we implement the carry trade strategy on USD and Canadian Dollar on data from Jan 2000 to Jan 2023

In [13]:

```
import pandas as pd
import numpy as np
```

In [4]:

```
# upload database - monthly yield

data = pd.read_csv("Hw3_Interest_Rates.csv")

# change date to datetime and make it as index
data['date'] = pd.to_datetime(data['date'])

data.set_index('date', inplace = True)
data
```

Out[4]:

date	United States	Euro Area	Canada	Australia	Switzerland	United Kingdom	Japan	Norway	
2000-02-01	0.004875	0.002901	0.004333	0.004709	0.002012	0.004985	NaN	0.004781	(
2000-03-01	0.004978	0.003070	0.004395	0.004781	0.002349	0.004983	NaN	0.004868	(
2000-04-01	0.005088	0.003214	0.004482	0.004899	0.002663	0.005036	NaN	0.005049	(
2000-05-01	0.005427	0.003564	0.004770	0.005112	0.002596	0.005048	NaN	0.005262	(
2000-06-01	0.005442	0.003676	0.004794	0.005049	0.002787	0.004980	NaN	0.005435	(
...	...	...	...	...	...	...	...	...	
2022-09-01	0.002636	0.000839	0.003106	0.002271	NaN	0.002393	-0.000017	0.002418	(
2022-10-01	0.003153	0.001182	0.003459	0.002426	NaN	0.002782	-0.000013	0.002734	(
2022-11-01	0.003643	0.001508	0.003572	0.002515	NaN	0.002887	-0.000013	0.002879	(
2022-12-01	0.003683	0.001706	0.003715	0.002604	NaN	0.003097	-0.000013	0.002677	(
2023-01-01	0.003763	0.001933	0.003886	0.002725	NaN	0.003282	NaN	0.002685	(

276 rows × 10 columns

In [5]:

```
# extract yields only for Canada and US

irs = data[['United States', 'Canada']]
irs.fillna(0, inplace = True)
```

```
C:\Users\Kanika\AppData\Local\Temp\ipykernel_25516\1761978037.py:4: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  irs.fillna(0, inplace = True)
```

In [7]:

```
# extract exchange rate data for Canada and US

ex = pd.read_excel("ex_rates.xlsx")
ex['Date'] = pd.to_datetime(ex['Unnamed: 0'])
ex.set_index('Date', inplace = True)
exchange = ex[['Canada']]
```

In [8]:

```
exchange['United States'] = 1
exchange
```

```
C:\Users\Kanika\AppData\Local\Temp\ipykernel_25516\303839078.py:1: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  exchange['United States'] = 1
```

Out[8]:

Date	Canada	United States
2000-02-01	1.4496	1
2000-03-01	1.4494	1
2000-04-01	1.4801	1
2000-05-01	1.4965	1
2000-06-01	1.4806	1
...	...	...
2022-09-01	1.3707	1
2022-10-01	1.3649	1
2022-11-01	1.3508	1
2022-12-01	1.3603	1
2023-01-01	1.3350	1

276 rows × 2 columns

In [9]:

```
# calculate the currencies with the highest and lowest yields each period
maxI = irs.idxmax(axis = 1)
minI = irs.idxmin(axis = 1)
```

In [10]:

```
irs['max yield'] = maxI
irs['min yield'] = minI
irs
```

```
C:\Users\Kanika\AppData\Local\Temp\ipykernel_25516\3411724993.py:1: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  irs['max yield'] = maxI
C:\Users\Kanika\AppData\Local\Temp\ipykernel_25516\3411724993.py:2: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  irs['min yield'] = minI
```

Out[10]:

date	United States	Canada	max yield	min yield
2000-02-01	0.004875	0.004333	United States	Canada
2000-03-01	0.004978	0.004395	United States	Canada
2000-04-01	0.005088	0.004482	United States	Canada
2000-05-01	0.005427	0.004770	United States	Canada
2000-06-01	0.005442	0.004794	United States	Canada
...	...	...	...	...
2022-09-01	0.002636	0.003106	Canada	United States
2022-10-01	0.003153	0.003459	Canada	United States
2022-11-01	0.003643	0.003572	United States	Canada
2022-12-01	0.003683	0.003715	Canada	United States
2023-01-01	0.003763	0.003886	Canada	United States

276 rows × 4 columns

In [11]:

```
yld = pd.DataFrame([maxI, minI]).T
yld.columns = ["High Yield", "Low Yield"]
yld
```

Out[11]:

date	High Yield	Low Yield
2000-02-01	United States	Canada
2000-03-01	United States	Canada
2000-04-01	United States	Canada
2000-05-01	United States	Canada
2000-06-01	United States	Canada
...	...	...
2022-09-01	Canada	United States
2022-10-01	Canada	United States
2022-11-01	United States	Canada
2022-12-01	Canada	United States
2023-01-01	Canada	United States

276 rows × 2 columns

In [14]:

```
# trading strategy: go short of currency with lower interest rate and go long on currenc

# profits

profits = np.array([])
for j in range(len(exchange)-1):

    # identify the country with the highest (long)        # identifies if US has greater
    # and lowest (short) yield
    long = maxI[j]
    short = minI[j]

    # get the exchange rate at t0 and t+1
    # for the short
    sts0 = exchange[short][j]                             # shorts currency with lower y
    sts1 = exchange[short][j+1]

    # get the monthly interest rate
    # for the short
    si = irs[short][j]

    # calculate the amount owed
    owed = 10000*sts0*si/sts1

    # get the exchange rate at t0 and t+1
    # for the long
    stl0 = exchange[long][j]
    stl1 = exchange[long][j+1]

    # get the monthly interest rate for the long
    li = irs[long][j]

    # calculate the ending balance
    balance = 10000*stl0*li/stl1

    # calculate the profit
    profit = balance - owed

    # store the profits
    profits = np.append(profits, profit)

    print(profit)


profits = pd.DataFrame(profits, index = irs.index[:-1], columns = ["Profit"])
```

```
5.421077894553498
6.737543773026594
6.559779278525497
6.059702522100785
6.686190080858822
5.763369614317831
6.64673516262247
6.545148354461595
6.959217135125087
4.939699362779699
5.486677917624917
2.2513333285232306
1.6343207323132063
0.526040465179058
0.6358161004577383
3.937801348761802
4.947707723536382
5.008451039521432
4.168120146660655
```

In [15]:

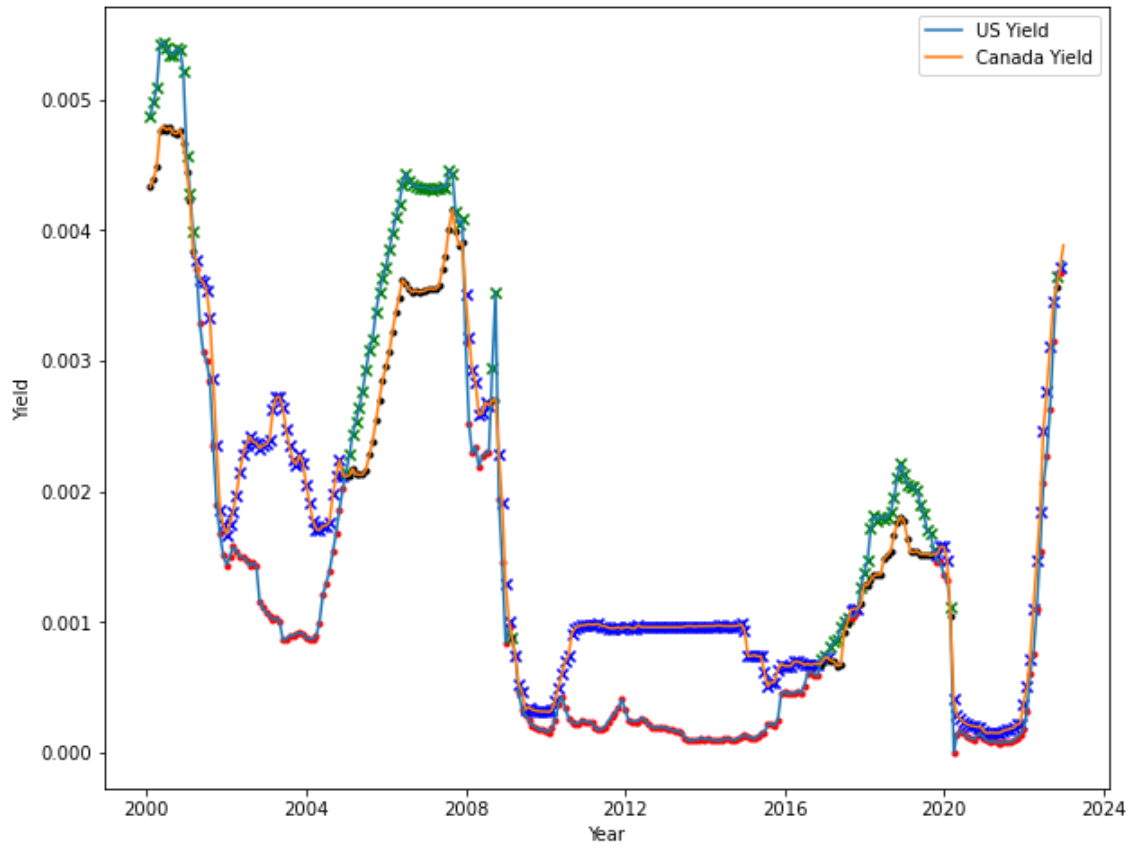
```
# visualize the trading strategy

import matplotlib.pyplot as plt

# Plot the yields of US and Canada
fig, ax = plt.subplots(figsize=(10, 8))
ax.plot(irs["United States"], label='US Yield')
ax.plot(irs['Canada'], label='Canada Yield')

# Add trading signals as dots and 'x'
for j in range(len(profits)):
    if maxI[j] == "United States":
        ax.scatter(irs.index[j], irs.iloc[j]["United States"], color='green', marker='x'
    elif minI[j] == "United States":
        ax.scatter(irs.index[j], irs.iloc[j]["United States"], color='red', marker='.')
    if maxI[j] == "Canada":
        ax.scatter(irs.index[j], irs.iloc[j]["Canada"], color='blue', marker='x')
    elif minI[j] == "Canada":
        ax.scatter(irs.index[j], irs.iloc[j]["Canada"], color='black', marker='.')

# Set plot labels and legend
ax.set_ylabel('Yield')
ax.set_xlabel('Year')
ax.legend()
plt.show()
```



In [16]:

```
# yearly profits
year_profit = profits.Profit.resample('Y').sum()
year_profit = pd.DataFrame(year_profit)
year_profit.head()
```

Out[16]:

	Profit
date	
2000-12-31	67.805141
2001-12-31	36.587541
2002-12-31	88.645102
2003-12-31	181.597026
2004-12-31	74.696160

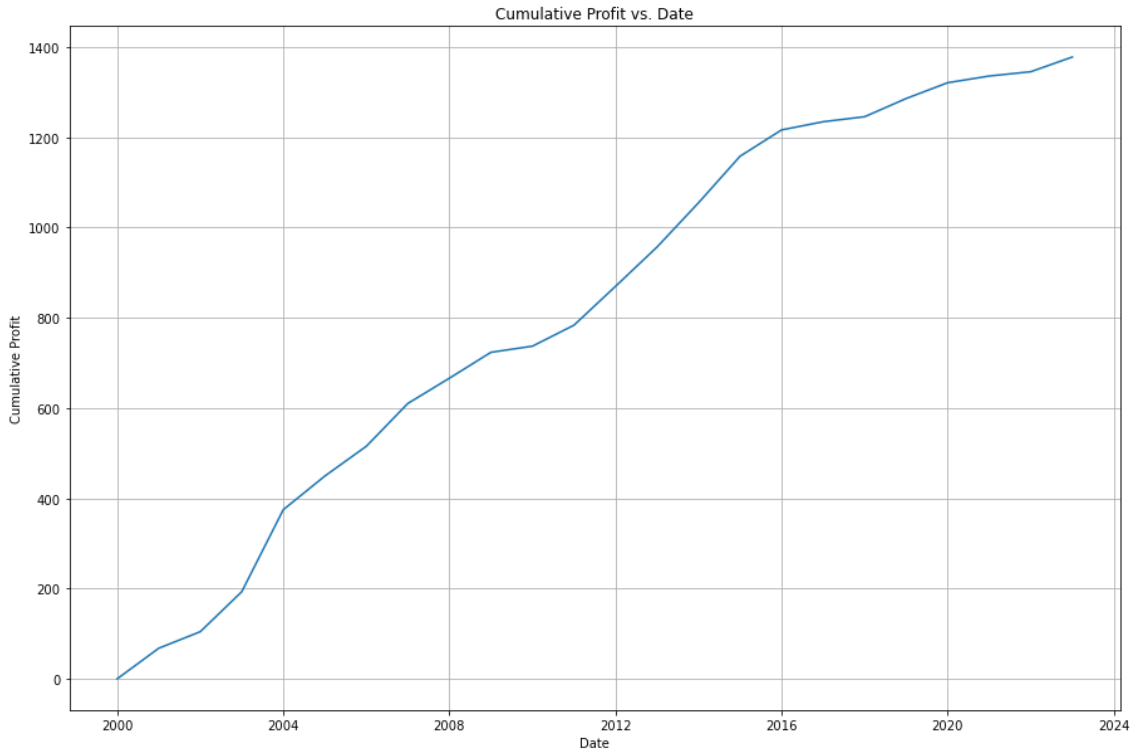
In [17]:

```
# equity curve without reinvestment

cp = profits.resample('Y').sum().cumsum().append(pd.DataFrame(0, index = [pd.to_datetime
plt.figure(figsize = (15, 10))
plt.title('Cumulative Profit vs. Date')
plt.plot(cp)
plt.ylabel('Cumulative Profit')
plt.xlabel('Date')

plt.grid()
```

```
C:\Users\Kanika\AppData\Local\Temp\ipykernel_25516\3299341746.py:3: Futur
eWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  cp = profits.resample('Y').sum().cumsum().append(pd.DataFrame(0, index
= [pd.to_datetime("12/31/1999")], columns = ['Profit'])).sort_index()
```



In [18]:

```
# total rate of return

ror = (profits.Profit.resample('Y').sum()/10000)*100
print('total rate of return', ror)
```

```
total rate of return date
2000-12-31    0.678051
2001-12-31    0.365875
2002-12-31    0.886451
2003-12-31    1.815970
2004-12-31    0.746962
2005-12-31    0.660168
2006-12-31    0.947868
2007-12-31    0.558681
2008-12-31    0.574497
2009-12-31    0.136027
2010-12-31    0.464515
2011-12-31    0.860125
2012-12-31    0.872129
2013-12-31    0.981036
2014-12-31    1.030178
2015-12-31    0.582829
2016-12-31    0.181215
2017-12-31    0.111248
2018-12-31    0.402984
2019-12-31    0.350268
2020-12-31    0.148329
2021-12-31    0.095004
2022-12-31    0.325876
Freq: A-DEC, Name: Profit, dtype: float64
```

In [21]:

```
# annualized return

t = len(ror)
annual_return = ((((ror/100)+1).cumprod()[-1])**(1/t)-1)
annual_return
```

Out[21]:

```
0.005982091621892005
```

In [28]:

```
# alpha and beta

# we first need to get the market rate of return and risk free rate

# get market returns

import yfinance as yf

# Download data for the S&P 500 index
spy = yf.Ticker("^GSPC")
snp = spy.history(period="max")

# Select only the closing price column
snp_close = snp[["Close"]]

snp_close.index = pd.to_datetime(snp_close.index)
snp_close

# convert closing price to returns
snp_close = snp_close.pct_change()

# get yearly data

snp_yearly = snp_close.resample('Y').mean()
snp_yearly.dropna()

# get risk free rate

rf = pd.read_csv('1-year-treasury-rate-yield-chart-2.csv')
rf.set_index('date', inplace=True)
rf.index = pd.to_datetime(rf.index)

# convert into annual data

rf1 = rf.copy()
rf1.resample('Y').mean()
rf1

# merge market free interest rate with market return

snp_rf = pd.merge_asof(snp_yearly, rf1, left_index = True, right_index = True)
snp_rf.dropna()

# merge above dataset with ror dataset

final = pd.merge_asof(snp_rf, ror, left_index = True, right_index = True)
final.dropna()

# calculate excess market return
final['excess market return'] = final['Close'] - final[' value']
final.dropna()
```

Out[28]:

Date	Close	value	Profit	excess market return
2000-12-31 00:00:00-05:00	-0.000327	5.3200	0.678051	-5.320327
2001-12-31 00:00:00-05:00	-0.000472	2.1700	0.365875	-2.170472
2002-12-31 00:00:00-05:00	-0.000922	1.3200	0.886451	-1.320922
2003-12-31 00:00:00-05:00	0.000987	1.2600	1.815970	-1.259013
2004-12-31 00:00:00-05:00	0.000366	2.7500	0.746962	-2.749634
2005-12-31 00:00:00-05:00	0.000138	4.3800	0.660168	-4.379862
2006-12-31 00:00:00-05:00	0.000529	5.0000	0.947868	-4.999471
2007-12-31 00:00:00-05:00	0.000189	3.3400	0.558681	-3.339811
2008-12-31 00:00:00-05:00	-0.001587	0.3700	0.574497	-0.371587
2009-12-31 00:00:00-05:00	0.000983	0.4700	0.136027	-0.469017
2010-12-31 00:00:00-05:00	0.000542	0.2900	0.464515	-0.289458
2011-12-31 00:00:00-05:00	0.000107	0.1200	0.860125	-0.119893
2012-12-31 00:00:00-05:00	0.000536	0.1600	0.872129	-0.159464
2013-12-31 00:00:00-05:00	0.001054	0.1300	0.981036	-0.128946
2014-12-31 00:00:00-05:00	0.000454	0.2500	1.030178	-0.249546
2015-12-31 00:00:00-05:00	0.000019	0.6500	0.582829	-0.649981
2016-12-31 00:00:00-05:00	0.000395	0.8106	0.181215	-0.810205
2017-12-31 00:00:00-05:00	0.000716	1.7316	0.111248	-1.730884
2018-12-31 00:00:00-05:00	-0.000199	2.6300	0.402984	-2.630199
2019-12-31 00:00:00-05:00	0.001038	1.5900	0.350268	-1.588962
2020-12-31 00:00:00-05:00	0.000832	0.1000	0.148329	-0.099168
2021-12-31 00:00:00-05:00	0.000980	0.3900	0.095004	-0.389020
2022-12-31 00:00:00-05:00	-0.000745	4.6400	0.325876	-4.640745
2023-12-31 00:00:00-05:00	0.000905	4.6400	0.325876	-4.639095

In [30]:

```
# run regression

import statsmodels.api as sm

X = sm.add_constant(final['excess market return']).dropna()
y = final['Profit'].dropna()

X_Y = pd.merge_asof(X, y, left_index=True, right_index=True, direction='nearest').dropna

x_y = X_Y[['const', 'excess market return']]

model = sm.OLS(X_Y['Profit'], x_y).fit()
alpha, beta = model.params

print("Alpha: ", alpha)
print("Beta: ", beta)
```

```
Alpha:  0.5986933008162544
Beta:  -0.009146323160839676
```

In [31]:

```
# sharpe ratio

# get excess return

final['excess return'] = final['Profit'] - final[' value']
```

In [32]:

```
# get standard deviation

avg_return = np.mean(final['excess return'])
volatility = np.std(final['excess return'])

# Assume you have the risk-free rate stored in a variable called rf_rate
sharpe_ratio = (avg_return) / volatility

print("Sharpe Ratio: ", sharpe_ratio)
```

```
Sharpe Ratio:  -0.7020453124645395
```

In [33]:

```
# gini coefficient

def GINI_COEF(returns):
    # get the number of periods -> will allow us to calculate the areas
    periods = len(returns)

    # sort values and sum to calculate the lorenz curve
    LorenzCurve = np.cumsum(returns.sort_values(by="Returns"))
    # start from 0
    LorenzCurve = pd.DataFrame({'Returns': [0]}).append(LorenzCurve)
    Line = LorenzCurve.copy()
    # Form the line that encompasses A and B
    Line["Returns"] = np.arange(0, 1 + 1 / periods, 1 / periods) * max(LorenzCurve["Retu

    # calculate the area of A+B
    UpArea = 0
    for i in range(1, len(returns)):
        if not np.isnan(LorenzCurve.iloc[i, 0]):
            UpArea += ((Line.iloc[i, 0] - LorenzCurve.iloc[i, 0] + Line.iloc[i - 1, 0] -
    # calculate the area of A+B+C
    if min(LorenzCurve["Returns"]) < 0:
        AllArea = (np.abs(min(LorenzCurve["Returns"])) * periods) + ((max(LorenzCurve["R
    else:
        AllArea = ((max(LorenzCurve["Returns"]) * periods) / 2)
    gini = UpArea / AllArea
    return gini

returns = final[['Profit']]
returns.columns = ['Returns']

GINI_COEF(returns)
```

```
C:\Users\Kanika\AppData\Local\Temp\ipykernel_25516\2608047756.py:8: Futur
eWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  LorenzCurve = pd.DataFrame({'Returns': [0]}).append(LorenzCurve)
```

Out[33]:

```
-0.09884817577093283
```