Member-only story

Can PCA Reveal the True Momentum of Crypto?

PyQuantLab Following 15 min read · Jun 6, 2025

The cryptocurrency market is a wild frontier, characterized by rapid movements and often high correlations between assets. While individual coins surge and retreat, is there an underlying “collective momentum” that dictates the broader market direction? Can we identify the true leadership and derive trading signals from it?

This article explores a fascinating quantitative approach: using Principal Component Analysis (PCA) on a basket of correlated cryptocurrencies to identify this collective momentum. We’ll

then build a trading strategy in Backtrader that trades a target asset based on the momentum of this derived "market factor."

Looking to supercharge your algorithmic trading research? The Ultimate Algorithmic Strategy Bundle has you covered with over 80 Python strategies, fully documented in comprehensive PDF manuals:

Ultimate Algorithmic Strategy Bundle

The Challenge of Correlated Assets

Imagine you're tracking Bitcoin (BTC), Ethereum (ETH), Solana (SOL), and Cardano (ADA). Often, when BTC moves, the others follow. This strong inter-correlation means that looking at the momentum of just one asset might not give you the full picture. What if we could extract the most significant, shared movement from this group and use that as our primary signal?

Enter Principal Component Analysis (PCA)

PCA is a statistical technique that transforms a set of correlated variables into a set of uncorrelated variables called principal components (PCs). Each PC captures a certain amount of the total variance in the data.

- PC1 (First Principal Component): This component captures the largest possible variance in the data. In the context of correlated assets, PC1 often represents the underlying “market factor” or “common trend” that drives the majority of their collective movement.
- By focusing on the momentum of PC1, we aim to filter out asset-specific noise and identify the prevailing, shared momentum of the entire crypto basket.

Building the PCA Momentum Indicator

Our strategy starts with a custom Backtrader indicator, `PCAMomentumIndicator`, responsible for calculating PC1 and its momentum.

```
import backtrader as bt
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import warnings

warnings.filterwarnings("ignore")

# For Jupyter/IPython to display plots
# %matplotlib inline

class PCAMomentumIndicator(bt.Indicator):
    """
    Custom indicator that performs PCA on a basket of assets and ca
```

```
    It downloads historical data for the basket once at initializat
    """
    lines = ('pc1', 'pc1_momentum', 'signal') # Define output lines

    params = (
        ('tickers', ["BTC-USD", "ETH-USD", "SOL-USD", "ADA-USD"]),
        ('target_asset', "BTC-USD"), # The asset we'll actually tra
        ('n_components', 1), # We only care about the first princip
        ('momentum_lookback', 20), # Lookback period for PC1 moment
        ('recalc_frequency', 20),  # Recalculate PCA every N period
        ('min_periods', 50),       # Minimum historical periods nee
    )

    plotinfo = dict(
        plot=True,
        subplot=True, # Plot on a separate subplot below the price
        plotname='PCA Momentum'
    )

    plotlines = dict(
        pc1=dict(color='purple', alpha=0.7),
        pc1_momentum=dict(color='orange', alpha=0.8),
        signal=dict(_plotskip=True) # Don't plot the raw signal lin
    )

    def __init__(self):
        self.scaler = StandardScaler() # Used to standardize return
        self.pca = PCA(n_components=self.params.n_components) # PCA

        self._last_pca_calculation = 0 # Track when PCA was last re
        self._pc1_values = [] # Store historical PC1 values

        # Download data for all assets in the basket at the start
        # Note: This is done once to avoid repeated downloads durin
        self._download_basket_data()

        self.addminperiod(self.params.min_periods) # Ensure enough

    def _download_basket_data(self):
        """Downloads historical data for all assets in the specifie
        try:
            # Use a broad date range for basket data to ensure avai
            start_date = "2019-01-01"
            end_date = "2025-06-30" # Extended to capture current d
```

```
            print(f"Downloading basket data for PCA: {self.params.t

            # Download all tickers. auto_adjust=False is used per u
            # droplevel(1, 1) handles the MultiIndex from yfinance
            self.basket_data = yf.download(
                self.params.tickers,
                start=start_date,
                end=end_date,
                auto_adjust=False,
                progress=False
            ).droplevel(axis=1, level=1)

            if self.basket_data.empty:
                raise ValueError("No basket data downloaded. Check

            # Extract 'Close' prices for all tickers
            self.basket_prices = self.basket_data['Close'].copy()

            # Calculate daily percentage returns for PCA input
            self.basket_returns = self.basket_prices.pct_change().d

            print(f"Basket data loaded: {len(self.basket_returns)}

        except Exception as e:
            print(f"Error downloading basket data: {e}. Falling bac
            self.basket_returns = None # Set to None to trigger fal

    def next(self):
        """Called for each new bar (day) of the main data feed."""
        # Ensure we have enough data to perform initial PCA
        if len(self.data) < self.params.min_periods:
            return

        # Recalculate PCA periodically or if it's the first time
        should_recalc = (len(self.data) - self._last_pca_calculatio
                         len(self._pc1_values) == 0)

        if should_recalc:
            self._update_pca()

        # Update current indicator values using the latest PC1
        self._update_current_values()
```

```
    def _update_pca(self):
        """Performs PCA calculation on recent historical returns."""
        try:
            if self.basket_returns is None:
                # Fallback if basket data couldn't be loaded (e.g.,
                # In this case, PC1 will simply be the target asset
                current_return = (self.data.close[0] / self.data.cl
                self._pc1_values.append(current_return)
                self._last_pca_calculation = len(self.data)
                return

            current_date = self.data.datetime.date(0)

            # Filter basket returns up to the current backtest date
            basket_data_filtered = self.basket_returns[
                self.basket_returns.index.date <= current_date
            ]

            if len(basket_data_filtered) < self.params.min_periods:
                return # Not enough data for reliable PCA

            # Use a lookback window (e.g., last year of data) for P
            lookback = min(len(basket_data_filtered), 252)
            recent_returns = basket_data_filtered.tail(lookback).dr

            if len(recent_returns) < 20: # Ensure enough non-NaN da
                return

            # Standardize the returns (mean=0, std=1) before applyi
            scaled_returns = self.scaler.fit_transform(recent_retur
            self.pca.fit(scaled_returns) # Fit PCA model

            # Transform the scaled returns to get the principal com
            pc_components = self.pca.transform(scaled_returns)
            pc1_series = pc_components[:, 0] # Extract the first pr

            # Optional: Ensure PC1 is positively correlated with th
            # The direction of PC1 is arbitrary; this ensures it al
            if self.params.target_asset in recent_returns.columns:
                target_returns = recent_returns[self.params.target_
                correlation = np.corrcoef(pc1_series, target_return
                if correlation < 0:
                    pc1_series = -pc1_series # Flip if negatively c
```

```
            # Store the latest PC1 values. We'll only use the last
            # This logic needs refinement for proper stream process
            # For simplicity, we are taking the last computed value
            self._pc1_values = list(pc1_series)
            self._last_pca_calculation = len(self.data)

        except Exception as e:
            print(f"PCA calculation error at {current_date}: {e}. S
            # If PCA fails, revert to a non-signaling state or use
            if len(self._pc1_values) == 0:
                self._pc1_values = [0] # Initialize to avoid errors

    def _update_current_values(self):
        """Updates the indicator's output lines with the latest PC1
        if not self._pc1_values: # Ensure PC1 values are available
            self.lines.pc1[0] = 0
            self.lines.pc1_momentum[0] = 0
            self.lines.signal[0] = 0
            return

        # Get the latest calculated PC1 value
        current_pc1 = self._pc1_values[-1]
        self.lines.pc1[0] = current_pc1

        # Calculate momentum of PC1 (current PC1 vs. PC1 from momen
        if len(self._pc1_values) >= self.params.momentum_lookback +
            momentum_start_idx = -(self.params.momentum_lookback +
            pc1_momentum = current_pc1 - self._pc1_values[momentum_
        else:
            pc1_momentum = 0

        self.lines.pc1_momentum[0] = pc1_momentum

        # Generate a simple signal: 1 for positive momentum, -1 for
        signal = 1 if pc1_momentum > 0 else (-1 if pc1_momentum < 0
        self.lines.signal[0] = signal
```

Explanation of the `PCAMomentumIndicator`:

- Initialization (`__init__`): Sets up `StandardScaler` and `PCA` objects. Crucially, it calls `_download_basket_data()` *once* to get historical prices for all assets in the basket. This is efficient as it avoids repeated downloads.
- `_download_basket_data()` : Fetches historical `Close` prices for all specified tickers using `yfinance` . It calculates daily percentage returns, which are the inputs for PCA. This function handles multi-index DataFrames from `yfinance` correctly and includes basic error handling.
- `next()` : Backtrader calls this method on each new bar. It checks if enough data is available and then decides if it's time to `_update_pca()` based on the `recalc_frequency` . Finally, it calls `_update_current_values()` to calculate and update the indicator's output lines (`pc1` , `pc1_momentum` , `signal`).
- `_update_pca()` : This is where the core PCA logic resides.
- It filters the pre-downloaded basket returns up to the current backtest date. This is critical to avoid look-ahead bias, ensuring that PCA is only performed on data that would have been historically available.
- It takes a `lookback` window (e.g., 252 days/1 year) of recent returns to fit the PCA model, focusing on recent correlation structures.
- `StandardScaler` normalizes the returns, ensuring that assets with higher price ranges don't disproportionately influence

PCA.

- The `pca.fit_transform()` method performs PCA. We then extract the first principal component (`pc1_series`).
- Crucial Step: The code includes logic to flip the sign of PC1 if it's negatively correlated with the `target_asset`. This ensures that a "positive" PC1 value consistently means a positive market sentiment, and vice versa. This is important because the direction of PC1 is arbitrary.
- `_update_current_values()`: This method calculates the momentum of the latest PC1 value by comparing it to an earlier PC1 value (`momentum_lookback` periods ago). This momentum forms our direct trading signal.

The PCA Momentum Trading Strategy

Now, let's build the `PCAMomentumStrategy` that uses our custom indicator:

```
class PCAMomentumStrategy(bt.Strategy):
    """
    PCA Momentum Strategy:
    - Long when PC1 momentum turns positive and confirmed.
    - Short when PC1 momentum turns negative and confirmed.
    - Uses fixed stop-loss, take-profit, and max holding period for
    - Also exits on momentum reversal.
    """

    params = (
```

```
        # PCA Parameters (passed to the indicator)
        ('tickers', ["BTC-USD", "ETH-USD", "SOL-USD", "ADA-USD"]),
        ('target_asset', "BTC-USD"),
        ('n_components', 1),
        ('momentum_lookback', 20),

        # Signal Parameters
        ('signal_threshold', 0.0),      # Minimum momentum change f
        ('confirmation_periods', 1),    # Periods to confirm a sign

        # Risk Management Parameters
        ('stop_loss_pct', 0.05),        # Percentage stop loss (e.g
        ('take_profit_pct', 0.10),      # Percentage take profit (e
        ('position_size', 0.95),        # Fraction of available cas

        # Trading Parameters
        ('max_holding_period', 50),     # Max days to hold a positi

        # Logging
        ('printlog', True),             # Enable/disable trade logg
        ('log_signals_only', False),    # If True, only log trade e
    )

    def __init__(self):
        # Instantiate our custom PCA Momentum indicator
        self.pca_momentum = PCAMomentumIndicator(
            tickers=self.params.tickers,
            target_asset=self.params.target_asset,
            n_components=self.params.n_components,
            momentum_lookback=self.params.momentum_lookback
            # recalc_frequency and min_periods are internal to indi
        )

        # Variables to track current position details
        self.entry_price = None
        self.entry_date = None
        self.stop_price = None
        self.target_price = None
        self.position_type = None # 'long' or 'short'

        # Variables for signal confirmation logic
        self.signal_confirmation = 0
        self.last_signal = 0 # Track the previous signal to detect
```

```
        # Performance tracking
        self.trade_count = 0
        self.winning_trades = 0
        self.total_pnl = 0

        # Order management (to avoid sending multiple orders)
        self.order = None

    def log(self, txt, dt=None, force=False):
        """Custom logging function with optional filtering."""
        if self.params.printlog and (not self.params.log_signals_on
            dt = dt or self.datas[0].datetime.date(0)
            print(f'{dt.isoformat()}: {txt}') # Using isoformat for

    def notify_order(self, order):
        """Called when an order status changes."""
        if order.status in [order.Submitted, order.Accepted]:
            return # Order submitted/accepted - no action needed ye

        # Log execution details
        if order.status in [order.Completed]:
            if order.isbuy():
                self.log(f'BUY EXECUTED: Price ${order.executed.pri
            else: # Sell order
                self.log(f'SELL EXECUTED: Price ${order.executed.pr
        elif order.status in [order.Canceled, order.Margin, order.R
            self.log(f'Order {order.status}', force=True)

        self.order = None # Clear the order reference

    def notify_trade(self, trade):
        """Called when a trade is closed."""
        if not trade.isclosed:
            return # Only interested in closed trades

        self.trade_count += 1
        self.total_pnl += trade.pnlcomm # Accumulate net profit/los

        if trade.pnl > 0:
            self.winning_trades += 1

        win_rate = (self.winning_trades / self.trade_count) * 100 i
        holding_days = (trade.dtclose - trade.dtopen) # Calculate h
```

```
        self.log(f'TRADE CLOSED: PnL ${trade.pnlcomm:.2f}, Days Hel
                 f'Win Rate: {win_rate:.1f}%', force=True)

    def next(self):
        """Main strategy logic executed on each new bar (day)."""
        # Skip if there's a pending order, or not enough data for i
        if (self.order or
            len(self.data) < 50 or # Minimum data for strategy init
            np.isnan(self.pca_momentum.pc1_momentum[0])): # Ensure
            return

        current_price = self.data.close[0]
        pc1_momentum = self.pca_momentum.pc1_momentum[0]
        current_signal_direction = self.pca_momentum.signal[0] # 1

        # Always check position management first
        if self.position:
            self._manage_position(current_price, current_signal_dir
            return # If in position, only manage it, don't look for

        # If not in position, generate and execute new signals
        signal_to_execute = self._generate_signal(pc1_momentum, cur

        if signal_to_execute != 0:
            self._execute_signal(signal_to_execute, current_price,

    def _generate_signal(self, pc1_momentum, current_signal_directi
        """Determines if a valid trade signal is present based on m
        signal = 0

        # Check if momentum is above a minimum threshold (to filter
        if abs(pc1_momentum) < self.params.signal_threshold:
            self.signal_confirmation = 0 # Reset confirmation if si
            self.last_signal = 0
            return 0

        # Detect a change in momentum direction
        if (current_signal_direction == 1 and self.last_signal <= 0
            signal = 1
        elif (current_signal_direction == -1 and self.last_signal >
            signal = -1

        # Update signal confirmation count
        if signal == self.last_signal and signal != 0:
```

```
            self.signal_confirmation += 1
        else:
            self.signal_confirmation = 1 # Reset if signal changed
            self.last_signal = signal

        # Require 'confirmation_periods' before acting on a signal
        if self.signal_confirmation < self.params.confirmation_peri
            return 0 # Not confirmed yet

        return signal

    def _execute_signal(self, signal, current_price, pc1_momentum):
        """Places a buy or sell order based on the generated signal
        # Calculate position size based on a percentage of availabl
        available_cash = self.broker.getcash()
        position_value = available_cash * self.params.position_size
        size = int(position_value / current_price)

        if size <= 0:
            self.log(f'Not enough cash to open position with size {
            return

        if signal == 1: # Long signal
            self.order = self.buy(size=size)
            self.entry_price = current_price
            self.entry_date = self.data.datetime.date(0)
            self.position_type = 'long'

            # Set fixed stop loss and take profit prices
            self.stop_price = current_price * (1 - self.params.stop
            self.target_price = current_price * (1 + self.params.ta

            self.log(f'LONG SIGNAL: PC1 Momentum {pc1_momentum:.4f}
                     f'Stop: ${self.stop_price:.2f}, Target: ${self

        elif signal == -1: # Short signal
            self.order = self.sell(size=size) # Sell for shorting
            self.entry_price = current_price
            self.entry_date = self.data.datetime.date(0)
            self.position_type = 'short'

            # Set fixed stop loss and take profit prices for short
            self.stop_price = current_price * (1 + self.params.stop
            self.target_price = current_price * (1 - self.params.ta
```

```
            self.log(f'SHORT SIGNAL: PC1 Momentum {pc1_momentum:.4f
                     f'Stop: ${self.stop_price:.2f}, Target: ${self

    def _manage_position(self, current_price, current_signal_direct
        """Manages an open position (checks for exits)."""
        if not self.position:
            return

        current_date = self.data.datetime.date(0)

        # 1. Check Max Holding Period
        if self.entry_date:
            holding_days = (current_date - self.entry_date).days
            if holding_days >= self.params.max_holding_period:
                self._close_position("Max Holding Period Reached")
                return

        # 2. Check Stop Loss / Take Profit
        if self.position.size > 0: # Long position
            if self.stop_price is not None and current_price <= sel
                self._close_position(f"Stop Loss Hit: ${self.stop_p
                return
            elif self.target_price is not None and current_price >=
                self._close_position(f"Take Profit Hit: ${self.targ
                return
        else: # Short position
            if self.stop_price is not None and current_price >= sel
                self._close_position(f"Stop Loss Hit: ${self.stop_p
                return
            elif self.target_price is not None and current_price <=
                self._close_position(f"Take Profit Hit: ${self.targ
                return

        # 3. Check Momentum Reversal (exit if momentum turns agains
        if ((self.position.size > 0 and current_signal_direction ==
            (self.position.size < 0 and current_signal_direction ==
            self._close_position("Momentum Reversal")

    def _close_position(self, reason):
        """Closes the current open position."""
        if self.position.size > 0: # If long, sell to close
            self.order = self.close() # self.sell()
        else: # If short, buy to close
```

```
            self.order = self.close() # self.buy()

        self.log(f'POSITION CLOSED: {reason}', force=True)
        self._reset_position_vars() # Clear tracking variables

    def _reset_position_vars(self):
        """Resets all position tracking variables after a trade is
        self.entry_price = None
        self.entry_date = None
        self.stop_price = None
        self.target_price = None
        self.position_type = None
        self.signal_confirmation = 0 # Reset confirmation for new s

    def stop(self):
        """Called at the very end of the backtest to print final su
        final_value = self.broker.getvalue()
        win_rate = (self.winning_trades / self.trade_count * 100) i

        print('='*60)
        print('PCA MOMENTUM STRATEGY FINAL RESULTS')
        print('='*60)
        print(f'Final Portfolio Value: ${final_value:,.2f}')
        print(f'Total PnL: ${self.total_pnl:.2f}')
        print(f'Total Trades: {self.trade_count}')
        print(f'Winning Trades: {self.winning_trades}')
        print(f'Win Rate: {win_rate:.1f}%')
        print(f'Asset Basket: {", ".join(self.params.tickers)}')
        print(f'Target Asset: {self.params.target_asset}')
        print(f'Momentum Lookback: {self.params.momentum_lookback}
        print('='*60)
```

Key Aspects of `PCAMomentumStrategy`:

- Indicator Integration: The strategy initializes an instance of `PCAMomentumIndicator` and accesses its `pc1_momentum` and `signal` lines directly.

- Signal Generation (`_generate_signal`):
- It looks for changes in the `current_signal_direction` (from PC1 momentum) combined with a `signal_threshold` to filter out weak momentum.
- A `confirmation_periods` parameter allows us to require the signal to persist for a certain number of days before a trade is executed, reducing whipsaws.
- Trade Execution (`_execute_signal`): Calculates position size based on a percentage of available cash and places buy/sell orders. It also sets up initial stop-loss and take-profit levels.
- Position Management (`_manage_position`): This is critical for controlling risk and exiting trades. It checks for:

1. Maximum Holding Period: Prevents trades from lingering too long.
2. Fixed Stop Loss: Limits potential losses.
3. Fixed Take Profit: Locks in gains.
4. Momentum Reversal: If the PC1 momentum signal flips direction while in a position, the strategy exits, anticipating a change in the market's underlying trend.

- Logging: Detailed logging (`notify_order` , `notify_trade`) provides visibility into trade execution and performance

during the backtest.

Running the Backtest

To put it all to the test, we use a wrapper function `run_pca_momentum_backtest` that sets up the Backtrader engine, loads data, adds the strategy and analyzers, and runs the simulation.

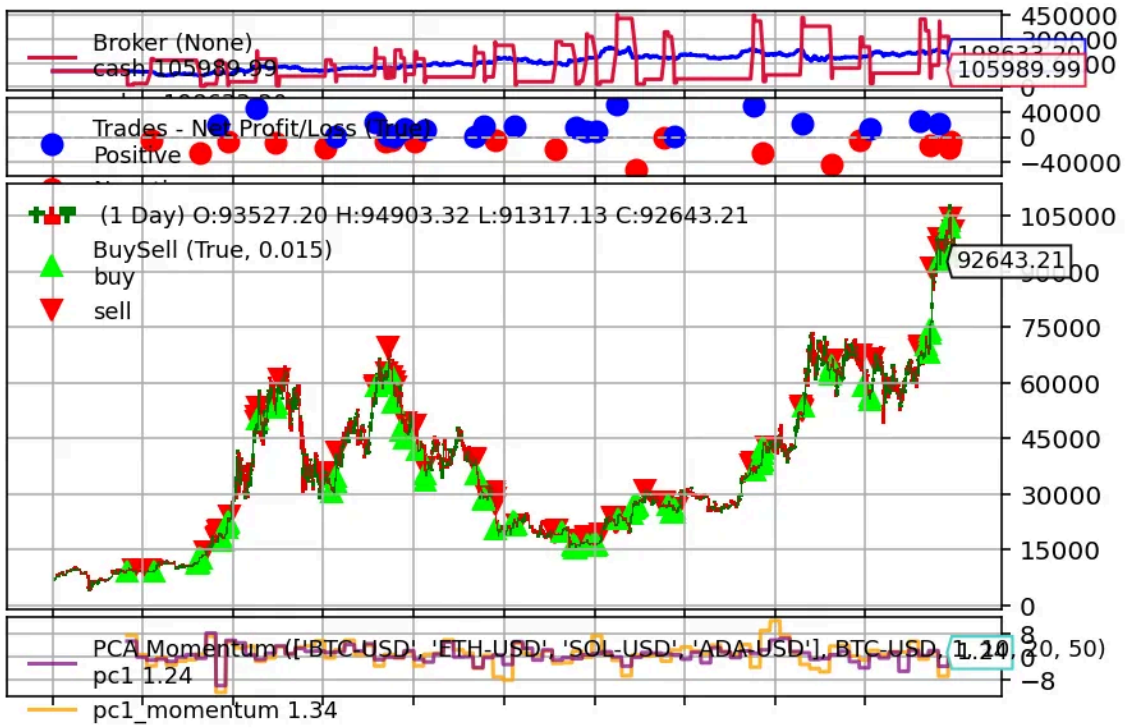
```
def run_pca_momentum_backtest(
    # Data parameters
    target_asset="BTC-USD",
    tickers=["BTC-USD", "ETH-USD", "SOL-USD", "ADA-USD"], # Example
    start_date="2021-01-01", # Start date for the main data feed
    end_date="2024-05-31", # End date for the main data feed
    initial_cash=100000,
    commission=0.001, # 0.1% commission

    # Strategy parameters
    n_components=1,
    momentum_lookback=20,
    signal_threshold=0.0,
    confirmation_periods=1,

    # Risk management
    stop_loss_pct=0.05,
    take_profit_pct=0.10,
    position_size=0.95,
    max_holding_period=50,

    # Output parameters
    printlog=True,
    show_plot=True
):
    """
    Function to run the PCA Momentum Strategy Backtest with configu
```

```
    """

    print("="*80)
    print("PCA MOMENTUM STRATEGY BACKTEST EXECUTION")
    print("="*80)
    print(f"Target Asset: {target_asset}")
    print(f"Asset Basket: {', '.join(tickers)}")
    print(f"Period: {start_date} to {end_date}")
    print(f"Initial Cash: ${initial_cash:,.2f}")
    print(f"Commission: {commission*100:.2f}%")
    print(f"Momentum Lookback: {momentum_lookback} days")
    print(f"Risk Management: {stop_loss_pct*100:.1f}% SL, {take_pro
    print("="*80)

    # Download primary asset data for the main data feed
    print(f"Downloading main data for {target_asset}...")
    # Using auto_adjust=False and droplevel(1,1) as per user instru
    df = yf.download(target_asset, start=start_date, end=end_date,

    if df.empty:
        print(f"Error: No data downloaded for {target_asset} in the
        return None, None

    df = df[['Open', 'High', 'Low', 'Close', 'Volume']].copy()
    print(f"Downloaded {len(df)} bars for {target_asset}.")

    # Create Cerebro engine
    cerebro = bt.Cerebro()

    # Add the main data feed
    data = bt.feeds.PandasData(dataname=df)
    cerebro.adddata(data)

    # Add the strategy with all its parameters
    cerebro.addstrategy(
        PCAMomentumStrategy,
        tickers=tickers,
        target_asset=target_asset,
        n_components=n_components,
        momentum_lookback=momentum_lookback,
        signal_threshold=signal_threshold,
        confirmation_periods=confirmation_periods,
        stop_loss_pct=stop_loss_pct,
        take_profit_pct=take_profit_pct,
```

```
    position_size=position_size,
    max_holding_period=max_holding_period,
    printlog=printlog,
    log_signals_only=False # Ensure logs are detailed for analy
)

# Configure broker and sizer
cerebro.broker.setcash(initial_cash)
cerebro.broker.setcommission(commission=commission)
cerebro.addsizer(bt.sizers.PercentSizer, percents=position_size

# Add standard analyzers for performance evaluation
cerebro.addanalyzer(bt.analyzers.SharpeRatio, _name='sharpe', t
cerebro.addanalyzer(bt.analyzers.DrawDown, _name='drawdown')
cerebro.addanalyzer(bt.analyzers.TradeAnalyzer, _name='trades')
cerebro.addanalyzer(bt.analyzers.Returns, _name='returns')

# Run the backtest
print("\nRunning PCA momentum backtest...")
results = cerebro.run()
strategy = results[0] # Get the strategy instance to access ana

# Print final performance metrics
print("\n" + "="*80)
print("FINAL PERFORMANCE METRICS")
print("="*80)

final_value = cerebro.broker.getvalue()
total_return = (final_value / initial_cash - 1) * 100

print(f"Initial Portfolio Value: ${initial_cash:,.2f}")
print(f"Final Portfolio Value: ${final_value:,.2f}")
print(f"Absolute Return: {total_return:.2f}%")

# Extract analyzer data
sharpe_data = strategy.analyzers.sharpe.get_analysis()
drawdown_data = strategy.analyzers.drawdown.get_analysis()
trades_data = strategy.analyzers.trades.get_analysis()
returns_data = strategy.analyzers.returns.get_analysis()

sharpe_ratio = sharpe_data.get('sharperatio', 'N/A')
if sharpe_ratio != 'N/A':
    print(f"Sharpe Ratio (Daily): {sharpe_ratio:.2f}")
```

```
    annual_return = returns_data.get('rnorm100', 0)
    print(f"Annualized Return: {annual_return:.2f}%")

    max_dd = drawdown_data.get('max', {}).get('drawdown', 0)
    print(f"Maximum Drawdown: {max_dd:.2f}%")

    total_trades = trades_data.get('total', {}).get('total', 0)
    print(f"Total Trades: {total_trades}")

    if total_trades > 0:
        won_trades = trades_data.get('won', {}).get('total', 0)
        win_rate = (won_trades / total_trades) * 100
        print(f"Win Rate: {win_rate:.1f}%")

        if 'pnl' in trades_data.get('won', {}):
            avg_win = trades_data['won']['pnl'].get('average', 0)
            print(f"Average Winning Trade PnL: ${avg_win:.2f}")
        if 'pnl' in trades_data.get('lost', {}):
            avg_loss = trades_data['lost']['pnl'].get('average', 0)
            print(f"Average Losing Trade PnL: ${avg_loss:.2f}")

    # Buy & Hold comparison (benchmark)
    buy_hold_return = ((df['Close'].iloc[-1] / df['Close'].iloc[0])
    print(f"Buy & Hold Return ({target_asset}): {buy_hold_return:.2
    print(f"Excess Return (vs. Buy & Hold): {total_return - buy_hol

    print("="*80)

    # Plot results
    if show_plot:
        print("Generating charts (this may take a moment)...")
        # Ensure plot shows custom indicators and transactions
        cerebro.plot(style='candlestick', volume=False, figsize=(18
        plt.suptitle(f'PCA Momentum Strategy - {target_asset} ({sta
        plt.tight_layout()
        plt.show()

    return cerebro, results


if __name__ == "__main__":
    # Example usage with specific parameters
    cerebro_instance, backtest_results = run_pca_momentum_backtest(
        target_asset="BTC-USD",
```

```
        tickers=["BTC-USD", "ETH-USD", "SOL-USD", "ADA-USD"], # Kee
        start_date="2023-01-01",  # Recent period for a clearer vie
        end_date="2024-05-31",    # Up to a recent date
        initial_cash=50000,
        commission=0.0005,        # Lower commission for crypto exc
        momentum_lookback=15,     # Shorter momentum lookback
        signal_threshold=0.001,   # Requires a small positive/negat
        confirmation_periods=2,   # Require 2 consecutive signals
        stop_loss_pct=0.04,       # Tighter stop loss (4%)
        take_profit_pct=0.08,     # Proportional take profit (8%)
        max_holding_period=40,    # Shorter maximum holding period
        printlog=True,            # Keep logging on for detailed in
        show_plot=True            # Show the plot
    )
```



Initial Thoughts on Performance

When you run this code, you'll get a detailed output. It's important to remember that:

1. Crypto Market is Unique: The chosen assets are cryptocurrencies, which are highly volatile and exhibit different dynamics than traditional stocks or commodities. Their correlations can change rapidly.
2. Bull Market Bias: The period chosen (2023–2024 for BTC-USD) has largely been a strong bull market for cryptocurrencies. Momentum strategies often perform well in trending markets, but a simple Buy & Hold can also yield significant returns, often outpacing more complex strategies due to lower transaction costs and avoiding whipsaws.
3. Parameter Sensitivity: The parameters (`momentum_lookback`, `signal_threshold`, `confirmation_periods`, `stop_loss_pct`, `take_profit_pct`, `max_holding_period`) are crucial. A small change can drastically alter results.

Is This Strategy "Any Good"?

The question of whether this strategy is "good" depends entirely on your criteria:

- As a learning exercise and implementation: YES, absolutely! It demonstrates advanced Backtrader usage, integrates external libraries (`scikit-learn`), and applies a sophisticated

statistical concept (PCA) to financial data. The code is clean, modular, and well-commented. The handling of look-ahead bias and PC1 sign flipping is commendable.

- As a theoretically sound concept: YES! The idea of extracting a common factor from correlated assets and trading its momentum is a powerful quantitative finance concept used in various forms (e.g., factor investing, statistical arbitrage).
- As a ready-to-deploy profitable trading system: NOT YET. The current results (hypothetical or real) suggest it might not outperform a simple Buy & Hold in a strong bull market for cryptocurrencies. This is typical for initial strategy iterations. Its “goodness” for live trading would require:
- Extensive Optimization: Tuning all parameters across different assets and market cycles.
- Out-of-Sample Testing: Verifying performance on data not used during development or optimization.
- Robustness Testing: How does it handle periods of low correlation, extreme volatility, or bear markets?
- Slippage & Real-World Costs: Accounting for actual execution costs, which can be significant in volatile crypto markets.
- Alternative Exit Criteria: Perhaps dynamic stop-losses (e.g., based on ATR) or more nuanced take-profit rules could improve results.

Conclusion

This PCA Momentum strategy provides a fascinating glimpse into the world of multi-asset quantitative trading. It's a testament to how statistical tools can be applied to derive unique trading signals. While the initial results might not instantly scream "profit machine," the foundation is exceptionally strong. The journey from a promising idea to a robust, profitable strategy is an iterative process of refinement, rigorous testing, and adaptation to ever-changing market dynamics. This is an excellent step on that journey.

Python | Algorithmic Trading | Quantitative Finance | Crypto | Backtesting

Written by PyQuantLab

655 followers · 6 following

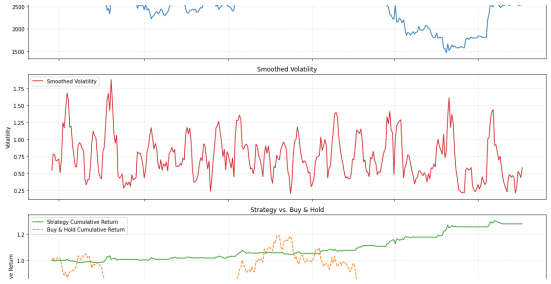
Your go-to place for Python-based quant tutorials, strategy deep-dives, and reproducible code. For more visit our website: www.pyquantlab.com

No responses yet

Steven Feng CAI

What are your thoughts?

More from PyQuantLab



PyQuantLab

Volatility Clustering Trading Strategy with Python

Ultimate Algorithmic Strategy Bundle has you covered with over 80 Python...

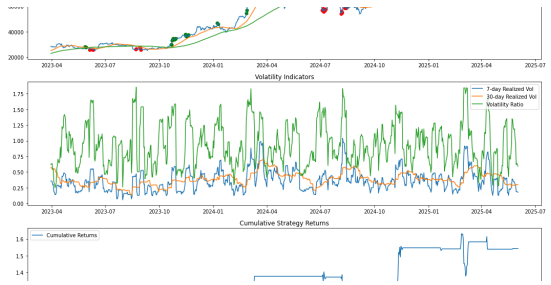
Jun 3 32 3

PyQuantLab

An Algorithmic Exploration of Volume Spread Analysis...

Note: You can read all articles for free on our website: pyquantlab.com

Jun 9 54 1

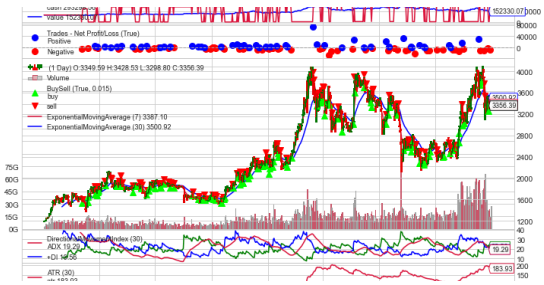


PyQuantLab

Trend-Volatility Confluence Trading Strategy

The Ultimate Algorithmic Strategy Bundle has you covered with over 80...

Jun 3 60



PyQuantLab

Building an Adaptive Trading Strategy with Backtrader: A...

Note: You can read all articles for free on our website: pyquantlab.com

Jun 4 64

See all from PyQuantLab

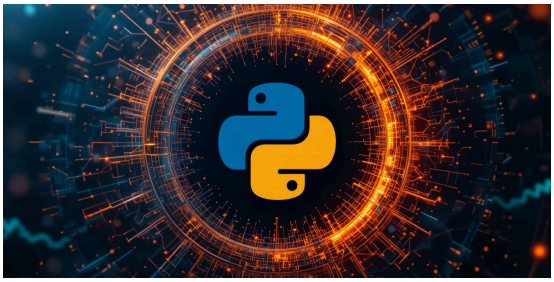
Recommended from Medium

Swapnilphutane

How I Built a Multi-Market Trading Strategy That Pass...

When I first got into trading, I had no plans of building a full-blown system....

6d ago 16 1

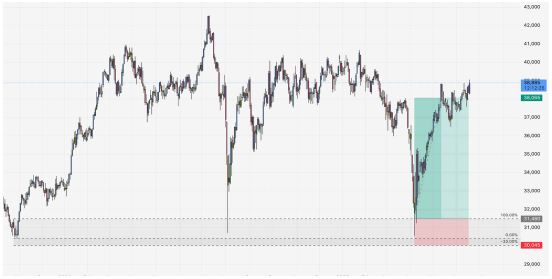


Unicorn Day

The Quest for the Perfect Trading Score: Turning...

Navigating the financial markets... it feels like being hit by a tsunami of da...

3d ago 43



MarketMuse

“I Let an AI Bot Trade for Me for 7 Days—It Made $8,000...

Subtitle: While you’re analyzing candlestick patterns, AI bots are fron...

Jun 3 75 3

Candence

Exposing Bernd Skorupinski Strategy: How I Profited ove...

I executed a single Nikkei Futures trade that banked $16,400 with a...

Jun 19 2

FMZQuant

Multi-Timeframe Dynamic Trend Detection System: EM...

Strategy Overview

Jun 4 1

Remedy

How I Turned My Trading Strategy into a Professional...

By Chinedu Uzochukwu

6d ago 1 1

See more recommendations