Pairs Trading - Correlation

Motivation:

- Security A - (Something Correlated to it) is more mean-reverting. Similar securities should trade similarily. Hence, any discrepancies between the correlated pairs should converge
- When there is a discrepancy, we form a long-short spread trade and bet on the discrepancy converging
- Pairs trading is one of the most popular statistical arbitrage strategies in traditional markets. We want to test the performance of this strategy in the cryptocurrency market, as it is still relatively new and should be fertile grounds for finding market inefficiencies
- While cointegration is a robust statistical method for identifying long-term relationships, Correlation is simpler and more responsive to short-term relationships, making it suitable for the fast-moving cryptocurrency market.

In [1]:
```
from datetime import datetime
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller

import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
from itertools import combinations
```

Data

We selected the top 1,000 currencies sorted by volume from CoinGecko. Then, we filtered out coins with fewer than 90% of daily price data points between 2018 and 2024.

In [2]:
```
from_pickle = pd.read_pickle('Data/CoinGecko_px_vol_1D.pkl')
```

In [3]:
```
price_columns = [col for col in from_pickle.columns if 'price' in col]
crypto_px = from_pickle[price_columns]
crypto_px.columns = crypto_px.columns.droplevel(1)
```

In [4]:
```
crypto_px = crypto_px.loc['2018-01-01':]
crypto_px = crypto_px.drop(columns=['TUSD', 'DAI', 'WBTC', 'WETH', 'US
```

In [5]:
```python
# Calculate the total number of data points
total_data_points = len(crypto_px)

# Calculate the number of non-NA/null entries for each coin
non_null_counts = crypto_px.notnull().sum()

# Calculate the threshold for 90% of the data points
threshold = 0.90 * total_data_points

# Filter out columns with fewer than 90% of data points
crypto_px = crypto_px.loc[:, non_null_counts >= threshold]

crypto_px
```

Out[5]:

	BTC	XRP	BNB	DOGE	ADA	TRX	LTC	LIN
Date								
2018-01-01	14093.606831	2.310120	8.828099	0.009091	0.747140	0.051654	230.462120	0.75103
2018-01-02	15321.932852	2.455290	9.090393	0.009335	0.807430	0.080893	255.048185	0.68938
2018-01-03	15583.885538	3.125710	9.886323	0.009592	1.075401	0.098107	248.042194	0.70462
2018-01-04	15976.365194	3.220050	9.675758	0.010098	1.179347	0.218139	244.834372	1.03682
2018-01-05	18336.922980	2.931380	16.488523	0.013841	1.077821	0.231673	254.138525	0.99657
...	...	...	...	...	...	...	...	.
2024-08-12	58804.234500	0.552884	503.472306	0.100570	0.328140	0.127716	59.691360	10.01253
2024-08-13	59350.074333	0.568612	518.752217	0.107647	0.338874	0.126661	61.414267	10.56164
2024-08-14	60601.223178	0.576440	523.553455	0.106469	0.340317	0.128873	63.411783	10.57749
2024-08-15	58739.193822	0.568488	523.842262	0.102529	0.335251	0.130504	63.906408	10.39698
2024-08-16	57624.116929	0.561137	519.989791	0.100275	0.325393	0.130136	65.236601	10.17100

2568 rows × 86 columns

We have 86 coins and will select pairs from among them.

In [6]:
```python
coins_ret = crypto_px / crypto_px.shift() - 1
```

In [7]:
```python
def compute_turnover(port):
    to = (port.fillna(0)-port.shift().fillna(0)).abs().sum(1)
    return to
```

In [8]:
```python
def compute_sharpe_ratio(rets):
    mean_rets = rets.mean()*252
    vol = rets.std()*np.sqrt(252)
    sharpe_ratio = mean_rets / vol
    return sharpe_ratio
```

In [9]:
```python
def compute_stats(rets):
    stats={}
    stats['avg'] = rets.mean()*252
    stats['vol'] = rets.std()*np.sqrt(252)
    stats['sharpe'] = stats['avg']/stats['vol']
    stats['hit_rate'] = rets[rets>0].count() / rets.count()
    stats = pd.DataFrame(stats)
    return stats
```

In [10]:
```python
def drawdown(px):
    return (px / (px.expanding(min_periods=1).max()) - 1)
```

In [11]:
```python
def duration(px):
    peak = px.expanding(min_periods=1).max()
    res = pd.DataFrame(index=px.index,columns=px.columns)
    for col in px.columns:
        for dt in px.index:
            if px.loc[dt,col] >= peak.loc[dt,col]:
                res.loc[dt,col] = 0
            else:
                res.loc[dt,col] = res.loc[:dt,col].iloc[-2] + 1
    return res
```

Pairs Selection

Pairs are selected and updated every 3 months to adapt to the fast-paced cryptocurrency market:

- **Rolling Correlation**: Every six months, compute the rolling correlation of the log prices for each pair of assets over the past 2 year:

$$\text{corr}(\log(p_{i,t}), \log(p_{j,t}))$$

- **Selection Criteria**: For each coin i, select the top n pairs with the highest absolute correlation values, provided:

$$|\text{corr}(p_{i,t}, p_{j,t})| \geq 0.9$$

```
In [12]: def select_pairs(crypto_px, top_n=5, corr_threshold=0.9):
    """
    Compute correlation for all pairs, select the top `n` correlated p
    with absolute correlation >= `corr_threshold`, and display the res

    Parameters:
    crypto_px (DataFrame): DataFrame containing price data of assets.
    top_n (int): The number of top pairs to select for each coin.
    corr_threshold (float): The threshold for absolute correlation to

    Returns:
    list: A sorted list of unique correlated pairs.
    """
    # Compute the correlation matrix
    corr_matrix = crypto_px.corr()

    # Initialize a set to store unique pairs
    final_pairs_set = set()

    # Initialize a dictionary to store top pairs for each coin
    top_pairs_per_coin = {}

    # Iterate over each coin in the correlation matrix
    for coin in corr_matrix.columns:
        # Sort the pairs by the absolute value of the correlation (hig
        sorted_pairs = corr_matrix[coin].sort_values(ascending=False,

        # Filter pairs with absolute correlation >= corr_threshold and
        filtered_pairs = sorted_pairs[(sorted_pairs.abs() >= corr_thre

        # Select the top `n` pairs for this coin
        top_pairs = filtered_pairs.head(top_n)

        # Store the result in the dictionary
        top_pairs_per_coin[coin] = top_pairs

        # Add the selected pairs to the final set
        final_pairs_set.update((coin, pair) for pair in top_pairs.inde

    # Convert the set to a sorted list
    final_pairs = sorted(final_pairs_set)

    # Determine the update date as the end date of crypto_px + 1 day
    update_date = crypto_px.index[-1] + pd.Timedelta(days=1)
    print(f"Pairs Updated date: {update_date.strftime('%Y-%m-%d')}")

    # Display the top correlated pairs for each coin with their actual
    sorted_coins = sorted(top_pairs_per_coin.keys())

    # Print the final pairs and the count
    print(f"Final pairs to be traded: {final_pairs}")
    print(f"Number of pairs to be traded: {len(final_pairs)}")

    return final_pairs
```

Trading Strategy

1. Signal Generation

- **Residual Calculation:**

$$\epsilon_t = \log(p_{i,t}) - (\beta_t \log(p_{j,t}) + \alpha_t)$$

where:

  - $\log(p_{i,t})$ and $\log(p_{j,t})$ are the log prices of coins i and j at time t.
  - β_t is calculated as:

$$\beta_t = \mathrm{Corr}_t \times \frac{\mathrm{Vol}_{j,t}}{\mathrm{Vol}_{i,t}}$$

with:

  - Corr_t is the 90-day rolling correlation between $\log(p_{i,t})$ and $\log(p_{j,t})$.
  - $\mathrm{Vol}_{i,t}$ and $\mathrm{Vol}_{j,t}$ are the 90-day rolling volatilities of $\log(p_{i,t})$ and $\log(p_{j,t})$, respectively.
  - α_t is calculated as:

$$\alpha_t = \mu_{\log(p_{j,t})} - \beta_t \cdot \mu_{\log(p_{i,t})}$$

where:

  - $\mu_{\log(p_{j,t},90)}$ is the 90-day rolling mean of $\log(p_{j,t})$.
  - $\mu_{\log(p_{i,t},90)}$ is the 90-day rolling mean of $\log(p_{i,t})$.

- **Z-Score Calculation:**

$$z_t = \frac{\epsilon_t - \mu_t}{\sigma_t}$$

where:

  - μ_t is the 90-day rolling mean of the spread.
  - σ_t is the 90-day rolling standard deviation of the spread.

2. Portfolio Construction

- **Entry Signals:**
  - **Short** coin i and **long** β_t units of coin j if $z_t > 1$.
  - **Long** coin i and **short** β_t units of coin j if $z_t < -1$.
- **Exit Signals:**
  - Close the position when z_t moves to any of the following thresholds:

$$z_t \geq -\text{threshold} \quad \text{or} \quad z_t \leq \text{threshold}$$

  where threshold is one of the values: 0.1, 0.2, 0.5, or 0.7.

In [13]:

```
def gen_signals(px, pairs, window=90):
    signal_df = {}
    for pair in pairs:
        asset_i, asset_j = pair

        # Forward-fill missing values and replace zeros with NaNs
        px_i = px[asset_i].replace(0, np.nan).ffill()
        px_j = px[asset_j].replace(0, np.nan).ffill()

        # Apply log transformation
        log_px_i = np.log(px_i)
        log_px_j = np.log(px_j)

        # Calculate rolling covariance and variance
        rolling_cov = log_px_i.rolling(window=window, min_periods=1).c
        rolling_var = log_px_i.rolling(window=window, min_periods=1).v

        # Calculate beta and alpha
        beta = rolling_cov / rolling_var
        alpha = log_px_j.rolling(window=window).mean() - beta * log_px

        # Calculate spread for time t using beta and alpha
        spread = log_px_i - (beta * log_px_j + alpha)

        # Calculate rolling mean and standard deviation of the spread
        spread_mean = spread.rolling(window=window, min_periods=1).mea
        spread_std = spread.rolling(window=window, min_periods=1).std(

        # Calculate the z-score for time t using spread(t), mean(t), a
        z_score = (spread - spread_mean) / spread_std

        # Store beta, alpha, spread, and z-score in a multi-level colu
        signal_df[(pair, 'beta')] = beta
        signal_df[(pair, 'alpha')] = alpha
        signal_df[(pair, 'spread')] = spread
        signal_df[(pair, 'z_score')] = z_score

    # Convert the dictionary to a DataFrame
    signal_df = pd.DataFrame(signal_df)

    return signal_df
```

```
In [14]: def gen_port(signal_df, pairs, crypto_px, threshold=0.5):
    # Initialize a DataFrame with the same index and columns as crypto
    pos = pd.DataFrame(index=signal_df.index, columns=crypto_px.column
    for pair in pairs:
        asset_i, asset_j = pair
        # Access z-scores and betas for this pair
        z_scores = signal_df[(pair, 'z_score')]
        betas = signal_df[(pair, 'beta')]

        # Set positions based on z-scores
        pos.loc[z_scores > 1, asset_i] = -1  # Short one unit of asset
        pos.loc[z_scores < -1, asset_i] = 1  # Long one unit of asset_
        pos.loc[(z_scores.abs() <= threshold), asset_i] = 0  # Exit si

        pos.loc[z_scores > 1, asset_j] = betas  # Long beta units of a
        pos.loc[z_scores < -1, asset_j] = -betas  # Short beta units o
        pos.loc[(z_scores.abs() <= threshold), asset_j] = 0  # Exit si

    # Forward-fill missing values
    pos = pos.ffill()
    # Normalize to ensure a fully-invested portfolio
    pos = pos.divide(pos.abs().sum(axis=1), axis=0).fillna(0)
    return pos
```

Performance Evaluation

We evaluate the strategy's performance across four exit thresholds: 0.1, 0.2, 0.5, and 0.7. For each threshold, the following key metrics are calculated:

- **Sharpe Ratio**: Risk-adjusted return of the strategy.
- **Transaction Costs**: Average costs incurred due to trading.
- **Holding Period**: Average number of days a position is held.
- **Turnover**: Average daily proportion of the portfolio that is traded.
- **Annualized Return**: Average yearly return of the strategy.
- **Annualized Volatility**: Standard deviation of returns on an annual basis, indicating risk.

Cryptocurrencies can have commissions of ~7bps. While total slippage is unknown and will depend on the trader’s volume as well, let’s assume another 13 bps. So total all-in execution costs will be 20 bps for market-orders.

In [15]:
```
# Define the thresholds and initialize the arrays for storing metrics
end_of_insample = pd.Timestamp('2019-12-31')
last_available_date = crypto_px.index[-1]
thresholds = [0.1, 0.2, 0.5, 0.7]
metrics = {
    'Sharpe Ratio': np.zeros(len(thresholds)),
    'Return': np.zeros(len(thresholds)),
    'Volatility': np.zeros(len(thresholds)),
    'Holding Period': np.zeros(len(thresholds)),
    'Turnover': np.zeros(len(thresholds)),
    'Transaction Costs': np.zeros(len(thresholds)),
}
# Dictionary to store strat_net_ret for each threshold
strat_net_ret_dict = {}

# Set the start of the out-of-sample period
start_of_out_sample = end_of_insample + pd.DateOffset(days=1)

# Define half-year periods for updating pairs
update_dates = pd.date_range(start=start_of_out_sample, end=crypto_px.

# Initialize an empty DataFrame to store the complete portfolio over a
full_portfolio = pd.DataFrame(index=crypto_px.loc[start_of_out_sample:

# Loop over each threshold to calculate metrics
for i, threshold in enumerate(thresholds):
    for start_date in update_dates:
        # Define the end of the update period (6 months later)
        end_date = start_date + pd.DateOffset(months=3) - pd.DateOffse

        # Adjust end_date if it exceeds the last available date
        if end_date > last_available_date:
            end_date = last_available_date

        # Select the in-sample period for pair selection
        insample_start = start_date - pd.DateOffset(years=2)
        insample_end = start_date - pd.DateOffset(days=1)

        # Re-select pairs based on the ADF test or any other criteria
        updated_pairs = select_pairs(crypto_px.loc[insample_start:insa

        # Generate signals for the new pairs
        signal_df = gen_signals(crypto_px, updated_pairs, window=90)
        signal_df = signal_df.loc[start_date:end_date]

        # Generate portfolio for the selected pairs
        port = gen_port(signal_df, updated_pairs, crypto_px, threshold

        # Store the generated portfolio in the full_portfolio DataFram
        full_portfolio.loc[start_date:end_date, :] = port

    # Calculate out-of-sample daily returns for the entire period
    out_sample_ret = coins_ret.loc[start_of_out_sample:][full_portfoli
    strat_gross_ret = (full_portfolio.shift() * out_sample_ret).sum(ax

    # Calculate net returns with transaction costs
    to = compute_turnover(full_portfolio)
    tcost_bps = 20  # (commissions + slippage)
    strat_net_ret = strat_gross_ret.subtract(to * tcost_bps * 1e-4, f

    # Store the net returns for the current threshold
```

```
    strat_net_ret_dict[threshold] = strat_net_ret

    # Compute Sharpe ratio
    sharpe_ratio = compute_sharpe_ratio(strat_net_ret)
    metrics['Sharpe Ratio'][i] = sharpe_ratio

    # Compute transaction costs
    total_tcost = to * tcost_bps * 1e-4
    metrics['Transaction Costs'][i] = total_tcost.sum()

    # Compute holding period (average number of days a position is hel
    metrics['Holding Period'][i] = 2/to.mean()

    # Store turnover
    metrics['Turnover'][i] = to.mean()

    # Compute return
    returns = strat_net_ret.mean()
    metrics['Return'][i] = returns

    # Compute volatility
    volatility = strat_net_ret.std()
    metrics['Volatility'][i] = volatility

# Convert metrics to a DataFrame for easier visualization
metrics_df = pd.DataFrame(metrics, index=thresholds)
```

```
EC'), ('ZIL', 'CTXC'), ('ZIL', 'GLM'), ('ZIL', 'NULS'), ('ZIL', 'ON
T'), ('ZIL', 'ZRX'), ('ZRX', 'DCR'), ('ZRX', 'ELF'), ('ZRX', 'NUL
S'), ('ZRX', 'XLM'), ('ZRX', 'ZIL')]
Number of pairs to be traded: 364
Pairs Updated date: 2020-04-01
Final pairs to be traded: [('ADA', 'ARDR'), ('ADA', 'GNO'), ('ADA',
'IOST'), ('ADA', 'SNT'), ('ADA', 'XEM'), ('ADX', 'LRC'), ('ADX', 'RE
Q'), ('ADX', 'SNT'), ('ADX', 'STMX'), ('ADX', 'SYS'), ('AMB', 'BL
Z'), ('AMB', 'ELF'), ('AMB', 'GAS'), ('AMB', 'GFT'), ('AMB', 'STRA
X'), ('ANT', 'GNO'), ('ANT', 'ICX'), ('ANT', 'LRC'), ('ANT', 'NEO'),
('ANT', 'SWFTC'), ('ARDR', 'NEO'), ('ARDR', 'QTUM'), ('ARDR', 'SN
T'), ('ARDR', 'STORJ'), ('ARDR', 'XEM'), ('ARK', 'GAS'), ('ARK', 'PI
VX'), ('ARK', 'STORJ'), ('ARK', 'STRAX'), ('ARK', 'SYS'), ('BCH', 'A
DA'), ('BCH', 'DASH'), ('BCH', 'EOS'), ('BCH', 'IOST'), ('BCH', 'IOT
A'), ('BLZ', 'ELF'), ('BLZ', 'GAS'), ('BLZ', 'IOTA'), ('BLZ', 'OM
G'), ('BLZ', 'SNT'), ('BTG', 'ARDR'), ('BTG', 'DASH'), ('BTG', 'WAX
P'), ('BTG', 'XEM'), ('BTG', 'XMR'), ('CTXC', 'ELF'), ('CTXC', 'GA
S'), ('CTXC', 'ICX'), ('CTXC', 'LRC'), ('CTXC', 'SYS'), ('CVC', 'AR
K'), ('CVC', 'BLZ'), ('CVC', 'OMG'), ('CVC', 'SNT'), ('CVC', 'STRA
```

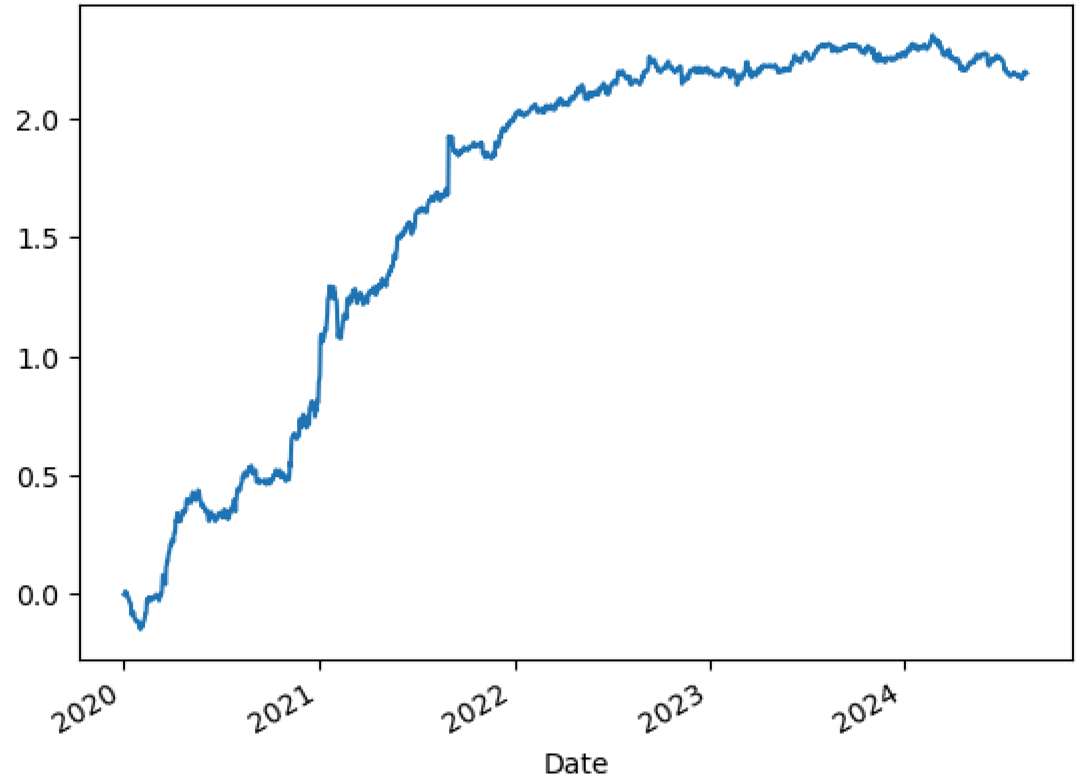
In [16]:
```
print(metrics_df)
```

```
     Sharpe Ratio    Return  Volatility  Holding Period  Turnover  \
0.1      1.566602  0.001296    0.013131        8.785549  0.227647
0.2      1.508085  0.001336    0.014058        8.279351  0.241565
0.5      1.358545  0.001370    0.016012        8.102245  0.246845
0.7      1.333609  0.001484    0.017663        8.315641  0.240511

     Transaction Costs
0.1           0.770356
0.2           0.817455
0.5           0.835324
0.7           0.813888
```

The performance across different thresholds is relatively robust, with only minor variations. Interestingly, the correlation-based method appears to outperform the cointegration approach

In [17]:
```
strat_net_ret = strat_net_ret_dict[0.1]
strat_net_ret.cumsum().plot()
```

Out[17]: `<Axes: xlabel='Date'>`



The major drawback is that the performance has been steady during 2022-2024, even though the pairs are updated frequently.

We also compare the performance of our strategy against our benchmark, specifically a buy-and-hold strategy for Bitcoin. We will evaluate key metrics including alpha and beta, maximum drawdowns and maximum drawdown duration.

In [18]:
```
buy_and_hold_btc = coins_ret['BTC'][start_of_out_sample:]
buy_and_hold_btc
```

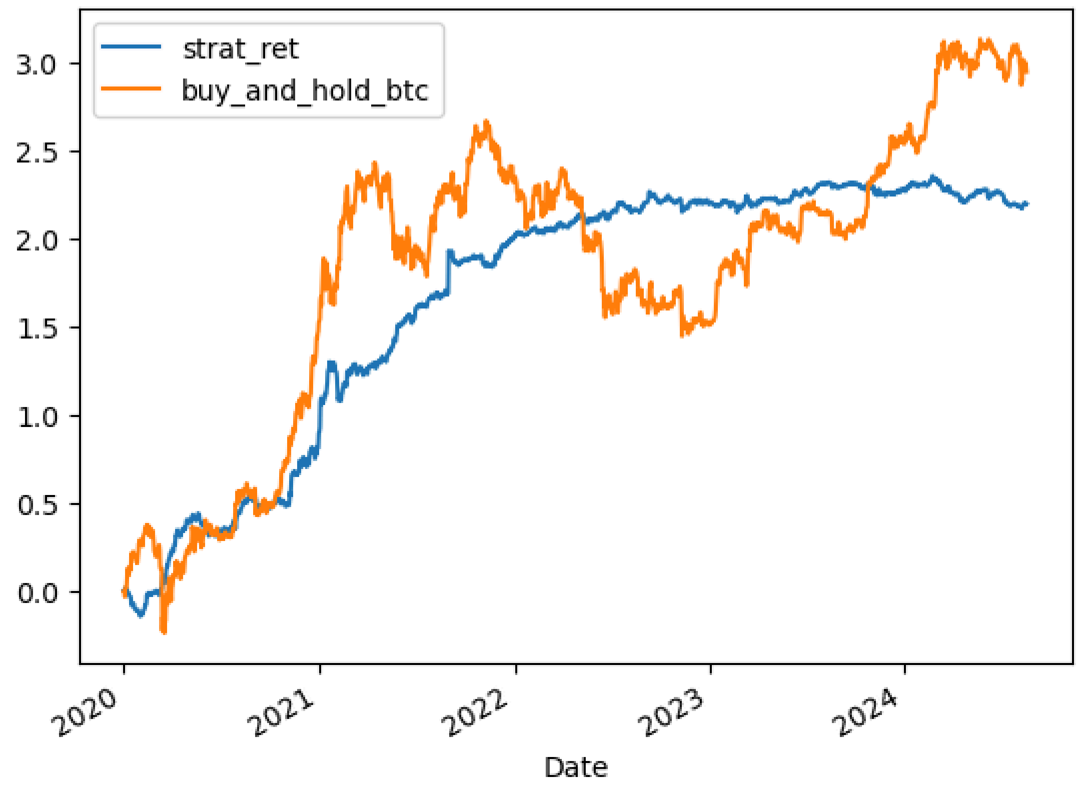
Out[18]:
```
Date
2020-01-01   -0.006253
2020-01-02   -0.000194
2020-01-03   -0.031958
2020-01-04    0.048320
2020-01-05    0.006990
                ...
2024-08-12   -0.034218
2024-08-13    0.009282
2024-08-14    0.021081
2024-08-15   -0.030726
2024-08-16   -0.018984
Name: BTC, Length: 1692, dtype: float64
```

In [19]:
```
full_sample_ret = pd.DataFrame({
    'strat_ret': strat_net_ret,
    'buy_and_hold_btc': buy_and_hold_btc
})

full_sample_stats = compute_stats(full_sample_ret)
print(full_sample_stats)
```

```
                       avg       vol    sharpe  hit_rate
strat_ret         0.326556  0.208449  1.566602  0.524232
buy_and_hold_btc  0.439585  0.535531  0.820839  0.513626
```

In [20]:
```
full_sample_ret.cumsum().plot()
```

Out[20]: <Axes: xlabel='Date'>

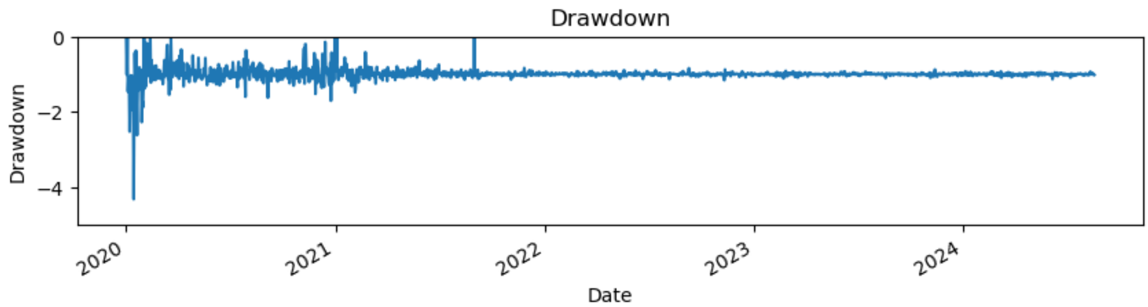


In [21]:
```
corr = full_sample_ret.rolling(252).corr(full_sample_ret['buy_and_holc
vol = full_sample_ret.rolling(252).std()

beta = (corr*vol).divide(vol['buy_and_hold_btc'], 0)

# Computing Point-in-Time Residual Returns
resid = full_sample_ret - beta.multiply(full_sample_ret['buy_and_hold_
print(resid)
print(resid.corr())

# The information ratio
IR = resid.mean()/resid.std()*np.sqrt(252)
print(f"Information ratio is {IR['strat_ret']}")
```

```
             strat_ret  buy_and_hold_btc
Date
2020-01-01         NaN               NaN
2020-01-02         NaN               NaN
2020-01-03         NaN               NaN
2020-01-04         NaN               NaN
2020-01-05         NaN               NaN
...                ...               ...
2024-08-12 -0.003478      1.457168e-16
2024-08-13   0.00932     -3.989864e-17
2024-08-14 -0.000546     -9.367507e-17
2024-08-15 -0.002998      1.353084e-16
2024-08-16  -0.00505      8.326673e-17

[1692 rows x 2 columns]
                  strat_ret  buy_and_hold_btc
strat_ret          1.000000          0.028531
buy_and_hold_btc   0.028531          1.000000
Information ratio is 1.2531577620603185
```

The correlation of the residual returns with Bitcoin is 0.0285, suggesting that the strategy captures true alpha

In [22]:
```
# Plot drawdown
dd = drawdown(full_sample_ret['strat_ret'])
plt.figure(figsize=(10, 2))
dd.plot()
plt.ylim(-5, 0)
plt.title('Drawdown')
plt.xlabel('Date')
plt.ylabel('Drawdown')
plt.show()
print(f"The max drawdown is {dd.min()}")
```




```
The max drawdown is -4.325608146626549
```

In [23]:
```
ddd = duration(full_sample_ret.cumsum())
ddd
```

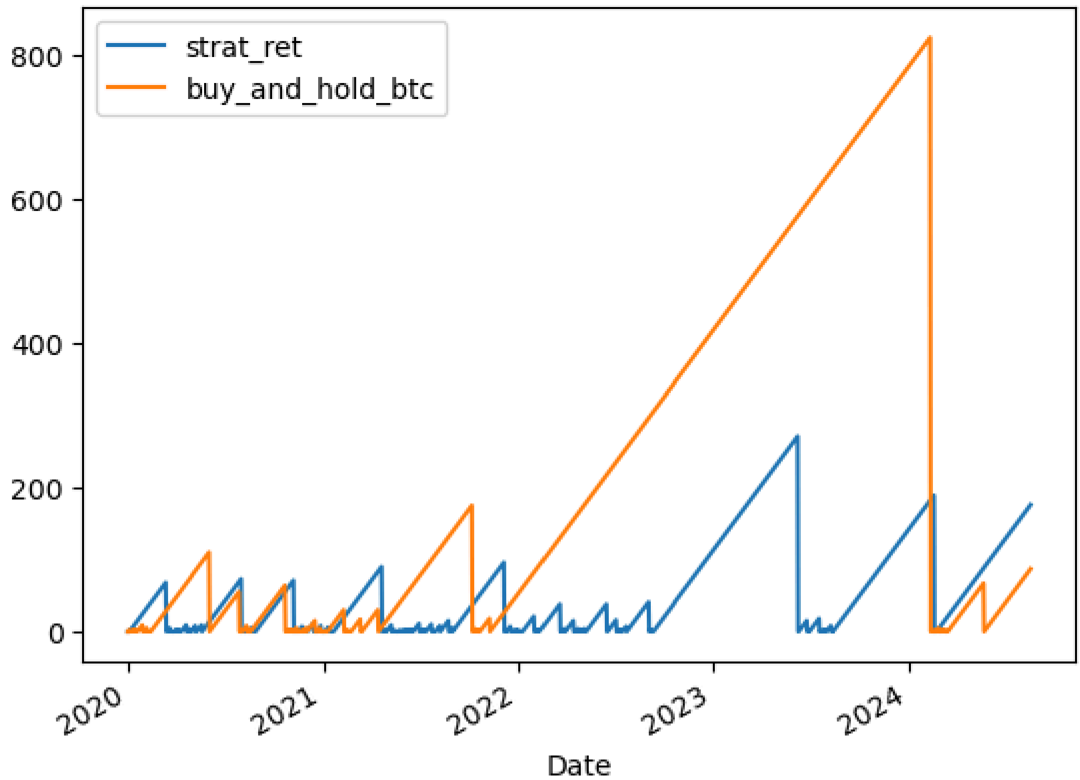
Out[23]:

	strat_ret	buy_and_hold_btc
Date		
2020-01-01	0	0
2020-01-02	0	1
2020-01-03	1	2
2020-01-04	0	0
2020-01-05	1	0
...	...	...
2024-08-12	172	83
2024-08-13	173	84
2024-08-14	174	85
2024-08-15	175	86
2024-08-16	176	87

1692 rows × 2 columns

In [24]:
```
ddd.plot()
```

Out[24]:
```
<Axes: xlabel='Date'>
```



In [25]:
```
# maximum drawdown duration
ddd.max()
```

Out[25]:
```
strat_ret          271
buy_and_hold_btc   824
dtype: object
```

Conclusion:

1. The correlation-based strategy outperforms the cointegration-based strategy.
2. Updating pairs every 3 months yields better performance, aligning with the fast-moving crypto market.
3. Selecting the top 3 to 5 pairs for each coin works better than choosing just the top pair.

In [ ]:
```

```