# trade_mcsim

October 31, 2024

## 1 Monte Carlo Trade Simulation

[192]:
```
#import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from itertools import groupby
```

I only included the original data in what I loaded, then I reduced it down to only the original trade results. Here are the first few rows so you can see what I used.

[201]:
```
# Load the Trade Data
file_path = "C:/Users/earne/OneDrive/Upwork/trade_monte_carlo/data/" # Enter␣
 ↪your file path here
file_name = "trade_data.csv" #enter name of csv file
trade_results = pd.read_csv(file_path + file_name)
trade_results.head(5)
```

[201]:

| | trade | cum_return | capital | %_drawdown_from_max | losses |
|---|---|---|---|---|---|
| 0 | -1.00 | -1.00 | 99,000.00 | -0.01 | 1.00 |
| 1 | 3.41 | 2.41 | 102,375.90 | 0.00 | NaN |
| 2 | -1.00 | 1.41 | 101,352.14 | -0.01 | 1.00 |
| 3 | 4.85 | 6.26 | 106,267.72 | 0.00 | NaN |
| 4 | -1.00 | 5.26 | 105,205.04 | -0.01 | 1.00 |

Number of rows and columns

[202]:
```
trade_results.shape
```

[202]:
```
(156, 5)
```

[203]:
```
#fill the NaN values with 0 in "losses" column
trade_results['losses'] = trade_results['losses'].fillna(0)
# Only use the trade results column:
trade_results = trade_results['trade']
trade_results.head(5)
```

[203]:
```
0   -1.00
1    3.41
```

```
2   -1.00
3    4.85
4   -1.00
Name: trade, dtype: float64
```

Here I used 1000 random trades sampled from the original 156, then repeated that simulation 1000 times.

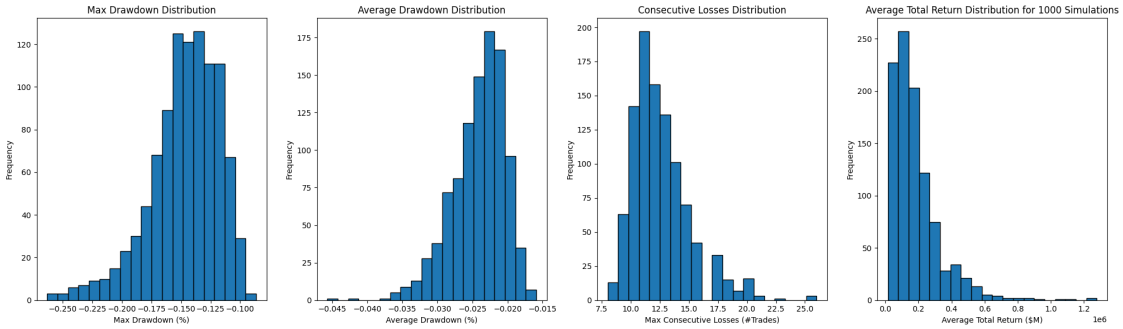
[195]:
```
# Parameters for the simulation
num_simulations = 1000
num_trades_per_simulation = 1000
initial_capital = 100000

# Function to handle zero and negative values in cumulative return calculation
# Function to calculate drawdowns and consecutive losses

def calculate_drawdowns(trades):
    cumulative_returns = np.cumprod(1 + trades / 100)  # Convert percentage to␣
 ↪decimal
    peak = np.maximum.accumulate(cumulative_returns)
    drawdowns = np.where(peak != 0, (cumulative_returns - peak) / peak, 0)
    max_drawdown = np.min(drawdowns)

    # Calculate consecutive losses
    losses = (trades < 0).astype(int)  # 1 if loss, 0 otherwise
    max_consecutive_losses = max([sum(1 for _ in group) for key, group in␣
 ↪groupby(losses) if key == 1], default=0)

    return max_drawdown, np.mean(drawdowns), max_consecutive_losses
```

[196]:
```
# Run the Monte Carlo Simulation
max_drawdowns = []
average_drawdowns = []
consecutive_losses = []
average_total_returns = []

for _ in range(num_simulations):
    sampled_trades = np.random.choice(trade_results, num_trades_per_simulation,␣
 ↪replace=True)
    max_dd, avg_dd, max_loss_streak = calculate_drawdowns(sampled_trades)
    max_drawdowns.append(max_dd)
    average_drawdowns.append(avg_dd)
    consecutive_losses.append(max_loss_streak)

     # Calculate total return starting from initial capital
    capital = initial_capital
    for trade in sampled_trades:
```

```
        capital *= (1 + trade / 100)  # Convert percent return to decimal and␣
↪apply to capital

    # Calculate average return for this simulation
    total_return = (capital - initial_capital) / initial_capital * 100  # Total␣
↪return in percentage
    average_total_returns.append(total_return)
```

Distributions of some results

[197]:
```
# Plot histogram of results
plt.figure(figsize=(20, 6))

# Max Drawdown
plt.subplot(1, 4, 1)
plt.hist(max_drawdowns, bins=20, edgecolor='black')
plt.title("Max Drawdown Distribution")
plt.xlabel("Max Drawdown (%)")
plt.ylabel("Frequency")

# Average Drawdown
plt.subplot(1, 4, 2)
plt.hist(average_drawdowns, bins=20, edgecolor='black')
plt.title("Average Drawdown Distribution")
plt.xlabel("Average Drawdown (%)")
plt.ylabel("Frequency")

# Consecutive Losses
plt.subplot(1, 4, 3)
plt.hist(consecutive_losses, bins=20, edgecolor='black')
plt.title("Consecutive Losses Distribution")
plt.xlabel("Max Consecutive Losses (#Trades)")
plt.ylabel("Frequency")

# Average returns for each 1000 trade simulation
plt.subplot(1, 4, 4)
plt.hist(average_total_returns, bins=20, edgecolor='black')
plt.title("Average Total Return Distribution for 1000 Simulations")
plt.xlabel("Average Total Return ($M)")
plt.ylabel("Frequency")

plt.tight_layout()
plt.show()
```




Here are the requested summary statistics. Since we are simulating 1000 trades instead of 156, the strategy results in higher returns over time (or more trades).

[208]:
```
# Set global formatting option for pandas
pd.options.display.float_format = '{:,.2f}'.format

months = 20

# Simplified code for creating the summary statistics DataFrame
average_total_returns_mean = np.mean(average_total_returns) + initial_capital
average_total_returns_sd = np.std(average_total_returns) + initial_capital
average_total_returns_max = np.max(average_total_returns) + initial_capital
average_total_returns_min = np.min(average_total_returns) + initial_capital

gain_percent = [(value - initial_capital) / initial_capital * 100 for value in
                [average_total_returns_mean, average_total_returns_max,␣
 ↪average_total_returns_min]]
gain_percent_sd = (average_total_returns_sd / initial_capital) * 100  # SD for␣
 ↪gain percent

ret_per_month = [gain / months for gain in gain_percent]
ret_per_month_sd = gain_percent_sd / months  # SD for monthly return

ret_dd_ratio = [gain / (dd * 100) * -1 for gain, dd in zip(gain_percent, [np.
 ↪mean(max_drawdowns), np.max(max_drawdowns), np.min(max_drawdowns)])]

# Create the DataFrame with rows in the correct order
summary_stats_df = pd.DataFrame({
    'Total return': [average_total_returns_mean, np.nan,␣
 ↪average_total_returns_max, average_total_returns_min],
    'Gain (%)': [gain_percent[0], gain_percent_sd, gain_percent[1],␣
 ↪gain_percent[2]],
    'Ret. per month (%)': [ret_per_month[0], ret_per_month_sd,␣
 ↪ret_per_month[1], ret_per_month[2]],
```

```
    'Max DD (%)': [np.mean(max_drawdowns) * 100, np.std(max_drawdowns), np.
 ↪max(max_drawdowns) * 100, np.min(max_drawdowns) * 100],
    'Average dd (%)': [np.mean(average_drawdowns) * 100, np.
 ↪std(average_drawdowns), np.max(average_drawdowns) * 100, np.
 ↪min(average_drawdowns) * 100],
    'Consec_loss': [np.mean(consecutive_losses), np.nan, np.
 ↪min(consecutive_losses), np.max(consecutive_losses)],
    'Ret.//dd.': [ret_dd_ratio[0], np.nan, ret_dd_ratio[1], ret_dd_ratio[2]]
}, index=['Average', 'SD', 'Best', 'Worst'])

summary_stats_df = summary_stats_df.fillna('')

# Display the DataFrame
summary_stats_df
```

[208]:

| | Total return | Gain (%) | Ret. per month (%) | Max DD (%) | \ |
|---|---|---|---|---|---|
| Average | 282,229.51 | 182.23 | 9.11 | -14.61 | |
| SD | | 249.91 | 12.50 | 0.03 | |
| Best | 1,377,379.43 | 1,277.38 | 63.87 | -8.65 | |
| Worst | 115,541.48 | 15.54 | 0.78 | -26.33 | |

| | Average dd (%) | Consec_loss | Ret.//dd. |
|---|---|---|---|
| Average | -2.41 | 12.51 | 12.48 |
| SD | 0.00 | | |
| Best | -1.59 | 8.00 | 147.70 |
| Worst | -4.57 | 26.00 | 0.59 |

Adding the median return for context

[207]:
```
median_return = {'Median Return' : np.median(average_total_returns) +␣
 ↪initial_capital}

median_return
```

[207]:
```
{'Median Return': 245196.73020187137}
```