```
!pip install backtrader yfinance pandas matplotlib
```

Show hidden output

```
import backtrader as bt
import yfinance as yf
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
from itertools import product
import warnings
warnings.filterwarnings('ignore')

# ==================== CONFIGURATION ====================
CONFIG = {
    'START_DATE': datetime.now() - timedelta(days=2*365),  # 2 years back
    'END_DATE': datetime.now(),
    'INITIAL_CASH': 100000,
    'COMMISSION': 0.001,  # 0.1%
    'TIMEFRAME': 'daily',  # Options: 'daily', '1h', '15m', '5m'
    'MAX_STRATEGIES_TO_TEST': 10000,  # Maximum combinations to test
    'TOP_N_RESULTS': 20,  # Show top N best strategies
}

# NIFTY 50 stocks (as of 2024)
NIFTY50_TICKERS = [
    'RELIANCE.NS', 'TCS.NS', 'HDFCBANK.NS', 'INFY.NS', 'ICICIBANK.NS',
    'HINDUNILVR.NS', 'ITC.NS', 'SBIN.NS', 'BHARTIARTL.NS', 'KOTAKBANK.NS',
    'LT.NS', 'AXISBANK.NS', 'ASIANPAINT.NS', 'MARUTI.NS', 'HCLTECH.NS',
    'WIPRO.NS', 'ULTRACEMCO.NS', 'TITAN.NS', 'BAJFINANCE.NS', 'NESTLEIND.NS',
    'SUNPHARMA.NS', 'TECHM.NS', 'ONGC.NS', 'NTPC.NS', 'TATAMOTORS.NS',
    'POWERGRID.NS', 'M&M.NS', 'TATASTEEL.NS', 'BAJAJFINSV.NS', 'ADANIPORTS.NS',
    'COALINDIA.NS', 'HINDALCO.NS', 'DIVISLAB.NS', 'HEROMOTOCO.NS', 'BRITANNIA.NS',
    'CIPLA.NS', 'EICHERMOT.NS', 'DRREDDY.NS', 'JSWSTEEL.NS', 'GRASIM.NS',
    'INDUSINDBK.NS', 'APOLLOHOSP.NS', 'BPCL.NS', 'TATACONSUM.NS', 'SHRIRAMFIN.NS',
    'ADANIENT.NS', 'SBILIFE.NS', 'HDFCLIFE.NS', 'BAJAJ-AUTO.NS', 'LTIM.NS'
]

# ==================== STRATEGY PARAMETER RANGES ====================
STRATEGY_PARAMS = {
    'MeanReversion': {
        'bb_period': [10, 20, 30, 50],
        'bb_devfactor': [1.5, 2, 2.5, 3],
    },
    'Momentum': {
        'fast_period': [5, 10, 15, 20],
        'slow_period': [30, 50, 100, 200],
    },
    'RSI': {
        'period': [7, 14, 21, 28],
        'oversold': [20, 25, 30, 35],
        'overbought': [65, 70, 75, 80],
    },
    'MACD': {
        'fast': [8, 12, 16],
        'slow': [21, 26, 30],
        'signal': [7, 9, 11],
    },
    'DualMA': {
        'fast': [5, 10, 20],
        'slow': [50, 100, 200],
    },
    'TripleMA': {
        'fast': [5, 10],
        'medium': [20, 50],
        'slow': [100, 200],
    },
    'Stochastic': {
        'period': [14, 21],
        'period_dfast': [3, 5],
        'oversold': [20, 30],
        'overbought': [70, 80],
    },
    'CCI': {
        'period': [14, 20, 30],
        'entry': [100, 150, 200],
    },
    'ADX': {
        'period': [14, 20],
        'threshold': [20, 25, 30],
```

```
        },
        'ATR': {
            'period': [14, 21],
            'multiplier': [2, 3, 4],
        },
    }


    # ==================== STRATEGIES ====================
    class MeanReversionStrategy(bt.Strategy):
        params = (('bb_period', 20), ('bb_devfactor', 2))

        def __init__(self):
            self.bbands = bt.indicators.BollingerBands(
                self.data.close, period=self.params.bb_period, devfactor=self.params.bb_devfactor
            )
            self.order = None

        def next(self):
            if self.order:
                return
            if not self.position:
                if self.data.close[0] < self.bbands.lines.bot[0]:
                    self.order = self.buy()
            else:
                if self.data.close[0] > self.bbands.lines.top[0]:
                    self.order = self.sell()

        def notify_order(self, order):
            if order.status in [order.Completed]:
                self.order = None


    class MomentumStrategy(bt.Strategy):
        params = (('fast_period', 10), ('slow_period', 50))

        def __init__(self):
            self.fast_ma = bt.indicators.SMA(self.data.close, period=self.params.fast_period)
            self.slow_ma = bt.indicators.SMA(self.data.close, period=self.params.slow_period)
            self.crossover = bt.indicators.CrossOver(self.fast_ma, self.slow_ma)
            self.order = None

        def next(self):
            if self.order:
                return
            if not self.position:
                if self.crossover > 0:
                    self.order = self.buy()
            else:
                if self.crossover < 0:
                    self.order = self.sell()

        def notify_order(self, order):
            if order.status in [order.Completed]:
                self.order = None


    class RSIStrategy(bt.Strategy):
        params = (('period', 14), ('oversold', 30), ('overbought', 70))

        def __init__(self):
            self.rsi = bt.indicators.RSI(self.data.close, period=self.params.period)
            self.order = None

        def next(self):
            if self.order:
                return
            if not self.position:
                if self.rsi[0] < self.params.oversold:
                    self.order = self.buy()
            else:
                if self.rsi[0] > self.params.overbought:
                    self.order = self.sell()

        def notify_order(self, order):
            if order.status in [order.Completed]:
                self.order = None


    class MACDStrategy(bt.Strategy):
        params = (('fast', 12), ('slow', 26), ('signal', 9))
```

```
        def __init__(self):
            self.macd = bt.indicators.MACD(
                self.data.close, period_me1=self.params.fast,
                period_me2=self.params.slow, period_signal=self.params.signal
            )
            self.crossover = bt.indicators.CrossOver(self.macd.macd, self.macd.signal)
            self.order = None

        def next(self):
            if self.order:
                return
            if not self.position:
                if self.crossover > 0:
                    self.order = self.buy()
            else:
                if self.crossover < 0:
                    self.order = self.sell()

        def notify_order(self, order):
            if order.status in [order.Completed]:
                self.order = None


    class DualMAStrategy(bt.Strategy):
        params = (('fast', 10), ('slow', 50))

        def __init__(self):
            self.fast_ema = bt.indicators.EMA(self.data.close, period=self.params.fast)
            self.slow_ema = bt.indicators.EMA(self.data.close, period=self.params.slow)
            self.crossover = bt.indicators.CrossOver(self.fast_ema, self.slow_ema)
            self.order = None

        def next(self):
            if self.order:
                return
            if not self.position:
                if self.crossover > 0:
                    self.order = self.buy()
            else:
                if self.crossover < 0:
                    self.order = self.sell()

        def notify_order(self, order):
            if order.status in [order.Completed]:
                self.order = None


    class TripleMAStrategy(bt.Strategy):
        params = (('fast', 5), ('medium', 20), ('slow', 100))

        def __init__(self):
            self.fast = bt.indicators.SMA(self.data.close, period=self.params.fast)
            self.medium = bt.indicators.SMA(self.data.close, period=self.params.medium)
            self.slow = bt.indicators.SMA(self.data.close, period=self.params.slow)
            self.order = None

        def next(self):
            if self.order:
                return
            if not self.position:
                if self.fast[0] > self.medium[0] > self.slow[0]:
                    self.order = self.buy()
            else:
                if self.fast[0] < self.medium[0]:
                    self.order = self.sell()

        def notify_order(self, order):
            if order.status in [order.Completed]:
                self.order = None


    class StochasticStrategy(bt.Strategy):
        params = (('period', 14), ('period_dfast', 3), ('oversold', 20), ('overbought', 80))

        def __init__(self):
            self.stoch = bt.indicators.Stochastic(
                self.data, period=self.params.period, period_dfast=self.params.period_dfast
            )
            self.order = None

        def next(self):
            if self.order:
```

```
                return
            if not self.position:
                if self.stoch.percK[0] < self.params.oversold and self.stoch.percD[0] < self.params.oversold:
                    self.order = self.buy()
            else:
                if self.stoch.percK[0] > self.params.overbought and self.stoch.percD[0] > self.params.overbought:
                    self.order = self.sell()

        def notify_order(self, order):
            if order.status in [order.Completed]:
                self.order = None


    class CCIStrategy(bt.Strategy):
        params = (('period', 20), ('entry', 100))

        def __init__(self):
            self.cci = bt.indicators.CCI(self.data, period=self.params.period)
            self.order = None

        def next(self):
            if self.order:
                return
            if not self.position:
                if self.cci[0] < -self.params.entry:
                    self.order = self.buy()
            else:
                if self.cci[0] > self.params.entry:
                    self.order = self.sell()

        def notify_order(self, order):
            if order.status in [order.Completed]:
                self.order = None


    class ADXStrategy(bt.Strategy):
        params = (('period', 14), ('threshold', 25))

        def __init__(self):
            self.adx = bt.indicators.ADX(self.data, period=self.params.period)
            self.dmi_plus = self.adx.lines.plusDI
            self.dmi_minus = self.adx.lines.minusDI
            self.order = None

        def next(self):
            if self.order:
                return
            if not self.position:
                if self.adx[0] > self.params.threshold and self.dmi_plus[0] > self.dmi_minus[0]:
                    self.order = self.buy()
            else:
                if self.dmi_minus[0] > self.dmi_plus[0]:
                    self.order = self.sell()

        def notify_order(self, order):
            if order.status in [order.Completed]:
                self.order = None


    class ATRStrategy(bt.Strategy):
        params = (('period', 14), ('multiplier', 3))

        def __init__(self):
            self.atr = bt.indicators.ATR(self.data, period=self.params.period)
            self.sma = bt.indicators.SMA(self.data.close, period=50)
            self.order = None

        def next(self):
            if self.order:
                return

            upper_band = self.sma[0] + (self.params.multiplier * self.atr[0])
            lower_band = self.sma[0] - (self.params.multiplier * self.atr[0])

            if not self.position:
                if self.data.close[0] < lower_band:
                    self.order = self.buy()
            else:
                if self.data.close[0] > upper_band:
                    self.order = self.sell()

        def notify_order(self, order):
```

```
            if order.status in [order.Completed]:
                self.order = None


    # ==================== DATA FETCHING ====================
    def get_data(ticker, start_date, end_date):
        """Download stock data from Yahoo Finance"""
        try:
            df = yf.download(ticker, start=start_date, end=end_date, auto_adjust=True, progress=False)

            if df.empty:
                return None

            if isinstance(df.columns, pd.MultiIndex):
                df.columns = df.columns.get_level_values(0)

            df.columns = [str(col).lower() for col in df.columns]
            df.rename(columns={'adj close': 'close'}, inplace=True)

            # Make sure we have all required columns
            required_cols = ['open', 'high', 'low', 'close', 'volume']
            for col in required_cols:
                if col not in df.columns:
                    return None

            # Return the dataframe, not the feed (we'll create feed in cerebro)
            return df
        except Exception as e:
            print(f"Error downloading {ticker}: {e}")
            return None


    # ==================== OPTIMIZER ====================
    def run_single_backtest(strategy_class, params, data_df, ticker, silent=True):
        """Run a single backtest and return results"""
        try:
            cerebro = bt.Cerebro()
            cerebro.addstrategy(strategy_class, **params)

            # Create data feed here inside cerebro context
            data = bt.feeds.PandasData(dataname=data_df)
            cerebro.adddata(data)

            cerebro.broker.setcash(CONFIG['INITIAL_CASH'])
            cerebro.broker.setcommission(commission=CONFIG['COMMISSION'])

            cerebro.addanalyzer(bt.analyzers.SharpeRatio, _name='sharpe', riskfreerate=0.06)
            cerebro.addanalyzer(bt.analyzers.DrawDown, _name='drawdown')
            cerebro.addanalyzer(bt.analyzers.Returns, _name='returns')
            cerebro.addanalyzer(bt.analyzers.TradeAnalyzer, _name='trades')

            results = cerebro.run()
            strat = results[0]

            final_value = cerebro.broker.getvalue()
            pnl = final_value - CONFIG['INITIAL_CASH']
            pnl_pct = (pnl / CONFIG['INITIAL_CASH']) * 100

            sharpe = strat.analyzers.sharpe.get_analysis().get('sharperatio', None)
            drawdown = strat.analyzers.drawdown.get_analysis()
            max_dd = drawdown.max.drawdown if hasattr(drawdown, 'max') else 0

            trades = strat.analyzers.trades.get_analysis()
            total_trades = trades.total.closed if hasattr(trades.total, 'closed') else 0

            win_rate = 0
            if total_trades > 0 and hasattr(trades, 'won'):
                won = trades.won.total if hasattr(trades.won, 'total') else 0
                win_rate = (won / total_trades * 100) if total_trades > 0 else 0

            return {
                'ticker': ticker,
                'strategy': strategy_class.__name__,
                'params': str(params),
                'final_value': final_value,
                'pnl': pnl,
                'return_pct': pnl_pct,
                'sharpe': sharpe if sharpe else 0,
                'max_drawdown': max_dd,
                'total_trades': total_trades,
                'win_rate': win_rate,
            }
```

```
        except Exception as e:
            if not silent:
                print(f"Error in backtest: {e}")
            return None


    def optimize_strategies():
        """Test all strategy combinations and find best performers"""
        print("="*80)
        print("NIFTY 50 STRATEGY OPTIMIZER")
        print("="*80)
        print(f"Configuration:")
        print(f"  - Period: {CONFIG['START_DATE'].strftime('%Y-%m-%d')} to {CONFIG['END_DATE'].strftime('%Y-%m-%d')}
        print(f"  - Initial Capital: ₹{CONFIG['INITIAL_CASH']:,}")
        print(f"  - Commission: {CONFIG['COMMISSION']*100}%")
        print(f"  - Max Strategies to Test: {CONFIG['MAX_STRATEGIES_TO_TEST']}")
        print("="*80)

        # Download all data first
        print("\nDownloading NIFTY 50 stock data...")
        stock_data = {}
        for ticker in NIFTY50_TICKERS:
            data_df = get_data(ticker, CONFIG['START_DATE'], CONFIG['END_DATE'])
            if data_df is not None and not data_df.empty:
                stock_data[ticker] = data_df
                print(f"✓ {ticker}")
            else:
                print(f"✗ {ticker} - Failed")

        print(f"\nSuccessfully loaded {len(stock_data)} stocks")

        # Generate all strategy combinations
        print("\nGenerating strategy combinations...")
        all_results = []
        test_count = 0

        strategy_classes = {
            'MeanReversion': MeanReversionStrategy,
            'Momentum': MomentumStrategy,
            'RSI': RSIStrategy,
            'MACD': MACDStrategy,
            'DualMA': DualMAStrategy,
            'TripleMA': TripleMAStrategy,
            'Stochastic': StochasticStrategy,
            'CCI': CCIStrategy,
            'ADX': ADXStrategy,
            'ATR': ATRStrategy,
        }

        for strategy_name, strategy_class in strategy_classes.items():
            param_names = list(STRATEGY_PARAMS[strategy_name].keys())
            param_values = [STRATEGY_PARAMS[strategy_name][p] for p in param_names]

            for param_combo in product(*param_values):
                params = dict(zip(param_names, param_combo))

                for ticker, data_df in stock_data.items():
                    if test_count >= CONFIG['MAX_STRATEGIES_TO_TEST']:
                        break

                    result = run_single_backtest(strategy_class, params, data_df, ticker)
                    if result:
                        all_results.append(result)
                        test_count += 1

                        if test_count % 100 == 0:
                            print(f"Progress: {test_count}/{CONFIG['MAX_STRATEGIES_TO_TEST']} tests completed...")

                if test_count >= CONFIG['MAX_STRATEGIES_TO_TEST']:
                    break

            if test_count >= CONFIG['MAX_STRATEGIES_TO_TEST']:
                break

        print(f"\nCompleted {test_count} backtests!")

        # Convert to DataFrame and sort
        df_results = pd.DataFrame(all_results)
        df_results = df_results.sort_values('return_pct', ascending=False)

        # Display top results
        print("\n" + "="*120)
```

```
        print(f"TOP {CONFIG['TOP_N_RESULTS']} BEST PERFORMING STRATEGIES")
        print("="*120)

        top_results = df_results.head(CONFIG['TOP_N_RESULTS'])

        # Format the table
        pd.set_option('display.max_columns', None)
        pd.set_option('display.width', None)
        pd.set_option('display.max_colwidth', 50)

        print(top_results[['ticker', 'strategy', 'params', 'return_pct', 'sharpe', 'max_drawdown', 'total_trades', '

        print("\n" + "="*120)
        print("SUMMARY STATISTICS")
        print("="*120)
        print(f"Total Strategies Tested: {len(df_results)}")
        print(f"Profitable Strategies: {len(df_results[df_results['return_pct'] > 0])}")
        print(f"Loss-Making Strategies: {len(df_results[df_results['return_pct'] < 0])}")
        print(f"Average Return: {df_results['return_pct'].mean():.2f}%")
        print(f"Best Return: {df_results['return_pct'].max():.2f}%")
        print(f"Worst Return: {df_results['return_pct'].min():.2f}%")
        print(f"Average Sharpe Ratio: {df_results['sharpe'].mean():.2f}")
        print("="*120)

        # Save results to CSV
        df_results.to_csv('backtest_results.csv', index=False)
        print("\nFull results saved to 'backtest_results.csv'")

        return df_results


    # ==================== MAIN EXECUTION ====================
    if __name__ == '__main__':
        results = optimize_strategies()
```

```
Worst Return: -3.35%
Average Sharpe Ratio: -141.40
================================================================================================================================

Full results saved to 'backtest_results.csv'
```

```
!pip install plotly tabulate pandas seaborn matplotlib
```

```
Requirement already satisfied: plotly in /usr/local/lib/python3.12/dist-packages (5.24.1)
Collecting tabulate
  Downloading tabulate-0.9.0-py3-none-any.whl.metadata (34 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dist-packages (0.13.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.12/dist-packages (from plotly) (9.1.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from plotly) (25.0)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (12.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->
Downloading tabulate-0.9.0-py3-none-any.whl (35 kB)
Installing collected packages: tabulate
Successfully installed tabulate-0.9.0
```

```
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import seaborn as sns
import matplotlib.pyplot as plt
from tabulate import tabulate

# ==================== CONFIGURATION ====================
CSV_FILE = '/content/backtest_results.csv'
TOP_N = 30  # Number of top strategies to display

# ==================== LOAD DATA ====================
print("Loading backtest results...")
df = pd.read_csv(CSV_FILE)

# Clean and prepare data
df['sharpe'] = pd.to_numeric(df['sharpe'], errors='coerce').fillna(0)
df['return_pct'] = pd.to_numeric(df['return_pct'], errors='coerce').fillna(0)
df['max_drawdown'] = pd.to_numeric(df['max_drawdown'], errors='coerce').fillna(0)
df['win_rate'] = pd.to_numeric(df['win_rate'], errors='coerce').fillna(0)
df['total_trades'] = pd.to_numeric(df['total_trades'], errors='coerce').fillna(0)

print(f"Loaded {len(df)} backtest results\n")

# ==================== BEAUTIFUL TABLE DISPLAY ====================
print("="*150)
print(f"{'TOP PERFORMING STRATEGIES':^150}")
print("="*150)

top_strategies = df.nlargest(TOP_N, 'return_pct')

# Format for display
display_df = top_strategies[['ticker', 'strategy', 'return_pct', 'sharpe', 'max_drawdown', 'total_trades', 'win_
display_df.columns = ['Ticker', 'Strategy', 'Return %', 'Sharpe', 'Max DD %', 'Trades', 'Win Rate %']
display_df['Return %'] = display_df['Return %'].apply(lambda x: f"{x:.2f}")
display_df['Sharpe'] = display_df['Sharpe'].apply(lambda x: f"{x:.3f}")
display_df['Max DD %'] = display_df['Max DD %'].apply(lambda x: f"{x:.2f}")
display_df['Win Rate %'] = display_df['Win Rate %'].apply(lambda x: f"{x:.2f}")

# Print beautiful table
print(tabulate(display_df, headers='keys', tablefmt='fancy_grid', showindex=False))
print("="*150 + "\n")

# ==================== SUMMARY STATISTICS TABLE ====================
print("="*150)
print(f"{'SUMMARY STATISTICS':^150}")
print("="*150)

summary_stats = pd.DataFrame({
```

```
        'Metric': [
            'Total Strategies Tested',
            'Profitable Strategies',
            'Loss-Making Strategies',
            'Win Rate',
            'Average Return %',
            'Median Return %',
            'Best Return %',
            'Worst Return %',
            'Std Dev of Returns %',
            'Average Sharpe Ratio',
            'Average Max Drawdown %',
            'Average Trades per Strategy'
        ],
        'Value': [
            len(df),
            len(df[df['return_pct'] > 0]),
            len(df[df['return_pct'] < 0]),
            f"{(len(df[df['return_pct'] > 0]) / len(df) * 100):.2f}%",
            f"{df['return_pct'].mean():.2f}%",
            f"{df['return_pct'].median():.2f}%",
            f"{df['return_pct'].max():.2f}%",
            f"{df['return_pct'].min():.2f}%",
            f"{df['return_pct'].std():.2f}%",
            f"{df['sharpe'].mean():.3f}",
            f"{df['max_drawdown'].mean():.2f}%",
            f"{df['total_trades'].mean():.1f}"
        ]
    })

    print(tabulate(summary_stats, headers='keys', tablefmt='fancy_grid', showindex=False))
    print("="*150 + "\n")

    # ==================== STRATEGY PERFORMANCE TABLE ====================
    print("="*150)
    print(f"{'STRATEGY TYPE PERFORMANCE':^150}")
    print("="*150)

    strategy_performance = df.groupby('strategy').agg({
        'return_pct': ['mean', 'median', 'max', 'min', 'std', 'count'],
        'sharpe': 'mean',
        'win_rate': 'mean',
        'max_drawdown': 'mean'
    }).round(2)

    strategy_performance.columns = ['Avg Return %', 'Median Return %', 'Best Return %', 'Worst Return %', 'Std Dev %
    strategy_performance = strategy_performance.sort_values('Avg Return %', ascending=False)

    print(tabulate(strategy_performance, headers='keys', tablefmt='fancy_grid'))
    print("="*150 + "\n")

    # ==================== STOCK PERFORMANCE TABLE ====================
    print("="*150)
    print(f"{'TOP 20 STOCKS BY AVERAGE RETURN':^150}")
    print("="*150)

    stock_performance = df.groupby('ticker').agg({
        'return_pct': ['mean', 'max', 'min', 'count'],
        'sharpe': 'mean',
        'win_rate': 'mean'
    }).round(2)

    stock_performance.columns = ['Avg Return %', 'Best Return %', 'Worst Return %', 'Strategies Tested', 'Avg Sharpe
    stock_performance = stock_performance.sort_values('Avg Return %', ascending=False).head(20)

    print(tabulate(stock_performance, headers='keys', tablefmt='fancy_grid'))
    print("="*150 + "\n")

    # ==================== VISUALIZATIONS ====================
    print("Generating visualizations...\n")

    # 1. Top 20 Strategies Bar Chart
    fig1 = go.Figure()
    top_20 = df.nlargest(20, 'return_pct')
    fig1.add_trace(go.Bar(
        x=top_20.index,
        y=top_20['return_pct'],
        text=top_20['return_pct'].apply(lambda x: f"{x:.2f}%"),
        textposition='outside',
        marker=dict(
            color=top_20['return_pct'],
            colorscale='RdYlGn',
```

```
            showscale=True,
            colorbar=dict(title="Return %")
        ),
        hovertemplate='<b>%{customdata[0]}</b><br>Strategy: %{customdata[1]}<br>Return: %{y:.2f}%<br>Sharpe: %{custo
        customdata=np.column_stack((top_20['ticker'], top_20['strategy'], top_20['sharpe'], top_20['max_drawdown']))
    ))
    fig1.update_layout(
        title='Top 20 Strategies by Return %',
        xaxis_title='Strategy Index',
        yaxis_title='Return %',
        height=600,
        template='plotly_dark',
        showlegend=False
    )
    fig1.show()

    # 2. Return Distribution Histogram
    fig2 = go.Figure()
    fig2.add_trace(go.Histogram(
        x=df['return_pct'],
        nbinsx=50,
        marker=dict(
            color='skyblue',
            line=dict(color='black', width=1)
        ),
        hovertemplate='Return Range: %{x}<br>Count: %{y}<extra></extra>'
    ))
    fig2.add_vline(x=0, line_dash="dash", line_color="red", annotation_text="Breakeven")
    fig2.add_vline(x=df['return_pct'].mean(), line_dash="dash", line_color="green", annotation_text="Mean")
    fig2.update_layout(
        title='Distribution of Returns Across All Strategies',
        xaxis_title='Return %',
        yaxis_title='Frequency',
        height=500,
        template='plotly_white'
    )
    fig2.show()

    # 3. Sharpe Ratio vs Return Scatter Plot
    fig3 = px.scatter(
        df,
        x='sharpe',
        y='return_pct',
        color='strategy',
        size='total_trades',
        hover_data=['ticker', 'win_rate', 'max_drawdown'],
        title='Risk-Adjusted Returns: Sharpe Ratio vs Return %',
        labels={'sharpe': 'Sharpe Ratio', 'return_pct': 'Return %'},
        height=700
    )
    fig3.update_layout(template='plotly_dark')
    fig3.show()

    # 4. Strategy Performance Comparison Box Plot
    fig4 = go.Figure()
    for strategy in df['strategy'].unique():
        strategy_data = df[df['strategy'] == strategy]['return_pct']
        fig4.add_trace(go.Box(
            y=strategy_data,
            name=strategy,
            boxmean='sd'
        ))
    fig4.update_layout(
        title='Return Distribution by Strategy Type',
        yaxis_title='Return %',
        xaxis_title='Strategy',
        height=600,
        template='plotly_white',
        showlegend=False
    )
    fig4.show()

    # 5. Top Stocks Performance
    top_stocks = df.groupby('ticker')['return_pct'].mean().nlargest(15)
    fig5 = go.Figure()
    fig5.add_trace(go.Bar(
        x=top_stocks.values,
        y=top_stocks.index,
        orientation='h',
        marker=dict(
            color=top_stocks.values,
            colorscale='Viridis',
```

```
            showscale=True
        ),
        text=top_stocks.values.round(2),
        textposition='outside',
        hovertemplate='<b>%{y}</b><br>Avg Return: %{x:.2f}%<extra></extra>'
    ))
    fig5.update_layout(
        title='Top 15 Stocks by Average Return %',
        xaxis_title='Average Return %',
        yaxis_title='Stock Ticker',
        height=600,
        template='plotly_dark'
    )
    fig5.show()

    # 6. Heatmap: Strategy vs Stock Performance
    pivot_table = df.pivot_table(
        values='return_pct',
        index='strategy',
        columns='ticker',
        aggfunc='mean'
    )
    top_stocks_list = df.groupby('ticker')['return_pct'].mean().nlargest(15).index
    pivot_subset = pivot_table[top_stocks_list]

    fig6 = go.Figure(data=go.Heatmap(
        z=pivot_subset.values,
        x=pivot_subset.columns,
        y=pivot_subset.index,
        colorscale='RdYlGn',
        text=np.round(pivot_subset.values, 2),
        texttemplate='%{text}',
        textfont={"size": 10},
        hovertemplate='Strategy: %{y}<br>Stock: %{x}<br>Return: %{z:.2f}%<extra></extra>'
    ))
    fig6.update_layout(
        title='Strategy Performance Heatmap (Top 15 Stocks)',
        xaxis_title='Stock Ticker',
        yaxis_title='Strategy',
        height=600,
        template='plotly_dark'
    )
    fig6.show()

    # 7. Win Rate vs Return Scatter
    fig7 = px.scatter(
        df[df['total_trades'] > 5],  # Only strategies with meaningful trade count
        x='win_rate',
        y='return_pct',
        color='strategy',
        size='total_trades',
        hover_data=['ticker', 'sharpe', 'max_drawdown'],
        title='Win Rate vs Return % (Min 5 trades)',
        labels={'win_rate': 'Win Rate %', 'return_pct': 'Return %'},
        height=700
    )
    fig7.update_layout(template='plotly_white')
    fig7.show()

    # 8. Drawdown vs Return Scatter
    fig8 = px.scatter(
        df,
        x='max_drawdown',
        y='return_pct',
        color='sharpe',
        size='total_trades',
        hover_data=['ticker', 'strategy', 'win_rate'],
        title='Risk Analysis: Max Drawdown vs Return %',
        labels={'max_drawdown': 'Max Drawdown %', 'return_pct': 'Return %'},
        height=700,
        color_continuous_scale='RdYlGn'
    )
    fig8.update_layout(template='plotly_dark')
    fig8.show()

    # 9. Strategy Count Pie Chart
    strategy_counts = df['strategy'].value_counts()
    fig9 = go.Figure(data=[go.Pie(
        labels=strategy_counts.index,
        values=strategy_counts.values,
        hole=0.4,
        textinfo='label+percent',
```

```
    hovertemplate='<b>%{label}</b><br>Tests: %{value}<br>Percentage: %{percent}<extra></extra>'
)])
fig9.update_layout(
    title='Distribution of Tested Strategies',
    height=600,
    template='plotly_dark'
)
fig9.show()

# 10. Performance Metrics Comparison
top_10 = df.nlargest(10, 'return_pct')
fig10 = make_subplots(
    rows=2, cols=2,
    subplot_titles=('Return %', 'Sharpe Ratio', 'Max Drawdown %', 'Win Rate %'),
    specs=[[{'type': 'bar'}, {'type': 'bar'}],
           [{'type': 'bar'}, {'type': 'bar'}]]
)

# Return %
fig10.add_trace(go.Bar(
    x=top_10.index,
    y=top_10['return_pct'],
    name='Return %',
    marker_color='lightgreen',
    text=top_10['return_pct'].round(2),
    textposition='outside'
), row=1, col=1)

# Sharpe Ratio
fig10.add_trace(go.Bar(
    x=top_10.index,
    y=top_10['sharpe'],
    name='Sharpe',
    marker_color='lightblue',
    text=top_10['sharpe'].round(2),
    textposition='outside'
), row=1, col=2)

# Max Drawdown
fig10.add_trace(go.Bar(
    x=top_10.index,
    y=top_10['max_drawdown'],
    name='Max DD %',
    marker_color='salmon',
    text=top_10['max_drawdown'].round(2),
    textposition='outside'
), row=2, col=1)

# Win Rate
fig10.add_trace(go.Bar(
    x=top_10.index,
    y=top_10['win_rate'],
    name='Win Rate %',
    marker_color='gold',
    text=top_10['win_rate'].round(2),
    textposition='outside'
), row=2, col=2)

fig10.update_layout(
    title_text='Top 10 Strategies - Key Metrics Comparison',
    showlegend=False,
    height=800,
    template='plotly_white'
)
fig10.show()

print("\n" + "="*150)
print("VISUALIZATION COMPLETE!")
print("="*150)
print("\nAll charts displayed above. Scroll up to see:")
print("  1. Top 20 Strategies Bar Chart")
print("  2. Return Distribution Histogram")
print("  3. Sharpe Ratio vs Return Scatter")
print("  4. Strategy Performance Box Plot")
print("  5. Top Stocks Performance")
print("  6. Strategy vs Stock Heatmap")
print("  7. Win Rate vs Return Analysis")
print("  8. Risk Analysis (Drawdown vs Return)")
print("  9. Strategy Distribution Pie Chart")
print(" 10. Top 10 Metrics Comparison")
print("="*150)
```

```
Loading backtest results...
Loaded 6372 backtest results

================================================================================================================
                                                TOP PERFORMING STRATEGIES
================================================================================================================
```

Ticker	Strategy	Return %	Sharpe	Max DD %	Trades	Win Rate %
MARUTI.NS	RSIStrategy	5.57	-1.99	1.01	3	100
ULTRACEMCO.NS	StochasticStrategy	4.66	-3.847	0.83	6	100
MARUTI.NS	MACDStrategy	4.62	-2.057	2.02	16	31.25
MARUTI.NS	MACDStrategy	4.53	-2.048	2.1	16	37.5
MARUTI.NS	RSIStrategy	4.46	-3.751	1.39	6	66.67
ULTRACEMCO.NS	StochasticStrategy	4.44	-4.141	0.89	6	100
ULTRACEMCO.NS	StochasticStrategy	4.39	-3.906	0.9	7	100
MARUTI.NS	MACDStrategy	4.32	-2.292	1.65	16	31.25
MARUTI.NS	StochasticStrategy	4.32	-2.681	1.4	6	83.33
MARUTI.NS	RSIStrategy	4.25	-3.439	2.64	1	100
MARUTI.NS	RSIStrategy	4.2	-3.413	2.64	1	100
MARUTI.NS	RSIStrategy	4.01	-2.349	1.75	1	100
MARUTI.NS	RSIStrategy	4.01	-2.349	1.75	1	100
DIVISLAB.NS	MeanReversionStrategy	4	-5.646	0.56	11	81.82
ULTRACEMCO.NS	RSIStrategy	3.9	-4.543	0.86	10	100
MARUTI.NS	RSIStrategy	3.89	-3.579	1.42	5	80
MARUTI.NS	DualMAStrategy	3.8	-3.085	1.95	5	20
ULTRACEMCO.NS	StochasticStrategy	3.71	-4.339	0.9	7	100
MARUTI.NS	MACDStrategy	3.69	-2.081	2.51	18	27.78
ULTRACEMCO.NS	RSIStrategy	3.66	-5.343	0.86	9	100
MARUTI.NS	DualMAStrategy	3.63	-3.175	2.21	5	40
ULTRACEMCO.NS	MeanReversionStrategy	3.59	-5.658	1.46	3	100
MARUTI.NS	MACDStrategy	3.54	-2.187	2.34	18	27.78
ULTRACEMCO.NS	MeanReversionStrategy	3.54	-8.845	1.03	17	70.59
ULTRACEMCO.NS	RSIStrategy	3.54	-4.447	1.06	12	83.33
MARUTI.NS	RSIStrategy	3.49	-2.263	1.76	1	100
ULTRACEMCO.NS	MeanReversionStrategy	3.48	-5.525	1.47	4	100
MARUTI.NS	MomentumStrategy	3.46	-2.596	2.68	9	22.22
ULTRACEMCO.NS	ATRStrategy	3.45	-5.78	1.47	4	100
ULTRACEMCO.NS	ATRStrategy	3.45	-5.78	1.47	4	100

```
================================================================================================================

================================================================================================================
                                                SUMMARY STATISTICS
================================================================================================================
```

Metric	Value
Total Strategies Tested	6372
Profitable Strategies	3790
Loss-Making Strategies	2582
Win Rate	59.48%
Average Return %	0.18%
Median Return %	0.04%
Best Return %	5.57%

Worst Return %	-3.35%
Std Dev of Returns %	0.68%
Average Sharpe Ratio	-141.395
Average Max Drawdown %	0.53%
Average Trades per Strategy	7.4

==========

==========

STRATEGY TYPE PERFORMANCE

==========

strategy	Avg Return %	Median Return %	Best Return %	Worst Return %	Std Dev %
MeanReversionStrategy	0.3	0.12	4	-2.22	0.67
RSIStrategy	0.3	0.12	5.57	-1.22	0.69
CCIStrategy	0.29	0.14	3.36	-1.49	0.63
ATRStrategy	0.26	0.11	3.45	-1.21	0.65
StochasticStrategy	0.2	0.03	4.66	-2.31	0.77
MomentumStrategy	0.14	0.04	3.46	-2.87	0.64
DualMAStrategy	0.13	0	3.8	-1.82	0.71
MACDStrategy	0.02	-0.05	4.62	-1.56	0.63
TripleMAStrategy	-0.04	-0.04	2.63	-3.35	0.58

==========

==========

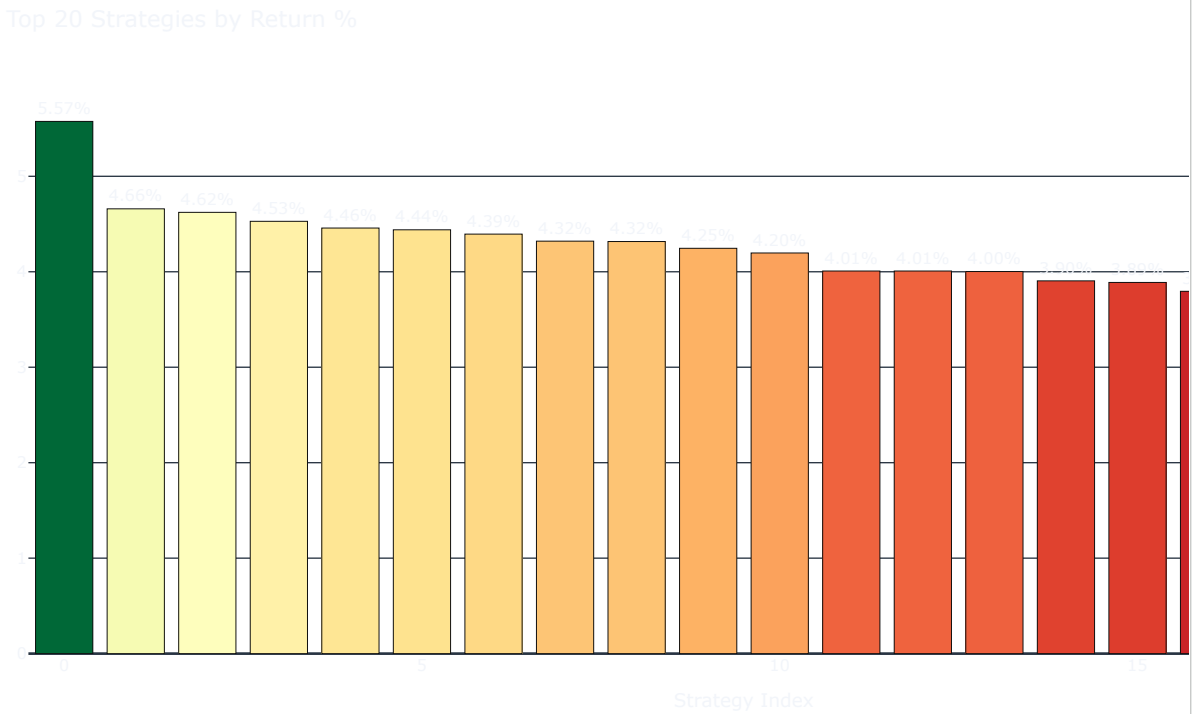
TOP 20 STOCKS BY AVERAGE RETURN

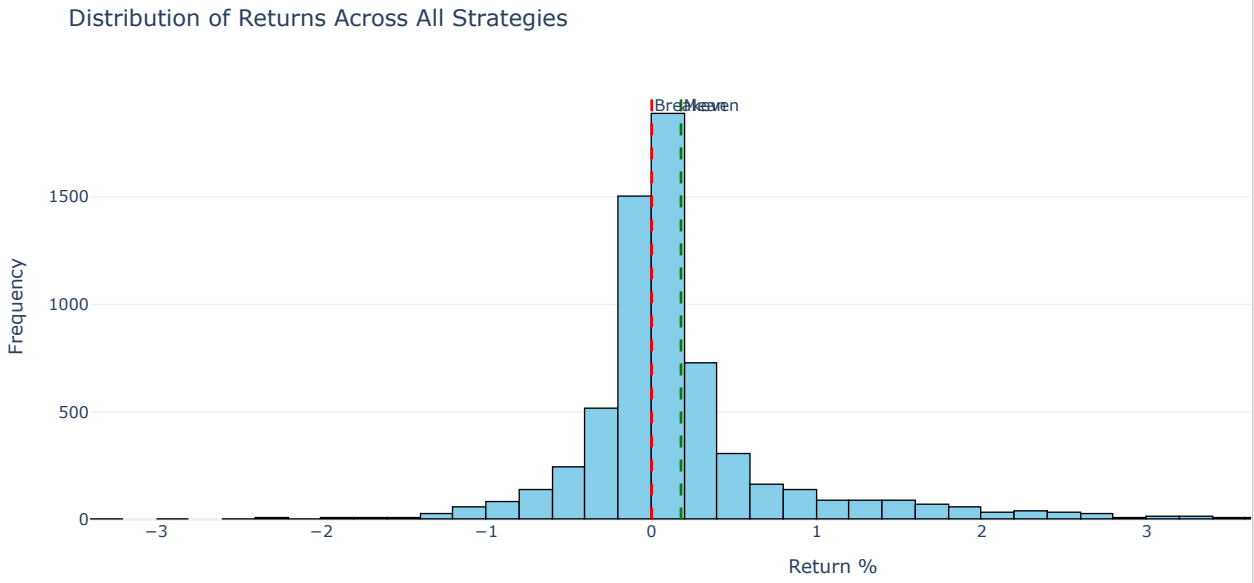
==========

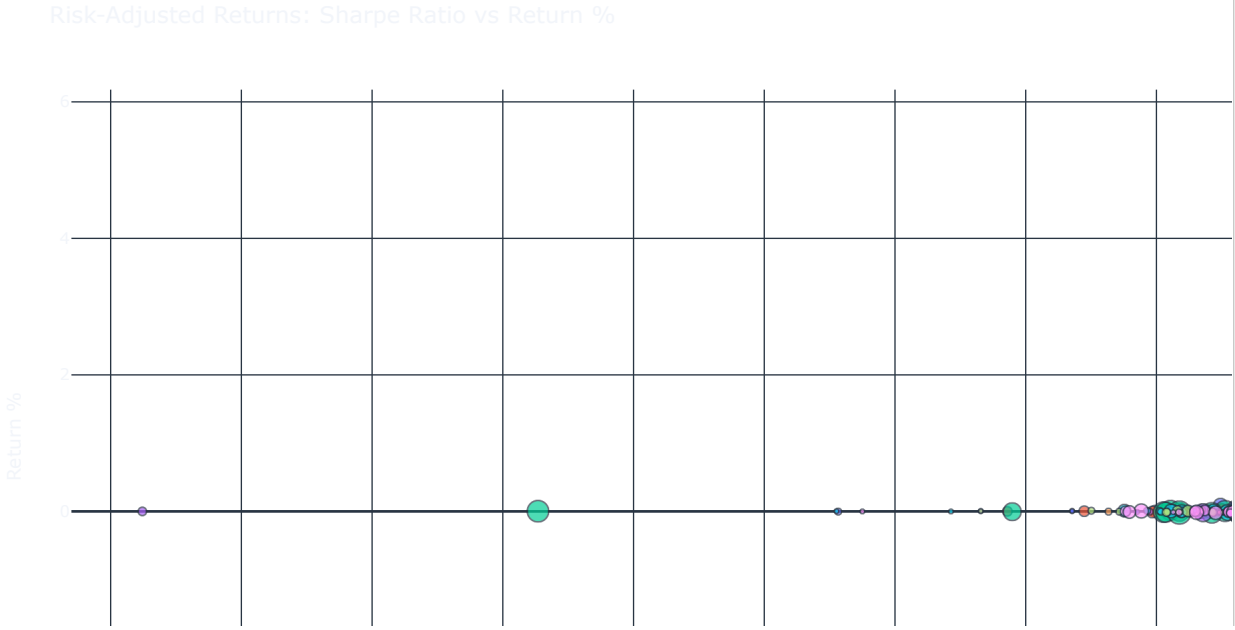
ticker	Avg Return %	Best Return %	Worst Return %	Strategies Tested	Avg Sharpe	Av
MARUTI.NS	2.17	5.57	0.21	141	-5.88	
EICHERMOT.NS	1.51	3.09	0.12	108	-10.59	
ULTRACEMCO.NS	1.07	4.66	-3.35	123	-15.78	
DIVISLAB.NS	0.98	4	-1.2	123	-38.4	
M&M.NS	0.73	2.12	-0.97	113	-36.3	
APOLLOHOSP.NS	0.51	2.48	-1.91	120	-24.37	
HEROMOTOCO.NS	0.44	2.54	-1.22	130	-20.17	
BRITANNIA.NS	0.31	1.93	-1.22	131	-19.29	
BHARTIARTL.NS	0.28	0.82	-0.11	110	-147.54	
TITAN.NS	0.27	1.54	-1.22	133	-45.49	
LT.NS	0.27	1.79	-0.98	123	-25.72	
GRASIM.NS	0.25	1.24	-0.76	119	-44.02	
BAJAJ-AUTO.NS	0.24	2.36	-2.31	133	-31.27	
HINDALCO.NS	0.17	0.46	-0.07	143	-110.33	
SBILIFE.NS	0.17	0.62	-0.3	131	-49.07	
KOTAKBANK.NS	0.15	0.81	-0.52	134	-73.06	
BAJAJFINSV.NS	0.14	0.83	-0.45	127	-58.04	
HDFCBANK.NS	0.12	0.34	-0.06	132	-185.1	
SHRIRAMFIN.NS	0.11	0.36	-0.11	127	-319.91	
ICICIBANK.NS	0.11	0.56	-0.28	120	-294.55	

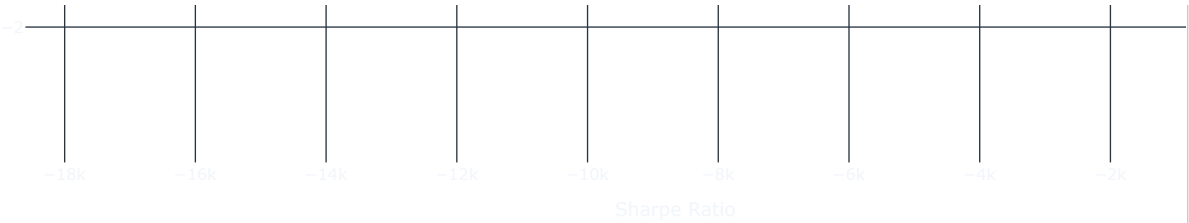
==========

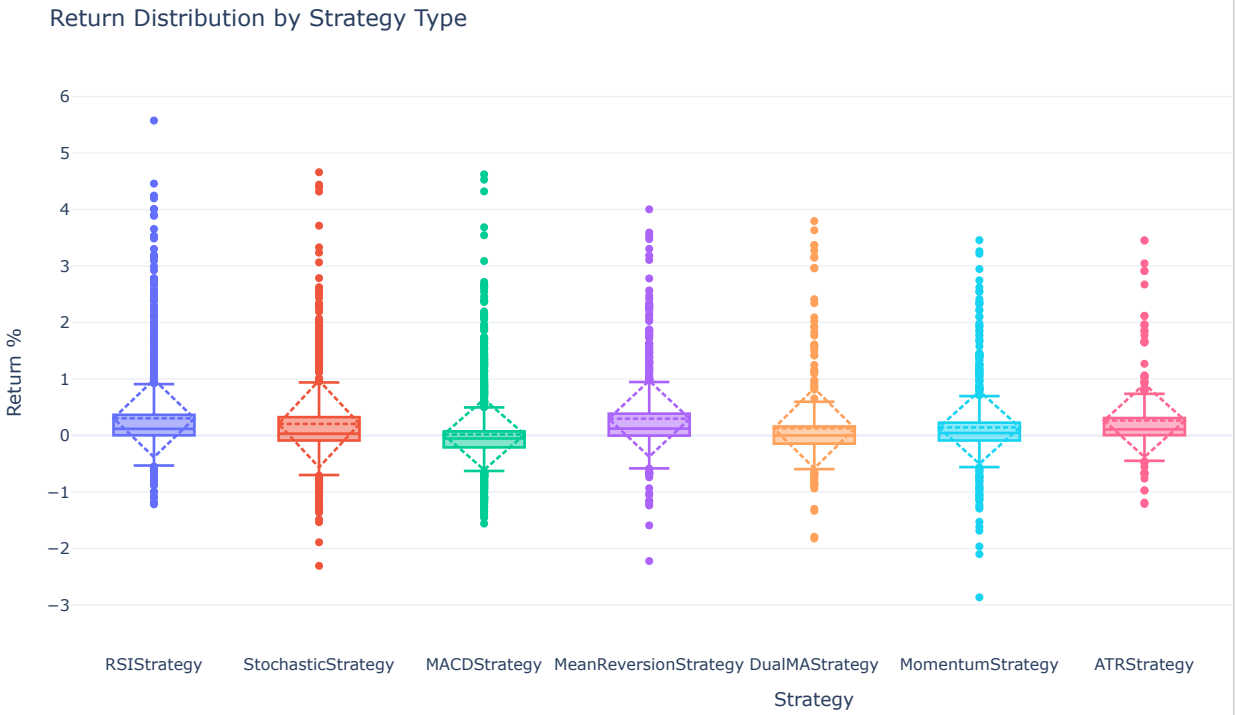
Generating visualizations...

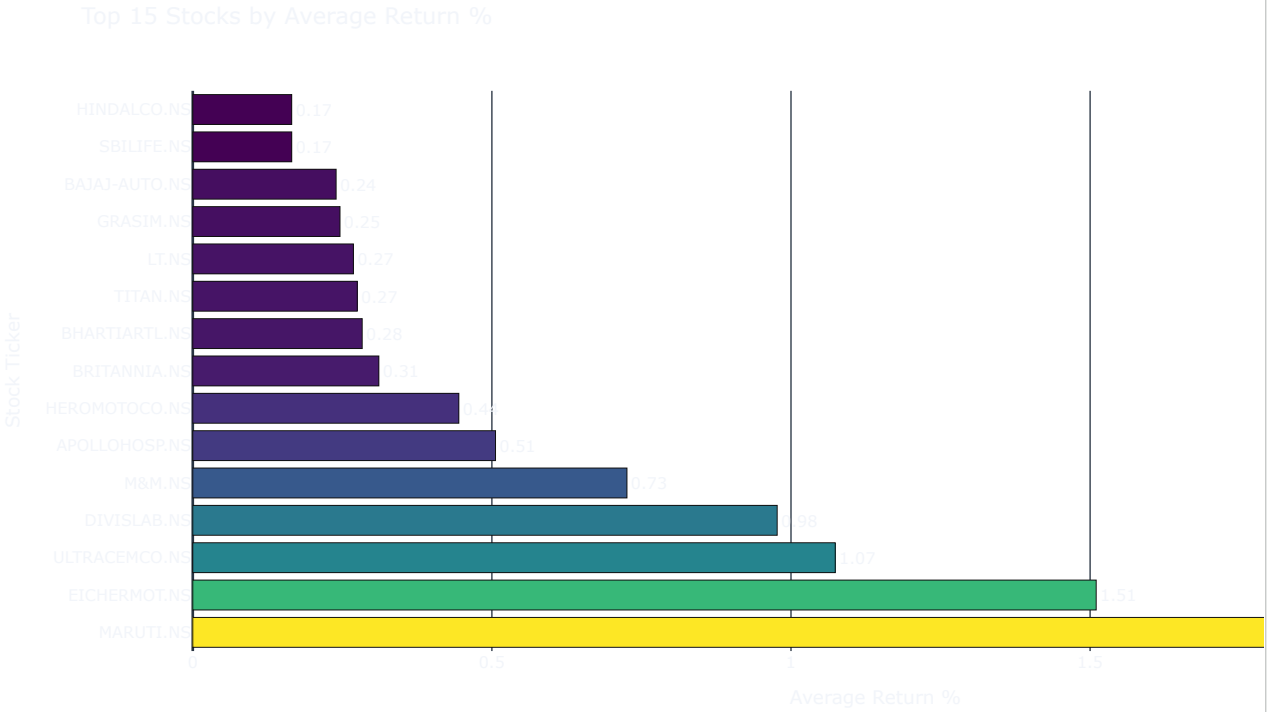

Top 20 Strategies by Return %
5.57%
4.66%
4.62%
4.53%
4.46%
4.44%
4.39%
4.32%
4.32%
4.25%
4.20%
4.01%
4.01%
4.00%
Return %
Strategy Index

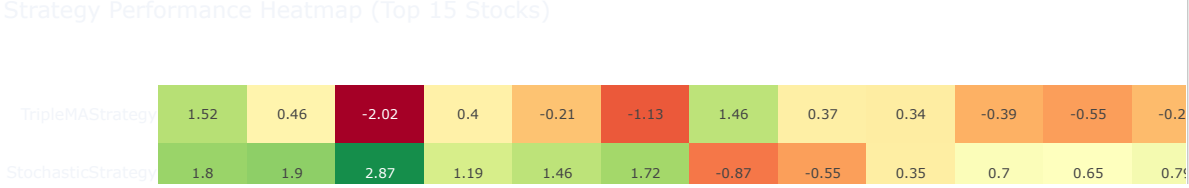

Distribution of Returns Across All Strategies
Frequency
1500
1000
500
0
−3
−2
−1
0
1
2
3
Return %

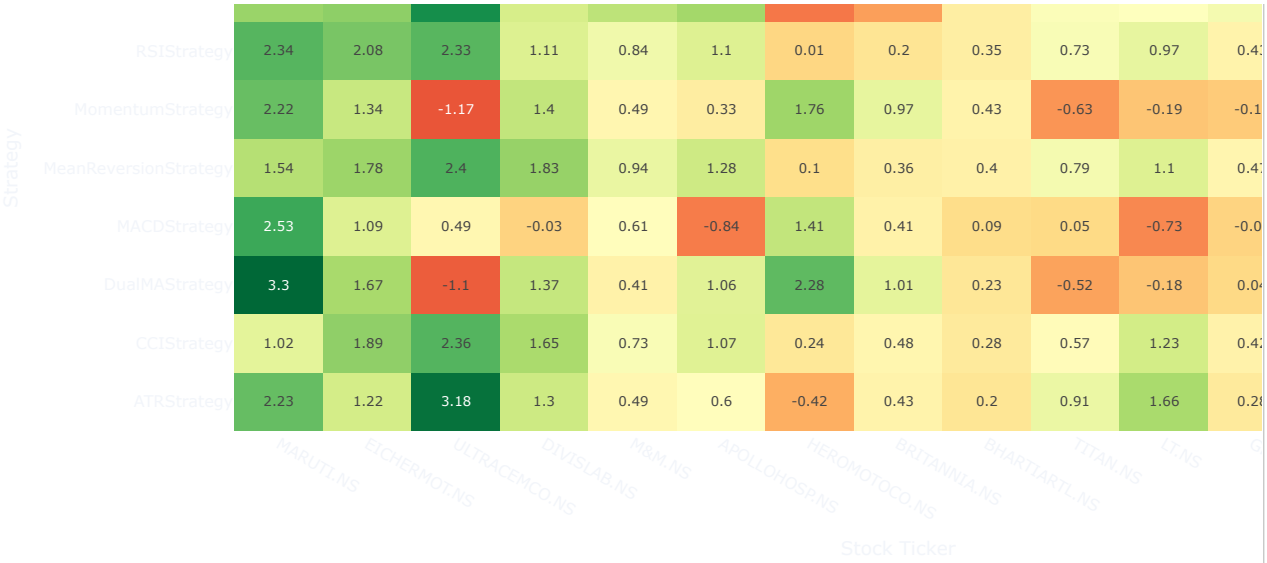

Risk-Adjusted Returns: Sharpe Ratio vs Return %
Return %

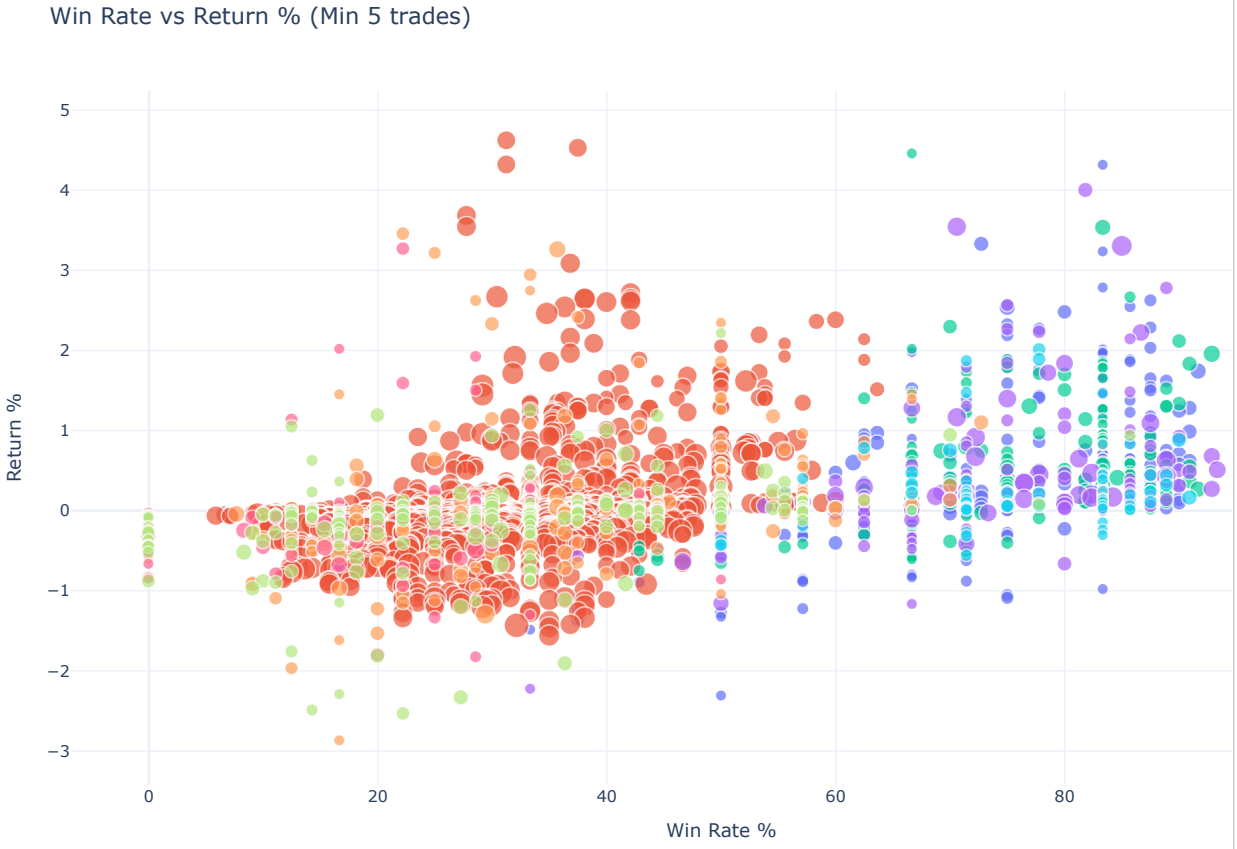

−18k
−16k
−14k
−12k
−10k
−8k
−6k
−4k
−2k
Sharpe Ratio

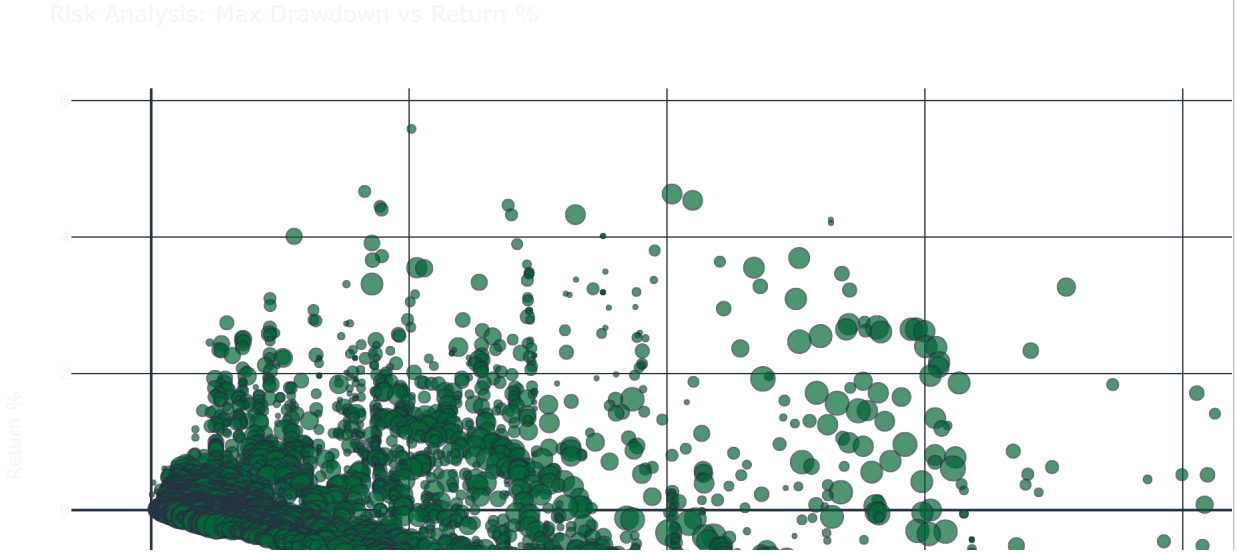

Return Distribution by Strategy Type
Return %
6
5
4
3
2
1
0
−1
−2
−3
RSIStrategy
StochasticStrategy
MACDStrategy
MeanReversionStrategy
DualMAStrategy
MomentumStrategy
ATRStrategy
Strategy

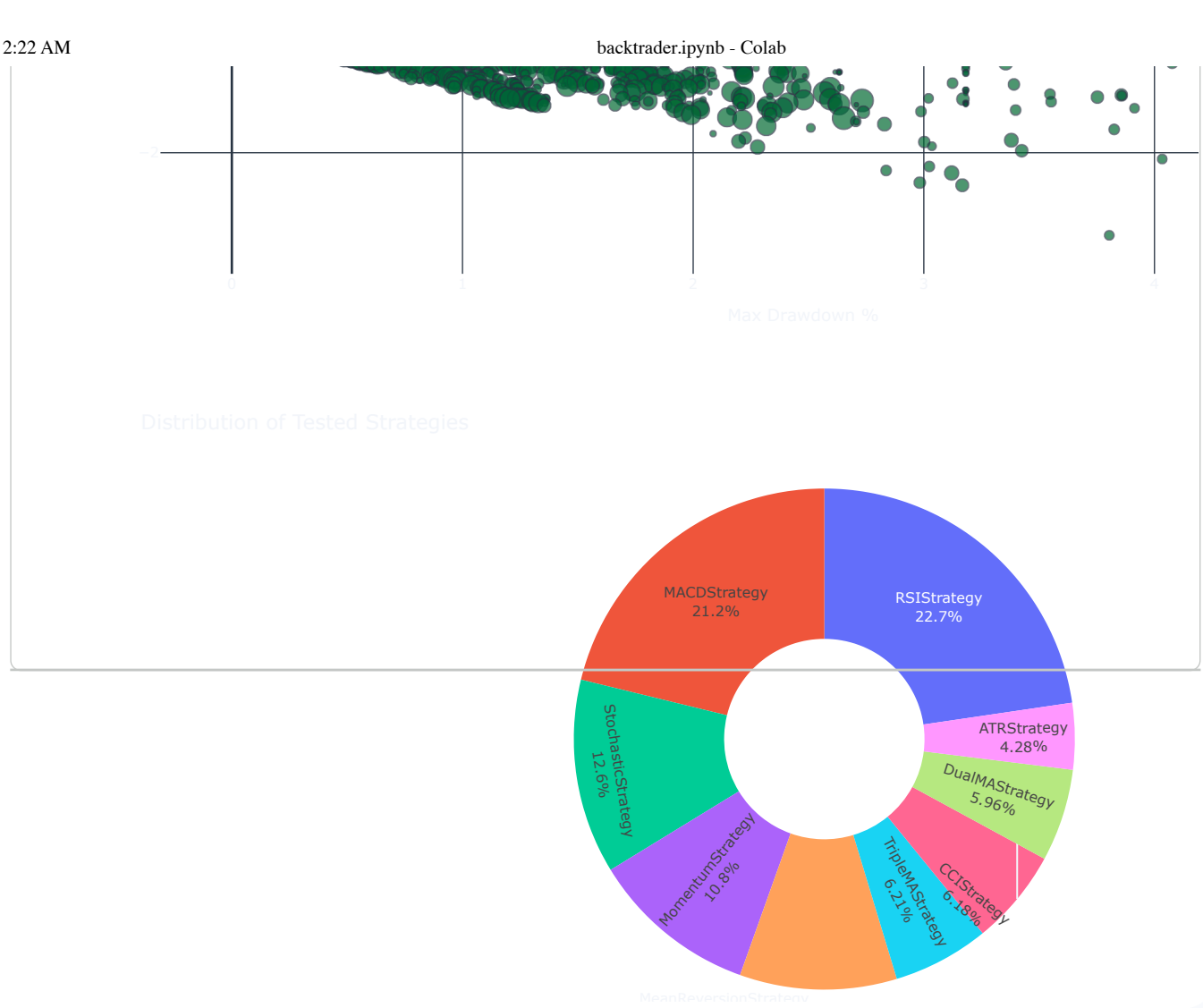

Top 15 Stocks by Average Return %
Stock Ticker
HINDALCO.NS
SBILIFE.NS
BAJAJ-AUTO.NS
GRASIM.NS
LT.NS
TITAN.NS
BHARTIARTL.NS
BRITANNIA.NS
HEROMOTOCO.NS
APOLLOHOSP.NS
M&M.NS
DIVISLAB.NS
ULTRACEMCO.NS
EICHERMOT.NS
MARUTI.NS
0.17
0.17
1.24
0.25
0.27
0.27
1.28
1.31
0.42
0.51
0.73
0.98
1.07
1.51
0
0.5
1
1.5
Average Return %


Strategy Performance Heatmap (Top 15 Stocks)
TripleMAStrategy
1.52
0.46
-2.02
0.4
-0.21
-1.13
1.46
0.37
0.34
-0.39
-0.55
-0.2
StochasticStrategy
1.8
1.9
2.87
1.19
1.46
1.72
-0.87
-0.55
0.35
0.7
0.65
0.7


RSIStrategy 2.34 2.08 2.33 1.11 0.84 1.1 0.01 0.2 0.35 0.73 0.97
MomentumStrategy 2.22 1.34 -1.17 1.4 0.49 0.33 1.76 0.97 0.43 -0.63 -0.19
MeanReversionStrategy 1.54 1.78 2.4 1.83 0.94 1.28 0.1 0.36 0.4 0.79 1.1
MACDStrategy 2.53 1.09 0.49 -0.03 0.61 -0.84 1.41 0.41 0.09 0.05 -0.73
DualMAStrategy 3.3 1.67 -1.1 1.37 0.41 1.06 2.28 1.01 0.23 -0.52 -0.18
CCIStrategy 1.02 1.89 2.36 1.65 0.73 1.07 0.24 0.48 0.28 0.57 1.23
ATRStrategy 2.23 1.22 3.18 1.3 0.49 0.6 -0.42 0.43 0.2 0.91 1.66
Strategy
MARUTI.NS EICHERMOT.NS ULTRACEMCO.NS DIVISLAB.NS M&M.NS APOLLOHOSP.NS HEROMOTOCO.NS BRITANNIA.NS BHARTIARTL.NS TITAN.NS LT.NS
Stock Ticker


Win Rate vs Return % (Min 5 trades)
Return %
5
4
3
2
1
0
−1
−2
−3
0
20
40
60
80
Win Rate %


Risk Analysis: Max Drawdown vs Return %
Return %


0
1
2
3
4
Max Drawdown %
Distribution of Tested Strategies
MACDStrategy
21.2%
RSIStrategy
22.7%
ATRStrategy
4.28%
DualMAStrategy
5.96%
CCIStrategy
6.18%
TripleMAStrategy
6.21%
MomentumStrategy
10.8%
StochasticStrategy
12.6%