Value-at-Risk forecasting from a GARCH-type model

MNT/USD with Inflation rate as the exogenous variable

- Date: March , 2023
- Purpose:Value-at-Risk forecasting of MNT/USD
- Data from from Jan 2016 to Dec 22 (GARCH-Model)
- Author: Isaac Osei-Mensah

In [ ]:
```
# Import packages
import pandas as pd
import numpy as np
import re
import quandl
import matplotlib.pyplot as plt
import seaborn as sns
import arch
from arch import *
from arch.__future__ import reindexing
from scipy.stats import norm
import warnings
```

A. Import Open-source data Jan 16 - Dec 22

- Souce :investing.com
- Dataset Used: I collected the daily exchange rate from January 2016 to December 2022

- Below is an overview of the head collected

In [ ]:
```
# Impoirt data and set date column as index
df =  pd.read_csv("C:/Users/Public/USD_MNT Historical Data_jan16-dec22.csv",
                  index_col=0, parse_dates=True)
# Print the head of the dataset
print(df.head())
```

```
            exchange    open    high     low  inflation
date
2016-01-01    1993.0  1993.0  1993.0  1993.0      0.554
2016-01-04    1991.5  1997.5  1999.5  1989.5      0.554
2016-01-05    1993.5  1995.5  1997.5  1987.5      0.554
2016-01-06    1994.0  1995.0  1998.0  1994.0      0.554
2016-01-07    1994.5  1994.5  1998.5  1986.5      0.554
```

Question 1:

- **Collect one or more daily series of macroeconomic/financial indicator(s) which will influence the log return of MNT/USD from publicly available source. Please describe your choice and explain why**

Response:

- ## Inflation Rate

I collected inflation as a variable that would highly influence the log return of MNT/USD

In flation is a measure of the rate at which prices for goods and services are increasing within an economy. Inflation rate can have a significant impact on the log return of MNT/USD since it affects the purchasing power of the currency, which in turn affects the demand for MNT.

If the inflation rate in Mongolia is higher than the inflation in other countries, the value of the currency will decrease relative to those currencies. This will lead to a weaker MNT/USD exchange rate. On the other hand, if the inflation rate in Mongolia is lower than the inflation in other countries, the value of the currency will increase, leading to a stronger MNT/USD exchange rate.

Investors and traders often monitor inflation rates closely as they can affect the interest rates set by central banks. Central banks may raise interest rates to combat high inflation, which can make the currency more attractive to investors seeking higher yields. Conversely, low inflation rates may lead central banks to lower interest rates to stimulate economic growth, which can lead to a weaker currency.

Question 2:

- **Run a GARCH-type model on the log return of MNT/USD with the series in (1) as the exogenous variable(s) in the mean process**

CLEAN DATA

- **Transforms data from USD/MNT to MNT/USD**
- **Data was obtained in USD/MNT format and**
- **Must be inverted to MNT/USD**

In [ ]:
```
for j in ['exchange', 'open', 'high', 'low']:
    df[j] = (1/ df[j])
print(df.head())
```

```
            exchange      open      high       low  inflation
date
2016-01-01  0.000502  0.000502  0.000502  0.000502      0.554
2016-01-04  0.000502  0.000501  0.000500  0.000503      0.554
2016-01-05  0.000502  0.000501  0.000501  0.000503      0.554
2016-01-06  0.000502  0.000501  0.000501  0.000502      0.554
2016-01-07  0.000501  0.000501  0.000500  0.000503      0.554
```

Keep only the exchange rate and inflation as variables of interest

In [ ]:
```
df =  df[["exchange", "inflation"]]
print(df.head())
```

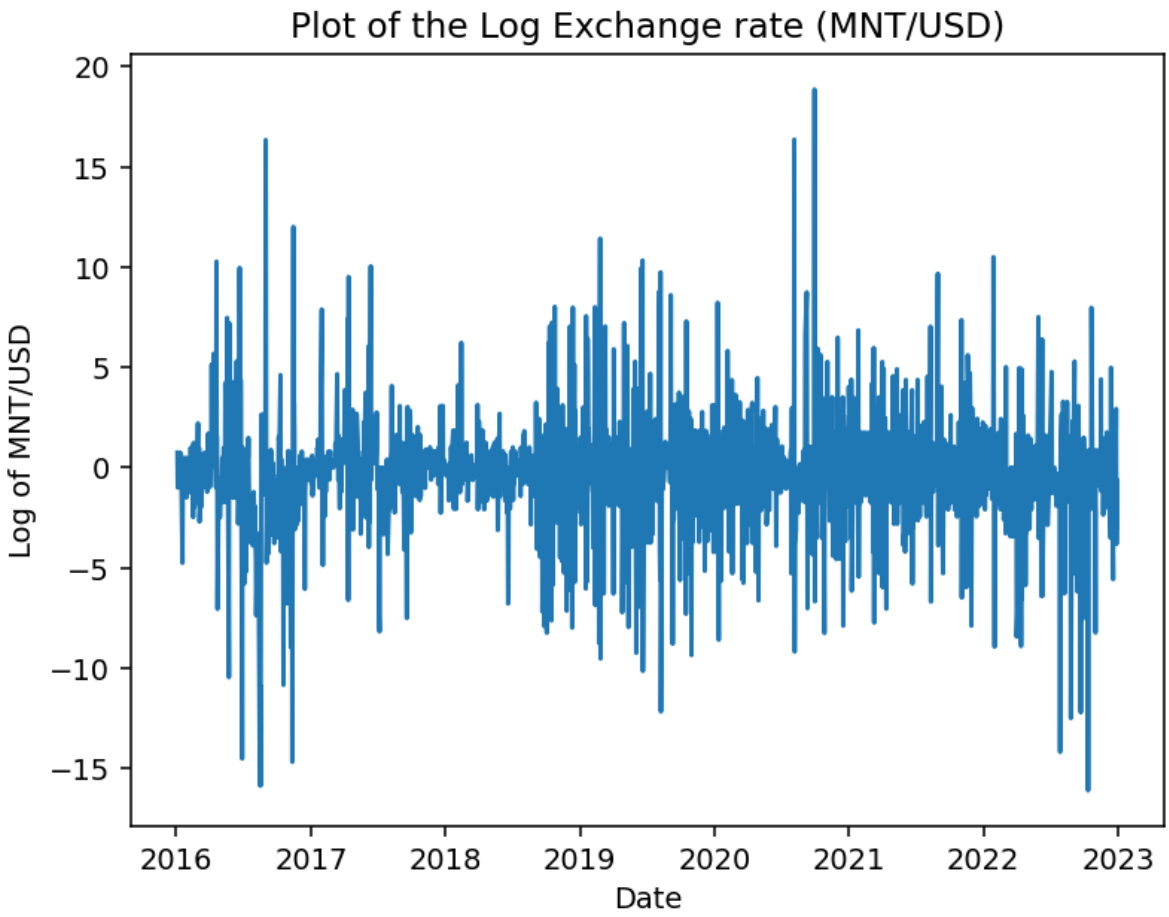
```
            exchange  inflation
date
2016-01-01  0.000502      0.554
2016-01-04  0.000502      0.554
2016-01-05  0.000502      0.554
2016-01-06  0.000502      0.554
2016-01-07  0.000501      0.554
```

Calculate the log returns of the exchange rate

In [ ]:
```
df['log_exchange'] = np.log(df['exchange']).diff()          # Use Numpy to calculat
keep = ~df.isin([np.nan, np.inf, -np.inf]).any(axis=1)      # Remove any nan or inf
df = df[keep].astype(np.float64)
df =  df*1000                                               # Multiply data by 1000
inflation = df['inflation']
print(df.head())
```

```
            exchange  inflation  log_exchange
date
2016-01-04  0.502134      554.0      0.752918
2016-01-05  0.501630      554.0     -1.003764
2016-01-06  0.501505      554.0     -0.250784
2016-01-07  0.501379      554.0     -0.250721
2016-01-08  0.501379      554.0      0.000000
```

In [ ]:
```
# Plot the log exchange series
plt.plot(df.log_exchange)
plt.title('Plot of the Log Exchange rate (MNT/USD)')
plt.xlabel('Date')
plt.ylabel('Log of MNT/USD')
plt.rcParams['figure.dpi'] = 140
```



Introduce treatment Startdate

- I chose January 1 2020 as the treatment start
- Data before 2020 would be treated as pre-treatment
- Data from 2020 would be treated as post-treatment

In [ ]:
```
treatment_date = '2020-01-01'
pre_treat_df = df[df.index < treatment_date]
post_treat_df = df[df.index >= treatment_date]
```

FIT THE GARCH MODEL

- GARCH model with Inflation as the exogenous variable in the mean process

In [ ]:
```
# Define the model with inflation as the exogenous variable in mean process
garch_model = arch_model(df['log_exchange'],
                  vol='GARCH', p=1, o=0, q=1,
                  mean='ARX', lags=1, x=inflation)
```

In [ ]:
```
garch_fit = garch_model.fit()
```

```
Iteration:      1,   Func. Count:      8,   Neg. LLF: 7975600228583957.0
Iteration:      2,   Func. Count:     25,   Neg. LLF: 20743.576365425695
Iteration:      3,   Func. Count:     34,   Neg. LLF: 2818325.475245868
Iteration:      4,   Func. Count:     43,   Neg. LLF: 39374.85640711709
Iteration:      5,   Func. Count:     52,   Neg. LLF: 2789322.289110828
Iteration:      6,   Func. Count:     65,   Neg. LLF: 27890.057393729257
Iteration:      7,   Func. Count:     75,   Neg. LLF: 7191.96405308031
Iteration:      8,   Func. Count:     82,   Neg. LLF: 6860.699217380448
Iteration:      9,   Func. Count:     89,   Neg. LLF: 6150.268202284114
Iteration:     10,   Func. Count:     96,   Neg. LLF: 12240.55757809024
Iteration:     11,   Func. Count:    106,   Neg. LLF: 6582.622620413569
Iteration:     12,   Func. Count:    114,   Neg. LLF: 4524.574289606903
Iteration:     13,   Func. Count:    121,   Neg. LLF: 5151.040690086271
Iteration:     14,   Func. Count:    129,   Neg. LLF: 4492.535127995128
Iteration:     15,   Func. Count:    137,   Neg. LLF: 4444.206712538129
Iteration:     16,   Func. Count:    144,   Neg. LLF: 4399.837474303617
Iteration:     17,   Func. Count:    151,   Neg. LLF: 4970.379230920644
Iteration:     18,   Func. Count:    159,   Neg. LLF: 5095.062627644459
Iteration:     19,   Func. Count:    168,   Neg. LLF: 7318.871246265678
Iteration:     20,   Func. Count:    176,   Neg. LLF: 4493.183420260303
Iteration:     21,   Func. Count:    184,   Neg. LLF: 4357.817875562576
Iteration:     22,   Func. Count:    192,   Neg. LLF: 4341.095839566184
Iteration:     23,   Func. Count:    199,   Neg. LLF: 4340.113896243827
Iteration:     24,   Func. Count:    206,   Neg. LLF: 4339.14023504838
Iteration:     25,   Func. Count:    213,   Neg. LLF: 4338.816714082992
Iteration:     26,   Func. Count:    220,   Neg. LLF: 4338.484311305332
Iteration:     27,   Func. Count:    227,   Neg. LLF: 4337.965181898584
Iteration:     28,   Func. Count:    234,   Neg. LLF: 4333.305586803837
Iteration:     29,   Func. Count:    241,   Neg. LLF: 4517.650354168045
Iteration:     30,   Func. Count:    249,   Neg. LLF: 5194.276112746981
Iteration:     31,   Func. Count:    257,   Neg. LLF: 7343.076816746314
Iteration:     32,   Func. Count:    265,   Neg. LLF: 4415.023997840084
Iteration:     33,   Func. Count:    273,   Neg. LLF: 4496.5617293715995
Iteration:     34,   Func. Count:    281,   Neg. LLF: 4345.187522256843
Iteration:     35,   Func. Count:    289,   Neg. LLF: 4318.247645111089
Iteration:     36,   Func. Count:    296,   Neg. LLF: 4316.419418637646
Iteration:     37,   Func. Count:    303,   Neg. LLF: 4314.565671702103
Iteration:     38,   Func. Count:    310,   Neg. LLF: 4316.496844425337
Iteration:     39,   Func. Count:    318,   Neg. LLF: 4319.881267877157
Iteration:     40,   Func. Count:    326,   Neg. LLF: 4312.097575927701
Iteration:     41,   Func. Count:    333,   Neg. LLF: 4312.002396146303
Iteration:     42,   Func. Count:    340,   Neg. LLF: 4311.990392204319
Iteration:     43,   Func. Count:    347,   Neg. LLF: 4311.988733662793
Iteration:     44,   Func. Count:    354,   Neg. LLF: 4311.988543715968
Iteration:     45,   Func. Count:    361,   Neg. LLF: 4311.98851048148
Iteration:     46,   Func. Count:    368,   Neg. LLF: 4311.988508399111
Iteration:     47,   Func. Count:    374,   Neg. LLF: 4311.988508398996
Optimization terminated successfully    (Exit mode 0)
            Current function value: 4311.988508399111
            Iterations: 47
            Function evaluations: 374
            Gradient evaluations: 47
```

In [ ]:
```
# Print the summary of the model
print(garch_fit.summary())
```

```
                           AR-X - GARCH Model Results
==============================================================================
Dep. Variable:           log_exchange   R-squared:                       0.021
Mean Model:                      AR-X   Adj. R-squared:                  0.020
Vol Model:                     GARCH   Log-Likelihood:               -4311.99
Distribution:                 Normal   AIC:                           8635.98
Method:            Maximum Likelihood   BIC:                           8668.92
                                        No. Observations:                 1790
Date:                Mon, Mar 13 2023   Df Residuals:                     1787
Time:                        11:32:23   Df Model:                            3
                                Mean Model
==============================================================================
                     coef    std err          t      P>|t|       95.0% Conf. Int.
------------------------------------------------------------------------------
Const             -0.0971      0.190     -0.510      0.610      [ -0.470,  0.276]
log_...nge[1]     -0.2450  3.483e-02     -7.035  1.999e-12      [ -0.313, -0.177]
inflation     -6.2584e-06  3.400e-05     -0.184      0.854 [-7.290e-05,6.038e-05]
                            Volatility Model
=============================================================================
                 coef    std err          t      P>|t|     95.0% Conf. Int.
-----------------------------------------------------------------------------
omega          0.3320      0.371      0.895      0.371    [ -0.395,  1.059]
alpha[1]       0.1205  6.275e-02      1.920  5.482e-02 [-2.487e-03,  0.243]
beta[1]        0.8523  9.660e-02      8.823  1.111e-18    [  0.663,  1.042]
=============================================================================

Covariance estimator: robust
```

Forecast Value-at-Risk at 5% confidence level

In [ ]:
```
forecast_horizon = 1
forecast = garch_fit.forecast(horizon=forecast_horizon,
                              x=inflation.iloc[-1:], method='simulation')
volatility = np.sqrt(forecast.variance)
VaR = -volatility * np.percentile(df["log_exchange"], 10)
var_10 = np.percentile(forecast.variance[-1:], 10)
print(forecast)
VaR = -volatility * np.percentile(df["log_exchange"], 10)
```

```
<arch.univariate.base.ARCHModelForecast object at 0x0000025E73DB3040>
```

In [ ]:
```
print(garch_fit.summary())
```

```
                     AR-X - GARCH Model Results
==============================================================================
Dep. Variable:           log_exchange   R-squared:                       0.021
Mean Model:                      AR-X   Adj. R-squared:                  0.020
Vol Model:                     GARCH   Log-Likelihood:               -4311.99
Distribution:                 Normal   AIC:                           8635.98
Method:            Maximum Likelihood   BIC:                           8668.92
                                        No. Observations:                 1790
Date:                Mon, Mar 13 2023   Df Residuals:                     1787
Time:                        11:32:23   Df Model:                            3
                                Mean Model
==============================================================================
                     coef    std err          t      P>|t|       95.0% Conf. Int.
------------------------------------------------------------------------------
Const             -0.0971      0.190     -0.510      0.610      [ -0.470,  0.276]
log_...nge[1]     -0.2450  3.483e-02     -7.035  1.999e-12      [ -0.313, -0.177]
inflation     -6.2584e-06  3.400e-05     -0.184      0.854 [-7.290e-05,6.038e-05]
                           Volatility Model
============================================================================
                 coef    std err          t      P>|t|     95.0% Conf. Int.
----------------------------------------------------------------------------
omega          0.3320      0.371      0.895      0.371    [ -0.395,  1.059]
alpha[1]       0.1205  6.275e-02      1.920  5.482e-02 [-2.487e-03,  0.243]
beta[1]        0.8523  9.660e-02      8.823  1.111e-18    [  0.663,  1.042]
============================================================================

Covariance estimator: robust
```

Question 3:

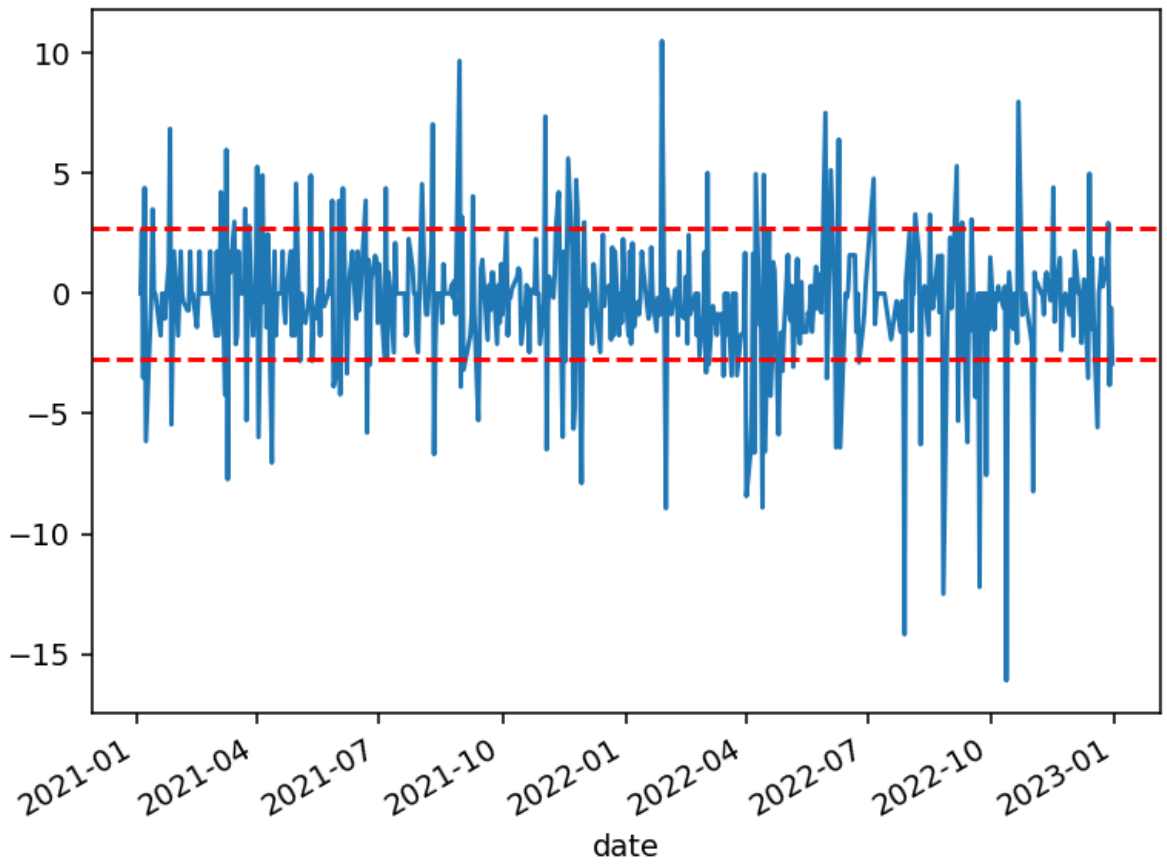
- **Based on (2), please prepare a plot of conditional distribution curve for a specific date, on which you should indicate the value at risk of 10% in total, both tails. Prepare a summary to interpret this plot.**

In [ ]:
```
date_selected = '2021-01-04'
# Conditional Districation Curve----
cond_var = garch_fit.conditional_volatility
# Calculate the 10% VaR
VaR = -1.645 * np.sqrt(cond_var[date_selected])
```

In [ ]:
```
cond_var = garch_fit.conditional_volatility
# Calculate the 10% VaR
VaR = -1.645 * np.sqrt(cond_var[date_selected])

fig, ax = plt.subplots()
df['log_exchange'][date_selected:].plot(ax=ax)
ax.axhline(y=-VaR, color='r', linestyle='--')
ax.axhline(y=VaR, color='r', linestyle='--')
#ax.set_title('Conditional Distribution on 2021-01-04')
#ax.set_xlabel('Date')
```

```
#ax.set_ylabel('Log Returns')
plt.show()
```



In [ ]: