Generate Orders from Positions

In [1]:
```
import io
import datetime
import pytz

import pandas as pd
import numpy as np
```

Data

In [13]:
```
UTC_TZ = pytz.timezone("UTC")

def read(s):
    df = pd.read_csv(io.StringIO(s), sep="|")
    df.rename(columns=lambda s: s.strip(), inplace=True)
    for col in df.columns:
        if col in ["dt", "rt"]:
            df[col] = pd.to_datetime(df[col])
        elif df.dtypes[col] == np.dtype("O"):
            df[col] = df[col].apply(lambda s: s.strip())
        else:
            pass
    return df.set_index([c for c in df.columns if c != "value"])['value']
```

1. rt - reference time: when the strategy wants to achieve given target position
2. dt - decision time: when the strategy decided to achieve that position

4ex: Strategy S1 runs at 9am and generates a target position of 10e6 EUR at 9:30am, `dt` will be 9am and `rt` will be 9:30.

In [14]:
```
# Target Positions
tpos = read("""
    rt                  | strategy | asset | dt                  | valu
    2023-08-29 07:00:00z | s1       | PLN   | 2023-08-29 07:00:00z | 4e6
    2023-08-29 16:00:00z | s1       | CZK   | 2023-08-29 07:00:00z | -24e
    2023-08-30 07:00:00z | s1       | PLN   | 2023-08-30 07:00:00z | 8e6
    2023-08-30 16:00:00z | s1       | CZK   | 2023-08-30 07:00:00z | -48e
    2023-08-29 07:00:00z | s2       | EUR   | 2023-08-29 07:00:00z | 1.8e
    2023-08-29 16:00:00z | s2       | AUD   | 2023-08-29 07:00:00z | -1.5
    2023-08-30 07:00:00z | s2       | EUR   | 2023-08-30 07:00:00z | 3.6e
    2023-08-30 16:00:00z | s2       | AUD   | 2023-08-30 07:00:00z | -1e6
    2023-08-29 07:00:00z | s3       | CZK   | 2023-08-29 07:00:00z | 12e6
    2023-08-29 16:00:00z | s3       | CZK   | 2023-08-29 07:00:00z | 18e6
    2023-08-30 07:00:00z | s3       | CZK   | 2023-08-30 07:00:00z | 18e6
    2023-08-30 16:00:00z | s3       | CZK   | 2023-08-30 07:00:00z | 24e6
    """)
```

In [15]:
```
fx_rates = read("""
    rt                  | asset | value
    2023-08-29 07:00:00z | PLN   | 3.934
    2023-08-29 16:00:00z | PLN   | 3.924
```

```
    2023-08-30 07:00:00z | PLN   | 3.914
    2023-08-30 16:00:00z | PLN   | 3.904

    2023-08-29 07:00:00z | CZK   | 23.12
    2023-08-29 16:00:00z | CZK   | 23.08
    2023-08-30 07:00:00z | CZK   | 23.02
    2023-08-30 16:00:00z | CZK   | 23.01

    2023-08-29 07:00:00z | EUR   | 1.116
    2023-08-29 16:00:00z | EUR   | 1.119
    2023-08-30 07:00:00z | EUR   | 1.121
    2023-08-30 16:00:00z | EUR   | 1.122

    2023-08-29 07:00:00z | AUD   | 0.672
    2023-08-29 16:00:00z | AUD   | 0.682
    2023-08-30 07:00:00z | AUD   | 0.689
    2023-08-30 16:00:00z | AUD   | 0.690
    """)
```

In [16]:
```
min_order_size_usd = read("""
    asset | value
    PLN   | 5e5
    CZK   | 5e5
    EUR   | 1e6
    AUD   | 1e6
""")
```

In [17]:
```
trading_session = pd.Timestamp("2023-08-30 16:00:00z")
```

Easy Task: Generate Orders (total and by strategy) as of 2023-08-30 16:00 (Local Ccy)

In [94]:
```
order_by_strategy_and_asset_local_ccy = pd.DataFrame([
    {'strategy': 's1', 'asset': 'PLN', 'order_local_ccy': 0.0},
    {'strategy': 's1', 'asset': 'CZK', 'order_local_ccy': -24000000.0},
    {'strategy': 's2', 'asset': 'EUR', 'order_local_ccy': 0.0},
    {'strategy': 's2', 'asset': 'AUD', 'order_local_ccy': 500000.0},
    {'strategy': 's3', 'asset': 'CZK', 'order_local_ccy': 6000000.0}
])
order_by_strategy_and_asset_local_ccy
```

Out[94]:

	strategy	asset	order_local_ccy
0	s1	PLN	0.0
1	s1	CZK	-24000000.0
2	s2	EUR	0.0
3	s2	AUD	500000.0
4	s3	CZK	6000000.0

In [93]:
```
order_by_asset_local_ccy = pd.DataFrame([
    {'asset': 'AUD', 'order_local_ccy': 500000.0},
```

```
    {'asset': 'CZK', 'order_local_ccy': -18000000.0},
    {'asset': 'EUR', 'order_local_ccy': 0.0},
    {'asset': 'PLN', 'order_local_ccy': 0.0}
])
order_by_asset_local_ccy
```

Out[93]:

	asset	order_local_ccy
0	AUD	500000.0
1	CZK	-18000000.0
2	EUR	0.0
3	PLN	0.0

Easy / Medium Task: Generate Orders (total and by strategy) as of 2023-08-30 16:00 (USD)

Note inverted prices of EUR and AUD

In [62]:

```
order_by_strategy_and_asset_usd = pd.DataFrame([
    {'strategy': 's1', 'asset': 'PLN', 'order_usd': 0.0},
    {'strategy': 's1', 'asset': 'CZK', 'order_usd': -1043024.7718383311},
    {'strategy': 's2', 'asset': 'EUR', 'order_usd': 0.0},
    {'strategy': 's2', 'asset': 'AUD', 'order_usd': 344999.99999999994},
    {'strategy': 's3', 'asset': 'CZK', 'order_usd': 260756.19295958278}
])
order_by_strategy_and_asset_usd
```

Out[62]:

	strategy	asset	order_usd
0	s1	PLN	0.000000e+00
1	s1	CZK	-1.043025e+06
2	s2	EUR	0.000000e+00
3	s2	AUD	3.450000e+05
4	s3	CZK	2.607562e+05

In [66]:

```
order_by_asset_usd = pd.DataFrame([
    {'asset': 'AUD', 'order_usd': 344999.99999999994},
    {'asset': 'CZK', 'order_usd': -782268.5788787483},
    {'asset': 'EUR', 'order_usd': 0.0},
    {'asset': 'PLN', 'order_usd': 0.0}
])
order_by_asset_usd
```

Out[66]:

	asset	order_usd
0	AUD	345000.000000
1	CZK	-782268.578879
2	EUR	0.000000
3	PLN	0.000000

Hard Task: Apply Minimum Order Size and generate new target position after this session

Context: In practice there is often minimum cost we need to pay when trading. This means the orders must be of certain size to make economic sense. Therefore, if order is below the limit size it will not be executed and this needs to be fed back to the target position, so that during next strategy it trades orders knowing that past position is as of T-2, not T-1 (since we skipped T-1 orders)

In [67]:
```
""" Minimum absolute size of the order in USD, below which we do not trad
min_order_size_usd
```

Out[67]:
```
asset
PLN     500000.0
CZK     500000.0
EUR    1000000.0
AUD    1000000.0
Name: value, dtype: float64
```

In [84]:
```
valid_orders_usd = pd.DataFrame([
    {'asset': 'AUD', 'order_usd': 344999.99999999994, 'valid_orders': 0.0
    {'asset': 'CZK','order_usd': -782268.5788787483, 'valid_orders': -782
    {'asset': 'EUR', 'order_usd': 0.0, 'valid_orders': 0.0},
    {'asset': 'PLN', 'order_usd': 0.0, 'valid_orders': 0.0}
])
valid_orders_usd
```

Out[84]:

	asset	order_usd	valid_orders
0	AUD	345000.000000	0.000000
1	CZK	-782268.578879	-782268.578879
2	EUR	0.000000	0.000000
3	PLN	0.000000	0.000000

In [126…
```
"""
Here I find exactly which orders were scaled down and by what scalar,
which I will need when calculating final orders by strategy and asset
"""
valid_orders_usd['fx_rate'] = valid_orders_usd['asset'].apply(
    lambda x: 1/fx_rates.loc[trading_session][x] if x in ['EUR', 'AUD'] e
)
```

```
valid_orders_usd['valid_orders_local'] = valid_orders_usd['valid_orders']
valid_orders_usd['scalar'] = (valid_orders_usd['valid_orders'] / valid_or
valid_orders_scalar = valid_orders_usd.set_index('asset')['scalar']
valid_orders_scalar
```

Out[126…

```
asset
AUD    0.0
CZK    1.0
EUR    1.0
PLN    1.0
Name: scalar, dtype: float64
```

In [129…

```
o_sa_local = order_by_strategy_and_asset_local_ccy # for convenience
o_sa_local['order_local_ccy_validated'] = o_sa_local.apply(
    lambda row: valid_orders_scalar[row['asset']] * row['order_local_ccy'
)
order_as_valid = o_sa_local.copy()
order_as_valid['dt'] = pd.Timestamp("2023-08-30 07:00:00z")
order_as_valid['rt'] = trading_session
order_as_valid
```

Out[129…

	strategy	asset	order_local_ccy	order_local_ccy_validated	dt	
0	s1	PLN	0.0	0.0	2023-08-30 07:00:00+00:00	16:
1	s1	CZK	-24000000.0	-24000000.0	2023-08-30 07:00:00+00:00	16:
2	s2	EUR	0.0	0.0	2023-08-30 07:00:00+00:00	16:
3	s2	AUD	500000.0	0.0	2023-08-30 07:00:00+00:00	16:
4	s3	CZK	6000000.0	6000000.0	2023-08-30 07:00:00+00:00	16:

In [130…

```
""" Adding those orders back to positions to create final target position

original_positions = tpos.unstack(['strategy', 'asset']).ffill()
original_positions_drop_last_row = original_positions[
    original_positions.index.get_level_values('rt') != trading_session]

clean_new_orders = order_as_valid.set_index(['rt', 'dt', 'strategy', 'ass

new_last_row = original_positions_drop_last_row.iloc[-1] + clean_new_orde
```

In [131…

```
""" Final Result """
modified_position_final = pd.concat([original_positions_drop_last_row, ne
modified_position_final
```

Out[131...

	strategy	s1		s2	
	asset	PLN	CZK	EUR	AUD
rt	dt				
2023-08-29 07:00:00+00:00	2023-08-29 07:00:00+00:00	4000000.0	NaN	1800000.0	NaN
2023-08-29 16:00:00+00:00	2023-08-29 07:00:00+00:00	4000000.0	-24000000.0	1800000.0	-1500000.0
2023-08-30 07:00:00+00:00	2023-08-30 07:00:00+00:00	8000000.0	-24000000.0	3600000.0	-1500000.0
2023-08-30 16:00:00+00:00	2023-08-30 07:00:00+00:00	8000000.0	-48000000.0	3600000.0	-1500000.0