imaginator: An R Package for Detailed Claim Simulation

Introducing imaginator

What is it?

Figure 1: definition

Why does this package exist?

- Stanard [1985] is no longer on the syllabus - let's teach these young kids how it's done.
- Nothing comparable to the Meyers and Shi research database for individual claims
- Brian thinks everything should be a package

From [Venter, 1998]

> The fact that Stanard used the simulation method consistent with the BF emergence pattern […] suggests that actuaries may be more comfortable with the BF emergence assumptions than with those of the chain ladder. Or perhaps it just means that no one would be likely to think of simulating losses by the chain ladder method.

About the package

- On CRAN, click here, easy install
- There will probably be updates
- Please report bugs and suggest new features! click me

```r
# install official release
install.packages("imaginator")

# install beta
devtools::install_github("PirateGrunt/imaginator")
```

```r
library(imaginator)
```

Function helpers

Function helpers

- Use functions to create functions
- Function parameters are vectorized to return a list of functions

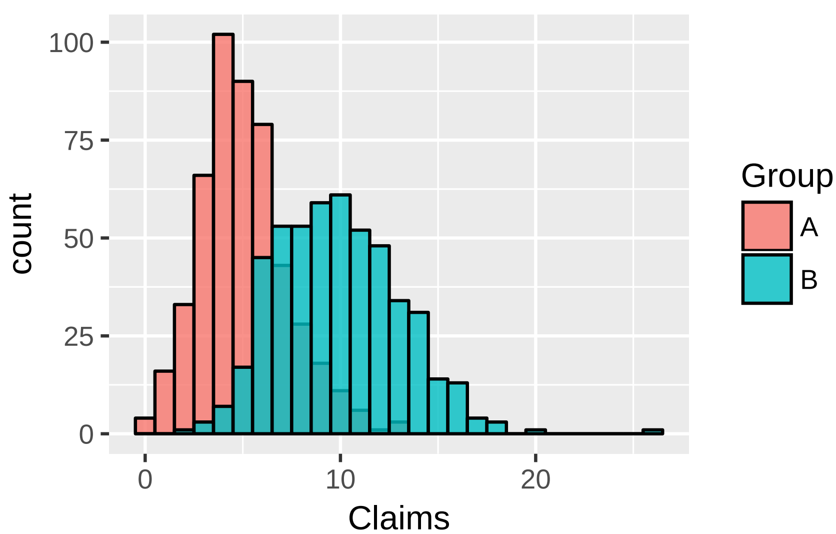
```
pois5 <- PoissonHelper(5)
pois10 <- PoissonHelper(10)
class(pois10)
## [1] "function"
```

How does that look?

```
library(ggplot2)
set.seed(1234)

dfClaims <- rbind(data.frame(Group = "A", Claims = pois5(500)),
    data.frame(Group = "B", Claims = pois10(500)))

plt <- ggplot(dfClaims, aes(Claims, fill = Group))
plt <- plt + geom_histogram(binwidth = 1, color = "black",
    alpha = 0.8, position = "identity")
```



Vectorization

Passing in a vector of parameters will return a list of functions

```
pois <- PoissonHelper(c(5, 10))
summary(pois)
```

```
##      Length Class  Mode
## [1,] 1      -none- function
## [2,] 1      -none- function
```

Simulating across the list

```
pois <- PoissonHelper(c(5, 10))
lapply(pois, function(x) {
    summary(x(50))
})
## [[1]]
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##    0.00    4.00    5.00    4.94    7.00
##    Max.
##    9.00
##
## [[2]]
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##    4.00    8.00   10.00   10.02   12.00
##    Max.
##   17.00
```

Simulating policies

Simulate Policies

```
set.seed(1234)
dfPolicies <- SimulatePolicies(N = 2, NumYears = 5)
```

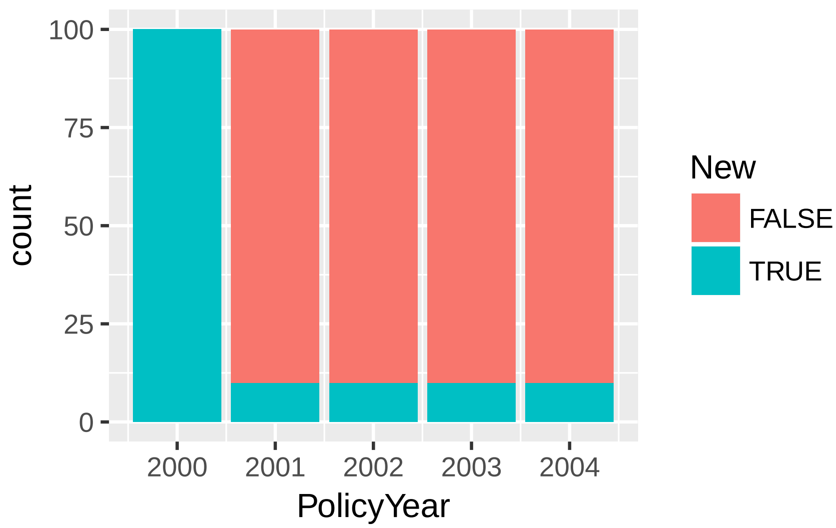
PolicyEffectiveDate	PolicyExpirationDate	Exposure	PolicyholderID
2000-02-12	2001-02-10	1	1
2000-08-16	2001-08-15	1	2
2001-02-11	2002-02-10	1	1
2001-08-16	2002-08-15	1	2
2002-02-11	2003-02-10	1	1
2002-08-16	2003-08-15	1	2

Growth and retention

- Retention is a number between 0 and 1, representing the % of policies which renew
- Growth is a non-negative number indicating the portion of new policies relative to the expiring book.

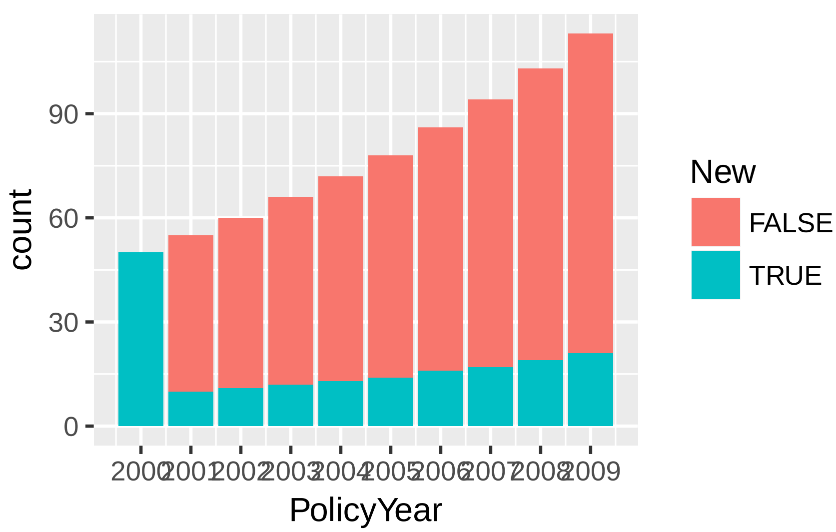
Growth and retention equal

```
dfPolicies <- SimulatePolicies(N = 100, NumYears = 5,
    Retention = 0.9, Growth = 0.1)
```



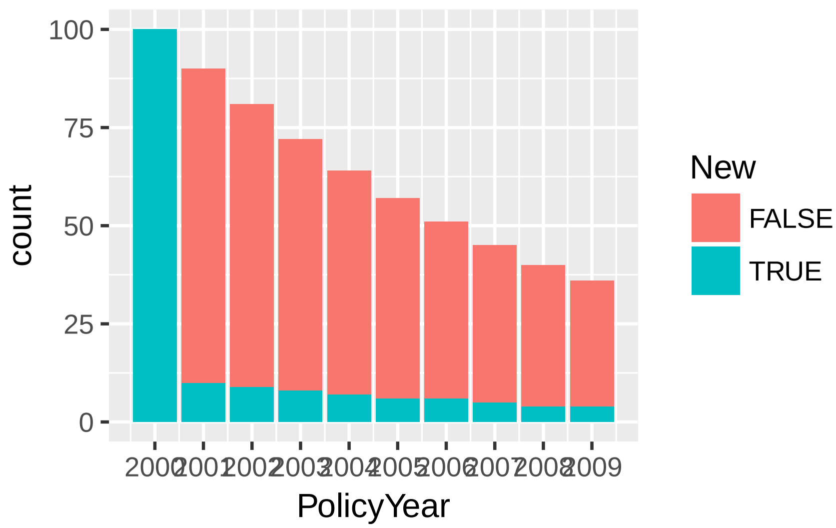
Gradually expanding book

```
dfPolicies <- SimulatePolicies(N = 50, NumYears = 10,
    Retention = 0.9, Growth = 0.2)
```



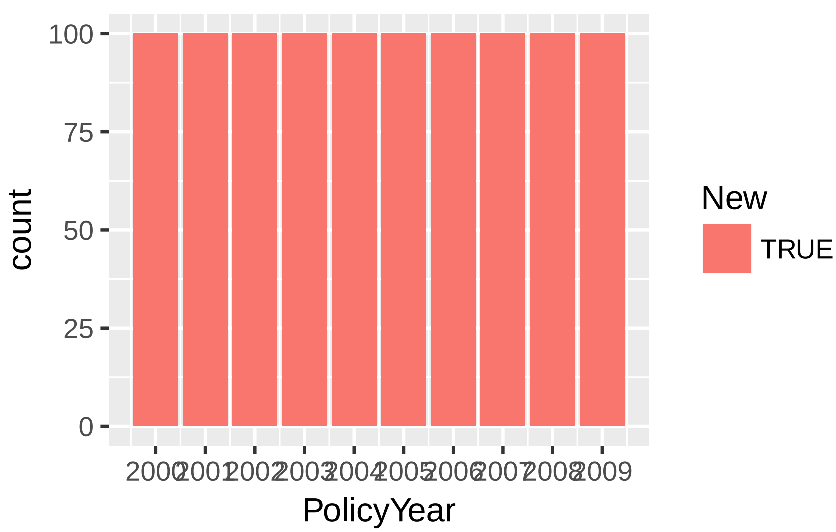
Gradually contracting book

```
dfPolicies <- SimulatePolicies(N = 100, NumYears = 10,
    Retention = 0.8, Growth = 0.1)
```



Complete turnover every year

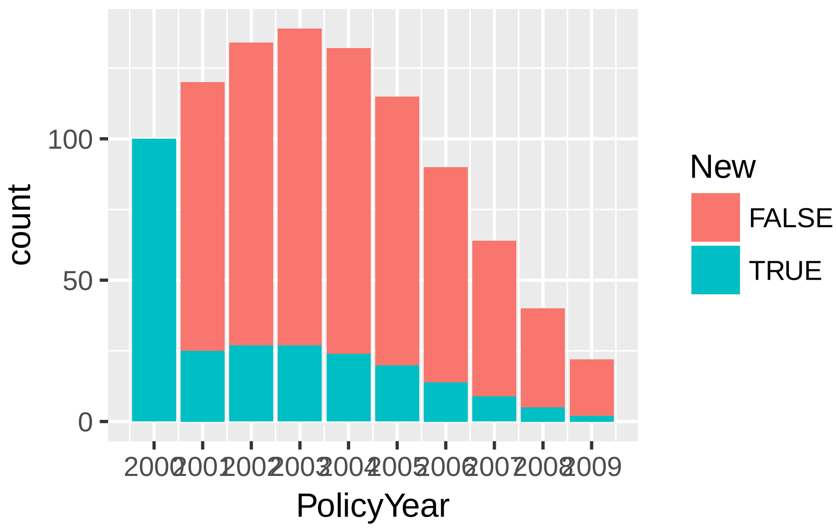
```
dfPolicies <- SimulatePolicies(N = 100, NumYears = 10,
    Retention = 0, Growth = 1)
```



Different growth and retention by year

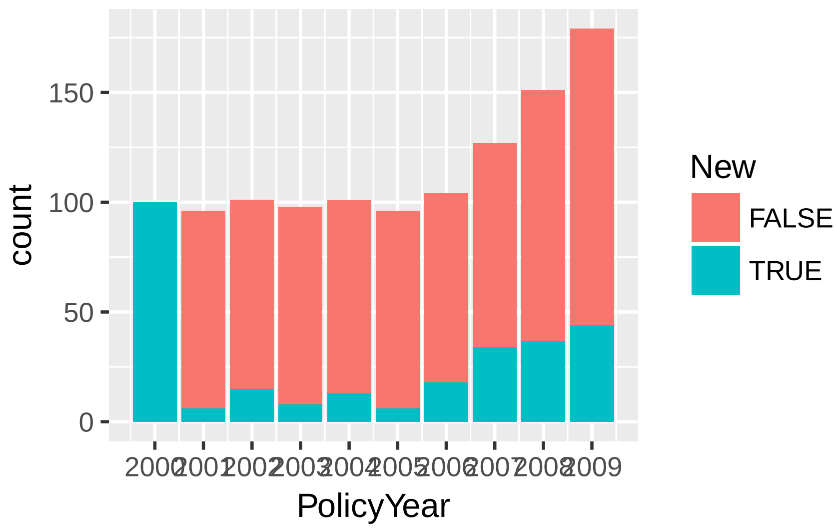
Note these differences are deterministic

```
dfPolicies <- SimulatePolicies(N = 100, NumYears = 10,
    Retention = seq(length.out = 9, from = 0.95,
        to = 0.5), Growth = seq(length.out = 9,
        from = 0.25, to = 0.05))
```



Stochastic growth

```
dfPolicies <- SimulatePolicies(N = 100, NumYears = 10,
    Retention = 0.9, Growth = runif(9, 0.05, 0.35))
```



Additional columns

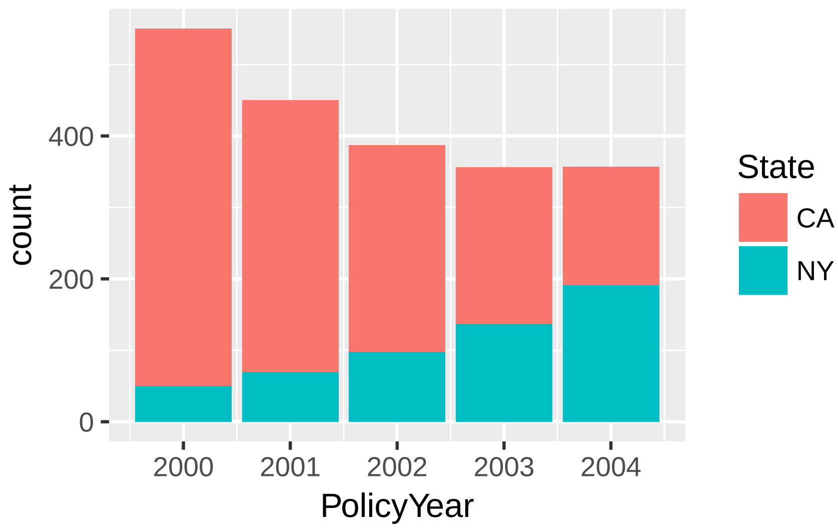
Used to add descriptive names for the set of policies. Theses may then be bound into a single data frame.

Below, we simulate decline in one state and rapid growth in another.

```
dfGL_CA <- SimulatePolicies(N = 500, NumYears = 5,
    Retention = 0.75, Growth = 0.01, AdditionalColumns = list(Line = "GL",
        State = "CA"))

dfGL_NY <- SimulatePolicies(N = 50, NumYears = 5,
    Retention = 0.9, Growth = 0.5, AdditionalColumns = list(Line = "GL",
        State = "NY"))

dfGL <- dplyr::bind_rows(dfGL_CA, dfGL_NY)
```



Claims by wait time

The Algorithm

- Start with a data frame of policies.
- For each row, simulate number of claims
- For each claim, simulate the number of transactions
- Simulate lags until occurrence, lag until report, lag until transaction
- Each transaction has a random severity

Get some policies

```
set.seed(12345)
dfPolicy <- SimulatePolicies(2, 2001:2005)
```

PolicyEffectiveDate	PolicyExpirationDate	Exposure	PolicyholderID
2001-09-21	2002-09-20	1	1
2001-11-16	2002-11-15	1	2
2002-11-16	2003-11-15	1	2
2002-09-21	2003-09-20	1	1
2003-11-16	2004-11-15	1	2

Now create some transactions

```
dfClaimTransactions <- ClaimsByWaitTime(dfPolicy,
    ClaimFrequency = FixedHelper(2), PaymentFrequency = FixedHelper(3),
```

```
    OccurrenceWait = FixedHelper(10), ReportWait = FixedHelper(5),
    PayWait = FixedHelper(5), PaySeverity = FixedHelper(50))
```

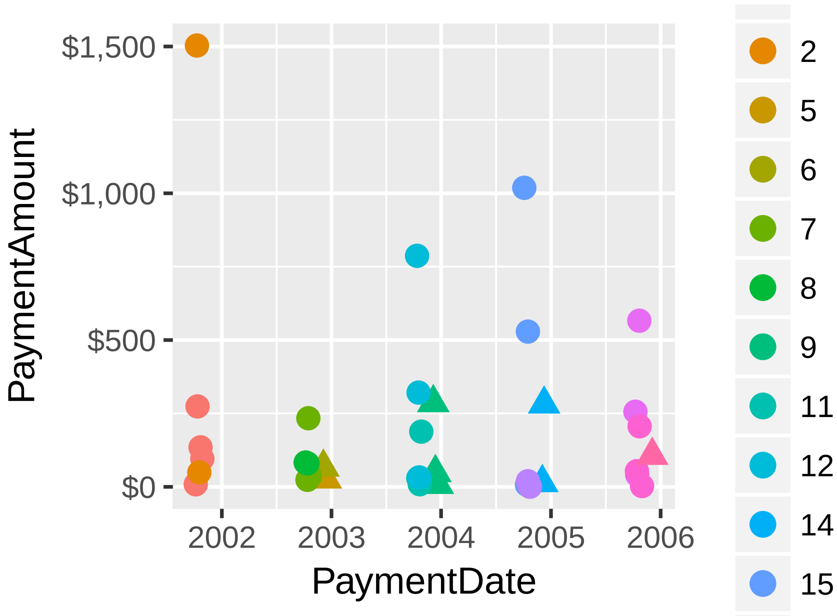
And here they are

ClaimID	OccurrenceDate	ReportDate	PaymentDate	PaymentAmount
1	2001-10-01	2001-10-06	2001-10-11	50
1	2001-10-01	2001-10-06	2001-10-16	50
1	2001-10-01	2001-10-06	2001-10-21	50
2	2001-10-01	2001-10-06	2001-10-11	50
2	2001-10-01	2001-10-06	2001-10-16	50
2	2001-10-01	2001-10-06	2001-10-21	50
3	2001-11-26	2001-12-01	2001-12-06	50
3	2001-11-26	2001-12-01	2001-12-11	50
3	2001-11-26	2001-12-01	2001-12-16	50

Some columns have been removed

Stochastic amounts

```
dfClaimTransactions <- ClaimsByWaitTime(dfPolicy,
    ClaimFrequency = FixedHelper(2), PaymentFrequency = PoissonHelper(2),
    OccurrenceWait = PoissonHelper(10), ReportWait = PoissonHelper(5),
    PayWait = PoissonHelper(5), PaySeverity = LognormalHelper(log(50),
        0.5 * log(50)))
```



More claims, please

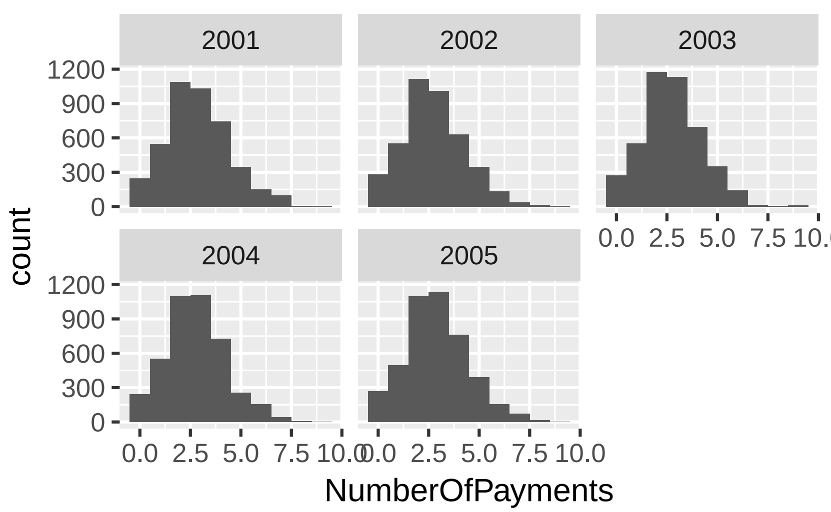
```
dfPolicy <- SimulatePolicies(1000, 2001:2005)

dfClaimTransactions <- ClaimsByWaitTime(dfPolicy,
    ClaimFrequency = PoissonHelper(2 * seq.int(5)),
    PaymentFrequency = PoissonHelper(2), OccurrenceWait = PoissonHelper(180),
    ReportWait = PoissonHelper(90), PayWait = PoissonHelper(45),
    PaySeverity = LognormalHelper(log(50), 0.5 *
        log(50))) %>% mutate(PolicyYear = lubridate::year(PolicyEffectiveDate))

dfPolicyClaims <- dfClaimTransactions %>% mutate(NumClaims = ifelse(is.na(ClaimID),
    0, 1)) %>% group_by(PolicyholderID, PolicyYear) %>%
    summarize(NumClaims = sum(NumClaims))
```

```
lapply(split(dfPolicyClaims, dfPolicyClaims$PolicyYear),
    function(x) {
        summary(x$NumClaims)
    })
## $`2001`
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##   0.000   2.000   4.000   4.259   6.000
##    Max.
```

```
##  20.000
##
## $`2002`
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##    0.00    1.00    4.00    4.12    6.00
##    Max.
##   16.00
##
## $`2003`
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##   0.000   2.000   4.000   4.358   6.000
##    Max.
##  24.000
##
## $`2004`
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##   0.000   1.000   3.000   4.186   6.000
##    Max.
##  19.000
##
## $`2005`
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##   0.000   2.000   4.000   4.386   7.000
##    Max.
##  24.000
```



Claims by lag

Claims by lag

- Basically chain ladder for individual claims
- Distinguishes bewtten IBNYR and IBNER
- My biggest complaint is that a structure - evaluation dates - is imposed on our data
- Could be a gateway drug to more advanced individual claim models
- Lag is ambiguously defined - on purpose. It could be policy year or accident year.
- Does it make sense? To me, not so much. *However* just because I can't see the math, doesn't mean that it isn't there.

First generate IBNYR data

```
set.seed(12345)
dfPolicy <- SimulatePolicies(2, 2001:2004)

dfIBNYR_Fixed <- ClaimsByFirstReport(dfPolicy,
    Frequency = FixedHelper(4:1), PaymentSeverity = FixedHelper(rep(250,
        4)), Lags = 1:4)
```

PolicyholderID	PolicyEffectiveDate	ClaimID	Lag	PaymentAmount
1	2001-09-21	1	1	250
1	2001-09-21	2	1	250
1	2001-09-21	3	1	250
1	2001-09-21	4	1	250
1	2001-09-21	33	2	250
1	2001-09-21	34	2	250
1	2001-09-21	35	2	250
1	2001-09-21	57	3	250
1	2001-09-21	58	3	250
1	2001-09-21	73	4	250

Our IBNYR count triangle - complete

PolicyholderID	PolicyYear	1	2	3	4
1	2001	4	3	2	1
1	2002	4	3	2	1
1	2003	4	3	2	1

PolicyholderID	PolicyYear	1	2	3	4
1	2004	4	3	2	1
2	2001	4	3	2	1
2	2002	4	3	2	1
2	2003	4	3	2	1
2	2004	4	3	2	1

… and incomplete

PolicyholderID	PolicyYear	1	2	3	4
1	2001	4	3	2	1
1	2002	4	3	2	
1	2003	4	3		
1	2004	4			
2	2001	4	3	2	1
2	2002	4	3	2	
2	2003	4	3		
2	2004	4			

Payments

PolicyholderID	PolicyYear	1	2	3	4
1	2001	1000	750	500	250
1	2002	1000	750	500	
1	2003	1000	750		
1	2004	1000			
2	2001	1000	750	500	250
2	2002	1000	750	500	
2	2003	1000	750		
2	2004	1000			

Now develop the claims

ClaimsByLinkRatio takes a data frame of claims by Lag and develops them as appropriate.

```
dfClaimsFixed <- ClaimsByLinkRatio(dfIBNYR_Fixed,
    Links = FixedHelper(c(2, 1.5, 1.25)), Lags = 1:4)
```

Note that we now have more than one observation for each claim

PolicyholderID	PolicyEffectiveDate	ClaimID	Lag	PaymentAmount
1	2001-09-21	1	1	250.0
1	2001-09-21	1	2	500.0
1	2001-09-21	1	3	750.0
1	2001-09-21	1	4	937.5

Add some variation

```
dfIBNYR_Variable <- ClaimsByFirstReport(dfPolicy,
    Frequency = PoissonHelper(4:1), PaymentSeverity = GammaHelper(rep(1500,
        4), rep(5, 4)), Lags = 1:4)

dfClaimsVariable <- ClaimsByLinkRatio(dfIBNYR_Variable,
    Links = GammaHelper(c(10, 15, 20), c(5, 10,
        18)), Lags = 1:4)
```

Making triangles

Claims by lag

You're pretty much done, unless you want to aggregate the results

```
dfAgg <- dfClaimsVariable %>% mutate(PolicyYear = lubridate::year(PolicyEffectiveDate)) %>%
    dplyr::filter(PolicyYear + Lag - 1 <= 2004) %>%
    group_by(PolicyYear, Lag) %>% summarise(Paid = sum(PaymentAmount),
    ClaimCount = n())
```

PolicyYear	Lag	Paid	ClaimCount
2001	1	4194.4898	14
2001	2	12812.4488	24
2001	3	19544.7604	28
2001	4	20718.3283	29
2002	1	894.7109	3
2002	2	3893.3006	7
2002	3	7734.5361	13
2003	1	1529.1519	5
2003	2	4475.5000	12
2004	1	2078.8361	7

Claims by wait time

We need to impose an evaluation date structure on the data

```
first_eval <- min(dfClaimTransactions$PolicyEffectiveDate)
lubridate::month(first_eval) <- 12
lubridate::day(first_eval) <- 31

last_eval <- max(dfClaimTransactions$PaymentDate,
    na.rm = TRUE)
lubridate::month(last_eval) <- 12
lubridate::day(last_eval) <- 31
evalDates <- seq.Date(from = first_eval, to = last_eval,
    by = "year")
evalDates
## [1] "2001-12-31" "2002-12-31" "2003-12-31"
## [4] "2004-12-31" "2005-12-31" "2006-12-31"
## [7] "2007-12-31"
```

And then aggregate

Just a bit of jiu-jitsu

```
ComposeDiagonal <- function(df, eval_date) {
    df <- df %>% dplyr::filter(PaymentDate <=
        eval_date) %>% mutate(PolicyYear = lubridate::year(PolicyEffectiveDate),
        Lag = lubridate::year(eval_date) - PolicyYear +
            1) %>% group_by(PolicyYear, Lag) %>%
        summarise(PaymentAmount = sum(PaymentAmount,
            na.rm = TRUE))
}

dfTriangle <- lapply(evalDates, function(x) {
    ComposeDiagonal(dfClaimTransactions, x)
})
dfTriangle <- do.call(rbind, dfTriangle)
```

And we have a triangle

PolicyYear	Lag	PaymentAmount
2001	1	66464.06
2001	2	1198941.36
2001	3	1242390.00
2001	4	1242390.00
2001	5	1242390.00

PolicyYear	Lag	PaymentAmount
2001	6	1242390.00
2001	7	1242390.00

But, hang on

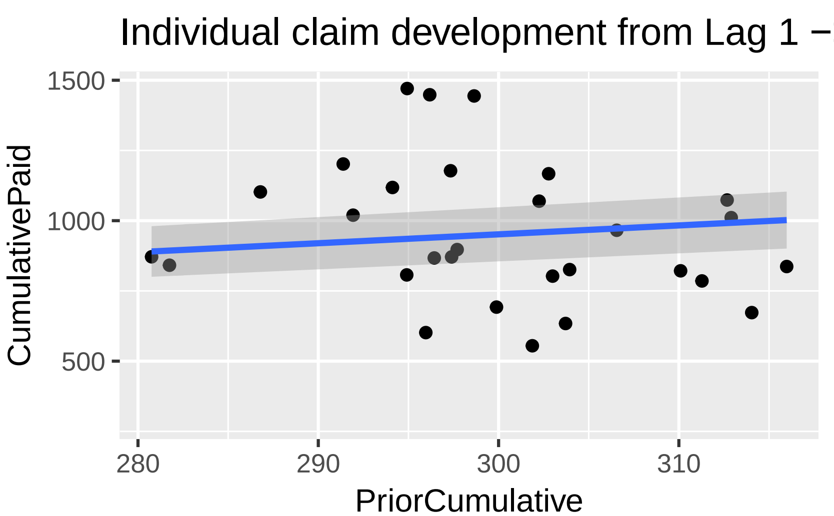
Why are you creating triangles?

Individual claim data contains all of the features that you need to build a model.

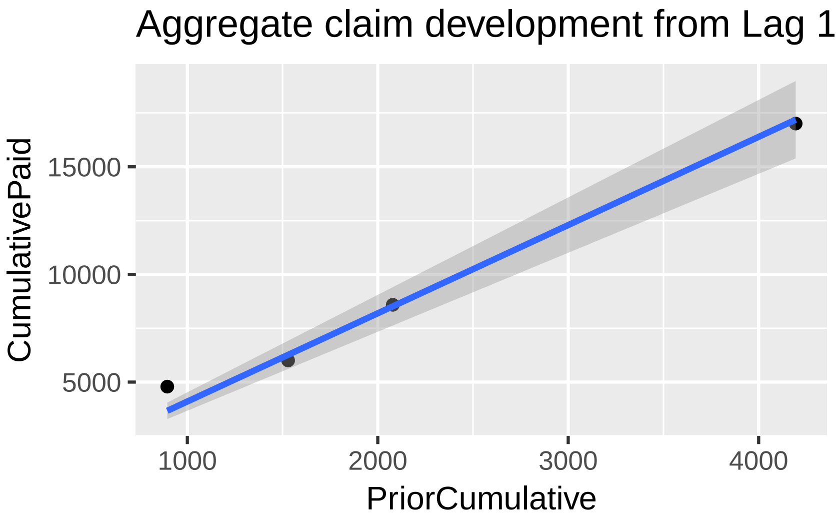
Create a chain ladder by forming cumulative amounts and lagging

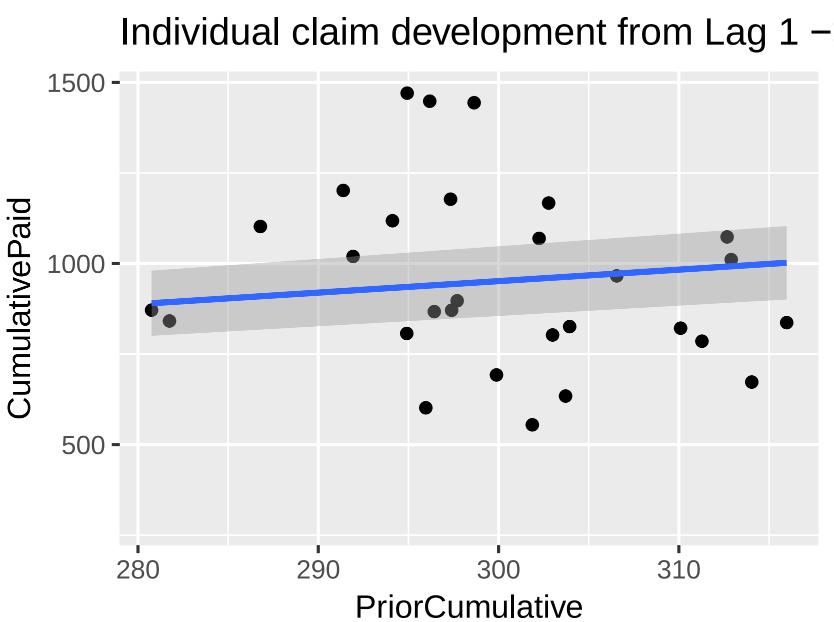
```
dfChainLadder <- dfClaimsVariable %>% arrange(ClaimID,
    Lag) %>% group_by(ClaimID) %>% mutate(CumulativePaid = cumsum(PaymentAmount),
    PriorCumulative = dplyr::lag(CumulativePaid))
```

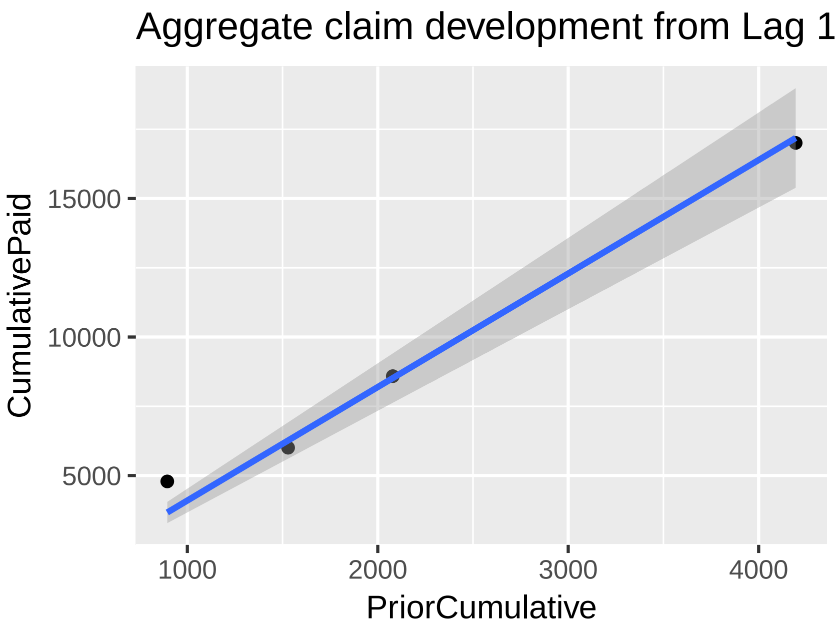
How does that look?



And aggregate







And remember

- We had to drop a number of points in the chart because they were IBNYR claims, i.e. they had no prior cumulative.
- Common aggregate triangles miss this.

Triangles for wait time

This slide will be complete by the time I present.

Wrapping up

Go forth and simulate!

- If you have individual claim data, **start using it!**
- If you don't have enough, simulate as a starting point to study the dynamics
- If you curious about "what if", `imaginator` can help you contemplate scenarios
- Want more features? Make suggestions on GitHub: click me

Thank you!

The source code for these slides may be found here: `https://github.com/PirateGrunt/clrs_2017_imaginator`

Bibliography

References

James N. Stanard. A simulation test of prediction errors of loss reserve estimation techniques. 1985. URL https://www.casact.org/pubs/proceed/proceed85/85124.pdf.

Gary Venter. Testing the assumptions of age-to-age factors. 1998. URL https://www.casact.org/pubs/proceed/proceed98/980807.pdf.