Analyze market depth for the Ekubo Protocol (v3)

In [1]:
```
#extracting data from parquet and converting the data to a data frame

import pyarrow.parquet as pq
import pandas as pd

# Load Parquet file using PyArrow
parquet_file = r'C:\Users\*******\Documents\OpenBLockLabs\data.parquet'
table = pq.read_table(parquet_file)

# Convert PyArrow Table to Pandas DataFrame
df = table.to_pandas()
```

Displaying data frame, df

In [2]:
```
df
```

Out[2]:

	BLOCK_NUMBER	BLOCK_TIMESTAMP	
0	328311	2023-10-16 15:58:15	0x068ae08b780c44b6d5674540a49bbda991
1	328253	2023-10-16 15:05:36	0x00bdc788d3ce6493eb7adefba441fe4e04
2	328269	2023-10-16 15:23:34	0x03b06aff66afbc79d2e5c2a48919d4c773
3	328545	2023-10-16 20:56:28	0x034398a7788f3a78c60956da2d128c56b2
4	328205	2023-10-16 13:42:10	0x051ca2db4564744fc6de3545e34e2f623a
...	...	...	
3024736	492219	2024-01-02 19:26:50	0x001af375dff0f8af7d64e892fc33559e22
3024737	471827	2023-12-17 09:39:57	0x0716db0afcbe45ba267725a9847c336f3c
3024738	433742	2023-11-26 06:49:25	0x052148716b31563c003777d41ccb7f5861e
3024739	435432	2023-11-26 12:28:29	0x05c58ef329b425eddffa58045a95495ab3
3024740	467117	2023-12-12 10:35:32	0x07ab0f0feb6a31a4601c37ed1f46c3de9a

3024741 rows × 22 columns

In [3]:
```
# printing the types of data this table contains
print(df.dtypes)
```

```
BLOCK_NUMBER                object
BLOCK_TIMESTAMP      datetime64[ns]
TX_HASH                     object
TX_ID                       object
POOL_ID                     object
TOKEN0_ADDRESS              object
TOKEN1_ADDRESS              object
EVENT_NAME                  object
FROM_ADDRESS                object
TO_ADDRESS                  object
TOKEN0_RAW_AMOUNT           object
TOKEN0_DECIMALS             object
TOKEN0_REAL_AMOUNT          object
TOKEN1_RAW_AMOUNT           object
TOKEN1_DECIMALS             object
TOKEN1_REAL_AMOUNT          object
FEE_TIER                   float32
LIQUIDITY_AMOUNT            object
LOWER_TICK                  object
UPPER_TICK                  object
SWAP_TICK                   object
TICK_SPACING                object
dtype: object
```

In [4]:
```
#Displaying all the columns to understand the data frame better
pd.set_option('display.max_columns', None)
df
```

Out[4]:

	BLOCK_NUMBER	BLOCK_TIMESTAMP	
0	328311	2023-10-16 15:58:15	0x068ae08b780c44b6d5674540a49bbda991
1	328253	2023-10-16 15:05:36	0x00bdc788d3ce6493eb7adefba441fe4e04
2	328269	2023-10-16 15:23:34	0x03b06aff66afbc79d2e5c2a48919d4c773
3	328545	2023-10-16 20:56:28	0x034398a7788f3a78c60956da2d128c56b2
4	328205	2023-10-16 13:42:10	0x051ca2db4564744fc6de3545e34e2f623a
...	...	...	
3024736	492219	2024-01-02 19:26:50	0x001af375dff0f8af7d64e892fc33559e22
3024737	471827	2023-12-17 09:39:57	0x0716db0afcbe45ba267725a9847c336f3c
3024738	433742	2023-11-26 06:49:25	0x052148716b31563c003777d41ccb7f5861e
3024739	435432	2023-11-26 12:28:29	0x05c58ef329b425eddffa58045a95495ab3
3024740	467117	2023-12-12 10:35:32	0x07ab0f0feb6a31a4601c37ed1f46c3de9al

3024741 rows × 22 columns

EKUBO PROTOCOL

In [ ]:
```
```

STRK/ETH

In [7]:
```
strk_eth_data = df.copy()  # Make a copy of the DataFrame
```

```
# Define current price of STRK to ETH form CoinGecko
current_price_strk_to_eth = 0.00061052709

# Define depth (e.g., 10%)
depth = 0.1
strk_eth_data['TOKEN0_REAL_AMOUNT'] = strk_eth_data['TOKEN0_REAL_AMOUNT'].astype
strk_eth_data['TOKEN1_REAL_AMOUNT'] = strk_eth_data['TOKEN1_REAL_AMOUNT'].astype

# Convert TOKEN0_REAL_AMOUNT to ETH0 and TOKEN1_REAL_AMOUNT to ETH1
strk_eth_data['ETH0'] = strk_eth_data['TOKEN0_REAL_AMOUNT'] * current_price_strk
strk_eth_data['ETH1'] = strk_eth_data['TOKEN1_REAL_AMOUNT'] * current_price_strk

# Calculate lower and upper bounds of the price range
lower_bound = current_price_strk_to_eth / (1.000001 + depth)
upper_bound = current_price_strk_to_eth * (1.000001 + depth)

# Filter data within the specified price range for 'ETH0'
tokens_within_range_eth0 = strk_eth_data[(strk_eth_data['ETH0'] >= lower_bound)

# Filter data within the specified price range for 'ETH1'
tokens_within_range_eth1 = strk_eth_data[(strk_eth_data['ETH1'] >= lower_bound)

# tokens_within_range is the DataFrame where we want to store the filtered data
tokens_within_range = pd.DataFrame({
    'ETH0': tokens_within_range_eth0['ETH0'],
    'ETH1': tokens_within_range_eth1['ETH1']
})
```

In [8]:
```
market_depth0 = tokens_within_range['ETH0'].sum()
market_depth1 = tokens_within_range['ETH1'].sum()


print("Market depth0 in terms of ETH:", market_depth0)
print("Market depth1 in terms of ETH:", market_depth1)
```

```
Market depth0 in terms of ETH: 58.91110288554696
Market depth1 in terms of ETH: 11.428490385065668
```

In [ ]:
```

```

STRK/USDC

In [11]:
```
strk_USDC_data = df.copy()  # Make a copy of the DataFrame

# Define current price of STRK to USDC
current_price_strk_to_USDC = 2.12

# Define depth (e.g., 10%)
depth = 0.1
# Changing type to float due to TypeError for unsupported operands
strk_USDC_data['TOKEN0_REAL_AMOUNT'] = strk_USDC_data['TOKEN0_REAL_AMOUNT'].asty
strk_USDC_data['TOKEN1_REAL_AMOUNT'] = strk_USDC_data['TOKEN1_REAL_AMOUNT'].asty

# Convert TOKEN0_REAL_AMOUNT to USDC0 and TOKEN1_REAL_AMOUNT to USDC1
strk_USDC_data['USDC0'] = strk_USDC_data['TOKEN0_REAL_AMOUNT'] * current_price_s
strk_USDC_data['USDC1'] = strk_USDC_data['TOKEN1_REAL_AMOUNT'] * current_price_s

# Calculate lower and upper bounds of the price range
lower_bound = current_price_strk_to_USDC / (1.000001 + depth)
upper_bound = current_price_strk_to_USDC * (1.000001 + depth)
```

```
# Filter data within the specified price range for 'USDC'
tokens_within_range_USDC0 = strk_USDC_data[(strk_USDC_data['USDC0'] >= lower_bou

# Filter data within the specified price range for 'USDC1'
tokens_within_range_USDC1 = strk_USDC_data[(strk_USDC_data['USDC1'] >= lower_bou

# tokens_within_range is the DataFrame where we want to store the filtered data
tokens_within_range['USDC0'] = tokens_within_range_USDC0['USDC0']
tokens_within_range['USDC1'] = tokens_within_range_USDC1['USDC1']
```

In [12]:
```
market_depth_USDC0 = tokens_within_range['USDC0'].sum()
market_depth_USDC1 = tokens_within_range['USDC1'].sum()


print("Market depth0 in terms of USDC:", market_depth_USDC0)
print("Market depth1 in terms of USDC:", market_depth_USDC1)
```

```
Market depth0 in terms of USDC: 204563.466819072
Market depth1 in terms of USDC: 39684.39732353108
```

In [ ]:
```

```

ETH/USDC

In [17]:
```
strk_ETH_to_USDC_data = df.copy()  # Make a copy of the DataFrame

# Define current price of ETH to USDC
current_price_ETH_to_USDC = 3469.66

# Define depth (e.g., 10%)
depth = 0.1

# Convert ETH0 to USDC0 and ETH1 to USDC1
strk_ETH_to_USDC_data['USEH0'] = strk_eth_data['ETH0'] * current_price_ETH_to_US
strk_ETH_to_USDC_data['USEH1'] = strk_eth_data['ETH1'] * current_price_ETH_to_US

# Calculate lower and upper bounds of the price range
lower_bound = current_price_ETH_to_USDC / (1.000001 + depth)
upper_bound = current_price_ETH_to_USDC * (1.000001 + depth)

# Filter data within the specified price range for 'USDC0'
tokens_within_range_USDC_ETH0 = strk_ETH_to_USDC_data[(strk_ETH_to_USDC_data['US

# Filter data within the specified price range for 'USDC1'
tokens_within_range_USDC_ETH1 = strk_ETH_to_USDC_data[(strk_ETH_to_USDC_data['US

# tokens_within_range is the DataFrame where we want to store the filtered data
tokens_within_range['USEH0'] = tokens_within_range_USDC_ETH0['USEH0']
tokens_within_range['USEH1'] = tokens_within_range_USDC_ETH1['USEH1']
```

In [19]:
```
market_depth_USDC_ETH0 = tokens_within_range['USEH0'].sum()
market_depth_USDC_ETH1 = tokens_within_range['USEH1'].sum()


print("Market depth0 in terms of ETH to USDC:", market_depth_USDC_ETH0)
print("Market depth1 in terms of ETH to USDC:", market_depth_USDC_ETH1)
```

```
Market depth0 in terms of ETH to USDC: 3661.070041166157
Market depth1 in terms of ETH to USDC: 38786.16246178437
```

In [ ]:

USDC/USDT

In [20]:
```
strk_USDC_to_USDT_data = df.copy()  # Make a copy of the DataFrame

# Define current price of USDC to USDT
current_price_USDC_to_USDT = 1

# Define depth (e.g., 10%)
depth = 0.1

# Convert USDC0 to USDT0 and USDC1 to USDT1
strk_USDC_to_USDT_data['USDT0'] = strk_USDC_data['USDC0'] * current_price_USDC_t
strk_USDC_to_USDT_data['USDT1'] = strk_USDC_data['USDC1'] * current_price_USDC_t

# Calculate lower and upper bounds of the price range
lower_bound = current_price_USDC_to_eth / (1.000001 + depth)
upper_bound = current_price_USDC_to_eth * (1.000001 + depth)

# Filter data within the specified price range for 'USDT0'
tokens_within_range_USDT_ETH0 = strk_USDC_to_USDT_data[(strk_USDC_to_USDT_data['

# Filter data within the specified price range for 'USDT1'
tokens_within_range_USDT_ETH1 = strk_USDC_to_USDT_data[(strk_USDC_to_USDT_data['

# tokens_within_range is the DataFrame where we want to store the filtered data
tokens_within_range['USDT0'] = tokens_within_range_USDT_ETH0['USDT0']
tokens_within_range['USDT1'] = tokens_within_range_USDT_ETH1['USDT1']
```

In [21]:
```
market_depth_USDC_USDT0 = tokens_within_range['USDT0'].sum()
market_depth_USDC_USDT1 = tokens_within_range['USDT1'].sum()

print("Market depth0 in terms of USDC to USDT:", market_depth_USDC_USDT0)
print("Market depth1 in terms of USDC to USDT:", market_depth_USDC_USDT1)
```

```
Market depth0 in terms of USDC to USDT: 0.0005898061895879057
Market depth1 in terms of USDC to USDT: 0.0
```

In [ ]:

PROFIT AND LOSS CALCULATIONS FOR A SET OF HYPOTHETICAL POSITIONS

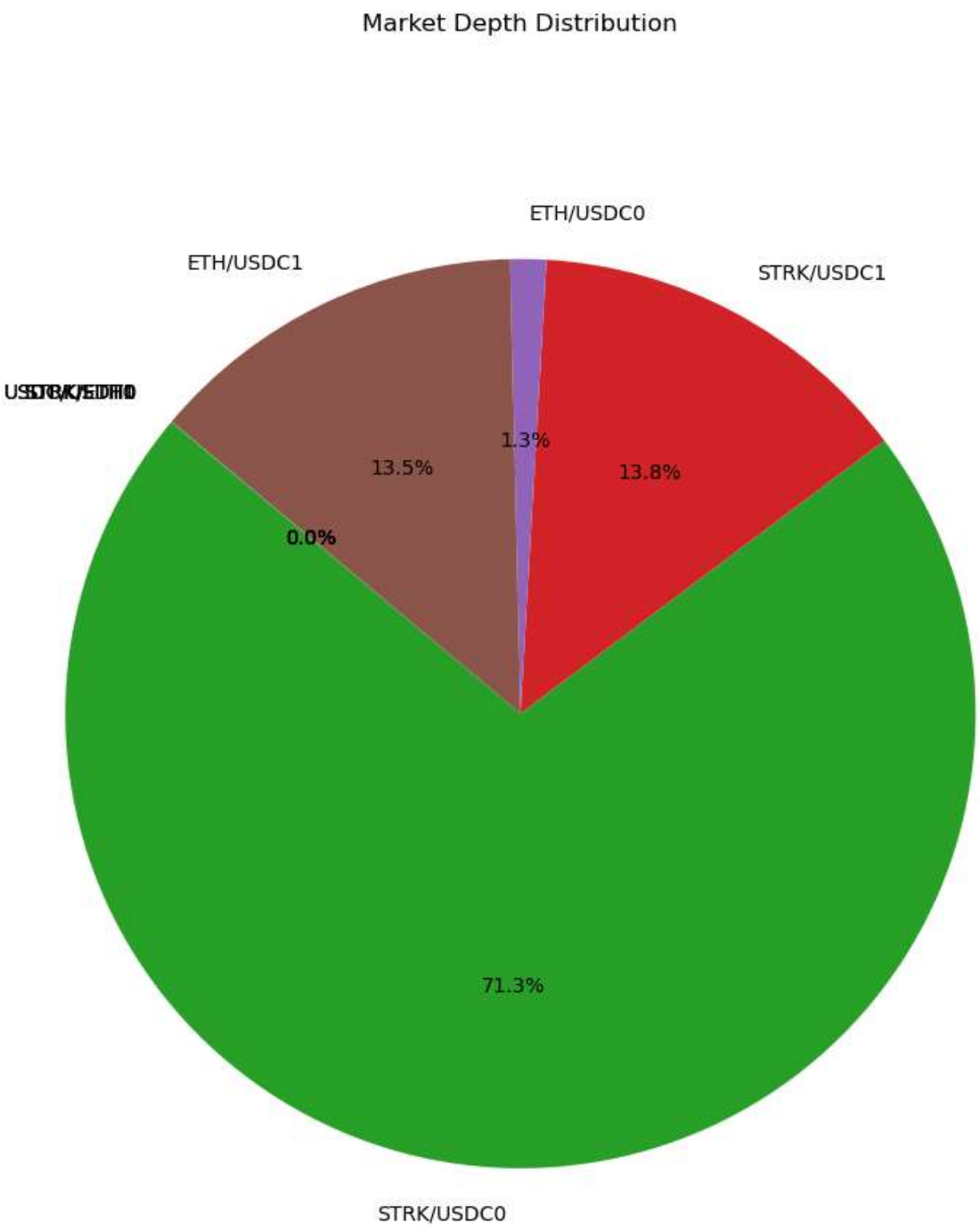
In [37]:
```
import numpy as np

# Function to generate random LP positions
def generate_hypothetical_lp_positions(num_positions):
    data = {
        "EVENT_NAME": np.random.choice(["Mint", "Burn", "Swap"], size=num_positi
        "LIQUIDITY_AMOUNT": np.random.uniform(1, 3024742 , size=num_positions),
        "TOKEN0_REAL_AMOUNT": np.random.uniform(1, 3024742, size=num_positions),
        "TOKEN1_REAL_AMOUNT": np.random.uniform(1, 3024742, size=num_positions),
        # Add other relevant attributes as needed
    }
    return pd.DataFrame(data)

# Simulate PnL calculation for hypothetical LP positions
def simulate_pnl(lp_positions):
    lp_pnl = {}
    for index, row in lp_positions.iterrows():
        event = row["EVENT_NAME"]
```

```
        liquidity_amount = row["LIQUIDITY_AMOUNT"]
        token0_amount = row["TOKEN0_REAL_AMOUNT"]
        token1_amount = row["TOKEN1_REAL_AMOUNT"]

        if event == "Mint":
            pnl = (token0_amount + token1_amount) - liquidity_amount
        elif event == "Burn":
            pnl = liquidity_amount - (token0_amount + token1_amount)
        elif event == "Swap":
            pnl = token0_amount - token1_amount

        lp_pnl[index] = pnl

    return lp_pnl

# Generate hypothetical LP positions
num_positions = 1000  # Number of hypothetical LP positions to generate
hypothetical_lp_positions = generate_hypothetical_lp_positions(num_positions)

# Simulate PnL calculation
hypothetical_lp_pnl = simulate_pnl(df)

# Determine the most profitable hypothetical LPs
most_profitable_hypothetical_lps = sorted(hypothetical_lp_pnl.items(), key=lambd

# Print the outcomes
print("Most Profitable Hypothetical LPs:")
for index, pnl in most_profitable_hypothetical_lps:
    print("LP Index:", index, "| PnL:", pnl)
```

```
Most Profitable Hypothetical LPs:
LP Index: 1012297 | PnL: 618316.0976280000177212059498
LP Index: 1012298 | PnL: 618316.0976280000177212059498
LP Index: 1010321 | PnL: 434004.4910610000079032033682
LP Index: 1877463 | PnL: 395814.3470830000005662441254
LP Index: 1877464 | PnL: 394278.8890470000042114406824
LP Index: 1877971 | PnL: 394101.4638050000066868960857
LP Index: 1878284 | PnL: 393927.6715960000001359730959
LP Index: 1869543 | PnL: 393750.2874960000044666230679
LP Index: 1878287 | PnL: 393567.5056849999818950891495
LP Index: 1878285 | PnL: 393401.5722649999952409416437
```

PIE CHART FOR MARKET DEPTH

In [31]:
```
# Token labels
tokens = ['STRK/ETH0', 'STRK/ETH1', "STRK/USDC0", "STRK/USDC1", "ETH/USDC0", "ET

# Market depth values
market_depth = [market_depth0, market_depth1, market_depth_USDC0, market_depth_U
                market_depth_USDC_ETH0, market_depth_USDC_ETH1, market_depth_USD

# Creating a pie chart
plt.figure(figsize=(9, 12))
plt.pie(market_depth, labels=tokens, autopct='%1.1f%%', startangle=140)
plt.title('Market Depth Distribution')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
plt.show()
```



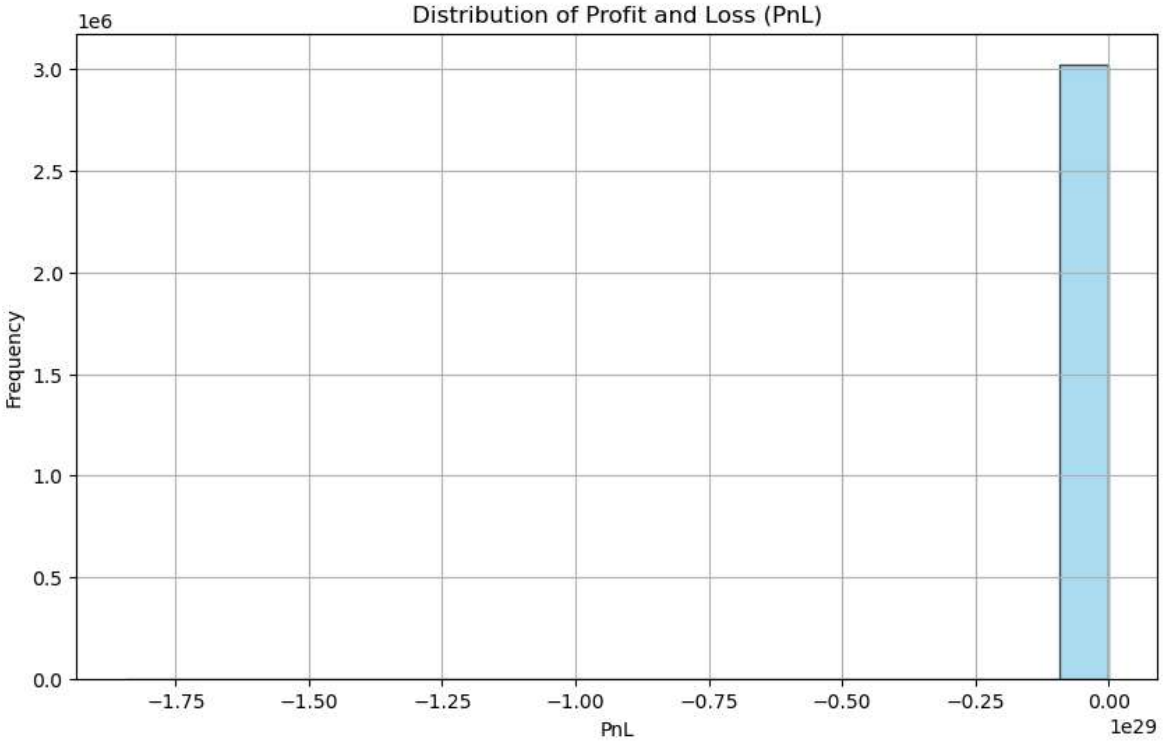
In [32]:
```
# Market depth values
market_depth = [market_depth0, market_depth1, market_depth_USDC0, market_depth_U
                market_depth_USDC_ETH0, market_depth_USDC_ETH1, market_depth_USD

# Convert market depth to numpy array for statistical calculations
market_depth_array = np.array(market_depth)

# Calculate statistics
mean_depth = np.mean(market_depth_array)
median_depth = np.median(market_depth_array)
std_dev_depth = np.std(market_depth_array)
min_depth = np.min(market_depth_array)
max_depth = np.max(market_depth_array)
```

```
# Print statistics
print("Statistics of Market Depth Distribution:")
print("----------------------------------------")
print("Mean Depth:", mean_depth)
print("Median Depth:", median_depth)
print("Standard Deviation of Depth:", std_dev_depth)
print("Minimum Depth:", min_depth)
print("Maximum Depth:", max_depth)
```

```
Statistics of Market Depth Distribution:
----------------------------------------
Mean Depth: 35845.6796035788
Median Depth: 1859.9905720258519
Standard Deviation of Depth: 65821.2872210776
Minimum Depth: 0.0
Maximum Depth: 204563.466819072
```

In [38]:

```
# Convert PnL dictionary to DataFrame
lp_pnl_df = pd.DataFrame(hypothetical_lp_pnl.items(), columns=["LP_Index", "PnL"

# 1. Calculate Cumulative PnL
cumulative_pnl = lp_pnl_df["PnL"].sum()

# 2. Visualize PnL Distribution
plt.figure(figsize=(10, 6))
plt.hist(lp_pnl_df["PnL"], bins=20, color="skyblue", edgecolor="black", alpha=0.
plt.title("Distribution of Profit and Loss (PnL)")
plt.xlabel("PnL")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()

# 3. Identify Profitable vs. Unprofitable LPs
profitable_lps = lp_pnl_df[lp_pnl_df["PnL"] > 0]
unprofitable_lps = lp_pnl_df[lp_pnl_df["PnL"] <= 0]

# 4. Profitability Metrics
average_pnl_per_lp = lp_pnl_df["PnL"].mean()
total_profit = profitable_lps["PnL"].sum()
total_loss = unprofitable_lps["PnL"].sum()
profit_margin = (total_profit / (total_profit + abs(total_loss))) * 100
roi = (total_profit / cumulative_pnl) * 100

# 5. Print Summary
print("Profitability Analysis Summary:")
print("------------------------------")
print("Cumulative PnL: {:.2f}".format(cumulative_pnl))
print("Average PnL per LP: {:.2f}".format(average_pnl_per_lp))
print("Total Profit: {:.2f}".format(total_profit))
print("Total Loss: {:.2f}".format(total_loss))
print("Profit Margin: {:.2f}%".format(profit_margin))
print("Return on Investment (ROI): {:.2f}%".format(roi))
```




```
Profitability Analysis Summary:
------------------------------
Cumulative PnL: -4154131419188415433578335792 00.00
Average PnL per LP: -137338417378162794037248.00
Total Profit: 1970321389.98
Total Loss: -4154131419188415433598000775 00.00
Profit Margin: 0.00%
Return on Investment (ROI): -0.00%
```