**Домашна 4 – рефакторирање (објаснување)**

Во рамки на делот со рефакторирање на кодот од Домашна 4, го рефакторирав сегментот со осцилаторите со употреба на template design pattern. (Направените промени во споредба со минатата верзија се однесуваат на патеката Domashna3/flask/oscilators)

Првично, за 6-те типа осцилатори имавме дефинирано 6 посебни функции (ao.py, cci.py, cmo.py, DMI.py, rsi.py и stochastic\_o.py) кои го читаат и претпроцесираат dataframe-от за соодветниот тикер и исцртуваат график врз база на внесените параметри. Сите функции главно се состојеа од boilerplate код, со мали модификации во делот на користење на соодветниот индикатор за пресметката и во делот на исцртувањето на графикот.

Затоа се одлучив да создадам една апстрактна базна класа (Oscillator\_Generator) од којашто ќе наследуваат сите различни типови на осцилатори. Објектите на наследените класи при инстанцирање го повикуваат конструкторот на базната класа, со влезни параметри <tiker> и <interval>, и само го повикуваат методот generate\_graph(). Овој метод по редослед ги повикува предефинираните методи: read\_dataframe(), prepare\_data() и plot(). Методот plot(), меѓу другото, во базната класа ги повикува методите get\_results() и customize\_plot(), коишто се апстрактни.

get\_results() се однесува на делот на користење на соодветниот индикатор (indicators) од библиотеката stock\_indicators.

Во customize\_plot() се прават сите специфични прилагодувања на графикот за секој од осцилаторите (поставување на вредностите на x и y-оските, лабелите, насловите, линиите и сл.)

Само во овие две функции се наоѓаше целата разлика помеѓу класите за осцилаторите, затоа се решив да ги декларирам како апстрактни методи и да ги преоптоварам во изведените класи на Oscillator\_Generator.

Во продолжение како пример е даден кодот за Awesome Oscillator, пред и после рефакторирање:

def calcAO(tiker, interval, start\_date, end\_date, short\_window):  
 data=pd.read\_csv(f'temp\_stocks/temp\_data/{tiker}.csv')  
 df = pd.DataFrame(data)  
 df = df.tail(int(interval)+34)  
 df.dropna(inplace=True)  
  
 df['max'] = [  
 ''.join([c if c not in '.,' else ',' if c == '.' else '.' for c in str(num)])  
 if pd.notnull(num) else '' # Handle NaN or missing values  
 for num in df['max']  
 ]  
 # print(df['min'])  
 df['min'] = [  
 ''.join([c if c not in '.,' else ',' if c == '.' else '.' for c in str(num)])  
 if pd.notnull(num) else '' # Handle NaN or missing values  
 for num in df['min']  
 ]  
 # print(df['min'])  
 df['min'] = df['min'].str.replace(',', '').astype(float)  
 df['max'] = df['max'].str.replace(',', '').astype(float)  
 # Step 1: Calculate the Median Price  
 df['max']=pd.to\_numeric(df['max'],errors='coerce')  
 df['min']=pd.to\_numeric(df['min'],errors='coerce')  
 df['Median\_Price'] = (df['max'] + df['min']) / 2  
  
  
 # Step 2: Calculate 5-period and 34-period SMAs  
 df['SMA\_5'] = df['Median\_Price'].rolling(window=5).mean()  
 df['SMA\_34'] = df['Median\_Price'].rolling(window=34).mean()  
  
 # Step 3: Calculate the Awesome Oscillator (AO)  
 df['AO'] = df['SMA\_5'] - df['SMA\_34']  
 print(df['AO'])  
 df = df.dropna(subset=['AO'])  
 # Step 4: Plot the Awesome Oscillator  
 plt.figure(figsize=(12, 6))  
 plt.bar(df['Date'], df['AO'], color=['red' if ao < 0 else 'green' for ao in df['AO']], width=1.0)  
 plt.title('Awesome Oscillator (AO)')  
 plt.xlabel('Date')  
 plt.ylabel('AO Value')  
 plt.grid()  
 # plt.tight\_layout()  
 img\_io = io.BytesIO()  
 plt.savefig(img\_io, format='png')  
 img\_io.seek(0)  
 plt.close()  
  
 return img\_io

Вака изгледа кодот за Awesome Oscillator по направеното рефакторирање:

class AO\_Generator(Oscillator\_Generator):  
 def \_\_init\_\_(self, tiker, interval):  
 super().\_\_init\_\_(tiker, interval)  
  
 def get\_results(self):  
 median\_price = (self.df['max'] + self.df['min']) / 2  
 # Step 2: Calculate 5-period and 34-period SMAs  
 sma\_5 = median\_price.rolling(window=5).mean()  
 sma\_34= median\_price.rolling(window=34).mean()  
 # Step 3: Calculate the Awesome Oscillator (AO)  
 ao = sma\_5 - sma\_34  
 return ao  
  
 def customize\_plot(self):  
 ao = self.get\_results()  
 plt.bar(self.df['Date'], ao, color=['red' if ao\_ < 0 else 'green' for ao\_ in ao], width=1.0)  
 plt.title('Awesome Oscillator (AO)')  
 plt.xlabel('Date')  
 plt.ylabel('AO Value')

А ова е кодот на апстрактната класа:

import pandas as pd  
import matplotlib.pyplot as plt  
from stock\_indicators.indicators.common import Quote  
import io  
from abc import ABC, abstractmethod  
  
class Oscillator\_Generator(ABC):  
 def \_\_init\_\_(self, tiker, interval):  
 self.df = None  
 self.tiker = tiker  
 self.interval = interval  
  
 def generate\_graph(self):  
 self.read\_dataframe()  
 self.prepare\_data()  
 return self.plot()  
  
 def read\_dataframe(self):  
 # Step 1: Read CSV and clean temp\_data  
 df = pd.read\_csv(f"temp\_stocks/temp\_data/{self.tiker}.csv")  
 df.dropna(inplace=True)  
 self.df = df  
  
 def prepare\_data(self):  
 # Step 2: Fix 'max' and 'min' columns (convert to numeric)  
 self.df = self.df.tail(int(self.interval) + 20)  
 self.df['max'] = self.df['max'].astype(str).str.replace('.', '').str.replace(',', '.')  
 self.df['min'] = self.df['min'].astype(str).str.replace('.', '').str.replace(',', '.')  
 self.df['max'] = pd.to\_numeric(self.df['max'], errors='coerce')  
 self.df['min'] = pd.to\_numeric(self.df['min'], errors='coerce')  
 # Step 3: Clean up 'avg\_price' column  
 self.df['avg\_price'] = self.df['avg\_price'].astype(str).str.replace('.', '').str.replace(',', '.')  
 self.df['avg\_price'] = pd.to\_numeric(self.df['avg\_price'], errors='coerce')  
 # Step 4: Parse Date column properly  
 self.df['Date'] = pd.to\_datetime(self.df['Date'], format='%d.%m.%Y', errors='coerce')  
  
 def get\_quotes(self):  
 # Step 5: Convert to Quote objects  
 quotes = [  
 Quote(row.Date, None, row['max'], row['min'], row['avg\_price'], None)  
 for \_, row in self.df.iterrows()  
 if pd.notnull(row['max']) and pd.notnull(row['min']) and pd.notnull(row['avg\_price'])  
 ]  
 return quotes  
  
 def plot(self):  
 # Step 8: Plot CMO  
 plt.figure(figsize=(12, 6))  
 #implemented in the abstract class' extensions  
 self.customize\_plot()  
 #the rest is boilerplate  
 plt.legend()  
 plt.grid()  
 plt.tight\_layout()  
 #Generate and return img\_io object  
 img\_io = io.BytesIO()  
 plt.savefig(img\_io, format='png')  
 img\_io.seek(0)  
 plt.close()  
 return img\_io  
  
 @abstractmethod  
 def customize\_plot(self):  
 pass  
  
 @abstractmethod  
 def get\_results(self):  
 pass