Kernel Density Estimation of ECM model

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
from statsmodels.tsa.vector_ar.vecm import VECM, select_coint_rank, select_order

from helper_functions import stock_prices, stock_list
```

Input Data

It's always better to have a large sample while performing hypothesis testing. So the sample size has been increased from 521 in previous project (i.e. `01_pair_trading`) to 720.

[ ]:
```
interval = 720
```

We read the list of cointegrated tickers that have been calculated in the 1st project.

[ ]:
```
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 = pd.concat([df, df_tmp])
file.close()
```

Helper Functions

The same cointegration function as was in the first project (i.e. `01_pair_trading`).

We need to test for cointegration for each pair again, because the sample size has increased.

[ ]:
```
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, vecm_res.beta, vecm_res.const_coint
```

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
```

Distribution Tests

For normality, we perform 5 tests.

1. jarque_bera
2. anderson
3. cramervonmises
4. lilliefors
5. Kolmogorov-Smirnov

The last one will be explained later.

[ ]:
```
from scipy import stats
from statsmodels.stats.diagnostic import lilliefors


def is_normal_jb(x) -> bool:
    test = stats.jarque_bera(x)
    return test.pvalue > 0.05

def is_normal_ad(x) -> bool:
    test = stats.anderson(x)
    return test.statistic < test.critical_values[2]

def is_normal_crm(x) -> bool:
    test = stats.cramervonmises(x, 'norm')
    return test.pvalue > 0.05

def is_normal_lil(x) -> bool:
    test = lilliefors(x,  dist='norm')
    return test[1] > 0.05
```

Kolmogorov-Smirnov test

This function receives a series and a distribution name and performs a Kolmogorov-Smirnov test. If returns the test result and the parameters that best fits the series.

[ ]:
```
import scipy
def ks_test(data, dist_name, p_val_tresh=0.01):
    y, x = np.histogram(data, bins=100, density=True)
    x = [(this + x[i + 1]) / 2.0 for i, this in enumerate(x[0:-1])]

    dist = eval("scipy.stats."+ dist_name)
    if (dist_name == "nbinom"):
        p = np.mean(data)/(np.std(data)**2)
        n = np.mean(data)*p/(1.0 - p)
        if n<0 or p<0 or p>1:
            return True, np.nan, np.nan, np.nan
        param = (n, p)
    else:
        param = dist.fit(data)

    dist_fitted = dist(*param)

    ks_stat, ks_pval = stats.kstest(data, dist_fitted.cdf)
    return (ks_pval < p_val_tresh), dist, param, x
```

We implenet this test for normal and other distributions at the same time.

The Logic is as follows:

1. For a series, we find the best parameters for each distribution that fits the data best.
2. We perform the `kstest` on the data and the distribution.
3. If the null hypothesis rejected, Then we can conclude that the distribution doesn't fit the data well.
4. If the null hypothesis is not rejected, We can loosely conclude that the distribuiton fits the data well enough and we save it for the plotting process later.

[ ]:
```
def test_for_dist(data, ticker1, ticker2, indice_path):

    fitted_normal_methods = []
    fitted_dists = []

    normal_methods = ["jb", "ad", "crm", "lil"]
    for method in normal_methods:
        fn = eval(f"is_normal_{method}")
        if fn(data):
            fitted_normal_methods.append(method)


    options = ["norm", "lognorm", "chi2", "t", "beta", "gamma", "weibull_min",␣
 ↪"nbinom"]

    hs = plt.hist(data, bins=80, density=True, label="data");
    for dist_name in options:
        is_h0_rejected, dist,  param, x =  ks_test(data, dist_name)
```

```
        if is_h0_rejected:
            continue
        else:
            fitted_dists.append(dist_name)
            if dist_name == "nbinom":
                h = plt.plot(x, dist.pmf(x, *param), label=dist_name);
            else:
                h = plt.plot(x, dist.pdf(x, *param), label=dist_name);

    plt.title(f"{ticker1} & {ticker2}")
    plt.legend();
    plt.savefig(rf'{indice_path}/{ticker1} & {ticker2}.png')
    plt.close()

    return fitted_normal_methods, fitted_dists
```

For each cointegrated pair, We build the ECM model and test for normality and other distributions. Finally, we save the results.

[ ]:
```
import shutil
import os

PATH = r'./plots/'
if os.path.exists(PATH):
    shutil.rmtree(PATH)
os.makedirs(PATH)

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

pairs = []
for indice in ['Dow Jones', 'CAC 40', 'Dax', 'Teh50']:
    indice_path = PATH + indice
    os.makedirs(indice_path)

    print(indice, '>>', flush=True)
    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]
        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')
        cols = data.columns

        for i in range(len(cols)-1):
            for j in range(i+1, len(cols)):
                try:
                    is_cointegrated, BJ2n, C0J2n = get_cointegration_params(data.
↪dropna(how='any'))
                except:
                    continue
                if not is_cointegrated:
                    continue

                ecm = np.matmul(data, BJ2n) + C0J2n
                x = ecm[0].values
                fitted_normal_methods, fitted_dists = test_for_dist(x, ticker1,␣
↪ticker2, indice_path)
                pairs.append({
                    'sym1': cols[i],
                    'sym2': cols[j],
                    'indice': indice,
                    'fitted_normal_methods': fitted_normal_methods,
                    'fitted_dists': fitted_dists
                })


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