Intro

LendingClub is a platform for peer-to-peer lending. If you're not familiar with it, understand the following:

- **Peer-to-peer (P2P) lending:** It's crowdfunding peoples' loans and they are expected to pay back with interest.
- **Note:** A piece of a loan. If somebody wanted to borrow $1000, a note could be a $25 chunk of that.
- **Secondary market:** Where investors can buy or sell notes. Investors might sell for profit, or they might sell at a discount because they think there's too much risk in holding a note.
- **Charged off:** When a borrower stops paying his/her loan and there is no chance of recovering any money. This is bad for the investors.

Here's what I did:

- Downloaded existing loan data which LendingClub provides.
- Found patterns in that data to recognize a good or bad loan.
- Created script find notes based on my patterns.
- Created interface to buy those notes easily.

Step 1. Get the dataset

Extract

Every quarter, LendingClub updates a list of all loans they've issued and what the status of those loans are. It's a large dataset, with plenty of *interest*ing fields. There are multiple csv files with rows about each loan such as loan amount, interest rate, purpose, date issued, current status, et al.

The .csv files have 100k+ rows per file:

Name	Date modified	Type	Size
LoanStats3a_securev1.csv	3/1/2016 10:48 PM	OpenOffice.org 1....	33,965 KB
LoanStats3b_securev1.csv	1/25/2016 4:18 PM	OpenOffice.org 1....	125,905 KB
LoanStats3c_securev1.csv	1/25/2016 4:19 PM	OpenOffice.org 1....	142,312 KB
LoanStats3d_securev1.csv	1/25/2016 4:20 PM	OpenOffice.org 1....	250,457 KB

More than enough data for LibreOffice to handle on my computer (it's actually not responding in the screenshot below)

Load

Having worked with the LAMP (Linux, Apache, MySQL, PHP) stack, I recognized that a few hundred thousand rows and many numerical columns would be a good fit for a relational database like MySQL. However at time, MongoDB was quite popular and I wanted to see what the hype was about. This was my opportunity to try it! [Lesson to be learned here.]

After getting MongoDB set up on my computer, I was ready to import the data in the csv files. There is a tool called mongoimport to do this. **Because LendingClub updates the csv files once a quarter, I expected the need to have to redo this step, so I wrote down the steps I took.** When I needed to import data again a few months later, I was so happy I did.

```
import steps:
    Download all CSV's
    Delete first line in each CSV
    Clear out the loans database
    Run command for each csv:
    D:\Mongo\bin\mongoimport.exe /h localhost /p 27017 /d lc /c loans /numInsertionWorkers 2 /type csv /headerline /fi
```

Clean (Transform)

Below is an example of a row from the csv files. Some fields didn't look necessary, and some needed to be converted to a proper type. Using an appropriate data type can save space and can make tasks such as sorting and searching faster.

```
"1069410","1303652","21000","21000","20975"," 60 months","
19.91%","555.33","E","E4","Costco","7 years","RENT","50000","Verified","Dec-2011","Charged
Off","n","https://www.lendingclub.com/browse/loanDetail.action?loan_id=1069410","
Borrower added on 12/20/11 > consolidation  of credit cards.<br>","debt_consolidation","
Bill pay
of","980xx","WA","21.58","0","Sep-1998","680","684","1","","","7","0","19448","97.6%","14"
,"f","0.00","0.00","18319.14","18297.35","8990.81","9328.33","0.0","0.0","0.0","Oct-2014",
"555.33","","Jan-2016","634","630","0","","1","INDIVIDUAL","","","","0","","","","","","",
"","","","","","","","","","",""
```

- URL isn't necessary. With just a loan id, it can be reconstructed.
- Some dates should be converted from string.
- Interest rates should be converted from string.

This can be done directly in MongoDB with Javascript. I put a number of code blocks like below into "post_processing.js" so I can run them all each time I import data. Although convenient, it's slow. It would have been a better to clean first, then load.

```
/*
Fix dates
*/
db.loans.find({
    issue_d: {
        $type: 2
    }
}).forEach( function( elem ) {
    var newDate = elem.issue_d.replace( '-', ' 1 ' );
    elem.issue_d = new Date( newDate );
    db.loans.save( elem );
});
```

Step 2. “Analysis”

My goal here was to find patterns that classify a loan as good or bad. A good loan being one that is fully paid off, and a bad loan being one that is late, in default, or charged off. My strategy was to pick values of a data columns(*features*) and see what the ratio good loans:bad loans was, and compare that to the average ratio of good:bad.

In pseudocode, this is what I did:

```
For loans with purpose “small_business”:
      if the loan is bad:
            numerator++
      total++
Ratio_feature = numerator / total

Reset numerator and total

For all loans:
      If the loan is bad:
            numerator++
      Total++
Ratio_average = numerator / total
```

I then expanded the number of features and their values and calculated ratios for each combination. It’s far from elegant.

```
var home_ownership_steps =
{
    "0" : "RENT",
    "1" : "MORTGAGE",
    "2" : "OWN",
    "3" : "NONE",
    //"4" : "OTHER",
    //"5" : "ANY"
};


for( var emp_length_key in emp_length_steps )
{
    for( var verification_status_key in verification_status_steps )
    {
        for( var purpose_key in purpose_steps )
        {
            for( var home_ownership_key in home_ownership_steps )
            {
                for( var income_min = annual_inc_start; income_min <= annual_inc_end; income_min += annual_inc_step_size )
                {
```

I even tried language processing (if you’d be kind enough to allow me to call it that). In the loan description, did borrowers use lowercase/capital “i” when referring to themselves? Were there any keywords that could be red flags?

```
var descriptionsToTry = [
/.*/i ,
// /.\s+[A-Z]/, // Capital letter after a period...actually worse in a lot of cases
// /improve/i,
// /credit/i,
// /smart/i,
// /late/i,
// /history/i
// /soon/i
```

```
//
];

for( var key in descriptionsToTry ) {
    var description = descriptionsToTry[key];

    var numerator = db.loans.count({
        desc: description
    })
```

I didn't reach any solid conclusions from this, but it was worth trying. With more sophisticated language processing techniques, I think I would have had a better chance of finding actionable patterns.

MapReduce

MapReduce is a way to filter and process a large amount of records in parallel. With each record in your dataset, you apply some function **(map)** to transform/preprocess that record. Then, you output(emit) a key and value for each record. Finally, everything with the same key gets grouped together and processed in the **reduce** stage.

I was able to use MapReduce to do my "analysis" in a much cleaner, faster way. In my map function, I emitted a key of all the features I thought would be important in guessing if a borrower was "good" or "bad." The value emitted contained whether or not that loan was bad, and a constant "1" used to count the number of loans with the same key. The reduce function simply added up the counts.

```
// Map function
var key = {
    emp_length: this.emp_length,
    verification_status: this.verification_status,
    purpose: this.purpose,
    annual_inc: roundDown(this.annual_inc, 20000),
    home_ownership: this.home_ownership,
    fico_range_low: roundDown(this.fico_range_low, 50),
    total_acc: roundDown(this.total_acc, 10),
    revol_util: roundDown(this.revol_util, 10),
    loan_amnt: roundDown(this.loan_amnt, 3000)
};

var bad = 0;
if (this.loan_status == "Default" || this.loan_status == "Charged Off") {
        bad = 1;
}

emit(key, {
        count_bad: bad,
        count_total: 1
});


// Reduce function
reducedVal = {
        count_bad: 0,
        count_total: 0
};

for (var idx = 0; idx < array_values.length; idx++) {
        reducedVal.count_bad += array_values[idx].count_bad;
        reducedVal.count_total += array_values[idx].count_total;
}

return reducedVal;
```

Below is what the result of my MapReduce looks like. _id is the key I emitted.

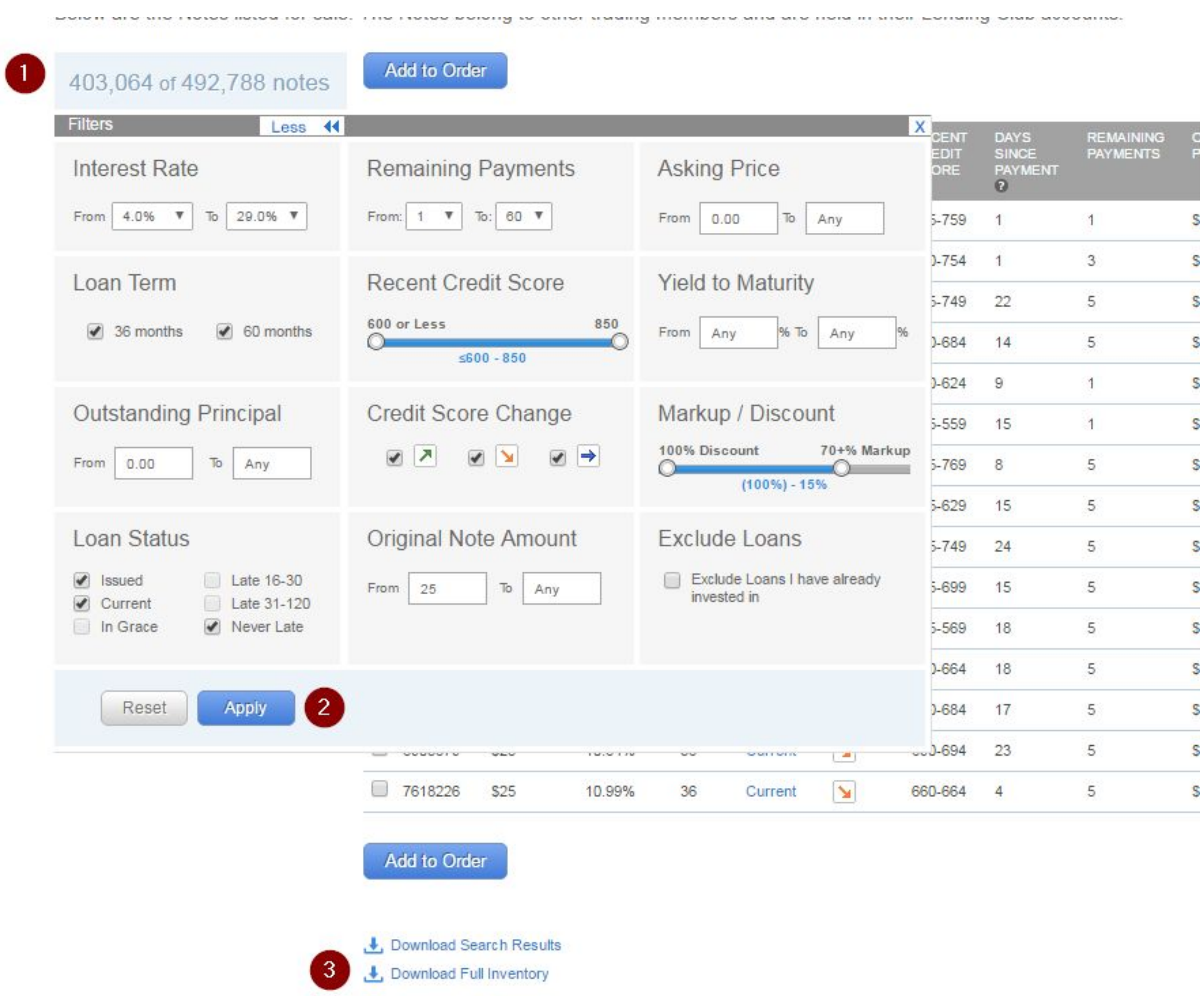
```
/* 1 */
{
    "_id" : {
        "emp_length" : "1 year",
        "verification_status" : "Not Verified",
        "purpose" : "car",
        "annual_inc" : 0,
        "home_ownership" : "RENT",
        "fico_range_low" : 650,
        "total_acc" : 0,
        "revol_util" : 30,
        "loan_amnt" : 3000
    },
    "value" : {
        "bad" : 0,
        "total" : 1,
        "ratio" : 0
    }
}

/* 2 */
{
    "_id" : {
        "emp_length" : "1 year",
        "verification_status" : "Not Verified",
        "purpose" : "car",
        "annual_inc" : 0,
        "home_ownership" : "RENT",
        "fico_range_low" : 650,
        "total_acc" : 0,
        "revol_util" : 50,
        "loan_amnt" : 3000
    },
    "value" : {
        "bad" : 0,
        "total" : 1,
        "ratio" : 0
    }
}
```

Later, I can use this collection which I called “person_analysis” to estimate risk for a loan. If I’m happy with the ratio, I’ll buy a note for that loan.

Step 3. Scrape secondary market loans

The secondary market allows investors to buy or sell existing notes

(1) Typically there are hundreds of thousands of notes available on the secondary market.
(2) FolioFN Provides a way to filter all these notes.
(3) You can then download a csv file containing only filtered notes or all notes on the secondary market (400k in the screenshot)



Using "Download Full Inventory" is easy, but it can show notes from loans I already invested in. I don't want to buy multiple notes from the same loan because it increases the penalty if that loan defaults. At the time I worked on this, the Foliofn API didn't exist so I had to go to this page each day, apply my filters, and then download the results. After maybe 2 days I was sick of it so I decided to automate.

PhantomJS lets me pretend to be a user clicking things in a browser (headless browser). CasperJS makes using PhantomJS easier, and is what I used to accomplish this task.

```
casper.then( function() {
    this.capture( 'filled filters.png' );
});

casper.thenClick( "#filter-body input[value=Apply]" );

casper.wait( 2000, function() {
    this.capture( 'filtered results.png' );
    console.log( "Downloading notes" );
    casper.download( casper.getElementInfo( '#notesDownloadSearchResultsLink' )['attributes']['href'],
'browseNotes.csv' );
})
```

Step 4. Find and buy good notes

Loan buying backend

At this point I have

1. A way to get notes that are currently for sale
2. A database collection of existing loans
3. A database collection of predicted risk

So I put them all together into a PHP script.

First run the CasperJS script to download the notes for sale for me.

```
system( 'casperjs --ssl-protocol=tlsv1 C:\casperjs\bin\lendingclub.js' );
```

Then, I load all those notes into a temporary MongoDB collection called *notes*. I could've just parsed the csv lines in PHP but I wanted to do it with MongoDB.

```
system( 'D:\Mongo\bin\mongoimport.exe /h localhost /p 27017 /d lc /c notes /numInsertionWorkers 2 /type
csv /headerline C:\xampp\htdocs\lc\browseNotes.csv');
```

Next I go through each of these notes for sale, and look up the original loan data using LoanId.

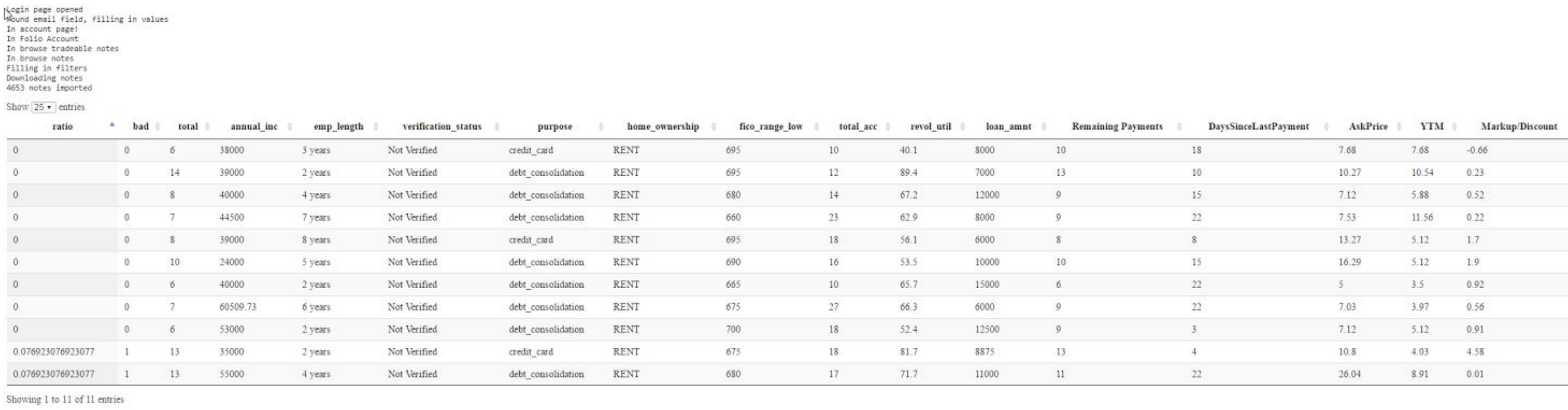
```
$loanInMongo = $loansDB->findOne(
    json_decode( '
        {
            "id": ' . $folioNote['LoanId'] . '
        }
    ')
);
```

If I get a match, I'll run the loan data against my predicted risk collection and add the risk to that note data. Then, I can go back and query all the notes that have an estimated risk and filter by some maximum risk and some minimum sample size. I print the result in JSON for the frontend to display.

```
$notesDB->find(
    Array(
        "ratio" => Array (
            '$lt' => .08
            ),
        "total" => Array (
            '$gt' => 5
            )

        )
    );
```

Loan buying front-end

I used DataTables in the frontend to display the notes for the ability to sort columns. I also added a column with links to Open (see more details about the note) or Add to order (add to cart).



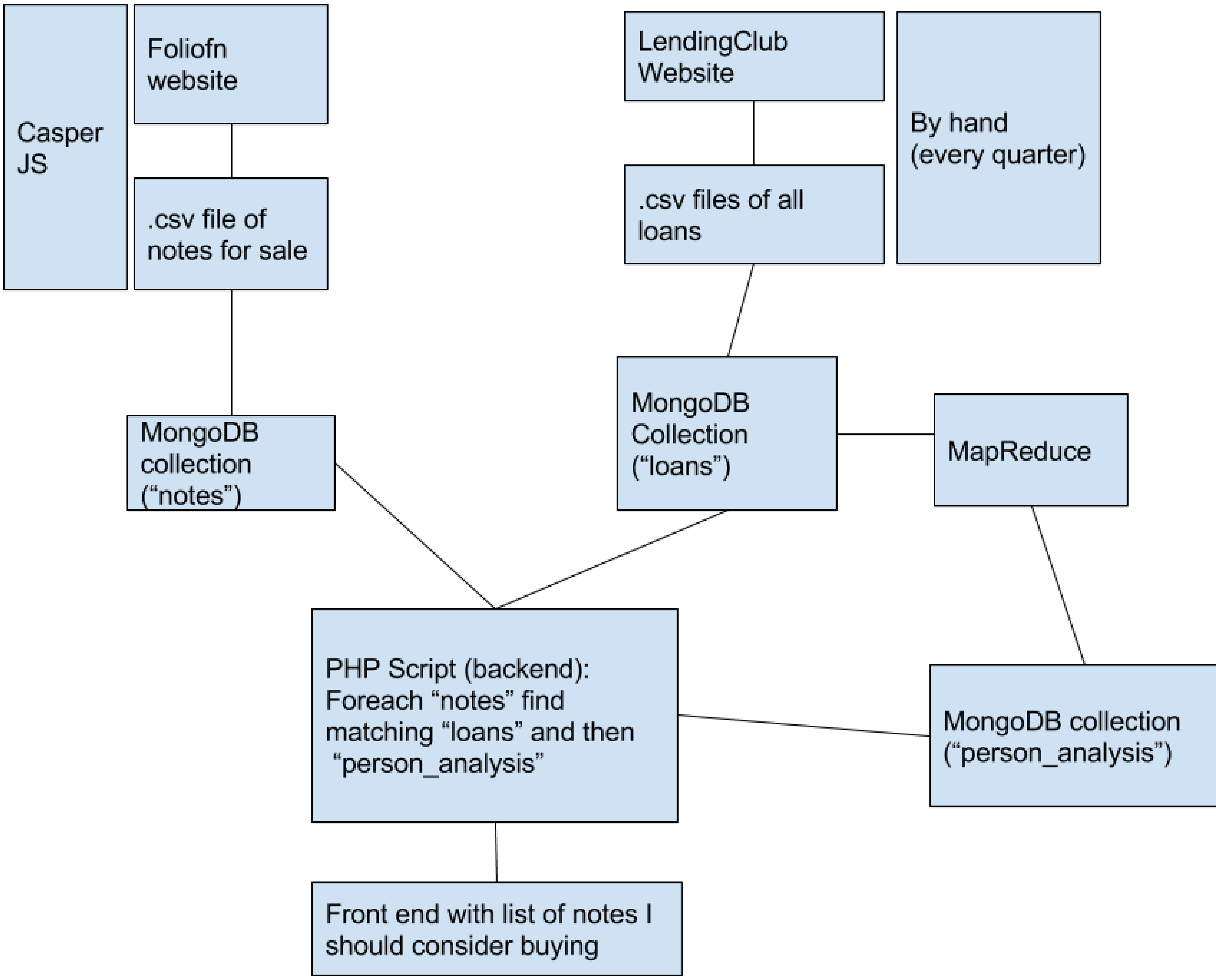
```
Login page opened
Found email field, filling in values
In account page!
In Folio Account
In browse tradeable notes
In browse notes
Filling in filters
Downloading notes
4653 notes imported
```

Show 25 entries

ratio	bad	total	annual_inc	emp_length	verification_status	purpose	home_ownership	fico_range_low	total_acc	revol_util	loan_amnt	Remaining Payments	DaysSinceLastPayment	AskPrice	YTM	Markup/Discount
0	0	6	38000	3 years	Not Verified	credit_card	RENT	695	10	40.1	8000	10	18	7.68	7.68	-0.66
0	0	14	39000	2 years	Not Verified	debt_consolidation	RENT	695	12	89.4	7000	13	10	10.27	10.54	0.23
0	0	8	40000	4 years	Not Verified	debt_consolidation	RENT	680	14	67.2	12000	9	15	7.12	5.88	0.52
0	0	7	44500	7 years	Not Verified	debt_consolidation	RENT	660	23	62.9	8000	9	22	7.53	11.56	0.22
0	0	8	39000	8 years	Not Verified	credit_card	RENT	695	18	56.1	6000	8	8	13.27	5.12	1.7
0	0	10	24000	5 years	Not Verified	debt_consolidation	RENT	690	16	53.5	10000	10	15	16.29	5.12	1.9
0	0	6	40000	2 years	Not Verified	debt_consolidation	RENT	665	10	65.7	15000	6	22	5	3.5	0.92
0	0	7	60509.73	6 years	Not Verified	debt_consolidation	RENT	675	27	66.3	6000	9	22	7.03	3.97	0.56
0	0	6	53000	2 years	Not Verified	debt_consolidation	RENT	700	18	52.4	12500	9	3	7.12	5.12	0.91
0.076923076923077	1	13	35000	2 years	Not Verified	credit_card	RENT	675	18	81.7	8875	13	4	10.8	4.03	4.58
0.076923076923077	1	13	55000	4 years	Not Verified	debt_consolidation	RENT	680	17	71.7	11000	11	22	26.04	8.91	0.01

Showing 1 to 11 of 11 entries

Block Diagram



Along came Pandas

The Mongo+PHP+CasperJS system was great for a while. But then I found Pandas, a library for Python that makes data munging and analysis much more convenient. I didn't write about all of it here, but it's on GitHub (https://github.com/dharik/Lending-Club).

Lessons Learned

Use the best tool for the job.

Early on, I recognized that the loan data was a good fit for a relational database but I instead used MongoDB. It was nice to learn and use MongoDB as well as get experience with MapReduce.

Now, I could have used MongoDB in a way that it's better suited, such as putting each note as a subdocument for the loan it belongs to. But I stopped due to time and LendingClub having some big problems.

Use version control.

While I did my "analysis," all along the way I was modifying my files and not saving copies of them anywhere. There was some good stuff I found that I ended up erasing just because I wanted to try something else. It would've helped to be able to go back to a previous state.

Recognize Terminology

Many of the steps I performed on my own intuition have names, which I tried to incorporate in my writing. For example, my "Step 1" was an Extract, Transform, Load (ETL) pipeline. Knowing what the process I'm working on is called can make it easier to study and learn.