In [32]:
```
import numpy as np # linear algebra
import pandas as pd # data processing
# Load trade data
trade_data_raw = pd.read_csv(r'CSV/ES.TradeData(17.6.24-14.10.24).csv')
feature_data_raw = pd.read_csv(r'CSV/ES.FeatureData(21m,1Y,28.10.24).csv')
```

TradePairing

In [33]:
```
print("\nFirst Few Rows of Trade Data:")
print(trade_data_raw.head(5))
# Define the columns to drop
columns_to_drop = ['Position ID', 'Connection name', 'Comment', 'Unnamed: 15', 'Trade ID', 'Order ID', 'Account']
# Drop the specified columns from the trade_data_raw DataFrame
trade_data_clean = trade_data_raw.drop(columns=columns_to_drop, errors='ignore')
# Reset the index to make it easier to pair trades
trade_data_clean.reset_index(drop=True, inplace=True)

# Check if trade_data_clean has an even number of rows
if len(trade_data_clean) % 2 != 0:
    print("Warning: Odd number of trades; the last trade will be ignored.")
    trade_data_clean = trade_data_clean.iloc[1:]  # Drop the last row to make the count even

# Create a list to store paired trades
paired_trades = []

# Iterate through the DataFrame in steps of 2 to pair trades
for i in range(0, len(trade_data_clean), 2):
    # Extract the closing trade (current row) and opening trade (next row)
    closing_trade = trade_data_clean.iloc[i]
    opening_trade = trade_data_clean.iloc[i + 1]

    # Store the paired trade details
    paired_trades.append({
        'Opening Date/Time': opening_trade['Date/Time'],
        'Closing Date/Time': closing_trade['Date/Time'],
        'Symbol': opening_trade['Symbol'],
        'Opening Side': opening_trade['Side'],
        'Closing Order Type': closing_trade['Order type'],
```

```
        'Quantity': abs(opening_trade['Quantity']),  # Absolute value of quantity
        'Price': closing_trade['Price'],  # Closing price as trade outcome price
        'Gross PnL': closing_trade['Gross P/L'],  # PnL from the closing trade
        'Fees': opening_trade['Fee'] + closing_trade['Fee'],  # Sum of opening and closing fees
        'Net PnL': closing_trade['Gross P/L'] + (opening_trade['Fee'] + closing_trade['Fee'])  # Net PnL after fees
    })

# Convert the paired trades into a DataFrame
trade_data_clean = pd.DataFrame(paired_trades)

# Display the first few rows of the summarized trades
print("\nTrade Summary DataFrame:")
print(trade_data_clean.head())
```

```
First Few Rows of Trade Data:
         Account                      Date/Time Symbol  Side Order type  \
0  Account (USD)   2024-10-14 7:05:00 AM -05:00     ES  Sell     Market
1  Account (USD)   2024-10-14 7:03:00 AM -05:00     ES  Sell    Tr.stop
2  Account (USD)   2024-10-14 1:12:01 AM -05:00     ES   Buy     Market
3  Account (USD)  2024-10-13 11:44:05 PM -05:00     ES   Buy    Tr.stop
4  Account (USD)  2024-10-13 10:01:00 PM -05:00     ES  Sell     Market

   Quantity    Price  Gross P/L   Fee  Net P/L  Comment  Unnamed: 11
0        -1  5867.75        0.0 -2.18    -2.18      NaN          NaN
1        -1  5867.75      500.0 -2.18   497.82      NaN          NaN
2         1  5857.75        0.0 -2.18    -2.18      NaN          NaN
3         1  5857.00     -212.5 -2.18  -214.68      NaN          NaN
4        -1  5852.75        0.0 -2.18    -2.18      NaN          NaN
Warning: Odd number of trades; the last trade will be ignored.

Trade Summary DataFrame:
               Opening Date/Time              Closing Date/Time Symbol  \
0   2024-10-14 1:12:01 AM -05:00   2024-10-14 7:03:00 AM -05:00     ES
1  2024-10-13 10:01:00 PM -05:00  2024-10-13 11:44:05 PM -05:00     ES
2   2024-10-13 6:36:00 PM -05:00   2024-10-13 9:04:39 PM -05:00     ES
3   2024-10-11 3:59:00 PM -05:00   2024-10-13 6:36:00 PM -05:00     ES
4   2024-10-11 2:46:00 PM -05:00   2024-10-11 3:58:07 PM -05:00     ES

  Opening Side Closing Order Type  Quantity    Price  Gross PnL  Fees  Net PnL
0          Buy            Tr.stop         1  5867.75      500.0 -4.36   495.64
1         Sell            Tr.stop         1  5857.00     -212.5 -4.36  -216.86
2          Buy            Tr.stop         1  5853.25     -237.5 -4.36  -241.86
3         Sell             Market         1  5858.25      112.5 -4.36   108.14
4          Buy            Tr.stop         1  5860.25      -12.5 -4.36   -16.86
```

FeatureCleaning

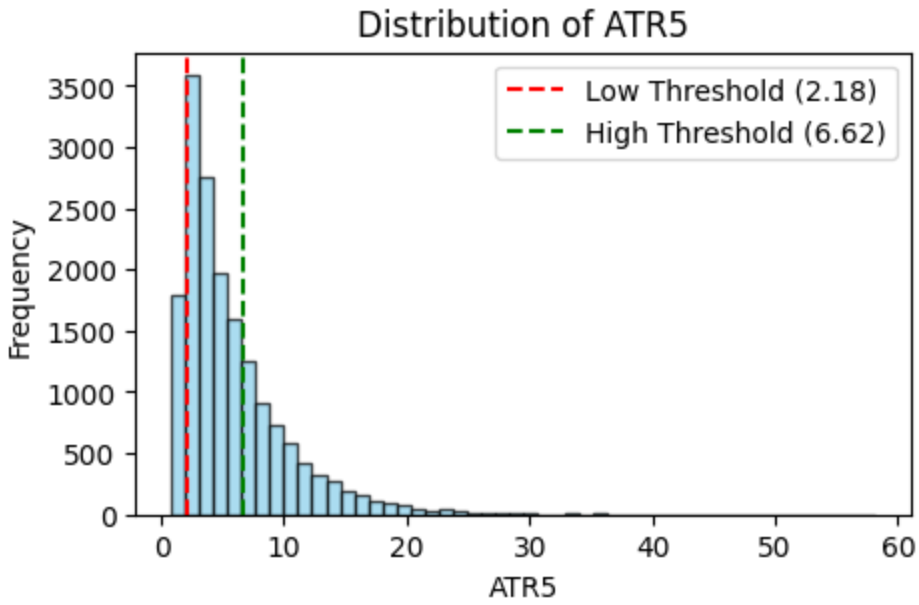
In [34]:

```python
# Clean NaN columns
first_non_nan_indices = feature_data_raw.notna().idxmax()
print(f"\n{first_non_nan_indices}")
first_valid_index = first_non_nan_indices.max()  # Use the earliest non-NaN index
print(f"\nFirst non-NaN index for market regime: {first_valid_index}")

# Create a new DataFrame with only rows from `start_index` onward
feature_data_clean = feature_data_raw.iloc[first_valid_index:].reset_index(drop=True)
# Drop all columns that contain 'Unnamed' in their name
```

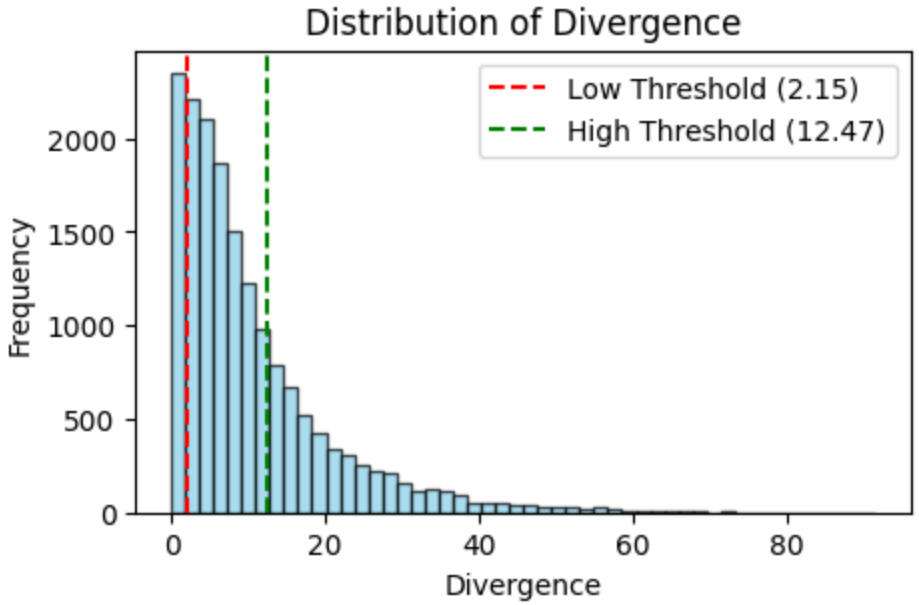
```python
feature_data_clean = feature_data_clean.loc[:, ~feature_data_clean.columns.str.contains('^Unnamed')]

# Rename columns to the desired names
column_rename_map = {
    "EMA (8, 21, 55)_EMA1": "EMA8",
    "EMA (8, 21, 55)_EMA2": "EMA21",
    "EMA (8, 21, 55)_EMA3": "EMA55",
    "ATR (5: SMA)_ATR": "ATR5",
    "AROON (34)_Aroon up Line": "AroonUp",
    "AROON (34)_Aroon down Line": "AroonDown",
}
feature_data_clean.rename(columns=column_rename_map, inplace=True)

# Display the cleaned and renamed feature data
print("\nClean Feature Data")
print(feature_data_clean.head())

import matplotlib.pyplot as plt
# Define a function to plot distribution with thresholds
def plot_distribution_with_thresholds(df, column):
    # Calculate median and standard deviation
    median = df[column].median()
    std = df[column].std()

    # Define thresholds
    low_threshold = median - 0.5 * std
    high_threshold = median + 0.5 * std

    # Plot the histogram
    plt.figure(figsize=(5, 3))
    plt.hist(df[column], bins=50, alpha=0.7, color='skyblue', edgecolor='black')
    plt.axvline(low_threshold, color='red', linestyle='--', label=f'Low Threshold ({low_threshold:.2f})')
    plt.axvline(high_threshold, color='green', linestyle='--', label=f'High Threshold ({high_threshold:.2f})')

    # Add labels and title
    plt.title(f"Distribution of {column}")
    plt.xlabel(column)
    plt.ylabel("Frequency")
    plt.legend()
    plt.show()

    print(f"low threshold: {low_threshold}")
```

```
    print(f"high threshold: {high_threshold}")

# Plot distribution for 'ATR5' column
plot_distribution_with_thresholds(feature_data_clean, 'ATR5')

# Plot distribution for 'EMA8 - EMA55' (Divergence) column
feature_data_clean['Divergence'] = (feature_data_clean['EMA8'] - feature_data_clean['EMA55']).abs()  # Calculate dive
plot_distribution_with_thresholds(feature_data_clean, 'Divergence')

# Calculate thresholds for ATR5 using Median ± 0.5 * Std
atr_median = feature_data_clean['ATR5'].median()
atr_std = feature_data_clean['ATR5'].std()
atr_low = atr_median - 0.5 * atr_std
atr_high = atr_median + 0.5 * atr_std
# Calculate divergence and then thresholds for Divergence using Median ± 0.5 * Std
feature_data_clean['Divergence'] = (feature_data_clean['EMA8'] - feature_data_clean['EMA55']).abs()  # Calculate dive
divergence_median = feature_data_clean['Divergence'].median()
divergence_std = feature_data_clean['Divergence'].std()
divergence_low = divergence_median - 0.5 * divergence_std
divergence_high = divergence_median + 0.5 * divergence_std


# Calculate regimeClass, atrBucket, and divergenceBucket and add them as new columns
feature_data_clean['RegimeClass'] = feature_data_clean.apply(
    lambda row: 'UpTrend' if row['EMA8'] > row['EMA21'] and row['AroonUp'] > 66 else
                'DownTrend' if row['EMA8'] < row['EMA21'] and row['AroonDown'] > 66 else
                'NoTrend', axis=1
)

feature_data_clean['ATRBucket'] = feature_data_clean['ATR5'].apply(
    lambda x: 'High' if x > atr_high else 'Low' if x < atr_low else 'Medium'
)

feature_data_clean['DivergenceBucket'] = (feature_data_clean['EMA8'] - feature_data_clean['EMA55']).abs().apply(
    lambda x: 'High' if x > divergence_high else 'Low' if x < divergence_low else 'Medium'
)

# Display the modified DataFrame
print("\nFeature Data with Regime Classifications:")
print(feature_data_clean.head())
```

```
DateTime                       0
EMA (8, 21, 55)_EMA1           8
EMA (8, 21, 55)_EMA2          21
EMA (8, 21, 55)_EMA3          55
ATR (5: SMA)_ATR               4
AROON (34)_Aroon up Line      33
AROON (34)_Aroon down Line    33
Unnamed: 7                     0
dtype: int64

First non-NaN index for market regime: 55

Clean Feature Data
                        DateTime         EMA8        EMA21        EMA55   ATR5  \
0  2023-10-30 1:03:00 PM -05:00  4165.780153  4164.430392  4160.642857   9.45
1  2023-10-30 1:24:00 PM -05:00  4169.384563  4166.027629  4161.405612  10.15
2  2023-10-30 1:45:00 PM -05:00  4172.854660  4167.752390  4162.248269   9.85
3  2023-10-30 2:06:00 PM -05:00  4175.220291  4169.183991  4163.007259   9.45
4  2023-10-30 2:27:00 PM -05:00  4178.949115  4171.258174  4164.042714   9.95

      AroonUp  AroonDown
0   73.529412   5.882353
1  100.000000   2.941176
2  100.000000  73.529412
3   97.058824  70.588235
4  100.000000  67.647059
```




```
low threshold: 2.1768695660246324
high threshold: 6.623130433975368
```



```
low threshold: 2.1463089245365436
high threshold: 12.466862886972802

Feature Data with Regime Classifications:
                       DateTime         EMA8        EMA21        EMA55   ATR5  \
0  2023-10-30 1:03:00 PM -05:00  4165.780153  4164.430392  4160.642857   9.45
1  2023-10-30 1:24:00 PM -05:00  4169.384563  4166.027629  4161.405612  10.15
2  2023-10-30 1:45:00 PM -05:00  4172.854660  4167.752390  4162.248269   9.85
3  2023-10-30 2:06:00 PM -05:00  4175.220291  4169.183991  4163.007259   9.45
4  2023-10-30 2:27:00 PM -05:00  4178.949115  4171.258174  4164.042714   9.95

      AroonUp  AroonDown  Divergence RegimeClass ATRBucket DivergenceBucket
0   73.529412   5.882353    5.137295     UpTrend      High           Medium
1  100.000000   2.941176    7.978951     UpTrend      High           Medium
2  100.000000  73.529412   10.606391     UpTrend      High           Medium
3   97.058824  70.588235   12.213032     UpTrend      High           Medium
4  100.000000  67.647059   14.906401     UpTrend      High             High
```

Combining Trades and Features

In [35]:

```python
trade_data_clean['Opening Date/Time'] = pd.to_datetime(trade_data_clean['Opening Date/Time'])
trade_data_clean['Closing Date/Time'] = pd.to_datetime(trade_data_clean['Closing Date/Time'])
feature_data_clean['DateTime'] = pd.to_datetime(feature_data_clean['DateTime'])
# Sort the raw feature data and trade data by their time columns to prepare for merge
feature_data_clean = feature_data_clean.sort_values(by='DateTime')
trade_data_clean = trade_data_clean.sort_values(by='Opening Date/Time')

# Perform an asof merge to align trade entries with the nearest preceding time in feature_data_clean
# This will align each trade with the closest previous time in the `feature_data_clean`
trade_data_featured = pd.merge_asof(
    trade_data_clean,
    feature_data_clean,
    left_on='Opening Date/Time',
    right_on='DateTime',
    direction='backward'
)

# Drop the 'DateTime' column from the merged DataFrame as it's now redundant
trade_data_featured = trade_data_featured.drop(columns=['DateTime'])

# Create the TradeProfitBinary column based on Net PnL
trade_data_featured['TradeProfitBinary'] = (trade_data_featured['Net PnL'] > 0).astype(int)
```

```
print("\nFeatured Trade Data")
print(trade_data_featured.head())
```

```
C:\Users\prath\AppData\Local\Temp\ipykernel_111484\877092753.py:3: UserWarning: Could not infer format, so each eleme
nt will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please s
pecify a format.
  feature_data_clean['DateTime'] = pd.to_datetime(feature_data_clean['DateTime'])
Featured Trade Data
          Opening Date/Time         Closing Date/Time Symbol Opening Side  \
0 2024-06-17 12:18:00-05:00 2024-06-17 12:25:30-05:00     ES          Buy
1 2024-06-17 12:26:00-05:00 2024-06-17 12:55:15-05:00     ES          Buy
2 2024-06-17 13:17:00-05:00 2024-06-17 13:33:17-05:00     ES          Buy
3 2024-06-17 13:36:00-05:00 2024-06-17 14:46:00-05:00     ES          Buy
4 2024-06-17 15:12:00-05:00 2024-06-17 15:18:14-05:00     ES         Sell

  Closing Order Type  Quantity    Price  Gross PnL  Fees  Net PnL  ...  \
0            Tr.stop         1  5455.25      237.5 -4.36   233.14  ...
1            Tr.stop         1  5464.25      337.5 -4.36   333.14  ...
2            Tr.stop         1  5476.00      225.0 -4.36   220.64  ...
3            Tr.stop         1  5486.00      312.5 -4.36   308.14  ...
4            Tr.stop         1  5493.00     -237.5 -4.36  -241.86  ...

         EMA21        EMA55   ATR5     AroonUp  AroonDown  Divergence  \
0  5438.235853  5435.523723   6.45  100.000000  79.411765    7.164947
1  5440.373503  5436.460376   8.20  100.000000  76.470588   10.464145
2  5445.300416  5438.720490   8.50  100.000000  70.588235   16.919035
3  5448.364014  5440.159044  10.00  100.000000  67.647059   20.671698
4  5463.704252  5448.247772   8.30   91.176471  52.941176   32.688439

   RegimeClass ATRBucket DivergenceBucket TradeProfitBinary
0      UpTrend    Medium           Medium                1
1      UpTrend      High           Medium                1
2      UpTrend      High             High                1
3      UpTrend      High             High                1
4      UpTrend      High             High                0

[5 rows x 21 columns]
```

Regime Stats

In [36]:

```
# Shuffle the DataFrame to randomize the order of trades
trade_data_shuffle1 = trade_data_featured.sample(frac=1, random_state=34).reset_index(drop=True)
trade_data_shuffle2 = trade_data_featured.sample(frac=1, random_state=55).reset_index(drop=True)
trade_data_shuffle3 = trade_data_featured.sample(frac=1, random_state=89).reset_index(drop=True)

# Calculate the size of each subset (approximately one-third of the total rows)
subset_size = len(trade_data_featured) // 3

# Split the data into three parts
subset_1 = trade_data_shuffle1.iloc[:subset_size]
subset_2 = trade_data_shuffle1.iloc[subset_size:2 * subset_size]
subset_3 = trade_data_shuffle1.iloc[2 * subset_size:]
subset_4 = trade_data_shuffle2.iloc[:subset_size]
subset_5 = trade_data_shuffle2.iloc[subset_size:2 * subset_size]
subset_6 = trade_data_shuffle2.iloc[2 * subset_size:]
subset_7 = trade_data_shuffle3.iloc[:subset_size]
subset_8 = trade_data_shuffle3.iloc[subset_size:2 * subset_size]
subset_9 = trade_data_shuffle3.iloc[2 * subset_size:]

def analyze_data(data, subset_name):
    # Group by ATR, regime, and opening side
    performance = data.groupby(['Opening Side', 'ATRBucket', 'RegimeClass', 'DivergenceBucket'], observed=False).agg(
        'Net PnL': ['mean', 'sum', 'count'],
        'TradeProfitBinary': 'mean'  # Mean gives the Win% directly
    }).reset_index()

     # Flatten the column names
    performance.columns = ['_'.join(col).strip('_') for col in performance.columns.values]

    # Display the summarized performance for the combined conditions
    return performance

# Analyze each subset by the combined conditions
subset_1_performance = analyze_data(subset_1, "Subset 1")
subset_2_performance = analyze_data(subset_2, "Subset 2")
subset_3_performance = analyze_data(subset_3, "Subset 3")
subset_4_performance = analyze_data(subset_4, "Subset 4")
subset_5_performance = analyze_data(subset_5, "Subset 5")
subset_6_performance = analyze_data(subset_6, "Subset 6")
subset_7_performance = analyze_data(subset_7, "Subset 7")
subset_8_performance = analyze_data(subset_8, "Subset 8")
```

```
subset_9_performance = analyze_data(subset_9, "Subset 9")

# Define a function to filter and sum Total Net PnL for a given subset
def filter_and_sum_pnl(performance_data, subset_name):
    # Filter the performance DataFrame to include only rows where Average Net PnL > 10 and Trade Count > 5
    filtered_performance = performance_data[
        (performance_data['Net PnL_mean'] > 10) &
        (performance_data['Net PnL_count'] > 5)
    ]

    return filtered_performance

# Apply the filter and summation function to each subset
subset_1_filtered = filter_and_sum_pnl(subset_1_performance, "Subset 1")
subset_2_filtered = filter_and_sum_pnl(subset_2_performance, "Subset 2")
subset_3_filtered = filter_and_sum_pnl(subset_3_performance, "Subset 3")
subset_4_filtered = filter_and_sum_pnl(subset_4_performance, "Subset 4")
subset_5_filtered = filter_and_sum_pnl(subset_5_performance, "Subset 5")
subset_6_filtered = filter_and_sum_pnl(subset_6_performance, "Subset 6")
subset_7_filtered = filter_and_sum_pnl(subset_7_performance, "Subset 7")
subset_8_filtered = filter_and_sum_pnl(subset_8_performance, "Subset 8")
subset_9_filtered = filter_and_sum_pnl(subset_9_performance, "Subset 9")

# Combine the filtered subsets
combined_subsets = pd.concat([
    subset_1_filtered[['Opening Side', 'ATRBucket', 'RegimeClass', 'DivergenceBucket', 'Net PnL_sum', 'Net PnL_mean',
    subset_2_filtered[['Opening Side', 'ATRBucket', 'RegimeClass', 'DivergenceBucket', 'Net PnL_sum', 'Net PnL_mean',
    subset_3_filtered[['Opening Side', 'ATRBucket', 'RegimeClass', 'DivergenceBucket', 'Net PnL_sum', 'Net PnL_mean',
    subset_4_filtered[['Opening Side', 'ATRBucket', 'RegimeClass', 'DivergenceBucket', 'Net PnL_sum', 'Net PnL_mean',
    subset_5_filtered[['Opening Side', 'ATRBucket', 'RegimeClass', 'DivergenceBucket', 'Net PnL_sum', 'Net PnL_mean',
    subset_6_filtered[['Opening Side', 'ATRBucket', 'RegimeClass', 'DivergenceBucket', 'Net PnL_sum', 'Net PnL_mean',
    subset_7_filtered[['Opening Side', 'ATRBucket', 'RegimeClass', 'DivergenceBucket', 'Net PnL_sum', 'Net PnL_mean',
    subset_8_filtered[['Opening Side', 'ATRBucket', 'RegimeClass', 'DivergenceBucket', 'Net PnL_sum', 'Net PnL_mean',
    subset_9_filtered[['Opening Side', 'ATRBucket', 'RegimeClass', 'DivergenceBucket', 'Net PnL_sum', 'Net PnL_mean',
], ignore_index=True)

# Group by conditions and compute aggregates
final_regimes = combined_subsets.groupby(
    ['Opening Side', 'ATRBucket', 'RegimeClass', 'DivergenceBucket'], observed=True
).agg(
    Count=('Opening Side', 'size'),
    AvgTotalNetPnl=('Net PnL_sum', 'mean'),
```

```
    AvgWinRate=('TradeProfitBinary_mean', 'mean'),
    AvgMeanNetPnl=('Net PnL_mean', 'mean')
).reset_index()

# Filter for Count >= 5 and sort by Count in descending order
final_regimes = final_regimes[final_regimes['Count'] >= 5]
#.sort_values(by='Count', ascending=False)

# Display the filtered DataFrame
print("\nNumber of times each regime appeared in the filtered results across the nine subsets:")
print(final_regimes)
```

```
Number of times each regime appeared in the filtered results across the nine subsets:
   Opening Side ATRBucket RegimeClass DivergenceBucket  Count  AvgTotalNetPnl  \
2           Buy      High   DownTrend           Medium      7     4272.945714
4           Buy      High     NoTrend           Medium      5     1809.952000
7           Buy      High     UpTrend           Medium      8     3612.302500
8           Buy    Medium   DownTrend           Medium      6     1168.026667
9           Buy    Medium     NoTrend              Low      7     1230.194286
13         Sell      High   DownTrend              Low      7     1418.277143
15         Sell      High     NoTrend             High      8     1903.027500
16         Sell      High     NoTrend           Medium      6     2364.156667
21         Sell    Medium     NoTrend           Medium      6     1242.063333
22         Sell    Medium     UpTrend             High      5      752.156000

    AvgWinRate  AvgMeanNetPnl
2     0.430013      87.447242
4     0.420493      83.947610
7     0.502971      78.308422
8     0.565278     146.265000
9     0.615646     180.564178
13    0.570728     145.894812
15    0.460924      78.150450
16    0.504025     122.926908
21    0.442026      62.936595
22    0.296024      73.548092
```

Backtesting

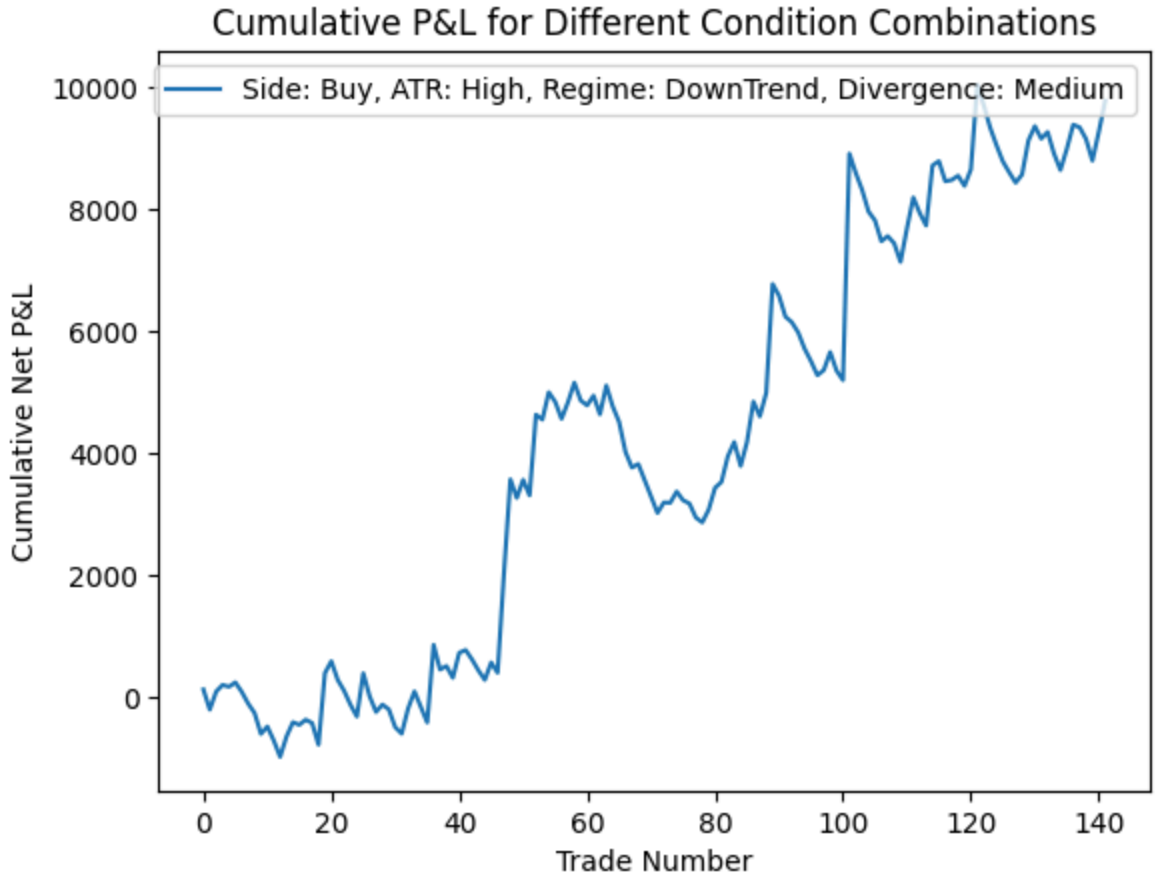
In [37]:

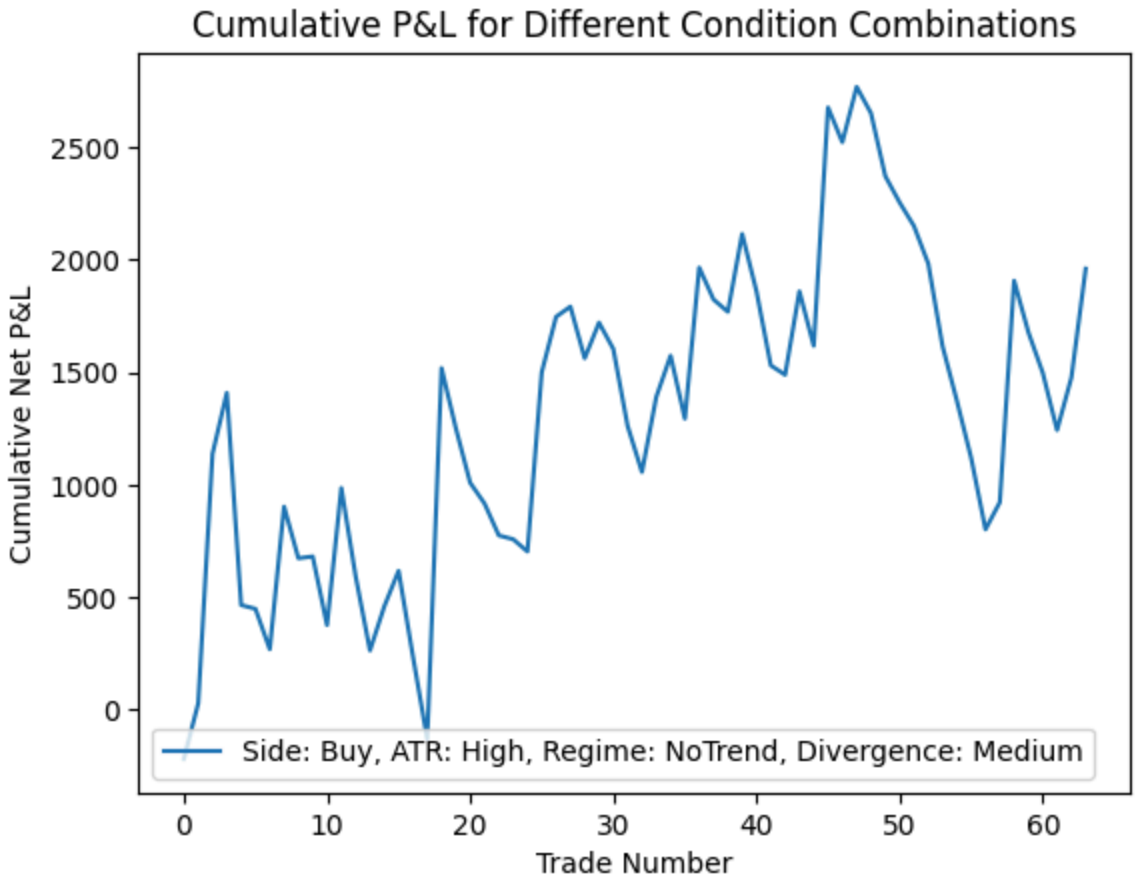
```
import matplotlib.pyplot as plt

# Extract the unique combinations in the order they appear
```

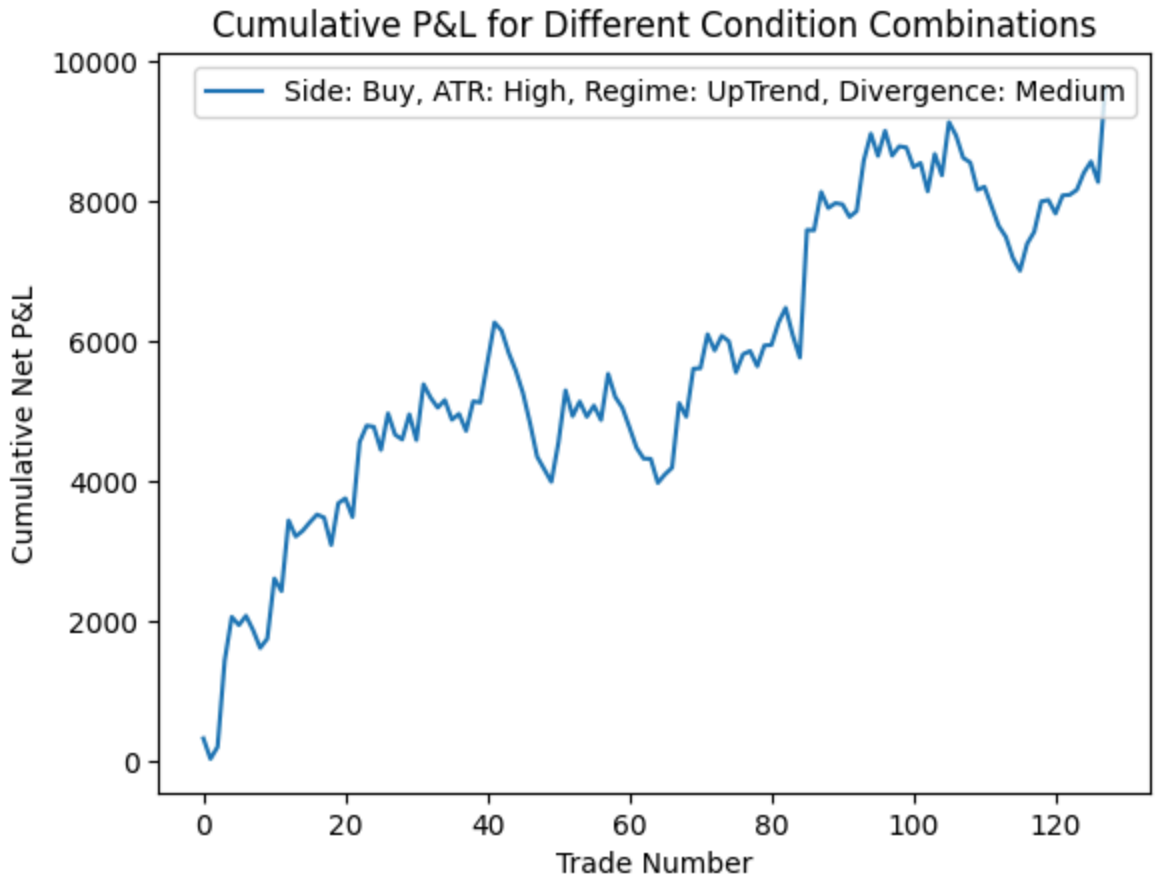
```python
target_conditions_list = (
    final_regimes[['Opening Side', 'ATRBucket', 'RegimeClass', 'DivergenceBucket']]
    .drop_duplicates()
    .to_dict(orient='records')
)

# Display the ordered target conditions list
print("Ordered Target Conditions List:")
for condition in target_conditions_list:
    print(condition)


# List to store the results
backtest_results = []

# Loop through each combination
for conditions in target_conditions_list:
    # Filter trades based on the current combination of conditions
    filtered_data = trade_data_featured[
        (trade_data_featured['ATRBucket'] == conditions['ATRBucket']) &
        (trade_data_featured['RegimeClass'] == conditions['RegimeClass']) &
        (trade_data_featured['DivergenceBucket'] == conditions['DivergenceBucket']) &
        (trade_data_featured['Opening Side'] == conditions['Opening Side'])
    ].copy()

    # Calculate Backtest Metrics for the current combination
    total_trades = len(filtered_data)
    total_pnl = filtered_data['Net PnL'].sum()
    win_rate = filtered_data['TradeProfitBinary'].mean() * 100  # Convert to percentage
    average_pnl_per_trade = filtered_data['Net PnL'].mean()

    # Calculate Drawdown
    filtered_data['Cumulative PnL'] = filtered_data['Net PnL'].cumsum()
    drawdown = filtered_data['Cumulative PnL'].cummax() - filtered_data['Cumulative PnL']
    max_drawdown = drawdown.max()

    # Store the results in a dictionary
    backtest_results.append({
        'Side': conditions['Opening Side'],
        'ATRBucket': conditions['ATRBucket'],
        'RegimeClass': conditions['RegimeClass'],
        'DivergenceBucket': conditions['DivergenceBucket'],
```

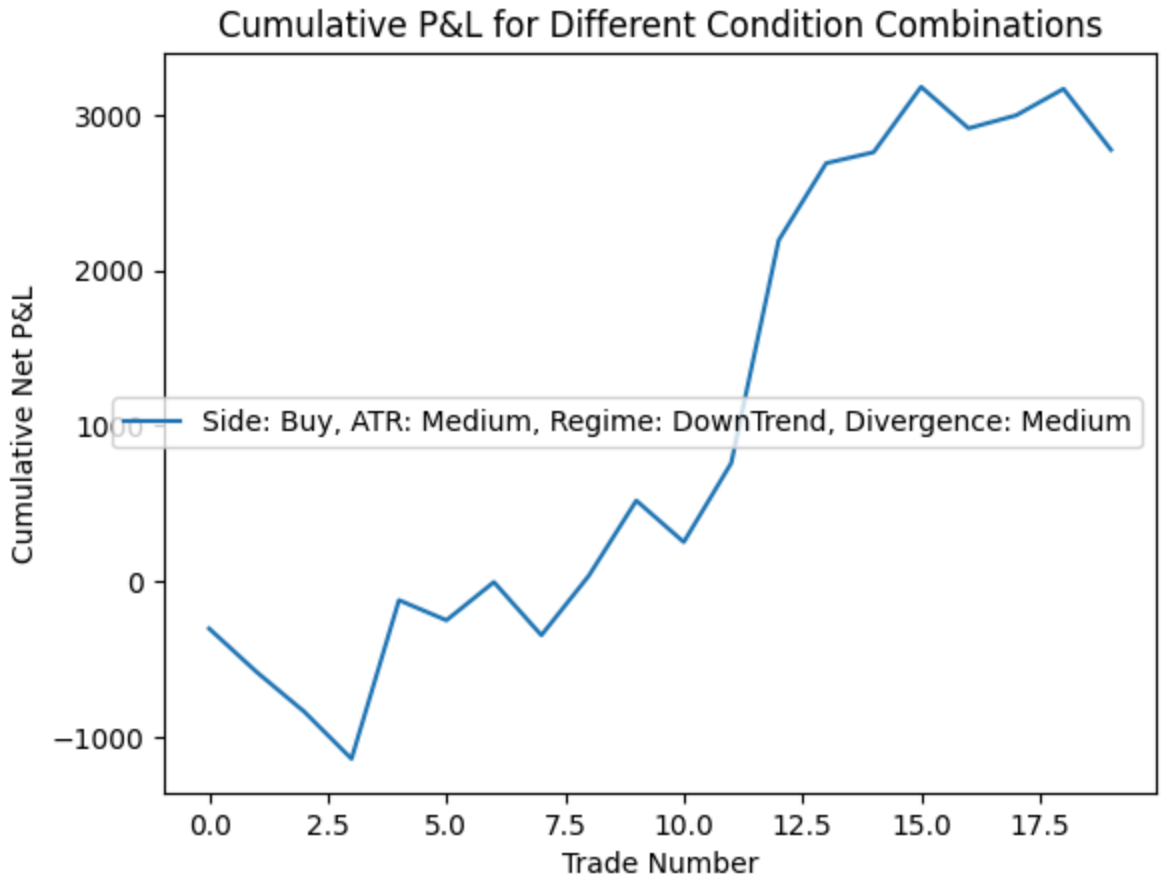
```
        'Total Trades': total_trades,
        'Total Net P&L': total_pnl,
        'Win Rate (%)': win_rate,
        'Avg P&L per Trade': average_pnl_per_trade,
        'Max Drawdown': max_drawdown
    })

    # Create a label for the plot based on the current condition
    label = f"Side: {conditions['Opening Side']}, ATR: {conditions['ATRBucket']}, Regime: {conditions['RegimeClass']}

    # Plot cumulative PnL for this combination
    plt.plot(filtered_data['Cumulative PnL'].values, label=label)
    plt.xlabel('Trade Number')
    plt.ylabel('Cumulative Net P&L')
    plt.title('Cumulative P&L for Different Condition Combinations')
    plt.legend()
    plt.show()

# Convert results to a DataFrame for easy viewing
backtest_results_df = pd.DataFrame(backtest_results)

# Display the backtest results
print("\nBacktest Results for Multiple Condition Combinations:")
print(backtest_results_df)

# Calculate overall totals and averages
total_net_pnl = backtest_results_df['Total Net P&L'].sum()
total_trades = backtest_results_df['Total Trades'].sum()
average_win_rate = (backtest_results_df['Win Rate (%)'] * backtest_results_df['Total Trades']).sum() / total_trades
# Display the aggregated stats
print("\nOverall Backtest Statistics Across All Conditions:")
print(f"Total Net P&L: {total_net_pnl:.2f}")
print(f"Total Trades: {total_trades}")
print(f"Average Win Rate (%): {average_win_rate:.2f}")
```

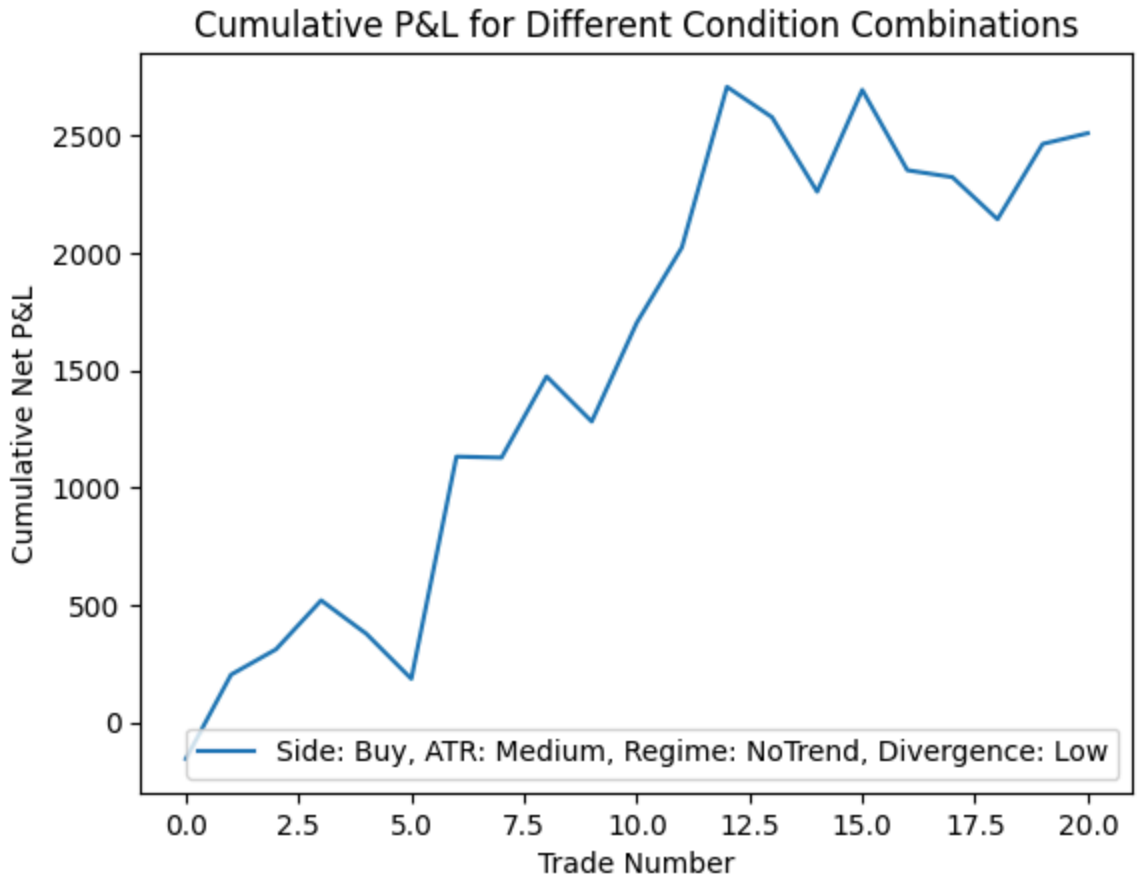
```
Ordered Target Conditions List:
{'Opening Side': 'Buy', 'ATRBucket': 'High', 'RegimeClass': 'DownTrend', 'DivergenceBucket': 'Medium'}
{'Opening Side': 'Buy', 'ATRBucket': 'High', 'RegimeClass': 'NoTrend', 'DivergenceBucket': 'Medium'}
{'Opening Side': 'Buy', 'ATRBucket': 'High', 'RegimeClass': 'UpTrend', 'DivergenceBucket': 'Medium'}
{'Opening Side': 'Buy', 'ATRBucket': 'Medium', 'RegimeClass': 'DownTrend', 'DivergenceBucket': 'Medium'}
{'Opening Side': 'Buy', 'ATRBucket': 'Medium', 'RegimeClass': 'NoTrend', 'DivergenceBucket': 'Low'}
{'Opening Side': 'Sell', 'ATRBucket': 'High', 'RegimeClass': 'DownTrend', 'DivergenceBucket': 'Low'}
{'Opening Side': 'Sell', 'ATRBucket': 'High', 'RegimeClass': 'NoTrend', 'DivergenceBucket': 'High'}
{'Opening Side': 'Sell', 'ATRBucket': 'High', 'RegimeClass': 'NoTrend', 'DivergenceBucket': 'Medium'}
{'Opening Side': 'Sell', 'ATRBucket': 'Medium', 'RegimeClass': 'NoTrend', 'DivergenceBucket': 'Medium'}
{'Opening Side': 'Sell', 'ATRBucket': 'Medium', 'RegimeClass': 'UpTrend', 'DivergenceBucket': 'High'}
```

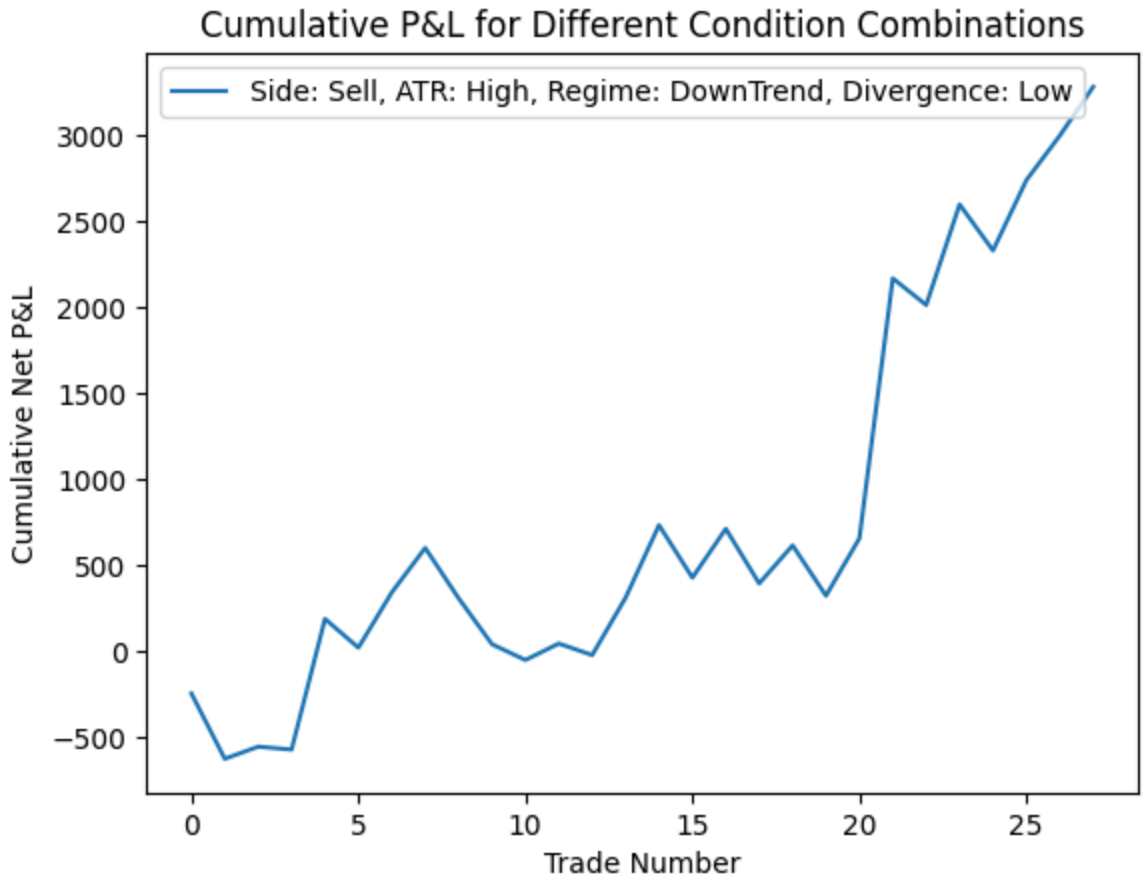


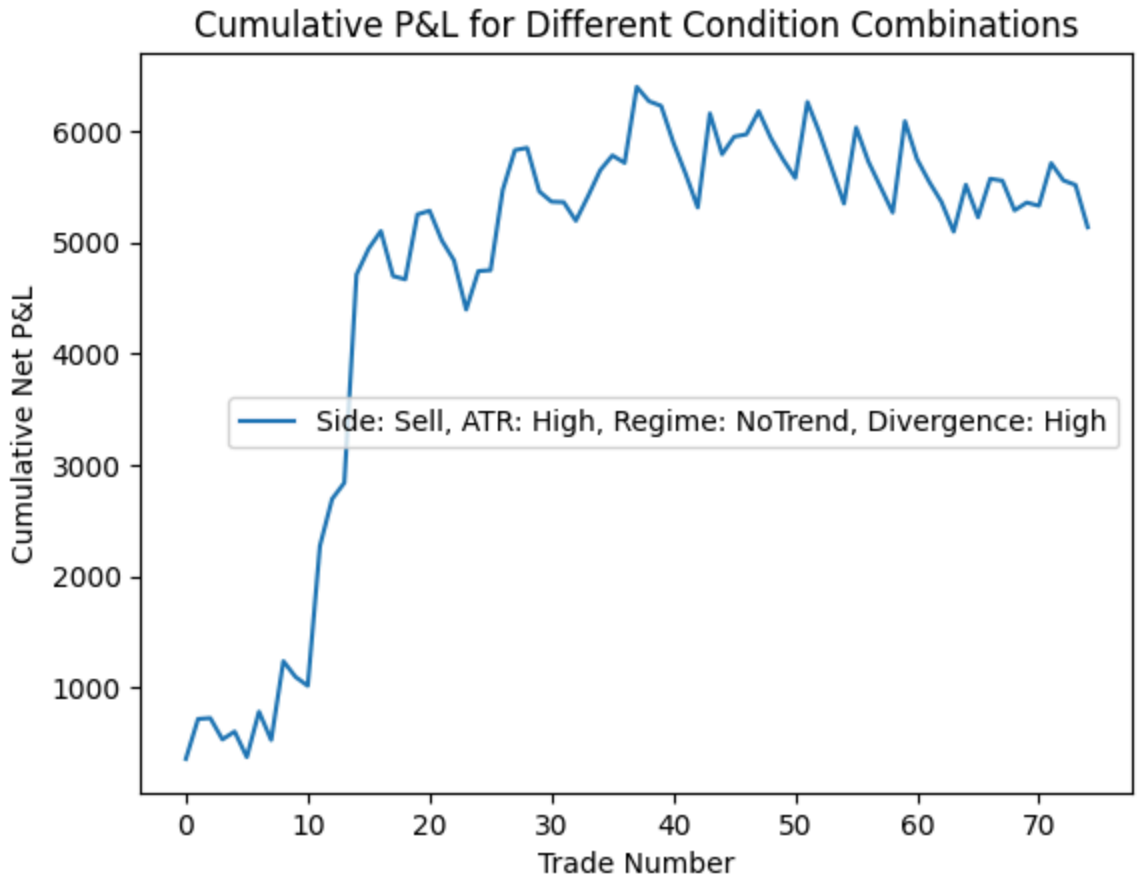

Cumulative P&L for Different Condition Combinations
Cumulative Net P&L
2500
2000
1500
1000
500
0
Side: Buy, ATR: High, Regime: NoTrend, Divergence: Medium
0
10
20
30
40
50
60
Trade Number

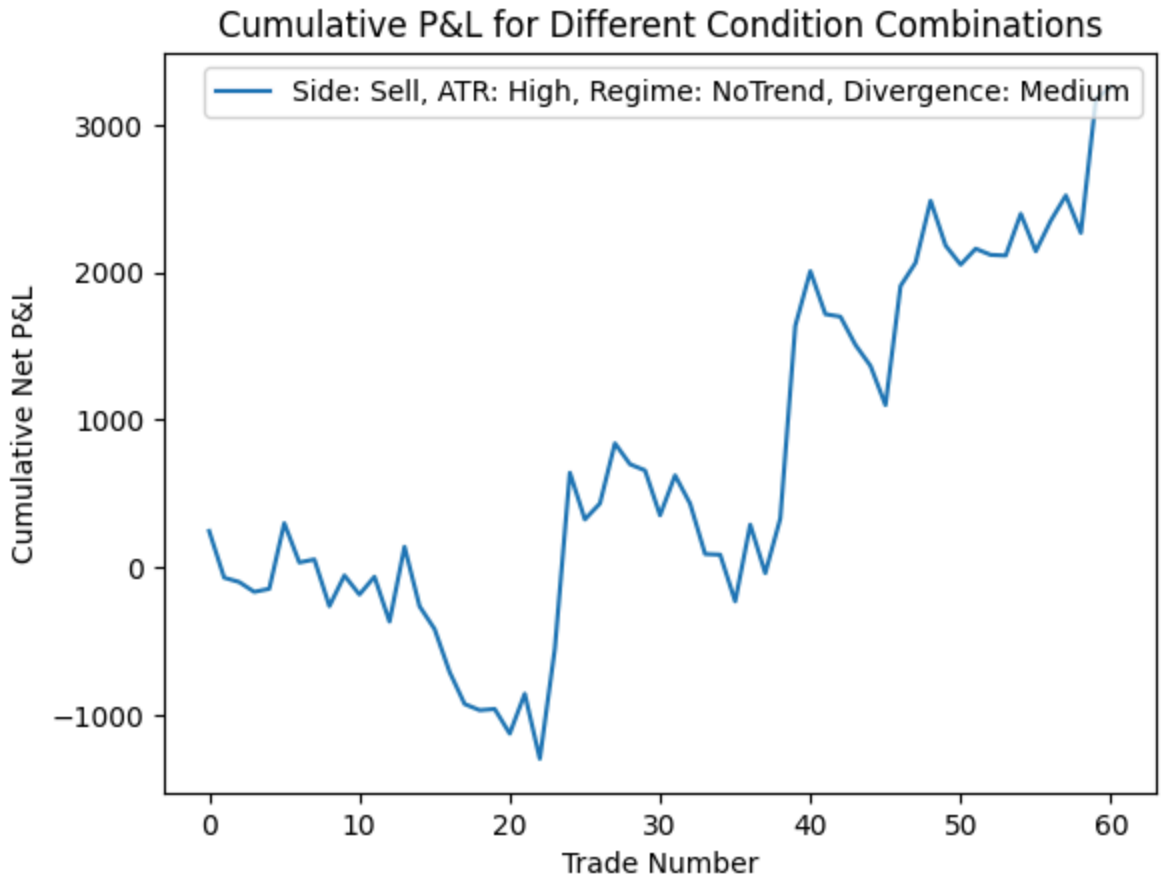

Cumulative P&L for Different Condition Combinations
Side: Buy, ATR: High, Regime: UpTrend, Divergence: Medium
Cumulative Net P&L
Trade Number
10000
8000
6000
4000
2000
0
0
20
40
60
80
100
120

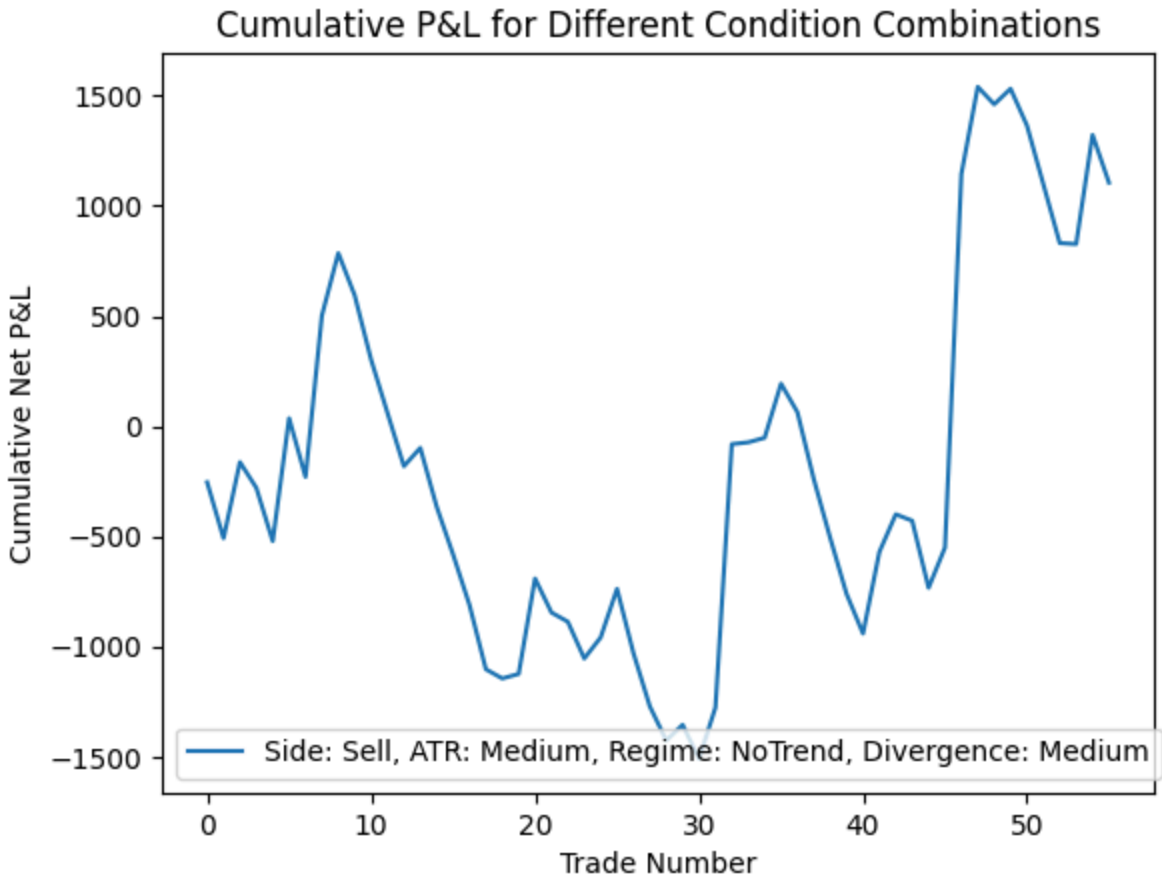

Cumulative P&L for Different Condition Combinations
Cumulative Net P&L
Trade Number
Side: Buy, ATR: Medium, Regime: DownTrend, Divergence: Medium
3000
2000
1000
0
−1000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

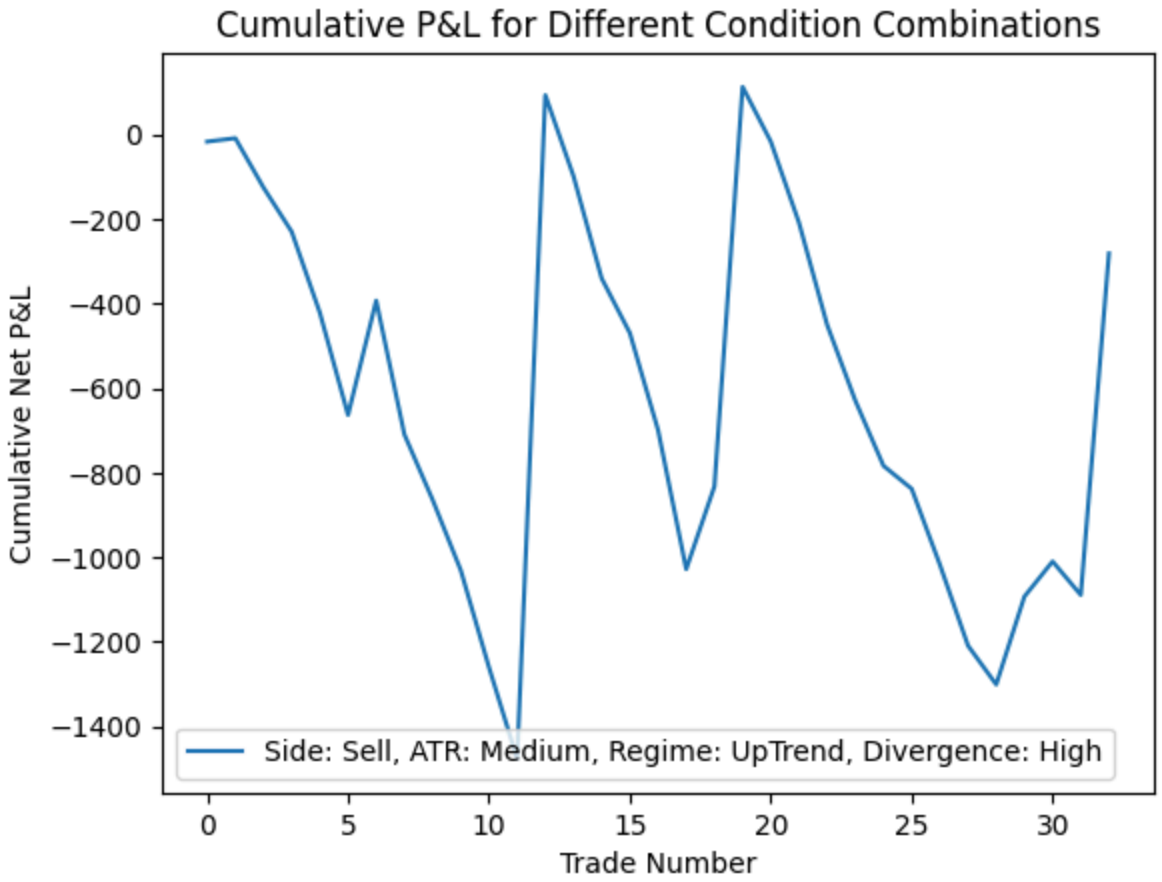

Cumulative P&L for Different Condition Combinations
Cumulative Net P&L
2500
2000
1500
1000
500
0
Side: Buy, ATR: Medium, Regime: NoTrend, Divergence: Low
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Trade Number


Cumulative P&L for Different Condition Combinations
Side: Sell, ATR: High, Regime: DownTrend, Divergence: Low
Cumulative Net P&L
3000
2500
2000
1500
1000
500
0
−500
0
5
10
15
20
25
Trade Number


Cumulative P&L for Different Condition Combinations
Cumulative Net P&L
Trade Number
Side: Sell, ATR: High, Regime: NoTrend, Divergence: High
1000
2000
3000
4000
5000
6000
0
10
20
30
40
50
60
70


Cumulative P&L for Different Condition Combinations
Side: Sell, ATR: High, Regime: NoTrend, Divergence: Medium
Cumulative Net P&L
3000
2000
1000
0
−1000
0
10
20
30
40
50
60
Trade Number


Cumulative P&L for Different Condition Combinations
Cumulative Net P&L
1500
1000
500
0
−500
−1000
−1500
0
10
20
30
40
50
Trade Number
Side: Sell, ATR: Medium, Regime: NoTrend, Divergence: Medium


Cumulative P&L for Different Condition Combinations
Cumulative Net P&L
0
−200
−400
−600
−800
−1000
−1200
−1400
0
5
10
15
20
25
30
Trade Number
Side: Sell, ATR: Medium, Regime: UpTrend, Divergence: High

```
Backtest Results for Multiple Condition Combinations:
   Side ATRBucket RegimeClass DivergenceBucket  Total Trades  Total Net P&L  \
0   Buy      High   DownTrend           Medium           142        9768.38
1   Buy      High     NoTrend           Medium            64        1958.46
2   Buy      High     UpTrend           Medium           128        9629.42
3   Buy    Medium   DownTrend           Medium            20        2775.30
4   Buy    Medium     NoTrend              Low            21        2508.44
5  Sell      High   DownTrend              Low            28        3277.92
6  Sell      High     NoTrend             High            75        5135.50
7  Sell      High     NoTrend           Medium            61        3259.04
8  Sell    Medium     NoTrend           Medium            56        1105.84
9  Sell    Medium     UpTrend             High            33        -281.38

   Win Rate (%)  Avg P&L per Trade  Max Drawdown
0     42.253521          68.791408       2287.20
1     37.500000          30.600937       1964.24
2     49.218750          75.229844       2287.78
3     55.000000         138.765000        838.08
4     52.380952         119.449524        563.66
5     53.571429         117.068571        650.58
6     45.333333          68.473333       1300.86
7     44.262295          53.426885       1599.12
8     39.285714          19.747143       2295.92
9     24.242424          -8.526667       1468.60

Overall Backtest Statistics Across All Conditions:
Total Net P&L: 39136.92
Total Trades: 628
Average Win Rate (%): 43.79
```

Next steps are to test a couple of these conditions that seem to possess an edge on more data and other symbols followed by a walk-forward before demo deployment*