Name: Yang Yuan   Class: AI15

Project Report

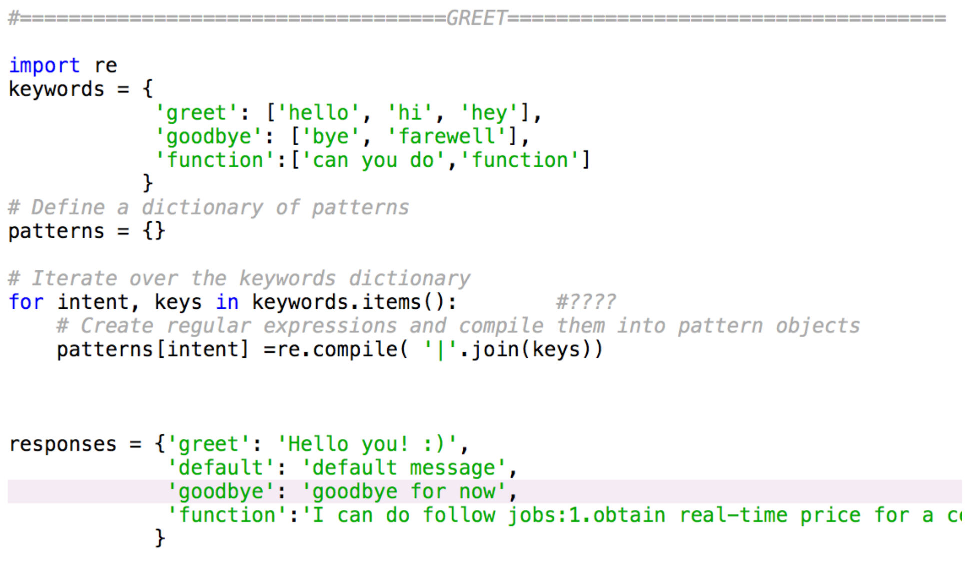
➢ Introduction

In this project, a chatbot is built in Python and can hold conversation and provide stock information. It runs on the Wechat platform and get data through the API of *iexfinance.*

In order to realize building functions include information processing, finding results through API and outputting results in a dialog form, I basically apply the knowledge of regular expression, pattern matching, keyword extraction and syntactic transformation to finish the meeting part and use the construction of local basic chatbot system based on Rasa NLU as well as multi-round technology of state machine to finish the stock consulting part.

➢ Greeting Part

In the greeting part, the most important thing is to find out the information related to greeting and output the corresponding answer. A dictionary of keywords and their intents and a dictionary includes the answer corresponding to the intent are built to solve it as shown in the Fig.1 above.

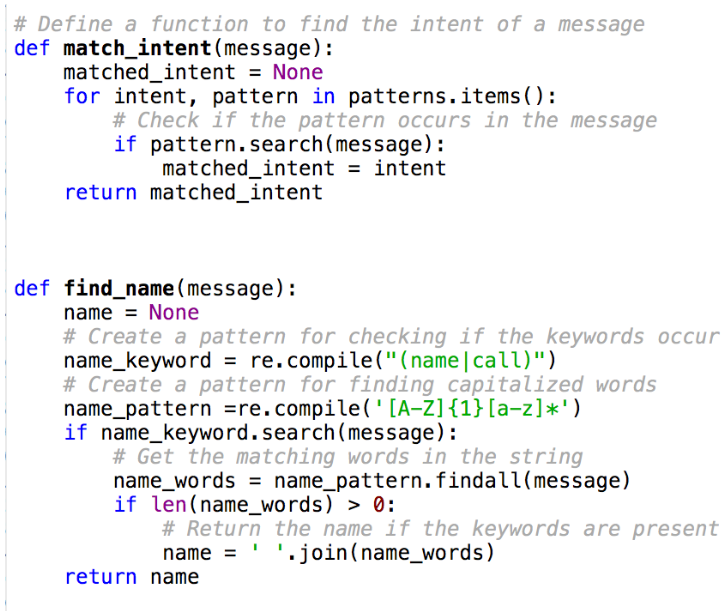


```
#==================================GREET==================================

import re
keywords = {
            'greet': ['hello', 'hi', 'hey'],
            'goodbye': ['bye', 'farewell'],
            'function':['can you do','function']
           }
# Define a dictionary of patterns
patterns = {}

# Iterate over the keywords dictionary
for intent, keys in keywords.items():        #????
    # Create regular expressions and compile them into pattern objects
    patterns[intent] =re.compile( '|'.join(keys))


responses = {'greet': 'Hello you! :)',
             'default': 'default message',
             'goodbye': 'goodbye for now',
             'function':'I can do follow jobs:1.obtain real-time price for a c
            }
```

Fig.1 Dictionary related to greeting part

The function `match_intent` is used to get intent from the message and send the intent information to function `respond`, and then respond will choose the answer corresponding to the intent to output.

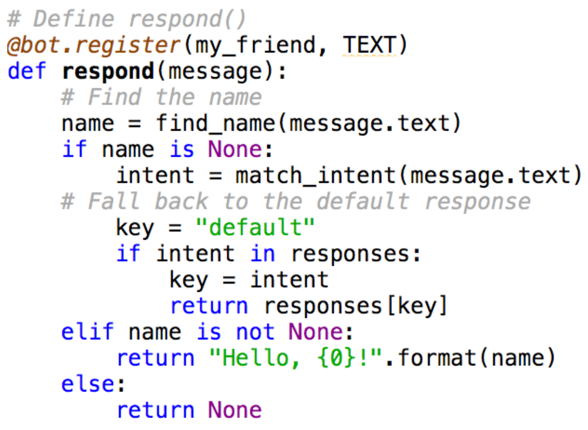
The function `find_name` uses regular expression to find the name in the message since the pattern for names usually begins with a capital letter and ends with a lower-case letter.



```python
# Define a function to find the intent of a message
def match_intent(message):
    matched_intent = None
    for intent, pattern in patterns.items():
        # Check if the pattern occurs in the message
        if pattern.search(message):
            matched_intent = intent
    return matched_intent


def find_name(message):
    name = None
    # Create a pattern for checking if the keywords occur
    name_keyword = re.compile("(name|call)")
    # Create a pattern for finding capitalized words
    name_pattern =re.compile('[A-Z]{1}[a-z]*')
    if name_keyword.search(message):
        # Get the matching words in the string
        name_words = name_pattern.findall(message)
        if len(name_words) > 0:
            # Return the name if the keywords are present
            name = ' '.join(name_words)
    return name
```

Fig.2 Functions related to match the intent

In Fig.3, it shows how decisions are made to figure out whether the robot should answer “Hello! :)” or answer with the user’s name such as “Hello, John!”

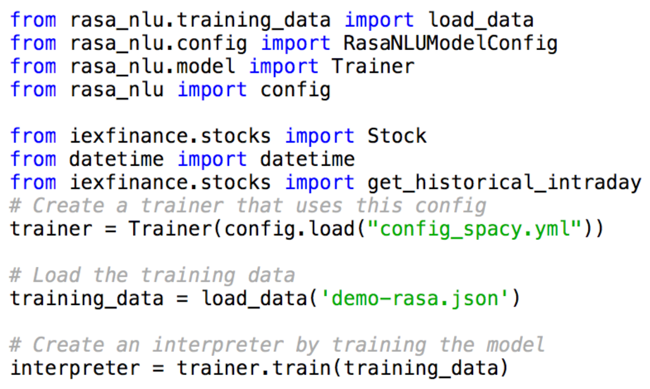


```python
# Define respond()
@bot.register(my_friend, TEXT)
def respond(message):
    # Find the name
    name = find_name(message.text)
    if name is None:
        intent = match_intent(message.text)
    # Fall back to the default response
        key = "default"
        if intent in responses:
            key = intent
            return responses[key]
    elif name is not None:
        return "Hello, {0}!".format(name)
    else:
        return None
```

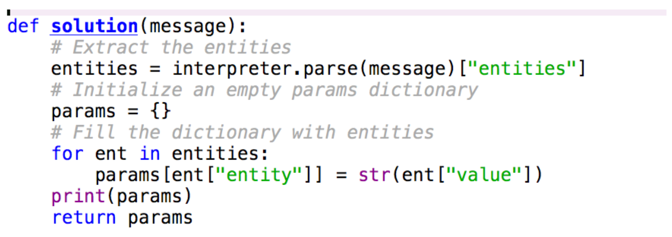
Fig.3 Function to output greeting answers

➢ **Stocks Part**

In the Stock part, we have the training part to identify the intent and entities in a message owe to some examples in the training data, as shown in the Fig.4. The function `interpreter.parse()` is used to extract entities from the whole message.



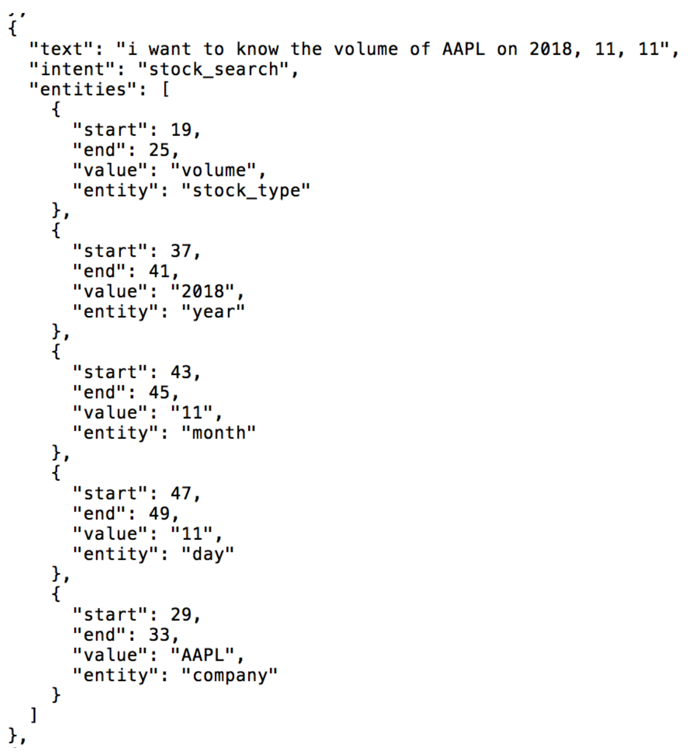
```python
from rasa_nlu.training_data import load_data
from rasa_nlu.config import RasaNLUModelConfig
from rasa_nlu.model import Trainer
from rasa_nlu import config

from iexfinance.stocks import Stock
from datetime import datetime
from iexfinance.stocks import get_historical_intraday
# Create a trainer that uses this config
trainer = Trainer(config.load("config_spacy.yml"))

# Load the training data
training_data = load_data('demo-rasa.json')

# Create an interpreter by training the model
interpreter = trainer.train(training_data)
```



```python
def solution(message):
    # Extract the entities
    entities = interpreter.parse(message)["entities"]
    # Initialize an empty params dictionary
    params = {}
    # Fill the dictionary with entities
    for ent in entities:
        params[ent["entity"]] = str(ent["value"])
    print(params)
    return params
```

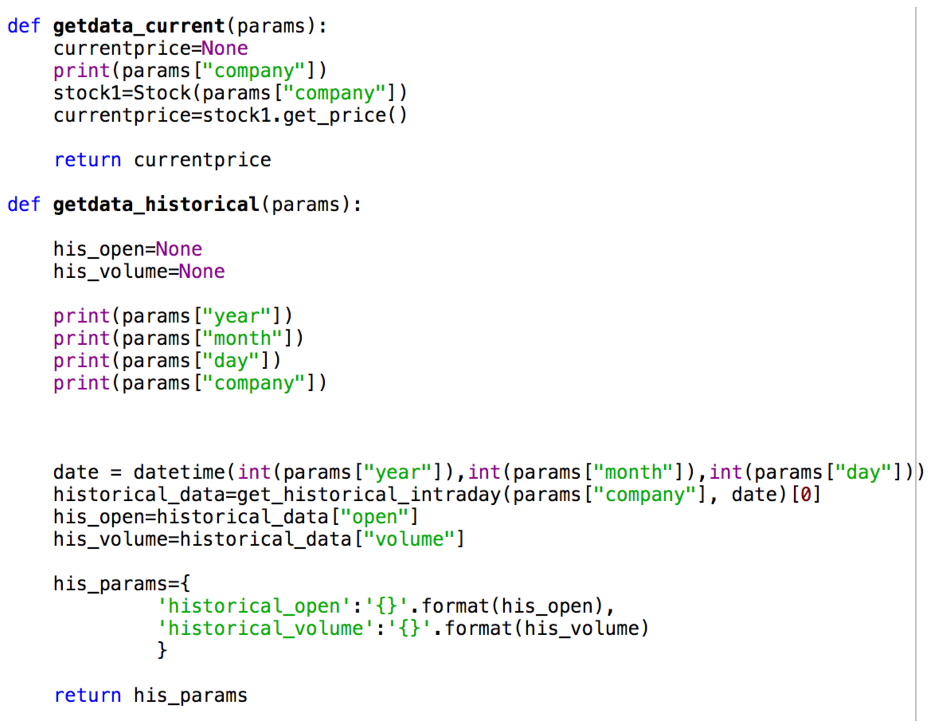
Fig.4 Training part

There is an example for a training data showed in json form:



```json
{
  "text": "i want to know the volume of AAPL on 2018, 11, 11",
  "intent": "stock_search",
  "entities": [
    {
      "start": 19,
      "end": 25,
      "value": "volume",
      "entity": "stock_type"
    },
    {
      "start": 37,
      "end": 41,
      "value": "2018",
      "entity": "year"
    },
    {
      "start": 43,
      "end": 45,
      "value": "11",
      "entity": "month"
    },
    {
      "start": 47,
      "end": 49,
      "value": "11",
      "entity": "day"
    },
    {
      "start": 29,
      "end": 33,
      "value": "AAPL",
      "entity": "company"
    }
  ]
},
```

Fig.5 Example in json form



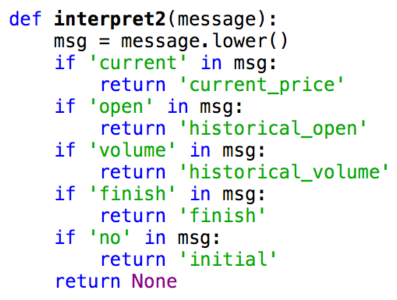
```python
def getdata_current(params):
    currentprice=None
    print(params["company"])
    stock1=Stock(params["company"])
    currentprice=stock1.get_price()

    return currentprice

def getdata_historical(params):

    his_open=None
    his_volume=None

    print(params["year"])
    print(params["month"])
    print(params["day"])
    print(params["company"])


    date = datetime(int(params["year"]),int(params["month"]),int(params["day"]))
    historical_data=get_historical_intraday(params["company"], date)[0]
    his_open=historical_data["open"]
    his_volume=historical_data["volume"]

    his_params={
             'historical_open':'{}'.format(his_open),
             'historical_volume':'{}'.format(his_volume)
             }

    return his_params
```

Fig. 6 Find useful data

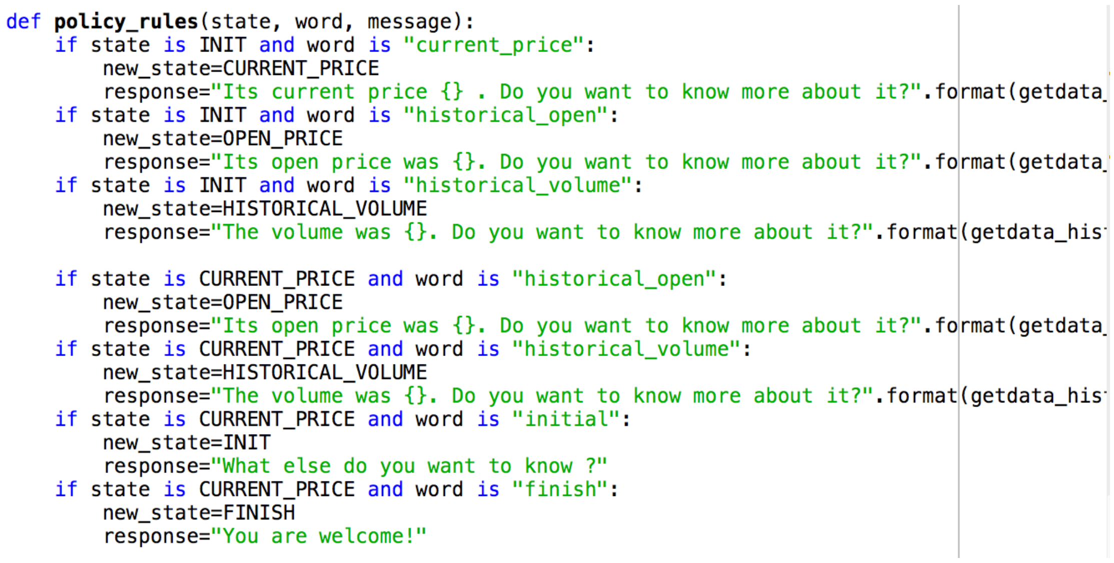
As shown in the Fig.6 above, function `getdata_current` extract the data about current price and then send the data to `policy_rules` and fill blanks in the sentences with figure data which the user wants to obtain. In the same way, the function `getdata_historical` send the historical data which is extract from the `params`. Therefore, the information will be successfully shown up in the final answer.

In order to realize multi-round consulting, I apply the knowledge of state machines. In this project, five states are used to express the begin of the consulting process, three functions of this chatbot and the end of conversation.

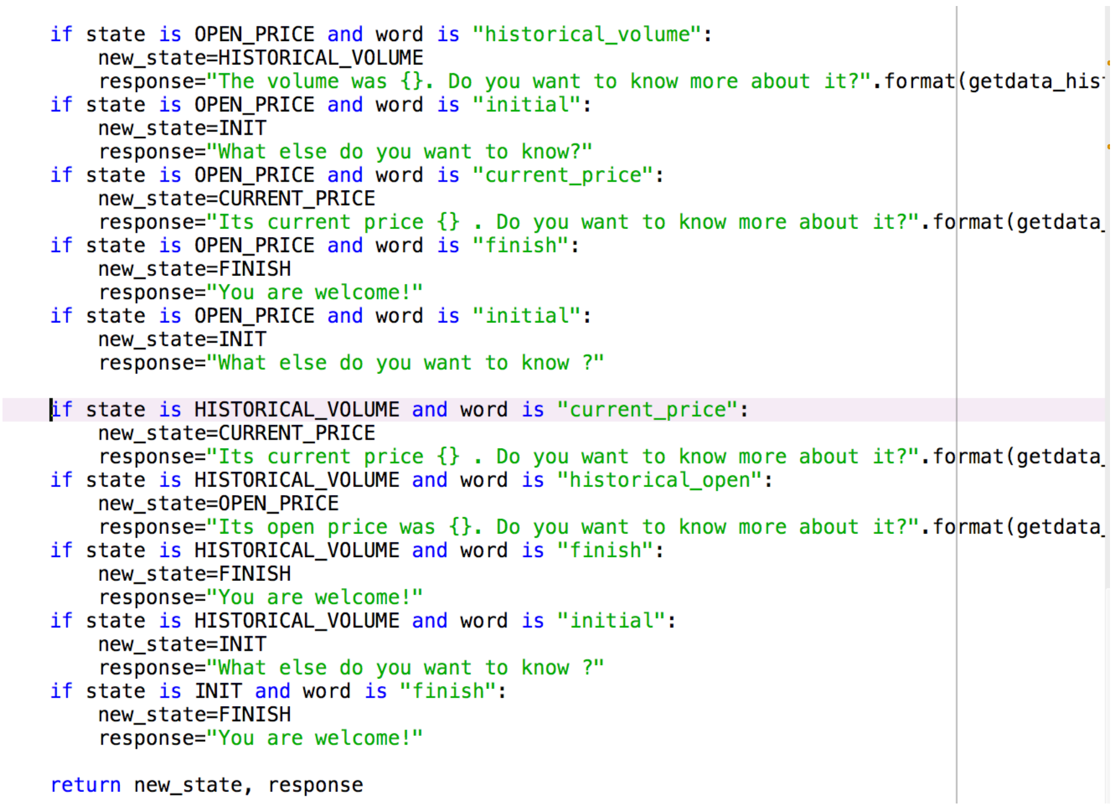
The function `interpret2` firstly find out the intent of the message and decide which state the next state will become.



```
def interpret2(message):
    msg = message.lower()
    if 'current' in msg:
        return 'current_price'
    if 'open' in msg:
        return 'historical_open'
    if 'volume' in msg:
        return 'historical_volume'
    if 'finish' in msg:
        return 'finish'
    if 'no' in msg:
        return 'initial'
    return None
```



```
def policy_rules(state, word, message):
    if state is INIT and word is "current_price":
        new_state=CURRENT_PRICE
        response="Its current price {} . Do you want to know more about it?".format(getdata_
    if state is INIT and word is "historical_open":
        new_state=OPEN_PRICE
        response="Its open price was {}. Do you want to know more about it?".format(getdata_
    if state is INIT and word is "historical_volume":
        new_state=HISTORICAL_VOLUME
        response="The volume was {}. Do you want to know more about it?".format(getdata_his

    if state is CURRENT_PRICE and word is "historical_open":
        new_state=OPEN_PRICE
        response="Its open price was {}. Do you want to know more about it?".format(getdata_
    if state is CURRENT_PRICE and word is "historical_volume":
        new_state=HISTORICAL_VOLUME
        response="The volume was {}. Do you want to know more about it?".format(getdata_his
    if state is CURRENT_PRICE and word is "initial":
        new_state=INIT
        response="What else do you want to know ?"
    if state is CURRENT_PRICE and word is "finish":
        new_state=FINISH
        response="You are welcome!"
```



```
if state is OPEN_PRICE and word is "historical_volume":
    new_state=HISTORICAL_VOLUME
    response="The volume was {}. Do you want to know more about it?".format(getdata_his
if state is OPEN_PRICE and word is "initial":
    new_state=INIT
    response="What else do you want to know?"
if state is OPEN_PRICE and word is "current_price":
    new_state=CURRENT_PRICE
    response="Its current price {} . Do you want to know more about it?".format(getdata_
if state is OPEN_PRICE and word is "finish":
    new_state=FINISH
    response="You are welcome!"
if state is OPEN_PRICE and word is "initial":
    new_state=INIT
    response="What else do you want to know ?"

if state is HISTORICAL_VOLUME and word is "current_price":
    new_state=CURRENT_PRICE
    response="Its current price {} . Do you want to know more about it?".format(getdata_
if state is HISTORICAL_VOLUME and word is "historical_open":
    new_state=OPEN_PRICE
    response="Its open price was {}. Do you want to know more about it?".format(getdata_
if state is HISTORICAL_VOLUME and word is "finish":
    new_state=FINISH
    response="You are welcome!"
if state is HISTORICAL_VOLUME and word is "initial":
    new_state=INIT
    response="What else do you want to know ?"
if state is INIT and word is "finish":
    new_state=FINISH
    response="You are welcome!"

return new_state, response
```

Fig. 7 State machine part

In this process, the initial state INIT will change if it finds out that the intent is to ask for one of the three functions. Then, depending on what the user wants to know, the present state changes to the corresponding function state. Once the user wants to finish the conversation, the present state changes into FINISH state. The whole process is shown as below.

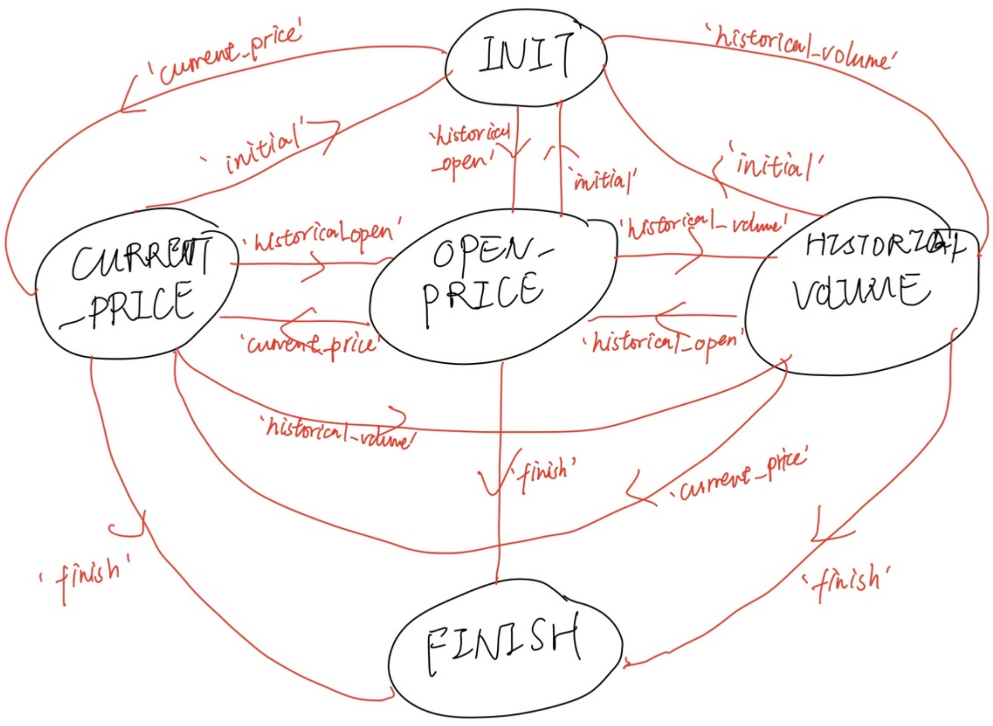


Fig. 8 State machine diagram

➢ Conclusion and Gains

In the project, I have made a chatbot which can do simple conversations and especially can tell users the stock information corresponding to what the user wants. In the greeting part, regular expression, pattern matching and keyword extraction are used to figure out several simple intents and sent correct answers to the user. In the stocks part, important entities are extracted through training model construction based on Rasa NLU technology. Multi-round and multiple query finally succeed due to the knowledge of state machines. However, the training data in the training file is still poor and need to be expanded.

I have learned a lot of new things through this whole project. Since this is the first time for me to program by Python, there are a lot of difficulties in transforming my thoughts to Python language. However, examples shown during the class are quite useful.

➢ References

[1] https://pypi.org/project/iexfinance/

[2] https://pypi.python.org/pypi/wxpy/0.3.9.8

[3] https://github.com/RasaHQ/rasa_nlu