K2 Algorithm for Causal Discovery in Financial Indicators

This notebook shows a demo of how K2 algorithm can be used to discover causal relationships among financial indicators. We will use a synthetic dataset of financial indicators and apply the K2 algorithm to learn the structure of the causal graph.

```
import numpy as np
import pandas as pd
import warnings
from scipy.stats import chi2_contingency
import yfinance as yf
from datetime import datetime, timedelta
```

```
class K2Algorithm:
    """
    K2 Algorithm for Bayesian Network Structure Learning

    The K2 algorithm discovers causal relationships between variables by
    constructing a Bayesian network structure. It greedily adds parents
    to each node to maximize the Bayesian Dirichlet score.

    Parameters:
    -----------
    max_parents : int
        Maximum number of parent nodes allowed for each variable
    """

    def __init__(self, max_parents=3):
        self.max_parents = max_parents
        self.structure = {}
        self.scores = {}

    def calculate_score(self, data, node, parents):
        """
        Calculate the K2 score for a node given its parents
        Uses Bayesian Dirichlet score
        """
        if len(parents) == 0:
            # No parents case
            counts = data[node].value_counts()
            n = len(data)
            r = len(counts)

            score = 0
            for count in counts.values:
                score += self._log_factorial(count)
            score -= self._log_factorial(n + r - 1)
            score += self._log_factorial(r - 1)
```

```
            return score

        # With parents case
        parent_cols = list(parents)
        score = 0
        parent_combinations = data[parent_cols].drop_duplicates()

        for _, parent_vals in parent_combinations.iterrows():
            mask = True
            for col in parent_cols:
                mask &= (data[col] == parent_vals[col])

            subset = data[mask][node]
            if len(subset) == 0:
                continue

            counts = subset.value_counts()
            n_ij = len(subset)
            r = len(data[node].unique())

            for count in counts.values:
                score += self._log_factorial(count)
            score -= self._log_factorial(n_ij + r - 1)
            score += self._log_factorial(r - 1)

        return score

    def _log_factorial(self, n):
        """Calculate log factorial efficiently"""
        if n <= 1:
            return 0
        return np.sum(np.log(np.arange(1, n + 1)))

    def fit(self, data, node_order=None):
        """
        Learn the Bayesian network structure using K2 algorithm

        Parameters:
        -----------
        data : DataFrame
            Discretized variables
        node_order : list, optional
            Causal ordering of nodes
        """
        if node_order is None:
            node_order = list(data.columns)

        self.structure = {node: [] for node in node_order}

        for i, node in enumerate(node_order):
            potential_parents = node_order[:i]

            if len(potential_parents) == 0:
                continue

            current_parents = []
```

```
            current_score = self.calculate_score(data, node, current_parents)

            improved = True
            while improved and len(current_parents) < self.max_parents:
                improved = False
                best_score = current_score
                best_parent = None

                for parent in potential_parents:
                    if parent not in current_parents:
                        test_parents = current_parents + [parent]
                        test_score = self.calculate_score(data, node, test_parents)

                        if test_score > best_score:
                            best_score = test_score
                            best_parent = parent
                            improved = True

                if improved:
                    current_parents.append(best_parent)
                    current_score = best_score

            self.structure[node] = current_parents
            self.scores[node] = current_score

        return self.structure

    def get_edges(self):
        """Return list of directed edges (parent -> child)"""
        edges = []
        for child, parents in self.structure.items():
            for parent in parents:
                edges.append((parent, child))
        return edges

    def find_all_paths(self, start, end, path=[]):
        """Find all causal paths from start node to end node"""
        path = path + [start]
        if start == end:
            return [path]
        paths = []
        for child, parents in self.structure.items():
            if start in parents and child not in path:
                newpaths = self.find_all_paths(child, end, path)
                paths.extend(newpaths)
        return paths
```

We will use BTCUSD, AAPL and GOLD data for the past 5 years to demonstrate the K2 algorithm for causal discovery. The K2 algorithm is a score-based method that identifies the most likely causal structure among a set of variables based on their observed data.

```
def fetch_market_data(ticker, start_date=None, end_date=None, period='1y'):
    """
    Fetch market data from Yahoo Finance
```

```
    Parameters:
    -----------
    ticker : str
        Ticker symbol:
        - 'BTC-USD' for Bitcoin
        - 'GC=F' for Gold Futures
        - 'AAPL' for Apple stock
    start_date : str, optional
        Start date in 'YYYY-MM-DD' format
    end_date : str, optional
        End date in 'YYYY-MM-DD' format
    period : str
        Period if dates not specified: '1mo', '3mo', '6mo', '1y', '2y', '5y'

    Returns:
    --------
    df : DataFrame
        DataFrame with Price column
    """
    print(f"Fetching data for {ticker}...")

    try:
        if start_date and end_date:
            data = yf.download(ticker, start=start_date, end=end_date, progress=False)
        else:
            data = yf.download(ticker, period=period, progress=False)

        if data.empty:
            raise ValueError(f"No data fetched for {ticker}")

        # Handle recent yfinance update where data['Close'] might be a DataFrame (MultiIndex)
        price_data = data['Close']
        if isinstance(price_data, pd.DataFrame):
            price_data = price_data.iloc[:, 0]

        df = pd.DataFrame({'Price': price_data})
        df = df.dropna()

        print(f"Fetched {len(df)} data points")
        print(f"Date range: {df.index[0].strftime('%Y-%m-%d')} to {df.index[-1].strftime('%Y-%
        print(f"Price range: ${df['Price'].min():.2f} - ${df['Price'].max():.2f}")

        return df

    except Exception as e:
        print(f"Error fetching data: {e}")
        return None
```

```
def calculate_technical_indicators(df):
    """
    Calculate comprehensive technical indicators

    Indicators:
    - RSI (Relative Strength Index)
    - EMA (Exponential Moving Averages)
    - MACD (Moving Average Convergence Divergence)
```

```
    - Bollinger Bands
    - ATR (Average True Range)
    - Magnitude % Move (target variable)
    """

    def calculate_rsi(prices, period=14):
        delta = prices.diff()
        gain = (delta.where(delta > 0, 0)).rolling(window=period).mean()
        loss = (-delta.where(delta < 0, 0)).rolling(window=period).mean()
        rs = gain / loss
        rsi = 100 - (100 / (1 + rs))
        return rsi

    def calculate_ema(prices, period):
        return prices.ewm(span=period, adjust=False).mean()

    def calculate_macd(prices, fast=12, slow=26, signal=9):
        ema_fast = calculate_ema(prices, fast)
        ema_slow = calculate_ema(prices, slow)
        macd_line = ema_fast - ema_slow
        signal_line = calculate_ema(macd_line, signal)
        return macd_line, signal_line

    # Calculate indicators
    df_ind = df.copy()

    df_ind['RSI_14'] = calculate_rsi(df_ind['Price'], 14)
    df_ind['EMA_9'] = calculate_ema(df_ind['Price'], 9)
    df_ind['EMA_20'] = calculate_ema(df_ind['Price'], 20)

    macd, signal = calculate_macd(df_ind['Price'])
    df_ind['MACD'] = macd
    df_ind['MACD_Signal'] = signal
    df_ind['MACD_Hist'] = df_ind['MACD'] - df_ind['MACD_Signal']

    # Bollinger Bands
    df_ind['BB_Middle'] = df_ind['Price'].rolling(window=20).mean()
    df_ind['BB_Std'] = df_ind['Price'].rolling(window=20).std()
    df_ind['BB_Upper'] = df_ind['BB_Middle'] + (df_ind['BB_Std'] * 2)
    df_ind['BB_Lower'] = df_ind['BB_Middle'] - (df_ind['BB_Std'] * 2)
    df_ind['BB_Width'] = (df_ind['BB_Upper'] - df_ind['BB_Lower']) / df_ind['BB_Middle']

    # ATR and Returns
    df_ind['Returns'] = df_ind['Price'].pct_change()
    df_ind['ATR'] = df_ind['Returns'].abs().rolling(window=14).mean() * 100

    # Target: Magnitude % move (5-period forward cumulative return)
    df_ind['Magnitude_Move_%'] = df_ind['Returns'].rolling(window=5).sum() * 100

    return df_ind.dropna()
```

K2 algorithm requires discrete variables. We'll convert continuous indicators into categorical bins.

```
def discretize_indicators(df, n_bins=5):
    """
```

```
    Discretize continuous indicators into categorical bins

    Parameters:
    -----------
    df : DataFrame
        DataFrame with continuous indicators
    n_bins : int
        Number of bins (typically 3-5)

    Returns:
    --------
    discretized : DataFrame
        DataFrame with discretized indicators
    """
    discretized = pd.DataFrame()

    indicators = ['RSI_14', 'EMA_9', 'EMA_20', 'MACD', 'MACD_Hist',
                  'BB_Width', 'ATR', 'Magnitude_Move_%']

    if n_bins == 5:
        labels = ['Very Low', 'Low', 'Medium', 'High', 'Very High']
    elif n_bins == 3:
        labels = ['Low', 'Medium', 'High']
    else:
        labels = [f'Bin_{i+1}' for i in range(n_bins)]

    for col in indicators:
        if col in df.columns:
            try:
                discretized[col] = pd.qcut(df[col], q=n_bins,
                                           labels=labels,
                                           duplicates='drop')
            except ValueError:
                discretized[col] = pd.cut(df[col], bins=n_bins,
                                          labels=labels)

    return discretized.dropna()
```

Statistical Analysis Functions

Functions to validate and analyze discovered causal relationships.

```
def calculate_mutual_information(df, var1, var2):
    """
    Calculate mutual information using Chi-square test
    """
    crosstab = pd.crosstab(df[var1], df[var2])
    chi2, p_value, dof, expected = chi2_contingency(crosstab)
    n = len(df)
    cramers_v = np.sqrt(chi2 / (n * (min(crosstab.shape) - 1)))
    return chi2, p_value, cramers_v

def analyze_conditional_probabilities(df, indicator, target):
    """
```

```
    Analyze conditional probability P(target | indicator)
    """
    crosstab = pd.crosstab(df[indicator], df[target], normalize='index') * 100
    return crosstab
```

```
ticker = 'BTC-USD'
asset_name = 'Bitcoin'

df_raw = fetch_market_data(ticker, period='3y')  # Last 3 years

print(f"\nFirst 5 rows of {asset_name} data:")
print(df_raw.head())
```

```
Fetching data for BTC-USD...
Fetched 1097 data points
Date range: 2023-02-08 to 2026-02-08
Price range: $20187.24 - $124752.53

First 5 rows of Bitcoin data:
                  Price
Date
2023-02-08  22939.398438
2023-02-09  21819.039062
2023-02-10  21651.183594
2023-02-11  21870.875000
2023-02-12  21788.203125
```

```
df_indicators = calculate_technical_indicators(df_raw)

print(f"\nCalculated {len(df_indicators.columns)-1} indicators")
print(f"  Total data points: {len(df_indicators)}")

print("\nIndicator Statistics:")
indicators_to_show = ['RSI_14', 'MACD', 'MACD_Hist', 'BB_Width', 'ATR', 'Magnitude_Move_%']
print(df_indicators[indicators_to_show].describe().round(2))

print("\nSample data with indicators:")
print(df_indicators[['Price', 'RSI_14', 'MACD_Hist', 'Magnitude_Move_%']].tail())
```

```
Calculated 14 indicators
  Total data points: 1078

Indicator Statistics:
        RSI_14     MACD  MACD_Hist  BB_Width      ATR  Magnitude_Move_%
count  1078.00  1078.00    1078.00   1078.00  1078.00           1078.00
mean     53.17   403.45     -16.63      0.16     1.71              0.65
std      17.29  1890.52     541.27      0.09     0.66              5.44
min       6.60 -5927.45   -2261.49      0.03     0.45            -21.34
25%      41.27  -667.08    -305.74      0.09     1.24             -2.32
50%      51.83   250.92     -30.59      0.14     1.63              0.26
75%      65.20  1319.73     287.09      0.19     2.03              3.46
max      96.26  7049.22    1882.86      0.47     4.17             22.19

Sample data with indicators:
                   Price     RSI_14    MACD_Hist  Magnitude_Move_%
Date
2026-02-04  73019.703125  17.627890 -1630.056596        -13.752471
2026-02-05  62702.097656  12.305688 -2261.493848        -21.335797
2026-02-06  70555.390625  28.123784 -2011.086886         -6.716588
2026-02-07  69281.968750  27.563531 -1798.945746        -10.748571
2026-02-08  71408.664062  32.680862 -1400.821976         -3.796286
```

```python
df_discrete = discretize_indicators(df_indicators, n_bins=5)

print(f"\nDiscretized into 5 bins")
print(f"  Final dataset shape: {df_discrete.shape}")

print("\nDistribution of Magnitude_Move_% bins:")
print(df_discrete['Magnitude_Move_%'].value_counts().sort_index())

print("\nSample discretized data:")
print(df_discrete.head(10))
```

```
Discretized into 5 bins
  Final dataset shape: (1078, 8)

Distribution of Magnitude_Move_% bins:
Magnitude_Move_%
Very Low     216
Low          215
Medium       216
High         215
Very High    216
Name: count, dtype: int64

Sample discretized data:
              RSI_14     EMA_9    EMA_20    MACD MACD_Hist BB_Width  \
Date
2023-02-27      High  Very Low  Very Low  Medium    Medium     High
2023-02-28      High  Very Low  Very Low  Medium    Medium     High
2023-03-01       Low  Very Low  Very Low  Medium    Medium     High
2023-03-02    Medium  Very Low  Very Low  Medium    Medium     High
2023-03-03  Very Low  Very Low  Very Low  Medium       Low     High
2023-04-04  Very Low  Very Low  Very Low  Medium       Low   Medium
2023-03-05  Very Low  Very Low  Very Low     Low       Low   Medium
2023-03-06  Very Low  Very Low  Very Low     Low       Low   Medium
2023-03-07  Very Low  Very Low  Very Low     Low       Low   Medium
2023-03-08  Very Low  Very Low  Very Low     Low       Low     High

                ATR Magnitude_Move_%
Date
2023-02-27  Very High             Low
2023-02-28  Very High         Very Low
2023-03-01     Medium             High
2023-03-02     Medium             High
2023-03-03     Medium         Very Low
2023-03-04     Medium         Very Low
2023-03-05        Low              Low
2023-03-06        Low         Very Low
2023-03-07        Low         Very Low
2023-03-08        Low              Low
```

CAUSAL STRUCTURE DISCOVERY WITH K2

```python
# Define node ordering - CRITICAL for K2
# Indicators come before target variable
node_order = ['RSI_14', 'EMA_9', 'EMA_20', 'MACD', 'MACD_Hist',
              'BB_Width', 'ATR', 'Magnitude_Move_%']

print(f"\nNode ordering (causal precedence):")
for i, node in enumerate(node_order, 1):
    print(f"  {i}. {node}")

# Initialize and fit K2
print("\nRunning K2 algorithm...")
k2 = K2Algorithm(max_parents=3)
structure = k2.fit(df_discrete[node_order], node_order=node_order)
```

```
print("\nCausal structure discovered!")
```

```
Node ordering (causal precedence):
  1. RSI_14
  2. EMA_9
  3. EMA_20
  4. MACD
  5. MACD_Hist
  6. BB_Width
  7. ATR
  8. Magnitude_Move_%

Running K2 algorithm...

Causal structure discovered!
```

Display Discovered Causal Relationships

```
for child, parents in structure.items():
    if parents:
        print(f"\n{child}:")
        print(f"  Direct causes (parents): {', '.join(parents)}")
        print(f"  Number of parents: {len(parents)}")
        print(f"  K2 Score: {k2.scores[child]:.2f}")
    else:
        print(f"\n{child}: No parents (root node)")

# Extract edges
edges = k2.get_edges()
print("\n" + "="*70)
print("DIRECTED EDGES (Parent → Child)")
print("="*70)
for parent, child in edges:
    print(f"  {parent} → {child}")

print(f"\nTotal edges discovered: {len(edges)}")
```

```
RSI_14: No parents (root node)

EMA_9: No parents (root node)

EMA_20:
  Direct causes (parents): EMA_9
  Number of parents: 1
  K2 Score: -333.90

MACD:
  Direct causes (parents): RSI_14, EMA_20
  Number of parents: 2
  K2 Score: -1306.50

MACD_Hist:
  Direct causes (parents): RSI_14, EMA_20
  Number of parents: 2
  K2 Score: -1343.79

BB_Width:
  Direct causes (parents): MACD, EMA_20, MACD_Hist
  Number of parents: 3
  K2 Score: -1414.77

ATR:
  Direct causes (parents): BB_Width, RSI_14, EMA_9
  Number of parents: 3
  K2 Score: -1449.70

Magnitude_Move_%:
  Direct causes (parents): MACD_Hist
  Number of parents: 1
  K2 Score: -1540.33

======================================================================
DIRECTED EDGES (Parent → Child)
======================================================================
  EMA_9 → EMA_20
  RSI_14 → MACD
  EMA_20 → MACD
  RSI_14 → MACD_Hist
  EMA_20 → MACD_Hist
  MACD → BB_Width
  EMA_20 → BB_Width
  MACD_Hist → BB_Width
  BB_Width → ATR
  RSI_14 → ATR
  EMA_9 → ATR
  MACD_Hist → Magnitude_Move_%

Total edges discovered: 12
```

Causal Paths to Target Variable

```python
target = 'Magnitude_Move_%'
all_paths = []

for node in structure.keys():
    if node != target:
        paths = k2.find_all_paths(node, target)
        all_paths.extend(paths)

print(f"\nDiscovered {len(all_paths)} causal paths:\n")
for i, path in enumerate(all_paths, 1):
    print(f"Path {i}: {' → '.join(path)}")

# Direct causes
direct_causes = structure[target]
print(f"\n{'='*70}")
print("KEY FINDING: DIRECT CAUSAL INDICATORS FOR MAGNITUDE MOVES")
print("="*70)
print(f"\nDirect causes of {target}:")
if direct_causes:
    for cause in direct_causes:
        print(f"  ==> {cause}")
else:
    print("  No direct causes found")

# Indirect causes
print(f"\nIndirect influences (through causal chains):")
indirect = set()
for path in all_paths:
    if len(path) > 2:
        indirect.add(path[0])

for cause in indirect:
    print(f"  → {cause}")
```

```
Discovered 4 causal paths:

Path 1: RSI_14 → MACD_Hist → Magnitude_Move_%
Path 2: EMA_9 → EMA_20 → MACD_Hist → Magnitude_Move_%
Path 3: EMA_20 → MACD_Hist → Magnitude_Move_%
Path 4: MACD_Hist → Magnitude_Move_%

======================================================================
KEY FINDING: DIRECT CAUSAL INDICATORS FOR MAGNITUDE MOVES
======================================================================

Direct causes of Magnitude_Move_%:
  ==> MACD_Hist

Indirect influences (through causal chains):
  → EMA_20
  → RSI_14
  → EMA_9
```

Statistical Validation

Validate the discovered causal relationships using statistical tests.

```
# Conditional probabilities for direct causes
if direct_causes:
    print("\nConditional Probability Analysis:")
    print("-" * 70)

    for cause in direct_causes:
        print(f"\nP(Magnitude_Move_% | {cause}):\n")
        cond_prob = analyze_conditional_probabilities(df_discrete, cause, target)
        print(cond_prob.round(2))
        print("\n(Values show percentage probabilities)")
```

```
Conditional Probability Analysis:
----------------------------------------------------------------------

P(Magnitude_Move_% | MACD_Hist):

Magnitude_Move_%  Very Low    Low  Medium   High  Very High
MACD_Hist
Very Low             53.24  21.76   14.35   8.33       2.31
Low                  29.30  27.91   21.40  14.88       6.51
Medium               12.04  29.17   24.54  22.69      11.57
High                  5.12  16.74   23.26  26.98      27.91
Very High             0.46   4.17   16.67  26.85      51.85

(Values show percentage probabilities)
```

```
# Chi-square test for all indicators
indicators = ['RSI_14', 'EMA_9', 'EMA_20', 'MACD', 'MACD_Hist', 'BB_Width', 'ATR']

results = []
for indicator in indicators:
    chi2, p_value, cramers_v = calculate_mutual_information(df_discrete, indicator, target)
    results.append({
        'Indicator': indicator,
        'Chi-square': chi2,
        'P-value': p_value,
        'Cramers_V': cramers_v,
        'Significant': 'Yes' if p_value < 0.05 else 'No',
        'Direct_Cause': 'Yes' if indicator in structure[target] else 'No'
    })

results_df = pd.DataFrame(results).sort_values('Chi-square', ascending=False)
print("\n", results_df.to_string(index=False))
```

```
 Indicator  Chi-square      P-value  Cramers_V Significant Direct_Cause
MACD_Hist  458.458341 1.905034e-87   0.326070         Yes          Yes
   RSI_14  357.250151 3.180838e-66   0.287837         Yes           No
     MACD  123.736159 1.047243e-18   0.169398         Yes           No
 BB_Width  108.745756 7.761857e-16   0.158806         Yes           No
      ATR  100.883707 2.365059e-14   0.152958         Yes           No
   EMA_20   52.656731 8.577403e-06   0.110506         Yes           No
    EMA_9   44.870204 1.452967e-04   0.102009         Yes           No
```

Create Network Adjacency Matrix

```
print("\nBayesian Network Structure (Adjacency Matrix):")

nodes = node_order
n = len(nodes)
adj_matrix = np.zeros((n, n), dtype=int)

node_to_idx = {node: i for i, node in enumerate(nodes)}

for child, parents in structure.items():
    child_idx = node_to_idx[child]
    for parent in parents:
        parent_idx = node_to_idx[parent]
        adj_matrix[parent_idx, child_idx] = 1

adj_df = pd.DataFrame(adj_matrix, index=nodes, columns=nodes)
print("\n", adj_df)
print("\n(1 = causal edge from row to column, 0 = no edge)")
```

```
Bayesian Network Structure (Adjacency Matrix):

                   RSI_14  EMA_9  EMA_20  MACD  MACD_Hist  BB_Width  ATR  \
RSI_14                  0      0       0     1          1         0    1
EMA_9                   0      0       1     0          0         0    1
EMA_20                  0      0       0     1          1         1    0
MACD                    0      0       0     0          0         1    0
MACD_Hist               0      0       0     0          0         1    0
BB_Width                0      0       0     0          0         0    1
ATR                     0      0       0     0          0         0    0
Magnitude_Move_%        0      0       0     0          0         0    0

                  Magnitude_Move_%
RSI_14                           0
EMA_9                            0
EMA_20                           0
MACD                             0
MACD_Hist                        1
BB_Width                         0
ATR                              0
Magnitude_Move_%                 0

(1 = causal edge from row to column, 0 = no edge)
```

How to interpret the results

1. Identifying the "Alpha" Factors

Look at the **"Direct Causes of Magnitude_Move_%"** section above.

- The indicators listed there are the **Primary Drivers**.

- **Action:** When building a trading strategy, these should be our main filter. The algorithm suggests these specific indicators have the most direct influence on price magnitude for this specific asset and timeframe.

2. Deriving Trading Rules

Look at the **"Conditional Probability Analysis"** table.

- This table converts the relationship into probabilities.
- **Example reading:** If `P(Magnitude_Move_% = High | RSI_14 = Low) > 50%`, it indicates a **Mean Reversion** tendency (Low RSI leading to High upward move).
- **Example reading:** If `P(Magnitude_Move_% = High | RSI_14 = High) > 50%`, it indicates a **Momentum** tendency (High RSI leading to continuation).

3. Validating Signal Strength

Look at the **"Chi-square Test"** table.

- **Cramér's V:** This represents the strength of the connection (0 to 1).
  - `V < 0.1` : Weak relationship (Noise).
  - `V > 0.15` : Moderate relationship (Useful).
  - `V > 0.3` : Strong relationship (High Predictive Power).
- **P-value:** If this is greater than 0.05, discard the indicator; the result is likely random chance.