Go Through Code for spread-lang

In this section, we will be running certain code from our electron based REPL and explain the use case with screenshots of output.

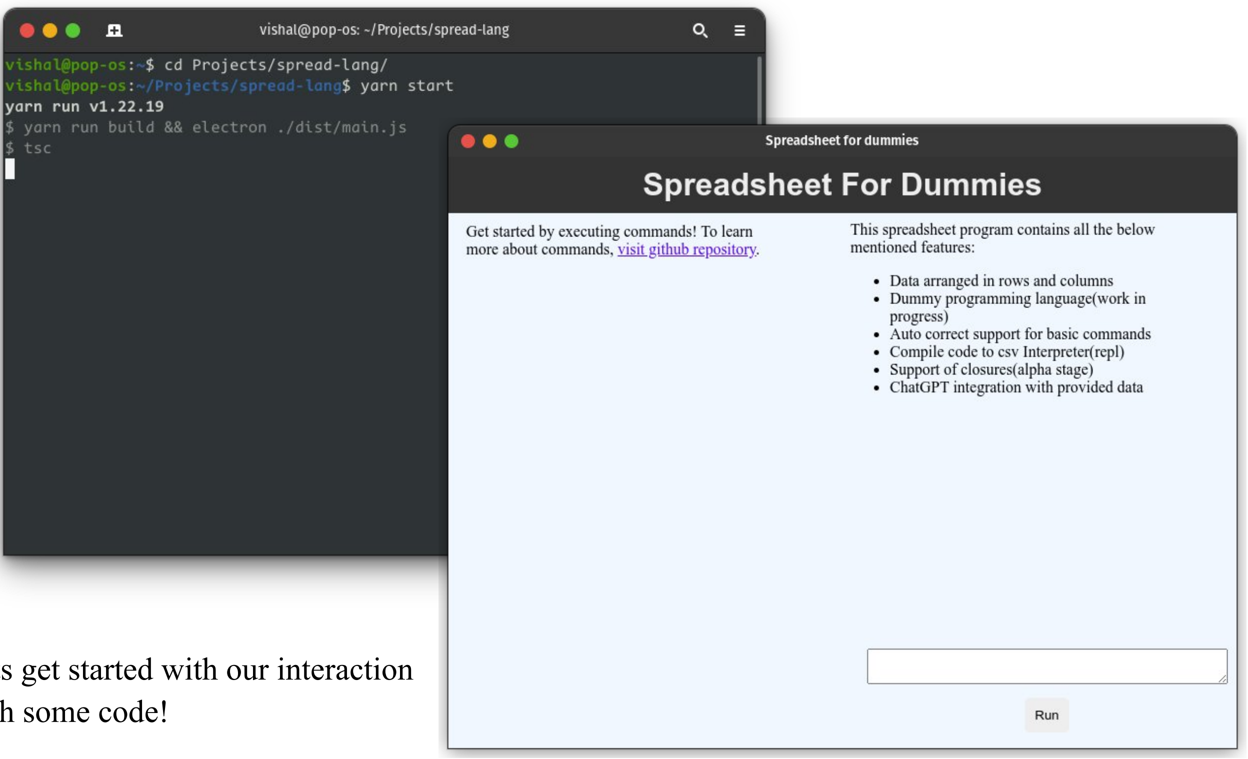
Installation

Follow the basic installation steps mentioned here.

Getting Started

Open the directory of spread-lang frontend.

Start the electron app using “yarn start” or “npm start”.



```
vishal@pop-os:~$ cd Projects/spread-lang/
vishal@pop-os:~/Projects/spread-lang$ yarn start
yarn run v1.22.19
$ yarn run build && electron ./dist/main.js
$ tsc
```

Spreadsheet For Dummies

Get started by executing commands! To learn more about commands, visit github repository.

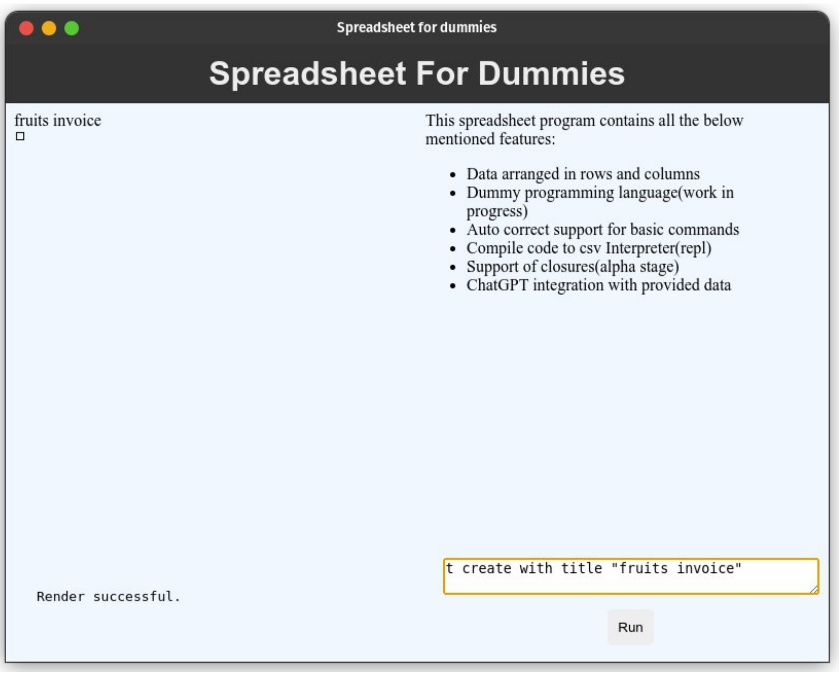
This spreadsheet program contains all the below mentioned features:

- Data arranged in rows and columns
- Dummy programming language(work in progress)
- Auto correct support for basic commands
- Compile code to csv Interpreter(repl)
- Support of closures(alpha stage)
- ChatGPT integration with provided data

Lets get started with our interaction with some code!

Lets first start by creating a table with title “fruits”, with command:

```
t create with title "fruits invoice"
```



Spreadsheet For Dummies

fruits invoice

This spreadsheet program contains all the below mentioned features:

- Data arranged in rows and columns
- Dummy programming language(work in progress)
- Auto correct support for basic commands
- Compile code to csv Interpreter(repl)
- Support of closures(alpha stage)
- ChatGPT integration with provided data

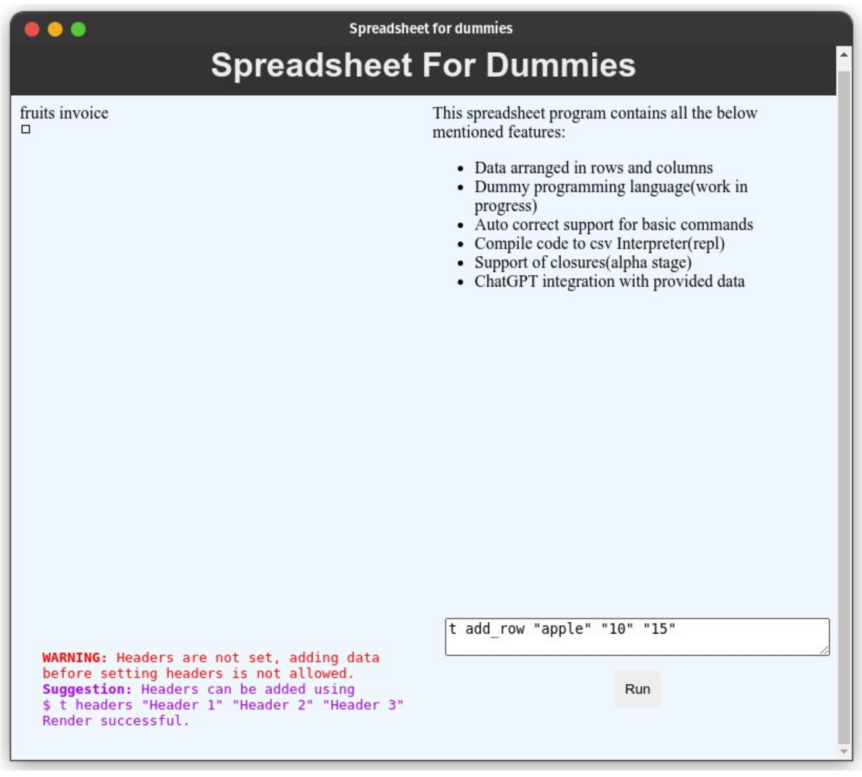
```
t create with title "fruits invoice"
```

```
Render successful.
```

Voila! **Our table is now shown in the table section.** Now lets add some data into it:

```
t add_row "apple" "10" "15"
```

Oh no, we forgot to follow a rule prescribed by the syntax. But the best part is, interpreter always yeilds some meaningful error message!



fruits invoice

This spreadsheet program contains all the below mentioned features:

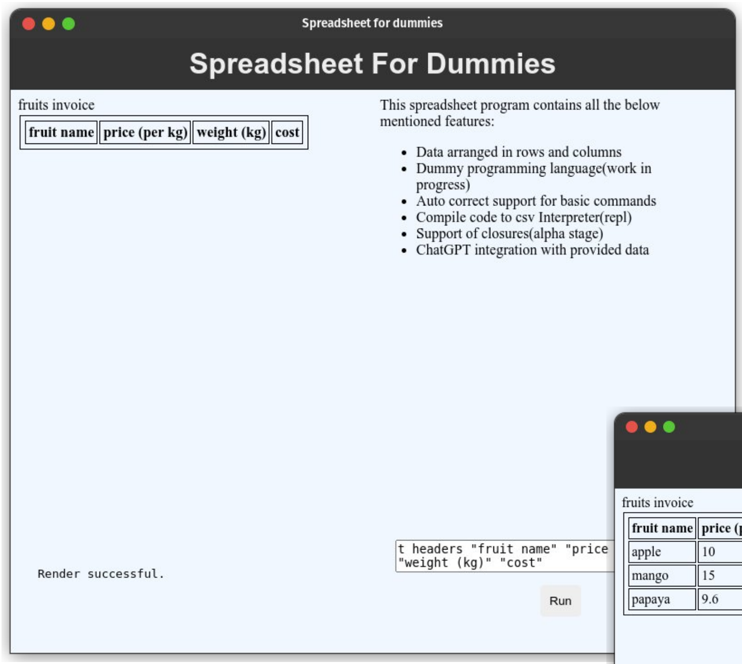
- Data arranged in rows and columns
- Dummy programming language(work in progress)
- Auto correct support for basic commands
- Compile code to csv Interpreter(repl)
- Support of closures(alpha stage)
- ChatGPT integration with provided data

```
t add_row "apple" "10" "15"
```

```
WARNING: Headers are not set, adding data
before setting headers is not allowed.
Suggestion: Headers can be added using
$ t headers "Header 1" "Header 2" "Header 3"
Render successful.
```

So yeah, lets add the headers first using:

```
t headers "fruit name" "price (per kg)" "weight (kg)" "cost"
```



fruits invoice

fruit name	price (per kg)	weight (kg)	cost

This spreadsheet program contains all the below mentioned features:

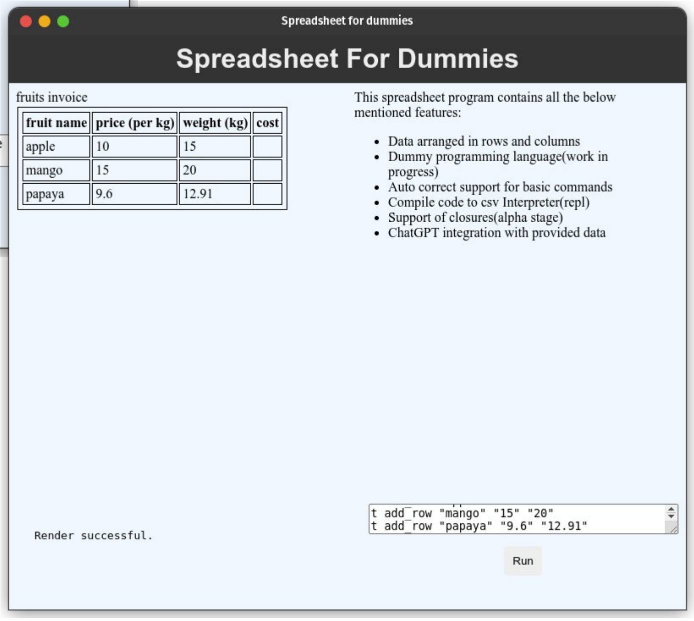
- Data arranged in rows and columns
- Dummy programming language(work in progress)
- Auto correct support for basic commands
- Compile code to csv Interpreter(repl)
- Support of closures(alpha stage)
- ChatGPT integration with provided data

```
t headers "fruit name" "price
"weight (kg)" "cost"
```

```
Render successful.
```

There we go, we can see our table forming with help of our code. Now lets try to add data again!

```
t add_row "apple" "10" "15"

t add_row "mango" "15" "20"

t add_row "papaya" "9.6"
"12.91"
```



fruits invoice

fruit name	price (per kg)	weight (kg)	cost
apple	10	15	
mango	15	20	
papaya	9.6	12.91	

This spreadsheet program contains all the below mentioned features:

- Data arranged in rows and columns
- Dummy programming language(work in progress)
- Auto correct support for basic commands
- Compile code to csv Interpreter(repl)
- Support of closures(alpha stage)
- ChatGPT integration with provided data

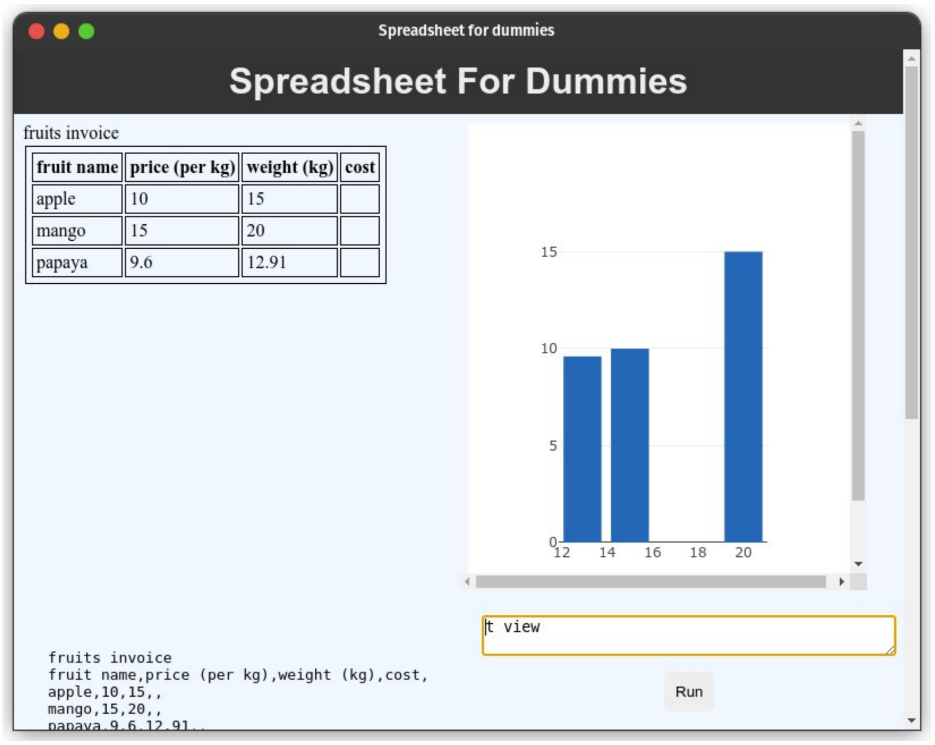
```
t add_row "mango" "15" "20"
t add_row "papaya" "9.6" "12.91"
```

```
Render successful.
```

Ah! That looks way better, doesn't it?

Now lets try to build the graph

```
t view
```



Spreadsheet For Dummies

fruits invoice

fruit name	price (per kg)	weight (kg)	cost
apple	10	15	
mango	15	20	
papaya	9.6	12.91	

```
t view
```

```
fruits invoice
fruit name,price (per kg),weight (kg),cost,
apple,10,15,,
mango,15,20,,
papaya,9.6,12.91,,
```

As per GPT-3 suggestion, a bar graph was plotted against price vs weight for each fruit!

This may not be the most perfect curve and lacks a lot of information, we expect improvements regarding this in the Future versions!

Okay, now lets add some more data, also try to fill the “Cost” section in our table.

```
t add_row "total"
```

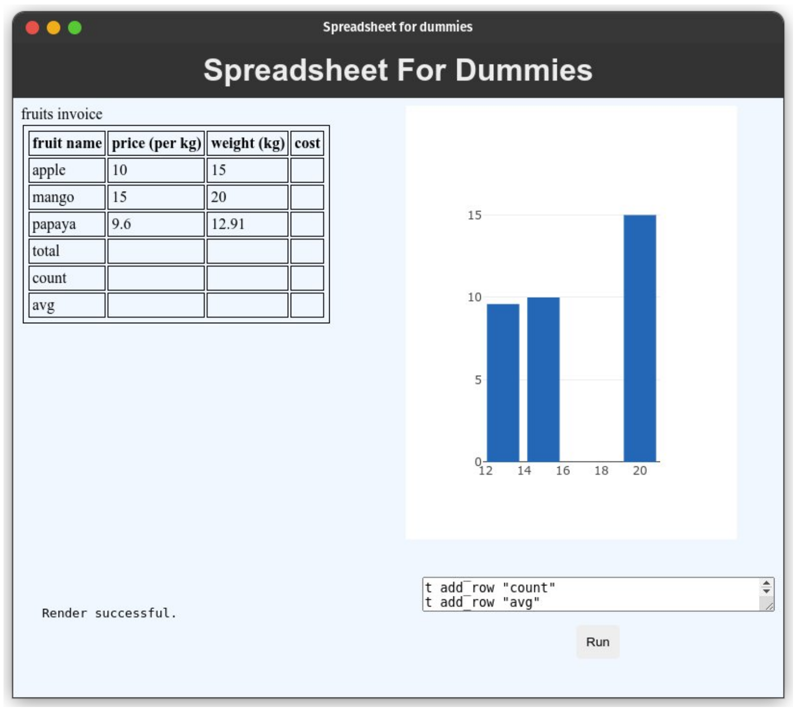
```
t add_row "count"
```

```
t add_row "avg"
```

Hmm, some more empty rows.

We know that for the column “cost”, we have to multiply price per kg with the weight. Basic maths? Lets do that! But wait, how do we do that?

Answer is **Closure**. Closures are a highly extensible piece which can be understood like a function which borrows the table as mutable and making required changes as we defined it.



Spreadsheet For Dummies

fruits invoice

fruit name	price (per kg)	weight (kg)	cost
apple	10	15	
mango	15	20	
papaya	9.6	12.91	
total			
count			
avg			

```
Render successful.
```

```
t add_row "count"
t add_row "avg"
```

Lets define a multiplication closure.

```
mul define ~cell.x 1 -,~cell.y 2 +!reg!1;each!reg!0!~op ~cell *;
```

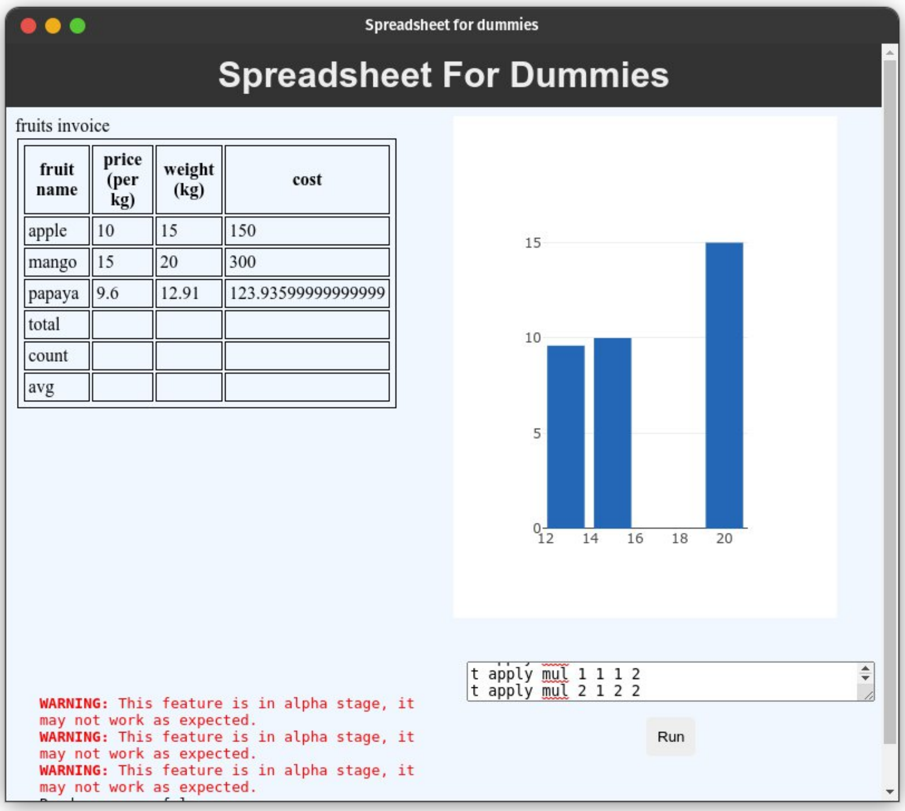
Well, at this point we need to understand how to define closures. Lets not dive very deep into it as it is still unstable and API may change, but lets get a briefing about the same.

This closure is basically named “mul” and it prints its output 2 columns ahead of where it was called, using an intermediate register (register 0) for calculations. The calculation is done as:

```
output = current_cell * register_0
```

If you understood that, then hats off! Otherwise no worries, afterall it takes time to learn a language especially when the language is an eso lang. So we defined our closures, lets use that!

```
t apply mul 0 1 0 2

t apply mul 1 1 1 2

t apply mul 2 1 2 2
```



fruits invoice

fruit name	price (per kg)	weight (kg)	cost
apple	10	15	150
mango	15	20	300
papaya	9.6	12.91	123.93599999999999
total			
count			
avg			

```
t apply mul 1 1 1 2
t apply mul 2 1 2 2
```

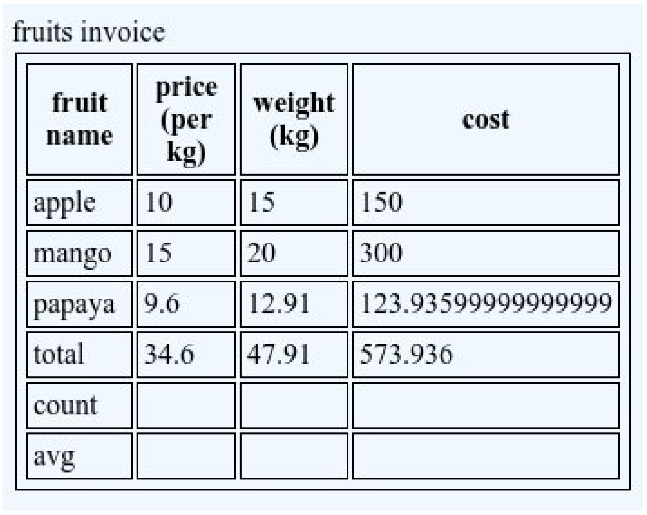
```
WARNING: This feature is in alpha stage, it
may not work as expected.
WARNING: This feature is in alpha stage, it
may not work as expected.
WARNING: This feature is in alpha stage, it
may not work as expected.
```

Mission successful ?

Yeah, look our colums are now filled, what we did is apply the multiplication closure on row 0 col 1 to row 0 col 2 and similary on row 1 col 1 to row 1 col 2 and row 2 col 1 to row 2 col 2.

Okay seems like we crossed one hurdle, now lets try to fill up rest of the table, lets define another closure “sum” and apply it on our columns to sum up all the values.

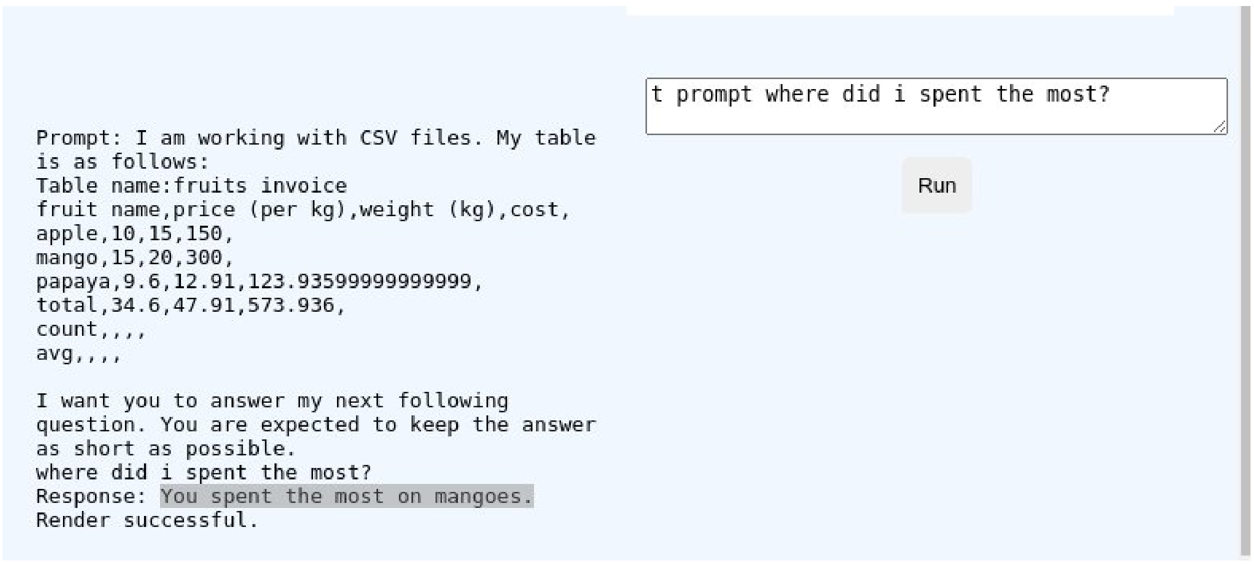
```
sum define auto!reg!1;each!reg!0!~op ~cell +;

t apply sum 0 1 2 1

t apply sum 0 2 2 2

t apply sum 0 3 2 3
```



fruits invoice

fruit name	price (per kg)	weight (kg)	cost
apple	10	15	150
mango	15	20	300
papaya	9.6	12.91	123.93599999999999
total	34.6	47.91	573.936
count			
avg			

Voila! We now certainly have much more data in our table. Oh wait didn’t the project mention we can use GPT-3 to query about the data? Lets try that out!



```
t prompt where did i spent the most?
```

Run

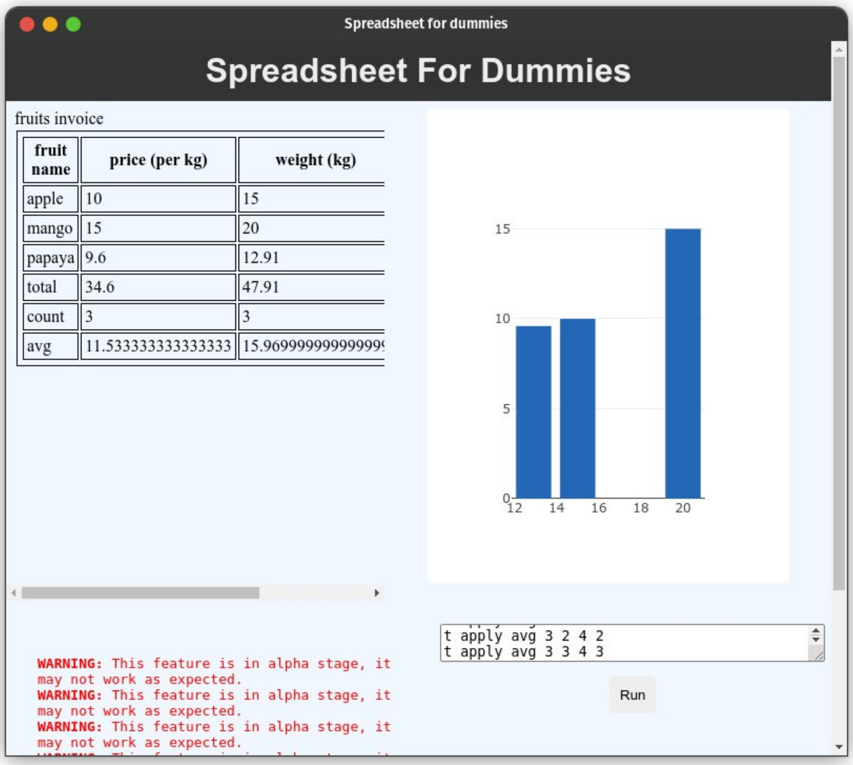
```
Prompt: I am working with CSV files. My table
is as follows:
Table name:fruits invoice
fruit name,price (per kg),weight (kg),cost,
apple,10,15,150,
mango,15,20,300,
papaya,9.6,12.91,123.93599999999999,
total,34.6,47.91,573.936,
count,,,,
avg,,,,

I want you to answer my next following
question. You are expected to keep the answer
as short as possible.
where did i spent the most?
Response: You spent the most on mangoes.
Render successful.
```

`t prompt where did i spent the most?`

So, Mangoes it is. Indeed my favorite ones! Obviously debug prints are enable to also let you transparently view how data is being processed. Yes, the data can always we converted to CSV, like it did in here.

Okay, enough of empty table, lets defined closures to fill up the rest of the table and apply it.

```
count define ~cell.x 1 +,~cell.y!reg!1;each!reg!0!~op 1 +;

avg define auto!reg!1;each!reg!0!~op ~cell /;

t apply count 0 1 2 1

t apply count 0 2 2 2

t apply count 0 3 2 3

t apply avg 3 1 4 1

t apply avg 3 2 4 2

t apply avg 3 3 4 3
```



fruits invoice

fruit name	price (per kg)	weight (kg)
apple	10	15
mango	15	20
papaya	9.6	12.91
total	34.6	47.91
count	3	3
avg	11.533333333333333	15.96999999999999

```
t apply avg 3 2 4 2
t apply avg 3 3 4 3
```

```
WARNING: This feature is in alpha stage, it
may not work as expected.
WARNING: This feature is in alpha stage, it
may not work as expected.
WARNING: This feature is in alpha stage, it
may not work as expected.
```

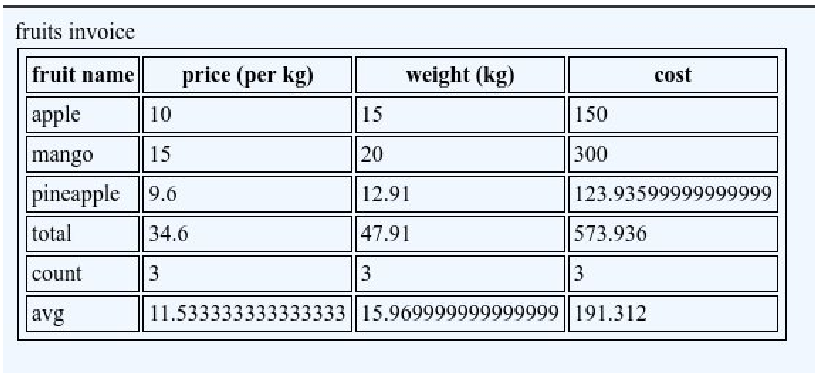
So that was the basic idea about the project. But wait, I am not really fond of "papaya", can't we just change it to something like "pineapple" ?

The answer is both yes and no.

The row ones created are immutable by default, which means you can't change the data once it is entered, but closures can! Remember the part when I said closures makes spread-lang highly extensible? They can not only do awesome operations but also work on our data mutably and hence modify the table! But why this complicated system? To teach the concept of "mutable" and "immutable". Go through are Agenda.

Lets modify our data using "modify" closure.

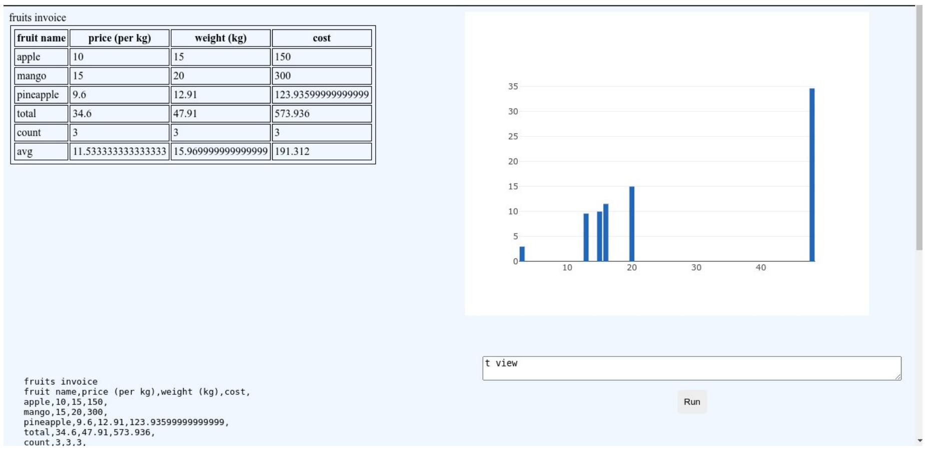
```
modify define ~cell.x 1 -,~cell.y!reg!1;raw!reg!0!pineapple;

t apply modify 2 0
```



fruits invoice

fruit name	price (per kg)	weight (kg)	cost
apple	10	15	150
mango	15	20	300
pineapple	9.6	12.91	123.93599999999999
total	34.6	47.91	573.936
count	3	3	3
avg	11.533333333333333	15.969999999999999	191.312

Convinient (or maybe complex, but again, it is an Eso lang)



fruits invoice

fruit name	price (per kg)	weight (kg)	cost
apple	10	15	150
mango	15	20	300
pineapple	9.6	12.91	123.93599999999999
total	34.6	47.91	573.936
count	3	3	3
avg	11.533333333333333	15.969999999999999	191.312

```
t view
```

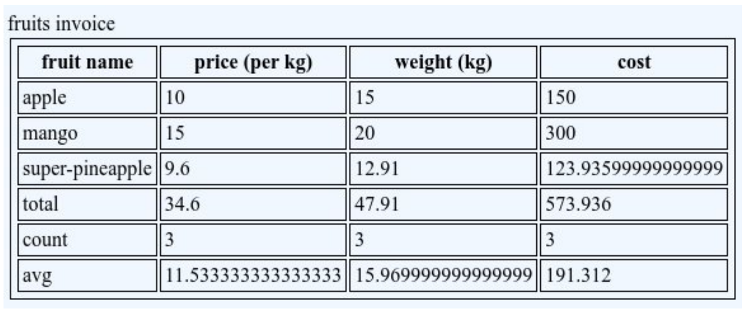
```
fruits invoice
fruit name,price (per kg),weight (kg),cost,
apple,10,15,150,
mango,15,20,300,
pineapple,9.6,12.91,123.93599999999999,
total,34.6,47.91,573.936,
count,3,3,3,
```

Trying to use plot feature again, we end up with this plot. Sadly it is not really an informative plot. That is our limitation :(

So this is my submission for Quine-Quest 1. Its not a finished project, but it was definitely fun to work on it.

Over and Out!

```
prepend define ~cell.x 1
-,~cell.y!reg!1;each!reg!0!supe
r- ~cell +;

t apply prepend 2 0
```



fruits invoice

fruit name	price (per kg)	weight (kg)	cost
apple	10	15	150
mango	15	20	300
super-pineapple	9.6	12.91	123.93599999999999
total	34.6	47.91	573.936
count	3	3	3
avg	11.533333333333333	15.969999999999999	191.312

super-pineapple

Techstack: Rust, Electron, Ploty, GPT-3