Portfolio Risk Metrics

Duration: 75 minutes
Level: Beginner-Intermediate
Author: Obelisk Core

Learning Objectives

By the end of this module, you will be able to:

- Calculate and interpret key portfolio risk metrics
- Implement Value at Risk (VaR) and Expected Shortfall calculations
- Analyze Sharpe Ratio, Sortino Ratio, and other performance metrics
- Build real-time risk monitoring systems
- Create comprehensive risk dashboards

Introduction to Portfolio Risk Metrics

Portfolio risk metrics provide quantitative measures to assess the risk and performance of your MEV trading strategies. These metrics help you understand how much risk you're taking, how well your strategies are performing, and whether your risk levels are appropriate.

Why Risk Metrics Matter

In MEV trading, risk metrics are essential because:

- **Multiple simultaneous positions**: You often run several strategies simultaneously
- **Variable risk exposure**: Different opportunities have different risk profiles
- **Market regime changes**: Risk characteristics change with market conditions
- **Regulatory requirements**: Many jurisdictions require risk reporting
- **Capital allocation**: Risk metrics guide optimal capital distribution

Types of Risk Metrics

Risk metrics can be categorized into:

1. **Return-based metrics** - Sharpe Ratio, Sortino Ratio, Information Ratio
2. **Drawdown-based metrics** - Maximum Drawdown, Calmar Ratio
3. **Value-based metrics** - Value at Risk (VaR), Expected Shortfall
4. **Distribution metrics** - Standard deviation, skewness, kurtosis

5. **Tail risk metrics** - Maximum loss, tail ratio

Core Performance Risk Metrics

1. Sharpe Ratio

The Sharpe Ratio measures risk-adjusted returns by comparing excess return to volatility.

Formula:

```
Sharpe Ratio = (Portfolio Return - Risk-Free Rate) / Standard
Deviation of Returns
```

Example Implementation:

```
import numpy as np
import pandas as pd

def calculate_sharpe_ratio(returns, risk_free_rate=0.02):
    """
    Calculate Sharpe Ratio for a portfolio

    Parameters:
    returns: Series of daily returns
    risk_free_rate: Annual risk-free rate (default 2%)
    """
    # Convert annual risk-free rate to daily
    daily_risk_free = (1 + risk_free_rate) ** (1/252) - 1

    # Calculate excess returns
    excess_returns = returns - daily_risk_free

    # Calculate annualized Sharpe ratio
    sharpe_ratio = np.mean(excess_returns) / np.std(returns) *
np.sqrt(252)

    return sharpe_ratio

# Example usage with MEV strategy returns
np.random.seed(42)
# Simulate daily returns for an arbitrage strategy
returns = np.random.normal(0.001, 0.015, 252)  # Mean 0.1% daily, 1.5%
volatility

sharpe = calculate_sharpe_ratio(pd.Series(returns))
print(f"Sharpe Ratio: {sharpe:.2f}")

# Interpretation:
# > 2.0: Excellent
# 1.0-2.0: Good
# 0.5-1.0: Acceptable
# < 0.5: Poor
```

Advantages:

- Widely accepted standard
- Easy to calculate and interpret
- Accounts for both risk and return

Disadvantages:

- Assumes normal distribution
- Penalizes upside volatility equally with downside
- Sensitive to return distribution

2. Sortino Ratio

Sortino Ratio improves on Sharpe by only considering downside volatility.

Formula:

```
Sortino Ratio = (Portfolio Return - Target Return) / Downside Deviation
Downside Deviation = sqrt(mean(min(return - target_return, 0)²))
```

Example Implementation:

```
def calculate_sortino_ratio(returns, target_return=0,
risk_free_rate=0.02):
    """
    Calculate Sortino Ratio - only considers downside risk
    """
    # Convert annual risk-free rate to daily
    daily_risk_free = (1 + risk_free_rate) ** (1/252) - 1

    # Calculate excess returns over risk-free rate
    excess_returns = returns - daily_risk_free

    # Calculate downside deviation
    downside_returns = np.minimum(excess_returns - target_return, 0)
    downside_deviation = np.sqrt(np.mean(downside_returns ** 2))

    # Calculate annualized Sortino ratio
    sortino_ratio = np.mean(excess_returns) / downside_deviation *
np.sqrt(252)

    return sortino_ratio

# Example usage
sortino = calculate_sortino_ratio(pd.Series(returns))
print(f"Sortino Ratio: {sortino:.2f}")

# Compare with Sharpe
print(f"Sharpe Ratio: {sharpe:.2f}")
print(f"Sortino is better for asymmetric strategies")
```

Use Cases:

- MEV strategies with asymmetric return distributions
- Strategies with limited downside risk
- Performance evaluation when upside volatility is beneficial

3. Information Ratio

Information Ratio measures excess return per unit of tracking error relative to a benchmark.

Formula:

```
Information Ratio = (Portfolio Return - Benchmark Return) / Tracking
Error
Tracking Error = Standard Deviation of (Portfolio Return - Benchmark
Return)
```

Example Implementation:

```
def calculate_information_ratio(portfolio_returns, benchmark_returns):
    """
    Calculate Information Ratio relative to a benchmark
    """
    # Calculate excess returns over benchmark
    excess_returns = portfolio_returns - benchmark_returns

    # Calculate tracking error
    tracking_error = np.std(excess_returns)

    # Calculate information ratio
    information_ratio = np.mean(excess_returns) / tracking_error *
np.sqrt(252)

    return information_ratio, tracking_error

# Example with MEV strategy vs market index
market_returns = np.random.normal(0.0005, 0.02, 252)  # Market returns
mev_returns = returns  # MEV strategy returns

info_ratio, tracking_error = calculate_information_ratio(
    pd.Series(mev_returns), pd.Series(market_returns)
)
print(f"Information Ratio: {info_ratio:.2f}")
print(f"Tracking Error: {tracking_error:.4f}")
```

Drawdown-Based Metrics

1. Maximum Drawdown

Maximum Drawdown measures the largest peak-to-trough decline in portfolio value.

Example Implementation:

```
def calculate_maximum_drawdown(returns):
    """
    Calculate Maximum Drawdown from a series of returns
    """
    # Calculate cumulative returns
    cumulative_returns = (1 + returns).cumprod()

    # Calculate running maximum (peak)
    running_max = cumulative_returns.expanding().max()

    # Calculate drawdown
    drawdown = (cumulative_returns - running_max) / running_max

    # Find maximum drawdown
    max_drawdown = drawdown.min()

    # Find drawdown duration
    in_drawdown = drawdown < 0
    drawdown_periods = []

    if in_drawdown.any():
        # Find drawdown start and end
        drawdown_start = None
        for i, is_dd in enumerate(in_drawdown):
            if is_dd and drawdown_start is None:
                drawdown_start = i
            elif not is_dd and drawdown_start is not None:
                drawdown_periods.append(i - drawdown_start)
                drawdown_start = None

        # Handle case where drawdown continues to end
        if drawdown_start is not None:
            drawdown_periods.append(len(in_drawdown) - drawdown_start)

    max_dd_duration = max(drawdown_periods) if drawdown_periods else 0

    return max_drawdown, max_dd_duration, drawdown

# Example usage
max_dd, dd_duration, dd_series =
calculate_maximum_drawdown(pd.Series(returns))
print(f"Maximum Drawdown: {max_dd:.2%}")
```

```
print(f"Maximum Drawdown Duration: {dd_duration} days")

# Plot drawdown series
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
cumulative_returns = (1 + pd.Series(returns)).cumprod()
plt.plot(cumulative_returns.values)
plt.title('Portfolio Value')
plt.ylabel('Value')

plt.subplot(2, 1, 2)
plt.fill_between(range(len(dd_series)), dd_series.values, 0, alpha=0.3,
color='red')
plt.title('Drawdown')
plt.ylabel('Drawdown %')
plt.xlabel('Days')
plt.show()
```

2. Calmar Ratio

Calmar Ratio relates returns to maximum drawdown.

Formula:

```
Calmar Ratio = Annualized Return / Maximum Drawdown
```

Example Implementation:

```
def calculate_calmar_ratio(returns, max_drawdown):
    """
    Calculate Calmar Ratio
    """
    annualized_return = (1 + returns.mean()) ** 252 - 1
    calmar_ratio = annualized_return / abs(max_drawdown)

    return calmar_ratio

# Example usage
calmar = calculate_calmar_ratio(pd.Series(returns), max_dd)
print(f"Calmar Ratio: {calmar:.2f}")

# Interpretation:
# > 1.0: Excellent
# 0.5-1.0: Good
# 0.25-0.5: Acceptable
# < 0.25: Poor
```

3. Pain Index

Pain Index measures the average depth and duration of drawdowns.

Formula:

```
Pain Index = Sum of (Drawdown_i × Duration_i) / Total Duration
```

Implementation:

```
def calculate_pain_index(drawdown_series):
    """
    Calculate Pain Index from drawdown series
    """
    in_drawdown = drawdown_series < 0

    if not in_drawdown.any():
        return 0

    # Calculate pain index
    pain_periods = []
    current_drawdown = 0
    duration = 0

    for i, (is_dd, dd) in enumerate(zip(in_drawdown, drawdown_series)):
        if is_dd:
            current_drawdown = min(current_drawdown, dd)
            duration += 1
        else:
            if duration > 0:
                pain_periods.append(abs(current_drawdown) * duration)
            current_drawdown = 0
            duration = 0

    # Handle drawdown that continues to end
    if duration > 0:
        pain_periods.append(abs(current_drawdown) * duration)

    pain_index = sum(pain_periods) / len(drawdown_series) if
pain_periods else 0

    return pain_index

# Example usage
pain_index = calculate_pain_index(dd_series)
print(f"Pain Index: {pain_index:.4f}")
```

Value-Based Risk Metrics

1. Value at Risk (VaR)

VaR estimates the maximum loss over a given time period at a specified confidence level.

Historical VaR

```
def calculate_historical_var(returns, confidence_level=0.05,
time_horizon=1):
    """
    Calculate Historical Value at Risk
    """
    # Sort returns in ascending order
    sorted_returns = np.sort(returns)

    # Calculate VaR index
    var_index = int(confidence_level * len(sorted_returns))

    # Get VaR value (typically negative)
    var = sorted_returns[var_index] * np.sqrt(time_horizon)

    return var

# Example usage
var_95 = calculate_historical_var(returns, confidence_level=0.05)
var_99 = calculate_historical_var(returns, confidence_level=0.01)

print(f"VaR (95%): {var_95:.2%}")
print(f"VaR (99%): {var_99:.2%}")
```

Parametric VaR (Normal Distribution Assumption)

```
def calculate_parametric_var(returns, confidence_level=0.05,
portfolio_value=100000):
    """
    Calculate Parametric VaR assuming normal distribution
    """
    # Calculate parameters
    mean_return = np.mean(returns)
    std_return = np.std(returns)

    # Get z-score for confidence level
    from scipy import stats
    z_score = stats.norm.ppf(confidence_level)

    # Calculate VaR
    var_return = mean_return + z_score * std_return
    var_value = portfolio_value * var_return

    return var_return, var_value

# Example usage
var_return, var_value = calculate_parametric_var(returns,
confidence_level=0.05)
print(f"VaR Return: {var_return:.2%}")
print(f"VaR Value: ${var_value:,.2f}")
```

2. Expected Shortfall (ES) / Conditional VaR

Expected Shortfall measures the average loss beyond the VaR threshold.

Implementation:

```
def calculate_expected_shortfall(returns, confidence_level=0.05):
    """
    Calculate Expected Shortfall (Conditional VaR)
    """
    # Calculate VaR first
    var = calculate_historical_var(returns, confidence_level)

    # Get returns worse than VaR
    tail_returns = returns[returns <= var]

    # Calculate expected shortfall
    expected_shortfall = np.mean(tail_returns)

    return expected_shortfall

# Example usage
es_95 = calculate_expected_shortfall(returns, confidence_level=0.05)
print(f"Expected Shortfall (95%): {es_95:.2%}")

# ES is always worse (more negative) than VaR
print(f"VaR (95%): {var_95:.2%}")
print(f"ES provides tail risk measure beyond VaR")
```

3. Modified VaR (Cornish-Fisher Expansion)

Modified VaR adjusts for non-normal return distributions.

```
def calculate_modified_var(returns, confidence_level=0.05):
    """
    Calculate Modified VaR using Cornish-Fisher expansion
    """
    from scipy import stats

    # Calculate moments
    mean_return = np.mean(returns)
    std_return = np.std(returns)
    skewness = stats.skew(returns)
    excess_kurtosis = stats.kurtosis(returns, fisher=True)

    # Get standard normal quantile
    z = stats.norm.ppf(confidence_level)

    # Cornish-Fisher expansion
    z_cf = (z +
            (z**2 - 1) * skewness / 6 +
            (z**3 - 3*z) * excess_kurtosis / 24 -
            (2*z**3 - 5*z) * skewness**2 / 36)

    # Calculate Modified VaR
    modified_var = mean_return + std_return * z_cf

    return modified_var

# Example usage with non-normal returns
# Create returns with high kurtosis (fat tails)
fat_tail_returns = np.random.standard_t(df=3, size=252) * 0.01
modified_var = calculate_modified_var(fat_tail_returns)
normal_var = calculate_historical_var(fat_tail_returns)

print(f"Normal VaR: {normal_var:.2%}")
print(f"Modified VaR: {modified_var:.2%}")
print(f"Modified VaR accounts for fat tails in return distribution")
```

Real-Time Risk Monitoring System

Portfolio Risk Monitor Class

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import json

class PortfolioRiskMonitor:
    def __init__(self, initial_capital=100000,
max_drawdown_limit=0.15):
        self.initial_capital = initial_capital
        self.max_drawdown_limit = max_drawdown_limit
        self.positions = {}
        self.trade_history = []
        self.risk_metrics = {}
        self.alerts = []

    def add_position(self, symbol, quantity, entry_price,
strategy_type):
        """Add a new position to the portfolio"""
        position_value = quantity * entry_price

        self.positions[symbol] = {
            'quantity': quantity,
            'entry_price': entry_price,
            'current_price': entry_price,  # Initialize with entry
price
            'strategy_type': strategy_type,
            'entry_time': datetime.now(),
            'position_value': position_value,
            'pnl': 0
        }

        return self.get_portfolio_value()

    def update_position_price(self, symbol, new_price):
        """Update current price for a position"""
        if symbol in self.positions:
            old_price = self.positions[symbol]['current_price']
            self.positions[symbol]['current_price'] = new_price

            # Update PnL
            quantity = self.positions[symbol]['quantity']
            self.positions[symbol]['pnl'] = quantity * (new_price -
```

```
old_price)

    def get_portfolio_value(self):
        """Calculate total portfolio value"""
        total_value = sum(pos['current_price'] * pos['quantity']
                          for pos in self.positions.values())
        return total_value

    def calculate_position_weights(self):
        """Calculate current position weights"""
        portfolio_value = self.get_portfolio_value()
        if portfolio_value == 0:
            return {}

        weights = {}
        for symbol, position in self.positions.items():
            weight = (position['current_price'] *
position['quantity']) / portfolio_value
            weights[symbol] = weight

        return weights

    def calculate_risk_metrics(self):
        """Calculate comprehensive risk metrics"""
        if len(self.trade_history) < 2:
            return {}

        returns = pd.Series(self.trade_history)

        # Basic metrics
        total_return = (self.get_portfolio_value() /
self.initial_capital) - 1
        volatility = returns.std() * np.sqrt(252)
        sharpe_ratio = self.calculate_sharpe_ratio(returns)

        # Drawdown metrics
        max_dd, dd_duration, _ =
self.calculate_maximum_drawdown(returns)
        calmar_ratio = (returns.mean() * 252) / abs(max_dd) if max_dd !
= 0 else 0

        # VaR metrics
```

```
        var_95 = self.calculate_historical_var(returns, 0.05)
        var_99 = self.calculate_historical_var(returns, 0.01)
        expected_shortfall = self.calculate_expected_shortfall(returns,
0.05)

        # Risk concentration
        weights = self.calculate_position_weights()
        concentration_risk = max(weights.values()) if weights else 0

        # Correlation risk (simplified)
        correlation_risk = self.calculate_correlation_risk()

        return {
            'total_return': total_return,
            'volatility': volatility,
            'sharpe_ratio': sharpe_ratio,
            'max_drawdown': max_dd,
            'max_drawdown_duration': dd_duration,
            'calmar_ratio': calmar_ratio,
            'var_95': var_95,
            'var_99': var_99,
            'expected_shortfall': expected_shortfall,
            'concentration_risk': concentration_risk,
            'correlation_risk': correlation_risk,
            'portfolio_value': self.get_portfolio_value(),
            'number_of_positions': len(self.positions),
            'positions_weights': weights
        }

    def calculate_sharpe_ratio(self, returns, risk_free_rate=0.02):
        """Calculate Sharpe ratio"""
        if len(returns) == 0:
            return 0
        daily_rf = (1 + risk_free_rate) ** (1/252) - 1
        excess_returns = returns - daily_rf
        return (excess_returns.mean() / returns.std()) * np.sqrt(252)
if returns.std() > 0 else 0

    def calculate_historical_var(self, returns, confidence_level=0.05):
        """Calculate historical VaR"""
        if len(returns) == 0:
            return 0
```

```
        sorted_returns = np.sort(returns)
        var_index = int(confidence_level * len(sorted_returns))
        return sorted_returns[var_index] if var_index <
len(sorted_returns) else sorted_returns[-1]

    def calculate_expected_shortfall(self, returns,
confidence_level=0.05):
        """Calculate Expected Shortfall"""
        if len(returns) == 0:
            return 0
        var = self.calculate_historical_var(returns, confidence_level)
        tail_returns = returns[returns <= var]
        return tail_returns.mean() if len(tail_returns) > 0 else var

    def calculate_maximum_drawdown(self, returns):
        """Calculate maximum drawdown"""
        if len(returns) == 0:
            return 0, 0, pd.Series([])

        cumulative_returns = (1 + returns).cumprod()
        running_max = cumulative_returns.expanding().max()
        drawdown = (cumulative_returns - running_max) / running_max

        max_drawdown = drawdown.min()

        # Calculate drawdown duration
        in_drawdown = drawdown < 0
        max_duration = 0
        current_duration = 0

        for is_dd in in_drawdown:
            if is_dd:
                current_duration += 1
                max_duration = max(max_duration, current_duration)
            else:
                current_duration = 0

        return max_drawdown, max_duration, drawdown

    def calculate_correlation_risk(self):
        """Calculate portfolio correlation risk (simplified)"""
        if len(self.positions) < 2:
```

```
            return 0

        # Get recent returns for all positions (simplified - normally
you'd have price history)
        strategy_types = [pos['strategy_type'] for pos in
self.positions.values()]
        unique_strategies = list(set(strategy_types))

        # Simplified correlation estimate based on strategy types
        if len(unique_strategies) == 1:
            return 0.8  # High correlation for single strategy
        elif len(unique_strategies) == 2:
            return 0.5  # Medium correlation
        else:
            return 0.3  # Low correlation for diverse strategies

    def check_risk_alerts(self):
        """Check for risk limit breaches and generate alerts"""
        metrics = self.calculate_risk_metrics()
        new_alerts = []

        # Check drawdown limit
        if metrics.get('max_drawdown', 0) < -self.max_drawdown_limit:
            new_alerts.append({
                'timestamp': datetime.now(),
                'type': 'DRAWDOWN_LIMIT',
                'message': f"Maximum drawdown
{metrics['max_drawdown']:.1%} exceeds limit {self.max_drawdown_limit:.
1%}",
                'severity': 'HIGH'
            })

        # Check concentration risk
        if metrics.get('concentration_risk', 0) > 0.5:
            new_alerts.append({
                'timestamp': datetime.now(),
                'type': 'CONCENTRATION_RISK',
                'message': f"Position concentration
{metrics['concentration_risk']:.1%} exceeds 50%",
                'severity': 'MEDIUM'
            })
```

```
        # Check correlation risk
        if metrics.get('correlation_risk', 0) > 0.7:
            new_alerts.append({
                'timestamp': datetime.now(),
                'type': 'CORRELATION_RISK',
                'message': f"High correlation risk detected:
{metrics['correlation_risk']:.1%}",
                'severity': 'MEDIUM'
            })

        # Check VaR limit (simplified)
        if metrics.get('var_95', 0) < -0.1:  # 10% daily VaR
            new_alerts.append({
                'timestamp': datetime.now(),
                'type': 'VAR_LIMIT',
                'message': f"Daily VaR {metrics['var_95']:.1%} exceeds
acceptable limit",
                'severity': 'HIGH'
            })

        self.alerts.extend(new_alerts)
        return new_alerts

    def generate_risk_report(self):
        """Generate comprehensive risk report"""
        metrics = self.calculate_risk_metrics()

        report = {
            'timestamp': datetime.now().isoformat(),
            'portfolio_summary': {
                'total_value': self.get_portfolio_value(),
                'total_return': f"{metrics.get('total_return', 0):.
2%}",
                'number_of_positions':
metrics.get('number_of_positions', 0)
            },
            'risk_metrics': {
                'sharpe_ratio': f"{metrics.get('sharpe_ratio', 0):.
2f}",
                'max_drawdown': f"{metrics.get('max_drawdown', 0):.
2%}",
                'volatility': f"{metrics.get('volatility', 0):.2%}",
```

```
                'var_95': f"{metrics.get('var_95', 0):.2%}",
                'calmar_ratio': f"{metrics.get('calmar_ratio', 0):.2f}"
            },
            'risk_alerts': len([a for a in self.alerts if a['severity']
in ['HIGH', 'MEDIUM']]),
            'position_concentration':
f"{metrics.get('concentration_risk', 0):.2%}",
            'correlation_risk': f"{metrics.get('correlation_risk', 0):.
2%}"
        }

        return report

# Example usage
monitor = PortfolioRiskMonitor(initial_capital=100000)

# Add some positions
monitor.add_position('ETH/USD', 10, 2000, 'arbitrage')
monitor.add_position('BTC/USD', 2, 40000, 'liquidation')

# Simulate some price movements and trades
for day in range(30):
    # Simulate daily P&L
    daily_pnl = np.random.normal(0.001, 0.02) *
monitor.get_portfolio_value()
    monitor.trade_history.append(daily_pnl)

    # Update prices
    monitor.update_position_price('ETH/USD', 2000 + np.random.normal(0,
50))
    monitor.update_position_price('BTC/USD', 40000 +
np.random.normal(0, 1000))

    # Check for alerts
    alerts = monitor.check_risk_alerts()
    if alerts:
        print(f"Day {day + 1}: {len(alerts)} new alerts")

# Generate risk report
report = monitor.generate_risk_report()
print("\nRisk Report:")
for category, metrics in report.items():
```

```
    if category != 'timestamp':
        print(f"\n{category.upper()}:")
        if isinstance(metrics, dict):
            for key, value in metrics.items():
                print(f"  {key}: {value}")
        else:
            print(f"  {metrics}")
```

Risk Dashboard Implementation

HTML/CSS Risk Dashboard

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Portfolio Risk Dashboard</title>
    <style>
        .dashboard {
            display: grid;
            grid-template-columns: repeat(auto-fit, minmax(300px,
1fr));
            gap: 20px;
            padding: 20px;
        }

        .metric-card {
            background: white;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.1);
        }

        .metric-value {
            font-size: 2em;
            font-weight: bold;
            margin: 10px 0;
        }

        .risk-low { color: #27ae60; }
        .risk-medium { color: #f39c12; }
        .risk-high { color: #e74c3c; }

        .alert-high { background: #e74c3c; color: white; }
        .alert-medium { background: #f39c12; color: white; }
        .alert-low { background: #27ae60; color: white; }

        .progress-bar {
            width: 100%;
            height: 10px;
            background: #ecf0f1;
            border-radius: 5px;
```

```
            overflow: hidden;
            margin: 10px 0;
        }

        .progress-fill {
            height: 100%;
            transition: width 0.3s ease;
        }

        .gauge {
            width: 100px;
            height: 100px;
            border-radius: 50%;
            display: flex;
            align-items: center;
            justify-content: center;
            margin: 0 auto;
            position: relative;
        }
    </style>
</head>
<body>
    <div class="dashboard">
        <!-- Portfolio Value -->
        <div class="metric-card">
            <h3>Portfolio Value</h3>
            <div class="metric-value" id="portfolioValue">$105,432</
div>
            <div class="progress-bar">
                <div class="progress-fill risk-low" style="width:
75%"></div>
            </div>
            <div>Total Return: <span class="risk-low">+5.43%</span></
div>
        </div>

        <!-- Sharpe Ratio -->
        <div class="metric-card">
            <h3>Sharpe Ratio</h3>
            <div class="metric-value risk-low" id="sharpeRatio">1.85</
div>
            <div>Risk-Adjusted Returns</div>
```

```
            <div class="progress-bar">
                <div class="progress-fill risk-low" style="width:
92%"></div>
            </div>
        </div>

        <!-- Maximum Drawdown -->
        <div class="metric-card">
            <h3>Max Drawdown</h3>
            <div class="metric-value risk-medium"
id="maxDrawdown">-3.2%</div>
            <div>Peak to Trough Decline</div>
            <div class="progress-bar">
                <div class="progress-fill risk-medium" style="width:
68%"></div>
            </div>
        </div>

        <!-- VaR (95%) -->
        <div class="metric-card">
            <h3>Value at Risk (95%)</h3>
            <div class="metric-value risk-medium" id="var95">-$1,234</
div>
            <div>Daily Loss Limit</div>
            <div class="progress-bar">
                <div class="progress-fill risk-medium" style="width:
45%"></div>
            </div>
        </div>

        <!-- Position Concentration -->
        <div class="metric-card">
            <h3>Position Concentration</h3>
            <div class="metric-value risk-low" id="concentration">35%</
div>
            <div>Largest Position</div>
            <div class="progress-bar">
                <div class="progress-fill risk-low" style="width:
35%"></div>
            </div>
        </div>
```

```
        <!-- Active Alerts -->
        <div class="metric-card">
            <h3>Risk Alerts</h3>
            <div class="metric-value risk-low" id="alertCount">2</div>
            <div>Active Monitoring</div>
            <div class="alert-high"
style="padding: 5px; margin: 5px 0; border-radius: 3px;">1 High
Priority</div>
            <div class="alert-medium" style="padding: 5px; margin: 5px
0; border-radius: 3px;">1 Medium Priority</div>
        </div>
    </div>

    <script>
        // Simulated real-time updates
        function updateMetrics() {
            // Simulate API call to get updated metrics
            const metrics = {
                portfolioValue: 105432 + Math.random() * 1000 - 500,
                sharpeRatio: 1.85 + (Math.random() - 0.5) * 0.1,
                maxDrawdown: -3.2 + (Math.random() - 0.5) * 0.5,
                var95: -1234 + Math.random() * 100 - 50,
                concentration: 35 + Math.random() * 5 - 2.5,
                alertCount: Math.floor(Math.random() * 3)
            };

            // Update display
            document.getElementById('portfolioValue').textContent =
                `$${metrics.portfolioValue.toLocaleString()}`;
            document.getElementById('sharpeRatio').textContent =
                metrics.sharpeRatio.toFixed(2);
            document.getElementById('maxDrawdown').textContent =
                `${metrics.maxDrawdown.toFixed(1)}%`;
            document.getElementById('var95').textContent =
                `$${metrics.var95.toFixed(0)}`;
            document.getElementById('concentration').textContent =
                `${metrics.concentration.toFixed(0)}%`;
            document.getElementById('alertCount').textContent =
                metrics.alertCount;
        }

        // Update every 5 seconds
```

```
        setInterval(updateMetrics, 5000);

        // Initial update
        updateMetrics();
    </script>
</body>
</html>
```

Advanced Risk Analytics

Monte Carlo VaR Simulation

```
def monte_carlo_var(returns, confidence_level=0.05, simulations=10000):
    """
    Calculate VaR using Monte Carlo simulation
    """
    # Calculate return statistics
    mean_return = np.mean(returns)
    std_return = np.std(returns)
    skewness = stats.skew(returns)
    excess_kurtosis = stats.kurtosis(returns, fisher=True)

    # Generate random scenarios
    scenarios = np.random.normal(mean_return, std_return, simulations)

    # Apply Cornish-Fisher adjustment for non-normality
    z_scores = (scenarios - mean_return) / std_return
    z_cf = (z_scores +
            (z_scores**2 - 1) * skewness / 6 +
            (z_scores**3 - 3*z_scores) * excess_kurtosis / 24)

    adjusted_scenarios = mean_return + std_return * z_cf

    # Calculate VaR
    var_index = int(confidence_level * simulations)
    sorted_scenarios = np.sort(adjusted_scenarios)
    var = sorted_scenarios[var_index]

    return var

# Example usage with real return data
mc_var = monte_carlo_var(returns, confidence_level=0.05)
hist_var = calculate_historical_var(returns, confidence_level=0.05)

print(f"Historical VaR: {hist_var:.2%}")
print(f"Monte Carlo VaR: {mc_var:.2%}")
```

Dynamic Risk Model

```
class DynamicRiskModel:
    def __init__(self, lookback_window=252):
        self.lookback_window = lookback_window
        self.volatility_regimes = ['low', 'normal', 'high']
        self.current_regime = 'normal'

    def detect_volatility_regime(self, returns):
        """Detect current volatility regime"""
        if len(returns) < self.lookback_window:
            return 'insufficient_data'

        recent_vol = returns.tail(self.lookback_window).std()
        historical_vol = returns.std()

        # Define thresholds (these should be calibrated to your
strategy)
        if recent_vol < historical_vol * 0.7:
            return 'low'
        elif recent_vol > historical_vol * 1.5:
            return 'high'
        else:
            return 'normal'

    def adjust_var_limits(self, base_var_limit, returns):
        """Adjust VaR limits based on current volatility regime"""
        regime = self.detect_volatility_regime(returns)

        adjustments = {
            'low': 0.8,      # Reduce limits in low volatility
            'normal': 1.0,   # Normal limits
            'high': 1.5      # Increase limits in high volatility
        }

        return base_var_limit * adjustments.get(regime, 1.0)

    def stress_test_portfolio(self, portfolio_returns,
shock_scenarios):
        """Perform stress tests on portfolio"""
        stress_results = {}

        for scenario_name, shock_params in shock_scenarios.items():
            shocked_returns = portfolio_returns.copy()
```

```
            # Apply shock to returns
            if 'volatility_multiplier' in shock_params:
                shocked_returns *=
shock_params['volatility_multiplier']

            if 'mean_shift' in shock_params:
                shocked_returns += shock_params['mean_shift']

            # Calculate stressed VaR
            stressed_var = np.percentile(shocked_returns, 5)

            stress_results[scenario_name] = {
                'var_95': stressed_var,
                'expected_shortfall':
np.mean(shocked_returns[shocked_returns <= stressed_var])
            }

        return stress_results

# Example usage
stress_scenarios = {
    'crypto_crash': {
        'volatility_multiplier': 3.0,
        'mean_shift': -0.05
    },
    'high_inflation': {
        'volatility_multiplier': 2.0,
        'mean_shift': -0.02
    },
    'liquidity_crisis': {
        'volatility_multiplier': 2.5,
        'mean_shift': -0.03
    }
}

dynamic_model = DynamicRiskModel()
stress_results = dynamic_model.stress_test_portfolio(returns,
stress_scenarios)

print("Stress Test Results:")
for scenario, results in stress_results.items():
```

```
    print(f"{scenario}: VaR = {results['var_95']:.2%}, ES =
{results['expected_shortfall']:.2%}")
```

Key Takeaways

1. **Multiple metrics provide better insight** - No single risk metric tells the complete story.
2. **Context matters** - Risk metrics should be interpreted in the context of your strategy and market conditions.
3. **Real-time monitoring is crucial** - Risk can change quickly in MEV markets.
4. **Tail risk is often underestimated** - Use Expected Shortfall alongside VaR.
5. **Diversification reduces correlation risk** - Monitor correlation between positions and strategies.
6. **Historical data may not predict future risk** - Include stress testing and scenario analysis.
7. **Risk limits should be dynamic** - Adjust limits based on market conditions and strategy performance.
8. **Automated monitoring prevents human error** - Use systems to monitor risk continuously.

Next Steps

In the next module, **Stop-Loss & Take-Profit Systems**, you'll learn to:

- Implement various stop-loss mechanisms
- Build automated take-profit systems
- Create trailing stops and dynamic exits
- Integrate risk management with position sizing
- Build emergency exit procedures

This will complete your foundational understanding of risk management before moving to advanced portfolio construction techniques.

Summary

Portfolio risk metrics provide the quantitative foundation for understanding and managing the risk in your MEV trading operations. By implementing comprehensive monitoring systems that track Sharpe ratios, drawdowns, VaR, and other key metrics, you can maintain optimal risk levels while maximizing returns.

Remember that risk management is not just about avoiding losses—it's about understanding the risk-return tradeoff and making informed decisions based on quantitative analysis. The most successful MEV traders are those who treat risk management as a core competency, not an afterthought.