

"Operation Research Project Report"

Prepared for

Dr. Koorush Ziarati

TA

Milad Barooni

Prepared by

Ali Maher, Mehdi Vakili

28th of Jan 2024

Introduction

The project involves implementing a sequential decision-making process for investment in **Gold**, **Stock**, and **Bond**, with decisions made weekly. Historical data is utilized, and a linear regression model is developed to predict future occurrences. The decision-making process is then integrated with linear programming using MiniZinc, ensuring an interpretable objective function and constraints.

Let's Get Through

- **Part 1**

We divided the project into three parts. One is the prediction part, which we should implement a regression for fitting a line to our points (here our points is the object prices and the dates). As specified in the project documentation, we should use only **linear regression** and no other types like quadratic.

In implementation of this part with MiniZinc, we ran into a few problems. First of all, in MiniZinc we cannot multiply one float variable to another float variable, because of the solvers' limitations. We also install a new solver for MiniZinc, named "JaCoB". But we still faced the previous problem. And as it is obvious, we cannot even use the power function for these kinds of variables. So, we cannot implement a good linear regression, because all the good linear regressions use power of two for measuring errors. Even it was not possible to use "absolute" function. At the end we use a trick for handling this problem:

```
constraint forall(i in 1..numberOfDays) (err[i] >= a * days[i] + b - price[i]);
```

```
constraint forall(i in 1..numberOfDays) (err[i] >= price[i] - a * days[i] - b);
```

this part of code works as the absolute function.

- **Part 2**

The second part, is implementing a MiniZinc file for decision-making. This file is based on optimization, by that I mean to **maximize** our profit from investment of gold, stock and the bond. I should mention that this investment is based on the predicted data which we have obtained from the last part. In this file also, we have some constraints for make sure that the calculated budget is true (for example we cannot sell negative number of gold).

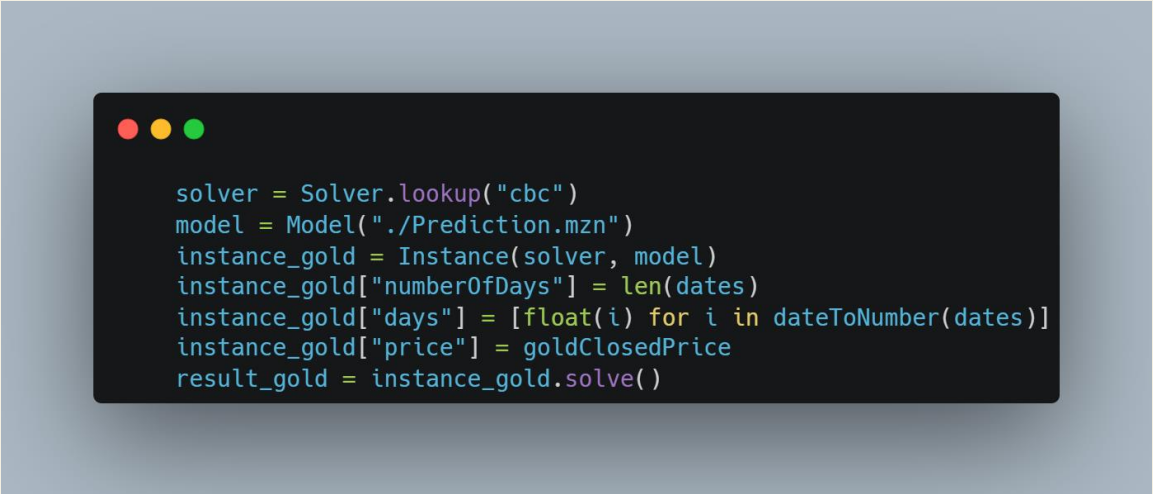
- **Part 3**

And the third part is about the data handling with *numpy* and *pandas*, time handling for different time period with *datetime* library, connecting the prediction part and the decision-making part and at the end showing the last results including each week's plots and decisions with a eye-catching view by using *streamlit* lib.

For data handling, at first, we want to use the "Anacondas" and "Jupyter Notebook" but we faced several problems such as the *async_loop*. And we decide to continue without that because of lack of time.

For the prediction, we use different window from the last data. And at the end we conclude for short term investment the best one is 4 days before the decision-making.

Let's show a sample of solving a MiniZinc file with python:



```
solver = Solver.lookup("cbc")
model = Model("./Prediction.mzn")
instance_gold = Instance(solver, model)
instance_gold["numberOfDays"] = len(dates)
instance_gold["days"] = [float(i) for i in dateToNumber(dates)]
instance_gold["price"] = goldClosedPrice
result_gold = instance_gold.solve()
```

Choosing the "coin-bc" as the solver, then creating the model from the "Prediction.mzn" MiniZinc file, creating new instance, then giving amount to our MiniZinc's variables.

And in order to be more functional we implement some functions:

```
def dateToNumber(dates):
    datesInNumber = []
    for d in dates:
        tempTime = datetime.datetime.strptime(d, "%Y-%m-%d")
        datesInNumber.append((tempTime - st.session_state.startTime).days)

    return datesInNumber
```

For handling more easily we convert the time periods to days.

```
def getLastMonthData(decisionDate, dataframe):
    dates = (dataframe[(dataframe["Date"] <= decisionDate.strftime("%Y-%m-%d")) & (dataframe["Date"]
    >= (decisionDate - datetime.timedelta(days=4)).strftime("%Y-%m-%d"))][["Date"].to_list()]
    prices = (dataframe[(dataframe["Date"] <= decisionDate.strftime("%Y-%m-%d")) & (dataframe["Date"]
    >= (decisionDate - datetime.timedelta(days=4)).strftime("%Y-%m-%d"))][["Close"].to_list()]

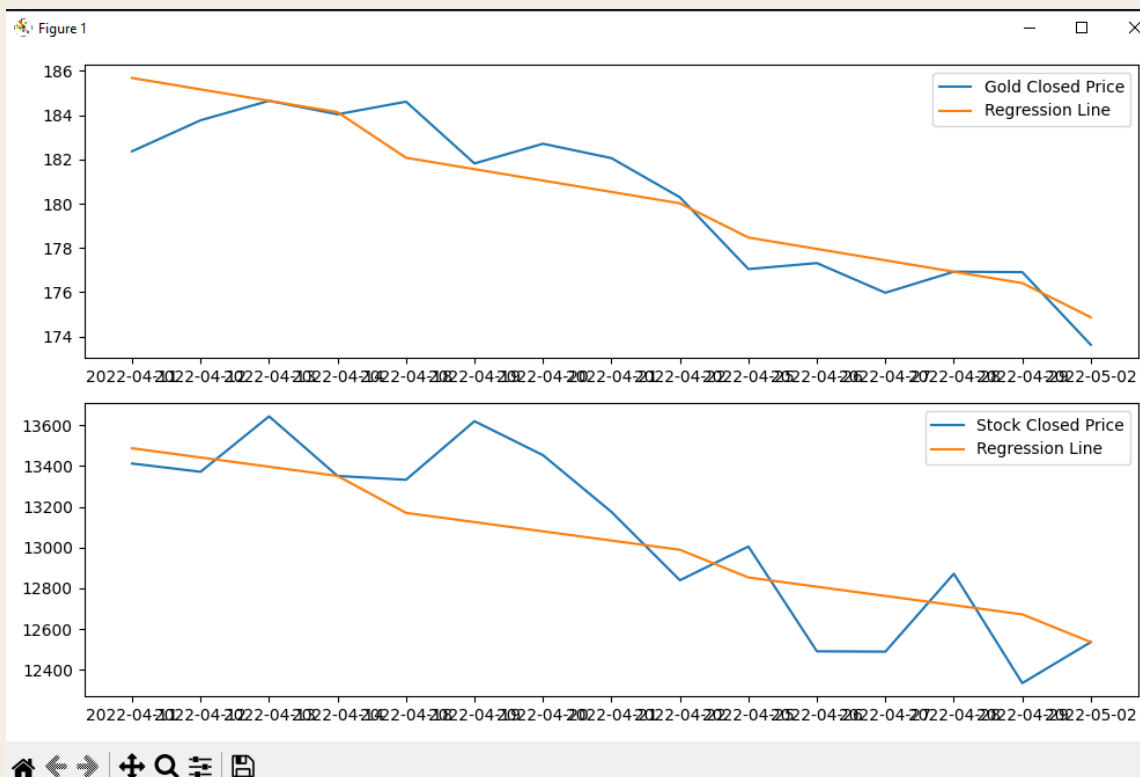
    return dates, prices
```

And this function is for getting the last data.

Visualizing: for showing data we use different approaches. We log in the terminal console, we plot the data with the *matplotlib*, and also, we show the complete results including the plots, data, decision-makings and our budget with a GUI with help of *streamlit* library.

```
Solution(objective=504.1060470010862, goldDiff=0, stockDiff=4.094132420680754, bondDiff=0.0, A1=0.0, _checker='')
date:2023-05-01, total : 50000.0, budget: -7.275957614183426e-12, goldAmount: 0, stockAmount: 4.094132420680754, bondAmount: 0

Solution(objective=0.0, goldDiff=0, stockDiff=0.0, bondDiff=0.0, A1=0.0, _checker='')
date:2023-05-08, total : 50181.45323034801, budget: -7.275957614183426e-12, goldAmount: 0, stockAmount: 4.094132420680754, bondAmount: 0
```



2023-05-08

Total

-7.275957614183426e-12

↑ 50181.453230348015

Gold Amount:

0

↑ 0

Stock Amount:

4.094132420680754

↑ 0.0

Bond Amount:

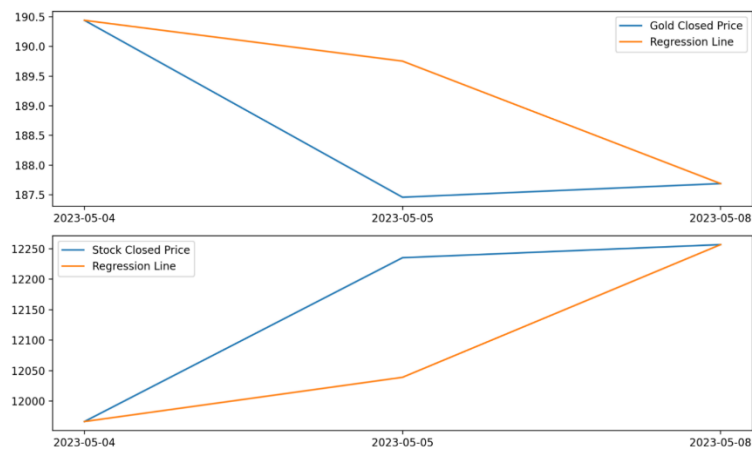
0

↑ 0.0

Total Budget:

50181.45323034801

↑ 0.0



Next Week

For running this app, you should write "*streamlit run main.py*". and then it will push you to local server made on the browser.

The project files with doc are also available on my [GitHub](#).

Thanks for your time