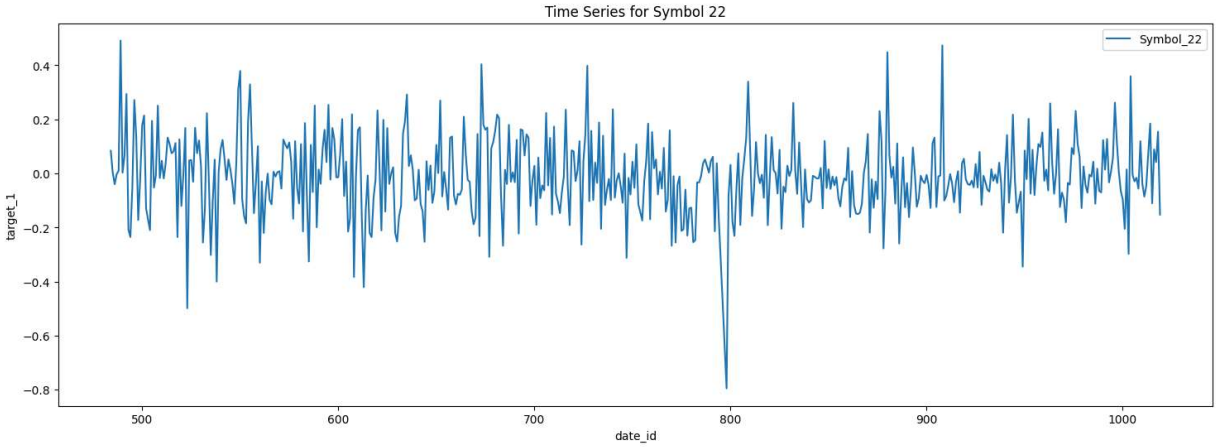

Time Series for Symbol 11
Symbol_11
target_1
date_id

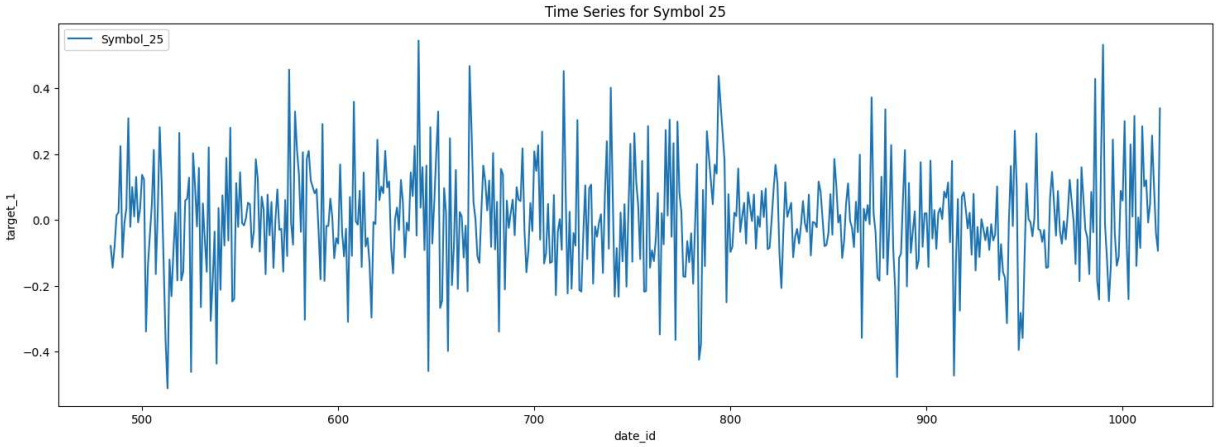

Time Series for Symbol 30
Symbol_30
target_1
date_id

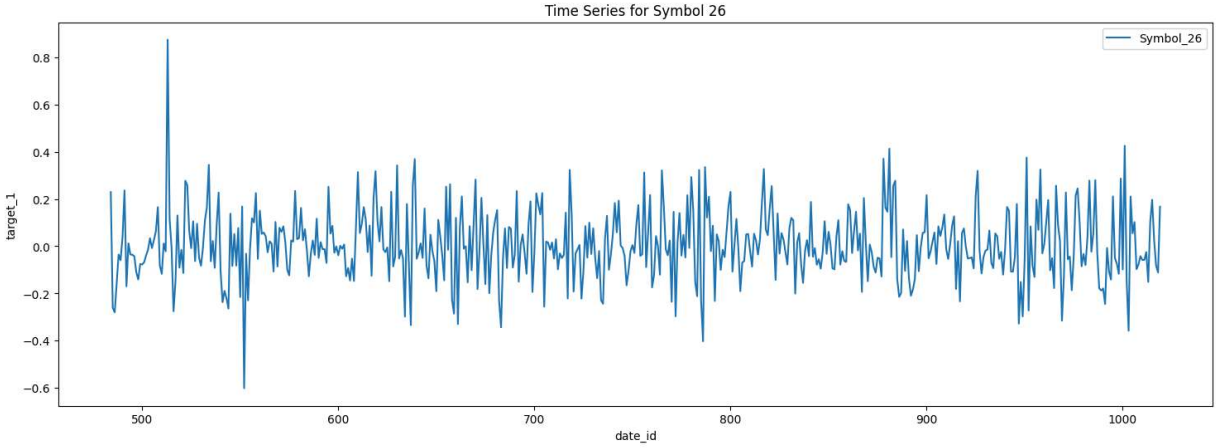

Time Series for Symbol 5
Symbol_5
target_1
date_id

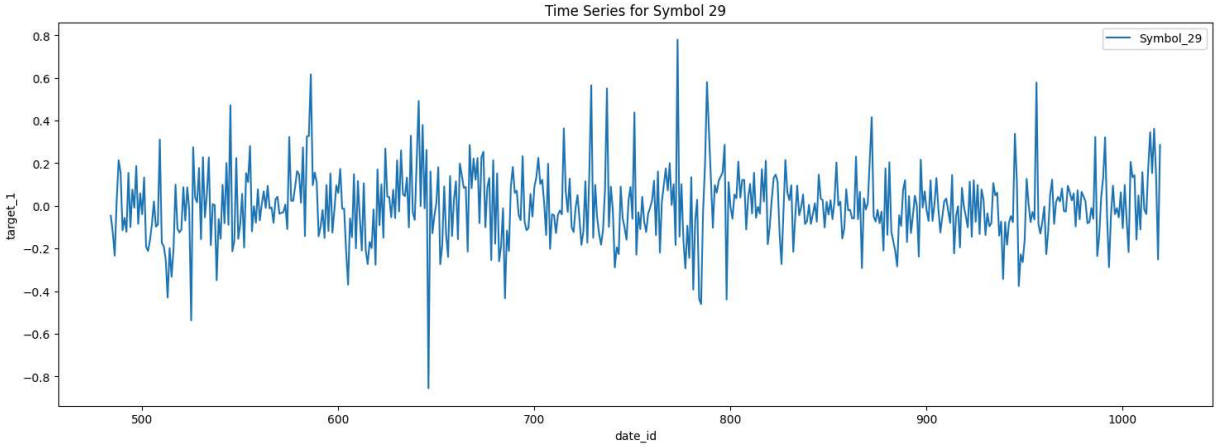

Time Series for Symbol 20
Symbol_20
target_1
date_id

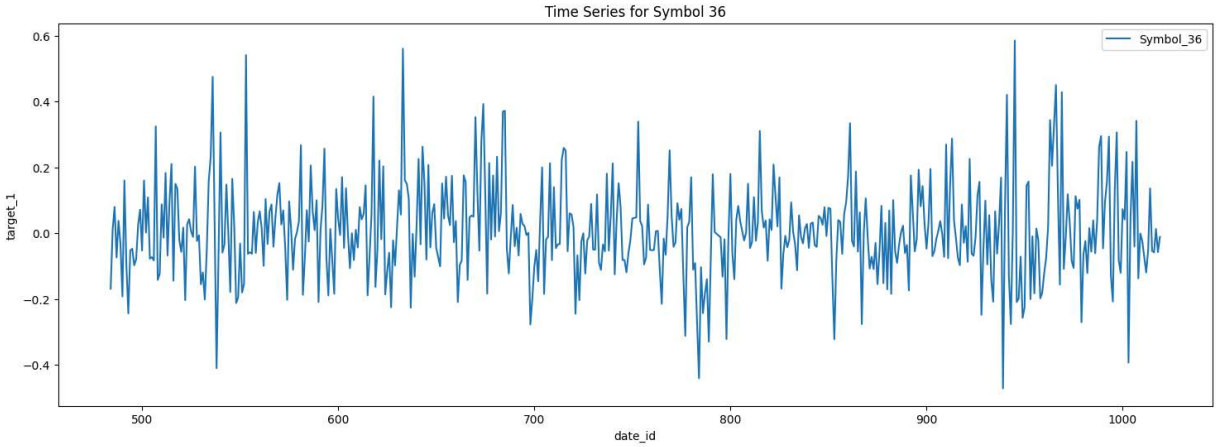

Time Series for Symbol 21
Symbol_21
target_1
date_id

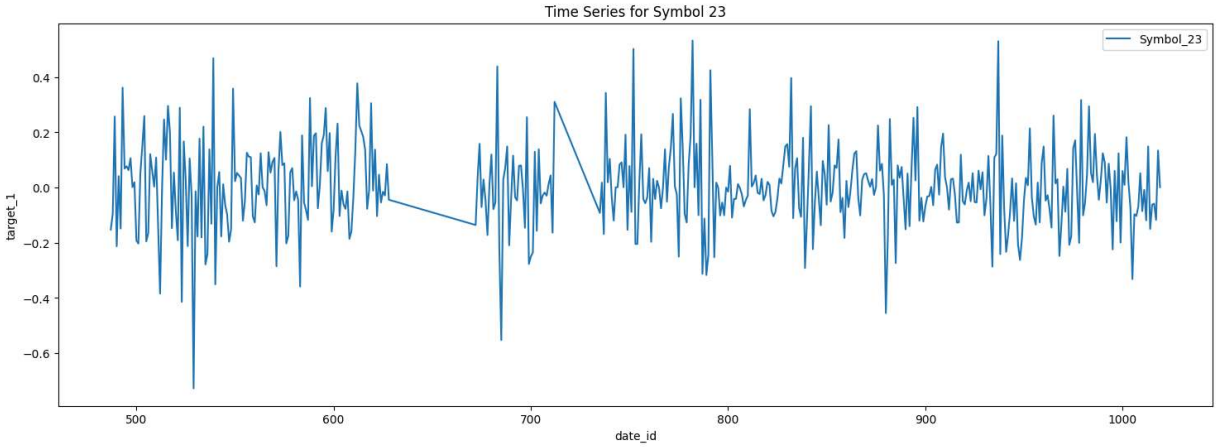

Time Series for Symbol 22
Symbol_22
target_1
date_id

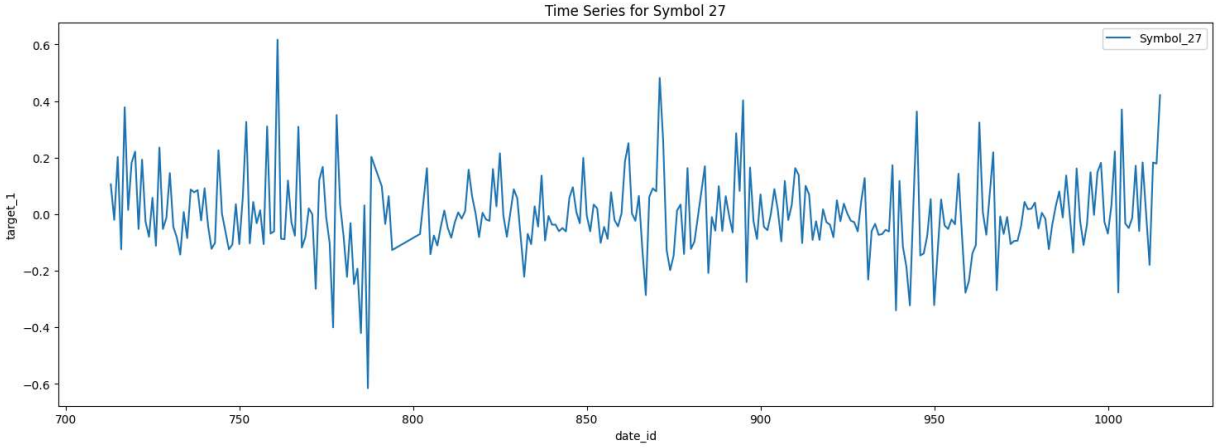

Time Series for Symbol 25
Symbol_25
target_1
date_id

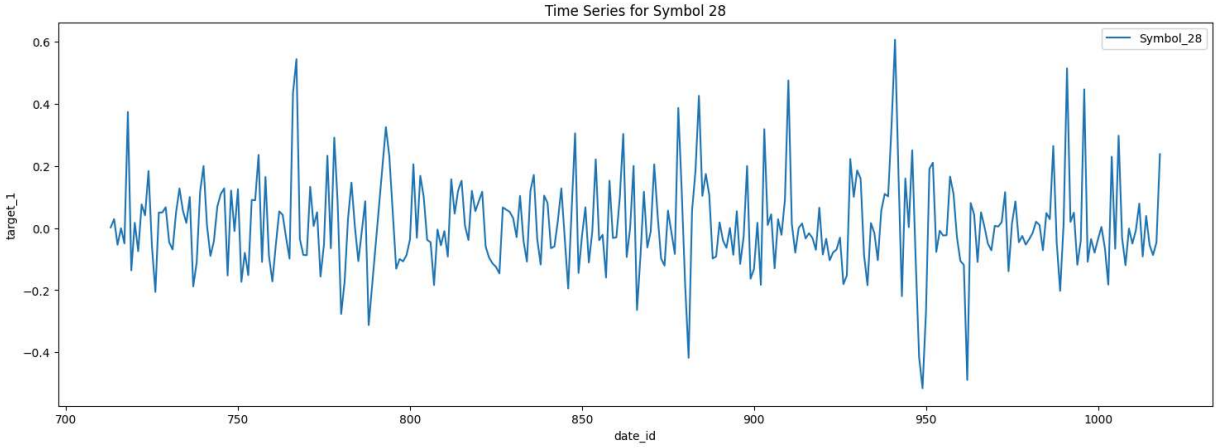

Time Series for Symbol 26
Symbol_26
target_1
date_id

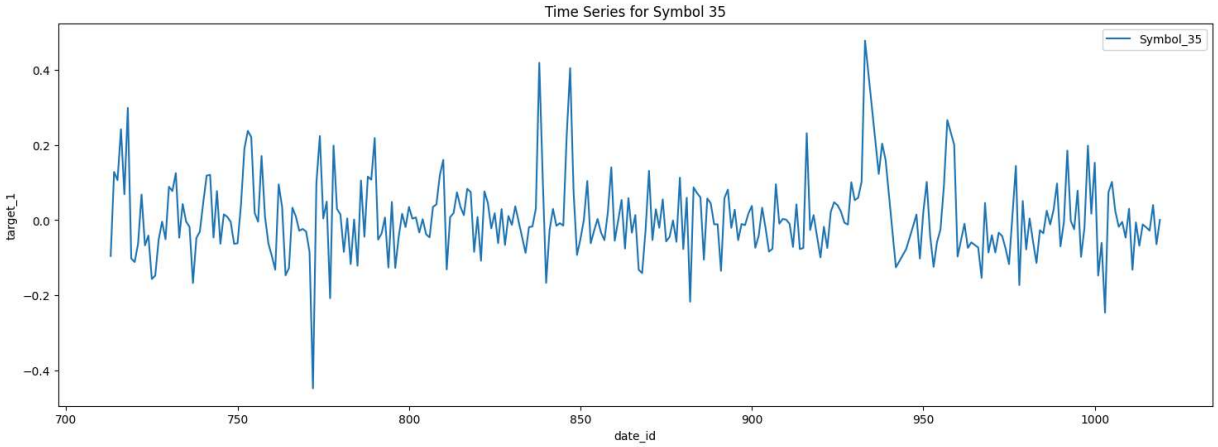

Time Series for Symbol 29
Symbol_29
target_1
date_id

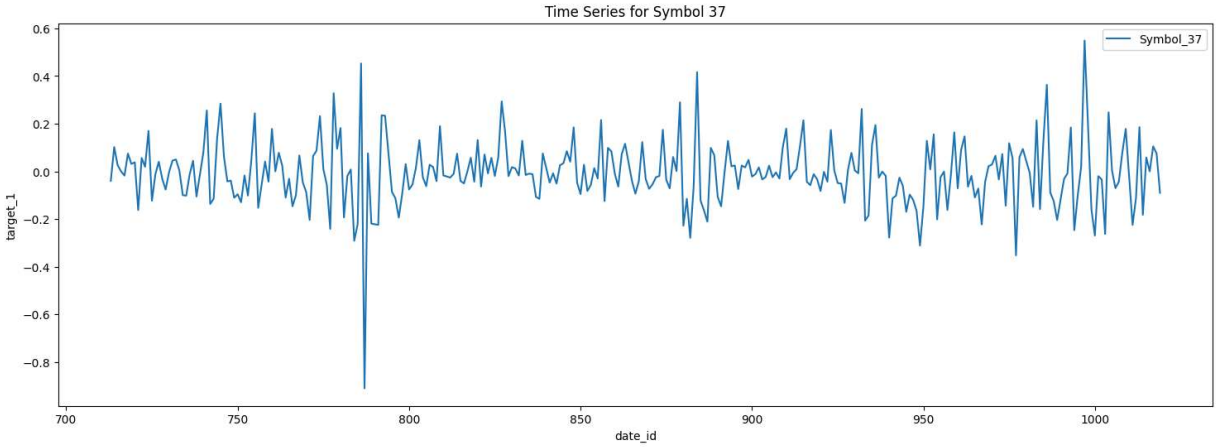

Time Series for Symbol 36
Symbol_36
target_1
date_id

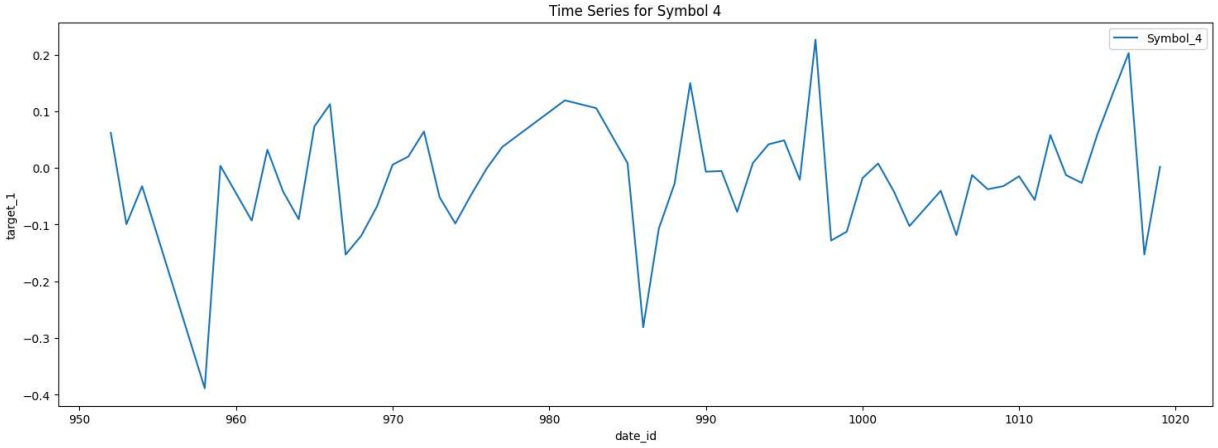

Time Series for Symbol 23
Symbol_23
target_1
date_id

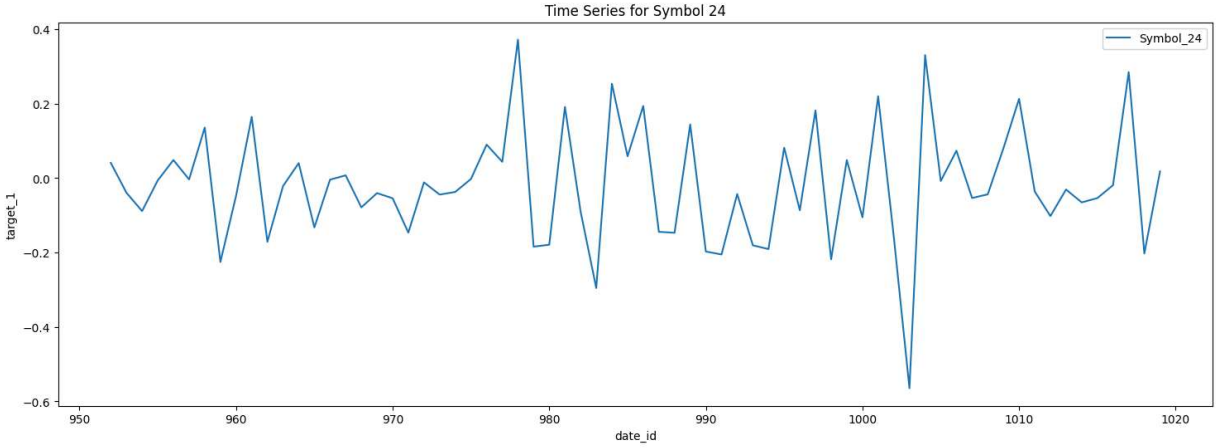

Time Series for Symbol 27
Symbol_27
target_1
date_id

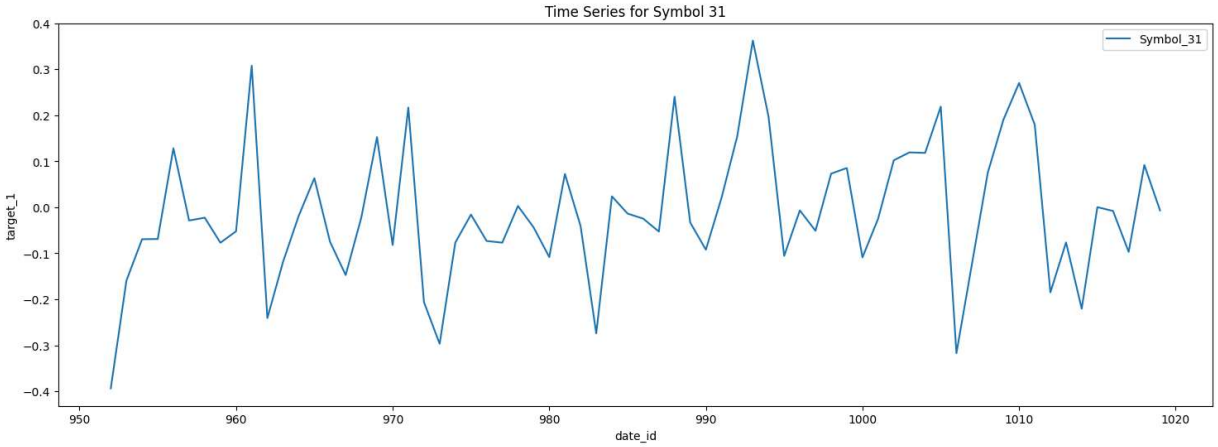

Time Series for Symbol 28
Symbol_28
target_1
date_id


Time Series for Symbol 35
Symbol_35
target_1
date_id

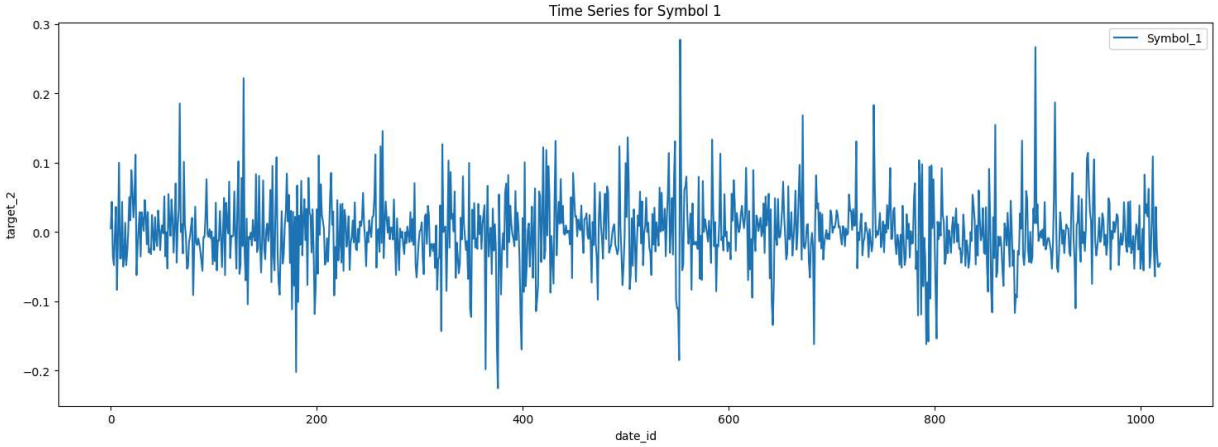

Time Series for Symbol 37
Symbol_37
target_1
date_id

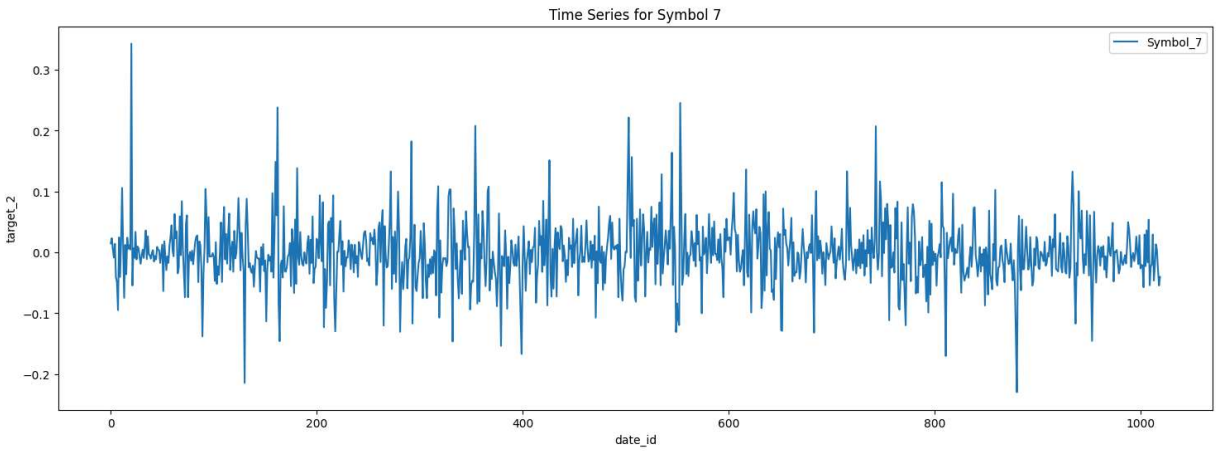

Time Series for Symbol 4
Symbol_4
target_1
date_id

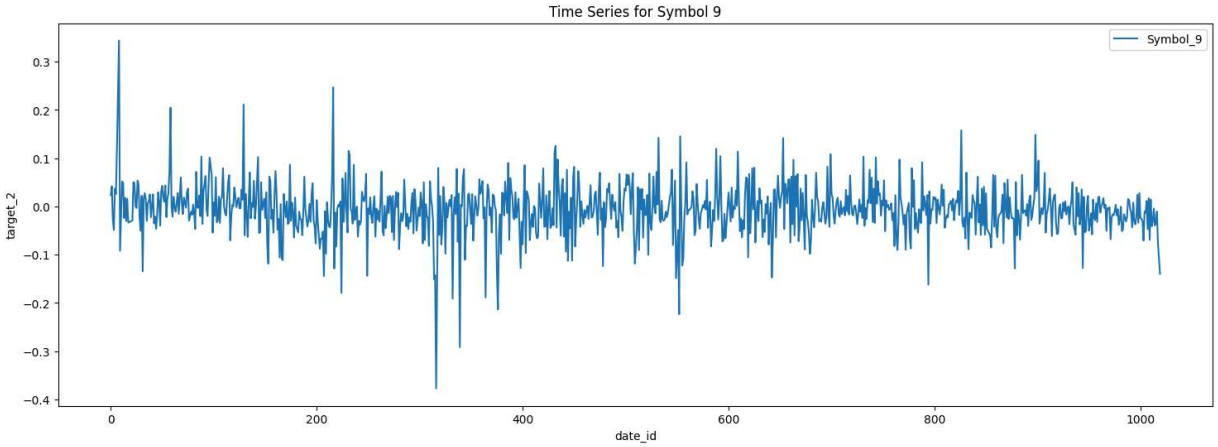

Time Series for Symbol 24
Symbol_24
target_1
date_id

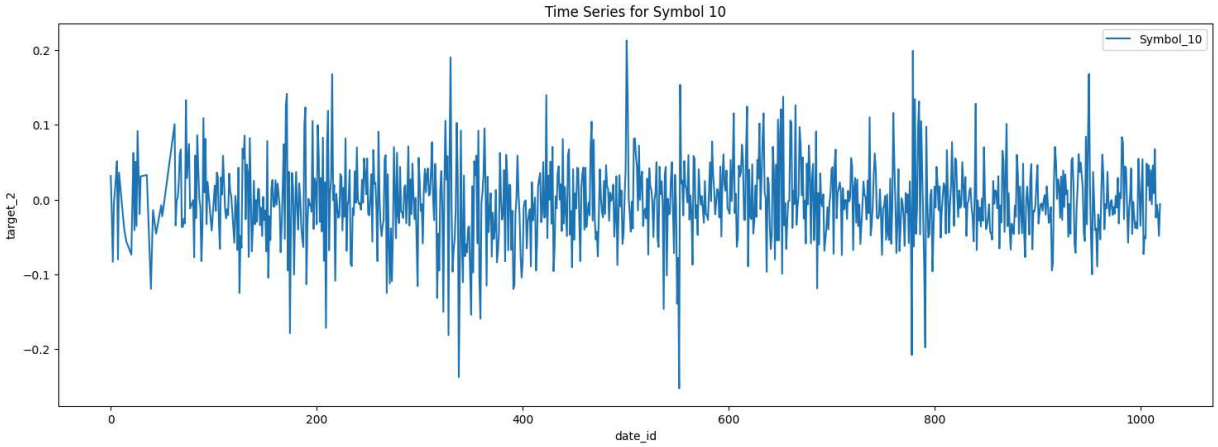


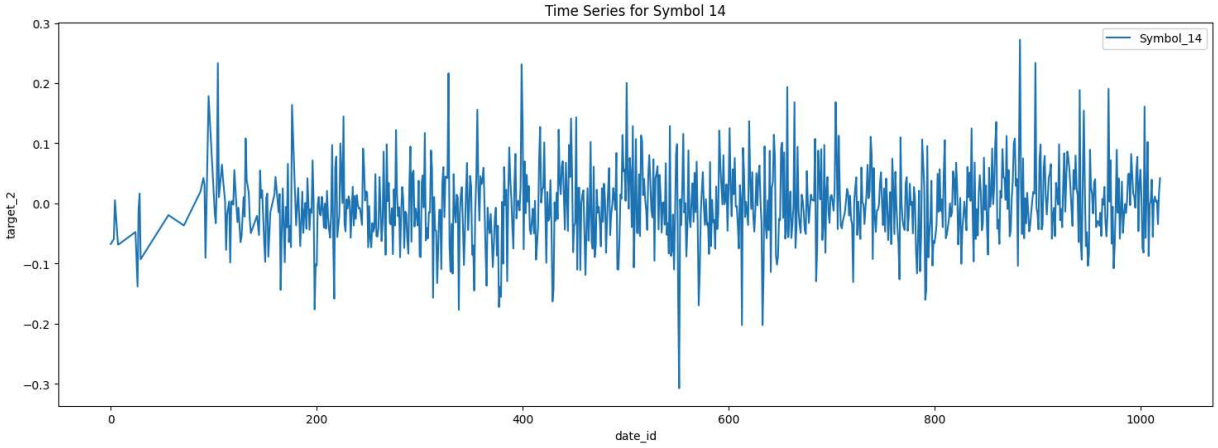
In [25]:
```
daily_avg = train_file.groupby(['date_id','symbol_id'])['target_2'].mean().reset_in

for symbol_id in train_file['symbol_id'].unique():
    symbol_data = daily_avg[daily_avg['symbol_id']==symbol_id]
    plt.figure(figsize=(18,6))
    plt.plot(symbol_data['date_id'], symbol_data['target_2'], label = f"Symbol_{sym
    plt.title(f"Time Series for Symbol {symbol_id}")
    plt.xlabel("date_id")
    plt.ylabel("target_2")
    plt.legend()
    plt.show()
```

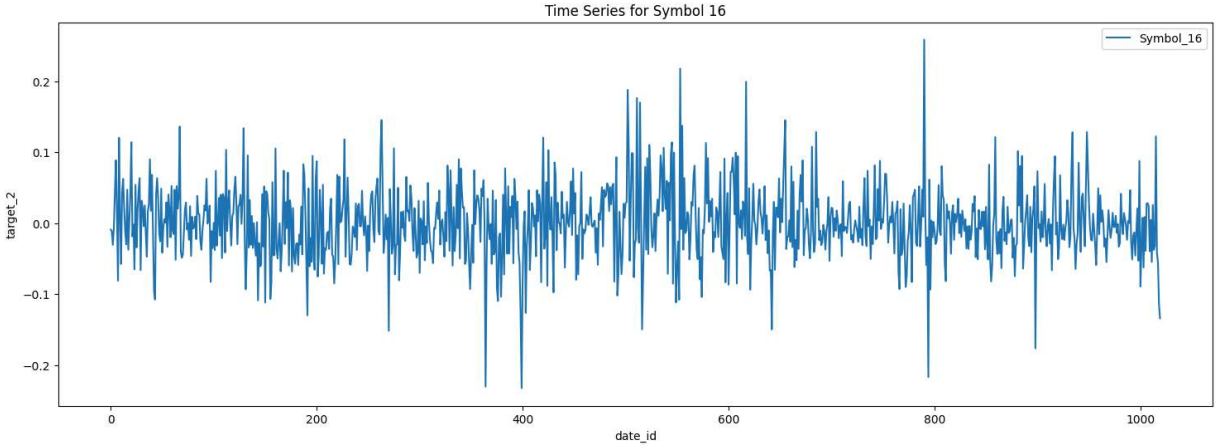


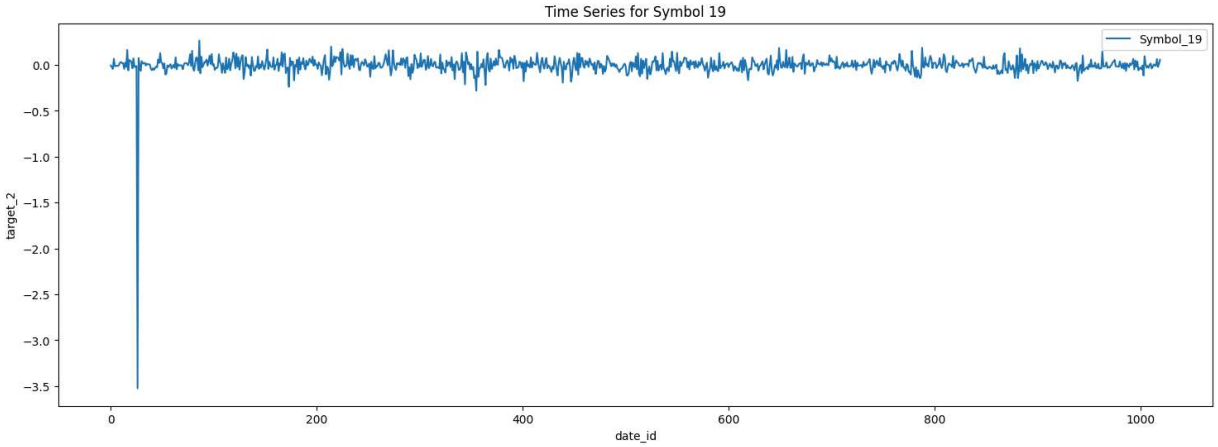


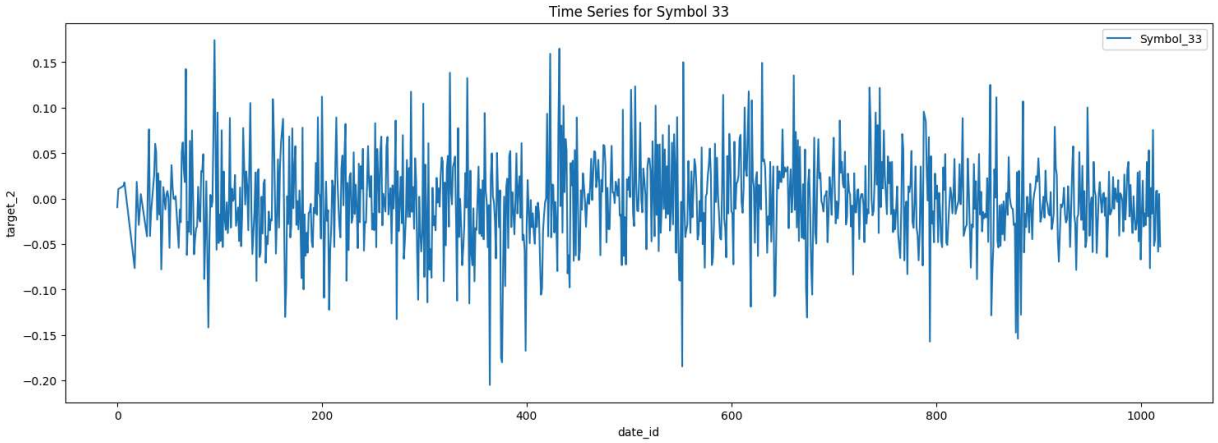

Time Series for Symbol 9
Symbol_9
target_2
date_id

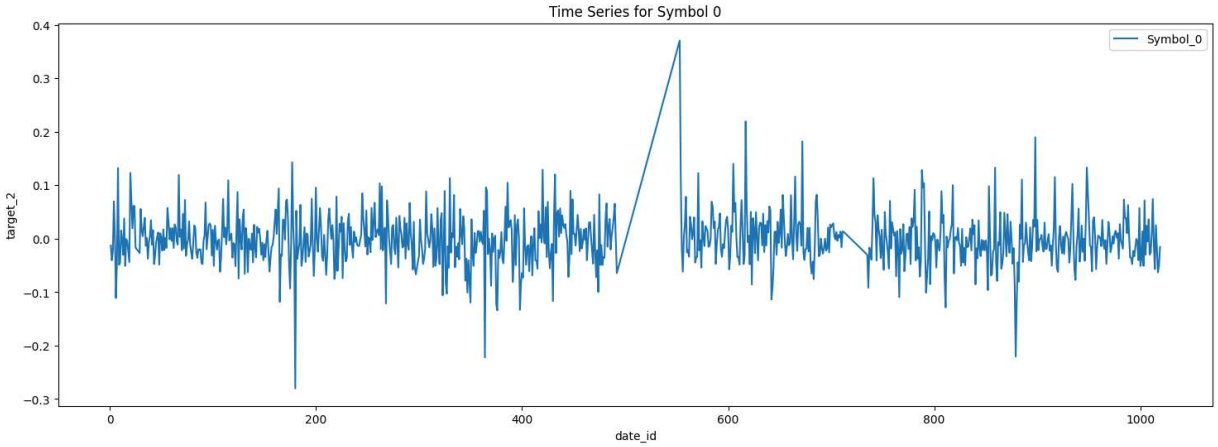

Time Series for Symbol 10
Symbol_10
target_2
date_id

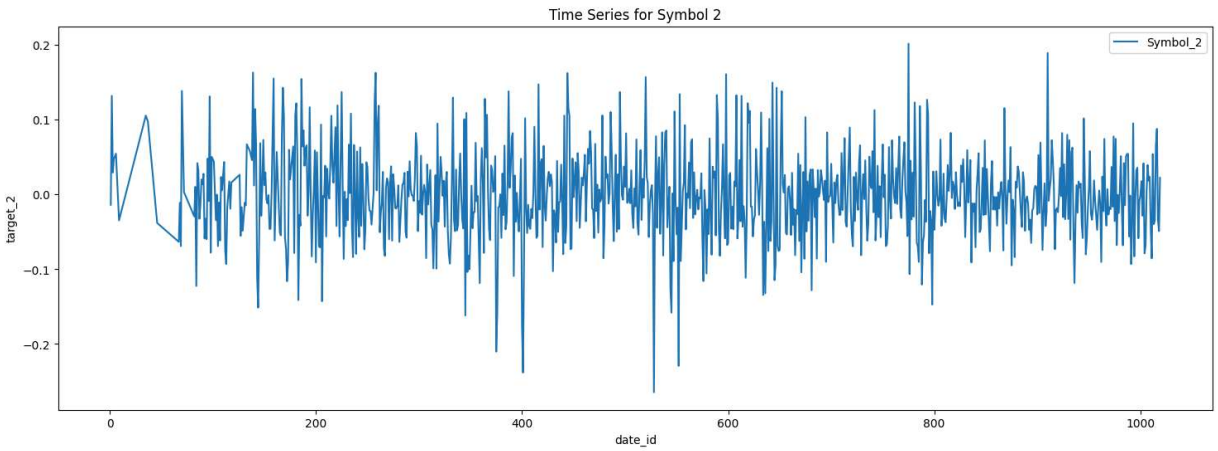

Time Series for Symbol 14
Symbol_14
target_2
date_id

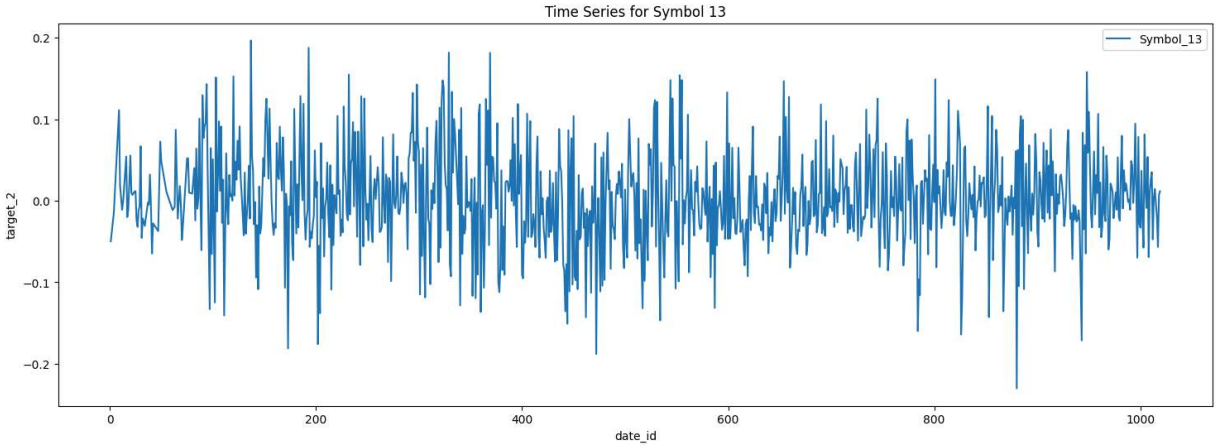

Time Series for Symbol 16
Symbol_16
target_2
date_id

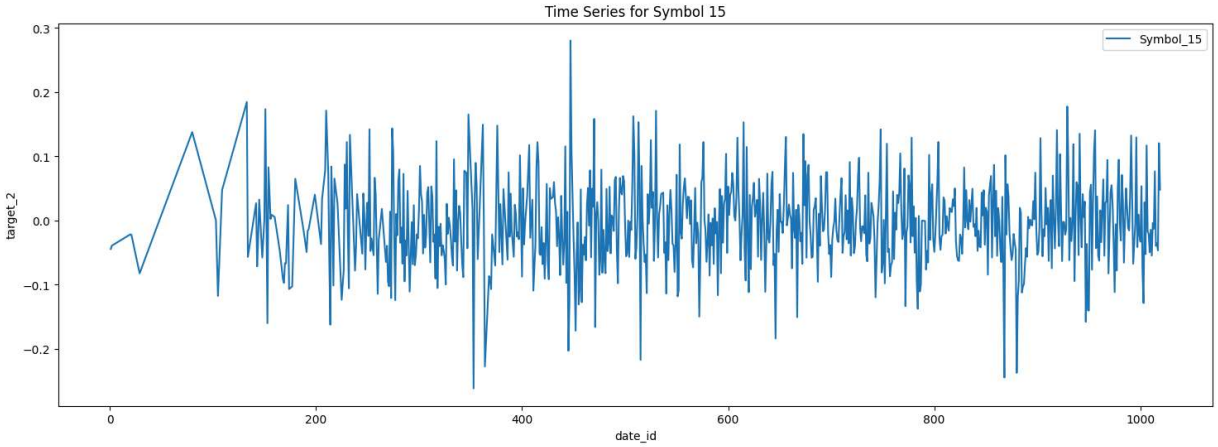

Time Series for Symbol 19
Symbol_19
target_2
date_id

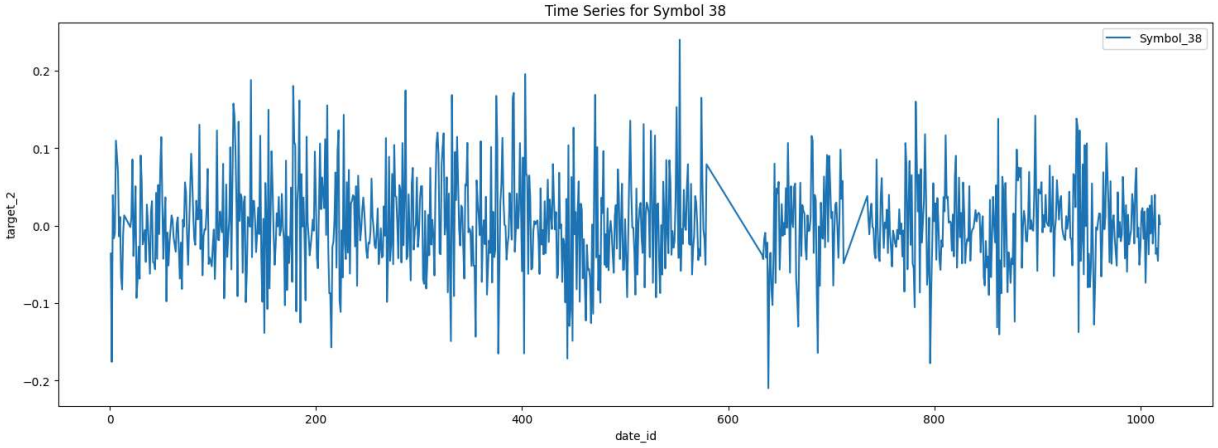

Time Series for Symbol 33
Symbol_33
target_2
date_id

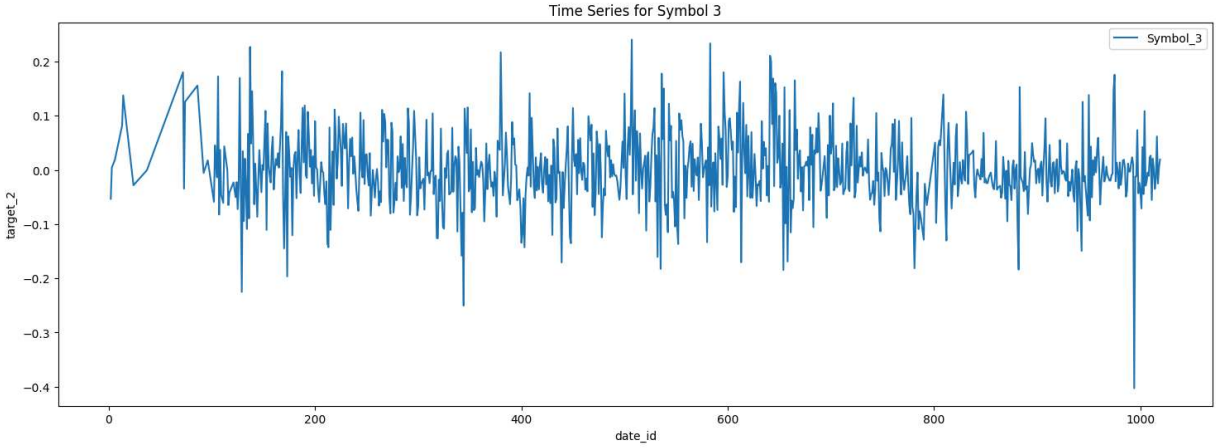

Time Series for Symbol 0
Symbol_0
target_2
date_id

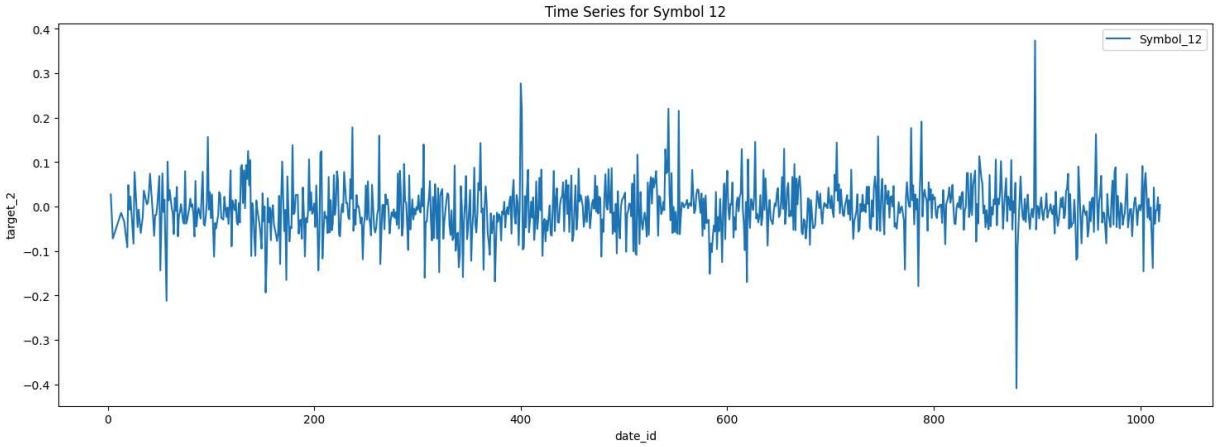

Time Series for Symbol 2
Symbol_2
target_2
date_id

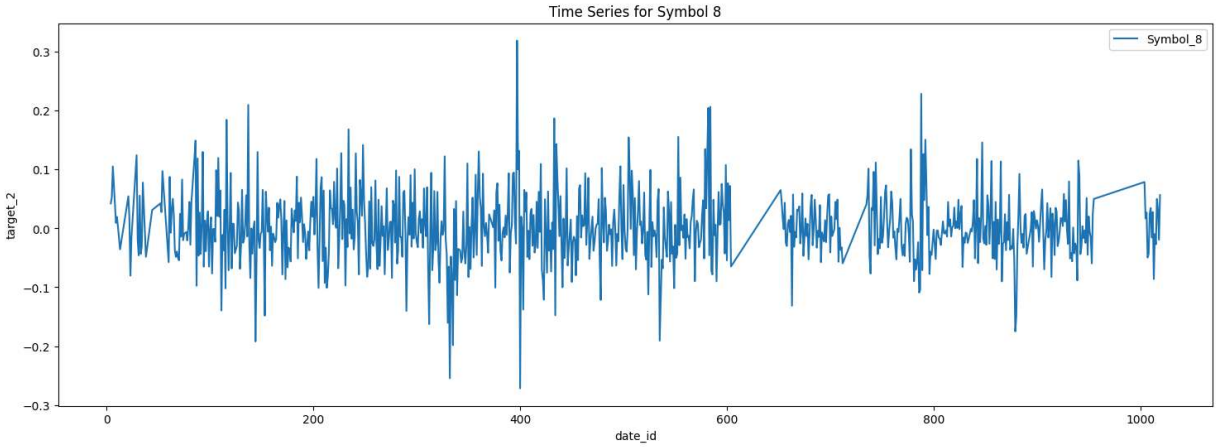

Time Series for Symbol 13
Symbol_13
target_2
date_id

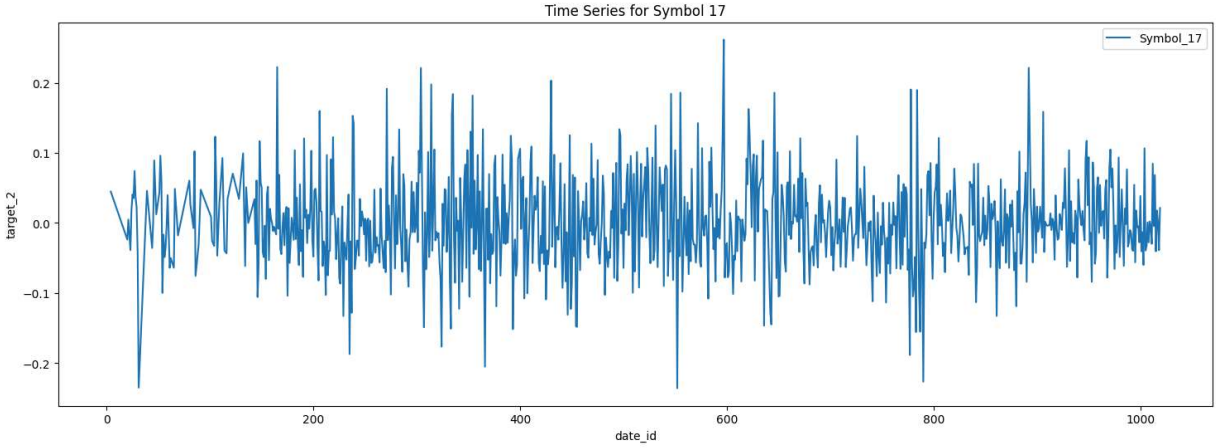

Time Series for Symbol 15
Symbol_15
target_2
date_id

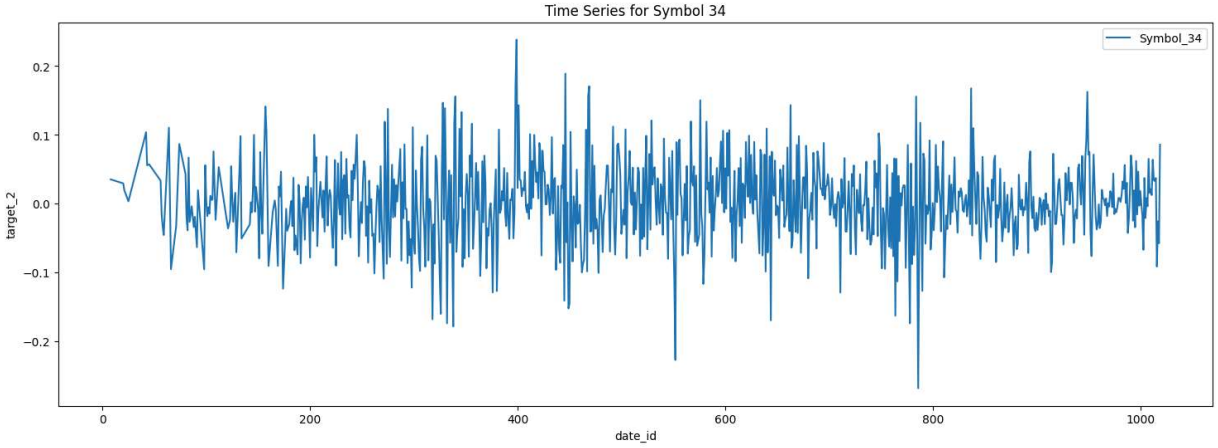

Time Series for Symbol 38
Symbol_38
target_2
date_id

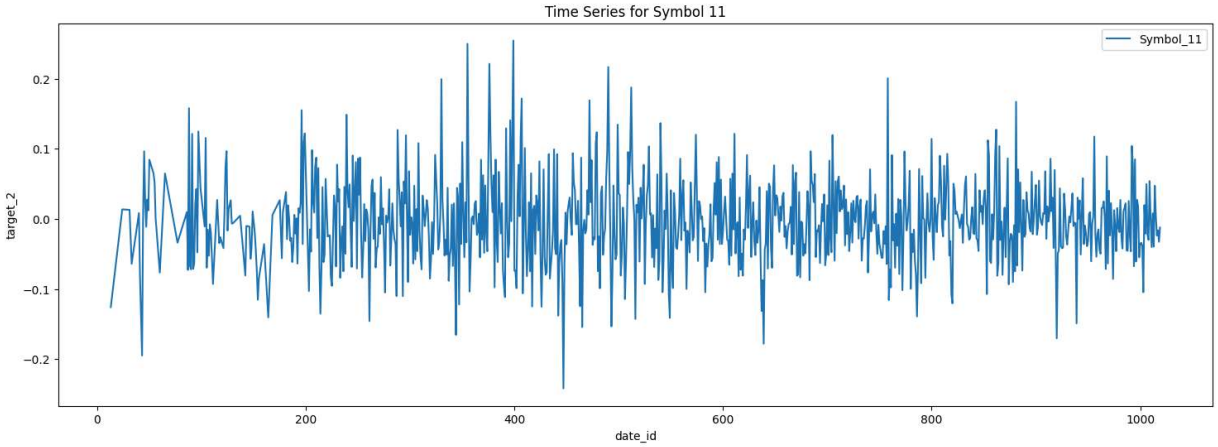

Time Series for Symbol 3
Symbol_3
target_2
date_id

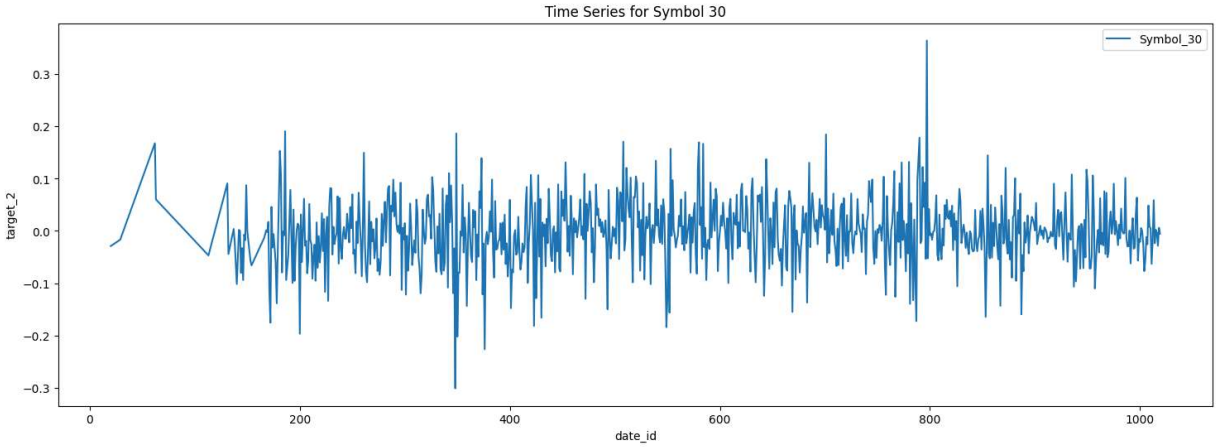

Time Series for Symbol 12
Symbol_12
target_2
date_id

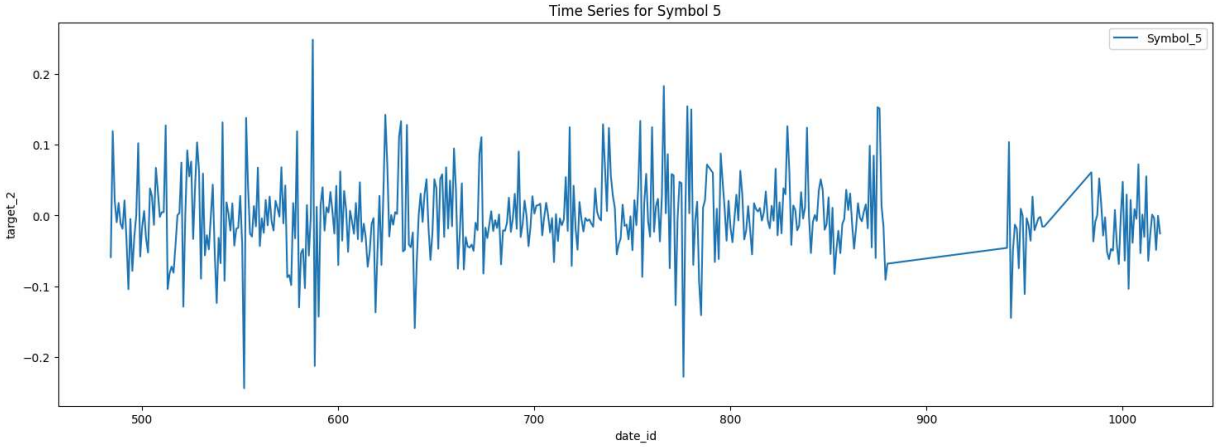

Time Series for Symbol 8
Symbol_8
target_2
date_id

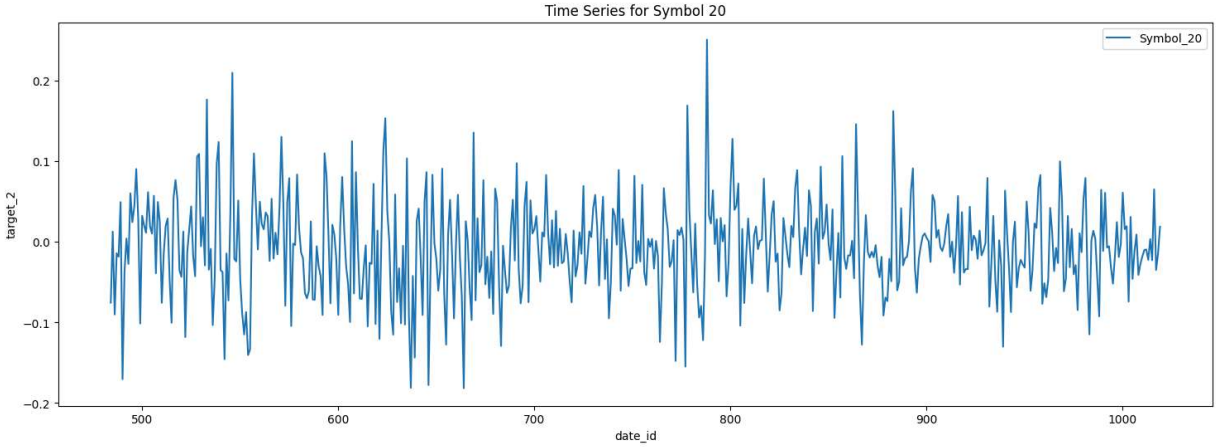

Time Series for Symbol 17
Symbol_17
target_2
date_id

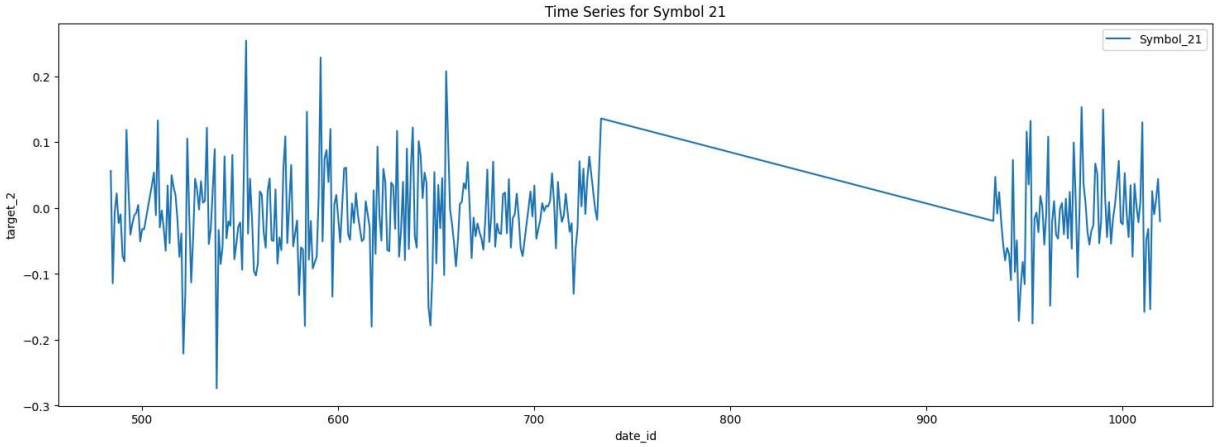

Time Series for Symbol 34
Symbol_34
target_2
date_id

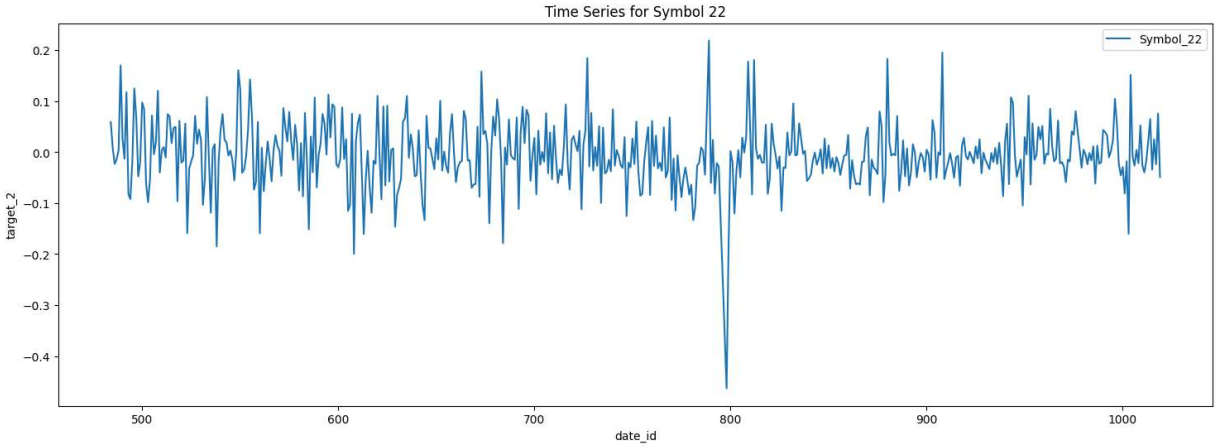

Time Series for Symbol 11
Symbol_11
target_2
date_id

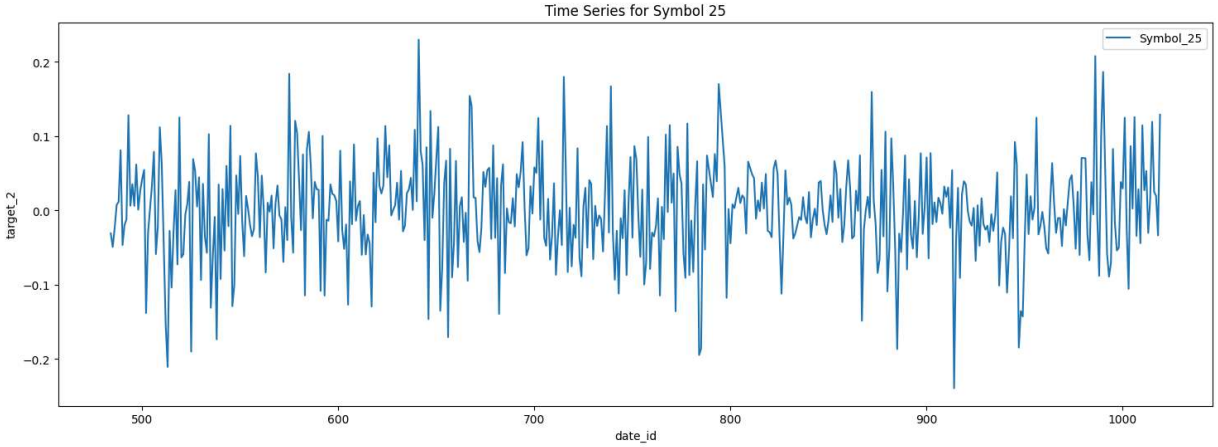

Time Series for Symbol 30
Symbol_30
target_2
date_id

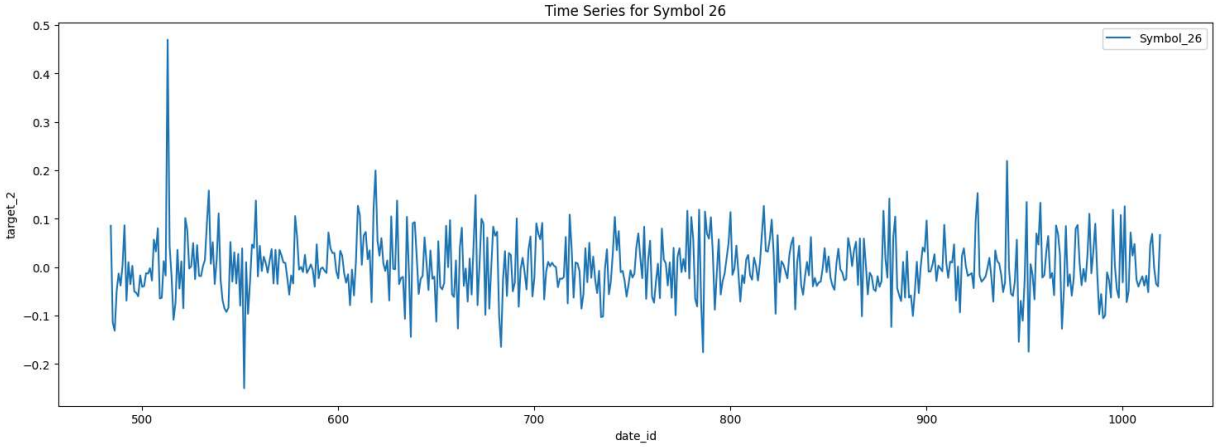

Time Series for Symbol 5
Symbol_5
target_2
date_id

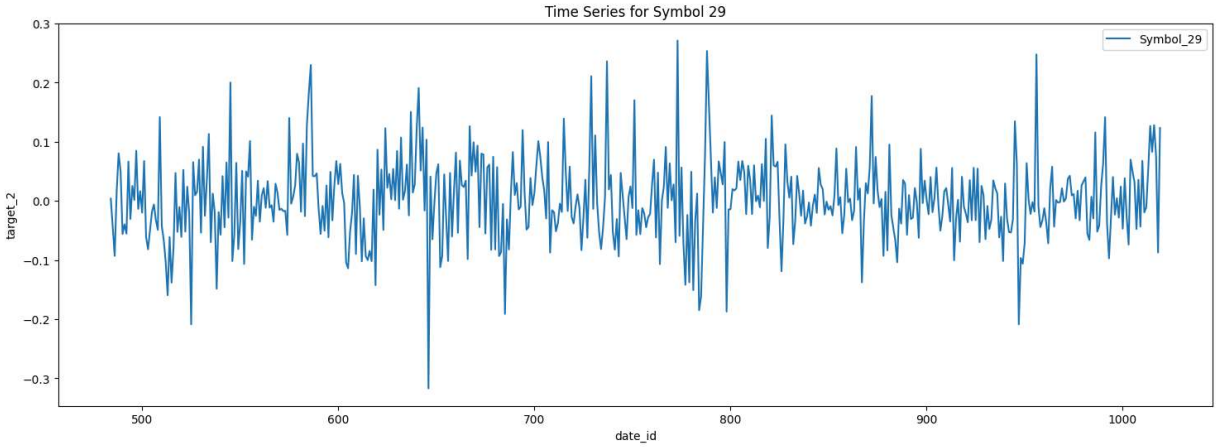

Time Series for Symbol 20
Symbol_20
target_2
date_id

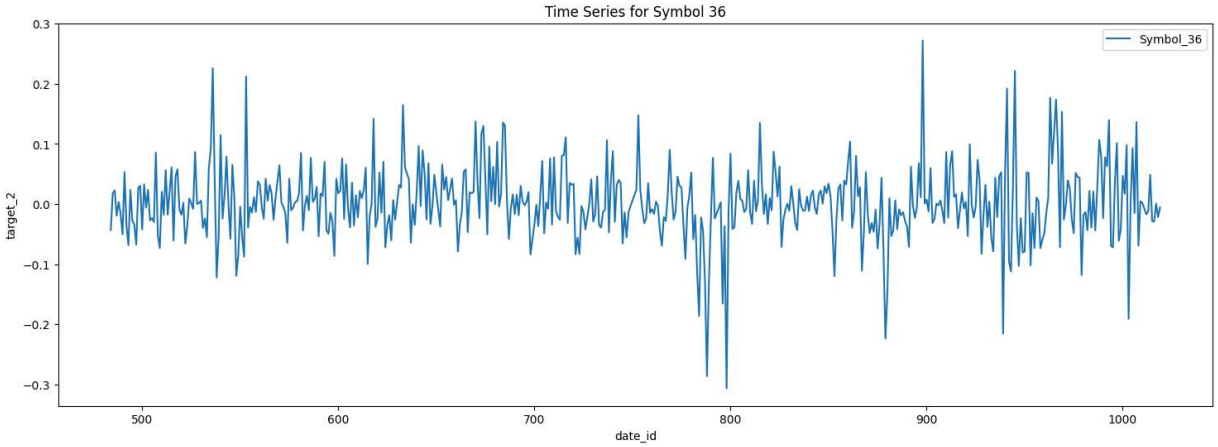

Time Series for Symbol 21
Symbol_21
target_2
date_id

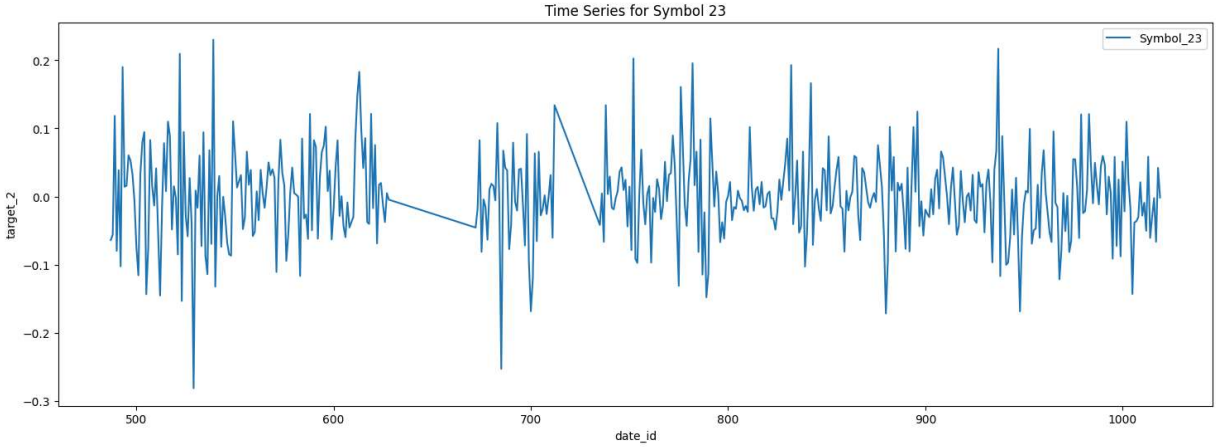

Time Series for Symbol 22
Symbol_22
target_2
date_id

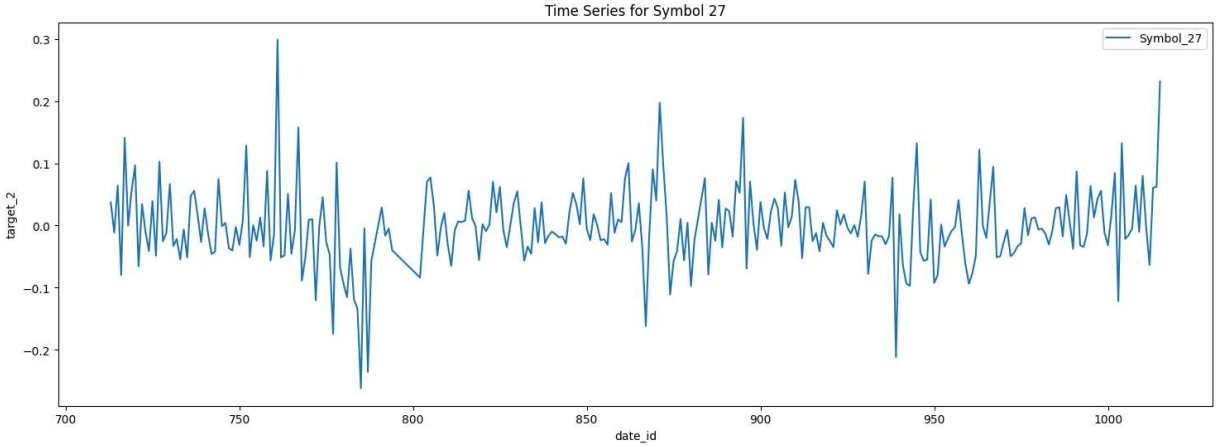

Time Series for Symbol 25
Symbol_25
target_2
date_id

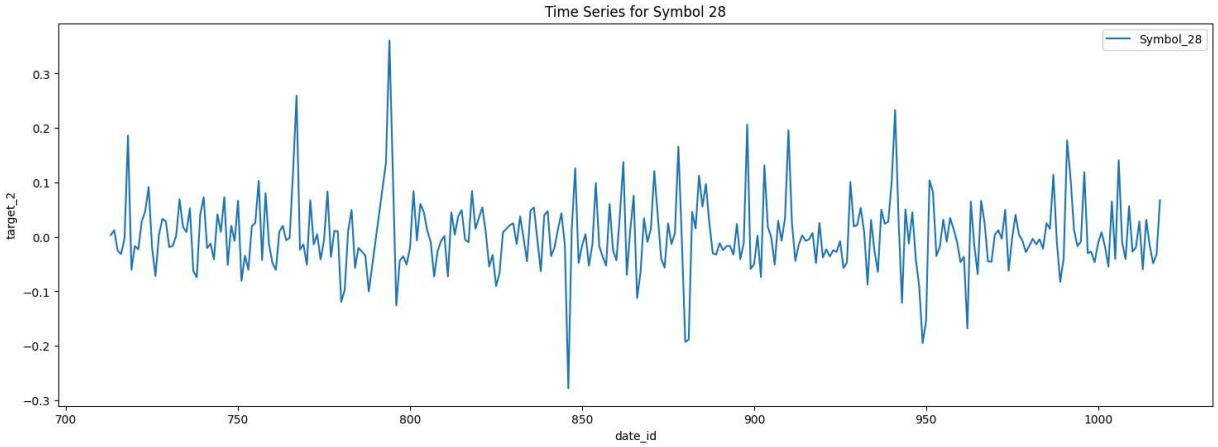

Time Series for Symbol 26
Symbol_26
target_2
date_id

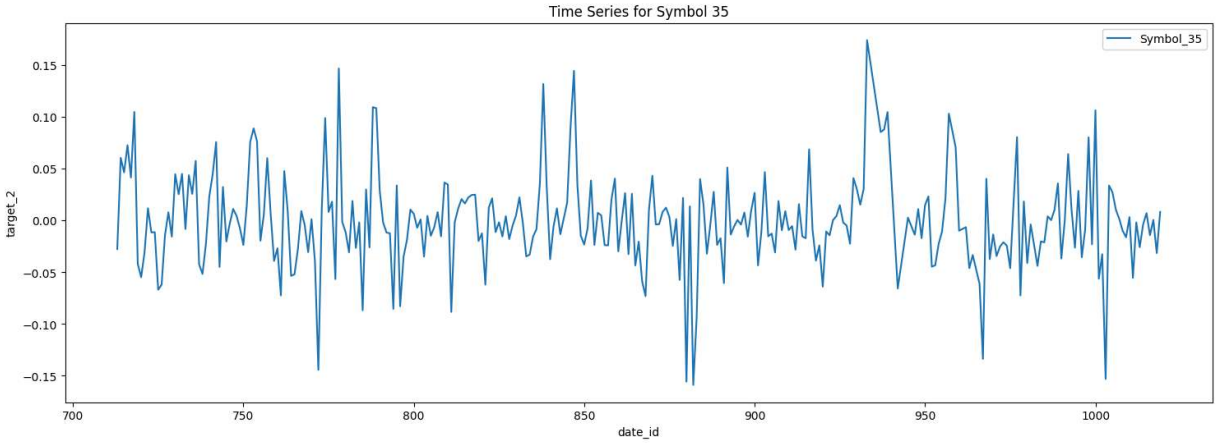

Time Series for Symbol 29
Symbol_29
target_2
date_id

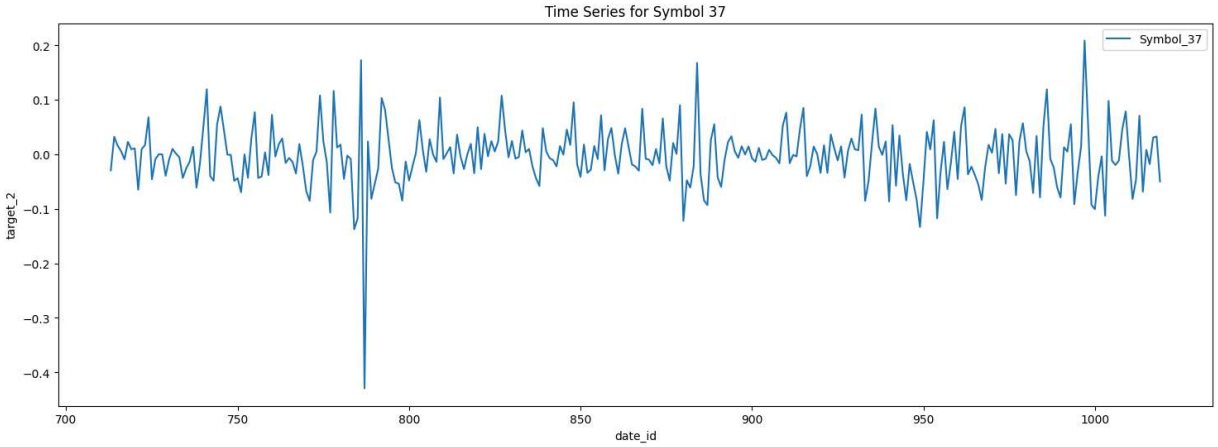

Time Series for Symbol 36
Symbol_36
target_2
date_id

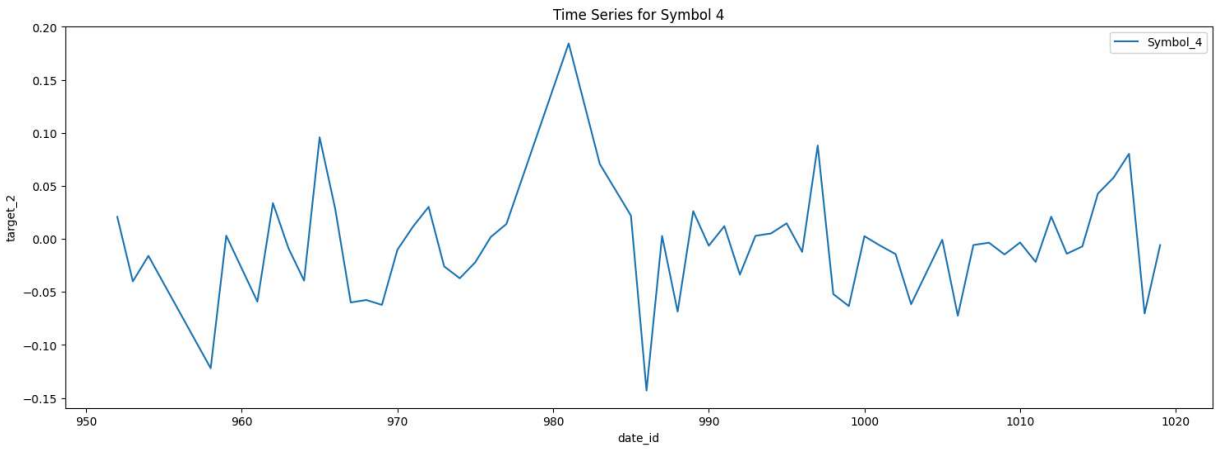

Time Series for Symbol 23
Symbol_23
target_2
date_id

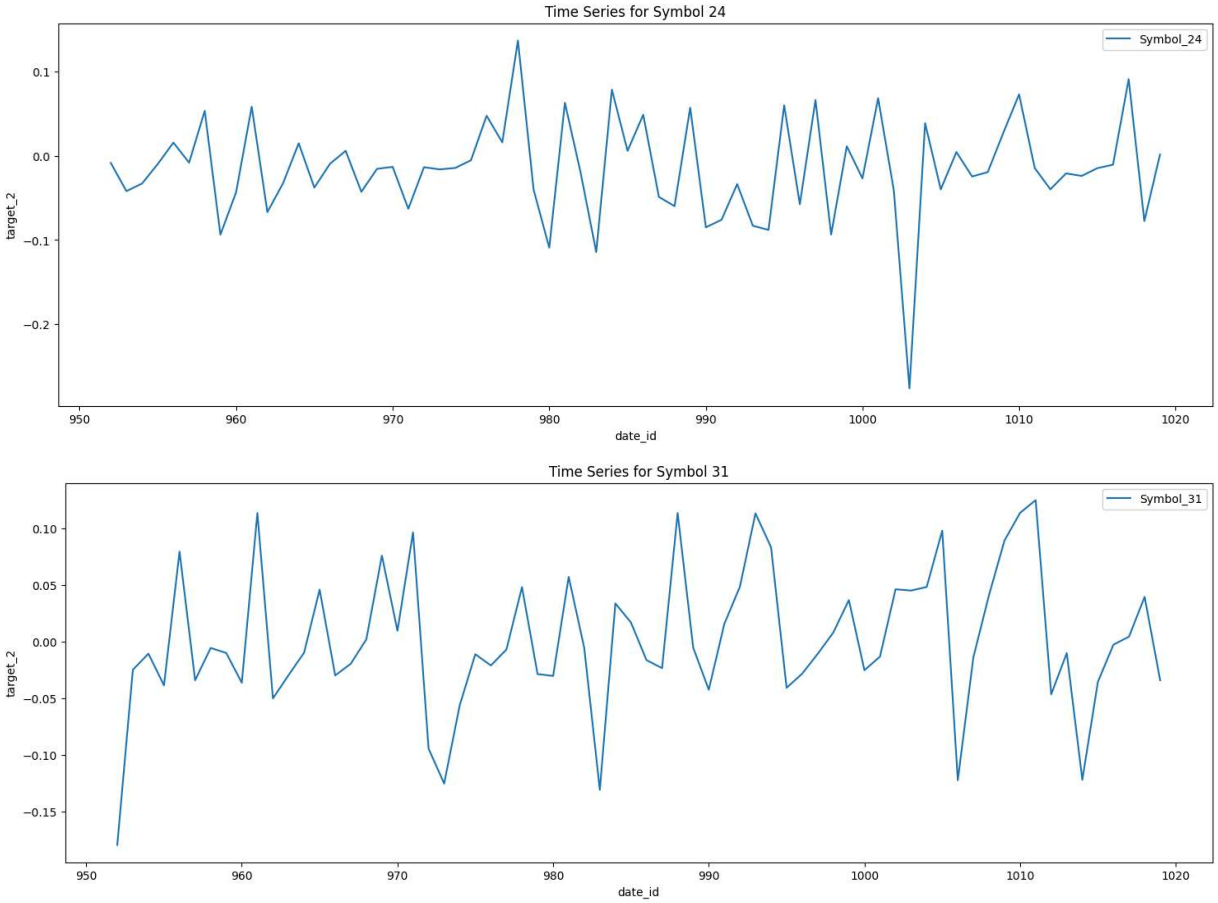

Time Series for Symbol 27
Symbol_27
target_2
date_id


Time Series for Symbol 28
Symbol_28
target_2
date_id


Time Series for Symbol 35
Symbol_35
target_2
date_id


Time Series for Symbol 37
Symbol_37
target_2
date_id

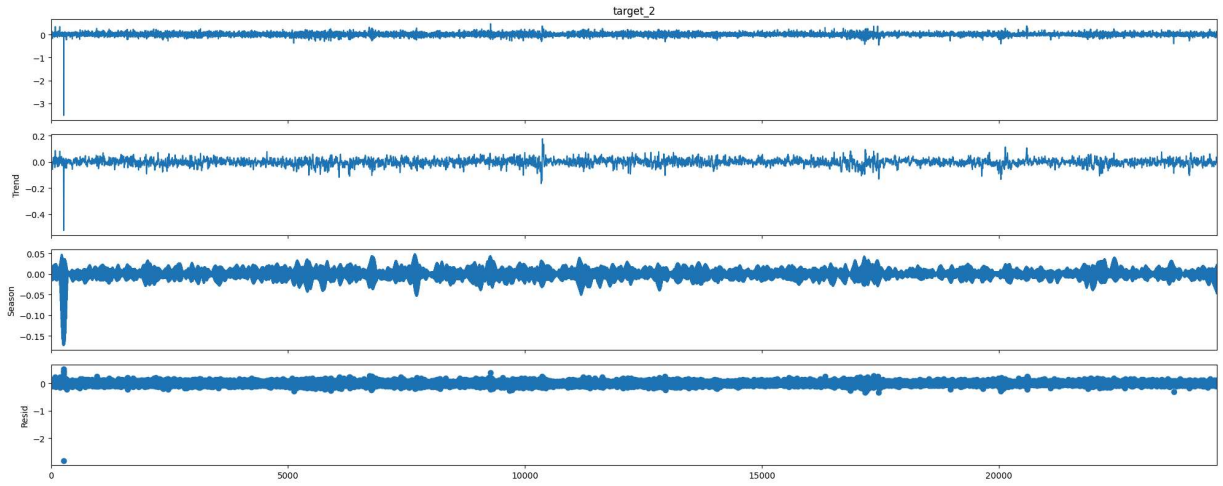

Time Series for Symbol 4
Symbol_4
target_2
date_id



The time series plot shows the average value of target over time for each symbol.

This helps identify trends, seasonality, or anomalies in the data over time. It's crucial for understanding the temporal behavior of the target variable.

In [27]:
```
from statsmodels.tsa.seasonal import STL

stl = STL(daily_avg['target_2'], seasonal = 31, period = 7)
result = stl.fit()
fig = result.plot()
fig.set_size_inches(20,8)
plt.tight_layout()
plt.show()
```

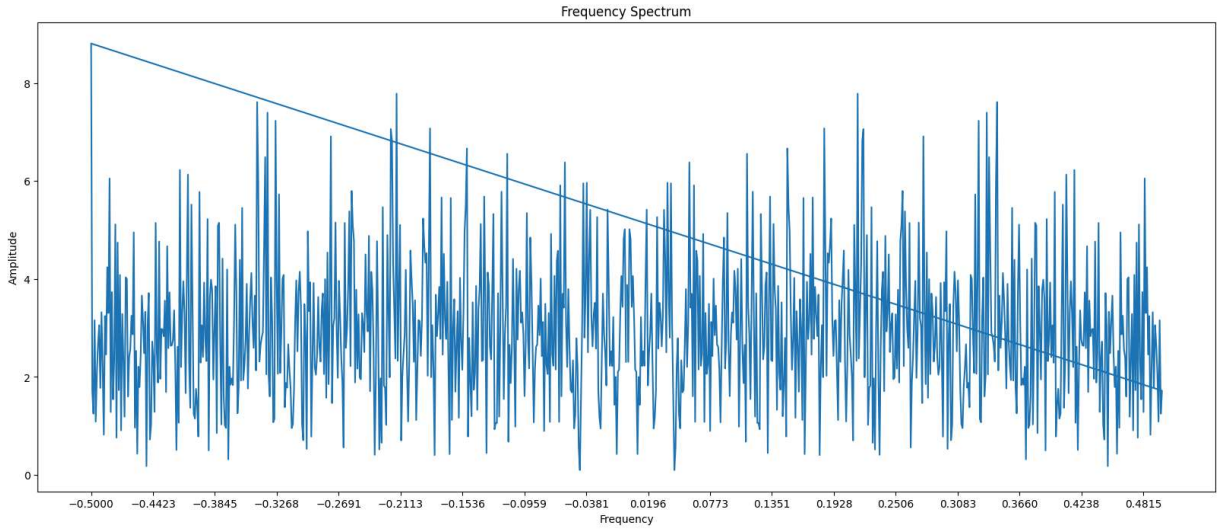


In [ ]:
```
stl = STL(daily_avg['target_2'], seasonal = 31, period = 7)

result = stl.fit()
fig = result.plot()
fig.set_size_inches(20,8)
plt.tight_layout()
plt.show()
```

The STL (Seasonal and Trend decomposition using Loess) decomposition breaks down the time series into trend, seasonal, and residual components.

This is useful for understanding the underlying patterns in the time series, such as long-term trends or recurring seasonal effects.

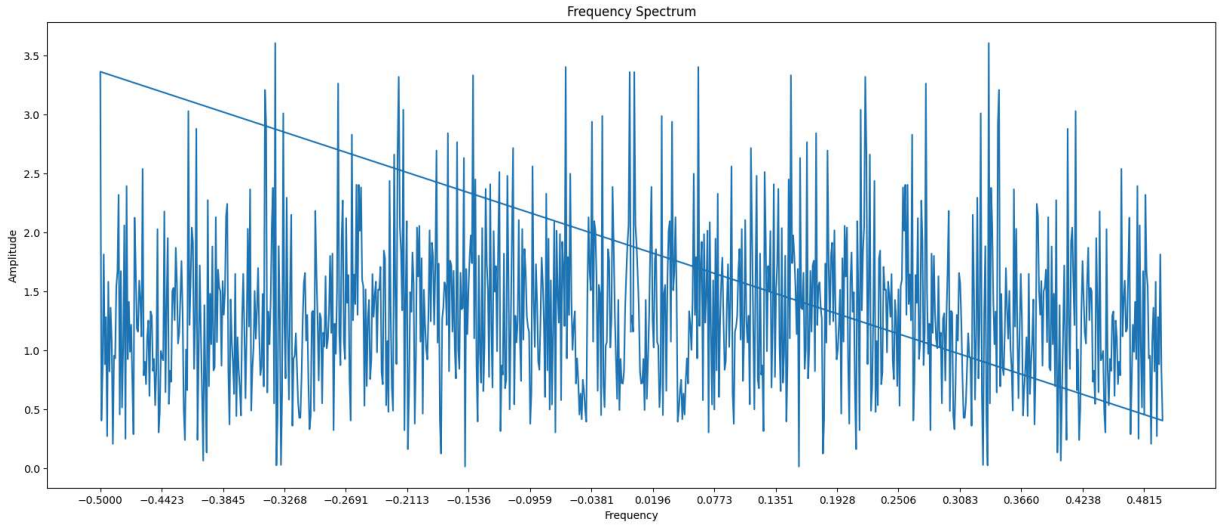
In [ ]:
```
from scipy.fft import fft, fftfreq

daily_avg = train_file.groupby(['date_id','symbol_id'])['target_1'].mean().reset_in

# Compute Fourier transform
symbol_zero = daily_avg[daily_avg['symbol_id']==0]
yf = fft(symbol_zero["target_1"].values)
xf = fftfreq(len(symbol_zero), 1)  # Assume unit frequency

# Plot frequency spectrum
plt.figure(figsize = (20,8))
plt.plot(xf, np.abs(yf))
plt.title("Frequency Spectrum")
plt.xticks(np.arange(min(xf), max(xf), step = xf.std()/5))
plt.xlabel("Frequency")
plt.ylabel("Amplitude")
plt.show()
```



The frequency spectrum plot shows the amplitude of different frequency components in the time series, obtained using the Fourier Transform.

This helps identify dominant frequencies in the data, which could correspond to periodic patterns or cycles.

In [29]:
```
daily_avg = train_file.groupby(['date_id','symbol_id'])['target_2'].mean().reset_in

# Compute Fourier transform
symbol_zero = daily_avg[daily_avg['symbol_id']==0]
yf = fft(symbol_zero["target_2"].values)
xf = fftfreq(len(symbol_zero), 1)  # Assume unit frequency

# Plot frequency spectrum
plt.figure(figsize = (20,8))
plt.plot(xf, np.abs(yf))
plt.title("Frequency Spectrum")
plt.xticks(np.arange(min(xf), max(xf), step = xf.std()/5))
plt.xlabel("Frequency")
plt.ylabel("Amplitude")
plt.show()
```



In [ ]: