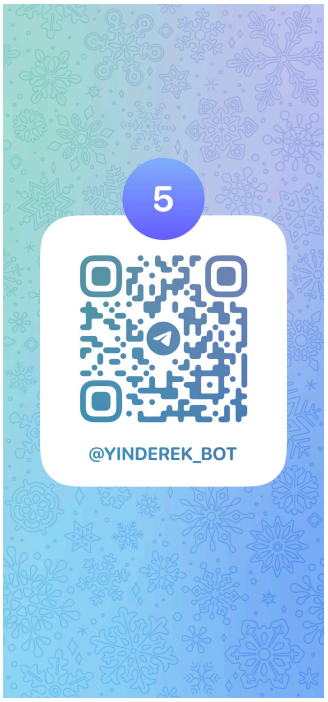
Providing Stock Information and Further Analyses via Telegram Chat-bot

Telegram Chat-bot is the simplest way to achieve human-computer interaction. By using some simple instructions like `\start` or other commands settled by us, programmers, Telegram Chat-bot can respond in certain ways.

GITHUB: https://github.com/Derek-Y-JQ/Telegram-chatbox-Chinese-stock

1. How to connect with Telegram Chat-bot

Telegram Chat-bot function is actually easily accessed. First search `@BotFather` in Telegram and then follow the instructions to create an individual bot. Here we've already created one whose user name is `@Yinderek_bot` which is now used for test.
This application allows user easily get data and analysis of some news immediately rather than search on the website or extract information by himself.



After creating a bot, we need to use `Python` to connect to it.

PYTHON

```
import logging
from telegram import Update, ReplyKeyboardMarkup
from telegram.ext import Application, CommandHandler, MessageHandler,
filters, CallbackContext, ConversationHandler
```

2. Financial Data Resource

We use `Tushare` to gain financial data resources. `Tushare` is a utility for crawling historical data of China's stocks, and it provides a relatively stable data interface for `Python`. It is simple and easy to use, and it is suitable for people who want to obtain data for quantitative analysis. `Tushare` provides a variety of data,

including but not limited to stock basic data, historical trading data, financial data, and so on. It is widely used in the field of quantitative investment in China.

PYTHON

```python
import tushare as ts
ts.set_token('my_token')
pro = ts.pro_api()
```

3. Chat-bot Basic Function Structure

3.0 Three States

PYTHON

```python
#Define States
ENTER_STOCK_CODE, GET_STOCK_DATA, FURTHER_MORE = range(3)
```

Enter_Stock_Code is a state which achieve a function to allow user to type in the stock code to get the stock information. Get_Stock_Data is a state which achieve a function to get the current stock data. Further_More is a state which achieve a function to give user more information about the stock, including trend, predict, gaining news and using LLM to analyse it and give some suggestions. All states will be clearly unfolded in the `main` function which we will discuss later.

3.1 Start

Generally speaking, users need to type `\start` to begin chatting with the bot. However, if users didn't know this and text other kind of message instead, the bot need to give a response to instruct users click a `\start` button.

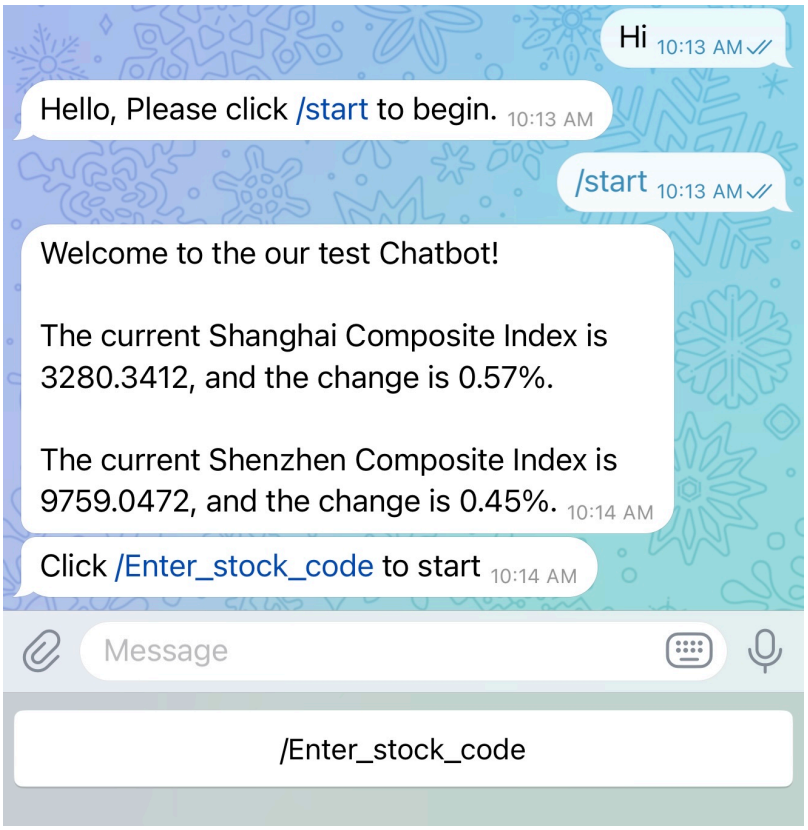
PYTHON

```
async def handle_uninitialized(update: Update, context: CallbackContext) ->
None:
    user_input = update.message.text # what the user send
    if "started" not in context.chat_data:
    context.chat_data["started"] = True
    await update.message.reply_text(f"Hello, Please click /start to
begin.")

    # Send button
    keyboard = [["/start"]]
    reply_markup = ReplyKeyboardMarkup(keyboard, one_time_keyboard=True)



# Define /start instruction function
async def start(update: Update, context: CallbackContext) -> None:
    context.chat_data["started"] = True # Set the chat state to started
    basic = get_ss_and_sz() # Give basic info about stock market
    await update.message.reply_text(f"Welcome to the our test Chatbot!
\n\nThe current Shanghai Composite Index is {basic['SH']}, and the change
is {basic['SH_change']:.2%}.\n\nThe current Shenzhen Composite Index is
{basic['SZ']}, and the change is {basic['SZ_change']:.2%}.")
    # Send button
    keyboard = [["/Enter_stock_code"]] # The next process
    reply_markup = ReplyKeyboardMarkup(keyboard, one_time_keyboard=True,
resize_keyboard=True)
    await update.message.reply_text("Click /Enter_stock_code to start",
reply_markup=reply_markup)
    return ENTER_STOCK_CODE
```

where `get_ss_and_sz` is a function to get Shanghai Composite Index and Shenzhen Composite Index.

PYTHON

```
def get_ss_and_sz():
    ss_df = pro.index_daily(ts_code='000001.SH')
    sz_df = pro.index_daily(ts_code='399001.SZ')
    ss_close = ss_df['close'].values[0]
    ss_open = ss_df['open'].values[0]
    sz_close = sz_df['close'].values[0]
    sz_open = sz_df['open'].values[0]
    ss_change = (ss_close - ss_open) / ss_open
    sz_change = (sz_close - sz_open) / sz_open
    return {"SH":ss_close, "SZ":sz_close, "SH_change":ss_change,
"SZ_change":sz_change} # Use a Hashmap to store the data
```



3.2 Enter Stock Code and Get Stock Data

These 2 states are used to allow user to type in the stock code to get the stock information.

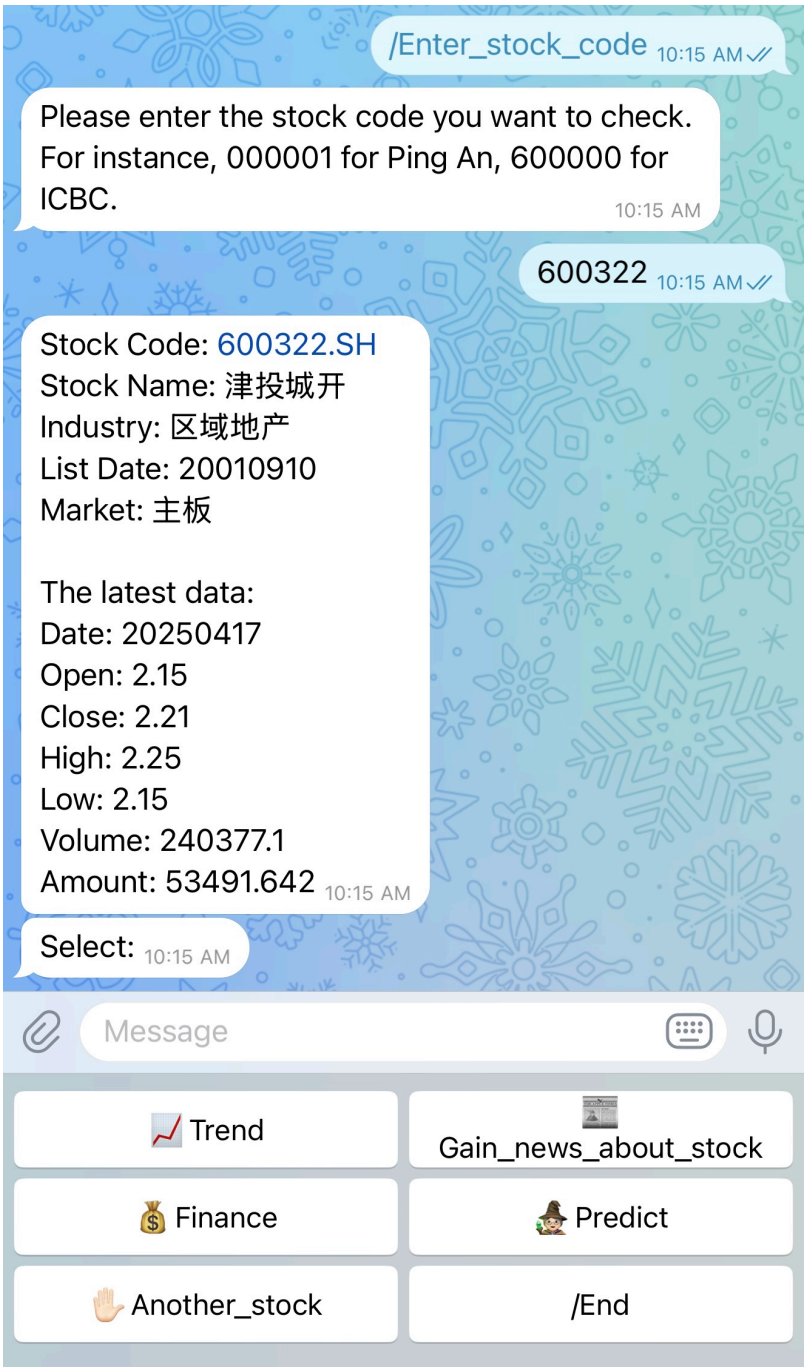
PYTHON

```
async def enter_stock_code(update: Update, context: CallbackContext) ->
int:
    await update.message.reply_text("Please enter the stock code you want
to check. For instance, 000001 for Ping An, 600000 for ICBC.")
    return GET_STOCK_DATA
```

`return GET_STOCK_DATA` means the next state is `GET_STOCK_DATA`, the chat-bot will jump to this state after user type in the stock code. Once jumping to the next state, the bot will call the function `get_stock_data` to get the stock data.

PYTHON

```
async def get_stock_data(update: Update, context: CallbackContext) -> int:
    """ Get the stock data, the detailed code is omitted here """
```



After doing this, the chat-bot will give user a keyboard to select what to do next. Click each button will jump to the corresponding state. Remember, we store which the stock code the user type so that we can use it in the next state. If the user type `Anoter_stock` then the conversation will go back to the `ENTER_STOCK_CODE` state, if the user type `/End` then the conversation will end and all chatting data will be cleared in the `context.chat_data`.

3.3 Further More

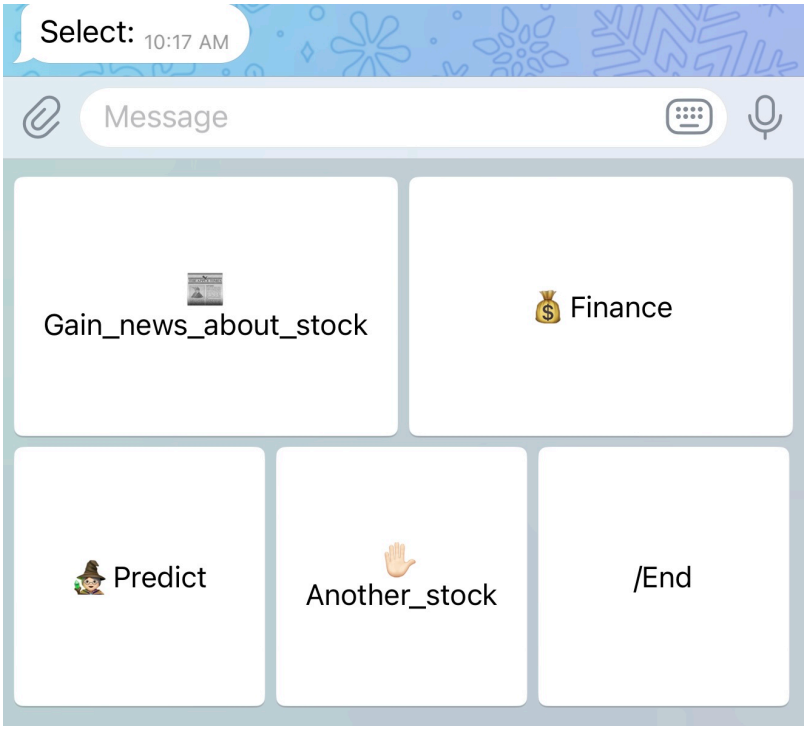
3.3.1 Trend

For this part, we use `matplotlib` to draw the trend of the stock, including Weekly and Monthly trend.

PYTHON

```
async def trend(update: Update, context: CallbackContext) -> int:
  """ Draw the trend of the stock, the detailed code is omitted here """
```

The keyboard will show the next step after the trend is drawn.



3.3.2 Predict

We used some machine learning models to predict the stock price in the future. Certainly we couldn't predict it accurately, neither can any existed models. But we could give a rough prediction based on the historical data. `LSTM` is we were considering to use.

3.3.2.1 Packages used:

PYTHON

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

3.3.2.2 Model training

Use data from 20220101 to 20241231 to train the model; The training size and test size are 80% and 20% respectively.
Use 2-layer LSTM and early-stopping to prevent overfitting.

PYTHON

```python
"""
part of the code
"""
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=
(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')

# early stopping to prevent overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=10)
"""
Some other codes...
"""
# train the model
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32, # Use 32 samples per gradient update
    validation_data=(X_test, y_test),
    verbose=1,
    callbacks=[early_stop]
)
```

For integrality, we also did the model evaluation on the test part of the data.

PYTHON

```
"""
part of the code
"""
rmse = np.sqrt(mean_squared_error(y_test, test_predict[:,0]))
mae = mean_absolute_error(y_test, test_predict[:,0])
relative_error = (mae / y_test.mean()) * 100
```

```
Epoch 39/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0028 - val_loss: 0.0042
Epoch 40/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0025 - val_loss: 0.0040
Epoch 41/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0029 - val_loss: 0.0040
Epoch 42/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0026 - val_loss: 0.0051
Epoch 43/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0030 - val_loss: 0.0048
Epoch 44/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0025 - val_loss: 0.0043
Epoch 45/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0031 - val_loss: 0.0039
Epoch 46/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0026 - val_loss: 0.0044
Epoch 47/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0028 - val_loss: 0.0039
Epoch 48/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0025 - val_loss: 0.0036
Epoch 49/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0024 - val_loss: 0.0040
Epoch 50/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0026 - val_loss: 0.0051
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

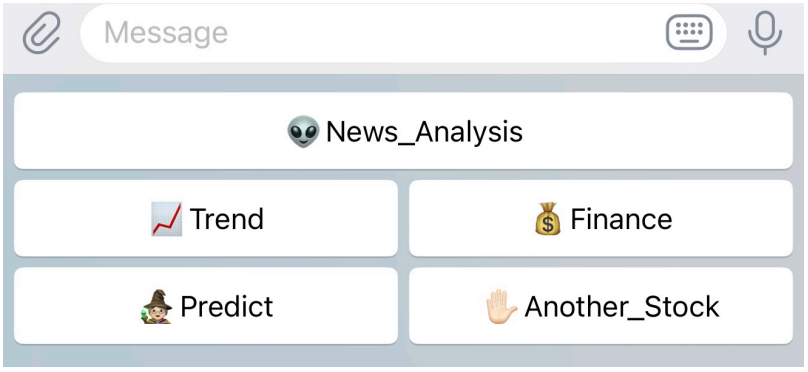
3.3.2.3 Prediction

Now we need to predict the next day price, using the latest 60 days data, as well as the model we just trained. Firstly we need to determine the nearest trading date, and then get the latest 60 valid dates data.

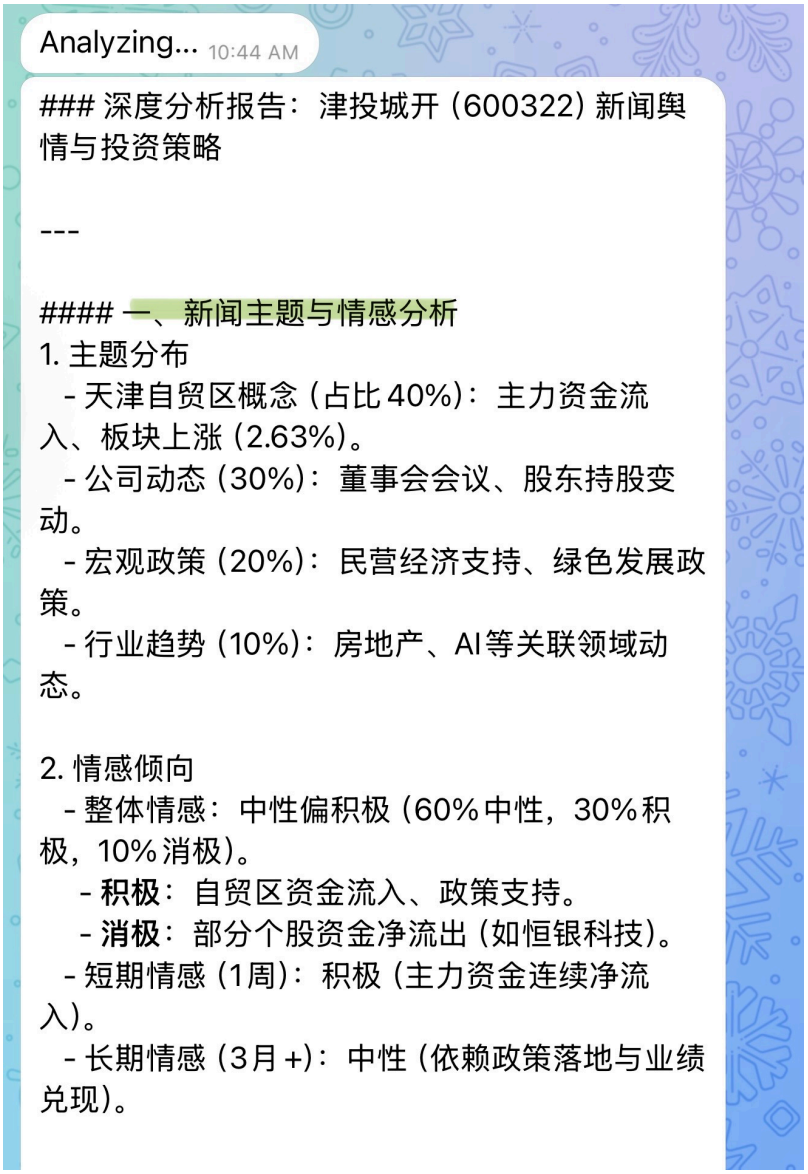
3.3.3 Gain news and analyzed by LLM

After firstly click the button `Gain_news_from_LLM`, the bot will automatically crawl the news about the stock and then use LLM to achieve this function. We decide to connect the bot with `Deepseek LLM` to achieve this function.

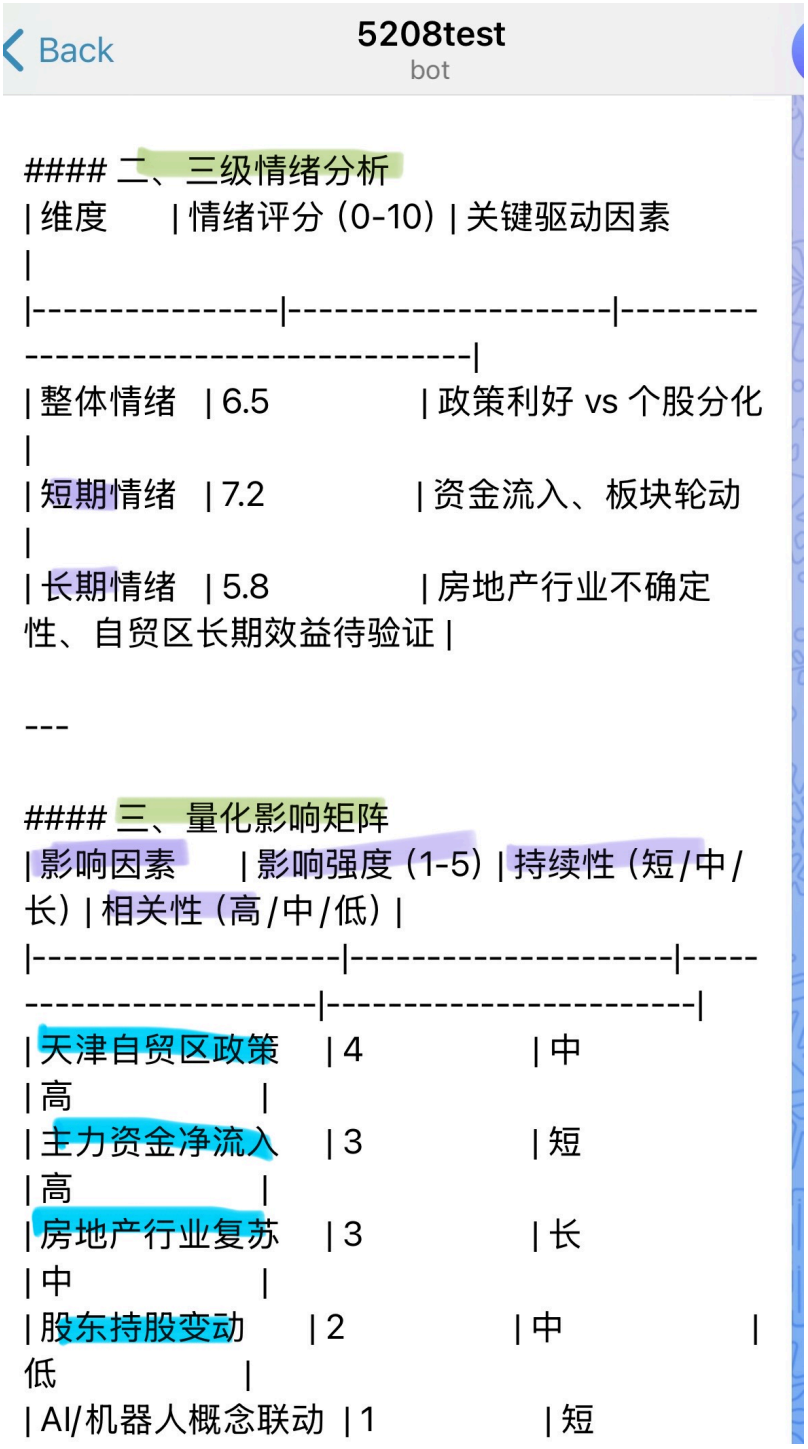
After the chat-box sending you the latest 5 news, you can click the 'news_analysis' boton to let the AI analyze the news and give some suggestions.



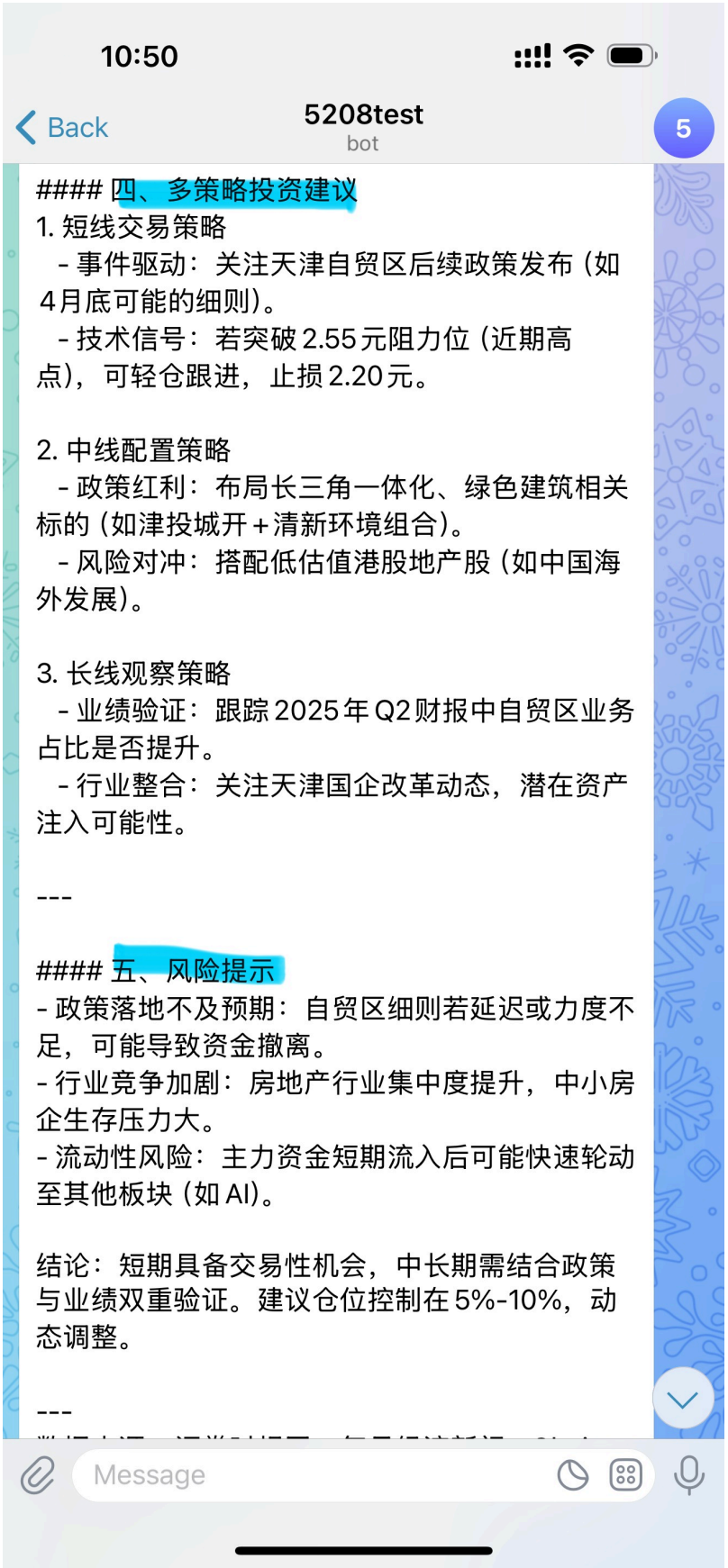
The first part is "Topics and sentiment analysis":



The second and third parts are 3-level sentiment analysis and factors analysis.



At last the chat box will give you suggestions and risk caution.



3.3.4 Finance

This part will give some basic finance information about the firm, including the P/E ratio, EPS, ROE, and so on, which are all important indicators for investors to make decisions. These data will be crawled from `Tushare`.

4 The Main File (Function) to Handle Conversation in Telegram Bot

The main function is used for handling the conversation in the Telegram bot. It is the core of the chat-bot. It must be written in the main file of our project.

```python
def main() -> None:
    token = "my_token" # The token of the bot
    # Create Application Object
    application = Application.builder().token(token).build()
    conv_handler = ConversationHandler(
        entry_points=[
            CommandHandler("start", start),
            MessageHandler(filters.TEXT & ~filters.COMMAND,
handle_uninitialized)
            ], # The entry point, which means we enter the conversation by
typing /start, or other text message which will be handled by
handle_uninitialized

        states={
            ENTER_STOCK_CODE:
[MessageHandler(filters.Regex("/Enter_stock_code"), enter_stock_code)],#
Once we type /Enter_stock_code, we will enter the state of ENTER_STOCK_CODE
and call the function enter_stock_code
            GET_STOCK_DATA: [MessageHandler(filters.TEXT& ~filters.COMMAND,
get_stock_data)],
            FURTHER_MORE: [
                    MessageHandler(filters.Regex("📈Trend"), get_trend),
                    MessageHandler(filters.Regex("📰Gain_info_from_LLM"),
gain_news_about_stock),
                    MessageHandler(filters.Regex("💰Finance"),
get_finance),
                    MessageHandler(filters.Regex("🔮Predict"), predict),
                    MessageHandler(filters.Regex("✋Another_stock"),
another_stock)
                ],
            },
        fallbacks=[
            CommandHandler("end", end)
        ]
    )


    application.add_handler(conv_handler) # Add the conversation handler to
the application
    application.add_error_handler(error)
    # Run the polling
    application.run_polling()


if __name__ == '__main__':
    main()
```