Jatin Deshpande (21b080014) - True Beacon Assignment

Pairs Trading Strategy

Before creating our strategy let us first analyse our data and the check for the pair given for pairs trading strategy:

In [1]:
```
# importing required libraries
import pandas as pd
import numpy as np
from scipy.stats import zscore
from statsmodels.tsa.stattools import coint
import matplotlib.pyplot as plt
%matplotlib inline
import statsmodels.api as sm
from statsmodels.regression.linear_model import OLS
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

In [3]:
```
# Taking data relevant to indian trading time which is given.
data=pd.read_parquet("data.parquet")
data = data.between_time('09:15', '15:30').copy()
data
```

Out[3]:

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 × 3 columns

In [4]:
```
# Handling the missing values
missing_values = data.isnull().sum()
if missing_values.sum() > 0:
    data_filled = data.fillna(method='ffill')
else:
    data_filled = data
```

Here we are using ffill because we don't want to miss important data_points from other row if we omit and also we dont want to exit/entry our trade due to such values, which can also happen if we bfill

Now if we can use them for paris trading we need to perform the following test, where we will check if they are cointegrated or not.

Here we use the "Engle Granger tests" - The idea of Engle-Granger test is simple. We perform a linear regression between the two asset prices and check if the residual is stationary using the Augmented Dick-Fuller (ADF) test.

Reference for the test and related concepts: 1) https://hudsonthames.org/an-introduction-to-cointegration/

2) https://arxiv.org/pdf/2211.07080

In [5]:
```
_, pvalue, _ = coint(data_filled['banknifty'],data_filled['nifty'])

if pvalue <0.05:
    print("The time series are cointegrated.")
else:
    print("The time series are not cointegrated.")
```

```
The time series are cointegrated.
```

Base model - A very simple z-score based model to give trading signals

In [6]:
```
data_filled['spread'] = data_filled['banknifty'] - data_filled['nifty']
```

In [7]:
```
data_filled['z_score'] = zscore(data_filled['spread'])

data_filled.head()
```

Out[7]:

time	banknifty	nifty	tte	spread	z_score
2021-01-01 09:15:00	0.286058	0.199729	27	0.086329	0.543747
2021-01-01 09:16:00	0.285381	0.200433	27	0.084948	0.491736
2021-01-01 09:17:00	0.284233	0.200004	27	0.084229	0.464628
2021-01-01 09:18:00	0.286104	0.199860	27	0.086244	0.540526
2021-01-01 09:19:00	0.285539	0.198951	27	0.086588	0.553505

Some important assumptions: 1) We will make two column one for long trade i.e we are going long on the spread if the value of the spread goes below a arbiratry threshold from mean and one

for short trade i.e short on spread if value of spread goes above a arbiratry threshold from mean. 2) We will not make multiple long or short trades at a time. 3) We will exit the trade if it goes beyond 5 days i.e 1875 minutes 3) We also assume that P/L for a trade will be difference between the P/L when we bought an

At each point of trade we are assuming that we know the entire data i.e all the datapoints from where we calculate the mean, this is not at all possible in actual market

In [ ]:
```
data_filled.insert(loc=5,column='signal_1',value=0)
data_filled.insert(loc=6,column='signal_2',value=0)


i=0
while i<len(data_filled)-30:
    #print(i)
    count=1
    if data_filled.iloc[i,4]>=1.5:
        data_filled.iloc[i,5]=2
        while ((i+count)<(len(data_filled)-1) and data_filled.iloc[i+count,4
            #print(count)
            data_filled.iloc[i+count,5]=1
            count=count+1
        data_filled.iloc[(i+count),5]=3
    if count>1875:
        i = i+count+1
    else:
        i=i+count

i=0
while i<len(data_filled)-30:
    #print(i)
    count=1
    if data_filled.iloc[i,4]<=-1.5:
        data_filled.iloc[i,6]=2
        while ((i+count)<(len(data_filled)-1) and data_filled.iloc[i+count,4
            #print(count)
            data_filled.iloc[i+count,6]=1
            count=count+1
        data_filled.iloc[(i+count),6]=3

    if count>1875:
        i = i+count+1
    else:
        i=i+count


data_filled
```

In [ ]:
```
plt.figure(figsize=(30,15))
plt.plot(data_filled['signal_1'],'g')
plt.plot(data_filled['signal_2'],'r')
plt.plot(data_filled['z_score'],'b')
```

In [ ]:
```
#Calculating P/L, when the signal is 2 it gives us the value when we entered
#Similarly calculating P/L when the signal is 3 gives us the value when we e
data_filled.insert(loc=7,column='P/L',value=0)
data_filled.loc[data_filled['signal_1']==2,'P/L']=data_filled['spread']*(dat
data_filled.loc[data_filled['signal_1']==3,'P/L']=data_filled['spread']*(dat
```

```
data_filled.loc[data_filled['signal_2']==2,'P/L']=data_filled['spread']*(dat
data_filled.loc[data_filled['signal_2']==3,'P/L']=data_filled['spread']*(dat
data_filled
```

In [ ]:
```
count_Entry_1 = data_filled['signal_1'].value_counts()[2]
count_Exit_1 = data_filled['signal_1'].value_counts()[3]

count_Entry_2 = data_filled['signal_2'].value_counts()[2]
count_Exit_2 = data_filled['signal_2'].value_counts()[3]

print(count_Entry_1,count_Exit_1,count_Entry_2,count_Exit_2)
```

In [ ]:
```
entry_1=data_filled.loc[data_filled['signal_1']==2,'P/L'].sum()
exit_1=data_filled.loc[data_filled['signal_1']==3,'P/L'].sum()
#this is shorting the spread hence entry - exit
return_1=(entry_1-exit_1)
print(return_1)
```

In [ ]:
```
entry_2=data_filled.loc[data_filled['signal_2']==2,'P/L'].sum()
exit_2=data_filled.loc[data_filled['signal_2']==3,'P/L'].sum()
#this is long on the spread hence exit - entry
return_2=(exit_2-entry_2)
print(return_2)
```

In [ ]:
```
total_trades = count_Entry_1 + count_Entry_2
avg_pnl = (return_1 + return_2)/total_trades # Average PNL
avg_pnl
```

In [ ]:
```
returns = []
for i in range(len(data_filled)):
    if data_filled.iloc[i,5]==2:
        ent = data_filled.iloc[i,7]
        count = 1
        while data_filled.iloc[i+count,5]!=3 and count<1805:
            count+=1
        ext = data_filled.iloc[i+count,7]
        returns.append(ent-ext)
    if data_filled.iloc[i,6]==2:
        ent = data_filled.iloc[i,7]
        count = 1
        while data_filled.iloc[i+count,6]!=3:
            count+=1
        ext = data_filled.iloc[i+count,7]
        returns.append(ext-ent)
returns
```

In [ ]:
```
returns_a = np.array(returns)
sharpe_ratio = (returns_a.mean()-.106)/returns_a.std()
sharpe_ratio
```

In [ ]:
```
def max_drawdown(returns):
    cumulative_returns = np.cumprod(1 + returns)
    peak = np.maximum.accumulate(cumulative_returns)
    drawdown = (cumulative_returns - peak) / peak
    max_drawdown = np.abs(np.min(drawdown))
    return max_drawdown
```

In [ ]:
```
maximum_drawdown = max_drawdown(returns_a)
print("Maximum Drawdown:", maximum_drawdown)
```

In [ ]:

In [ ]:

Goal --> Stabilise Spread

Z-Score Strategy,

$$\text{(BankNifty - Nifty)} \sim \text{f(Nifty)} = \alpha + (\beta - 1)\text{nifty} + \mu$$

Mean(Spread) is not constant for all observations but depends on the variable Nifty

New Strategy,

$$\text{Spread} = (\text{BankNifty} - \beta.\text{Nifty}) - \alpha = \mu$$

$$\text{Mean}(\text{Spread}) = 0; \text{Var}(\text{Spread}) = \sigma^2$$

Assumption -> $E[\mu] = 0$ (from normal equations of OLS)

A good estimator for $\mu \sim \hat{\mu}$(residual)

Approach 1 - Regression model without train/test spilt with some issues

In [ ]:

```
data_model=pd.read_parquet("data.parquet")
data_model = data_model.between_time('09:15', '15:30').copy()
missing_values = data_model.isnull().sum()
if missing_values.sum() > 0:
    data_m1 = data_model.fillna(method='ffill')
else:
    data_m1 = data_model
data_m1
```

In [ ]:

```
# Add a constant term to the independent variable (x)
x_with_const = sm.add_constant(data_m1['nifty'])

# Fit the linear regression model
model = sm.OLS(data_m1['banknifty'], x_with_const).fit()

# Get the predicted values (fitted values)
predicted_values = model.predict(x_with_const)

# Calculate residuals
data_m1['residuals'] = data_m1['banknifty'] - predicted_values

data_m1
```

In [ ]:

```
plt.plot(data_m1['residuals'])
```

In [ ]:

```
data_m1['z_score'] = zscore(data_m1['residuals'])
plt.plot(data_m1['z_score'])
data_m1
```

In [ ]:
```
data_m1.insert(loc=5,column='signal_1',value=0)
data_m1.insert(loc=6,column='signal_2',value=0)


i=0
while i<len(data_m1)-30:
    #print(i)
    count=1
    if data_m1.iloc[i,4]>=1.5:
        data_m1.iloc[i,5]=2
        while ((i+count)<(len(data_m1)-1) and data_m1.iloc[i+count,4]>=0.5 a
            #print(count)
            data_m1.iloc[i+count,5]=1
            count=count+1
        data_m1.iloc[(i+count),5]=3
    if count>1875:
        i = i+count+1
    else:
        i=i+count

i=0
while i<len(data_m1)-30:
    #print(i)
    count=1
    if data_m1.iloc[i,4]<=-1.5:
        data_m1.iloc[i,6]=2
        while ((i+count)<(len(data_m1)-1) and data_m1.iloc[i+count,4]<=-0.5
            #print(count)
            data_m1.iloc[i+count,6]=1
            count=count+1
        data_m1.iloc[(i+count),6]=3

    if count>1875:
        i = i+count+1
    else:
        i=i+count


data_m1
```

In [ ]:
```
plt.figure(figsize=(30,15))
plt.plot(data_m1['signal_1'],'g')
plt.plot(data_m1['signal_2'],'r')
plt.plot(data_m1['z_score'],'b')
```

In [ ]:
```
data_m1.insert(loc=7,column='P/L',value=0)
data_m1.loc[data_m1['signal_1']==2,'P/L']=data_m1['residuals']*(data_filled
data_m1.loc[data_m1['signal_1']==3,'P/L']=data_m1['residuals']*(data_filled
data_m1.loc[data_m1['signal_2']==2,'P/L']=data_m1['residuals']*(data_filled
data_m1.loc[data_m1['signal_2']==3,'P/L']=data_m1['residuals']*(data_filled
data_m1
```

In [ ]:
```
count_Entry_1 = data_m1['signal_1'].value_counts()[2]
count_Exit_1 = data_m1['signal_1'].value_counts()[3]

count_Entry_2 = data_m1['signal_2'].value_counts()[2]
count_Exit_2 = data_m1['signal_2'].value_counts()[3]

print(count_Entry_1,count_Exit_1,count_Entry_2,count_Exit_2)
```

In [ ]:
```
entry_1=data_m1.loc[data_m1['signal_1']==2,'P/L'].sum()
exit_1=data_m1.loc[data_m1['signal_1']==3,'P/L'].sum()
#this is shorting the spread hence entry - exit
return_1=(entry_1-exit_1)
print(return_1)
```

In [ ]:
```
entry_2=data_m1.loc[data_m1['signal_2']==2,'P/L'].sum()
exit_2=data_m1.loc[data_m1['signal_2']==3,'P/L'].sum()
#this is long on the spread hence exit - entry
return_2=(exit_2-entry_2)
print(return_2)
```

In [ ]:
```
total_trades = count_Entry_1 + count_Entry_2
avg_pnl = (return_1 + return_2)/total_trades # Average PNL
avg_pnl
```

In [ ]:
```
returns = []
for i in range(len(data_m1)):
    if data_m1.iloc[i,5]==2:
        ent = data_m1.iloc[i,7]
        count = 1
        while data_m1.iloc[i+count,5]!=3 and count<1805:
            count+=1
        ext = data_m1.iloc[i+count,7]
        returns.append(ent-ext)
    if data_m1.iloc[i,6]==2:
        ent = data_m1.iloc[i,7]
        count = 1
        while data_m1.iloc[i+count,6]!=3:
            count+=1
        ext = data_m1.iloc[i+count,7]
        returns.append(ext-ent)
returns
```

In [ ]:
```
returns_a = np.array(returns)
sharpe_ratio = (returns_a.mean()-.106)/returns_a.std()
sharpe_ratio
```

In [ ]:
```
def max_drawdown(returns):
    cumulative_returns = np.cumprod(1 + returns)
    peak = np.maximum.accumulate(cumulative_returns)
    drawdown = (cumulative_returns - peak) / peak
    max_drawdown = np.abs(np.min(drawdown))
    return max_drawdown

maximum_drawdown = max_drawdown(returns_a)
print("Maximum Drawdown:", maximum_drawdown)
```

Drawbacks and issues with the above model:

In [ ]:

In [ ]:

We will try to address some issues now in above model and move little towards a real case scenario

Approach 2 - Regression model with train/test spilt

In [ ]:
```
# Generating data and handling the missing values
data_3=pd.read_parquet("data.parquet")
data_3 = data_model.between_time('09:15', '15:30').copy()
missing_values = data.isnull().sum()
if missing_values.sum() > 0:
    data_reg = data_model.fillna(method='ffill')
else:
    data_reg = data_model
data_reg
```

In [ ]:
```
# Spilting the data into traaining data and testing data.
# Reason: As at particular time of trade we don't know the future values, he
# training data and then use them to produce signals on the test data.

train= data_reg.loc[:'2021-10-29 15:15:00'].copy()
test= data_reg.loc['2022-01-03 09:15:00':].copy()

#Rolling regression can also be used instead of spilting data
```

In [ ]:
```
# Now we first run a regression on training data to get the coefficients, ar
# to generate signals based on the z_score of spread calculated from this da

# Note: Here there are lot of assumptions in linear regression we are taking
# or "inefficient" estimates of parameters


# Add a constant term to the independent variable (x)
x_with_const = sm.add_constant(train['nifty'])

# Fit the linear regression model
model = sm.OLS(train['banknifty'], x_with_const).fit()

params = model.params

# Also we will need mean and std of residuals here to use them in Z-score o

# Get the predicted values (fitted values)
predicted_values = model.predict(x_with_const)

# Calculate residuals
train['residuals'] = train['banknifty'] - predicted_values
```

In [ ]:
```
# Storing the parameters we need:
residual_mean = train['residuals'].mean()
residual_std = train['residuals'].std()

#Regression parameters
params
```

In [ ]:
```
print(residual_mean, residual_std)
```

In [ ]:
```
alpha = params.const
beta = params.nifty
```

In [ ]:
```
# We have got our parameter from train data, let us calculate spread on tes
# residual_spread = y - y_hat = banknifty - (aplha + beta*nifty)

test['residual_spread'] = test['banknifty'] - alpha - beta*test['nifty']
test
```

In [ ]:
```
plt.plot(test['residual_spread'])
```

In [ ]:
```
# CALCULATING Z-Score with help of mean and std from training dataset and re
# obtained with the help of "estimated" parameters from training dataset

test['Z_Score'] = (test['residual_spread'] - residual_mean)/residual_std
test
```

In [ ]:
```
plt.plot(test['Z_Score'])
```

In the above graph we can see and say that our spread or the zscore that we got is not stationary as we want because it is not as much oscilating around 0 and hence the mean or the parameters of it may be changing wrt time, and hence this method might not give accurate prediction about when to long or short the spread and hence we might need to imporve on this model maybe by calculating rolling z_score which updates it means periodically and also perform rolling regression that will update its parameter to give more stable residuals!

There are various other methods as well which can be used, for now let us proceed with this Z_score and will comeback later to other methods so that we can refine and generate more good trading signals

In [ ]:
```
# Now we got our Z-scores for our resiudal spread we can decided the thhresh
# Here we assume that we can seprately place our long ansd short trade and i
# new trade of same type can only be initiated if we exit our first positio

# Before we decide on how are we trading let us create two columns in our te
# One of us which will store the long positions and other short positions.

# Here it is important to understand the notion behind going long on spread
# lies at heart of pair trading strategies


test.insert(loc=5,column='signal_1',value=0)
test.insert(loc=6,column='signal_2',value=0)


# Now let us understand some things on trades that we are entering and what
```

Here we decide our threshold for the diversion from mean based (Zscores) based on various factors.

But For now let us ignore the factors and arbitarily take some values to enter a trade and exit a trade.

So when the Z-score is greater than 1.5 (assuming) we will enter a short the spread position that is we short the bank_nifty and go long on nifty and as soon as this falls again below 0.5 (assumption) we exit the trade. Theory lies at core of pairs trading startegy.

Very Important point to note : We will also exit the trade if we are in the trade for more than 5 days i.e due to medium frequency startegy but in real case we may benefit from staying if the spread still has large deviation from mean.

Let us signal 2 if we are entering a trade as we get a signal and 3 when we exit, 1 is for when we are in the trade.

Also when we see P/L it can be seen that we won't be in a long and short position simulteanously

In [ ]:
```
i=0
while i<len(test)-30:
    #print(i)
    count=1
    if test.iloc[i,4]>=1.5:
        test.iloc[i,5]=2
        while ((i+count)<(len(test)-1) and test.iloc[i+count,4]>=0.5 and co
            #print(count)
            test.iloc[i+count,5]=1
            count=count+1
        test.iloc[(i+count),5]=3
    if count>1875:
        i = i+count+1
    else:
        i=i+count

i=0
while i<len(test)-30:
    #print(i)
    count=1
    if test.iloc[i,4]<=-1.5:
        test.iloc[i,6]=2
        while ((i+count)<(len(test)-1) and test.iloc[i+count,4]<=-0.5 and c
            #print(count)
            test.iloc[i+count,6]=1
            count=count+1
        test.iloc[(i+count),6]=3

    if count>1875:
        i = i+count+1
    else:
        i=i+count


test
```

Also note we have decided the signals based on Zscore which are actually just taking into account the Z_Score at that time,

there are various other startegies where we can build a momentum based startegy or some machine learning processes to create signals, for now let us proceed with this simple startegy and comeback later to other.

In [ ]:
```
# SHOWS ARE WHAT TIME WE ARE ENTERING THE TRADE AND EXITING IN TRADE AND IF
plt.plot(test['Z_Score'],'b')
plt.plot(test['signal_1'],'r')
plt.plot(test['signal_2'],'g')
```

In [ ]:
```
# let us also see number of trades we took for given time-period of test dat
count_Entry_short = test['signal_1'].value_counts()[2]
count_Exit_short = test['signal_1'].value_counts()[3]

count_Entry_long = test['signal_2'].value_counts()[2]
count_Exit_long = test['signal_2'].value_counts()[3]

print(count_Entry_short,count_Exit_short,count_Entry_long,count_Exit_long)
```

In [ ]:
```
# We calculate P/L for each entry and exit, and then for net P/L we take di

test.insert(loc=7,column='P/L',value=0)
test.loc[test['signal_1']==2,'P/L']=test['residual_spread']*((test['tte'])**
test.loc[test['signal_2']==2,'P/L']=test['residual_spread']*((test['tte'])**
test.loc[test['signal_1']==3,'P/L']=test['residual_spread']*((test['tte'])**
test.loc[test['signal_2']==3,'P/L']=test['residual_spread']*((test['tte'])**
test
```

In [ ]:
```
# Now let us calculate net_p/l first for both long and short positions:

entry_1=test.loc[test['signal_1']==2,'P/L'].sum()
exit_1=test.loc[test['signal_1']==3,'P/L'].sum()
#this is shorting the spread hence entry - exit
return_1=(entry_1-exit_1)
print(return_1)

entry_2=test.loc[test['signal_2']==2,'P/L'].sum()
exit_2=test.loc[test['signal_2']==3,'P/L'].sum()
#this is long on the spread hence exit - entry
return_2=(exit_2-entry_2)
print(return_2)
```

In [ ]:
```
# Now for comparing let us calculate average P/L per trade so that we can co

total_trades = count_Entry_long + count_Entry_short
avg_pnl = (return_1 + return_2)/total_trades # Average PNL
avg_pnl
```

In [ ]:
```
# We also for sharp ratio make a array and store returns for each trade exec

returns = []
for i in range(len(test)):
    if test.iloc[i,5]==2:
        ent = test.iloc[i,7]
        count = 1
        while test.iloc[i+count,5]!=3 and count<1805:
            count+=1
        ext = test.iloc[i+count,7]
        returns.append(ent-ext)
    if data_m1.iloc[i,6]==2:
        ent = data_m1.iloc[i,7]
```

```
        count = 1
        while data_m1.iloc[i+count,6]!=3:
            count+=1
        ext = data_m1.iloc[i+count,7]
        returns.append(ext-ent)
returns
```

In [ ]:
```
returns_a = np.array(returns)
sharpe_ratio = (returns_a.mean()-.106)/returns_a.std()
sharpe_ratio
```

In [ ]:
```
# Calculating the maximum drawdown from an array of returns.
def max_drawdown(returns):
    cumulative_returns = np.cumprod(1 + returns)
    peak = np.maximum.accumulate(cumulative_returns)
    drawdown = (cumulative_returns - peak) / peak
    max_drawdown = np.abs(np.min(drawdown))
    return max_drawdown

maximum_drawdown = max_drawdown(returns_a)
print("Maximum Drawdown:", maximum_drawdown)
```

In [ ]:

Approach 3 - Using a rolling regression model as mentioned in above model.

Lets hope we imporve our results!

In [ ]:
```
# Define the window size for the rolling regression
window_size = 60 # Example window size, adjust based on analysis needs
# Initialize the series to store the rolling regression residuals
rolling_residuals = pd.Series(index=data_m1.index)
# Perform rolling regression and calculate residuals
for start in range(len(data_filled) - window_size):
    end = start + window_size
    X = sm.add_constant(data_m1['nifty'][start:end])
    # Predictor variable with constant
    y = data_m1['banknifty'][start:end]
    # Response variable
    model = OLS(y, X).fit()
    predictions = model.predict(X)
    residuals = y - predictions
    rolling_residuals[end:end+1] = residuals.iloc[-1]
    # Fill the initial part of the series where rolling residuals are not av
    rolling_residuals = rolling_residuals.fillna(method='bfill')
    # Update the dataset with the rolling residuals
    data_m1['rolling_residual_spread'] = rolling_residuals
```

In [ ]:
```
plt.plot(data_m1['rolling_residual_spread'])
```

In [ ]:
```
data_m1
```

In [ ]:
```
data_m1['z_score_rolling'] = zscore(data_m1['rolling_residual_spread'])
data_m1
```

In [ ]:
```
plt.plot(data_m1['z_score_rolling'],'g')
```

In [ ]:
```
data_m1.insert(loc=10,column='signal_11',value=0)
data_m1.insert(loc=11,column='signal_22',value=0)


i=0
while i<len(data_m1)-30:
    #print(i)
    count=1
    if data_m1.iloc[i,9]>=1.5:
        data_m1.iloc[i,10]=2
        while ((i+count)<(len(data_m1)-1) and data_m1.iloc[i+count,9]>=0.5 a
            #print(count)
            data_m1.iloc[i+count,10]=1
            count=count+1
        data_m1.iloc[(i+count),10]=3
    if count>1875:
        i = i+count+1
    else:
        i=i+count

i=0
while i<len(data_m1)-30:
    #print(i)
    count=1
    if data_m1.iloc[i,9]<=-1.5:
        data_m1.iloc[i,11]=2
        while ((i+count)<(len(data_m1)-1) and data_m1.iloc[i+count,9]<=-0.5
            #print(count)
            data_m1.iloc[i+count,11]=1
            count=count+1
        data_m1.iloc[(i+count),11]=3

    if count>1875:
        i = i+count+1
    else:
        i=i+count



data_m1
```

In [ ]:
```
count_Entry_11 = data_m1['signal_11'].value_counts()[2]
count_Exit_11 = data_m1['signal_11'].value_counts()[3]

count_Entry_22 = data_m1['signal_22'].value_counts()[2]
count_Exit_22 = data_m1['signal_22'].value_counts()[3]

print(count_Entry_1,count_Exit_1,count_Entry_2,count_Exit_2)
```

In [ ]:
```
data_m1.insert(loc=12,column='P/L_2',value=0)
data_m1.loc[data_m1['signal_11']==2,'P/L_2']=data_m1['rolling_residual_sprea
data_m1.loc[data_m1['signal_22']==2,'P/L_2']=data_m1['rolling_residual_sprea
data_m1.loc[data_m1['signal_11']==3,'P/L_2']=data_m1['rolling_residual_sprea
data_m1.loc[data_m1['signal_22']==3,'P/L_2']=data_m1['rolling_residual_sprea
data_m1
```

In [ ]:
```
entry_1=data_m1.loc[data_m1['signal_11']==2,'P/L_2'].sum()
exit_1=data_m1.loc[data_m1['signal_11']==3,'P/L_2'].sum()
#this is shorting the spread hence entry - exit
return_1=(entry_1-exit_1)
print(return_1)

entry_2=data_m1.loc[data_m1['signal_22']==2,'P/L_2'].sum()
```

```
exit_2=data_m1.loc[data_m1['signal_22']==3,'P/L_2'].sum()
#this is long on the spread hence exit - entry
return_2=(exit_2-entry_2)
print(return_2)

total_trades = count_Entry_1 + count_Entry_2
avg_pnl = (return_1 + return_2)/total_trades # Average PNL
avg_pnl
```

In [ ]:
```
returns = []
for i in range(len(data_m1)):
    if data_m1.iloc[i,10]==2:
        ent = data_m1.iloc[i,12]
        count = 1
        while data_m1.iloc[i+count,10]!=3 and count<1805:
            count+=1
        ext = data_m1.iloc[i+count,12]
        returns.append(ent-ext)
    if data_m1.iloc[i,11]==2:
        ent = data_m1.iloc[i,12]
        count = 1
        while data_m1.iloc[i+count,11]!=3:
            count+=1
        ext = data_m1.iloc[i+count,12]
        returns.append(ext-ent)
returns

returns_a = np.array(returns)
sharpe_ratio = (returns_a.mean()-.106)/returns_a.std()
print(sharpe_ratio)

def max_drawdown(returns):
    """
    Calculate the maximum drawdown from an array of returns.

    Parameters:
    returns (np.array): Array of returns.

    Returns:
    float: Maximum drawdown.
    """
    cumulative_returns = np.cumprod(1 + returns)
    peak = np.maximum.accumulate(cumulative_returns)
    drawdown = (cumulative_returns - peak) / peak
    max_drawdown = np.abs(np.min(drawdown))
    return max_drawdown

maximum_drawdown = max_drawdown(returns_a)
print("Maximum Drawdown:", maximum_drawdown)
```

In [ ]:

Other Approaches and Conclusions

There are many other improvements in above 3 approaches that can be done and many different models that can be implemented.

If we try to break pairs trading as a whole into steps and then modify processes at each step that will optimize the results that we want, it can result into a robust strategy.

Though here we have not done optimization, it can be done by backtesting for several choice parameters.

Below I have specefied two approach or slight changes that can be brought in to produce some more efficient signals. Comparsion with above is not possible with the following models as they are based on various simplying assumptions, so they are just for information purpose. Important thing to note is the Mathematical approach behind one approach (Principal Component Analysis) and other one is based on chaning the way we create signals which is based on a momentum startegy (RSI)

In [ ]:

```

```

PCA Approach

In [ ]:

```
data_3=pd.read_parquet("data.parquet")
data_3 = data_model.between_time('09:15', '15:30').copy()
missing_values = data.isnull().sum()
if missing_values.sum() > 0:
    data_pca = data_model.fillna(method='ffill')
else:
    data_pca = data_model
data_pca
```

In [ ]:

```
# To apply PCA first we bring down both to same scale i.e normalise the data

# Calculate the IV spread between Bank Nifty and Nifty _ we are taking sprea
# we used earlier and using the resiudal spread instead of normal spread.

train_pca= data_pca.loc[:'2021-10-29 15:15:00'].copy()
test_pca= data_pca.loc['2022-01-03 09:15:00':].copy()

import statsmodels.api as sm

# Add a constant term to the independent variable (x)
x_with_const = sm.add_constant(train_pca['nifty'])

# Fit the linear regression model
model = sm.OLS(train_pca['banknifty'], x_with_const).fit()

params = model.params

alpha = params.const
beta = params.nifty

test_pca['residual_spread'] = test_pca['banknifty'] - alpha - beta*test_pca
test_pca
```

In [ ]:
```
# Standardize the IV Spread for PCA
scaler = StandardScaler()
iv_spread_scaled = scaler.fit_transform(test_pca[['residual_spread']])
```

In [ ]:
```
# Apply PCA to the scaled IV Spread
pca = PCA(n_components=1)  # Using 1 component since we're focusing on the 
pca.fit(iv_spread_scaled)
```

In [ ]:
```
# Transform the IV spread data to the principal component space
iv_spread_pca = pca.transform(iv_spread_scaled)
```

In [ ]:
```
# Add the principal component scores to the dataframe
test_pca['iv_spread_pc1'] = iv_spread_pca[:, 0]
test_pca
```

In [ ]:
```
### Explained Variance Ratio
explained_variance_ratio = pca.explained_variance_ratio_

### Now, let's prepare to generate trading signals based on the first princ
### A simple strategy: Buy (1) when the score of PC1 increases, Sell (-1) wh
test_pca['signal'] = test_pca['iv_spread_pc1'].diff().apply(lambda x: 1 if x
```

In [ ]:
```
# Calculate the P/L for trading signals, assuming 'tte' is the time to expi
# P/L = Spread * (TTE)^0.7 for trades
# Note: This simplistic model does not account for bid-ask spread, transact
test_pca['P/L'] = test_pca.apply(lambda row: row['residual_spread'] * (row[
test_pca
```

In [ ]:
```
# First let us see how many times we have gone long and how many times shor
print(test_pca['signal'].value_counts()[1])
print(test_pca['signal'].value_counts()[-1])
```

In [ ]:
```
# Calculate cumulative P/L as +1 in buying -1 is selling and we are ignoring
# So net P/L or the total P/L will be +1 x P/L + -1 x P/L
long_net_P_L= test_pca.loc[test_pca['signal']==1,'P/L'].sum()

short_net_P_L=test_pca.loc[test_pca['signal']==-1,'P/L'].sum()

#this is long on the spread hence exit - entry
Net_PL = long_net_P_L - short_net_P_L
print(Net_PL)
```

In the above case main aim is to show how PCA can be used to generate signals rather than comparing it to our base model. We are ignoring the closing of trades here and assuming that we can long and short as much as we can and netPL is calulated which will differ in real case beacause positions need to be closed.

Also we have directly performed the PCA analysis with help of libaries but the underlying maths is as follows:

In short we are identfying major factors that are affecting our spread (Bank nifty - Nifty), and by analysing them we are

creating signals. Reason to use is to reduce the complexity of data.

The method to calculate this component is based on matrix algebra, for reference: Link:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

SSSSSSSSSSSSSSSSSSSSSSSSS

Now let us refine our approach to generate signals by the Z-Scores

We will use a momentum indicator generally used to asses price, but we also can asses Z-scores obtained with it, which is RSI (Relative Strength Index).

RSI here - measures the speed and change of Z-score, it tells how significantly spread deviates from its mean.

In short we are checking with RSI if Z-score is peaking or botoming out, which then will potentially signal a reversal!!

In [ ]:
```
# Function to calculate RSI
def rsi(series, period=14):
    delta = series.diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=period).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=period).mean()
    rs = gain / loss
    return 100 - (100 / (1 + rs))

test['z_score_rsi'] = rsi(test['Z_Score'], period=14)
test
```

In [ ]:
```
# Handling some null values generated due to rolling nature of rsi:

test['z_score_rsi'] = test['z_score_rsi'].fillna(method='bfill')
test
```

In [ ]:
```
test['z_score_rsi'].describe()
```

In [ ]:
```
# Long trades logic
i = 0
```

```
while i < len(test)-30:
    count = 1
    if test.iloc[i]['z_score_rsi'] <= lower_threshold:
        test.iloc[i]['long_signal_rsi'] = 2  # Enter long trade
        while (i + count) < len(test)-1 and test.iloc[i + count]['z_score_rs
            test.iloc[i + count]['long_signal_rsi'] = 1  # Stay in long trad
            count += 1
        test.iloc[i + count]['long_signal_rsi'] = 3  # Exit long trade
    if count>1875:
        i = i+count+1
    else:
        i=i+count
```

In [ ]:

```
test['long_signal_rsi'].describe()
```

In [ ]:

```
count_Entry_long_rsi = test['long_signal_rsi'].value_counts()[2]
count_Exit_long_rsi = test['long_signal_rsi'].value_counts()[3]
```

In [ ]:

In [ ]:

In [ ]:

```
# We calculate P/L for each entry and exit, and then for net P/L we take di

test.insert(loc=7,column='P/L_rsi',value=0)
test.loc[test['short_signal_rsi']==2,'P/L_rsi']=test['residual_spread']*((te
test.loc[test['short_signal_rsi']==2,'P/L_rsi']=test['residual_spread']*((te
test.loc[test['long_signal_rsi']==3,'P/L_rsi']=test['residual_spread']*((tes
test.loc[test['long_signal_rsi']==3,'P/L_rsi']=test['residual_spread']*((tes
test
```

In [ ]:

```
# Now let us calculate net_p/l first for both long and short positions:

entry_1_rsi=test.loc[test['short_signal_rsi']==2,'P/L_rsi'].sum()
exit_1_rsi=test.loc[test['short_signal_rsi']==3,'P/L_rsi'].sum()
#this is shorting the spread hence entry - exit
return_1_rsi=(entry_1_rsi-exit_1_rsi)
print(return_1_rsi)

entry_2_rsi=test.loc[test['long_signal_rsi']==2,'P/L_rsi'].sum()
exit_2_rsi=test.loc[test['long_signal_rsi']==3,'P/L_rsi'].sum()
#this is long on the spread hence exit - entry
return_2_rsi=(exit_2_rsi-entry_2_rsi)
print(return_2_rsi)
```

In [ ]: