VECM Forecast

Imports

Importig Two helper functions from `./helper_functions`:

- `stock_list`: This functions gets an index name (e.g. 'Dow Jones', 'CAC 40', 'DAX', 'Teh50') and returns the list of stocks in that index.
- `stock_prices`: This functions recieves a list of tickers and returns a pandas dataframe containing prices of the corresponding tickers.

[ ]:
```
from helper_functions import stock_prices, stock_list
```

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Constants and Fucntions

We will work with the 521 trailing days of prices series. 126 last days will be the test period.

[ ]:
```
testsmpl=126

interval = 521
```

Cointegration function

This Function Works exactly like the `get_cointegration_params` function in the previous project (i.e. 01_pair_trading).

The only diffrence is that it returns lag_order and rank of the cointegration test. The returned values will be used to build a VECM model.

[ ]:
```
from statsmodels.tsa.vector_ar.vecm import coint_johansen
from statsmodels.tsa.api import VAR
from statsmodels.tsa.vector_ar.vecm import VECM, select_coint_rank, select_order

def get_cointegration_params(df, verbose=False):
    lag_order = select_order(df, maxlags=10, deterministic="ci")
    lag_order = lag_order.aic

    rank_test = select_coint_rank(df, 0, lag_order, method="trace",
                                  signif=0.05)

    is_cointegrated = rank_test.test_stats[0] > rank_test.crit_vals[0]
    if verbose:
        print(rank_test.summary())
    if not is_cointegrated:
        return False, np.NaN, np.NAN

    model = VECM(df, deterministic="ci",
                 k_ar_diff=lag_order,
                 coint_rank=rank_test.rank)
```

```
    vecm_res = model.fit()

    return True, lag_order, rank_test.rank
```

Input Data

We read the list of cointegrated tickers from the output of the previous project (i.e. 01_pair_trading).

[ ]:
```
df = pd.DataFrame()
file = pd.ExcelFile('../01_pair_trading/pairs_2023-01-15.xlsx')
sheet_names = ['Dow Jones', 'CAC 40', 'Dax', 'Teh50']
for sheet in sheet_names:
    df_tmp = pd.read_excel(file, sheet_name=sheet)
    df = df.append(df_tmp)
file.close()
```

Hepler functions

This functino calculates the MAPE of two seres. It receives a dataframe that has two columns: actual and forecasted values.

[ ]:
```
from sklearn.metrics import mean_absolute_percentage_error
def get_mape(df, ticker, pred_tag, test_count=126):
    df = df.dropna(how='any')
    test_true = df.iloc[-test_count:][ticker]
    test_pred = df.iloc[-test_count:][f'{ticker}_{pred_tag}']
    mapel = mean_absolute_percentage_error(
        test_true, test_pred
    )
    return mapel
```

This helper function Will convert the arabic glyphs to standard farsi glyphs. This will be helpful while looking Tehran50 tickers up:

[ ]:
```
def groom(s):
    s = s.replace(' ', ' '(
    s = s.replace(' ', ' '(
    return s
```

Forecast

We explain the code in 8 steps. The starting point of each step is commented in the code by the corresponding number:

1. We Will save all the plots in the `./preds_vecm/{index_name}/` directory. We will remove and recreate the directory each time we run the code.
2. We write a for loop that itterates over four indices: 'Dow Jones', 'CAC 40', 'Dax', 'Teh50'
3. We get the list of tickers that are cointegrated from the previous project.
4. We get the price of all the cointegrated tickers in each index. We split the data into train and test.
5. **One-Step ahead forecast**: For each day in the test period, we consider all leading days as the training set and build a VECM model based on the training data and forecast one step ahead.
6. **Dynamic Multi-Step ahead forecast**: We build a VECM model based on the first 521 days and forecast the 126 days ahead. VECM class uses Dynamic forecasting by default.

7. We plot the the actual values in conjunction with the forecasted values in one graph and save them in `./preds_vecm/{index_name}/` directory.
8. We save the mape value of our forecast in a dictinoary. The will later be used to comapre with Neural Network's forecasts.

[ ]:
```
import itertools
import os
# 1
PATH = r'./preds_vecm/'
if not os.path.exists(PATH):
    os.makedirs(PATH)

errors = []

import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')

# 2
for indice in ['Dow Jones', 'CAC 40', 'Dax', 'Teh50']:
    print(indice, '>>', flush=True)

    # Creating the required directories to save the plots
    PATH = rf'./preds_vecm/{indice}/'
    if not os.path.exists(PATH):
        os.makedirs(PATH)

    # 3
    df1 = df[df['indice']==indice]
    tickers = stock_list.get_stock_list(index=indice)
    isTSE = (indice == 'Teh50')
    if isTSE:
        tickers = [groom(x) for x in tickers]
    data_historical = stock_prices.get_prices(tickers, isTSE)


    for i in range(df1.shape[0]):
        ticker1, ticker2, indice = df1.iloc[i]

        # 4
        data_historical1 = data_historical[[ticker1, ticker2]]
        data_historical1 = data_historical1.dropna(how='all')
        data = data_historical1[-interval:]
        limitPer = len(data) * .85
        data = data.dropna(thresh=limitPer, axis=1)
        data = np.log(data)
        data = data.dropna(how='any')
        data_train = data[:-testsmpl]
        data_test = data[-testsmpl:]
        df_train = data_train.copy()

        # 5
```

```
        # 1Step
        df_train = data_train.copy()
        is_cointegrated, lag_order, rank = get_cointegration_params(df_train)
        if not is_cointegrated:
            continue

        df_predictions = pd.DataFrame()
        for d in range(testsmpl):
            model = VECM(df_train, deterministic="ci",
                    k_ar_diff=lag_order,
                    coint_rank=rank)

            vecm_res = model.fit()
            pred = vecm_res.predict(steps=1)
            data.loc[data_test.iloc[d].name, f'{ticker1}_1step'] = pred[0][0]
            data.loc[data_test.iloc[d].name, f'{ticker2}_1step'] = pred[0][1]
            df_train = df_train.append(data_test.iloc[d])

        # 6
        # Multistep
        df_train = data_train.copy()
        is_cointegrated, lag_order, rank = get_cointegration_params(df_train)
        if not is_cointegrated:
            continue

        model = VECM(df_train, deterministic="ci",
                    k_ar_diff=lag_order,
                    coint_rank=rank)
        vecm_res = model.fit()
        preds = vecm_res.predict(steps=testsmpl)
        for i, pred in enumerate(preds):
            data.loc[data_test.iloc[i].name, f'{ticker1}_multi'] = pred[0]
            data.loc[data_test.iloc[i].name, f'{ticker2}_multi'] = pred[1]

        # 7
        # Plotting
        ax = data.plot(figsize=(15, 8));
        ax.figure.savefig(rf'./preds_vecm/{indice}/{ticker1}_{ticker2}.png');
        plt.close()

        # 8
        # mape
        for ticker, tag in list(itertools.product([ticker1, ticker2], ['1step',␣
↪'multi'])):
            mape=get_mape(data, ticker=ticker, pred_tag=tag, test_count=testsmpl)
            errors.append({
                'tag': f'vecm_{tag}',
                'ticker': ticker,
                'pair': ticker2 if ticker==ticker1 else ticker1,
                'mape': mape*100,
                'indice': indice
```

```
            })

filename = rf'./vecm_mape.xlsx'
writer = pd.ExcelWriter(filename, engine='xlsxwriter')
df_errors = pd.DataFrame(errors)
for index, group_df in df_errors.groupby("indice"):
    group_df.to_excel(writer, sheet_name=str(index),index=False)
writer.save()
```