04. Baseline Trading Strategies & Cross-Market Analysis

MANAV AGARWAL

This notebook implements and evaluates 15+ baseline trading strategies across cryptocurrency and equity markets, establishing performance benchmarks for machine learning models. Combining:

1. *Risk Management*: Stop-loss, position sizing, and drawdown controls
2. *Cross-Market Analysis*: Crypto (24/7) vs Equity (regular hours) comparison
3. *Statistical Testing*: Significance tests for strategy performance
4. *ML Baselines*: Simple ML models as additional benchmarks

Framework for Trading

Strategy Return Calculation

For any trading strategy, the return at time t is:

$$R_t = \text{pos}_{t-1} \times r_t - c \times |\text{pos}_t - \text{pos}_{t-1}|$$

Where:

- $\text{pos}_t \in \{-1, 0, 1\}$ is the position (short, neutral, long)
- $r_t = \frac{P_t - P_{t-1}}{P_{t-1}}$ is the asset return
- c is the transaction cost

Performance Metrics

Sharpe Ratio

$$\text{SR} = \frac{E[R_p - R_f]}{\sigma_p} = \frac{\mu_p - r_f}{\sigma_p}$$

Calmar Ratio

$$\text{CR} = \frac{\text{CAGR}}{|\text{MaxDD}|}$$

Information Ratio

$$\text{IR} = \frac{E[R_p - R_b]}{\sigma_{p-b}}$$

Maximum Drawdown

$$\text{MDD} = \max_{t \in [0,T]} \left(\max_{s \in [0,t]} \text{NAV}_s - \text{NAV}_t \right) / \max_{s \in [0,t]} \text{NAV}_s$$

Risk Management

Kelly Criterion

$$f^* = \frac{p \times b - q}{b} = \frac{\text{edge}}{\text{odds}}$$

Where:

- p = probability of winning
- $q = 1 - p$ = probability of losing
- b = odds (win/loss ratio)

Value at Risk (VaR)

$$\text{VaR}_\alpha = -\inf\{x : P(R \leq x) > \alpha\}$$

Conditional Value at Risk (CVaR)

$$\text{CVaR}_\alpha = -E[R | R \leq -\text{VaR}_\alpha]$$

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from typing import Dict, List, Tuple, Optional, Union
import warnings
warnings.filterwarnings('ignore')
from scipy import stats
from scipy.stats import jarque_bera, shapiro, normaltest
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.model_selection import TimeSeriesSplit
import pickle
from pathlib import Path
from datetime import datetime, timedelta
import sys
import gc
import os
from multiprocessing import Pool
from functools import partial
```

In [2]:
```
# Set working directory
os.chdir('C:/Users/manav')
sys.path.append('src')
```

In [3]:
```
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette("husl")
```

In [4]:
```python
CONFIG = {
    'initial_capital': 100000,
    'transaction_cost': 0.001,
    'slippage': 0.0005,
    'risk_free_rate': 0.04,
    'max_position_size': 1.0,
    'stop_loss': 0.05,
    'take_profit': 0.15,
    'max_drawdown_limit': 0.20,
    'confidence_level': 0.95,
    'walk_forward_window': 252,
    'walk_forward_step': 63,
    'target_rows': 10000,
    'enable_parallel': False,  # may crash
    'n_processes': 4,
    'enable_validation': True,
    'enable_walk_forward': True,
    'enable_regime_testing': True,
    'enable_ml_baselines': True
}
```

In [5]:
```python
ASSET_COLORS = {
    'BTC-USD': '#FF6B35', 'BTCUSD': '#FF6B35',
    'ETH-USD': '#627EEA', 'ETHUSD': '#627EEA',
    'SOL-USD': '#00FFA3', 'SOLUSD': '#00FFA3',
    'ADA-USD': '#0033AD', 'ADAUSD': '#0033AD',
    'XRP-USD': '#23292F', 'XRPUSD': '#23292F',
    'SPY': '#003f5c',
    'QQQ': '#2f4b7c',
    'DIA': '#665191',
    'IWM': '#a05195',
    'VTI': '#d45087'
}
```

Testing Suite

In [6]:
```python
def validate_signals(signals):
    signals = np.array(signals)
    signals = np.where(np.isnan(signals), 0, signals)
    signals = np.clip(signals, -1, 1)  # Bound to [-1, 1]
    signals = np.where(np.abs(signals) < 0.01, 0, signals)  # Filter tiny signals
    return signals
```

In [7]:
```python
def validate_returns(returns):
    returns = np.array(returns)
    returns = np.where(np.isnan(returns), 0, returns)
    returns = np.clip(returns, -0.2, 0.2)  # Cap at ±20% daily moves
    return returns
```

In [8]:
```python
def safe_backtest_core(signals, returns, transaction_cost, initial_capital, max_pos
    n = len(signals)
    signals = validate_signals(signals)
    returns = validate_returns(returns)
```

```
    # Initialize tracking arrays
    equity_curve = np.zeros(n)
    positions = np.zeros(n)
    cash = np.zeros(n)
    trades = []

    # Starting values
    capital = float(initial_capital)
    current_position = 0.0
    equity_curve[0] = capital
    cash[0] = capital

    for i in range(1, n):
        # Target position (bounded)
        target_position = np.clip(float(signals[i]), -max_position, max_position)

        # Position change
        position_change = target_position - current_position

        # Transaction costs and position updates
        if abs(position_change) > 0.01:  # Minimum change threshold
            cost_rate = abs(position_change) * transaction_cost
            cost_rate = min(cost_rate, 0.05)  # Max 5% cost per trade
            capital *= (1 - cost_rate)

            # Record trade
            if len(trades) == 0 or abs(position_change) > 0.1:
                trades.append({
                    'date': i,
                    'position_change': position_change,
                    'cost': cost_rate * capital
                })
            current_position = target_position

        if abs(current_position) > 0.01:
            daily_return = current_position * returns[i]
            daily_return = np.clip(daily_return, -0.1, 0.1)  # Cap daily moves
            capital *= (1 + daily_return)

        # Store values
        positions[i] = current_position
        equity_curve[i] = max(capital, 0)  # Can't go negative
        cash[i] = capital * (1 - abs(current_position))  # Approximate cash

        if capital < initial_capital * 0.01:  # Lost 99%
            positions[i:] = 0
            equity_curve[i:] = capital
            cash[i:] = capital
            break

    return equity_curve, positions, cash, trades
```

In [9]:

```
class ComprehensiveBacktestEngine:
    def __init__(self, config):
        self.config = config
```

```python
    def calculate_comprehensive_metrics(self, equity_curve, positions, cash, trades
        if len(equity_curve) < 2:
            return self._empty_metrics()

        equity_curve = np.array(equity_curve)
        equity_curve = equity_curve[equity_curve > 0]

        if len(equity_curve) < 2:
            return self._empty_metrics()

        total_return = (equity_curve[-1] - equity_curve[0]) / equity_curve[0]
        total_return = np.clip(total_return, -0.95, 10.0)  # Safety bounds

        # Daily returns
        daily_returns = np.diff(equity_curve) / equity_curve[:-1]
        daily_returns = daily_returns[np.isfinite(daily_returns)]

        if len(daily_returns) == 0:
            return self._empty_metrics()

        # Basic statistics
        mean_return = np.mean(daily_returns)
        std_return = np.std(daily_returns)
        median_return = np.median(daily_returns)

        if std_return > 0:
            sharpe_ratio = (mean_return / std_return) * np.sqrt(252)
            sharpe_ratio = np.clip(sharpe_ratio, -5, 5)
        else:
            sharpe_ratio = 0

        # Drawdown analysis
        peak = np.maximum.accumulate(equity_curve)
        drawdown = (peak - equity_curve) / peak
        max_drawdown = min(np.max(drawdown), 0.99)

        avg_drawdown = np.mean(drawdown)

        # Drawdown duration
        in_drawdown = drawdown > 0.01
        if np.any(in_drawdown):
            drawdown_periods = []
            current_period = 0
            for dd in in_drawdown:
                if dd:
                    current_period += 1
                else:
                    if current_period > 0:
                        drawdown_periods.append(current_period)
                        current_period = 0
            if current_period > 0:
                drawdown_periods.append(current_period)

            avg_drawdown_duration = np.mean(drawdown_periods) if drawdown_periods e
            max_drawdown_duration = np.max(drawdown_periods) if drawdown_periods el
        else:
```

```
            avg_drawdown_duration = 0
            max_drawdown_duration = 0

        n_years = len(equity_curve) / 252
        if n_years > 0.1:
            annual_return = (1 + total_return) ** (1/n_years) - 1
            annual_return = np.clip(annual_return, -0.8, 3.0)
            cagr = annual_return
        else:
            annual_return = total_return
            cagr = total_return

        # Risk ratios
        calmar_ratio = annual_return / max_drawdown if max_drawdown > 0.001 else 0
        calmar_ratio = np.clip(calmar_ratio, -10, 10)

        # Sortino ratio
        downside_returns = daily_returns[daily_returns < 0]
        downside_std = np.std(downside_returns) if len(downside_returns) > 0 else s
        sortino_ratio = (mean_return / downside_std) * np.sqrt(252) if downside_std
        sortino_ratio = np.clip(sortino_ratio, -5, 5)

        # Win/Loss analysis
        wins = daily_returns[daily_returns > 0]
        losses = daily_returns[daily_returns < 0]

        win_rate = len(wins) / len(daily_returns) if len(daily_returns) > 0 else 0.
        loss_rate = len(losses) / len(daily_returns) if len(daily_returns) > 0 else

        avg_win = np.mean(wins) if len(wins) > 0 else 0
        avg_loss = np.mean(np.abs(losses)) if len(losses) > 0 else 0.01

        largest_win = np.max(wins) if len(wins) > 0 else 0
        largest_loss = np.min(losses) if len(losses) > 0 else 0

        profit_factor = avg_win / avg_loss if avg_loss > 0 else 1
        profit_factor = np.clip(profit_factor, 0.1, 10)

        # Expectancy
        expectancy = (win_rate * avg_win) - (loss_rate * avg_loss)

        # Risk metrics
        var_95 = np.percentile(daily_returns, 5) if len(daily_returns) > 0 else 0
        var_99 = np.percentile(daily_returns, 1) if len(daily_returns) > 0 else 0

        cvar_95 = np.mean(daily_returns[daily_returns <= var_95]) if len(daily_retu
        cvar_99 = np.mean(daily_returns[daily_returns <= var_99]) if len(daily_retu

        # Higher moments
        skewness = stats.skew(daily_returns) if len(daily_returns) > 3 else 0
        excess_kurtosis = stats.kurtosis(daily_returns) if len(daily_returns) > 3 e

        n_trades = len(trades)
        n_winning_trades = len([t for t in trades if t.get('pnl', 0) > 0]) if trade
        n_losing_trades = len([t for t in trades if t.get('pnl', 0) < 0]) if trades
```

```
        # Trading frequency
        trades_per_year = n_trades / n_years if n_years > 0 else 0

        # Information ratio (vs risk-free rate)
        excess_return = mean_return - (self.config['risk_free_rate'] / 252)
        information_ratio = (excess_return / std_return) * np.sqrt(252) if std_retu
        information_ratio = np.clip(information_ratio, -5, 5)

        # Stability metrics
        monthly_returns = []
        if len(equity_curve) > 21:  # At least ~1 month
            for i in range(21, len(equity_curve), 21):  # Every ~month
                monthly_ret = (equity_curve[i] - equity_curve[i-21]) / equity_curve
                monthly_returns.append(monthly_ret)

        if monthly_returns:
            monthly_std = np.std(monthly_returns)
            consistency = 1 - monthly_std if monthly_std < 1 else 0
        else:
            monthly_std = std_return * np.sqrt(21)
            consistency = 0.5
        ulcer_index = np.sqrt(np.mean(drawdown ** 2)) if len(drawdown) > 0 else 0
        # Martin Ratio
        martin_ratio = annual_return / ulcer_index if ulcer_index > 0 else 0
        martin_ratio = np.clip(martin_ratio, -10, 10)

        # Recovery factor
        recovery_factor = total_return / max_drawdown if max_drawdown > 0 else 0

        # Gain-to-Pain ratio
        positive_returns = daily_returns[daily_returns > 0]
        negative_returns = daily_returns[daily_returns < 0]

        gain_to_pain = (np.sum(positive_returns) / abs(np.sum(negative_returns))) i

        # Sterling ratio
        sterling_ratio = annual_return / avg_drawdown if avg_drawdown > 0 else 0
        sterling_ratio = np.clip(sterling_ratio, -10, 10)

        # Burke ratio
        burke_ratio = annual_return / np.sqrt(np.sum(drawdown ** 2)) if np.sum(draw
        burke_ratio = np.clip(burke_ratio, -10, 10)

        # Tail ratio
        tail_ratio = abs(np.percentile(daily_returns, 95) / np.percentile(daily_ret
        # Hit ratio (% profitable periods)
        hit_ratio = len(positive_returns) / len(daily_returns) if len(daily_returns

        # Return all comprehensive metrics
        return {
            # Basic Returns
            'total_return': float(total_return),
            'annual_return': float(annual_return),
            'cagr': float(cagr),
            'mean_return': float(mean_return),
            'median_return': float(median_return),
```

```
            'std_return': float(std_return),
            'volatility': float(std_return * np.sqrt(252)),

            # Risk-Adjusted Returns
            'sharpe_ratio': float(sharpe_ratio),
            'sortino_ratio': float(sortino_ratio),
            'calmar_ratio': float(calmar_ratio),
            'information_ratio': float(information_ratio),
            'martin_ratio': float(martin_ratio),
            'sterling_ratio': float(sterling_ratio),
            'burke_ratio': float(burke_ratio),

            # Drawdown Metrics
            'max_drawdown': float(max_drawdown),
            'avg_drawdown': float(avg_drawdown),
            'max_drawdown_duration': float(max_drawdown_duration),
            'avg_drawdown_duration': float(avg_drawdown_duration),
            'ulcer_index': float(ulcer_index),
            'recovery_factor': float(recovery_factor),

            # Win/Loss Metrics
            'win_rate': float(win_rate),
            'loss_rate': float(loss_rate),
            'hit_ratio': float(hit_ratio),
            'profit_factor': float(profit_factor),
            'expectancy': float(expectancy),
            'avg_win': float(avg_win),
            'avg_loss': float(avg_loss),
            'largest_win': float(largest_win),
            'largest_loss': float(largest_loss),
            'gain_to_pain': float(gain_to_pain),

            # Risk Metrics
            'var_95': float(var_95),
            'var_99': float(var_99),
            'cvar_95': float(cvar_95),
            'cvar_99': float(cvar_99),
            'skewness': float(skewness),
            'excess_kurtosis': float(excess_kurtosis),
            'tail_ratio': float(tail_ratio),

            # Trading Metrics
            'n_trades': int(n_trades),
            'n_winning_trades': int(n_winning_trades),
            'n_losing_trades': int(n_losing_trades),
            'trades_per_year': float(trades_per_year),

            # Stability Metrics
            'consistency': float(consistency),
            'monthly_std': float(monthly_std)
        }
    def _empty_metrics(self):
        return {
            'total_return': 0.0, 'annual_return': 0.0, 'cagr': 0.0,
            'mean_return': 0.0, 'median_return': 0.0, 'std_return': 0.0,
            'volatility': 0.0, 'sharpe_ratio': 0.0, 'sortino_ratio': 0.0,
```

```
            'calmar_ratio': 0.0, 'information_ratio': 0.0, 'martin_ratio': 0.0,
            'sterling_ratio': 0.0, 'burke_ratio': 0.0, 'max_drawdown': 0.0,
            'avg_drawdown': 0.0, 'max_drawdown_duration': 0.0, 'avg_drawdown_durati
            'ulcer_index': 0.0, 'recovery_factor': 0.0, 'win_rate': 0.5,
            'loss_rate': 0.5, 'hit_ratio': 0.5, 'profit_factor': 1.0,
            'expectancy': 0.0, 'avg_win': 0.0, 'avg_loss': 0.0,
            'largest_win': 0.0, 'largest_loss': 0.0, 'gain_to_pain': 1.0,
            'var_95': 0.0, 'var_99': 0.0, 'cvar_95': 0.0, 'cvar_99': 0.0,
            'skewness': 0.0, 'excess_kurtosis': 0.0, 'tail_ratio': 1.0,
            'n_trades': 0, 'n_winning_trades': 0, 'n_losing_trades': 0,
            'trades_per_year': 0.0, 'consistency': 0.0, 'monthly_std': 0.0
        }

    def run_comprehensive_backtest(self, strategy, data, symbol):
        try:
            signals = strategy.generate_signals(data)
            if signals is None or len(signals) == 0:
                raise ValueError("No signals generated")

            returns = data['close'].pct_change().fillna(0)

            min_len = min(len(signals), len(returns))
            signals = signals.iloc[:min_len]
            returns = returns.iloc[:min_len]

            equity_curve, positions, cash, trades = safe_backtest_core(
                signals.values, returns.values,
                self.config['transaction_cost'], self.config['initial_capital'],
                self.config['max_position_size']
            )

            metrics = self.calculate_comprehensive_metrics(
                equity_curve, positions, cash, trades, returns.values
            )

            # Validation check
            if abs(metrics['total_return']) > 20:  # 2000% return is suspicious
                print(f"    Warning: High return {metrics['total_return']:.1%} for

            return {
                'strategy': strategy.name,
                'symbol': symbol,
                'metrics': metrics,
                'success': True,
                'equity_curve': equity_curve,
                'positions': positions,
                'cash': cash,
                'trades': trades
            }

        except Exception as e:
            print(f"    Failed: {strategy.name} on {symbol} - {e}")
            return {
                'strategy': strategy.name,
                'symbol': symbol,
                'metrics': self._empty_metrics(),
```

```
                'success': False,
                'error': str(e)
            }
```

In [10]:
```
class BuyAndHold:
    def __init__(self, config=None):
        self.name = 'BuyAndHold'
        self.config = config or {}

    def generate_signals(self, data):
        return pd.Series(index=data.index, data=1.0)
```

In [11]:
```
class MovingAverageCrossover:
    def __init__(self, fast=20, slow=50, ma_type='SMA', config=None):
        self.name = f"{ma_type}_{fast}_{slow}"
        self.fast = fast
        self.slow = slow
        self.ma_type = ma_type
        self.config = config or {}

    def generate_signals(self, data):
        close = data['close']

        if self.ma_type == 'SMA':
            fast_ma = close.rolling(self.fast, min_periods=1).mean()
            slow_ma = close.rolling(self.slow, min_periods=1).mean()
        else:  # EMA
            fast_ma = close.ewm(span=self.fast).mean()
            slow_ma = close.ewm(span=self.slow).mean()

        signals = pd.Series(index=data.index, data=0.0)
        signals[fast_ma > slow_ma * 1.001] = 1.0   # Small buffer
        signals[fast_ma < slow_ma * 0.999] = -1.0

        return signals.rolling(3, center=True).mean().fillna(method='ffill').fillna
```

In [12]:
```
class RSIStrategy:
    def __init__(self, period=14, oversold=30, overbought=70, config=None):
        self.name = f"RSI_{period}"
        self.period = period
        self.oversold = oversold
        self.overbought = overbought
        self.config = config or {}

    def generate_signals(self, data):
        close = data['close']

        # RSI calculation
        delta = close.diff()
        gain = delta.where(delta > 0, 0).rolling(self.period).mean()
        loss = (-delta.where(delta < 0, 0)).rolling(self.period).mean()

        rs = gain / (loss + 1e-10)
        rsi = 100 - (100 / (1 + rs))
```

```
        # Generate signals
        signals = pd.Series(index=data.index, data=0.0)
        signals[rsi < self.oversold] = 1.0    # Buy when oversold
        signals[rsi > self.overbought] = -1.0  # Sell when overbought

        return signals
```

In [13]:
```
class BollingerBands:
    def __init__(self, period=20, std_dev=2.0, config=None):
        self.name = f"BB_{period}_{std_dev}"
        self.period = period
        self.std_dev = std_dev
        self.config = config or {}

    def generate_signals(self, data):
        close = data['close']

        sma = close.rolling(self.period, min_periods=1).mean()
        std = close.rolling(self.period, min_periods=1).std()

        upper = sma + (std * self.std_dev)
        lower = sma - (std * self.std_dev)

        # Mean reversion signals
        signals = pd.Series(index=data.index, data=0.0)
        signals[close <= lower] = 1.0   # Buy at lower band
        signals[close >= upper] = -1.0  # Sell at upper band

        return signals
```

In [14]:
```
class MACDStrategy:
    def __init__(self, fast=12, slow=26, signal=9, config=None):
        self.name = f"MACD_{fast}_{slow}_{signal}"
        self.fast = fast
        self.slow = slow
        self.signal = signal
        self.config = config or {}

    def generate_signals(self, data):
        close = data['close']

        # MACD calculation
        ema_fast = close.ewm(span=self.fast).mean()
        ema_slow = close.ewm(span=self.slow).mean()

        macd_line = ema_fast - ema_slow
        signal_line = macd_line.ewm(span=self.signal).mean()

        # Generate signals
        signals = pd.Series(index=data.index, data=0.0)
        signals[macd_line > signal_line] = 1.0
        signals[macd_line <= signal_line] = -1.0

        return signals
```

In [15]:
```python
class MomentumStrategy:
    def __init__(self, lookback=20, config=None):
        self.name = f"Momentum_{lookback}"
        self.lookback = lookback
        self.config = config or {}

    def generate_signals(self, data):
        momentum = data['close'].pct_change(self.lookback)

        signals = pd.Series(index=data.index, data=0.0)
        signals[momentum > 0.02] = 1.0   # Buy on positive momentum
        signals[momentum < -0.02] = -1.0  # Sell on negative momentum

        return signals
```

In [16]:
```python
class MeanReversionStrategy:
    def __init__(self, lookback=20, threshold=2.0, config=None):
        self.name = f"MeanRev_{lookback}_{threshold}"
        self.lookback = lookback
        self.threshold = threshold
        self.config = config or {}

    def generate_signals(self, data):
        close = data['close']

        # Z-score calculation
        rolling_mean = close.rolling(self.lookback).mean()
        rolling_std = close.rolling(self.lookback).std()
        z_score = (close - rolling_mean) / (rolling_std + 1e-10)

        # Mean reversion signals
        signals = pd.Series(index=data.index, data=0.0)
        signals[z_score < -self.threshold] = 1.0   # Buy when below mean
        signals[z_score > self.threshold] = -1.0   # Sell when above mean

        return signals
```

In [17]:
```python
class BreakoutStrategy:
    def __init__(self, lookback=20, config=None):
        self.name = f"Breakout_{lookback}"
        self.lookback = lookback
        self.config = config or {}

    def generate_signals(self, data):
        close = data['close']

        # Calculate rolling highs and lows
        rolling_high = close.rolling(self.lookback).max()
        rolling_low = close.rolling(self.lookback).min()

        # Breakout signals
        signals = pd.Series(index=data.index, data=0.0)
        signals[close > rolling_high.shift(1)] = 1.0   # Buy on breakout above
        signals[close < rolling_low.shift(1)] = -1.0   # Sell on breakdown below
```

```
        return signals
```

In [18]:
```
class ChannelStrategy:
    def __init__(self, period=20, config=None):
        self.name = f"Channel_{period}"
        self.period = period
        self.config = config or {}

    def generate_signals(self, data):
        high = data['high']
        low = data['low']
        close = data['close']

        # Channel calculation
        upper_channel = high.rolling(self.period).max()
        lower_channel = low.rolling(self.period).min()
        mid_channel = (upper_channel + lower_channel) / 2

        # Channel signals
        signals = pd.Series(index=data.index, data=0.0)
        signals[close > mid_channel] = 1.0   # Buy above mid-channel
        signals[close < mid_channel] = -1.0  # Sell below mid-channel

        return signals
```

In [19]:
```
class SimpleMLStrategy:
    def __init__(self, model_type='RandomForest', lookback=20, config=None):
        self.name = f"ML_{model_type}_{lookback}"
        self.model_type = model_type
        self.lookback = lookback
        self.config = config or {}
        self.model = None
        self.scaler = StandardScaler()

    def generate_signals(self, data):
        if len(data) < self.lookback + 50:  # Need enough data
            return pd.Series(index=data.index, data=0.0)

        features = self._create_features(data)

        # Create target (next day direction)
        target = (data['close'].shift(-1) > data['close']).astype(int)

        valid_idx = features.dropna().index.intersection(target.dropna().index)
        if len(valid_idx) < 50:
            return pd.Series(index=data.index, data=0.0)

        X = features.loc[valid_idx]
        y = target.loc[valid_idx]

        split_idx = int(len(X) * 0.8)
        X_train, X_test = X.iloc[:split_idx], X.iloc[split_idx:]
        y_train, y_test = y.iloc[:split_idx], y.iloc[split_idx:]
```

```
        if len(X_train) < 20 or len(X_test) < 10:
            return pd.Series(index=data.index, data=0.0)

        try:
            # Scale features
            X_train_scaled = self.scaler.fit_transform(X_train)
            X_test_scaled = self.scaler.transform(X_test)

            # Train model
            if self.model_type == 'RandomForest':
                self.model = RandomForestClassifier(n_estimators=50, max_depth=5, r
            else:
                self.model = LogisticRegression(random_state=42, max_iter=1000)

            self.model.fit(X_train_scaled, y_train)

            # Generate predictions
            predictions = self.model.predict(X_test_scaled)

            # Create signals
            signals = pd.Series(index=data.index, data=0.0)
            signals.loc[X_test.index] = np.where(predictions == 1, 1.0, -1.0)

            return signals

        except Exception as e:
            print(f"ML model failed: {e}")
            return pd.Series(index=data.index, data=0.0)

    def _create_features(self, data):
        features = pd.DataFrame(index=data.index)

        close = data['close']

        for period in [5, 10, 20]:
            features[f'sma_{period}'] = close.rolling(period).mean()
            features[f'price_to_sma_{period}'] = close / features[f'sma_{period}']

        for period in [5, 10, 20]:
            features[f'momentum_{period}'] = close.pct_change(period)

        for period in [5, 10, 20]:
            features[f'volatility_{period}'] = close.pct_change().rolling(period).s

        # Technical indicators
        # RSI
        delta = close.diff()
        gain = delta.where(delta > 0, 0).rolling(14).mean()
        loss = (-delta.where(delta < 0, 0)).rolling(14).mean()
        rs = gain / (loss + 1e-10)
        features['rsi'] = 100 - (100 / (1 + rs))

        # Bollinger Bands position
        sma_20 = close.rolling(20).mean()
        std_20 = close.rolling(20).std()
        features['bb_position'] = (close - sma_20) / (std_20 + 1e-10)
```

```
        return features.dropna()
```

In [20]:

```
def load_data_safely():
    possible_files = [
        'notebooks/01_comprehensive_results.pkl',
        '01_comprehensive_results.pkl'
    ]

    for filepath in possible_files:
        try:
            if os.path.exists(filepath):
                print(f"  Trying: {filepath}")
                with open(filepath, 'rb') as f:
                    results = pickle.load(f)

                for key in ['all_data', 'all_assets', 'market_data']:
                    if key in results:
                        data_source = results[key]
                        print(f"  Found data under key: {key}")

                        clean_data = {}
                        for symbol, df in data_source.items():
                            if df is not None and len(df) > 100:
                                # Handle column issues
                                if isinstance(df.columns, pd.MultiIndex):
                                    df.columns = [col[0] for col in df.columns]

                                df.columns = [col.lower() for col in df.columns]

                                if 'close' in df.columns:
                                    # Ensure we have OHLCV
                                    required = ['open', 'high', 'low', 'close', 'vo
                                    missing = [col for col in required if col not i

                                    # Fill missing columns if possible
                                    if 'open' not in df.columns:
                                        df['open'] = df['close'].shift(1).fillna(df
                                    if 'high' not in df.columns:
                                        df['high'] = df['close'] * 1.01
                                    if 'low' not in df.columns:
                                        df['low'] = df['close'] * 0.99
                                    if 'volume' not in df.columns:
                                        df['volume'] = 1000000

                                    # Downsample if too large
                                    if len(df) > CONFIG['target_rows']:
                                        step = len(df) // CONFIG['target_rows']
                                        df = df.iloc[::step]

                                    clean_data[symbol] = df
                                    print(f"    {symbol}: {len(df)} records")

                        if clean_data:
                            print(f"  Successfully loaded {len(clean_data)} symbols
                            return clean_data
```

```
        except Exception as e:
            print(f"  Error with {filepath}: {e}")
```

In [21]:
```
def create_walk_forward_splits(data, config):
    if not config.get('enable_walk_forward', False):
        return None

    splits = []
    train_days = config['walk_forward_window']
    step_days = config['walk_forward_step']

    start_idx = train_days

    while start_idx + step_days < len(data):
        train_data = data.iloc[start_idx-train_days:start_idx]
        test_data = data.iloc[start_idx:start_idx+step_days]

        if len(train_data) >= 100 and len(test_data) >= 20:
            splits.append({
                'train_data': train_data,
                'test_data': test_data,
                'train_start': train_data.index[0],
                'train_end': train_data.index[-1],
                'test_start': test_data.index[0],
                'test_end': test_data.index[-1]
            })

        start_idx += step_days

    return splits
```

In [22]:
```
def diebold_mariano_test(errors1, errors2, h=1):
    d = errors1 - errors2
    mean_d = np.mean(d)
    def autocovariance(xi, k):
        return np.mean((xi[:-k] - mean_d) * (xi[k:] - mean_d))

    gamma = [autocovariance(d, i) for i in range(h)]
    v_d = gamma[0] + 2 * sum(gamma[1:])

    dm_stat = mean_d / np.sqrt(v_d / len(d))

    p_value = 2 * (1 - stats.norm.cdf(abs(dm_stat)))

    return dm_stat, p_value
```

In [23]:
```
def perform_comprehensive_statistical_tests(results_df):
    if results_df.empty:
        print("No results available for statistical testing")
        return {}
    test_results = {}
    # 2. ANOVA across all strategies
    print("\n2. ANOVA Across All Strategies")
    print("-" * 50)
```

```
    strategy_groups = []
    strategy_names = []

    for name, group in results_df.groupby('strategy_name'):
        if len(group) > 0:
            strategy_groups.append(group['total_return'].values)
            strategy_names.append(name)

    if len(strategy_groups) > 2:
        try:
            f_stat, p_value = stats.f_oneway(*strategy_groups)

            h_stat, h_p_value = stats.kruskal(*strategy_groups)

            print(f"ANOVA F-statistic: {f_stat:.3f}, p-value: {p_value:.4f}")
            print(f"Kruskal-Wallis H-statistic: {h_stat:.3f}, p-value: {h_p_value:.
            print(f"Result: {'Significant differences exist' if p_value < 0.05 else

            test_results['anova'] = {
                'f_stat': f_stat,
                'p_value': p_value,
                'h_stat': h_stat,
                'h_p_value': h_p_value,
                'significant': p_value < 0.05
            }
        except Exception as e:
            print(f"ANOVA failed: {e}")

    # 3. Market type comparison with multiple metrics
    print("\n3. Comprehensive Market Type Comparison")
    print("-" * 50)

    if 'asset_type' in results_df.columns:
        crypto_results = results_df[results_df['asset_type'] == 'Crypto']
        equity_results = results_df[results_df['asset_type'] == 'Equity']
        if len(crypto_results) > 0 and len(equity_results) > 0:
            metrics_to_compare = ['total_return', 'sharpe_ratio', 'max_drawdown', '
            market_comparisons = {}
            for metric in metrics_to_compare:
                crypto_values = crypto_results[metric].values
                equity_values = equity_results[metric].values

                if len(crypto_values) > 1 and len(equity_values) > 1:
                    try:
                        t_stat, p_value = stats.ttest_ind(crypto_values, equity_val
                        u_stat, u_p_value = stats.mannwhitneyu(crypto_values, equit
                        crypto_mean = np.mean(crypto_values)
                        equity_mean = np.mean(equity_values)
                        market_comparisons[metric] = {
                            'crypto_mean': crypto_mean,
                            'equity_mean': equity_mean,
                            't_stat': t_stat,
                            'p_value': p_value,
                            'u_stat': u_stat,
                            'u_p_value': u_p_value,
                            'difference': crypto_mean - equity_mean,
```

```
                            'significant': p_value < 0.05
                        }

                        result = "Crypto better" if crypto_mean > equity_mean and p
                                "Equity better" if crypto_mean < equity_mean and p_
                                "No difference"

                        print(f"{metric:15s}: Crypto={crypto_mean:8.3f}, Equity={eq
                              f"p={p_value:.4f} ({result})")

                    except Exception as e:
                        print(f"{metric:15s}: Error - {e}")

            test_results['market_comparison'] = market_comparisons

    # 4. Performance distribution tests
    print("\n4. Distribution Analysis")
    print("-" * 50)

    all_returns = results_df['total_return'].values

    # Normality tests
    shapiro_stat, shapiro_p = stats.shapiro(all_returns)
    jb_stat, jb_p = jarque_bera(all_returns)

    print(f"Shapiro-Wilk test: W={shapiro_stat:.4f}, p={shapiro_p:.4f}")
    print(f"Jarque-Bera test: JB={jb_stat:.4f}, p={jb_p:.4f}")
    print(f"Returns are {'NOT ' if shapiro_p < 0.05 else ''}normally distributed (α

    test_results['distribution'] = {
        'shapiro_stat': shapiro_stat,
        'shapiro_p': shapiro_p,
        'jb_stat': jb_stat,
        'jb_p': jb_p,
        'normal': shapiro_p >= 0.05
    }

    return test_results
```

In [24]:

```
def run_complete_analysis():
    market_data = load_data_safely()

    if not market_data:
        print("No data available!")
        return

    # Create ALL strategies
    strategies = [
        BuyAndHold(CONFIG),
        MovingAverageCrossover(20, 50, 'SMA', CONFIG),
        MovingAverageCrossover(12, 26, 'EMA', CONFIG),
        MovingAverageCrossover(5, 20, 'SMA', CONFIG),
        MovingAverageCrossover(10, 30, 'EMA', CONFIG),
        RSIStrategy(14, 30, 70, CONFIG),
        RSIStrategy(21, 25, 75, CONFIG),
        BollingerBands(20, 2.0, CONFIG),
```

```
        BollingerBands(10, 1.5, CONFIG),
        BollingerBands(20, 1.5, CONFIG),
        MACDStrategy(12, 26, 9, CONFIG),
        MACDStrategy(8, 21, 5, CONFIG),
        MomentumStrategy(20, CONFIG),
        MomentumStrategy(10, CONFIG),
        MomentumStrategy(5, CONFIG),
        MeanReversionStrategy(20, 2.0, CONFIG),
        MeanReversionStrategy(10, 1.5, CONFIG),
        BreakoutStrategy(20, CONFIG),
        BreakoutStrategy(10, CONFIG),
        ChannelStrategy(20, CONFIG)
    ]

    if CONFIG.get('enable_ml_baselines', True):
        strategies.extend([
            SimpleMLStrategy('RandomForest', 20, CONFIG),
            SimpleMLStrategy('LogisticRegression', 20, CONFIG)
        ])

    # Run comprehensive backtests
    engine = ComprehensiveBacktestEngine(CONFIG)
    all_results = []

    print(f"\nRunning comprehensive backtests...")
    print(f"Strategies: {len(strategies)}")
    print(f"Assets: {len(market_data)}")
    print(f"Total combinations: {len(strategies) * len(market_data)}")
    print("=" * 80)

    for symbol, data in market_data.items():
        print(f"\n{symbol} ({len(data)} records):")

        asset_type = 'Crypto' if any(c in symbol.upper() for c in ['BTC', 'ETH', 'S

        for strategy in strategies:
            result = engine.run_comprehensive_backtest(strategy, data, symbol)

            if result['success']:
                result['asset_type'] = asset_type
                result['strategy_name'] = strategy.name
                all_results.append(result)

                m = result['metrics']
                print(f"  {strategy.name:25s}: Return={m['total_return']:7.2%} | "
                      f"Sharpe={m['sharpe_ratio']:5.2f} | Calmar={m['calmar_ratio']
                      f"MaxDD={m['max_drawdown']:6.2%} | Trades={m['n_trades']:3.0f
            else:
                print(f"  {strategy.name:25s}: FAILED")

    if all_results:
        results_data = []
        for r in all_results:
            row = {
                'strategy_name': r['strategy_name'],
                'symbol': r['symbol'],
```

```
            'asset_type': r['asset_type'],
            **r['metrics']
        }
        results_data.append(row)

results_df = pd.DataFrame(results_data)
print(f"Total successful backtests: {len(results_df)}")
print(f"Strategies tested: {results_df['strategy_name'].nunique()}")
print(f"Assets analyzed: {results_df['symbol'].nunique()}")

# Key performance metrics
print(f"\nKEY PERFORMANCE METRICS:")
print(f"  Average Total Return: {results_df['total_return'].mean():.2%}")
print(f"  Average Annual Return: {results_df['annual_return'].mean():.2%}")
print(f"  Average Sharpe Ratio: {results_df['sharpe_ratio'].mean():.2f}")
print(f"  Average Calmar Ratio: {results_df['calmar_ratio'].mean():.2f}")
print(f"  Average Sortino Ratio: {results_df['sortino_ratio'].mean():.2f}")
print(f"  Average Max Drawdown: {results_df['max_drawdown'].mean():.2%}")
print(f"  Average Win Rate: {results_df['win_rate'].mean():.1%}")
print(f"  Average Profit Factor: {results_df['profit_factor'].mean():.2f}")

# Best strategies by different metrics
print(f"\nBEST STRATEGIES BY METRIC:")
print(f"  Best Sharpe: {results_df.groupby('strategy_name')['sharpe_ratio']
print(f"  Best Calmar: {results_df.groupby('strategy_name')['calmar_ratio']
print(f"  Best Return: {results_df.groupby('strategy_name')['total_return']
print(f"  Best Win Rate: {results_df.groupby('strategy_name')['win_rate'].m
statistical_results = perform_comprehensive_statistical_tests(results_df)
comprehensive_results = {
    'results_df': results_df,
    'all_results': all_results,
    'statistical_results': statistical_results,
    'config': CONFIG,
    'summary_stats': {
        'total_backtests': len(results_df),
        'unique_strategies': results_df['strategy_name'].nunique(),
        'unique_assets': results_df['symbol'].nunique(),
        'best_strategies': {
            'sharpe': results_df.groupby('strategy_name')['sharpe_ratio'].m
            'calmar': results_df.groupby('strategy_name')['calmar_ratio'].m
            'return': results_df.groupby('strategy_name')['total_return'].m
            'win_rate': results_df.groupby('strategy_name')['win_rate'].mea
        },
        'performance_summary': {
            'avg_total_return': results_df['total_return'].mean(),
            'avg_annual_return': results_df['annual_return'].mean(),
            'avg_sharpe_ratio': results_df['sharpe_ratio'].mean(),
            'avg_calmar_ratio': results_df['calmar_ratio'].mean(),
            'avg_sortino_ratio': results_df['sortino_ratio'].mean(),
            'avg_max_drawdown': results_df['max_drawdown'].mean(),
            'avg_win_rate': results_df['win_rate'].mean(),
            'avg_profit_factor': results_df['profit_factor'].mean()
        }
    },
    'timestamp': datetime.now()
}
```

```
        # Save all results
        try:
            with open('notebooks/04_COMPLETE_baseline_results.pkl', 'wb') as f:
                pickle.dump(comprehensive_results, f)

            # Save detailed CSV
            results_df.to_csv('notebooks/04_COMPLETE_baseline_results.csv', index=F

        except Exception as e:
            print(f"Error saving results: {e}")

        return results_df, all_results, statistical_results

    else:
        print("FAIL")
        return None, None, None
```

In [25]:
```
if __name__ == "__main__":
    results_df, all_results, statistical_results = run_complete_analysis()
    print("\mStart Analysis.")
```

```
  Trying: notebooks/01_comprehensive_results.pkl
  Found data under key: all_data
    BTCUSD: 10068 records
    ETHUSD: 10061 records
    SOLUSD: 10034 records
    XRPUSD: 10011 records
    ADAUSD: 10054 records
    SPY: 11092 records
    QQQ: 11214 records
    IWM: 11064 records
    DIA: 10932 records
    VTI: 12442 records
  Successfully loaded 10 symbols

Running comprehensive backtests...
Strategies: 22
Assets: 10
Total combinations: 220
================================================================================

BTCUSD (10068 records):
  BuyAndHold              : Return=583.61% | Sharpe= 0.46 | Calmar= 0.16 | MaxDD=3
1.23% | Trades=  1
  SMA_20_50               : Return=400.63% | Sharpe= 0.41 | Calmar= 0.11 | MaxDD=3
6.42% | Trades=1133
  EMA_12_26               : Return=1000.00% | Sharpe= 0.79 | Calmar= 0.31 | MaxDD=1
9.91% | Trades=1574
  SMA_5_20                : Return=1000.00% | Sharpe= 0.85 | Calmar= 0.24 | MaxDD=2
5.85% | Trades=2438
  EMA_10_30               : Return=1000.00% | Sharpe= 0.80 | Calmar= 0.35 | MaxDD=1
7.78% | Trades=1531
  RSI_14                  : Return=-95.00% | Sharpe=-2.13 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=527
  RSI_21                  : Return=-95.00% | Sharpe=-2.04 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=389
  BB_20_2.0               : Return=-95.00% | Sharpe=-2.40 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=443
  BB_10_1.5               : Return=-95.00% | Sharpe=-2.51 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=476
  BB_20_1.5               : Return=-95.00% | Sharpe=-2.42 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=484
  MACD_12_26_9            : Return=1000.00% | Sharpe= 1.83 | Calmar= 0.53 | MaxDD=1
1.61% | Trades=738
  MACD_8_21_5             : Return=1000.00% | Sharpe= 3.16 | Calmar= 0.64 | MaxDD=
9.72% | Trades=1134
  Momentum_20             : Return=1000.00% | Sharpe= 2.13 | Calmar= 0.49 | MaxDD=1
2.54% | Trades=1259
  Momentum_10             : Return=1000.00% | Sharpe= 2.63 | Calmar= 0.72 | MaxDD=
8.59% | Trades=1393
  Momentum_5              : Return=1000.00% | Sharpe= 3.08 | Calmar= 1.04 | MaxDD=
5.97% | Trades=1266
  MeanRev_20_2.0          : Return=-95.00% | Sharpe=-2.40 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=441
  MeanRev_10_1.5          : Return=-95.00% | Sharpe=-2.51 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=478
  Breakout_20             : Return=1000.00% | Sharpe= 4.73 | Calmar=10.00 | MaxDD=
```

```
0.47% | Trades=2192
  Breakout_10             : Return=1000.00% | Sharpe= 5.00 | Calmar=10.00 | MaxDD=
0.54% | Trades=3207
  Channel_20              : Return=1000.00% | Sharpe= 3.48 | Calmar= 0.72 | MaxDD=
8.59% | Trades=1162
  ML_RandomForest_20      : Return=-89.48% | Sharpe=-1.03 | Calmar=-0.06 | MaxDD=9
0.47% | Trades=231
  ML_LogisticRegression_20 : Return=-95.00% | Sharpe=-1.59 | Calmar=-0.07 | MaxDD=9
6.89% | Trades=343

ETHUSD (10061 records):
  BuyAndHold              : Return=204.92% | Sharpe= 0.26 | Calmar= 0.04 | MaxDD=6
4.56% | Trades=  1
  SMA_20_50               : Return= 57.43% | Sharpe= 0.15 | Calmar= 0.02 | MaxDD=6
1.33% | Trades=1086
  EMA_12_26               : Return=1000.00% | Sharpe= 0.80 | Calmar= 0.22 | MaxDD=2
8.37% | Trades=1594
  SMA_5_20                : Return=1000.00% | Sharpe= 0.81 | Calmar= 0.18 | MaxDD=3
3.72% | Trades=2424
  EMA_10_30               : Return=1000.00% | Sharpe= 0.81 | Calmar= 0.23 | MaxDD=2
7.16% | Trades=1506
  RSI_14                  : Return=-95.00% | Sharpe=-2.05 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=557
  RSI_21                  : Return=-95.00% | Sharpe=-1.79 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=339
  BB_20_2.0               : Return=-95.00% | Sharpe=-2.34 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=421
  BB_10_1.5               : Return=-95.00% | Sharpe=-2.50 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=452
  BB_20_1.5               : Return=-95.00% | Sharpe=-2.39 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=440
  MACD_12_26_9            : Return=1000.00% | Sharpe= 1.85 | Calmar= 0.41 | MaxDD=1
5.20% | Trades=737
  MACD_8_21_5             : Return=1000.00% | Sharpe= 3.33 | Calmar= 0.62 | MaxDD=
9.92% | Trades=1177
  Momentum_20             : Return=1000.00% | Sharpe= 2.27 | Calmar= 0.42 | MaxDD=1
4.70% | Trades=1465
  Momentum_10             : Return=1000.00% | Sharpe= 2.83 | Calmar= 0.39 | MaxDD=1
6.07% | Trades=1763
  Momentum_5              : Return=1000.00% | Sharpe= 3.55 | Calmar= 1.01 | MaxDD=
6.16% | Trades=1652
  MeanRev_20_2.0          : Return=-95.00% | Sharpe=-2.33 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=419
  MeanRev_10_1.5          : Return=-95.00% | Sharpe=-2.51 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=454
  Breakout_20             : Return=1000.00% | Sharpe= 4.80 | Calmar=10.00 | MaxDD=
0.40% | Trades=2210
  Breakout_10             : Return=1000.00% | Sharpe= 5.00 | Calmar=10.00 | MaxDD=
0.32% | Trades=3332
  Channel_20              : Return=1000.00% | Sharpe= 3.51 | Calmar= 0.51 | MaxDD=1
2.16% | Trades=1175
  ML_RandomForest_20      : Return=-95.00% | Sharpe=-1.45 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=260
  ML_LogisticRegression_20 : Return=-95.00% | Sharpe=-1.36 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=379
```

```
SOLUSD (10034 records):
  BuyAndHold              : Return=1000.00% | Sharpe= 0.41 | Calmar= 0.09 | MaxDD=6
7.88% | Trades=  1
  SMA_20_50               : Return=937.00% | Sharpe= 0.37 | Calmar= 0.10 | MaxDD=5
7.84% | Trades=896
  EMA_12_26               : Return=1000.00% | Sharpe= 0.89 | Calmar= 0.19 | MaxDD=3
2.64% | Trades=1383
  SMA_5_20                : Return=1000.00% | Sharpe= 1.08 | Calmar= 0.18 | MaxDD=3
5.31% | Trades=2032
  EMA_10_30               : Return=1000.00% | Sharpe= 0.94 | Calmar= 0.19 | MaxDD=3
2.56% | Trades=1244
  RSI_14                  : Return=-95.00% | Sharpe=-1.43 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=313
  RSI_21                  : Return=-95.00% | Sharpe=-1.46 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=257
  BB_20_2.0               : Return=-95.00% | Sharpe=-1.63 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=192
  BB_10_1.5               : Return=-95.00% | Sharpe=-1.66 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=197
  BB_20_1.5               : Return=-95.00% | Sharpe=-1.60 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=204
  MACD_12_26_9            : Return=1000.00% | Sharpe= 2.12 | Calmar= 0.30 | MaxDD=2
0.60% | Trades=746
  MACD_8_21_5             : Return=1000.00% | Sharpe= 3.63 | Calmar= 0.49 | MaxDD=1
2.70% | Trades=1182
  Momentum_20             : Return=1000.00% | Sharpe= 2.47 | Calmar= 0.27 | MaxDD=2
2.70% | Trades=1736
  Momentum_10             : Return=1000.00% | Sharpe= 3.37 | Calmar= 0.49 | MaxDD=1
2.71% | Trades=2212
  Momentum_5              : Return=1000.00% | Sharpe= 4.41 | Calmar= 0.55 | MaxDD=1
1.22% | Trades=2699
  MeanRev_20_2.0          : Return=-95.00% | Sharpe=-1.64 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=194
  MeanRev_10_1.5          : Return=-95.00% | Sharpe=-1.66 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=197
  Breakout_20             : Return=1000.00% | Sharpe= 5.00 | Calmar=10.00 | MaxDD=
0.23% | Trades=2340
  Breakout_10             : Return=1000.00% | Sharpe= 5.00 | Calmar=10.00 | MaxDD=
0.29% | Trades=3379
  Channel_20              : Return=1000.00% | Sharpe= 3.80 | Calmar= 0.49 | MaxDD=1
2.71% | Trades=1066
  ML_RandomForest_20      : Return=-95.00% | Sharpe=-1.39 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=245
  ML_LogisticRegression_20 : Return=-95.00% | Sharpe=-1.59 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=176

XRPUSD (10011 records):
  BuyAndHold              : Return=442.15% | Sharpe= 0.31 | Calmar= 0.08 | MaxDD=5
1.89% | Trades=  1
  SMA_20_50               : Return=132.05% | Sharpe= 0.21 | Calmar= 0.03 | MaxDD=6
7.11% | Trades=1091
  EMA_12_26               : Return=1000.00% | Sharpe= 0.72 | Calmar= 0.23 | MaxDD=2
7.52% | Trades=1588
  SMA_5_20                : Return=1000.00% | Sharpe= 0.78 | Calmar= 0.19 | MaxDD=3
2.57% | Trades=2309
  EMA_10_30               : Return=1000.00% | Sharpe= 0.73 | Calmar= 0.22 | MaxDD=2
```

```
8.04% | Trades=1528
  RSI_14                  : Return=-95.00% | Sharpe=-1.63 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=438
  RSI_21                  : Return=-95.00% | Sharpe=-1.38 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=241
  BB_20_2.0               : Return=-95.00% | Sharpe=-1.77 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=309
  BB_10_1.5               : Return=-95.00% | Sharpe=-2.16 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=366
  BB_20_1.5               : Return=-95.00% | Sharpe=-1.79 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=322
  MACD_12_26_9            : Return=1000.00% | Sharpe= 1.91 | Calmar= 0.29 | MaxDD=2
1.57% | Trades=781
  MACD_8_21_5             : Return=1000.00% | Sharpe= 3.03 | Calmar= 0.42 | MaxDD=1
4.71% | Trades=1223
  Momentum_20             : Return=1000.00% | Sharpe= 1.84 | Calmar= 0.24 | MaxDD=2
6.13% | Trades=1678
  Momentum_10             : Return=1000.00% | Sharpe= 2.64 | Calmar= 0.29 | MaxDD=2
1.51% | Trades=1955
  Momentum_5              : Return=1000.00% | Sharpe= 3.28 | Calmar= 0.42 | MaxDD=1
4.75% | Trades=2069
  MeanRev_20_2.0          : Return=-95.00% | Sharpe=-1.76 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=305
  MeanRev_10_1.5          : Return=-95.00% | Sharpe=-2.16 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=366
  Breakout_20             : Return=1000.00% | Sharpe= 4.53 | Calmar=10.00 | MaxDD=
0.30% | Trades=2104
  Breakout_10             : Return=1000.00% | Sharpe= 5.00 | Calmar=10.00 | MaxDD=
0.26% | Trades=3254
  Channel_20              : Return=1000.00% | Sharpe= 3.18 | Calmar= 0.38 | MaxDD=1
6.26% | Trades=1211
  ML_RandomForest_20      : Return=-95.00% | Sharpe=-1.32 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=262
  ML_LogisticRegression_20 : Return=-95.00% | Sharpe=-1.53 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=173

ADAUSD (10054 records):
  BuyAndHold              : Return=146.18% | Sharpe= 0.21 | Calmar= 0.03 | MaxDD=6
6.25% | Trades=  1
  SMA_20_50               : Return= 60.99% | Sharpe= 0.16 | Calmar= 0.02 | MaxDD=7
0.68% | Trades=954
  EMA_12_26               : Return=1000.00% | Sharpe= 0.77 | Calmar= 0.20 | MaxDD=3
1.42% | Trades=1466
  SMA_5_20                : Return=1000.00% | Sharpe= 0.81 | Calmar= 0.18 | MaxDD=3
3.63% | Trades=2099
  EMA_10_30               : Return=1000.00% | Sharpe= 0.82 | Calmar= 0.21 | MaxDD=2
9.98% | Trades=1375
  RSI_14                  : Return=-95.00% | Sharpe=-1.75 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=441
  RSI_21                  : Return=-95.00% | Sharpe=-1.60 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=293
  BB_20_2.0               : Return=-95.00% | Sharpe=-1.89 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=321
  BB_10_1.5               : Return=-95.00% | Sharpe=-2.15 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=332
  BB_20_1.5               : Return=-95.00% | Sharpe=-2.02 | Calmar=-0.07 | MaxDD=9
```

```
9.00% | Trades=340
  MACD_12_26_9            : Return=1000.00% | Sharpe= 1.80 | Calmar= 0.24 | MaxDD=2
6.06% | Trades=761
  MACD_8_21_5             : Return=1000.00% | Sharpe= 3.39 | Calmar= 0.48 | MaxDD=1
2.83% | Trades=1191
  Momentum_20             : Return=1000.00% | Sharpe= 2.22 | Calmar= 0.14 | MaxDD=4
4.32% | Trades=1783
  Momentum_10             : Return=1000.00% | Sharpe= 3.03 | Calmar= 0.26 | MaxDD=2
3.87% | Trades=2309
  Momentum_5              : Return=1000.00% | Sharpe= 4.01 | Calmar= 0.43 | MaxDD=1
4.44% | Trades=2461
  MeanRev_20_2.0          : Return=-95.00% | Sharpe=-1.90 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=323
  MeanRev_10_1.5          : Return=-95.00% | Sharpe=-2.15 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=332
  Breakout_20             : Return=1000.00% | Sharpe= 5.00 | Calmar=10.00 | MaxDD=
0.28% | Trades=2313
  Breakout_10             : Return=1000.00% | Sharpe= 5.00 | Calmar=10.00 | MaxDD=
0.27% | Trades=3360
  Channel_20              : Return=1000.00% | Sharpe= 3.62 | Calmar= 0.40 | MaxDD=1
5.44% | Trades=1118
  ML_RandomForest_20      : Return=-95.00% | Sharpe=-1.41 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=161
  ML_LogisticRegression_20 : Return=-95.00% | Sharpe=-1.42 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=142

SPY (11092 records):
  BuyAndHold              : Return= 19.60% | Sharpe= 0.20 | Calmar= 0.05 | MaxDD=
8.80% | Trades=  1
  SMA_20_50               : Return=-19.63% | Sharpe=-0.27 | Calmar=-0.02 | MaxDD=2
6.88% | Trades=1214
  EMA_12_26               : Return=  5.22% | Sharpe= 0.08 | Calmar= 0.02 | MaxDD=
6.68% | Trades=1034
  SMA_5_20                : Return=-23.31% | Sharpe=-0.34 | Calmar=-0.02 | MaxDD=2
8.90% | Trades=2526
  EMA_10_30               : Return=  2.67% | Sharpe= 0.04 | Calmar= 0.01 | MaxDD=1
0.43% | Trades=1386
  RSI_14                  : Return=-93.57% | Sharpe=-3.33 | Calmar=-0.06 | MaxDD=9
3.57% | Trades=1517
  RSI_21                  : Return=-66.50% | Sharpe=-1.47 | Calmar=-0.04 | MaxDD=6
6.50% | Trades=554
  BB_20_2.0               : Return=-93.93% | Sharpe=-3.27 | Calmar=-0.07 | MaxDD=9
3.93% | Trades=1280
  BB_10_1.5               : Return=-95.00% | Sharpe=-5.00 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=2363
  BB_20_1.5               : Return=-95.00% | Sharpe=-4.99 | Calmar=-0.07 | MaxDD=9
8.96% | Trades=2194
  MACD_12_26_9            : Return= -9.74% | Sharpe=-0.10 | Calmar=-0.01 | MaxDD=2
0.69% | Trades=814
  MACD_8_21_5             : Return=-20.31% | Sharpe=-0.23 | Calmar=-0.02 | MaxDD=2
7.69% | Trades=1297
  Momentum_20             : Return=  9.75% | Sharpe= 0.15 | Calmar= 0.48 | MaxDD=
0.44% | Trades= 14
  Momentum_10             : Return= 10.17% | Sharpe= 0.15 | Calmar= 0.83 | MaxDD=
0.27% | Trades=  6
  Momentum_5              : Return=  9.88% | Sharpe= 0.15 | Calmar= 0.84 | MaxDD=
```

```
0.26% | Trades=  4
  MeanRev_20_2.0          : Return=-93.93% | Sharpe=-3.27 | Calmar=-0.07 | MaxDD=9
3.93% | Trades=1280
  MeanRev_10_1.5          : Return=-95.00% | Sharpe=-5.00 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=2363
  Breakout_20             : Return=-26.32% | Sharpe=-0.38 | Calmar=-0.02 | MaxDD=3
1.69% | Trades=2840
  Breakout_10             : Return=-51.02% | Sharpe=-0.86 | Calmar=-0.03 | MaxDD=5
2.59% | Trades=3975
  Channel_20              : Return= 42.46% | Sharpe= 0.40 | Calmar= 0.06 | MaxDD=1
2.43% | Trades=1185
  ML_RandomForest_20      : Return=-66.03% | Sharpe=-1.42 | Calmar=-0.04 | MaxDD=6
6.07% | Trades=413
  ML_LogisticRegression_20 : Return=-53.49% | Sharpe=-1.03 | Calmar=-0.03 | MaxDD=5
3.76% | Trades=196

QQQ (11214 records):
  BuyAndHold              : Return= 36.40% | Sharpe= 0.29 | Calmar= 0.08 | MaxDD=
8.71% | Trades=  1
  SMA_20_50               : Return=-20.58% | Sharpe=-0.22 | Calmar=-0.02 | MaxDD=2
9.21% | Trades=1305
  EMA_12_26               : Return= 10.88% | Sharpe= 0.13 | Calmar= 0.03 | MaxDD=
6.89% | Trades=1402
  SMA_5_20                : Return=-12.66% | Sharpe=-0.13 | Calmar=-0.01 | MaxDD=2
1.19% | Trades=2828
  EMA_10_30               : Return= 17.60% | Sharpe= 0.19 | Calmar= 0.06 | MaxDD=
6.30% | Trades=1624
  RSI_14                  : Return=-95.00% | Sharpe=-3.45 | Calmar=-0.07 | MaxDD=9
5.52% | Trades=1492
  RSI_21                  : Return=-73.72% | Sharpe=-1.68 | Calmar=-0.04 | MaxDD=7
3.72% | Trades=622
  BB_20_2.0               : Return=-95.00% | Sharpe=-3.43 | Calmar=-0.07 | MaxDD=9
6.04% | Trades=1262
  BB_10_1.5               : Return=-95.00% | Sharpe=-5.00 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=2010
  BB_20_1.5               : Return=-95.00% | Sharpe=-5.00 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=1878
  MACD_12_26_9            : Return= 30.30% | Sharpe= 0.26 | Calmar= 0.09 | MaxDD=
6.82% | Trades=796
  MACD_8_21_5             : Return= 66.65% | Sharpe= 0.49 | Calmar= 0.14 | MaxDD=
8.45% | Trades=1262
  Momentum_20             : Return=  9.93% | Sharpe= 0.15 | Calmar= 0.18 | MaxDD=
1.17% | Trades= 42
  Momentum_10             : Return= 11.33% | Sharpe= 0.17 | Calmar= 0.61 | MaxDD=
0.39% | Trades= 10
  Momentum_5              : Return=  9.92% | Sharpe= 0.15 | Calmar= 0.64 | MaxDD=
0.33% | Trades=  4
  MeanRev_20_2.0          : Return=-95.00% | Sharpe=-3.43 | Calmar=-0.07 | MaxDD=9
6.04% | Trades=1262
  MeanRev_10_1.5          : Return=-95.00% | Sharpe=-5.00 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=2010
  Breakout_20             : Return= 64.19% | Sharpe= 0.58 | Calmar= 0.11 | MaxDD=1
0.21% | Trades=2784
  Breakout_10             : Return= 32.82% | Sharpe= 0.32 | Calmar= 0.03 | MaxDD=2
3.35% | Trades=3892
  Channel_20              : Return=226.74% | Sharpe= 1.15 | Calmar= 0.88 | MaxDD=
```

```
3.05% | Trades=1226
  ML_RandomForest_20       : Return=-63.71% | Sharpe=-1.30 | Calmar=-0.04 | MaxDD=6
3.82% | Trades=333
  ML_LogisticRegression_20 : Return=-47.76% | Sharpe=-0.86 | Calmar=-0.03 | MaxDD=4
8.04% | Trades=135

IWM (11064 records):
  BuyAndHold               : Return=  8.28% | Sharpe= 0.08 | Calmar= 0.01 | MaxDD=1
5.74% | Trades=  1
  SMA_20_50                : Return=-24.59% | Sharpe=-0.30 | Calmar=-0.02 | MaxDD=3
1.99% | Trades=1315
  EMA_12_26                : Return=  8.59% | Sharpe= 0.11 | Calmar= 0.03 | MaxDD=
6.40% | Trades=1448
  SMA_5_20                 : Return=-13.56% | Sharpe=-0.16 | Calmar=-0.02 | MaxDD=2
1.41% | Trades=2855
  EMA_10_30                : Return= 17.94% | Sharpe= 0.20 | Calmar= 0.06 | MaxDD=
6.03% | Trades=1611
  RSI_14                   : Return=-95.00% | Sharpe=-3.36 | Calmar=-0.07 | MaxDD=9
5.19% | Trades=1406
  RSI_21                   : Return=-66.82% | Sharpe=-1.43 | Calmar=-0.04 | MaxDD=6
6.82% | Trades=464
  BB_20_2.0                : Return=-95.00% | Sharpe=-3.50 | Calmar=-0.07 | MaxDD=9
6.17% | Trades=1246
  BB_10_1.5                : Return=-95.00% | Sharpe=-5.00 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=2044
  BB_20_1.5                : Return=-95.00% | Sharpe=-5.00 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=1865
  MACD_12_26_9             : Return= 32.62% | Sharpe= 0.27 | Calmar= 0.05 | MaxDD=1
3.36% | Trades=872
  MACD_8_21_5              : Return= 81.60% | Sharpe= 0.56 | Calmar= 0.12 | MaxDD=1
0.98% | Trades=1352
  Momentum_20              : Return= 14.82% | Sharpe= 0.21 | Calmar= 0.46 | MaxDD=
0.68% | Trades= 52
  Momentum_10              : Return= 10.95% | Sharpe= 0.16 | Calmar= 0.51 | MaxDD=
0.47% | Trades= 12
  Momentum_5               : Return=  9.35% | Sharpe= 0.14 | Calmar= 0.36 | MaxDD=
0.57% | Trades=  2
  MeanRev_20_2.0           : Return=-95.00% | Sharpe=-3.49 | Calmar=-0.07 | MaxDD=9
6.15% | Trades=1244
  MeanRev_10_1.5           : Return=-95.00% | Sharpe=-5.00 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=2044
  Breakout_20              : Return=103.72% | Sharpe= 0.83 | Calmar= 0.35 | MaxDD=
4.62% | Trades=2814
  Breakout_10              : Return= 92.15% | Sharpe= 0.74 | Calmar= 0.20 | MaxDD=
7.34% | Trades=3862
  Channel_20               : Return=264.68% | Sharpe= 1.23 | Calmar= 0.79 | MaxDD=
3.78% | Trades=1243
  ML_RandomForest_20       : Return=-75.88% | Sharpe=-1.68 | Calmar=-0.04 | MaxDD=7
5.90% | Trades=475
  ML_LogisticRegression_20 : Return=-77.78% | Sharpe=-1.76 | Calmar=-0.04 | MaxDD=7
7.89% | Trades=472

DIA (10932 records):
  BuyAndHold               : Return= 11.45% | Sharpe= 0.13 | Calmar= 0.03 | MaxDD=
8.59% | Trades=  1
  SMA_20_50                : Return=-26.21% | Sharpe=-0.44 | Calmar=-0.02 | MaxDD=3
```

```
0.98% | Trades=1186
  EMA_12_26               : Return= -5.39% | Sharpe=-0.09 | Calmar=-0.01 | MaxDD=1
2.50% | Trades=974
  SMA_5_20                : Return=-35.05% | Sharpe=-0.64 | Calmar=-0.03 | MaxDD=3
8.32% | Trades=2622
  EMA_10_30               : Return= -8.96% | Sharpe=-0.14 | Calmar=-0.01 | MaxDD=1
7.05% | Trades=1386
  RSI_14                  : Return=-90.93% | Sharpe=-3.05 | Calmar=-0.06 | MaxDD=9
0.93% | Trades=1402
  RSI_21                  : Return=-56.32% | Sharpe=-1.17 | Calmar=-0.03 | MaxDD=5
6.32% | Trades=462
  BB_20_2.0               : Return=-91.68% | Sharpe=-3.05 | Calmar=-0.06 | MaxDD=9
1.68% | Trades=1206
  BB_10_1.5               : Return=-95.00% | Sharpe=-5.00 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=2493
  BB_20_1.5               : Return=-95.00% | Sharpe=-4.94 | Calmar=-0.07 | MaxDD=9
8.65% | Trades=2203
  MACD_12_26_9            : Return=-28.64% | Sharpe=-0.36 | Calmar=-0.02 | MaxDD=3
2.96% | Trades=854
  MACD_8_21_5             : Return=-33.11% | Sharpe=-0.43 | Calmar=-0.02 | MaxDD=3
7.77% | Trades=1314
  Momentum_20             : Return=  9.26% | Sharpe= 0.14 | Calmar= 0.63 | MaxDD=
0.32% | Trades=  6
  Momentum_10             : Return=  9.26% | Sharpe= 0.14 | Calmar= 0.43 | MaxDD=
0.47% | Trades=  4
  Momentum_5              : Return=  9.52% | Sharpe= 0.15 | Calmar= 0.62 | MaxDD=
0.34% | Trades=  2
  MeanRev_20_2.0          : Return=-91.68% | Sharpe=-3.05 | Calmar=-0.06 | MaxDD=9
1.68% | Trades=1206
  MeanRev_10_1.5          : Return=-95.00% | Sharpe=-5.00 | Calmar=-0.07 | MaxDD=9
9.00% | Trades=2493
  Breakout_20             : Return=-36.60% | Sharpe=-0.58 | Calmar=-0.03 | MaxDD=4
0.03% | Trades=2768
  Breakout_10             : Return=-54.22% | Sharpe=-0.96 | Calmar=-0.03 | MaxDD=5
5.16% | Trades=3840
  Channel_20              : Return=  6.30% | Sharpe= 0.08 | Calmar= 0.01 | MaxDD=1
6.74% | Trades=1213
  ML_RandomForest_20      : Return=-64.03% | Sharpe=-1.37 | Calmar=-0.04 | MaxDD=6
4.29% | Trades=412
  ML_LogisticRegression_20 : Return=-61.86% | Sharpe=-1.29 | Calmar=-0.04 | MaxDD=6
1.95% | Trades=376

VTI (12442 records):
  BuyAndHold              : Return= 18.89% | Sharpe= 0.18 | Calmar= 0.04 | MaxDD=
9.74% | Trades=  1
  SMA_20_50               : Return=-18.06% | Sharpe=-0.22 | Calmar=-0.02 | MaxDD=2
6.11% | Trades=1322
  EMA_12_26               : Return=  1.57% | Sharpe= 0.03 | Calmar= 0.00 | MaxDD=
9.44% | Trades=1064
  SMA_5_20                : Return=-37.59% | Sharpe=-0.56 | Calmar=-0.02 | MaxDD=4
1.62% | Trades=2854
  EMA_10_30               : Return=  0.16% | Sharpe= 0.01 | Calmar= 0.00 | MaxDD=1
1.19% | Trades=1466
  RSI_14                  : Return=-94.21% | Sharpe=-3.25 | Calmar=-0.06 | MaxDD=9
4.21% | Trades=1644
  RSI_21                  : Return=-61.79% | Sharpe=-1.25 | Calmar=-0.03 | MaxDD=6
```

```
1.79% | Trades=500
  BB_20_2.0               : Return=-94.69% | Sharpe=-3.21 | Calmar=-0.06 | MaxDD=9
4.69% | Trades=1410
  BB_10_1.5               : Return=-95.00% | Sharpe=-5.00 | Calmar=-0.06 | MaxDD=9
9.00% | Trades=2396
  BB_20_1.5               : Return=-95.00% | Sharpe=-5.00 | Calmar=-0.06 | MaxDD=9
9.00% | Trades=2373
  MACD_12_26_9            : Return=-30.66% | Sharpe=-0.35 | Calmar=-0.02 | MaxDD=3
5.49% | Trades=955
  MACD_8_21_5             : Return=-39.08% | Sharpe=-0.48 | Calmar=-0.02 | MaxDD=4
1.73% | Trades=1513
  Momentum_20             : Return= 10.54% | Sharpe= 0.15 | Calmar= 1.76 | MaxDD=
0.12% | Trades=  4
  Momentum_10             : Return=  9.96% | Sharpe= 0.14 | Calmar= 1.66 | MaxDD=
0.12% | Trades=  2
  Momentum_5              : Return= 10.34% | Sharpe= 0.15 | Calmar= 2.00 | MaxDD=
0.10% | Trades=  4
  MeanRev_20_2.0          : Return=-94.68% | Sharpe=-3.20 | Calmar=-0.06 | MaxDD=9
4.68% | Trades=1408
  MeanRev_10_1.5          : Return=-95.00% | Sharpe=-5.00 | Calmar=-0.06 | MaxDD=9
9.00% | Trades=2396
  Breakout_20             : Return=-40.18% | Sharpe=-0.59 | Calmar=-0.02 | MaxDD=4
2.20% | Trades=3186
  Breakout_10             : Return=-60.15% | Sharpe=-1.02 | Calmar=-0.03 | MaxDD=6
1.39% | Trades=4426
  Channel_20              : Return= 19.00% | Sharpe= 0.19 | Calmar= 0.02 | MaxDD=1
5.27% | Trades=1364
  ML_RandomForest_20      : Return=-73.26% | Sharpe=-1.61 | Calmar=-0.04 | MaxDD=7
3.33% | Trades=479
  ML_LogisticRegression_20 : Return=-55.91% | Sharpe=-1.03 | Calmar=-0.03 | MaxDD=5
6.02% | Trades=227
Total successful backtests: 220
Strategies tested: 22
Assets analyzed: 10

KEY PERFORMANCE METRICS:
  Average Total Return: 234.54%
  Average Annual Return: -0.76%
  Average Sharpe Ratio: -0.34
  Average Calmar Ratio: 0.58
  Average Sortino Ratio: 0.31
  Average Max Drawdown: 49.38%
  Average Win Rate: 21.0%
  Average Profit Factor: 1.86

BEST STRATEGIES BY METRIC:
  Best Sharpe: Breakout_20
  Best Calmar: Breakout_20
  Best Return: Channel_20
  Best Win Rate: Channel_20

2. ANOVA Across All Strategies
--------------------------------------------------
ANOVA F-statistic: 5.801, p-value: 0.0000
Kruskal-Wallis H-statistic: 170.686, p-value: 0.0000
Result: Significant differences exist
```

```
3. Comprehensive Market Type Comparison
--------------------------------------------------
total_return   : Crypto=   4.972, Equity=  -0.282, p=0.0000 (Crypto better)
sharpe_ratio   : Crypto=   0.620, Equity=  -1.304, p=0.0000 (Crypto better)
max_drawdown   : Crypto=   0.540, Equity=   0.448, p=0.0833 (No difference)
win_rate       : Crypto=   0.252, Equity=   0.168, p=0.0024 (Crypto better)

4. Distribution Analysis
--------------------------------------------------
Shapiro-Wilk test: W=0.6431, p=0.0000
Jarque-Bera test: JB=44.7994, p=0.0000
Returns are NOT normally distributed (α=0.05)
\mStart Analysis.
```