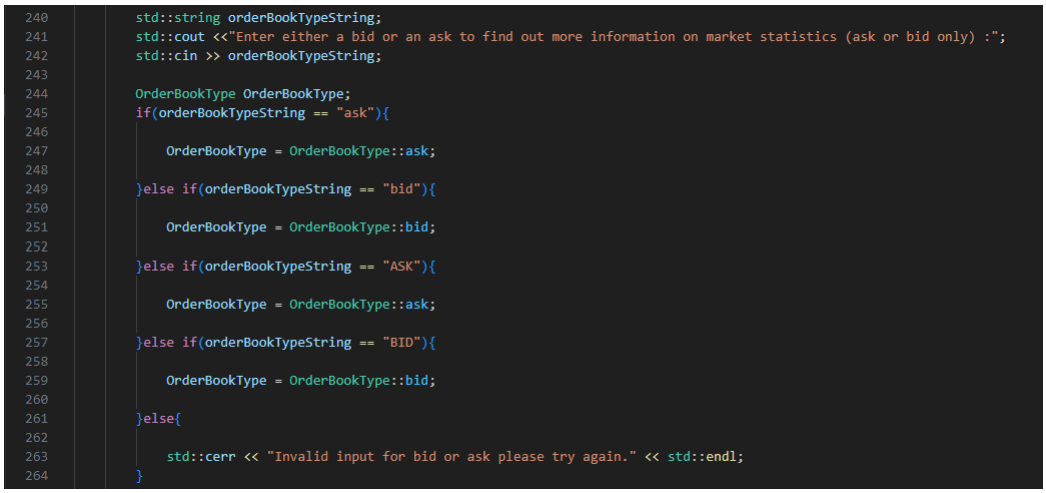
Part 1: Compute Candlestick Data

Firstly, in part 1. There are many products in the dataset. The user can select what kind of product and trade that is desired to view. A validation system is put in place, to check if the user has entered the correct input.

Figure 1:

```
210     //retrieve the user input to allow the user to have to option to choose what kind of product the user wants to view
211     std::string tradingPair;
212     std::cout << "Exchanges which are available:\nETH/BTC, DOGE/BTC, BTC/USDT, ETH/USDT, DOGE/USDT \nEnter the trading pair in the format (ETH/BTC) :";
213     std::cin >> tradingPair;
214
215     if(tradingPair == "ETH/BTC"){
216
217         tradingPair == "ETH/BTC";
218
219     }else if(tradingPair == "DOGE/BTC"){
220
221         tradingPair == "DOGE/BTC";
222
223     }else if(tradingPair == "BTC/USDT"){
224
225         tradingPair == "BTC/USDT";
226
227     }else if(tradingPair == "ETH/USDT"){
228
229         tradingPair == "ETH/USDT";
230
231     }else if(tradingPair == "DOGE/USDT"){
232
233         tradingPair == "DOGE/USDT";
234
235     }else{
236
237         std::cerr << "Invalid product input please try again." << std::endl;
238     }
```

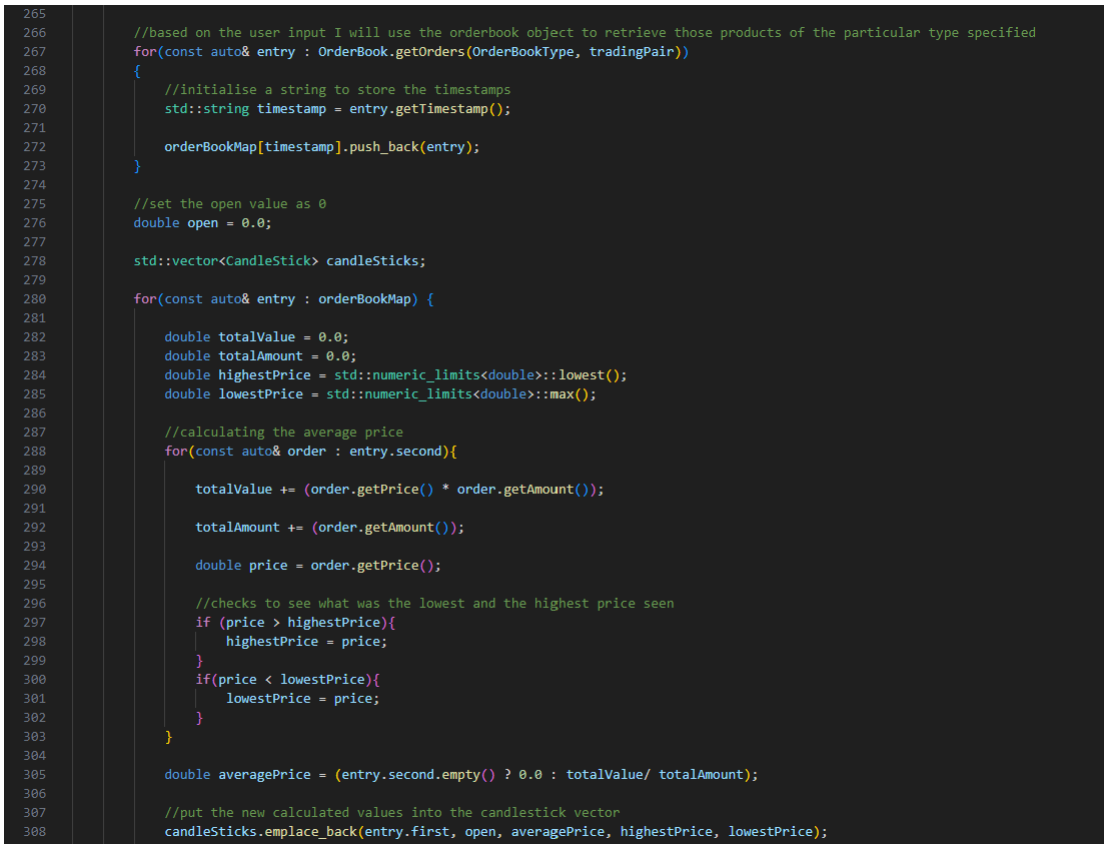
Figure 2:



```
240     std::string orderBookTypeString;
241     std::cout <<"Enter either a bid or an ask to find out more information on market statistics (ask or bid only) :";
242     std::cin >> orderBookTypeString;
243
244     OrderBookType OrderBookType;
245     if(orderBookTypeString == "ask"){
246
247         OrderBookType = OrderBookType::ask;
248
249     }else if(orderBookTypeString == "bid"){
250
251         OrderBookType = OrderBookType::bid;
252
253     }else if(orderBookTypeString == "ASK"){
254
255         OrderBookType = OrderBookType::ask;
256
257     }else if(orderBookTypeString == "BID"){
258
259         OrderBookType = OrderBookType::bid;
260
261     }else{
262
263         std::cerr << "Invalid input for bid or ask please try again." << std::endl;
264     }
```

Figure 1 and 2, user is asked to enter the product or type. Suggestions are provided to the user. However, in line 237. If an invalid product selected. An error is returned. But if the user has entered a valid product. For example, ETH/BTC then, ETH/BTC is returned and passed into the tradingPair variable.

Figure 3:



```
265
266         //based on the user input I will use the orderbook object to retrieve those products of the particular type specified
267         for(const auto& entry : OrderBook.getOrders(OrderBookType, tradingPair))
268         {
269             //initialise a string to store the timestamps
270             std::string timestamp = entry.getTimestamp();
271
272             orderBookMap[timestamp].push_back(entry);
273         }
274
275         //set the open value as 0
276         double open = 0.0;
277
278         std::vector<CandleStick> candleSticks;
279
280         for(const auto& entry : orderBookMap) {
281
282             double totalValue = 0.0;
283             double totalAmount = 0.0;
284             double highestPrice = std::numeric_limits<double>::lowest();
285             double lowestPrice = std::numeric_limits<double>::max();
286
287             //calculating the average price
288             for(const auto& order : entry.second){
289
290                 totalValue += (order.getPrice() * order.getAmount());
291
292                 totalAmount += (order.getAmount());
293
294                 double price = order.getPrice();
295
296                 //checks to see what was the lowest and the highest price seen
297                 if (price > highestPrice){
298                     highestPrice = price;
299                 }
300                 if(price < lowestPrice){
301                     lowestPrice = price;
302                 }
303             }
304
305             double averagePrice = (entry.second.empty() ? 0.0 : totalValue/ totalAmount);
306
307             //put the new calculated values into the candlestick vector
308             candleSticks.emplace_back(entry.first, open, averagePrice, highestPrice, lowestPrice);
```

Assuming successfully entered the values. Data will be passed into getOrders in line 267. Line 272, at every timestamp it is stored into the map. Line 290, total value and amount is calculated for average price. Line 297, prices are compared to find lowest and highest price. Line 308, calculated values are passed into the candlestick vector.

Figure 4:

```
307             //put the new calculated values into the candlestick vector
308             candleSticks.emplace_back(entry.first, open, averagePrice, highestPrice, lowestPrice);
309
310             open = averagePrice;
311
312             std::cout << "Timestamp: " << entry.first
313                       << ", Open: " << candleSticks.back().getOpen()
314                       << ", Close: " << candleSticks.back().getClose()
315                       << ", High: " << candleSticks.back().getHigh()
316                       << ", Low: " << candleSticks.back().getLow() << std::endl;
317         }
318         std::cout << "After processing" << std::endl;
319     } catch (const std::exception& e){
320         std::cerr << "Exception caught: " << e.what() << std::endl;
321     }catch(...) {
322         std::cerr << "Unknown exception caught" << std::endl;
323     }
324 }
325
326
```

Assuming values are being successfully passed into the candlestick vector. The data from the candlestick vector are printed.

Output:

```
CSVReader::readCSV read 1021772 entries
1: Print help
2: Print exchange stats
3: Make an offer
4: Make a bid
5: Print wallet
6: Continue
7: Print Market statistics for Candlesticks
8: Print Candlestick
9: Print Additional Candlestick
=============
Current time is: 2020/06/01 11:57:30.328127
Type in 1-9
7
You chose: 7
Before processing
Exchanges which are available:
ETH/BTC, DOGE/BTC, BTC/USDT, ETH/USDT, DOGE/USDT
Enter the trading pair in the format (ETH/BTC) :ETH/BTC
Enter either a bid or an ask to find out more information on market statistics (ask or bid only) :ask
```

```
Timestamp: 2020/06/01 14:55:35.126455, Open: 0.0250855, Close: 0.0250831, High: 0.02523, Low: 0.0249717
Timestamp: 2020/06/01 14:55:40.132374, Open: 0.0250831, Close: 0.0250882, High: 0.0252526, Low: 0.0249717
Timestamp: 2020/06/01 14:55:45.138960, Open: 0.0250882, Close: 0.025092, High: 0.02525, Low: 0.0249911
Timestamp: 2020/06/01 14:55:50.146194, Open: 0.025092, Close: 0.0250911, High: 0.02525, Low: 0.0249911
Timestamp: 2020/06/01 14:55:55.153586, Open: 0.0250911, Close: 0.0250921, High: 0.0252385, Low: 0.0249911
Timestamp: 2020/06/01 14:56:00.161324, Open: 0.0250921, Close: 0.0250908, High: 0.0252304, Low: 0.0249911
Timestamp: 2020/06/01 14:56:05.168496, Open: 0.0250908, Close: 0.0250912, High: 0.0252385, Low: 0.0249911
Timestamp: 2020/06/01 14:56:10.175357, Open: 0.0250912, Close: 0.0250863, High: 0.02523, Low: 0.0249899
Timestamp: 2020/06/01 14:56:15.181646, Open: 0.0250863, Close: 0.0250851, High: 0.0252304, Low: 0.0249899
Timestamp: 2020/06/01 14:56:20.188780, Open: 0.0250851, Close: 0.0250732, High: 0.0252274, Low: 0.0249825
Timestamp: 2020/06/01 14:56:25.195839, Open: 0.0250732, Close: 0.0250806, High: 0.0252385, Low: 0.0249745
Timestamp: 2020/06/01 14:56:30.202861, Open: 0.0250806, Close: 0.0250885, High: 0.0252526, Low: 0.0249886
Timestamp: 2020/06/01 14:56:35.210165, Open: 0.0250885, Close: 0.0250951, High: 0.0252684, Low: 0.0249886
After processing
```

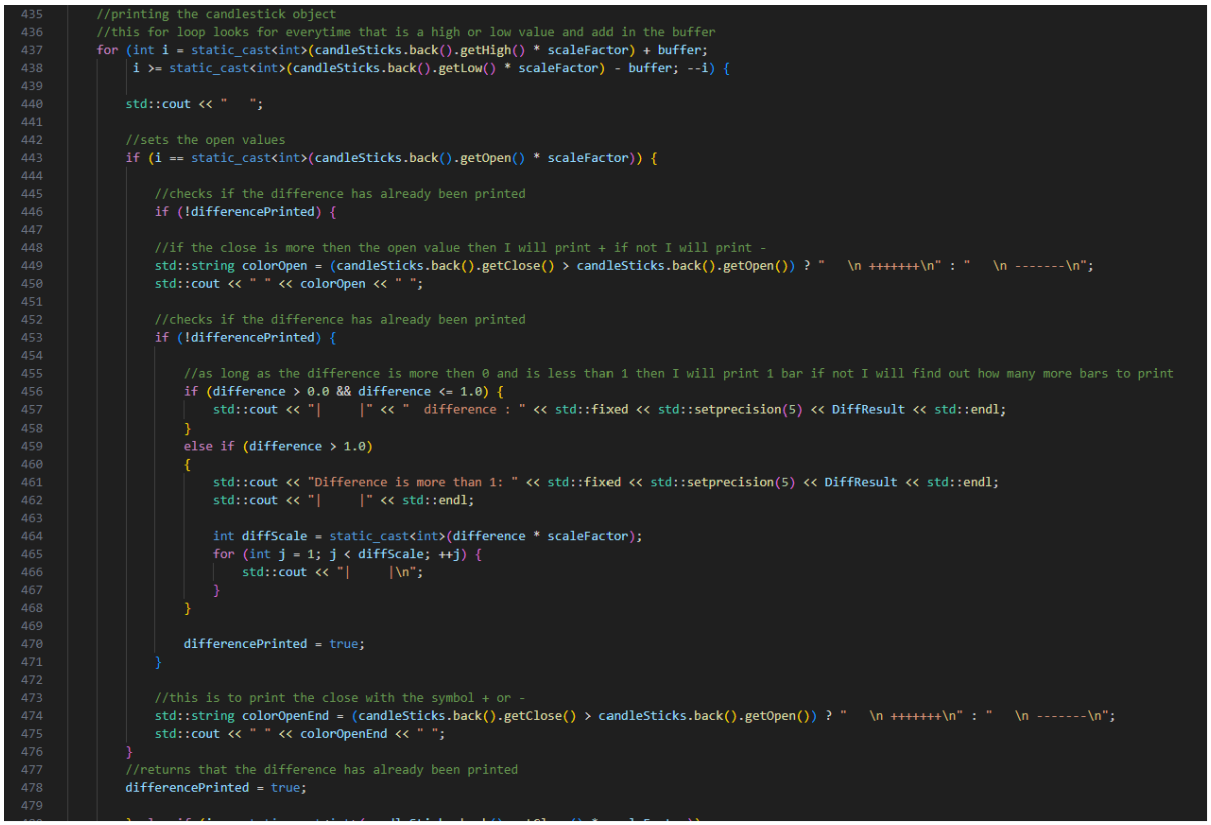
Part 2: Create a text-based plot of the Candlestick Data

Figure 5:

```
426         //the buffer is created to make spacing between each candlesticks objects being created onto the terminal
427         const int buffer = 10;
428
429         //the difference is being calculated by subtracting the open value and the close value of each candlesticks
430         double difference = std::abs(candleSticks.back().getOpen() - candleSticks.back().getClose());
431
432         //since the differences are so small I want to be able to see the difference between the open and close so I round off the to nearest 5 decimal places
433         double DiffResult = std::ceil(difference * 100000) / 100000;
```

Figure 5, a buffer was created to allow spacing between each candlestick. Line 430, the difference is calculated between the open and close. Line 433, difference calculated is rounded off to the nearest 5 decimal places. As values are too small. For readability and accuracy, difference is rounded off.

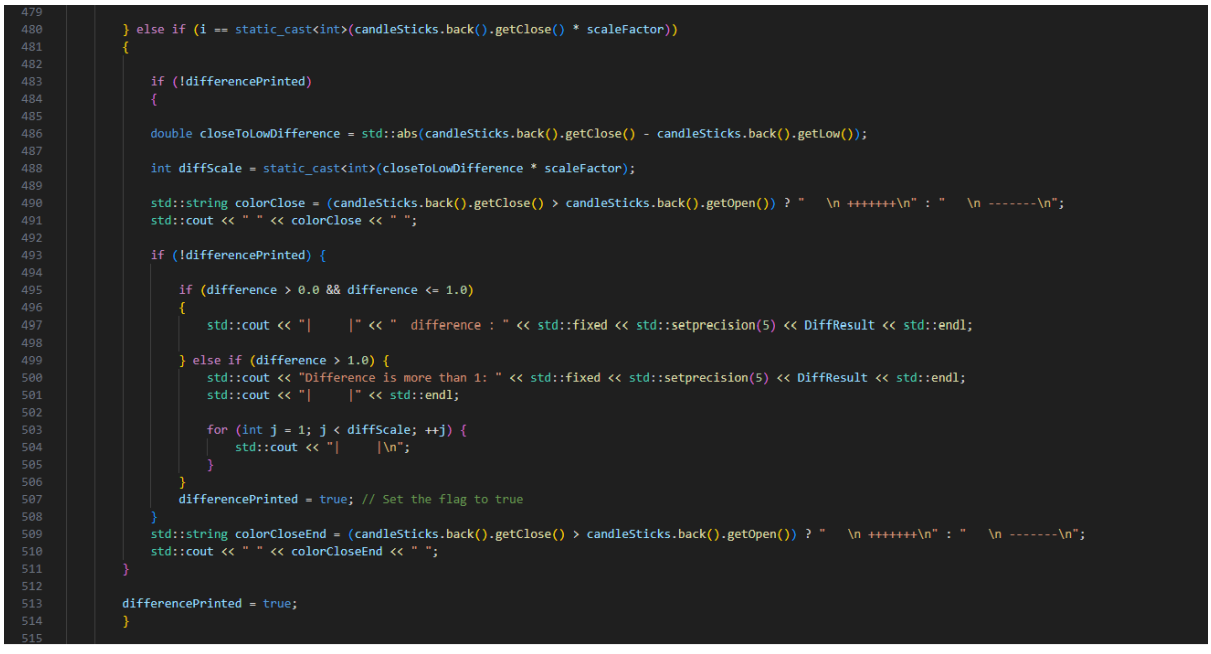
Figure 6:



```
435     //printing the candlestick object
436     //this for loop looks for everytime that is a high or low value and add in the buffer
437     for (int i = static_cast<int>(candleSticks.back().getHigh() * scaleFactor) + buffer;
438          i >= static_cast<int>(candleSticks.back().getLow() * scaleFactor) - buffer; --i) {
439
440         std::cout << "   ";
441
442         //sets the open values
443         if (i == static_cast<int>(candleSticks.back().getOpen() * scaleFactor)) {
444
445             //checks if the difference has already been printed
446             if (!differencePrinted) {
447
448             //if the close is more then the open value then I will print + if not I will print -
449             std::string colorOpen = (candleSticks.back().getClose() > candleSticks.back().getOpen()) ? "   \n +++++++\n" : "   \n -------\n";
450             std::cout << " " << colorOpen << " ";
451
452             //checks if the difference has already been printed
453             if (!differencePrinted) {
454
455                 //as long as the difference is more then 0 and is less than 1 then I will print 1 bar if not I will find out how many more bars to print
456                 if (difference > 0.0 && difference <= 1.0) {
457                     std::cout << "|     |" << "  difference : " << std::fixed << std::setprecision(5) << DiffResult << std::endl;
458                 }
459                 else if (difference > 1.0)
460                 {
461                     std::cout << "Difference is more than 1: " << std::fixed << std::setprecision(5) << DiffResult << std::endl;
462                     std::cout << "|     |" << std::endl;
463
464                     int diffScale = static_cast<int>(difference * scaleFactor);
465                     for (int j = 1; j < diffScale; ++j) {
466                         std::cout << "|     |\n";
467                     }
468                 }
469
470                 differencePrinted = true;
471             }
472
473             //this is to print the close with the symbol + or -
474             std::string colorOpenEnd = (candleSticks.back().getClose() > candleSticks.back().getOpen()) ? "   \n +++++++\n" : "   \n -------\n";
475             std::cout << " " << colorOpenEnd << " ";
476         }
477         //returns that the difference has already been printed
478         differencePrinted = true;
479
```

Figure 6, line 449. Checks if the close value is larger than the open value. Depending on the outcome it will either print a + or -. Line 453, if the difference is not yet printed then it will then check how much is the difference that is being printed.

Figure 7:



```
479
480         } else if (i == static_cast<int>(candleSticks.back().getClose() * scaleFactor))
481         {
482
483             if (!differencePrinted)
484             {
485
486             double closeToLowDifference = std::abs(candleSticks.back().getClose() - candleSticks.back().getLow());
487
488             int diffScale = static_cast<int>(closeToLowDifference * scaleFactor);
489
490             std::string colorClose = (candleSticks.back().getClose() > candleSticks.back().getOpen()) ? "   \n +++++++\n" : "   \n -------\n";
491             std::cout << " " << colorClose << " ";
492
493             if (!differencePrinted) {
494
495                 if (difference > 0.0 && difference <= 1.0)
496                 {
497                     std::cout << "|     |" << "  difference : " << std::fixed << std::setprecision(5) << DiffResult << std::endl;
498
499                 } else if (difference > 1.0) {
500                     std::cout << "Difference is more than 1: " << std::fixed << std::setprecision(5) << DiffResult << std::endl;
501                     std::cout << "|     |" << std::endl;
502
503                     for (int j = 1; j < diffScale; ++j) {
504                         std::cout << "|     |\n";
505                     }
506                 }
507                 differencePrinted = true; // Set the flag to true
508             }
509             std::string colorCloseEnd = (candleSticks.back().getClose() > candleSticks.back().getOpen()) ? "   \n +++++++\n" : "   \n -------\n";
510             std::cout << " " << colorCloseEnd << " ";
511         }
512
513         differencePrinted = true;
514         }
515
```

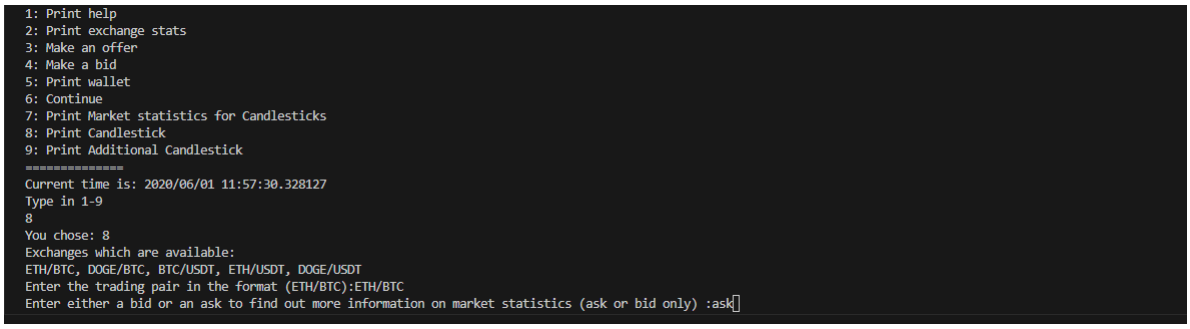
Figure 7, it has a similar concept to figure 6, but this is for the close value. Line 493, it is checked whether the difference has already been printed.

Figure 8:

```
516         else if (i == static_cast<int>(candleSticks.back().getClose() * scaleFactor))
517         {
518             std::string colorClose = (candleSticks.back().getClose() > candleSticks.back().getOpen()) ? "   \n +++++++\n" : "   \n -------\n";
519             std::cout << " " << colorClose << " ";
520 
521             if (!differencePrinted)
522             {
523                 double closeToLowDifference = std::abs(candleSticks.back().getClose() - candleSticks.back().getLow());
524 
525                 double closedDiffResult = std::ceil(difference * 100000) / 100000;
526                 std::cout << "Difference between Close and Low: " << std::fixed << std::setprecision(5) << closedDiffResult << std::endl;
527                 std::cout << "   |   " << std::endl;
528 
529                 int diffScale = static_cast<int>(closeToLowDifference * scaleFactor);
530                 for (int j = 1; j < diffScale; ++j)
531                 {
532                     std::cout << "   |   \n";
533                 }
534 
535                 differencePrinted = true;
536             }
537         } else {
538             std::cout << "*";
539         }
540         std::cout << '\n';
541     }
542 
543     std::cout << "==============\n";
544 
545     linesPrinted++;
546 
547     //prints only the first 10 lines of data
548     if (linesPrinted >= 10)
549     {
550         candleSticks.clear();
551         linesPrinted = 0;
552         break;
553     }
554   }
555 }
556 
557
```

Figure 8, difference between the close and low is calculated. Printed the difference as a symbol, |. If the difference between close and low is more than 1 then it will print that many | symbols as the difference between the close and low.

Output:



```
1: Print help
2: Print exchange stats
3: Make an offer
4: Make a bid
5: Print wallet
6: Continue
7: Print Market statistics for Candlesticks
8: Print Candlestick
9: Print Additional Candlestick
==============
Current time is: 2020/06/01 11:57:30.328127
Type in 1-9
8
You chose: 8
Exchanges which are available:
ETH/BTC, DOGE/BTC, BTC/USDT, ETH/USDT, DOGE/USDT
Enter the trading pair in the format (ETH/BTC):ETH/BTC
Enter either a bid or an ask to find out more information on market statistics (ask or bid only) :ask
```

```
   |
   |
   |
   |
=============
Timestamp: 2020/06/01 11:58:15.372284
Open: 0.12469
Close: 0.12469
High: 0.12590
Low: 0.12419
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |

 -------
 |     |  difference : 0.00001

 -------

   |
   |
```

Part 3: Plot a text graph of some other trading data

Figure 9:

```
559  void MerkelMain::readCandlestickData()
560  {
561      //opening the file
562      std::ifstream file("coin_Aave.csv");
563
564      //if the file is not able to open
565      if (!file.is_open())
566      {
567          std::cerr << "Error opening file." << std::endl;
568          return;
569      }
570
571      std::map<std::string, std::vector<OrderBookEntry>> orderBookMap;
572
573      std::string line;
574
575      while (std::getline(file, line))
576      {
577          std::istringstream iss(line);
578
579          std::string SNo_String,
580              Name,
581              Symbol,
582              Date,
583              HighString,
584              LowString,
585              OpenString,
586              CloseString,
587              VolumeString,
588              MarketCapString;
589
590          std::getline(iss, SNo_String, ',');
591          std::getline(iss, Name, ',');
592          std::getline(iss, Symbol, ',');
593          std::getline(iss, Date, ',');
594          std::getline(iss, HighString, ',');
595          std::getline(iss, LowString, ',');
596          std::getline(iss, OpenString, ',');
597          std::getline(iss, CloseString, ',');
598          std::getline(iss, VolumeString, ',');
599          std::getline(iss, MarketCapString, ',');
```

Figure 9, the file is read and split into its respecting variables to be stored. Line 579, variables are declared.

Figure 10:

```
600
601         try {
602             OrderBookEntry entry(std::stoi(SNo_String),
603                 Name,
604                 Symbol,
605                 Date,
606                 std::stod(HighString),
607                 std::stod(LowString),
608                 std::stod(OpenString),
609                 std::stod(CloseString),
610                 std::stod(VolumeString),
611                 std::stod(MarketCapString));
612
613             std::string timestamp = entry.getDate();
614             orderBookMap[timestamp].push_back(entry);
615         }
616         catch (const std::exception& e)
617         {
618             std::cerr << "Error processing line: " << e.what() << std::endl;
619         }
620     }
621
622     //close the file after reading it
623     file.close();
624
```

Figure 10, stored values are then gathered and stored into a map. Line 614, data is called to be stored into the map.

Figure 11:

```
624
625     double open = 0.0;
626     std::vector<CandleStick> candleSticks;
627
628     //this scaledown is to adjust the values that are being printed out onto the console
629     int scaledown = 1;
630
631     for (const auto& entry : orderBookMap)
632     {
633         double open = entry.second.front().getOpen();
634         double close = entry.second.front().getClose();
635         double high = entry.second.front().getHigh();
636         double low = entry.second.front().getLow();
637
638         candleSticks.emplace_back(entry.first, open, close, high, low);
639
640         std::cout << "Timestamp: " << entry.first
641                   << "\nOpen: " << candleSticks.back().getOpen() / scaledown
642                   << "\nClose: " << candleSticks.back().getClose() / scaledown
643                   << "\nHigh: " << candleSticks.back().getHigh() / scaledown
644                   << "\nLow: " << candleSticks.back().getLow() / scaledown << std::endl;
645
```

Figure 11, line 638. Data from the map is stored into the vector. By iterating through the map and only retrieving the relevant values and storing into the candlestick vector.

Figure 12:

```
628     //this scaledown is to adjust the values that are being printed out onto the console
629     int scaledown = 1;
630
631     for (const auto& entry : orderBookMap)
632     {
633         double open = entry.second.front().getOpen();
634         double close = entry.second.front().getClose();
635         double high = entry.second.front().getHigh();
636         double low = entry.second.front().getLow();
637
638         candleSticks.emplace_back(entry.first, open, close, high, low);
639
640         std::cout << "Timestamp: " << entry.first
641                   << "\nOpen: " << candleSticks.back().getOpen() / scaledown
642                   << "\nClose: " << candleSticks.back().getClose() / scaledown
643                   << "\nHigh: " << candleSticks.back().getHigh() / scaledown
644                   << "\nLow: " << candleSticks.back().getLow() / scaledown << std::endl;
645
646         //slack is to add some distance between each candlestick that is being printed
647         int slack = 10;
648
649         //everytime the for loop finds that there is a high or a low value it prints it with the slack
650         for (int i = static_cast<int>(candleSticks.back().getHigh() + slack);
651             i >= static_cast<int>(candleSticks.back().getLow() - slack); --i)
652         {
653         std::cout << "   ";
654
655         //if close is more than open then it will take the distance between the high and the close and prints |
656         if ((candleSticks.back().getClose() / scaledown) > (candleSticks.back().getOpen() / scaledown))
657         {
658             if ((i > static_cast<int>(candleSticks.back().getClose() / scaledown)) &&
659                 (i < static_cast<int>(candleSticks.back().getHigh() / scaledown)))
660             {
661                 std::cout << "   |   ";
662             }
663         }
664
665         //if the close is less than the open then the distance between the open and the high will then print |
666         if ((candleSticks.back().getClose() / scaledown) < (candleSticks.back().getOpen() / scaledown))
667         {
668             if ((i > static_cast<int>(candleSticks.back().getOpen() / scaledown)) &&
669                 (i < static_cast<int>(candleSticks.back().getHigh() / scaledown)))
670             {
671                 std::cout << "   |   ";
672             }
673         }
```

Figure 12, data from the candlestick vector is retrieved using get methods to return relevant data for comparison. Comparisons are done to see what kind of symbols to print out.

Output:

```
1: Print help
2: Print exchange stats
3: Make an offer
4: Make a bid
5: Print wallet
6: Continue
7: Print Market statistics for Candlesticks
8: Print Candlestick
9: Print Additional Candlestick
==============
Current time is: 2020/06/01 11:57:30.328127
Type in 1-9
9
```

```
=============
Timestamp: 2021-07-05 23:59:59
Open: 277.11053
Close: 307.82908
High: 317.38723
Low: 263.43388
```