Faizan Naghman
CS-171
Project, Part 4

Introduction:

I am planning to extend the chatbot to compute some basic queries about FOREX currencies. I thought it would be exciting to pick a domain where I could aim to do some quantitive analytics on the parsed input of the user. I also thought that having a real-valued dataset would probably make it more conducive for the chatbot to respond to a wider array of queries with a higher degree of mathematical reasoning. My goal is to make the knowledge base of the chatbot capable enough to parse the input query, determine the parameters and needed to yield the desired result, and compute/evalulate the output using those parameters. I am hoping to handle similar input: “Which Forex currency appreciated the most relative to USD yesterday?”.

Part 3 Progress:
I am fairly set on following the approach as explained above in Part-2 Report. I am not planning on deviating from the Part-2 introduction for the remainder of the project as of now.

Part 4 update:
Achieved the planned target reasonably closely to the original idea except for the prediction in Phase 4 is given based on daily moving averages instead of historical prices.

Technical Overview:

I would probably write the entire code in python. I am planning on using NumPy for calculations. I will also use an API to extract forex data in real-time. The information accessed from APIs will be relative exchange rates of the requested forex over a specified time period. All the computations needed to answer those queries will be done by me.

Part 3 Progress:

1) I am using the following API (Foreign exchange rates and currency conversion JSON API) to retrieve the required data. I used two API endpoints in this part

to achieve my planned objective for Phase 1 and Phase 2 of the project. Additionally, one API call was made to store all listed Forex currencies to prevent numerous API calls while assigning lexical categories to each word of the input sentence.

2) I did not feel the need to use NumPy as yet because neither of the Phases required the use of NumPy.
3) I have included the DateTime library to format the date as iso8601. Datetime library has not been used for any other purpose.
4) I also used the request library to make get calls to the API and JSON library to handle the retrieved payload from API.

Part 4 update:

Each query to the chatbot must be specified as the sequence of line first in the original version of the chatbot i.e:

```
 T, P = CYKParse.CYKParse(['what', 'is','the','recommended', 'advice', 'for', 'USD', 'to', 'PKR'], CYKParse.getGrammarWeather())
   sentenceTree=(getSentenceParse(T))
   updateRequestInfo(sentenceTree)
   reply()
```

List of addions to part 4:

1) Integrated an additional api to access 5 different features for a specified Forex Symbol
2) Integrated an api to retrieve the support, resistance and pivot points for a forex pair.
3) Integrated an API to return a recommended strategy, buy or sell or neutral, for a forex pair

Details about the domain, limitations and running the chatbot are attached in requirements.txt

Programming Phases:

Part-3:

- **Phase 1:** Complete the required grammar and lexical categories/entries required to process basic queries such as “What is one USD to AUD”. I will test the input from various APIs to see which one seems to work/has a convenient implementation.

- **Phase 2:** Extend phase 1 to respond to various tim7e-specific comparisons such as “What is one USD to AUD today/yesterday/1 year/6 months/ 18 hours”. The goal would be to successfully identify different units of time.

  Lexical Changes:
  I extended the getGrammarWeather() function to cover the required grammar for Phase 1 and 2. I added three additional lexical categories to uniquely identify the key elements required: **Forex, Date,** and **Amount.**
  Since the API has 1932 possible pairs of Forex, I thought it would be more convenient to make a set of all Forex currencies and store it in a separate module as a set that can be used to check if a word is Forex symbol. I used similar reasoning for dates and amount. Considering that a user can ask for any date between now and 1995, and the conversion amount (X USD to EUR) could be any positive real number, it did not make much sense to hardcode all of those possibilities.
  Hence, I have not hardcoded any rules for Forex, Date, and Amount. All words are conditionally checked in getGrammarLexicalRules method. The screenshot of implementation is attached below:

```
def getGrammarLexicalRules(grammar, word):
    if fs.checkForexSymbol(word):
        yield 'Forex', 0.25
    try:
        dt.date.fromisoformat(word)
        yield 'Date', 0.1

    except Exception:
            pass

    for rule in grammar['lexicon']:
        if rule[1] == word:
            yield rule[0], rule[2]
    try:
        float(word)
        yield 'Amount', 0.1
    except Exception:
            pass
```

Grammar Changes:
I have added the following categories to encompass the possible grammar inputs:
['NP', 'Date', None, 0.1],
['NP', 'Forex', 'DatePhrase', 0.1],
['NP', 'Forex', None, 0.1], #C1F Rule: NP-> Name[0.1]
['NP', 'AdverbPhrase', 'DatePhrase', 0.3], #Added for part 5 for types 'in' 'city'
['DatePhrase', 'Preposition', 'Date', 1.0]
['NP', 'Amount', 'NP', 0.2],

The aim was to constrain the Lexical categories to exist in a fairly rigid order. For instance, Amount must always be a prefix of a Forex (X dollars). It does not make much sense for a quantifier to be somewhere else in the sentence. Date must always follow a preposition in the context of the sentence (in 2008, on 2020-10-05). Two forex currencies must be connected by a preposition (USD to PKR, AUD to GBP). As long as the query is asked as a question beginning with What and How, these constrains do a good job at allowing decent variations of the sentences through a relatively simple set of rules.

Sample input:

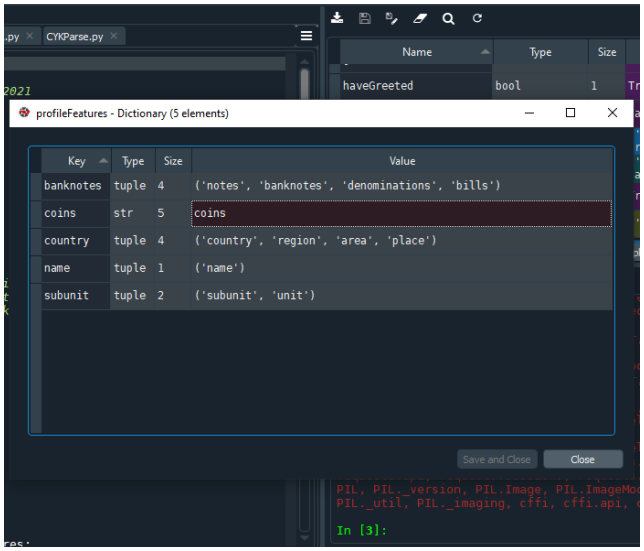
CYKParse(['what','is', 'the', 'worth', 'of', '94995.222', 'USD', 'in', 'PKR', 'on', '1997-08-01'], getGrammarWeather())

'S/0/10:[S[WQuestion what][VP[Verb is][NP[Article the][NP[NP[AdverbPhrase [Adverb worth]][NP[Article of][NP[Amount 94995.222][NP [Forex USD]]]]][AdverbPhrase[Preposition in][NP[Forex PKR][DatePhrase[Preposition on][Date 1997-08-01]]]]]]]']

Part-4 update:

Phase 3: Add to the range of answerable queries, such as, “How much is 1000 USD in Kuwaiti Dinar”, “Was Indian Rupee worth more than USD in 2004?”. These queries won’t necessarily require heavy computations to be performed on the data pulled from the APIs

3.1) The original approach turned out to be a bit more complex, so I decided to extend the answerable queries by processing queries requesting a specified profile features for all of the currencies such as [banknotes, subunits, country, name, coins]. Each feature can be referred to by one or more alternative words as shown in the screenshot below:



Sample inputs:

1) **What are the banknotes in {symbol}**
2) **What notes are in {symbol}**
3) **Which bills are in {symbol}**
4) **What is the subunit of {symbol}**
5) **What is the {Profile feature} of {symbol}**

6) **What {profile feature} is/are of {symbol}**

3.2) pivot call can be triggered by any of the words in the array -> ['pivots', 'pivot', 'indicators', 'support', 'resistance']

1) **What are the pivots of {symbol 1} to {symbol 2}**
2) **What is the recommended strategy for USD to PKR**

- **Phase 4:** Respond to complex queries that require some level of decision making/judgment, such as “how much did the USD appreciate relative to GBP from 2000 to 2020.

Part 4 update:

1) **Instead of historical data, a more accurate approach forecast is given by providing a trade recommendation-> [strong buy, buy, neutral, sell, strong sell]**
2) **Can be triggered by using any of the following words in the query: 'recommendation', 'recommended', 'strategy', 'prediction', 'forecast', 'advice'**

Input Handling:

I am planning on classifying all of the Forex currencies as a subset of nouns rather than defining a new category. Most of the keywords for comparisons are either adjectives or verbs, such as (rose, fell, increased, appreciate, buy, safe, worth), that can be decomposed into C1F/CNF grammar with some modifications. For phase 1, I might have to add/modify some lexical categories such as “WQuestions” to categorize different types of inquiries. For the later phases, I am planning on constraining the project around a finite set of keywords that can be used to make inquiries because all of the grammar/lexical rules will be hard-coded.
For example, the constraint might only allow the input sentence to use certain prepositions or conjunctions to relate two forex currencies (Is AUD performing better “against” USD today, How much is one AUD “to” USD, Did AUD appreciate more “than” USD in 2010?).

Part-3:

I deviated from my planned approach to reuse existing categories. I think it eased the process of separating the keywords that would be subsequently used as parameters for the external API.
I reused Articles, Adverbs, and Prepositions as appropriate.

Part 4)

Most of the additional words and categories required were sufficiently fitting a defined grammatical structure. Therefore I decided to reuse the grammar from earlier parts by grouping the required words into existing categories. Profilefeatures, recommendations, and pivot trigger words have been categorized as adverbs and minor extentions have been made to the grammar and lexicons.

Internal Representation:

I will need to include a new class to model json data objects retrieved through forex API (currencylayer API Documentation (apilayer)). A new rest call will be made to access data in real-time for every input inquiry to the chatbot. I don't think I need to store the retrieved data internally or on a dabatabse because the target information is fairly simple and does not branch out into further collections/sub-collections, and in fact only has a single ordered pair feature (price, time). Therefore, I think making a REST call seems less tedious and more efficient than trying to make my own dummy dataset. However, if the project ends up using more technical features such as moving averages, volatility measures, etc, I might have to consider using a database for more efficient access and storage of data.
For parsing, the current parse tree data model should be sufficient to work with all four phases of the project.
For every input inquiry, depending on the category of grammar and lexicons, the program will first parse it to identify the most probabilistic inquiry made. The program will then pick out the keywords and forex currencies to determine the particular comparison needed to perform (convert currency, measure relative performance for a specified time "appreciated/depreciated", give trade suggestion "positive about buying/selling"). This will be done by getting the associated words for each pre-determined key-lexicon category specific to that syntax and type of inquiry made.

I will use Numpy to do the math for most comparisons, such as calculating relative growth, and evaluating relative performance.

Part-3:

1) All the pair forex combinations and their latest conversion rates are stored in “ForexData.txt” in the root of the project. I could only make 500 API calls, so it was a necessity to have a local repo of the forex symbols and pairs to avoid recalling the API.
2) “Forexsymbols.py” module in the root of the folder is used as a pseudo-RestController to call the API. It also has two data objects that hold all possible ForexPairs as well as singular Forex symbols that are in the domain of the API. Forexpairs and forexSymbol using the following functions:

```
def checkForexSymbol(word):
    return word in forexSet

def checkForexPair(word):
    return word in forexPair
```

3) UpdateForexList can be called to update the “ForexData.txt” files. I usually do it once everyday to keep the data updated. It simply calls the API and dumps the received payload to a text file.
4) The module initializes the allowable forex pairs and forex symbols by extracting the relevant details (symbol and id) from the forexdata.txt file
5) The RESTCALL to API is made by BinaryForexConversion. If the user requires real-time latest conversion rates, then a time object is not required and the relevant key within the payload dictionary can be used to retrieve the current exchange rate. If the timeobject is present, some extra work needs to be done because the API does not support a static time-reference call at a single point in history (for reference: https://fcsapi.com/document/forex-api#historicalprice). The API can return as many as 900 different exchange rates of the forexpair within that time window. I just end up using the closest one to the date mentioned in the question.
6) I made some changes to Proj1.py file. RequestInfo only includes the keywords (currency1, currency2, amount, date) that are required for parsing.

7) The updateRequestMethod does intializes Date and Amount (if present) along with some conditions to ensure that requestInfo initializes currency1 and currency2 in the right order.

Part 4:

1) **Profile features are hardcoded and a function call can be made to see if a word refers to profile features.**
2) **Pivots and recommendation have a range of alternative words as references to trigger a response to retrieve pivot points and trading recommendation for a given forex pair.**

Output:

I am planning on reusing the existing grammar to produce output to the user. I will include Additional vocabulary terms in the lexicon categories that can produce intelligible output to the user. Depending on the type of question/inquiry, a syntatical structure would be determined, and populated with the return parameters that will be computed using the approach described in earlier sections.

Part 3:

Replies are currently categorized in three 4 ways

1) If the user has been greeted and the next input does not provide a valid pair of forex currencies, then the list of all valid pair is returned for the user to choose a valid pair
2) If the date is not given, then the output is of the following form:

```
if date is None:
        print('The latest conversion data suggests that as of this
moment {} {} in {} is {}'.format(amount, c1, c2, conversion))
```

3) Otherwise, the date is present and the output is given by the following code in continuation of the previous if statement.

```
        else:
            print('The conversion rate of {} {} in {} was {} on
{}'.format(amount, c1, c2, conversion, date))
```

Input:

T, P = CYKParse.CYKParse(['what', 'is', 'GBP', 'to', 'BTC'], CYKParse.getGrammarWeather())

Sample Output: getSentenceParse S/0/4
Hello, what forex conversion query would you like to run?
GBP BTC 1.0 None
200 ****INFO*****:Successfully retrieved conversion data
The latest conversion data suggests that as of this moment 1.0 GBP in BTC is 2.8e-05

T, P = CYKParse.CYKParse(['how','much', 'is', '805.898', 'USD', 'in', 'PKR', 'on', '2020-08-01'], CYKParse.getGrammarWeather())

getSentenceParse S/0/8
USD PKR 805.898 2020-08-01
200 ****INFO*****:Successfully retrieved conversion data0
The conversion rate of 805.898 USD in PKR was 134721.96865999998 on 2020-08-01

T, P = CYKParse.CYKParse(['what','is', 'the', 'worth', 'of', '94995.222', 'EUR', 'in', 'AUD', 'on', '1997-08-01'], CYKParse.getGrammarWeather())

getSentenceParse S/0/8
EUR AUD 94995.222 2020-08-01
200 ****INFO*****:Successfully retrieved conversion data
The conversion rate of 94995.222 EUR in AUD was 156856.1105664 on 2020-08-01

T, P = CYKParse.CYKParse(['what','is', 'the', 'value','of', '999999999999', 'USD', 'in', 'BTC', 'on', '2020-08-01'], CYKParse.getGrammarWeather())

getSentenceParse S/0/8
USD BTC 999999999999.0 2020-08-01
200 ****INFO*****:Successfully retrieved conversion data
The conversion rate of 999999999999.0 USD in BTC was 84699999.9999153 on 2020-08-01

Part 4 update:
Input:

```
T, P = CYKParse.CYKParse(['what', 'is','the','recommended', 'advice', 'for', 'USD', 'to', 'PKR'], CYKParse.getGrammarWeather())
   sentenceTree=(getSentenceParse(T))
   updateRequestInfo(sentenceTree)
   reply()

   T, P = CYKParse.CYKParse(['what','are', 'the', 'banknotes', 'of', 'USD'], CYKParse.getGrammarWeather())
   sentenceTree=(getSentenceParse(T))
   updateRequestInfo(sentenceTree)
   reply()

   T, P = CYKParse.CYKParse(['what','are', 'the', 'pivots', 'and', 'recommendation', 'for', 'USD', 'to', 'PKR'], CYKParse.getGrammarWeather())
   sentenceTree=(getSentenceParse(T))
   updateRequestInfo(sentenceTree)
   reply()

   T, P = CYKParse.CYKParse(['what','is', 'the', 'value', 'of', '40', 'USD', 'in', 'INR'], CYKParse.getGrammarWeather())
   sentenceTree=(getSentenceParse(T))
   updateRequestInfo(sentenceTree)
   reply()
```

Output:

```
Hello, what forex conversion query would you like to run?
The trading strategy based on the moving average is:  Strong Sell
The banknotes of USD are $1, $5, $10, $20, $50, $100
```

155.9117 is the pivot price of USD to PKR based on a support level 156.5983 and resistance level 155.0633

The trading strategy based on the moving average is: Strong Sell