Quant Research Assignment
Rimjhim Goel

March 24, 2024

Research Thesis

Nifty and Bank Nifty indices have significant overlap in terms of their constituents and weights. Our thesis is that their volatilities are also likely to be affected by similar market conditions and macroeconomic factors. We can thus construct a pairs trading strategy that attempts to capture any dispersion that may occur between the volatilities of the two indices

Approach: I have tried to make a base model using z-score as given in the assignment.

To build a better model, I have used linear regression ad used standard deviation as the analogue for z-score in base model. I did hypothesis testing using OLS test in python to check if banknifty and nifty values can fit in a linear regression model. I found r-sqaured value close to 0.9, and the relation

$$\text{Bank Nifty} = 1.1320 \times \text{Nifty} + 0.0481 \tag{1}$$

Results and findings:

To compare the two models, I used the profitablity measure and the same formula given in the assignment $\text{P/L} = \text{Spread} \times (\text{Time To Expiry})^{0.7}$. I have taken two thresholds, a narrow $(-1.5, 2)$ and a wider $(-2, 3)$ for base model as well as the linear regression model. Improvement was seen in profitability for both the cases.

Buy Threshold	Sell Threshold	Profit for z-score	Profit for Linear model
-2.0	3.0	55.135229	61.812865
-1.5	2.0	114.890729	121.435889

Why expect the result obtained:

- Spread used in base model is simply the difference, linear fitting is better than this. Spread is like scaling by addition only, whereas linear model scales by addition and a multiplication factor.
- Z-score assumes an underlying normal distribution, unlike linear fit.
- Ease of comparison is there because of similar underlying mathematical operations for the two models.

A brief summary of code.

- Loading, visualising and cleaning the data. Cleaning has been done by removing the non trading hours and interpolating the missing values.
- Calculating spread, Z-scores and simulating the trade assuming a threshold value.
- Defining function 'calculate profit loss' that calculates the value of profit or loss if the assumed threshold was applied on historical data provided in data set.
- Fitting the best-fit line to find a relationship between nifty and banknifty volatalities.
- Tailoring the profit calculating function for this new model and naming it 'calc pl'.
- comparing the values for different values of thresholds for both models.

Conclusion: There is indeed a relationship. So the proposed hypothesis cannot be rejected.

Following is the code snippets

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
```

[2]:
```
df = pd.read_parquet('data_tbe.parquet')

# Define trading hours
start_trading_time = pd.Timestamp('09:15:00').time()
end_trading_time = pd.Timestamp('15:30:00').time()

df_trading_hours = df.between_time(start_trading_time, end_trading_time)

# print(df_trading_hours.head())
df_trading_hours
```

[2]:

time	banknifty	nifty	tte
2021-01-01 09:15:00	0.286058	0.199729	27
2021-01-01 09:16:00	0.285381	0.200433	27
2021-01-01 09:17:00	0.284233	0.200004	27
2021-01-01 09:18:00	0.286104	0.199860	27
2021-01-01 09:19:00	0.285539	0.198951	27
...	...	...	...
2022-06-30 15:26:00	0.240701	0.214758	28
2022-06-30 15:27:00	0.240875	0.216558	28
2022-06-30 15:28:00	0.242115	0.216794	28

```
2022-06-30 15:29:00   0.243426  0.216455   28
2022-06-30 15:30:00   0.241907  0.216081   28

[180856 rows x 3 columns]
```

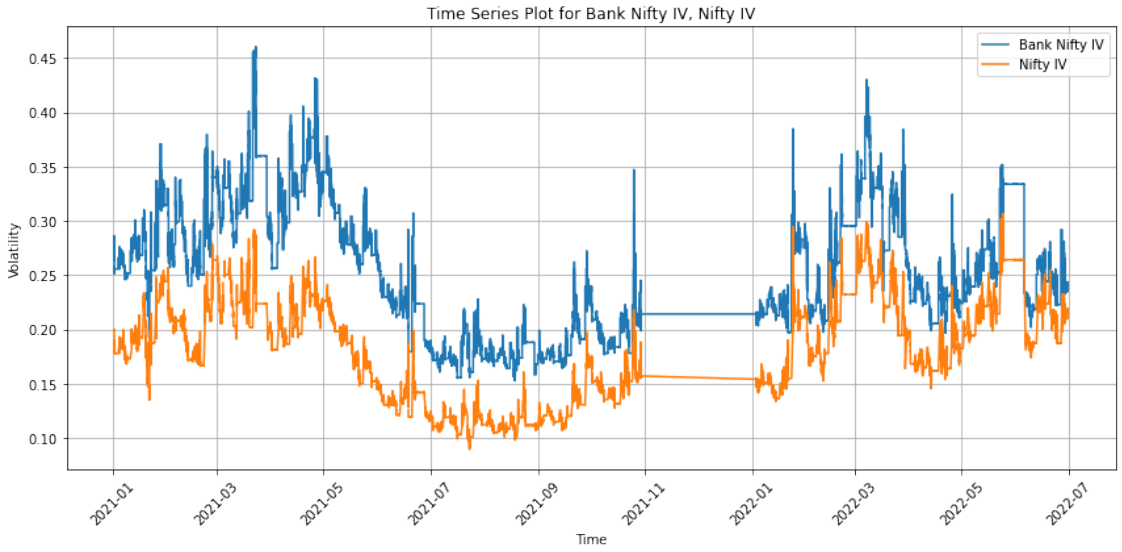
[3]:
```
df= df_trading_hours
```

[4]:
```
missing_values_per_column = df.isnull().sum()

# Check for missing values in the entire dataset
total_missing_values = df.isnull().sum().sum()

print("Missing values per column:")
print(missing_values_per_column)
print("\nTotal missing values in the dataset:", total_missing_values)
```

```
Missing values per column:
banknifty    370
nifty        477
tte            0
dtype: int64

Total missing values in the dataset: 847
```

[5]:
```
# df
```

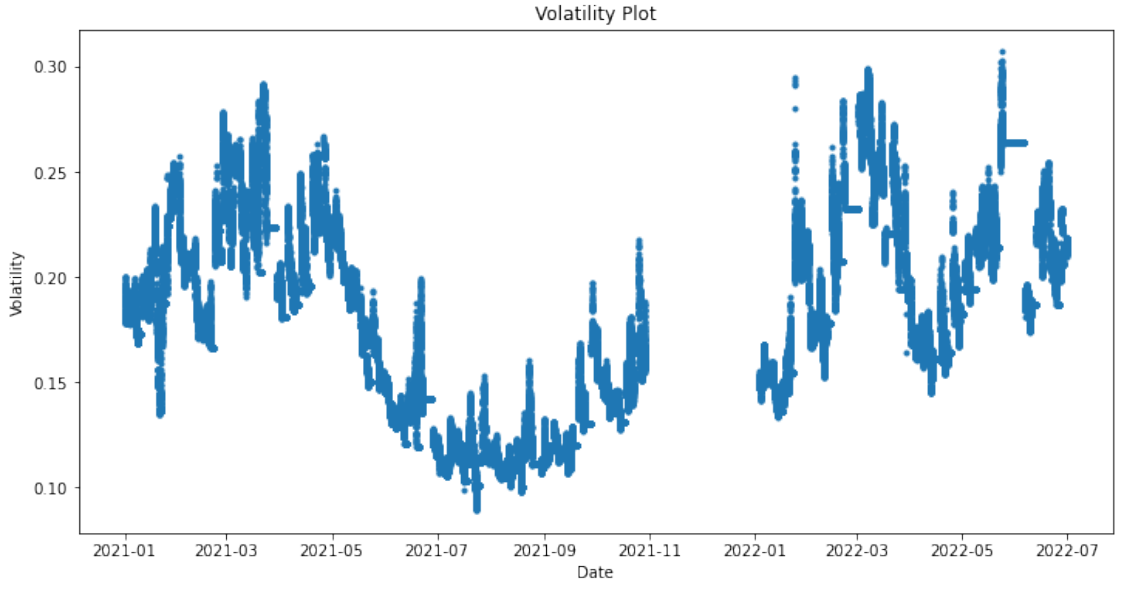
[6]:
```
plt.figure(figsize=(12, 6)) # Plot Bank Nifty IV
plt.plot(df.index, df['banknifty'], label='Bank Nifty IV',drawstyle='steps')
plt.plot(df.index, df['nifty'], label='Nifty IV')

plt.xlabel('Time')
plt.ylabel('Volatility')
plt.title('Time Series Plot for Bank Nifty IV, Nifty IV')
plt.legend()
plt.xticks(rotation=45)

plt.grid(True)
plt.tight_layout()
plt.show()
# df
```



[7]:
```
correlation = df['nifty'].corr(df['banknifty'])
print("Correlation between Nifty and Bank Nifty:", correlation)
```

```
Correlation between Nifty and Bank Nifty: 0.897291202690001
```

[8]:
```
plt.figure(figsize=(12, 6))
plt.scatter(df.index, df['nifty'], marker='.')

plt.xlabel('Date')
plt.ylabel('Volatility')
plt.title('Volatility Plot')
plt.show()
```



[9]:
```
start_date = '2021-11-01'
end_date = '2022-01-30'

df_investigation = df[start_date:end_date]
df_investigation

#concluding that 2021-11-01 to 2022-01-30 data is missing in the dataset and
# that is why the total number of rows was less than expected value of␣
 ↪375*(365+181)
```

[9]:

time	banknifty	nifty	tte
2022-01-03 09:15:00	0.214152	0.154115	24
2022-01-03 09:16:00	0.214935	0.155385	24
2022-01-03 09:17:00	0.216027	0.154671	24
2022-01-03 09:18:00	0.213095	0.153413	24
2022-01-03 09:19:00	0.213160	0.153133	24
...	...	...	...
2022-01-30 15:26:00	0.282907	0.211083	27
2022-01-30 15:27:00	0.282907	0.211083	27
2022-01-30 15:28:00	0.282907	0.211083	27
2022-01-30 15:29:00	0.282907	0.211083	27
2022-01-30 15:30:00	0.282907	0.211083	27

[10528 rows x 3 columns]

[10]:
```
# Calculate the percentage of missing values in each column
missing_percentage = (df.isnull().sum() / len(df)) * 100
missing_percentage
# plt.figure(figsize=(10, 6))
# missing_percentage.plot(kind='bar')
# plt.title('Percentage of Missing Values in Each Column')
# plt.xlabel('Columns')
# plt.ylabel('Percentage of Missing Values')
# plt.xticks(rotation=45)
# plt.show()
```

[10]:
```
banknifty    0.204583
nifty        0.263746
tte          0.000000
dtype: float64
```

[11]:
```
df.interpolate(method='polynomial', order=2, inplace=True)  # 2nd order␣
 ↪polynomial
```

```
print(df.isna().sum())  # Verify if any missing values remain
```

```
banknifty    0
nifty        0
tte          0
dtype: int64
```

[12]:
```
df.shape
```

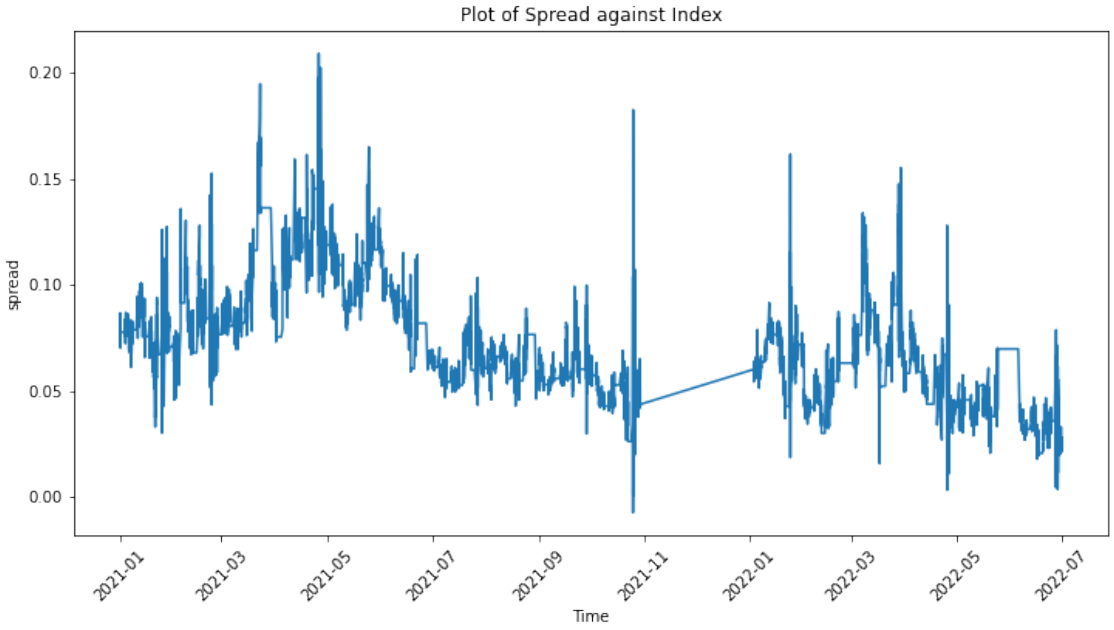
[12]:
```
(180856, 3)
```

[13]:
```
df['Spread'] = df['banknifty'] - df['nifty']
df= pd.DataFrame(df)
# df
```

[14]:
```
# Calculate mean and standard deviation of the spread
spread_mean = df['Spread'].mean()
spread_std = df['Spread'].std()

print(f"The mean of the spread is {spread_mean:.4f}.")
print(f"The standard deviation of the spread is {spread_std:.4f}.")
```

```
The mean of the spread is 0.0719.
The standard deviation of the spread is 0.0265.
```

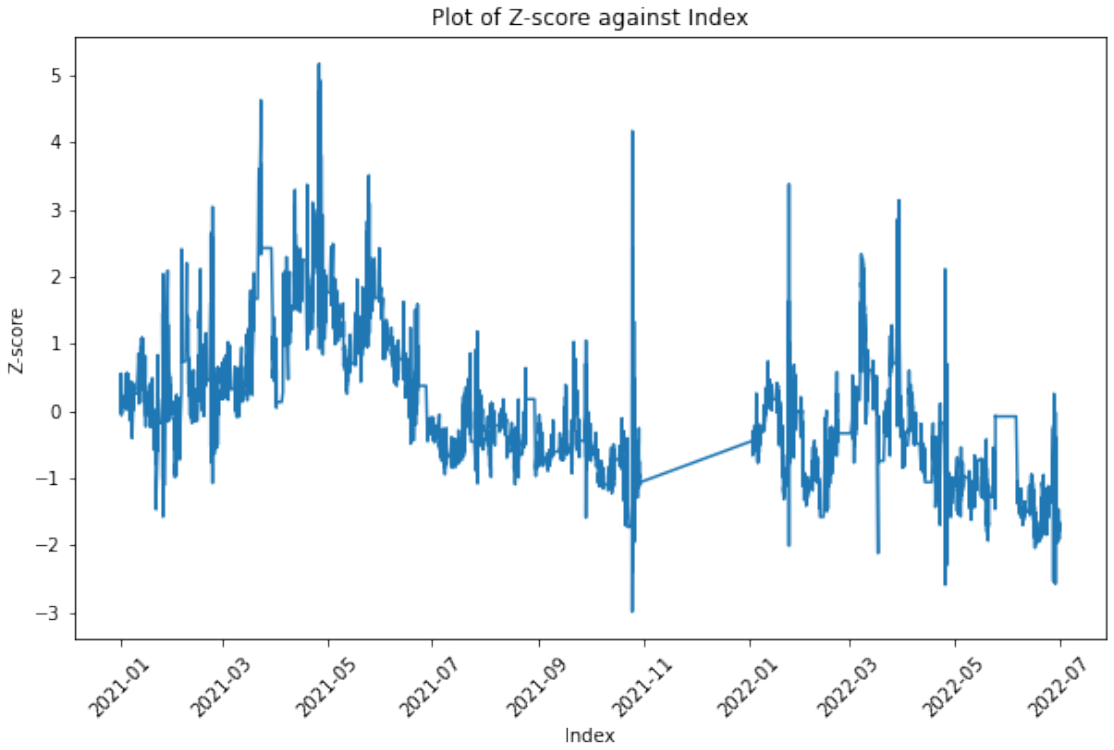
[15]:
```
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Spread'])

plt.xlabel('Time')
plt.ylabel('spread')
plt.title('Plot of Spread against Index')
plt.xticks(rotation=45)
plt.show()
```



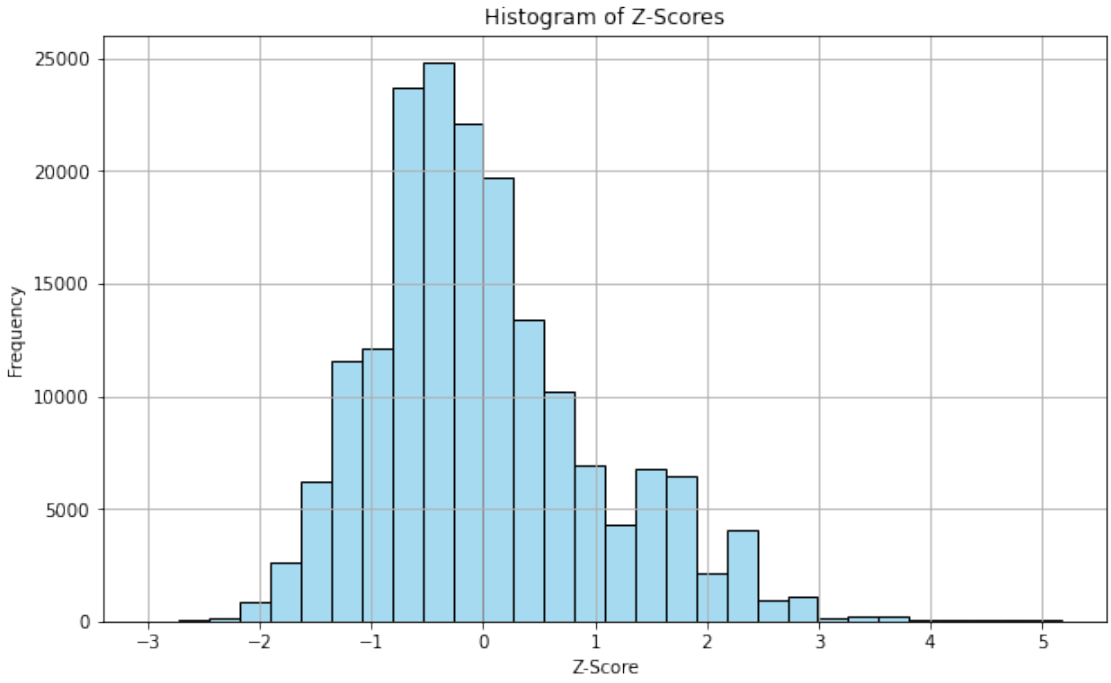
[16]:
```
# Calculate the z-score for each spread value
df['Z-Score'] = (df['Spread'] - spread_mean) / spread_std
df['Z-Score'].min(),df['Z-Score'].max()
```

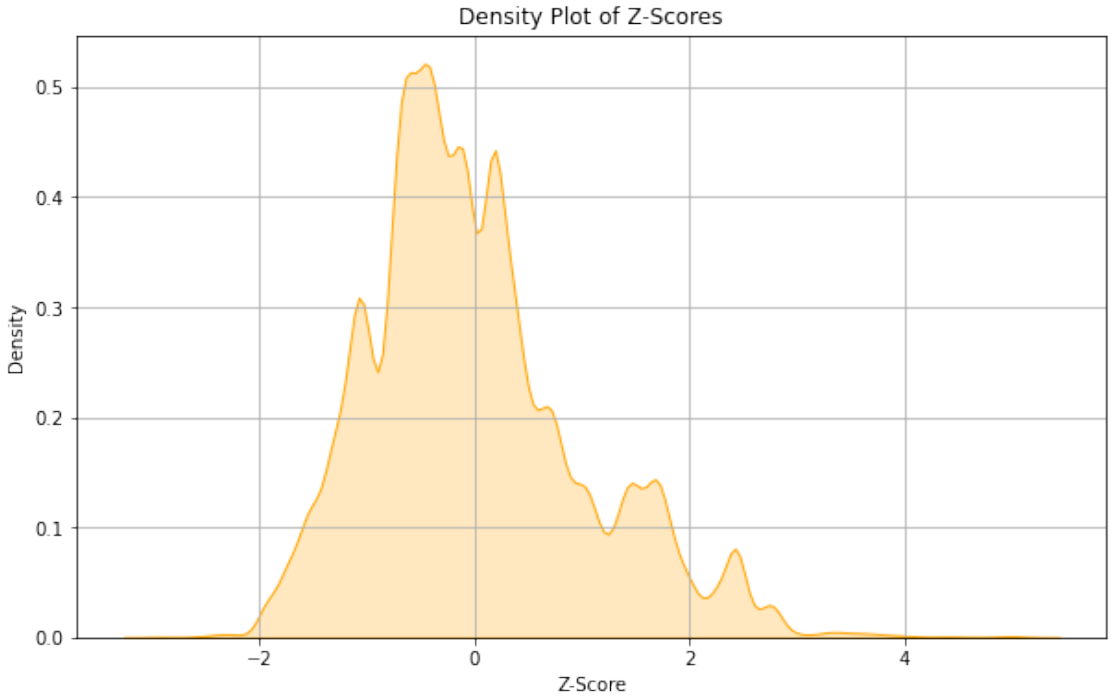
[16]: (-2.992490505974068, 5.167589485682149)

[17]:
```
plt.figure(figsize=(10, 6))
plt.plot(df.index, df['Z-Score'])

plt.xlabel('Index')
plt.ylabel('Z-score')
plt.title('Plot of Z-score against Index')
plt.xticks(rotation=45)

plt.show()
```



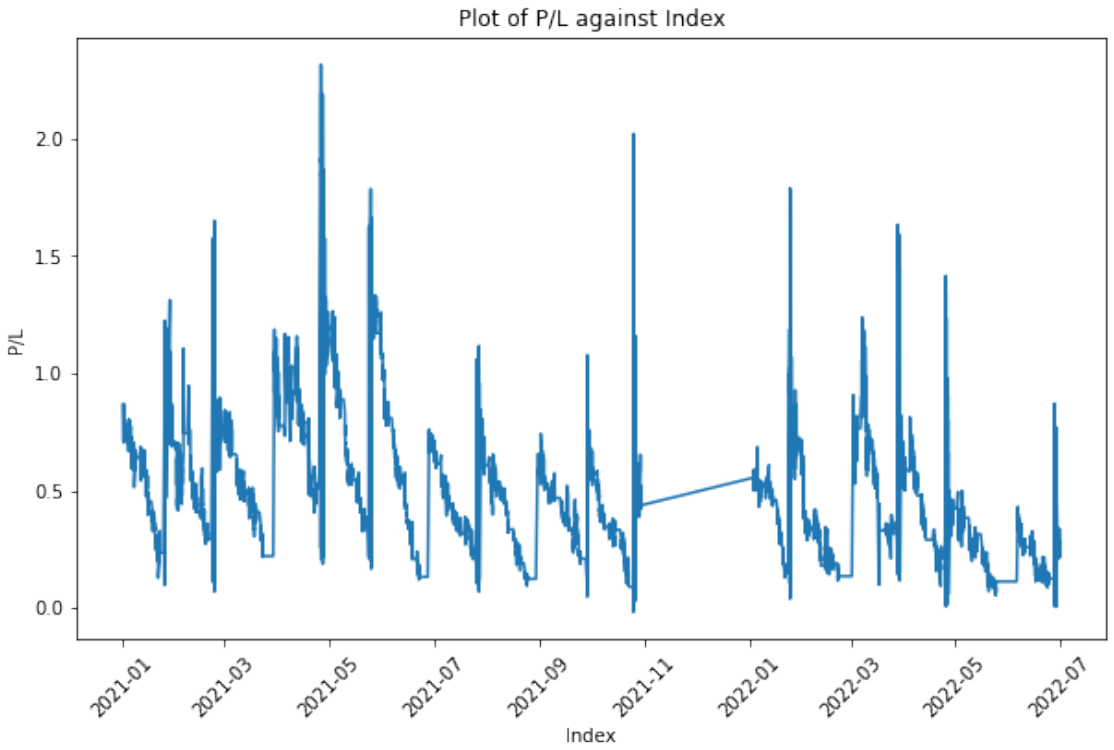
[18]:
```
# Plot a histogram of Z-scores
plt.figure(figsize=(10, 6))
sns.histplot(df['Z-Score'], bins=30, kde=False, color='skyblue')
plt.title('Histogram of Z-Scores')
plt.xlabel('Z-Score')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

# Plot a density plot of Z-scores
plt.figure(figsize=(10, 6))
sns.kdeplot(df['Z-Score'], color='orange', shade=True)
plt.title('Density Plot of Z-Scores')
plt.xlabel('Z-Score')
plt.ylabel('Density')
plt.grid(True)
plt.show()
```





[21]:
```
df['P/L'] = df['Spread'] * (df['tte'] ** 0.7)
# df
```

[22]:
```
plt.figure(figsize=(10, 6))
plt.plot(df.index, df['P/L'])

plt.xlabel('Index')
plt.ylabel('P/L')
plt.title('Plot of P/L against Index')
plt.xticks(rotation=45)

plt.show()
```



[ ]:
```
# df
```

[23]:
```
buy_threshold =  -1.5
sell_threshold =2

# Generate trading signals based on z-scores and thresholds
df['Signal'] = np.where(df['Z-Score'] < buy_threshold, 'Buy',
                        np.where(df['Z-Score'] > sell_threshold, 'Sell',␣
 ↪'Hold'))# Initialize variables to track trades and P/L
trades = []
current_position = None
entry_price = None
```

```
exit_price = None
current_pnl = 0

# Iterate over each row in the DataFrame
for index, row in df.iterrows():
    # Check if there's a signal to buy
    if row['Signal'] == 'Buy' and current_position != 'Buy':
        # Close existing position if any
        if current_position == 'Sell':
            exit_price = row['Spread'] * (row['tte'] ** 0.7)
            current_pnl += entry_price - exit_price
            trades.append((index, 'Sell', entry_price, exit_price, current_pnl))
        # Open new long position
        entry_price = row['Spread'] * (row['tte'] ** 0.7)
        current_position = 'Buy'
    # Check if there's a signal to sell
    elif row['Signal'] == 'Sell' and current_position != 'Sell':
        # Close existing position if any
        if current_position == 'Buy':
            exit_price = row['Spread'] * (row['tte'] ** 0.7)
            current_pnl += exit_price - entry_price
            trades.append((index, 'Buy', entry_price, exit_price, current_pnl))
        # Open new short position
        entry_price = row['Spread'] * (row['tte'] ** 0.7)
        current_position = 'Sell'
    # Hold position if no signal
    elif row['Signal'] == 'Hold':
        # Close existing position if any
        if current_position is not None:
            exit_price = row['Spread'] * (row['tte'] ** 0.7)
            if current_position == 'Buy':
                current_pnl += exit_price - entry_price
            elif current_position == 'Sell':
                current_pnl += entry_price - exit_price
            trades.append((index, current_position, entry_price, exit_price,␣
 ↪current_pnl))
            current_position = None

# Print the trades and their P/L
print("Trades:")
print("Index\t\tPosition\tEntry Price\tExit Price\tP/L")
for trade in trades:
    print(f"{trade[0]}\t{trade[1]}\t\t{trade[2]:.3f}\t\t{trade[3]:.
 ↪2f}\t\t{trade[4]:.3f}")
```

```
Trades:
Index           Position        Entry Price     Exit Price      P/L
2021-01-25 14:44:00     Sell            0.272           0.21            0.058
```

2021-01-25 15:19:00	Buy	0.331	0.59	0.316
2021-01-28 09:16:00	Sell	1.312	1.11	0.522
2021-02-05 10:10:00	Sell	1.026	0.99	0.563
2021-02-08 09:21:00	Sell	0.932	0.88	0.611
2021-02-16 09:19:00	Sell	0.596	0.58	0.629
2021-02-22 09:22:00	Sell	1.461	1.26	0.826
2021-02-22 09:30:00	Sell	1.568	0.13	2.266
2021-02-22 09:40:00	Sell	1.483	1.11	2.643
2021-02-22 09:44:00	Sell	1.473	1.19	2.927

Only few entries of the output have been shown here due to extremely large number of values.

[ ]:
```
# df
```

[43]:
```
def calculate_profit_loss(df, buy_threshold, sell_threshold):
    temp = df

    temp['Signal'] = np.where(temp['Z-Score'] < buy_threshold, 'Buy',
                        np.where(temp['Z-Score'] > sell_threshold, 'Sell',␣
↪'Hold'))# Initialize variables to track trades and P/L
    trades = []
    current_position = None
    entry_price = None
    exit_price = None
    current_pnl = 0

    # Iterate over each row in the DataFrame
    for index, row in temp.iterrows():
        # Check if there's a signal to buy
        if row['Signal'] == 'Buy' and current_position != 'Buy':
            # Close existing position if any
            if current_position == 'Sell':
                exit_price = row['Spread'] * (row['tte'] ** 0.7)
                current_pnl += entry_price - exit_price
                trades.append((index, 'Sell', entry_price, exit_price,␣
↪current_pnl))
            # Open new long position
            entry_price = row['Spread'] * (row['tte'] ** 0.7)
            current_position = 'Buy'
        # Check if there's a signal to sell
        elif row['Signal'] == 'Sell' and current_position != 'Sell':
            # Close existing position if any
            if current_position == 'Buy':
                exit_price = row['Spread'] * (row['tte'] ** 0.7)
                current_pnl += exit_price - entry_price
                trades.append((index, 'Buy', entry_price, exit_price,␣
↪current_pnl))
            # Open new short position
```

```
            entry_price = row['Spread'] * (row['tte'] ** 0.7)
            current_position = 'Sell'
        # Hold position if no signal
        elif row['Signal'] == 'Hold':
            # Close existing position if any
            if current_position is not None:
                exit_price = row['Spread'] * (row['tte'] ** 0.7)
                if current_position == 'Buy':
                    current_pnl += exit_price - entry_price
                elif current_position == 'Sell':
                    current_pnl += entry_price - exit_price
                trades.append((index, current_position, entry_price, exit_price,␣
 ↪current_pnl))
                current_position = None

    # Calculate total profit/loss
    total_profit_loss = current_pnl
    return total_profit_loss
```

[26]:
```
calculate_profit_loss(df,-1.5,2)
```

[26]:
```
114.89072928903005
```

[ ]:
```
# df
```

[44]:
```
calculate_profit_loss(df,-2,3)
```

[44]:
```
55.13522863747638
```

[27]:
```
thres_list = [(-2,3), (-1.5,2)]

# Create an empty DataFrame to store results
results_df = pd.DataFrame(columns=['Buy Threshold', 'Sell Threshold', 'Total␣
 ↪Profit/Loss'])

# Iterate through the threshold tuples
for threshold in thres_list:
    buy_threshold, sell_threshold = threshold
    total_profit_loss = calculate_profit_loss(df, buy_threshold, sell_threshold)
    results_df = results_df.append({'Buy Threshold': buy_threshold,
                                    'Sell Threshold': sell_threshold,
                                    'Total Profit/Loss': total_profit_loss},␣
 ↪ignore_index=True)

# Print the results DataFrame
print(results_df)
```

```
  Buy Threshold  Sell Threshold  Total Profit/Loss
```

```
0           -2.0             3.0          55.135229
1           -1.5             2.0         114.890729
```

[49]:
```
thres_list = [(-2,3), (-1.5,2)]

# Create an empty DataFrame to store results
results_df = pd.DataFrame(columns=['Buy Threshold', 'Sell Threshold', 'Total␣
 ↪Profit/Loss'])

# Iterate through the threshold tuples
for threshold in thres_list:
    buy_threshold, sell_threshold = threshold
    total_profit_loss = calculate_profit_loss(df, buy_threshold, sell_threshold)
    results_df = results_df.append({'Buy Threshold': buy_threshold,
                                    'Sell Threshold': sell_threshold,
                                    'Total Profit/Loss': total_profit_loss},␣
 ↪ignore_index=True)

# Print the results DataFrame
print(results_df)
```

```
   Buy Threshold  Sell Threshold  Total Profit/Loss
0           -2.0             3.0          55.135229
1           -1.5             2.0         114.890729
```

Perform the Engle-Granger cointegration test

nifty$_s$eries $= df['nifty']banknifty_s eries = df['banknifty']$

result = sm.tsa.stattools.coint(nifty$_s$eries, $banknifty_s eries)$

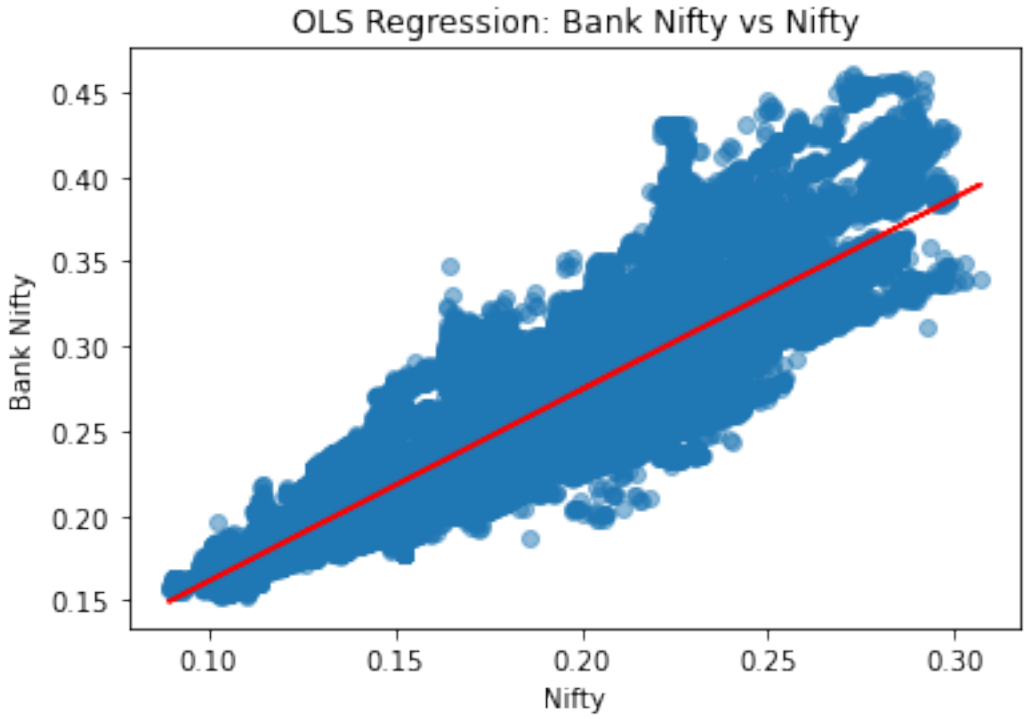
test$_s$tatistic $= result[0]p_v alue = result[1]$

print("Cointegration Test Results:") print(f"Test Statistic: test$_s$tatistic")$print(f"P - value : p_v alue")$

if $p_v alue < 0.05 : print("Rejectthenullhypothesis : Theseriesarecointegrated.")else : print("Failtorejectthenullhypothesis : Theseriesarenotcointegrated.")$

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              banknifty   R-squared:                       0.805
Model:                            OLS   Adj. R-squared:                  0.805
Method:                 Least Squares   F-statistic:                 7.472e+05
Date:                Sun, 24 Mar 2024   Prob (F-statistic):               0.00
Time:                        21:58:17   Log-Likelihood:             4.0465e+05
No. Observations:              180856   AIC:                        -8.093e+05
Df Residuals:                  180854   BIC:                        -8.093e+05
Df Model:                           1
Covariance Type:            nonrobust
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0481      0.000    196.914      0.000       0.048       0.049
nifty          1.1320      0.001    864.385      0.000       1.129       1.135
==============================================================================
Omnibus:                     8069.280   Durbin-Watson:                   0.021
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             9317.561
Skew:                           0.521   Prob(JB):                         0.00
Kurtosis:                       3.388   Cond. No.                         22.3
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```



[30]:
```
slope = model.params['nifty']
intercept = model.params['const']
mse = np.mean(residuals ** 2)
std=np.sqrt(mse)

print(f"Regression Line Equation: Bank Nifty = {slope:.4f} * Nifty + {intercept:.
↪4f}")
```

```
print("Mean Squared Error:", mse)
print("Standard deviation:", std)
```

```
Regression Line Equation: Bank Nifty = 1.1320 * Nifty + 0.0481
Mean Squared Error: 0.0006670163040880765
Standard deviation: 0.025826658786766758
```

[38]:
```
df['pred_banknifty']=model.predict(X)
df['diff']=(df['banknifty']-df['pred_banknifty'])/std
```

[38]:

time	banknifty	nifty	tte	Spread	Z-Score	P/L	Signal	pred_banknifty	diff	Signal2
2021-01-01 09:15:00	0.286058	0.199729	27	0.086329	0.543765	0.867184	Hold	0.274159	0.460713	Hold
2021-01-01 09:16:00	0.285381	0.200433	27	0.084948	0.491753	0.853317	Hold	0.274956	0.403665	Hold
2021-01-01 09:17:00	0.284233	0.200004	27	0.084229	0.464645	0.846089	Hold	0.274471	0.377996	Hold
2021-01-01 09:18:00	0.286104	0.199860	27	0.086244	0.540543	0.866325	Hold	0.274308	0.456733	Hold
2021-01-01 09:19:00	0.285539	0.198951	27	0.086588	0.553523	0.869786	Hold	0.273279	0.474717	Hold
...	...	...	...	...	...	...	...	...	...	...
2022-06-30 15:26:00	0.240701	0.214758	28	0.025943	-1.731329	0.267320	Buy	0.291172	-1.954226	Buy
2022-06-30 15:27:00	0.240875	0.216558	28	0.024317	-1.792609	0.250560	Buy	0.293210	-2.026406	Buy
2022-06-30 15:28:00	0.242115	0.216794	28	0.025321	-1.754764	0.260910	Buy	0.293476	-1.988713	Buy
2022-06-30 15:29:00	0.243426	0.216455	28	0.026971	-1.692598	0.277912	Buy	0.293094	-1.923098	Buy
2022-06-30 15:30:00	0.241907	0.216081	28	0.025827	-1.735700	0.266124	Buy	0.292669	-1.965477	Buy

[180856 rows x 10 columns]

[46]:
```
def calc_pl(df, buy_threshold, sell_threshold):
    tem=df
    tem['Signal'] = np.where(tem['diff'] < buy_threshold, 'Buy',
                        np.where(tem['diff'] > sell_threshold, 'Sell', 'Hold'))#␣
 ↪Initialize variables to track trades and P/L
    trades = []
    current_position = None
    entry_price = None
```

```
    exit_price = None
    current_pnl = 0

    # Iterate over each row in the DataFrame
    for index, row in tem.iterrows():
        # Check if there's a signal to buy
        if row['Signal'] == 'Buy' and current_position != 'Buy':
            # Close existing position if any
            if current_position == 'Sell':
                exit_price = row['Spread'] * (row['tte'] ** 0.7)
                current_pnl += entry_price - exit_price
                trades.append((index, 'Sell', entry_price, exit_price,␣
↪current_pnl))
            # Open new long position
            entry_price = row['Spread'] * (row['tte'] ** 0.7)
            current_position = 'Buy'
        # Check if there's a signal to sell
        elif row['Signal'] == 'Sell' and current_position != 'Sell':
            # Close existing position if any
            if current_position == 'Buy':
                exit_price = row['Spread'] * (row['tte'] ** 0.7)
                current_pnl += exit_price - entry_price
                trades.append((index, 'Buy', entry_price, exit_price,␣
↪current_pnl))
            # Open new short position
            entry_price = row['Spread'] * (row['tte'] ** 0.7)
            current_position = 'Sell'
        # Hold position if no signal
        elif row['Signal'] == 'Hold':
            # Close existing position if any
            if current_position is not None:
                exit_price = row['Spread'] * (row['tte'] ** 0.7)
                if current_position == 'Buy':
                    current_pnl += exit_price - entry_price
                elif current_position == 'Sell':
                    current_pnl += entry_price - exit_price
                trades.append((index, current_position, entry_price, exit_price,␣
↪current_pnl))
                current_position = None

    total_profit_loss = current_pnl
    return total_profit_loss
```

[42]:
```
df.drop(columns=['Signal2','Signal'])
```

[42]:
```
                     banknifty     nifty  tte    Spread   Z-Score       P/L  \
time
```

```
2021-01-01 09:15:00   0.286058  0.199729   27  0.086329  0.543765  0.867184
2021-01-01 09:16:00   0.285381  0.200433   27  0.084948  0.491753  0.853317
2021-01-01 09:17:00   0.284233  0.200004   27  0.084229  0.464645  0.846089
2021-01-01 09:18:00   0.286104  0.199860   27  0.086244  0.540543  0.866325
2021-01-01 09:19:00   0.285539  0.198951   27  0.086588  0.553523  0.869786
...                        ...       ...  ...       ...       ...       ...
2022-06-30 15:26:00   0.240701  0.214758   28  0.025943 -1.731329  0.267320
2022-06-30 15:27:00   0.240875  0.216558   28  0.024317 -1.792609  0.250560
2022-06-30 15:28:00   0.242115  0.216794   28  0.025321 -1.754764  0.260910
2022-06-30 15:29:00   0.243426  0.216455   28  0.026971 -1.692598  0.277912
2022-06-30 15:30:00   0.241907  0.216081   28  0.025827 -1.735700  0.266124

                     pred_banknifty      diff
time
2021-01-01 09:15:00        0.274159  0.460713
2021-01-01 09:16:00        0.274956  0.403665
2021-01-01 09:17:00        0.274471  0.377996
2021-01-01 09:18:00        0.274308  0.456733
2021-01-01 09:19:00        0.273279  0.474717
...                             ...       ...
2022-06-30 15:26:00        0.291172 -1.954226
2022-06-30 15:27:00        0.293210 -2.026406
2022-06-30 15:28:00        0.293476 -1.988713
2022-06-30 15:29:00        0.293094 -1.923098
2022-06-30 15:30:00        0.292669 -1.965477

[180856 rows x 8 columns]
```

[ ]:
```
# calc_pl(df,-1.5,2)
```

[47]:
```
thres_list = [(-2,3), (-1.5,2)]

# Create an empty DataFrame to store results
results = pd.DataFrame(columns=['Buy Threshold', 'Sell Threshold', 'Total Profit/
 ↪Loss'])

# Iterate through the threshold tuples
for threshold in thres_list:
    buy_threshold, sell_threshold = threshold
    total_profit_loss = calc_pl(df, buy_threshold, sell_threshold)
    results = results.append({'Buy Threshold': buy_threshold,
                              'Sell Threshold': sell_threshold,
                              'Total Profit/Loss': total_profit_loss},␣
 ↪ignore_index=True)

# Print the results DataFrame
print(results)
```

```
   Buy Threshold  Sell Threshold  Total Profit/Loss
0           -2.0             3.0          61.812865
1           -1.5             2.0         121.435889
```

[52]:
```
thres_list = [(-2, 3), (-1.5, 2)]

results_combined = pd.DataFrame(columns=['Buy Threshold', 'Sell Threshold',␣
 ↪'Total P/L for z-score', 'Total P/L for Linear model'])

for threshold in thres_list:
    buy_threshold, sell_threshold = threshold
    total_profit_loss_z = calculate_profit_loss(df, buy_threshold,␣
 ↪sell_threshold)
    total_profit_loss_l = calc_pl(df, buy_threshold, sell_threshold)
    results_combined = results_combined.append({'Buy Threshold': buy_threshold,
                                                'Sell Threshold': sell_threshold,
                                                'Total P/L for z-score':␣
 ↪total_profit_loss_z,
                                                'Total P/L for Linear model':␣
 ↪total_profit_loss_l}, ignore_index=True)
print(results_combined)
```

```
   Buy Threshold  Sell Threshold  Total P/L for z-score  \
0           -2.0             3.0              55.135229
1           -1.5             2.0             114.890729

   Total P/L for Linear model
0                   61.812865
1                  121.435889
```

$\text{window}_size = 30$

Compute rolling mean and standard deviation of the spread $\text{rolling}_mean = df['Spread'].rolling(window = window_size).mean() rolling_std = df['Spread'].rolling(window = window_size).std()$

Plot the original time series and the rolling statistics plt.figure(figsize=(10, 6)) plt.plot(df['Spread'], label='Spread') $plt.plot(rolling_mean, label = f'window_size - dayRollingMean', color = 'red') plt.fill_between(rolling_mean.index, rolling_mean - rolling_std, rolling_mean + rolling_std, color = 'lightgray', alpha = 0.4, label = 'RollingStdDev') plt.xlabel('Date') plt.ylabel('SpreadValue') plt.title('RollingAnalysisofSpread') plt.legend() plt.sh$