Proposal for reserves forecasting on car insurance products for All Nation Insurance Co.

Universidad Nacional de Colombia - Maestría en Actuaría y Finanzas (Dec.-23)

Business understanding

The aim of this document is to present a robust methodology for reserving forecasting which is an essential part of insurance business. In particular, the proposal is presented to All Nation Insurance Co. which is a large size insurer based in Miami, FL, the objective of this firm is to provide insurance for NY residents & businesses, Auto, Home & Business Insurance at low rates, and with a top notch service. Our purpose is to support that mission with data mining and analitycs solutions, bringing our customers the best suggestions, forecasts and outlooks to take properly informed solutions.

We are focusing on auto insurance products, defined by a contract (or policy) in which insurer takes the obligation (liability) to respond on car losses of the insured in case of a claim caused by a sinister. The pricing of the policy depends on risks asocciated with car model, insured profiling, market bemchmarks, type and extension of coverage, and other insurer's operational costs.

In this context, the insurers are seeking for the best information that could allow a better risk asessment in order to correctly manage liabilities. From a financial perspective, every policy that an insurer takes must have an adequate reserve in case the contract is executed. The benefit of the company comes from the inexecution of claims and is always a win to win for both parts as insured hedge from financial risks that could arise from possible car accidents. As regards the latter, the machine learning methodologies are striking insuring industry with outstanding performance of reserves forecasting over traditional procedures and this is the value offer that we are offering to this company.

Next, the objectives, quality criteria, procedures, budgeting, requirements, duration, analysis and results are provided for discussion as an offering of analytics services.

Statement of Business Objective:

The main goal of the project is to create a forecasting model on provisions for insurance companies. First, let's see some concepts related:

The insurance business is to offer benefits for covering several types of risks. This benefits are paid to the insured (or customer) in form of periodic payments (or premiums) when the determined claim takes place.

- Claims dynamics: Premiums refer to the regular payments that an insured person makes to the insurance company to keep their coverage active. The development of a claim requires time. The presence of this delay in the processing of a claim forces the insurer to have capital to settle these claims in the future.

- The inverted production cycle of the insurance market and the claim dynamics motivate the need for reserving and the design of predictive modeling tools to estimate reserves. In insurance, the premium income precedes the costs. An insurer will charge a client a premium, before actually knowing how costly the insurance policy or contract will become. In typical manufacturing industry this is not the case and the manufacturer knows - before selling a product - what the cost of producing this product was. At a specified evaluation moment the insurer will predict outstanding liabilities with respect to contracts sold in the past. This is the claims reserve or loss reserve; it is the capital necessary to settle open claims from past exposures. It is a very important element on the balance sheet of the insurer, more specifically on the liabilities side of the balance sheet, as could be a large source of costs that could be determinant for a company's financial results.

Statement of Data Mining Objective

- The objective of data mining procedure is to establish a solid base from where the diagnosis, modeling techniques and conclusions could be accurately and useful for the business. Particularly, the building of provisions data base includes identificaiton of claims, reserves and costs associated to insurances issued from 1987 to 1997, which are covering 10 year span of development of benefits.

- The data mining procedure will be used as source of information for the development of Machine Learning forecasting techniques which are intended to improve an industry standard, most known as the *chain ladder method*.

Statement of Success Criteria

As the business needs are related directly with the liabilities optimization, diminishing costs and making a more efficient use of capital resources, next are presented tow main indicators of the project's contributions:

- In line with the data mining objective, the success criteria must be determined by the improvement of backtesting metrics compared to the base model mentioned before, or proving that standard method is significantly better than any other.
- As well as the data analysis and business understanding is useful for modelling purposes, any relevant conclusion in this way must be included in results and could be a success measurement of the project.

Focuses on understanding the project objectives and requirements from a business perspective, then converting this knowledge into a data mining problem definition and a preliminary plan designed to achieve the objectives.

Situation Assessment

For this project, the insurance company has access to all data on claims and commercial auto insurance on sector, such as premium values, claim values, accident dates, cumulative paid losses, and assigned expenses. Currently, the insurance company has the required professionals and staff to complete the data mining project, as it has experts in reserve calculations and qualified personnel in data mining methodologies. The major risks of the project include implementing a method that performs worse than the traditional method, leading to poor solvency and potential financial difficulties for the company. The contingency plan is to abandon the data mining model if the objectives are not met annually and quickly revert to using the traditional Chain-Ladder method.

Inventory of Resources

- Hardware Requirements: Computers, servers, CPUs, GPUs, sufficient RAM, SSDs, Cloud Storage, good internet connection, server cluster, backup.
- Data Sources and Knowledge Warehouses: Data is stored on the company's servers, with access for project professionals. The company also has resources to acquire external databases.
- Personnel Resources: The project team consists of experts in insurance business, actuarial calculations, programming, and data analysis. Additionally, there are IT professionals with expertise in database administration, ETL, modeling, and data analysis.

Requirements, Assumptions, and Constraints

- Legal Requirements: Adherence to data handling norms such as unauthorized access, sharing, collection, retention, and use of data.

- Assumptions: No competition within the organization, data quality is maintained, and project sponsors will be shown the process and results.
- Verification of Constraints: Project personnel have the necessary credentials. No legal restrictions on using customer data. Financial support is available.

Risks and Contingencies

- Programming: Potential delays may be addressed by intensifying efforts or adjusting the project timeline.
- Financial: Refinancing or completing the project with available resources in case of resource shortage.
- Data Quality: Specialized team to handle errors in data.
- Results: If initial results are not impactful, reconsidering the need for a change or exploring alternative data mining methodologies.

Glosary

Data Mining: The process of discovering patterns, trends, and hidden knowledge in large datasets using statistical and machine learning techniques.

Reserve of Claims: An estimate of the amount of money an insurance company should reserve to cover pending and future insurance claims.

Claims Data: Detailed information about reported claims, including dates, descriptions, estimated costs, and payments made.

Claims Development: The process by which the costs of a claim increase or decrease over time as claims are investigated, processed, and resolved.

Claims Triangle: A tabular representation of claims over time, showing when they were reported, how much was paid in each period, and how much remains pending.

Development Rate: The average rate at which claim costs increase or decrease over time based on historical data analysis.

Reserve Model: A mathematical or statistical model used to forecast future claim costs and ultimately calculate the necessary reserves.

Reserve Adjustment: Changes made to reserve estimates as more data is obtained or the reserve model is updated.

Triangle Analysis: A technique used to estimate reserves based on information contained in the claims triangle.

Excess Loss: The amount of money an insurer is willing to pay above a certain limit before reinsurance coverage is activated.

Reinsurance: An agreement in which an insurance company transfers part of its risks to another insurance company or reinsurer to limit potential losses.

Commercial Liability Coverage: A type of insurance that provides protection against claims for bodily injury or property damage that may arise in the course of commercial operations.

Machine Learning Model: An approach that uses algorithms and machine learning techniques to analyze historical data and make predictions about future claims and costs.

Cross-Validation: A technique used to evaluate the accuracy and effectiveness of a machine learning model by dividing the data into training and testing sets.

Risk Classification: The process of categorizing different types of risks and assessing their probability and severity.

Insurance Premiums: Regular payments made by policyholders to the insurance company in exchange for insurance coverage.

Commercial Auto Insurance: A type of insurance that provides coverage for vehicles used for commercial purposes, such as trucks, vans, and vehicle fleets.

Policy: A legal contract that establishes the terms and conditions of insurance coverage, including covered risks, premiums, and other details.

Premiums: Regular payments made by the policyholder to the insurance company in exchange for insurance coverage.

Risk: The probability of an adverse event occurring that may lead to an insurance claim.

Claim: A request filed by an insured party to receive compensation for an event covered by the insurance policy.

Statistics: The analysis of numerical data and the application of statistical methods to obtain information about patterns and trends.

Segmentation: The division of a dataset into smaller groups or segments for more detailed analysis.

Adjustment: The modification of insurance reserve estimates based on new data or updated information.

Coverage: The scope and terms of protection provided by an insurance policy.

Fraud: The submission of false or misleading information with the purpose of obtaining undue insurance benefits.

Excess: The amount that an insurance company will not cover and must be assumed by the insured or by another form of insurance.

Estimation: A calculated or projected approximation of a value or quantity, such as the estimation of claim reserves.

Model: A set of algorithms and mathematical rules used to predict or analyze data based on historical patterns.

Experience: The history of claims and losses for an insurance company, used to make future estimates.

Learning: The ability of a computer system to improve its performance through experience and adaptation to new data.

Validation: The process of confirming the accuracy and effectiveness of a model or estimation method by comparing it with real data.

Regulations: Government laws and regulations governing the insurance industry and establishing standards for conduct and practices.

Reserve: The amount of money that an insurance company sets aside to cover future claims and obligations.

Claim: An incident or adverse event that results in an insurance claim.

History: A record of past events and related data, such as the claims history of an insured party.

Loss: The amount of money that an insurance company pays as a result of a claim made by an insured party.

Cost/Benefit Analysis

- Costs: Estimated at $USD 116,500, including data collection, results deployment, operational costs, and labor costs.

- Benefits: Improved reserve calculation, enhanced data organization, new insights, data-driven decision-making, competitive advantage, customer satisfaction, regulatory compliance, innovation.

Project Plan

- Understanding Business: 1 week
- Understanding Data: 1 week
- Data Preparation: 2 weeks
- Modeling: 3 weeks
- Evaluation: 1 week
- Deployment: 1 week

Tools and Techniques Evaluation

Python programming language using Jupyter Notebooks is chosen for its readability, simplicity, community support, multi-platform capability, extensive libraries, and ecosystem, machine learning and data science capabilities, web development, integration, automation, and active community, all of which contribute to the success of the data mining project.

Data understanding

The main aim of Initial Data Analysis (IDA) is to execute accurate analysis of available database. For accomplishing this, we must analyze some graphs and measures about market data displayed, statistical components and focus on some attributes of the sample that could impact the results of the study.

Metadata setup: here are presented the data displayed in auto insurance archives:

- GRCODE NAIC company code (including insurer groups and single insurers)
- GRNAME NAIC company name (including insurer groups and single insurers)
- AccidentYear Accident year(1988 to 1997)
- DevelopmentYear Development year (1988 to 1997)
- DevelopmentLag Development year (AY-1987 + DY-1987 - 1)
- IncurLoss_ Incurred losses and allocated expenses reported at year end
- CumPaidLoss_ Cumulative paid losses and allocated expenses at year end
- BulkLoss_ Bulk and IBNR reserves on net losses and defense and cost containment expenses reported at year end
- PostedReserve97_ Posted reserves in year 1997 taken from the Underwriting and

Investment Exhibit – Part 2A, including net losses unpaid and unpaid loss adjustment expenses

- EarnedPremDIR_ Premiums earned at incurral year - direct and assumed
- EarnedPremCeded_ Premiums earned at incurral year - ceded
- EarnedPremNet_ Premiums earned at incurral year - net

Initial Data Analysis

We are going to explore the data of auto insurance for ten years of develpment, in brief, we're seekeing to check data quality, consistency and fit to the pusrposes of this analysis.

Firstly, after loading the data, our goal is to check if the data base fits with consistency requirements on industry level.

Secondly, we're checking for data quality, in this sense, our goal is to detect if any null or missing data could affect the results.

Lastly, we're exploring on how to solve the data issues arisen from last steps.

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import chainladder as cl
import os
```

In [ ]:

```
df = pd.read_csv(os.path.normpath(os.getcwd() + os.sep + os.pardir)+"/data/pp
```

In [ ]:

```
df.head(5)
```

Out[ ]:

	GRCODE	GRNAME	AccidentYear	DevelopmentYear	DevelopmentLag	IncurLoss_B	CumPai
0	43	IDS Property Cas Ins Co	1988	1988	1	607	
1	43	IDS Property Cas Ins Co	1988	1989	2	647	
2	43	IDS Property Cas Ins Co	1988	1990	3	582	
3	43	IDS Property Cas Ins Co	1988	1991	4	598	
4	43	IDS Property Cas Ins Co	1988	1992	5	614	

In [ ]:

```
df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14600 entries, 0 to 14599
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   GRCODE              14600 non-null  int64 
 1   GRNAME              14600 non-null  object
 2   AccidentYear        14600 non-null  int64 
 3   DevelopmentYear     14600 non-null  int64 
 4   DevelopmentLag      14600 non-null  int64 
 5   IncurLoss_B         14600 non-null  int64 
 6   CumPaidLoss_B       14600 non-null  int64 
 7   BulkLoss_B          14600 non-null  int64 
 8   EarnedPremDIR_B     14600 non-null  int64 
 9   EarnedPremCeded_B   14600 non-null  int64 
 10  EarnedPremNet_B     14600 non-null  int64 
 11  Single              14600 non-null  int64 
 12  PostedReserve97_B   14600 non-null  int64 
dtypes: int64(12), object(1)
memory usage: 1.4+ MB
```

In [ ]:

```
df.dtypes
```

Out[ ]:
```
GRCODE                  int64
GRNAME                 object
AccidentYear            int64
DevelopmentYear         int64
DevelopmentLag          int64
IncurLoss_B             int64
CumPaidLoss_B           int64
BulkLoss_B              int64
EarnedPremDIR_B         int64
EarnedPremCeded_B       int64
EarnedPremNet_B         int64
Single                  int64
PostedReserve97_B       int64
dtype: object
```

It´s confirmed that the database has 100 columns by every insurance company sampled, that fact makes sense as the study has 10 years span and covers 10 lags of products incidence.

In [ ]:
```
df.groupby("GRCODE").size().to_frame().reset_index()
```

Out[ ]:

	GRCODE	0
0	43	100
1	266	100
2	353	100
3	388	100
4	460	100
...	...	...
141	42552	100
142	42749	100
143	42846	100
144	43354	100
145	43494	100

146 rows × 2 columns

In [ ]:
```
df.describe()
```

Out[ ]:

	GRCODE	AccidentYear	DevelopmentYear	DevelopmentLag	IncurLoss_B	CumP
count	14600.000000	14600.00000	14600.000000	14600.00000	1.460000e+04	1.46
mean	18162.013699	1992.50000	1997.000000	5.50000	8.351644e+04	7.2(
std	12698.810114	2.87238	4.062158	2.87238	7.806727e+05	6.9'
min	43.000000	1988.00000	1988.000000	1.00000	-8.000000e+00	-5.9(
25%	9466.000000	1990.00000	1994.000000	3.00000	1.420000e+02	1.09
50%	14954.500000	1992.50000	1997.000000	5.50000	1.940000e+03	1.65
75%	27766.000000	1995.00000	2000.000000	8.00000	8.767250e+03	7.31
max	43494.000000	1997.00000	2006.000000	10.00000	1.169300e+07	1.0

It´s reasonable to validate that the premiums that are ceded and the Direct Incurred Remuneration (DIR) are equal to the net premium. This is correctly reflected by the dataset.

In [ ]:

```
df[df.EarnedPremDIR_B+df.EarnedPremCeded_B == df.EarnedPremNet_B]
```

Out[ ]:

	GRCODE	GRNAME	AccidentYear	DevelopmentYear	DevelopmentLag	IncurLoss_B
100	266	Public Underwriters Grp	1988	1988	1	63
101	266	Public Underwriters Grp	1988	1989	2	172
102	266	Public Underwriters Grp	1988	1990	3	156
103	266	Public Underwriters Grp	1988	1991	4	149
104	266	Public Underwriters Grp	1988	1992	5	148
...	...	...	...	...	...	...
14495	43354	San Antonio Reins Co	1997	2002	6	0
14496	43354	San Antonio Reins Co	1997	2003	7	0
14497	43354	San Antonio Reins Co	1997	2004	8	0
14498	43354	San Antonio Reins Co	1997	2005	9	0
14499	43354	San Antonio Reins Co	1997	2006	10	0

4020 rows × 13 columns

From an illustrative perspective, the triangles are constructed from long form by pivoting the table on index varaibles, in this case we're taking as index the name of each insurance company and the year in which the losses were incurred by accidents in an aggregate manner.

In [ ]:

```
df_triangles = df.pivot_table(index=['GRNAME', 'AccidentYear'], columns='Deve
```

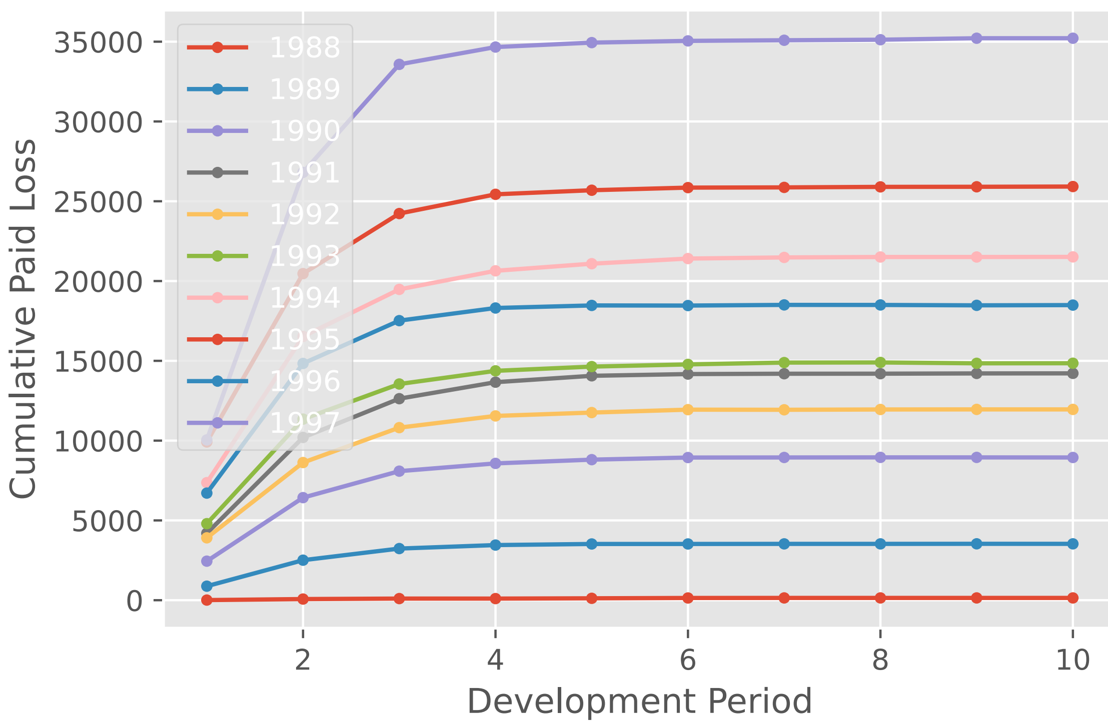
In order to see the behaviour of the cummulative losses, we see next the development on ten years of auto insurance policies for an specific company. It can be showed that the link ratio (the rate of change between losses of one year to another) keep stable after certain time. This functions are increasing but their increments are marginally decreasing. At industry level this means that the policies reached sufficient development after certain year.

In [ ]:
```
plt.style.use('ggplot')
plt.figure()

#for i in [df.GRNAME.unique()]:
#    df_triangles.loc[i].T.plot(label = i,marker='.')

df_triangles.loc["Public Underwriters Grp"].T.plot(label = 43,marker='.')
plt.ylabel('Cumulative Paid Loss')
plt.xlabel('Development Period')
#set_ylim(0, 150000)
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7fb704e5d940>
<Figure size 432x288 with 0 Axes>
```



In [ ]:
```
df_triangles
```

Out[ ]:

GRNAME	AccidentYear \ DevelopmentLag	1	2	3	4	5	6	7	8	9	10
Adriatic Ins Co	**1988**	14	17	17	30	30	30	30	30	30	30
	1989	2	3	3	3	3	3	3	3	3	3
	1990	0	0	0	0	0	0	0	0	0	0
	1991	0	0	0	0	0	0	0	0	0	0
	1992	0	0	0	0	0	0	0	0	0	0
...	**...**	...	...	...	...	...	...	...	...	...	...
Yasuda Fire & Marine Ins Co Of Amer	**1993**	227	497	915	877	1001	1024	1120	1122	1087	1087
	1994	289	659	747	804	817	824	824	824	824	824
	1995	284	560	745	804	893	894	897	897	897	897
	1996	279	523	735	736	749	783	783	783	783	783
	1997	263	429	524	560	560	560	559	559	559	559

1460 rows × 10 columns

In [ ]:

```
auto_triangles = cl.Triangle(df,
                             index = 'GRNAME',
                             origin="AccidentYear",
                             development="DevelopmentYear",
                             columns=["IncurLoss_B",
                                      "CumPaidLoss_B",
                                      "BulkLoss_B",
                                      "EarnedPremDIR_B",
                                      "EarnedPremCeded_B",
                                      "EarnedPremNet_B"],
                             cumulative=True)
```

In [ ]:

```
auto_triangles
```

Out[ ]:

	Triangle Summary
Valuation:	2006-12
Grain:	OYDY
Shape:	(146, 6, 19, 19)
Index:	[GRNAME]
Columns:	[IncurLoss_B, CumPaidLoss_B, BulkLoss_B, EarnedPremDIR_B, EarnedPremCeded_B, EarnedPremNet_B]

In [ ]:
```
auto_companies = auto_triangles.index
auto_triangles.index
```

Out[ ]:

	GRNAME
0	Adriatic Ins Co
1	Aegis Grp
2	Agency Ins Co Of MD Inc
3	Agway Ins Co
4	All Nation Ins Co
...	...
141	West Bend Mut Ins Grp
142	Wisconsin American Mut Ins Co
143	Wisconsin Mut Ins Co
144	Wolverine Mut Ins Co
145	Yasuda Fire & Marine Ins Co Of Amer

146 rows × 1 columns

In [ ]:
```
auto_triangles.loc["Adriatic Ins Co"]["CumPaidLoss_B"]
```

Out[ ]:

	12	24	36	48	60	72	84	96	108	120	132	144	156	168
1988	14.00	17.00	17.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00				
1989	2.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00				
1990														
1991														
1992														
1993														
1994														
1995														
1996														
1997														
1998														
1999														
2000														
2001														
2002														
2003														
2004														
2005														
2006														

From above information, we could stablish that there are missing some year development information. This is explained by the fact that some companies could not have some information on certain years for auto policies, as the product could be removed or opened on a different timespan than 1988 to 1997.

As our goal is to explore the deveolpment of losses in this specific timespan, we need to limit the data to this dates. In this sense, the next steps take on this and the data is censored up to 1997.

In [ ]:

```
df98 = df[df.DevelopmentYear <= 1997]
```

In [ ]:

```
df98.DevelopmentYear.unique()
```

Out[ ]:
```
array([1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997])
```

After limiting data, we're studying consistency of losses and their main indicators. For this purpose is useful to use the chainladder (cl) library on python, as it facilitates data triangles trasforming.

In [ ]:
```
auto_triangles = cl.Triangle(df98,
                             index = 'GRNAME',
                             origin="AccidentYear",
                             development="DevelopmentYear",
                             columns=["IncurLoss_B",
                                      "CumPaidLoss_B",
                                      "BulkLoss_B",
                                      "EarnedPremDIR_B",
                                      "EarnedPremCeded_B",
                                      "EarnedPremNet_B"],
                             cumulative=True)
auto_triangles.describe()
```

Out[ ]:

	development	IncurLoss_B	CumPaidLoss_B	BulkLoss_B	EarnedPremDIR_B	Earne
count	6381.000000	6.215000e+03	6.147000e+03	4.388000e+03	6.357000e+03	
mean	47.187588	1.014576e+05	7.836593e+04	1.666923e+04	1.241686e+05	
std	29.230155	8.412121e+05	6.667223e+05	1.781883e+05	1.022347e+06	
min	12.000000	-1.000000e+00	-5.900000e+01	-7.500000e+02	-1.000000e+01	
25%	24.000000	7.605000e+02	6.140000e+02	1.700000e+01	1.497000e+03	
50%	36.000000	3.290000e+03	2.579000e+03	1.250000e+02	6.282000e+03	
75%	72.000000	1.062700e+04	8.011000e+03	6.640000e+02	1.670500e+04	
max	120.000000	1.169300e+07	9.640098e+06	3.830524e+06	1.506571e+07	

In [ ]:
```
auto_triangles
```

Out[ ]:

	Triangle Summary
Valuation:	1997-12
Grain:	OYDY
Shape:	(146, 6, 10, 10)
Index:	[GRNAME]
Columns:	[IncurLoss_B, CumPaidLoss_B, BulkLoss_B, EarnedPremDIR_B, EarnedPremCeded_B, EarnedPremNet_B]

For this study, we're interested on losses development, so we're going to take this variable from available data.

In [ ]:
```
auto_triangles.loc["All Nation Ins Co"]["CumPaidLoss_B"]
```

Out[ ]:

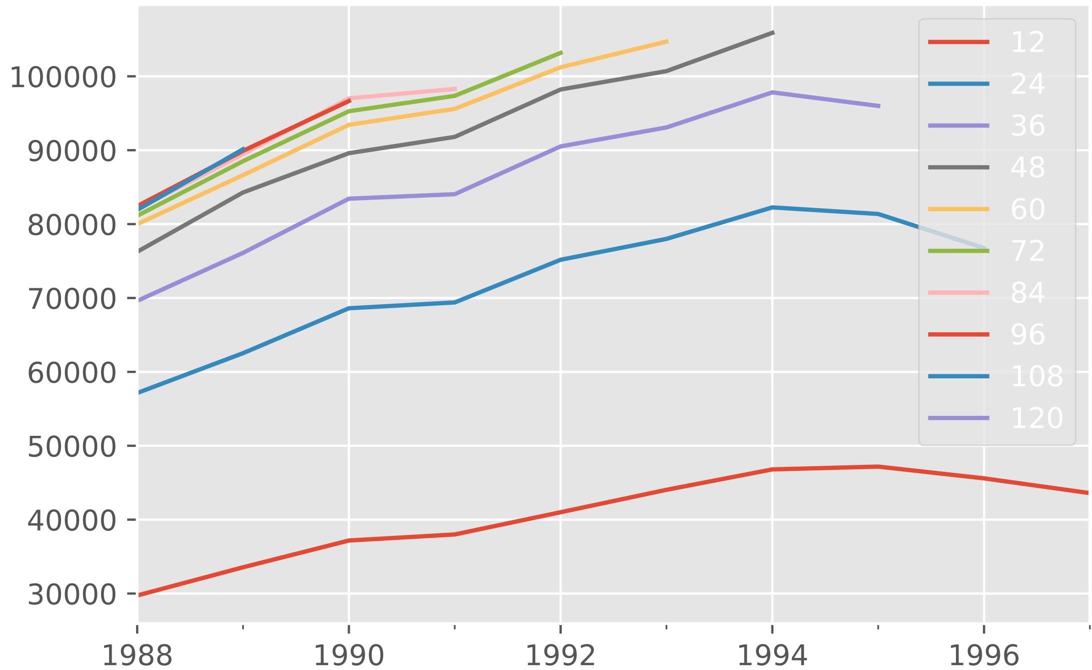
	12	24	36	48	60	72	84	96	108	120
1988	2,430	4,795	5,870	6,520	6,765	6,853	6,891	6,945	6,949	6,950
1989	2,579	4,896	6,024	6,653	6,850	6,885	6,996	7,013	7,014	
1990	2,824	5,180	6,412	6,872	7,037	7,113	7,134	7,142		
1991	2,718	5,216	6,289	6,797	6,912	6,955	6,991			
1992	3,189	6,680	8,288	8,973	9,240	9,350				
1993	4,365	7,228	8,840	9,677	9,851					
1994	3,311	6,588	8,034	8,716						
1995	4,595	8,174	9,345							
1996	3,448	4,627								
1997	218									

In [ ]:
```
auto_triangles_filt = auto_triangles["IncurLoss_B", "CumPaidLoss_B"]
```

In [ ]:
```
auto_triangles_filt["CumPaidLoss_B"].mean(axis=0).mean(axis=1).plot()
```

```
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/chainladder/core/pandas.py:364: RuntimeWarning: Mean of empty slice
  obj.values = func(obj.values, axis=axis, *args, **kwargs)
```

Out[ ]: <AxesSubplot:>



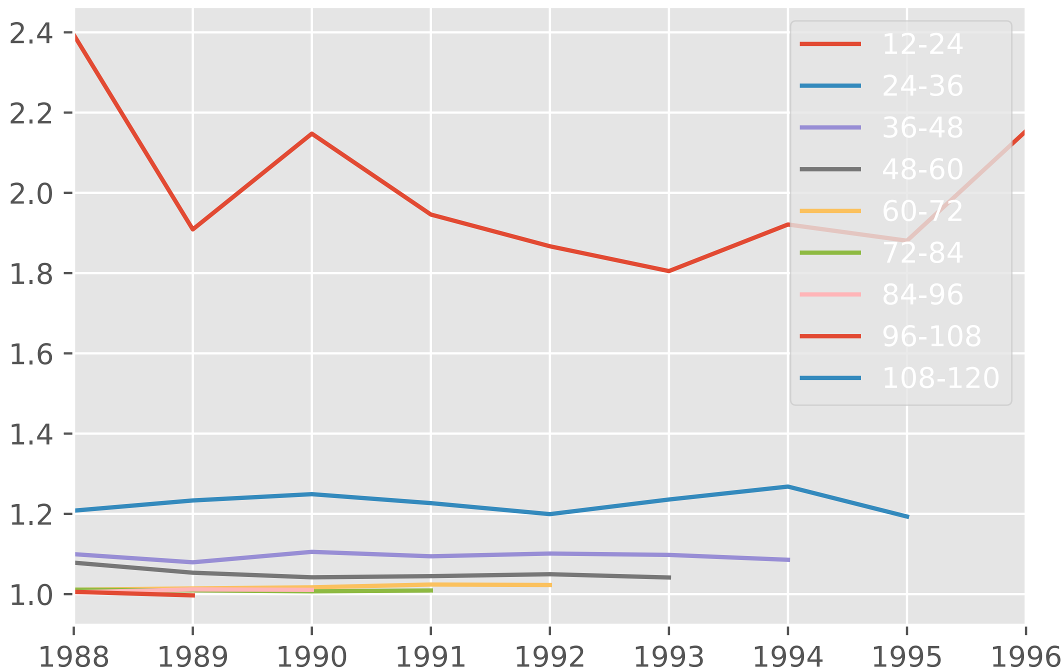
From graph above, we can see that the cummulative losses mean across sample is as expected, increasing with respect to develompent year.

In [ ]:
```
triangle_link = auto_triangles_filt.link_ratio
```

In [ ]:
```
auto_triangles_filt.link_ratio["CumPaidLoss_B"].mean(axis=0).mean(axis=1).plo
```

```
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/chainladder/core/pandas.py:364: RuntimeWarning: Mean of empty slice
  obj.values = func(obj.values, axis=axis, *args, **kwargs)
```

Out[ ]: <AxesSubplot:>



Furthermore, from graph of the mean of link ratio, we can see that there is significant stability on development rates from one year to another, there are some peaks for 12 to 24 months policies, but expected as full development of policies have taken place for the longest term considered and the most policies could have been claimed.

In [ ]:
```
triangle_link["CumPaidLoss_B"].iloc[3].heatmap()
```

Out[ ]:

	12-24	24-36	36-48	48-60	60-72	72-84	84-96	96-108	108-120
1988	1.7903	1.2220	1.0944	1.0479	1.0307	1.0056	1.0049	1.0003	1.0000
1989	2.0582	1.1708	1.0554	1.0563	1.0233	1.0011	1.0009	1.0004	
1990	2.2025	1.1739	1.0752	1.0450	1.0439	1.0145	1.0006		
1991	1.8214	1.2122	1.1054	1.0196	1.0178	1.0095			
1992	1.8192	1.2375	1.0983	1.0577	1.0080				
1993	2.0446	1.2109	1.1295	1.0552					
1994	1.8205	1.2816	1.1203						
1995	1.8281	1.1088							
1996	1.8173								

We can see, for an example, that there is no apparent pattern on link ratios distribution, as is expected.

In [ ]:
```
#triangle_mean_inc = auto_triangles_filt["IncurLoss_B"].mean(0).mean(1)
triangle_mean_paid = auto_triangles_filt["CumPaidLoss_B"].mean(0).mean(1)
```

```
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/chainladder/core/pandas.py:364: RuntimeWarning: Mean of empty slice
  obj.values = func(obj.values, axis=axis, *args, **kwargs)
```

In [ ]:
```
triangle_mean_paid
```

Out[ ]:

	12	24	36	48	60	72	84	96	108	120
1988	29,739	57,141	69,607	76,251	79,988	81,126	82,003	82,433	81,910	81,981
1989	33,554	62,532	76,107	84,283	86,622	88,513	89,477	89,913	90,126	
1990	37,176	68,608	83,440	89,597	93,434	95,274	97,018	96,652		
1991	37,996	69,397	84,047	91,802	95,595	97,359	98,290			
1992	41,005	75,171	90,515	98,216	101,221	103,171				
1993	44,038	77,998	93,079	100,703	104,699					
1994	46,801	82,258	97,827	105,886						
1995	47,180	81,372	95,994							
1996	45,589	76,751								
1997	43,593									

In [ ]:
```
triangles_demean = auto_triangles_filt - auto_triangles_filt.mean(0).mean(1)
```

```
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/chainladder/core/pandas.py:364: RuntimeWarning: Mean of empty slice
  obj.values = func(obj.values, axis=axis, *args, **kwargs)
```

In [ ]:
```
#triangles_sd_inc = auto_triangles_filt["IncurLoss_B"].std(0)
triangles_sd_paid = auto_triangles_filt["CumPaidLoss_B"].std(0)
```

```
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/numpy/lib/nanfunctions.py:1879: RuntimeWarning: Degrees of freedom <= 0 for slice.
  var = nanvar(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
```

In [ ]:
```
(triangles_sd_paid/triangle_mean_paid).heatmap()
```

Out[ ]:

	12	24	36	48	60	72	84	96	108	120
1988	8.0406	8.1016	8.0431	8.0299	8.0033	8.0449	8.0458	8.0476	8.0865	8.0875
1989	8.1803	8.2593	8.2165	8.1485	8.2142	8.2102	8.2066	8.2071	8.2078	
1990	8.2429	8.2568	8.2050	8.2491	8.2385	8.2339	8.1950	8.2284		
1991	8.1209	8.1239	8.0814	8.0538	8.0343	8.0304	8.0278			
1992	8.2513	8.1931	8.1634	8.1449	8.1608	8.1533				
1993	8.2972	8.3520	8.3163	8.2949	8.2749					
1994	8.4848	8.4495	8.3887	8.3725						
1995	8.5994	8.5143	8.4833							
1996	8.6393	8.6128								
1997	8.6975									

From an statistic perspective, we have to take care on variation coefficient. We estimated that for the sample on accident and development year, the mean could not be representative of this data, as the variation coefficient (standard deviation over mean) is significantly high. We conclude that, even the data is consistent on an industry level, the sample is sparse. This could be explained by the fact that the sample is very broad and we're taking into account some large and small insurers.

Business understanding

In order to establish some important assumptions and aligning study objectives with data, it's important to clarify the business dynamics. For that, we're going to deep on market measures.

- Monopolization: as we can see below, analyzing data from 1997, it's observed that this is a concentrated market as State Farm Mutual Group takes over 75% of premiums and has 76% of claims. Furthermore, along with United Services Automobile Association, both firms take over 86% of auto insurance market premiums and 86% of claims. If our goal is to predict the losses behaviour we must take into account that some companies could add noise to conclussions by their size.

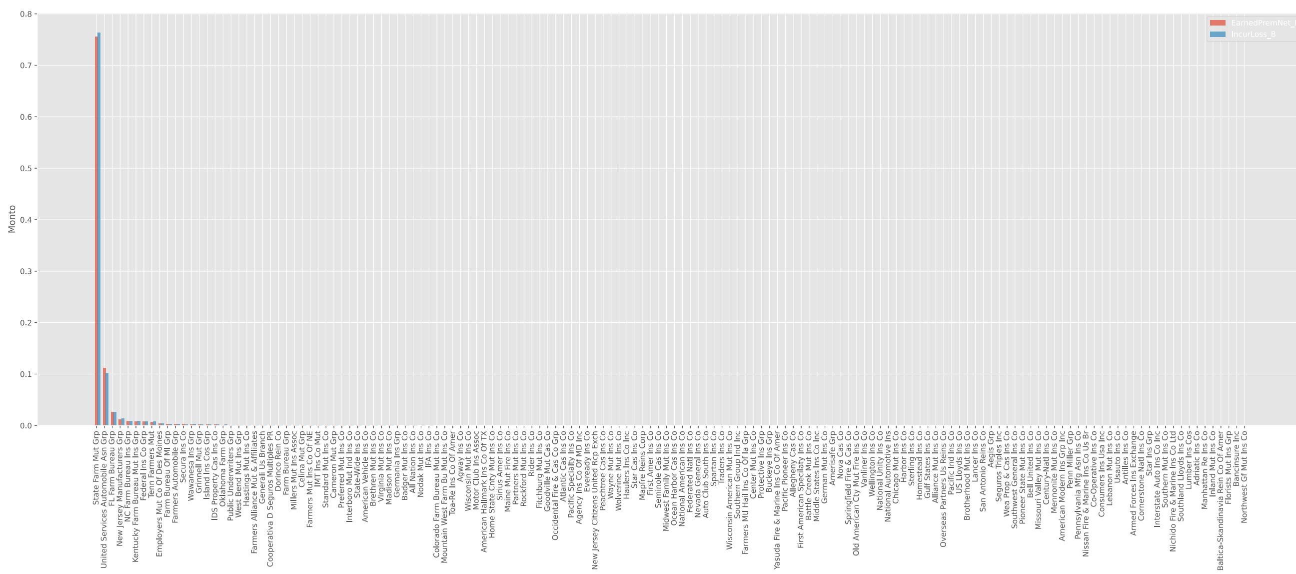
In [ ]:

```
df_business = df[(df['DevelopmentYear']==1997)].copy()
df_business = df_business[['GRNAME', 'AccidentYear', 'DevelopmentYear', 'Incu
grouped_data = df_business.groupby('GRNAME').agg({'IncurLoss_B': 'sum', 'Earn

# Sort by 'EarnedPremNet_C'
grouped_data_sorted = grouped_data.sort_values(by='EarnedPremNet_B', ascendin
plt.figure(figsize=(30, 10))

bar_width = 0.35
index_earned = range(len(grouped_data_sorted['GRNAME']))
index_incur = [i + bar_width for i in index_earned]
sum_prem = np.sum(grouped_data_sorted['EarnedPremNet_B'])
sum_loss = np.sum(grouped_data_sorted['IncurLoss_B'])

plt.bar(index_earned, grouped_data_sorted['EarnedPremNet_B']/sum_prem, bar_wi
plt.bar(index_incur, grouped_data_sorted['IncurLoss_B']/sum_loss, bar_width,
plt.ylabel('Monto')
plt.title('Premium and inurred losses')
plt.xticks(index_earned, grouped_data_sorted['GRNAME'], rotation='vertical')
plt.legend()
plt.show()
```



- Profitability: from a business perspective, we must to analyze if auto insurance agents are taking yields on their operations. The returns in insurance are critically dependent on the estimations of losses due to claims, ie, the risks analytics are an important step on budgeting and defining strategies.

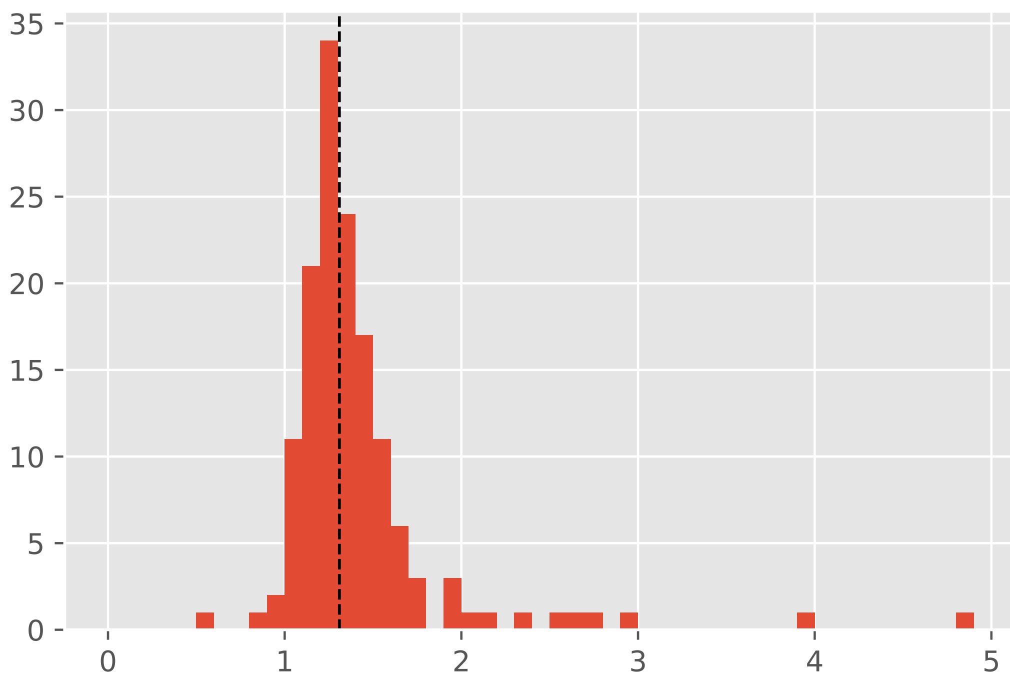
For 1997, the companies that belongs to this group showed that median profits are positive up to 31%, we can conclude that business is worthy for stakeholders as the distribution of returns is rightly assimetric. The utilities that this income/costs ratio could bring to company owners also depends on operational efficiency, and this benchmark of 31% indicates that the could set that margin from which they could obtain benefits.

In [ ]:

```
Profit_rate = grouped_data_sorted.EarnedPremNet_B/grouped_data_sorted.IncurLo
Profit_rate.drop_duplicates(inplace=True)
del Profit_rate[16]
Profit_rate.hist(bins = np.arange(start=0, step = 0.1, stop = 5), )
plt.axvline(Profit_rate.median(), color='k', linestyle='dashed', linewidth=1)
```

Out[ ]:

```
<matplotlib.lines.Line2D at 0x7fb6f784ef70>
```



In [ ]:

```
Profit_rate.median()
```

Out[ ]:

```
1.3096171435924726
```

Zeros treatment

In order to adress on missing data and homogeneity issues, in next steps we're cleaning dataset from companies that are incomplete information and didn't met with timespan requirements.

In [ ]:

```
df_triangles_clean = df98.pivot_table(index=['GRNAME', 'AccidentYear'], colum
```

In [ ]:

```
df_triangles_clean.head(40)
```

Out[ ]:

DevelopmentLag	GRNAME	AccidentYear	1	2	3	4	5	6	
0	Adriatic Ins Co	1988	14.0	17.0	17.0	30.0	30.0	30.0	3
1	Adriatic Ins Co	1989	2.0	3.0	3.0	3.0	3.0	3.0	
2	Adriatic Ins Co	1990	0.0	0.0	0.0	0.0	0.0	0.0	
3	Adriatic Ins Co	1991	0.0	0.0	0.0	0.0	0.0	0.0	
4	Adriatic Ins Co	1992	0.0	0.0	0.0	0.0	0.0	0.0	N
5	Adriatic Ins Co	1993	0.0	0.0	0.0	0.0	0.0	NaN	N
6	Adriatic Ins Co	1994	0.0	0.0	0.0	0.0	NaN	NaN	N
7	Adriatic Ins Co	1995	0.0	0.0	0.0	NaN	NaN	NaN	N
8	Adriatic Ins Co	1996	0.0	0.0	NaN	NaN	NaN	NaN	N
9	Adriatic Ins Co	1997	0.0	NaN	NaN	NaN	NaN	NaN	N
10	Aegis Grp	1988	110.0	370.0	491.0	541.0	572.0	586.0	60
11	Aegis Grp	1989	134.0	295.0	361.0	401.0	404.0	420.0	42
12	Aegis Grp	1990	69.0	293.0	366.0	388.0	399.0	408.0	40
13	Aegis	1991	57.0	220.0	240.0	242.0	242.0	242.0	24

	Grp								
14	Aegis Grp	1992	0.0	0.0	0.0	0.0	0.0	0.0	N
15	Aegis Grp	1993	0.0	0.0	0.0	0.0	0.0	NaN	N
16	Aegis Grp	1994	-1.0	0.0	0.0	0.0	NaN	NaN	N
17	Aegis Grp	1995	0.0	0.0	0.0	NaN	NaN	NaN	N
18	Aegis Grp	1996	0.0	0.0	NaN	NaN	NaN	NaN	N
19	Aegis Grp	1997	0.0	NaN	NaN	NaN	NaN	NaN	N
20	Agency Ins Co Of MD Inc	1988	0.0	0.0	0.0	0.0	0.0	0.0	
21	Agency Ins Co Of MD Inc	1989	0.0	0.0	0.0	0.0	0.0	0.0	
22	Agency Ins Co Of MD Inc	1990	33.0	72.0	142.0	144.0	144.0	144.0	14
23	Agency Ins Co Of MD Inc	1991	598.0	1533.0	1814.0	1881.0	1899.0	1904.0	191
24	Agency Ins Co Of MD Inc	1992	3199.0	5919.0	6653.0	6983.0	7106.0	7134.0	N
25	Agency Ins Co Of MD Inc	1993	2175.0	3493.0	3967.0	4152.0	4287.0	NaN	N
26	Agency Ins Co Of MD Inc	1994	1598.0	2680.0	3089.0	3187.0	NaN	NaN	N
27	Agency Ins Co Of MD Inc	1995	2296.0	4054.0	4421.0	NaN	NaN	NaN	N
28	Agency Ins Co Of MD Inc	1996	3118.0	4626.0	NaN	NaN	NaN	NaN	N
29	Agency Ins Co Of MD Inc	1997	2901.0	NaN	NaN	NaN	NaN	NaN	N

30	Agway Ins Co	1988	3286.0	5883.0	7189.0	7868.0	8245.0	8498.0	854
31	Agway Ins Co	1989	2904.0	5977.0	6998.0	7386.0	7802.0	7984.0	799
32	Agway Ins Co	1990	2519.0	5548.0	6513.0	7003.0	7318.0	7639.0	775
33	Agway Ins Co	1991	2755.0	5018.0	6083.0	6724.0	6856.0	6978.0	704
34	Agway Ins Co	1992	2046.0	3722.0	4606.0	5059.0	5351.0	5394.0	N
35	Agway Ins Co	1993	2287.0	4676.0	5662.0	6395.0	6748.0	NaN	N
36	Agway Ins Co	1994	2056.0	3743.0	4797.0	5374.0	NaN	NaN	N
37	Agway Ins Co	1995	1966.0	3594.0	3985.0	NaN	NaN	NaN	N
38	Agway Ins Co	1996	1872.0	3402.0	NaN	NaN	NaN	NaN	N
39	Agway Ins Co	1997	1821.0	NaN	NaN	NaN	NaN	NaN	N

In [ ]:

```
cleaning_condition1 = df_triangles_clean[df_triangles_clean.iloc[:,-10:].sum(
df_triangles_clean = df_triangles_clean[~df_triangles_clean["GRNAME"].isin(cl
```

In [ ]:

```
df_triangles_clean = df_triangles_clean.set_index(["GRNAME","AccidentYear"])
df_triangles_clean
```

Out[ ]:

	DevelopmentLag	1	2	3	4	5	6	7	8	
GRNAME	**AccidentYear**									
Agway Ins Co	**1988**	3286.0	5883.0	7189.0	7868.0	8245.0	8498.0	8546.0	8588.0	8
	1989	2904.0	5977.0	6998.0	7386.0	7802.0	7984.0	7993.0	8000.0	8
	1990	2519.0	5548.0	6513.0	7003.0	7318.0	7639.0	7750.0	7755.0	
	1991	2755.0	5018.0	6083.0	6724.0	6856.0	6978.0	7044.0	NaN	
	1992	2046.0	3722.0	4606.0	5059.0	5351.0	5394.0	NaN	NaN	
...	**...**	...	...	...	...	...	...	...	...	
Yasuda Fire & Marine Ins Co Of Amer	**1993**	227.0	497.0	915.0	877.0	1001.0	NaN	NaN	NaN	
	1994	289.0	659.0	747.0	804.0	NaN	NaN	NaN	NaN	
	1995	284.0	560.0	745.0	NaN	NaN	NaN	NaN	NaN	
	1996	279.0	523.0	NaN	NaN	NaN	NaN	NaN	NaN	
	1997	263.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

950 rows × 10 columns

In [ ]:

```
names_clean = df_triangles_clean.reset_index().GRNAME.unique()
```

In [ ]:

```
temp = df_triangles_clean.reset_index().iloc[: , 2:].to_numpy()

smaller_matrix_size = (10, 10)
smaller_matrices = []

# Iterate through the rows
for i in range(0, temp.shape[0], smaller_matrix_size[0]):
    for j in range(0, temp.shape[1], smaller_matrix_size[1]):
        # Extract a smaller matrix from the larger matrix
        smaller_matrix = temp[i:i+smaller_matrix_size[0], j:j+smaller_matrix_
        smaller_matrices.append(smaller_matrix)
```

In [ ]:

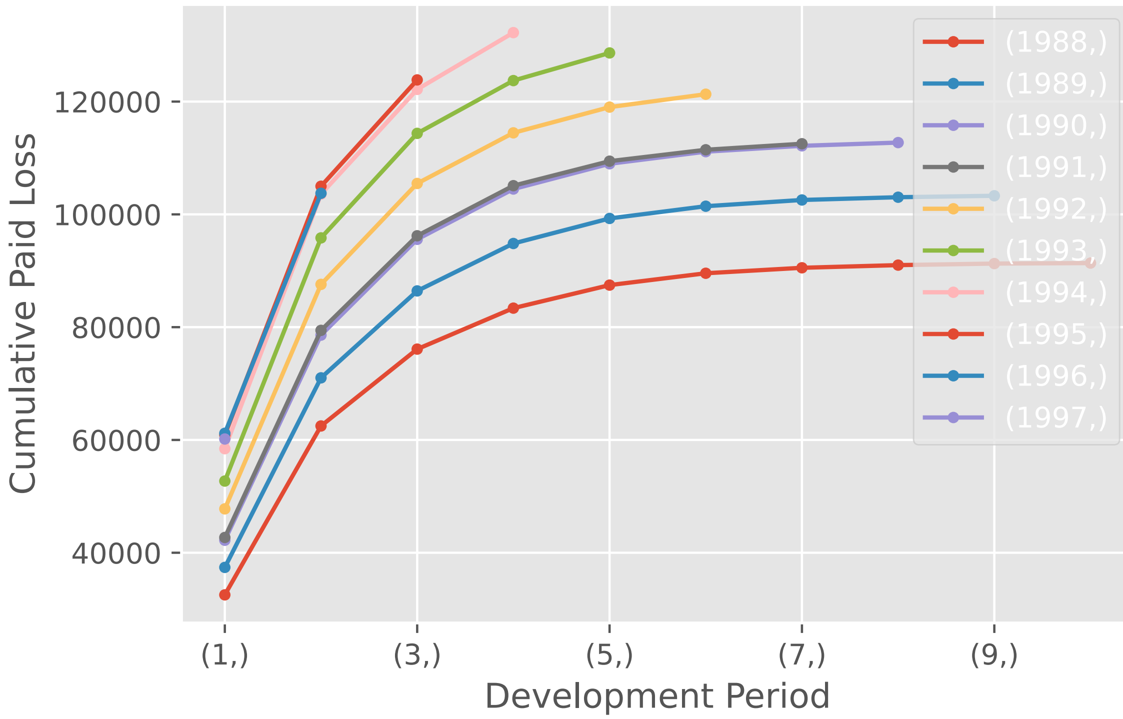
```
# Stack the matrices along a new axis
stacked_matrices = np.stack(smaller_matrices, axis=0)

# Calculate the mean along the new axis
mean = pd.DataFrame(np.mean(stacked_matrices, axis=0), columns = [np.arange(1
sd = pd.DataFrame(np.std(stacked_matrices, axis=0), columns = [np.arange(1,11
cv = sd/mean
```

In [ ]:
```
plt.figure()

pd.DataFrame(mean).T.plot(label = "Mean",marker='.')
plt.ylabel('Cumulative Paid Loss')
plt.xlabel('Development Period')
#set_ylim(0, 150000)
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7fb6efebd9a0>
<Figure size 432x288 with 0 Axes>
```



In [ ]:
```
#sum([d for d in smaller_matrices])
def sum_losses_in_triangles(matrices):
    triangle_sums = []
    for matrix in matrices:
        triangle_sums.append(np.nansum(np.triu(matrix)))
    return triangle_sums

# Sum up losses in each triangle
triangle_sums = sum_losses_in_triangles(smaller_matrices)

# Display the results
for i, triangle_sum in enumerate(triangle_sums):
    print(f"Triangle {i+1} Sum: {triangle_sum}")
```

```
Triangle 1 Sum: 217753.0
Triangle 2 Sum: 201254.0
Triangle 3 Sum: 9795.0
```

```
Triangle 4 Sum: 82247.0
Triangle 5 Sum: 310747.0
Triangle 6 Sum: 28948.0
Triangle 7 Sum: 2415.0
Triangle 8 Sum: 44004.0
Triangle 9 Sum: 182269.0
Triangle 10 Sum: 35397.0
Triangle 11 Sum: 316276.0
Triangle 12 Sum: 14664.0
Triangle 13 Sum: 13538.0
Triangle 14 Sum: 375974.0
Triangle 15 Sum: 367251.0
Triangle 16 Sum: 19527.0
Triangle 17 Sum: 11426.0
Triangle 18 Sum: 163664.0
Triangle 19 Sum: 240510.0
Triangle 20 Sum: 123035.0
Triangle 21 Sum: 1281652.0
Triangle 22 Sum: 90919.0
Triangle 23 Sum: 6811417.0
Triangle 24 Sum: 236309.0
Triangle 25 Sum: 661265.0
Triangle 26 Sum: 377297.0
Triangle 27 Sum: 739036.0
Triangle 28 Sum: 30523.0
Triangle 29 Sum: 270729.0
Triangle 30 Sum: 1832661.0
Triangle 31 Sum: 50381.0
Triangle 32 Sum: 25273.0
Triangle 33 Sum: 120119.0
Triangle 34 Sum: 10808.0
Triangle 35 Sum: 100993.0
Triangle 36 Sum: 99241.0
Triangle 37 Sum: 522032.0
Triangle 38 Sum: 12724.0
Triangle 39 Sum: 249954.0
Triangle 40 Sum: 72130.0
Triangle 41 Sum: 934.0
Triangle 42 Sum: 200301.0
Triangle 43 Sum: 57495.0
Triangle 44 Sum: 363627.0
Triangle 45 Sum: 208634.0
Triangle 46 Sum: 645782.0
Triangle 47 Sum: 2241280.0
Triangle 48 Sum: 3036.0
Triangle 49 Sum: 247427.0
Triangle 50 Sum: 71259.0
Triangle 51 Sum: 53909.0
Triangle 52 Sum: 63367.0
Triangle 53 Sum: 397338.0
Triangle 54 Sum: 4792.0
Triangle 55 Sum: 132584.0
Triangle 56 Sum: 118349.0
Triangle 57 Sum: 2307469.0
Triangle 58 Sum: 13162.0
Triangle 59 Sum: 15340.0
Triangle 60 Sum: 2385206.0
```

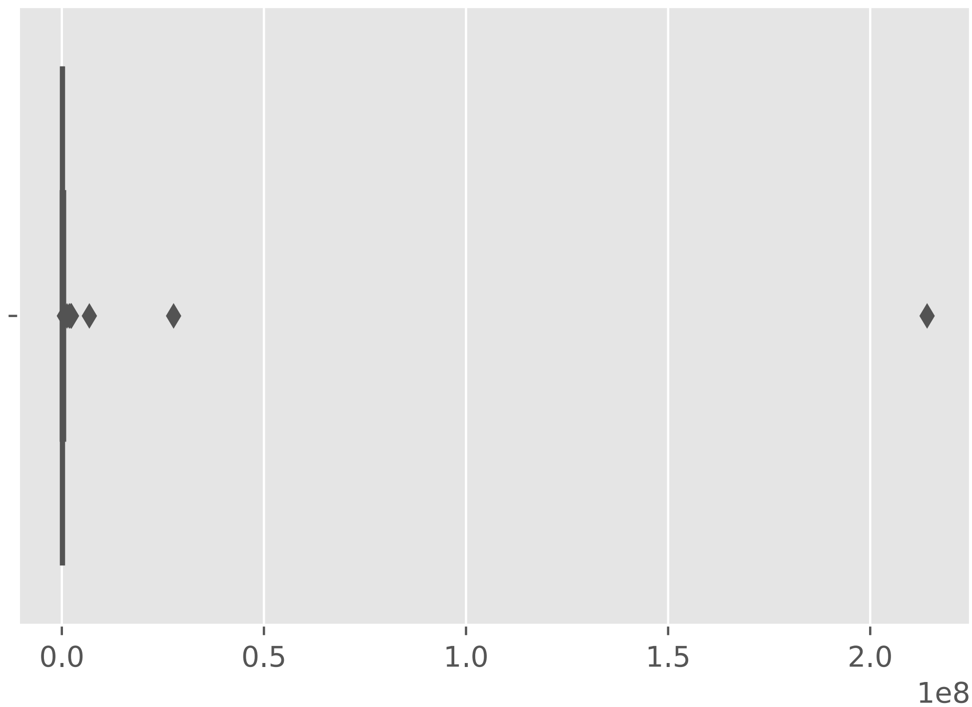
```
Triangle 61 Sum: 6560.0
Triangle 62 Sum: 153291.0
Triangle 63 Sum: 17916.0
Triangle 64 Sum: 62043.0
Triangle 65 Sum: 283740.0
Triangle 66 Sum: 28578.0
Triangle 67 Sum: 77881.0
Triangle 68 Sum: 103211.0
Triangle 69 Sum: 143282.0
Triangle 70 Sum: 160045.0
Triangle 71 Sum: 49355.0
Triangle 72 Sum: 100324.0
Triangle 73 Sum: 802075.0
Triangle 74 Sum: 4057.0
Triangle 75 Sum: 111901.0
Triangle 76 Sum: 6099.0
Triangle 77 Sum: 50561.0
Triangle 78 Sum: 7836.0
Triangle 79 Sum: 239321.0
Triangle 80 Sum: 73136.0
Triangle 81 Sum: 214075877.0
Triangle 82 Sum: 189394.0
Triangle 83 Sum: 1873182.0
Triangle 84 Sum: 84443.0
Triangle 85 Sum: 58033.0
Triangle 86 Sum: 9433.0
Triangle 87 Sum: 27651014.0
Triangle 88 Sum: 213071.0
Triangle 89 Sum: 553778.0
Triangle 90 Sum: 75603.0
Triangle 91 Sum: 346244.0
Triangle 92 Sum: 28856.0
Triangle 93 Sum: 120410.0
Triangle 94 Sum: 50197.0
Triangle 95 Sum: 41967.0
```

In [ ]:

```
sns.boxplot(triangle_sums)
```

```
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/seaborn/
_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg
: x. From version 0.12, the only valid positional argument will be `data`, and
passing other arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

Out[ ]: <AxesSubplot:>



In [ ]:

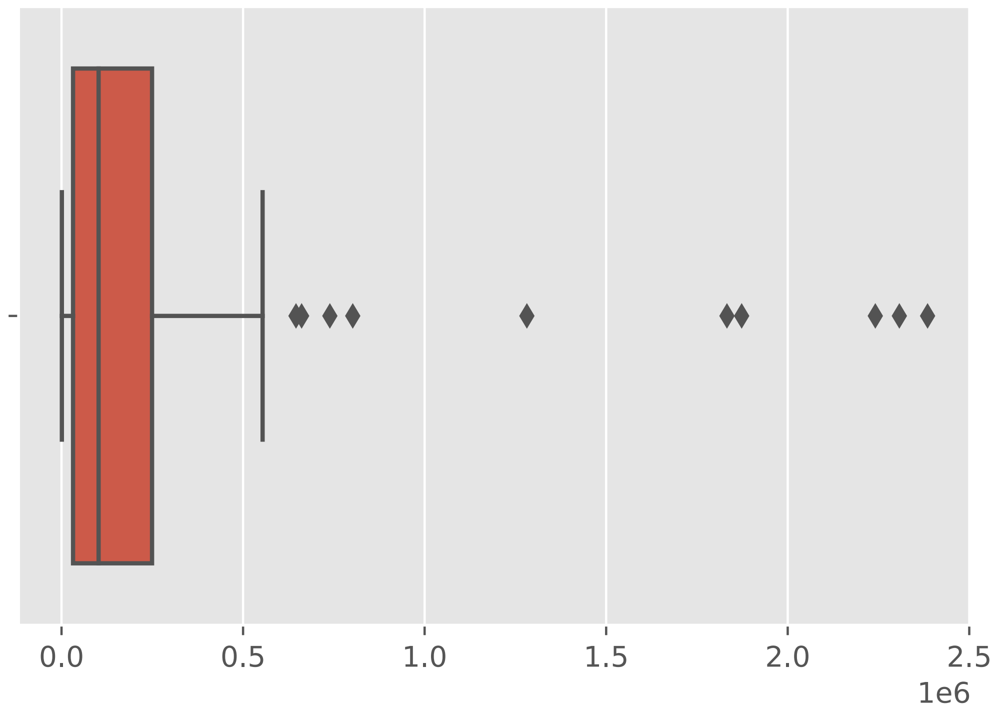
```
max_1 = triangle_sums.index(max(triangle_sums))
del triangle_sums[80]
max_2 = triangle_sums.index(max(triangle_sums))
del triangle_sums[85]
max_3 = triangle_sums.index(max(triangle_sums))
del triangle_sums[84]
max_4 = triangle_sums.index(max(triangle_sums))
del triangle_sums[82]
max_5 = triangle_sums.index(max(triangle_sums))
del triangle_sums[22]
```

In [ ]:

```
sns.boxplot(triangle_sums)
```

```
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/seaborn/
_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg
: x. From version 0.12, the only valid positional argument will be `data`, and
passing other arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

Out[ ]: <AxesSubplot:>



In [ ]:
```
removals = [80, 86, 85, 83, 22]
names_clean[[removals]]
```

```
/var/folders/s1/5zdy54cn7xv66qmfw8ds98yw0000gn/T/ipykernel_14169/462374934.py:
2: FutureWarning: Using a non-tuple sequence for multidimensional indexing is
deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this wi
ll be interpreted as an array index, `arr[np.array(seq)]`, which will result e
ither in an error or a different result.
  names_clean[[removals]]
```

Out[ ]:
```
array(['State Farm Mut Grp', 'United Services Automobile Asn Grp',
       'US Lloyds Ins Co', 'Toa-Re Ins Co Of Amer', 'FL Farm Bureau Grp'],
      dtype=object)
```

In [ ]:
```
names_clean = [e for e in names_clean if e not in removals]
```

In [ ]:
```
df_triangles_clean = df_triangles_clean.reset_index().drop(index=removals).se
```

In [ ]:

```
df_triangles_clean
```

Out[ ]:

GRNAME	AccidentYear \ DevelopmentLag	1	2	3	4	5	6	7	8
Agway Ins Co	1988	3286.0	5883.0	7189.0	7868.0	8245.0	8498.0	8546.0	8588.0
	1989	2904.0	5977.0	6998.0	7386.0	7802.0	7984.0	7993.0	8000.0
	1990	2519.0	5548.0	6513.0	7003.0	7318.0	7639.0	7750.0	7755.0
	1991	2755.0	5018.0	6083.0	6724.0	6856.0	6978.0	7044.0	NaN
	1992	2046.0	3722.0	4606.0	5059.0	5351.0	5394.0	NaN	NaN
...	...	...	...	...	...	...	...	...	...
Yasuda Fire & Marine Ins Co Of Amer	1993	227.0	497.0	915.0	877.0	1001.0	NaN	NaN	NaN
	1994	289.0	659.0	747.0	804.0	NaN	NaN	NaN	NaN
	1995	284.0	560.0	745.0	NaN	NaN	NaN	NaN	NaN
	1996	279.0	523.0	NaN	NaN	NaN	NaN	NaN	NaN
	1997	263.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN

945 rows × 10 columns

From above procedure, we droped the largest companies in order to homogeneize the losses, this allows us to get better estimations on mean based procedures across sample. We decided to sacrifice data form our sample to better understand the average behaviour of the auto insurance market from the study years.

References

- Balona, C., and Richman, R. (2020). The Actuary and IBNR Techniques: A Machine Learning Approach. Available at SSRN 3697256.
- Bogaardt, J., Hsu, K., et. al. (2022) Chainladder Python Package - cl. Retrieved from https://chainladder-python.readthedocs.io/en/latest
- Meyers, G. (2015). Stochastic loss reserving using Bayesian MCMC models. Arlington: Casualty Actuarial Society. Retrieved from: https://www.casact.org/publications-research/research/research-resources/loss-reserving-data-pulled-naic-schedule-p

Chain ladder method

We're going to implement the most used methodology for estimating the projected losses using data from run-off triangles.

The procedure consists on finding some factors by development year from which we could estimate the expected loss on a certein year by their next development. This is described as follows:

$$C_{i,j+1} = f_j \times C_{i,j}.$$

Where $C_{i,j}$ represents every cell of triangles. Then, the development factor tells you how the cumulative amount in development year j grows to the cumulative amount in year $j+1$. We need to find the $\hat{f}_j^{CL}$ estimated for each development year:

$$\begin{aligned} \hat{C}_{ij}^{CL} &= C_{i,I-i} \cdot \prod_{l=I-i}^{j-1} \hat{f}_l^{CL} \\ \hat{f}_j^{CL} &= \frac{\sum_{i=1}^{I-j-1} C_{i,j+1}}{\sum_{i=1}^{I-j-1} C_{ij}}, \end{aligned}$$

Where $C_{i,I-i}$ is on the last observed diagonal. It is clear that an important assumption of the chain-ladder method is that the proportional developments of claims from one development period to the next are similar for all occurrence years.

Objective

To ensure that sufficient funds are retained to cover the claims liabilities of a company, liabilities equal to an estimate of all claims to be reported in the future are held. Since the final, or ultimate, amount of claims arising from each accident year $i, C_{i,J}$, is not known, Incurred but Not Reoprted (IBNR) techniques are applied to produce an estimate of these claims $\hat{C}_{i,J}^k$ in each calendar year k.

Then, IBNR reserves for accident year i, R_{i,j^*}^k are calculated in a cummulative display as $\hat{C}_{i,J}^k - C_{i,j^*}$, where $j^* = k - i$, ie, in practical terms, is equal to the difference between the last row minus the largest diagonal of the triangle (latest observed claims). This expression can be rearranged in terms of the forecast incremental claims such that $R_{i,j}^k = \sum_{l=j^*+1}^{J} \hat{X}_{i,l}$. Anyway, the estimation of IBNR reserves depends on the estimate of ultimate losses (Balona and Richman, 2020).

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import chainladder as cl
import os
```

In [ ]:
```
input = pd.read_csv(os.path.normpath(os.getcwd() + os.sep + os.pardir)+"/data
```

In [ ]:
```
input = input[input.DevelopmentYear <= 1997]
```

In [ ]:
```
cleaning_cond = np.array(['Adriatic Ins Co', 'Aegis Grp', 'Agency Ins Co Of Mi
        'Allegheny Cas Co', 'American Modern Ins Grp Inc',
        'Armed Forces Ins Exchange', 'Auto Club South Ins Co',
        'Baltica-Skandinavia Rein Co Of Amer', 'Bancinsure Inc',
        'Bell United Ins Co', 'Century-Natl Ins Co', 'Co-Operative Ins Co',
        'Consumers Ins Usa Inc', 'Cornerstone Natl Ins Co',
        'Federated Natl Ins Co', 'First Amer Ins Co',
        'Florists Mut Ins Grp', 'Harbor Ins Co', 'Homestead Ins Co',
        'Inland Mut Ins Co', 'Interstate Auto Ins Co Inc', 'Lancer Ins Co',
        'Lumber Ins Cos', 'Manhattan Re Ins Co', 'Mennonite Mut Ins Co',
        'Middle States Ins Co Inc', 'National Automotive Ins',
        'Nevada General Ins Co', 'New Jersey Citizens United Rcp Exch',
        'Nichido Fire & Marine Ins Co Ltd', 'Northwest Gf Mut Ins Co',
        'Ocean Harbor Cas Ins Co', 'Overseas Partners Us Reins Co',
        'Pacific Ind Ins Co', 'Pacific Pioneer Ins Co',
        'Pacific Specialty Ins Co', 'Penn Miller Grp',
        'Pennsylvania Mfg Asn Ins Co', 'Pioneer State Mut Ins Co',
        'Protective Ins Grp', 'San Antonio Reins Co',
        'Seminole Cas Ins Co', 'Southern Group Ind Inc',
        'Southern Mut Ins Co', 'Southland Lloyds Ins Co', 'Star Ins Grp',
        'Sterling Ins Co', 'Usauto Ins Co', 'Vanliner Ins Co',
        'Wea Prop & Cas Ins Co', 'Wellington Ins Co', 'State Farm Mut Grp', 'U
        'US Lloyds Ins Co', 'Toa-Re Ins Co Of Amer', 'FL Farm Bureau Grp'])
```

In [ ]:
```
input = input[~input.GRNAME.isin(cleaning_cond)]
```

After importing and cleaning data as on Initial Data Analysis was intended, we are estimating the full run-off triangles for our sample through the Chain ladder `cl` library for python, as it allows us to manage large amount of triangles and derive some relevant conclusions from this methodology.

In [ ]:
```
auto_triangles = cl.Triangle(input,
                             index = 'GRCODE',
                             origin="AccidentYear",
                             development="DevelopmentYear",
                             columns=["IncurLoss_B"],
                             cumulative=True)
```

In [ ]:
```
auto_triangles
```

Out[ ]:

	Triangle Summary
Valuation:	1997-12
Grain:	OYDY
Shape:	(90, 1, 10, 10)
Index:	[GRCODE]
Columns:	[IncurLoss_B]

In [ ]:
```
auto_model = cl.Chainladder().fit(auto_triangles)
```

```
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/chainlad
der/utils/weighted_regression.py:58: RuntimeWarning: Mean of empty slice
  xp.nansum(w * x * y, axis) - xp.nansum(x * w, axis) * xp.nanmean(y, axis)
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/chainlad
der/utils/weighted_regression.py:62: RuntimeWarning: Mean of empty slice
  intercept = xp.nanmean(y, axis) - slope * xp.nanmean(x, axis)
```

In [ ]:
```
auto_model.full_triangle_.iloc[43]
```

Out[ ]:

	12	24	36	48	60	72	84	96	108	120	132
1988	6,433	6,452	6,908	6,437	6,612	6,703	6,719	6,718	6,705	6,704	6,704
1989	8,697	8,438	8,515	8,392	8,428	8,532	8,643	8,459	8,483	8,482	8,482
1990	12,738	11,754	11,308	11,260	11,384	11,294	11,317	11,296	11,304	11,303	11,303
1991	13,658	12,448	12,312	12,507	12,281	12,177	12,027	11,934	11,943	11,941	11,941
1992	12,228	10,932	11,206	10,688	10,649	10,624	10,624	10,542	10,550	10,548	10,548
1993	13,138	13,305	14,017	13,953	13,759	13,752	13,752	13,646	13,656	13,654	13,654
1994	15,577	17,912	17,790	17,594	17,560	17,551	17,551	17,415	17,428	17,425	17,425
1995	15,785	18,672	18,297	18,024	17,989	17,980	17,980	17,841	17,854	17,851	17,851
1996	23,907	23,613	23,717	23,363	23,317	23,306	23,306	23,126	23,143	23,139	23,139
1997	23,369	23,630	23,734	23,380	23,334	23,323	23,323	23,143	23,159	23,156	23,156

Results

Taking an example from the sample, this company could expect that the ultimate losses for the last year of development will be at USD $23,156.

In [ ]:

```
auto_model.full_triangle_.mean(0).mean(1)
```

Out[ ]:

	12	24	36	48	60	72	84	96	108	120	132
1988	8,928	8,959	9,018	8,990	8,943	8,760	8,705	8,675	8,700	8,724	8,724
1989	9,830	10,168	10,236	10,224	10,164	10,091	10,022	9,989	9,971	9,990	9,990
1990	11,541	11,819	11,857	11,634	11,599	11,439	11,357	11,337	11,333	11,355	11,355
1991	12,760	12,693	12,572	12,431	12,226	12,109	12,052	12,011	12,012	12,029	12,029
1992	14,026	13,450	13,219	13,086	13,005	12,866	12,788	12,749	12,737	12,769	12,769
1993	15,488	15,259	15,148	14,853	14,725	14,562	14,466	14,419	14,412	14,441	14,441
1994	16,644	16,164	16,031	15,821	15,690	15,528	15,434	15,388	15,372	15,409	15,409
1995	17,590	17,227	17,147	16,948	16,801	16,597	16,473	16,412	16,404	16,435	16,435
1996	18,433	18,186	18,106	17,906	17,739	17,523	17,396	17,333	17,335	17,380	17,380
1997	19,374	19,385	19,364	19,107	18,943	18,669	18,494	18,408	18,414	18,441	18,441

In this sense, the average expected loss for the last development year is USD $18,441.

In [ ]:

```
auto_model.ultimate_.mean(0)
```

Out[ ]:

	2261
1988	8,724
1989	9,990
1990	11,355
1991	12,029
1992	12,769
1993	14,441
1994	15,409
1995	16,435
1996	17,380
1997	18,441

However, the IBNR that the companies must have for possible claims are expected for last year at levels on $83,008. On the other hand, the balance until 1990 is positive on IBNR, that means that this group of insurers are overestimating the amount of total auto claims that must to, as is showed below.

In [ ]:

```
auto_model.ibnr_.mean(0)
```

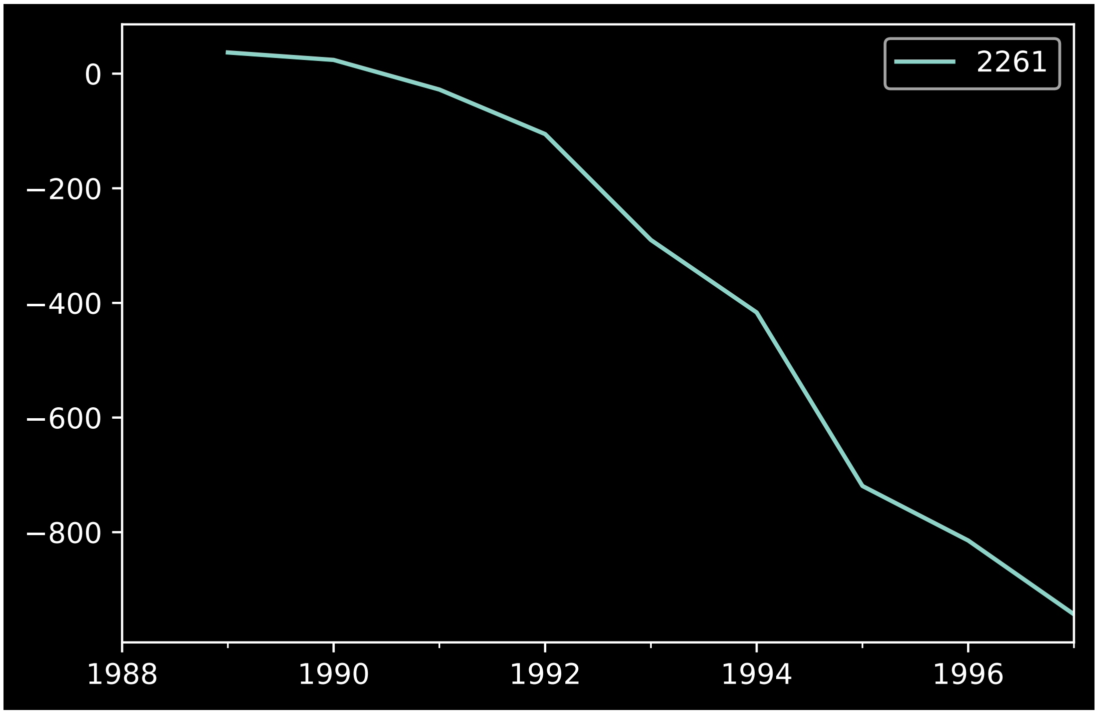
Out[ ]:

	2261
1988	
1989	1,702
1990	1,583
1991	-2,135
1992	-8,750
1993	-25,534
1994	-37,079
1995	-64,021
1996	-72,489
1997	-83,008

In [ ]:
```
auto_model.ibnr_.mean(0).plot()
```

```
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/chainlad
der/core/pandas.py:364: RuntimeWarning: Mean of empty slice
  obj.values = func(obj.values, axis=axis, *args, **kwargs)
```

Out[ ]: <AxesSubplot:>



Results evaluation

In order to verify the validity of above conclusions, we proceed to compute the Actual versus Expected (AvE) on full triangle. This measure could bring us dome relevant information about the errors that could arose from chainladder estimation procedure. This is the difference between the full triangle projected with the development factor estimations and the observed values on the upper run-off losses triangle.

In [ ]:
```
auto_AvE = auto_triangles - auto_model.full_expectation_
auto_AvE = auto_AvE[auto_AvE.valuation <= auto_triangles.valuation_date]
auto_actual_mean = auto_model.full_triangle_.mean(0).mean(1)[auto_model.full_
auto_AvE_percentage = 100*(auto_AvE.mean(0).mean(1)/auto_actual_mean)
```

```
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/chainlad
der/core/pandas.py:364: RuntimeWarning: Mean of empty slice
  obj.values = func(obj.values, axis=axis, *args, **kwargs)
```

In [ ]:

```
auto_AvE.mean(0).mean(1).heatmap()
```

```
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/chainlad
der/core/pandas.py:364: RuntimeWarning: Mean of empty slice
  obj.values = func(obj.values, axis=axis, *args, **kwargs)
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/pandas/i
o/formats/style.py:1126: RuntimeWarning: All-NaN slice encountered
  smin = np.nanmin(s.to_numpy()) if vmin is None else vmin
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/pandas/i
o/formats/style.py:1127: RuntimeWarning: All-NaN slice encountered
  smax = np.nanmax(s.to_numpy()) if vmax is None else vmax
```

Out[ ]:

	12	24	36	48	60	72	84	96	108	120
1988	-425.43	-232.89	-104.21	-41.88	-14.05	-11.01	-16.70	-29.94	-0.00	
1989	-718.18	-238.77	-108.38	-13.94	6.30	39.10	31.47	29.94	-0.00	
1990	-565.87	-144.76	-40.01	-10.49	45.14	0.56	-15.16			
1991	-26.03	148.89	74.68	70.75	-44.79	-29.00				
1992	611.48	152.62	-17.62	-16.43	7.31	-0.00				
1993	325.77	197.00	146.62	11.90						
1994	423.99	99.03	48.83	0.00						
1995	277.81	18.78	0.00							
1996	104.03	0.00								
1997	-0.00									

In [ ]:

```
auto_AvE_percentage.heatmap()
```

```
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/pandas/i
o/formats/style.py:1126: RuntimeWarning: All-NaN slice encountered
  smin = np.nanmin(s.to_numpy()) if vmin is None else vmin
/Users/raul/opt/miniconda3/envs/MiEntorno/lib/python3.8/site-packages/pandas/i
o/formats/style.py:1127: RuntimeWarning: All-NaN slice encountered
  smax = np.nanmax(s.to_numpy()) if vmax is None else vmax
```

Out[ ]:

	12	24	36	48	60	72	84	96	108	120
1988	-4.7649	-2.5995	-1.1556	-0.4659	-0.1571	-0.1257	-0.1919	-0.3452	-0.0000	
1989	-7.3057	-2.3481	-1.0589	-0.1363	0.0620	0.3875	0.3140	0.2998	-0.0000	
1990	-4.9032	-1.2248	-0.3374	-0.0902	0.3891	0.0049	-0.1335			
1991	-0.2040	1.1730	0.5940	0.5692	-0.3663	-0.2395				
1992	4.3596	1.1348	-0.1333	-0.1255	0.0562	-0.0000				
1993	2.1033	1.2911	0.9679	0.0801						
1994	2.5474	0.6126	0.3046	0.0000						
1995	1.5794	0.1090	0.0000							
1996	0.5644	0.0000								
1997	-0.0000									

From this, one could establish that the prediction errors would be grater for earlier years, this makes sense as the chain ladder procedure is intended to estimate the latest values of losses instead earliest. However, the differences are little in percentage points (larger up to 7.3\% and on average 0.1\%)

References

Balona, C., and Richman, R. (2020). The Actuary and IBNR Techniques: A Machine Learning Approach. Available at SSRN 3697256.

Regression models for IBNR estimates

After estimating IBNR using traditional methods, we're going to model run-off triangles data with some additional methods, in order to accomplish main project's objective.

The models that are presented next are a traditional linear regression, ridge regression and lasso regression, taking as input a variables `X` and `Y` for every company in the reequired timespan.

Model construction

This method is based on Kremer's(1982) approach exposed on Verrall(1985). Under lognormal and identically distributed assumptions (and every other that applies to a regression model) the chainladder procedure based on multiplicative display could be described as the following equation:

$$E(Z_{i,j}) = U_i S_j$$

Where U_i is a parameter for row i and S_j is a parameter for column j. Then, if $Y_{i,j} = \ln(Z_{i,j})$ and if $U_i = e^{\alpha_i + \mu} \sum_{j=1}^{t} e^{\beta_j}$, we have the equation

$$y = X\Beta + \varepsilon$$

Where X is a non-singular design matrix.

In [ ]:
```
import os
import itertools
import pandas as pd
import re
import math
import numpy as np

from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import LinearRegression
```

In [ ]:
```
def columnas(valores,variable):
    y = [re.findall("\\d+", j)[0] for j in valores]
    y = [int(i) for i in y]
    todas = list(set(y))
```

```
    df = pd.DataFrame()
    df[f"y_{variable}"] = y
    for k in todas:
        #print(k)
        df[f"{variable}_{k}"] = ([1 if k == j else 0 for j in y])
    return df

def matrix_X(df_triangulo):
    k = len(df_triangulo.columns)
    alpha = [f'a_{i}' for i in range(1,k+1)]
    mu    = [f'u_{i}' for i in range(1,k+1)]
    lists = [alpha, mu]
    df    = pd.DataFrame(list(itertools.product(*lists)), columns=['a', 'u'])

    alpha     = columnas(valores  = df.a, variable = 'a')
    mu        = columnas(valores=df.u, variable = 'u')
    df_col= pd.concat([alpha, mu], axis=1)


    df_col['y_a'] = df_col['y_a'].astype(str) + df_col['y_u'].astype(str)
    df_col['y_a'] = [int(i) for i in df_col['y_a']]
    df_col = df_col.drop(['y_u', 'u_1'], axis=1)
    df_col['a_1'] = 1
    df_col.rename(columns={'a_1': 'b0'}, inplace=True)
    df_col.rename(columns={'y_a': 'y_ii'}, inplace=True)
    #df_col = df_col.drop(['y_ii'], axis=1)
    return df_col

def matrix_y(df_triangulo):
    k = len(df_triangulo.columns)
    d0 = pd.DataFrame()
    for i in range(k):
        for j in range(k):
            d1 = pd.DataFrame({'y_ii': [int(f'{i+1}{j+1}')], 'Y': [math.log(d
            d0 = pd.concat([d0, d1], axis=0)
    return d0

def triangulo(df, grcode, entreno):

    if entreno:
        df_trinagulo = df[(df['GRCODE']== grcode ) & (df['DevelopmentYear']<=
    else:
        df_trinagulo = df[df['GRCODE']== grcode].copy()

    df_g          = df_trinagulo.groupby(["AccidentYear", "DevelopmentLag"]).a
    df_g.columns  = ['Pagos']
    df_g          = df_g.reset_index()
    pivot_data    = df_g.pivot(index='AccidentYear',columns='DevelopmentLag',v
    pivot_data    = pivot_data.drop('AccidentYear', axis=1).cumsum(axis=1)

    return pivot_data
```

In [ ]:

```
input = pd.read_csv(os.path.normpath(os.getcwd() + os.sep + os.pardir)+"/data

input = input[input.DevelopmentYear <= 1997]

cleaning_cond = np.array(['Adriatic Ins Co', 'Aegis Grp', 'Agency Ins Co Of M
        'Allegheny Cas Co', 'American Modern Ins Grp Inc',
        'Armed Forces Ins Exchange', 'Auto Club South Ins Co',
        'Baltica-Skandinavia Rein Co Of Amer', 'Bancinsure Inc',
        'Bell United Ins Co', 'Century-Natl Ins Co', 'Co-Operative Ins Co',
        'Consumers Ins Usa Inc', 'Cornerstone Natl Ins Co',
        'Federated Natl Ins Co', 'First Amer Ins Co',
        'Florists Mut Ins Grp', 'Harbor Ins Co', 'Homestead Ins Co',
        'Inland Mut Ins Co', 'Interstate Auto Ins Co Inc', 'Lancer Ins Co',
        'Lumber Ins Cos', 'Manhattan Re Ins Co', 'Mennonite Mut Ins Co',
        'Middle States Ins Co Inc', 'National Automotive Ins',
        'Nevada General Ins Co', 'New Jersey Citizens United Rcp Exch',
        'Nichido Fire & Marine Ins Co Ltd', 'Northwest Gf Mut Ins Co',
        'Ocean Harbor Cas Ins Co', 'Overseas Partners Us Reins Co',
        'Pacific Ind Ins Co', 'Pacific Pioneer Ins Co',
        'Pacific Specialty Ins Co', 'Penn Miller Grp',
        'Pennsylvania Mfg Asn Ins Co', 'Pioneer State Mut Ins Co',
        'Protective Ins Grp', 'San Antonio Reins Co',
        'Seminole Cas Ins Co', 'Southern Group Ind Inc',
        'Southern Mut Ins Co', 'Southland Lloyds Ins Co', 'Star Ins Grp',
        'Sterling Ins Co', 'Usauto Ins Co', 'Vanliner Ins Co',
        'Wea Prop & Cas Ins Co', 'Wellington Ins Co', 'State Farm Mut Grp', 'U
        'US Lloyds Ins Co', 'Toa-Re Ins Co Of Amer', 'FL Farm Bureau Grp'])

input = input[~input.GRNAME.isin(cleaning_cond)]

Lista_entidades_ceros = input[input.IncurLoss_B <= 0]["GRCODE"].unique()
input = input[~input.GRCODE.isin(Lista_entidades_ceros)]
#input.IncurLoss_B = input.IncurLoss_B+1 #deal with NaN from log transformati
```

In [ ]:

```
df_data         = input #pd.read_csv('medmal_pos.csv')
df_trg_entreno = triangulo(df_data, grcode=43, entreno=True)
df_trg_prueba  = triangulo(df_data, grcode=43, entreno=False)
df_trg_entreno
```

Out[ ]:

DevelopmentLag	1	2	3	4	5	6	7	8
0	607.0	1254.0	1836.0	2434.0	3048.0	3663.0	4278.0	4892.0
1	2254.0	5113.0	8092.0	10856.0	13682.0	16699.0	19689.0	22667.0
2	5843.0	13267.0	21574.0	30245.0	39311.0	48237.0	57004.0	65769.0
3	11422.0	27515.0	46163.0	65258.0	83911.0	102380.0	120787.0	NaN
4	19933.0	44095.0	72834.0	101163.0	129234.0	156956.0	NaN	NaN
5	24604.0	56734.0	90309.0	123078.0	156800.0	NaN	NaN	NaN
6	40735.0	84679.0	127190.0	168902.0	NaN	NaN	NaN	NaN
7	43064.0	86769.0	129678.0	NaN	NaN	NaN	NaN	NaN
8	41837.0	83141.0	NaN	NaN	NaN	NaN	NaN	NaN
9	44436.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [ ]:

```
Y = matrix_y(df_trg_entreno)
X = matrix_X(df_trg_entreno)

Y_X           = pd.merge(Y, X, on='y_ii', how='inner')
data_entreno = Y_X[Y_X['Y'].notna()]
data_entreno = data_entreno.drop(['y_ii'], axis=1)
data_entreno.head()
```

Out[ ]:

	Y	b0	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_10	u_2	u_3	u_4	u_5	u_6	u
0	6.408529	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	7.134094	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	
2	7.515345	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
3	7.797291	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	
4	8.022241	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	

In [ ]:

```
Y_prueba       = matrix_y(df_trg_prueba)
x_prueba       = Y_X[Y_X['Y'].notna()].drop(['Y'], axis=1)
data_prueba_   = pd.merge(Y_prueba, x_prueba, on='y_ii', how='inner')
data_prueba=data_prueba_.drop(['y_ii'], axis=1)
y_ii = data_prueba_['y_ii']

x_entreno = data_entreno.drop('Y', axis=1)  # Features
y_entreno = data_entreno['Y']  # Target variable
x_prueba  = data_prueba.drop('Y', axis=1)  # Features
y_prueba  = data_prueba['Y']  # Target variable
```

Models considered:

The next steps require estimating three type of models:

Ordinary Least Squares regression

The simple linear regression consists of generating a regression model (equation of a line) that explains the linear relationship between two variables. The dependent or response variable is identified as Y, and the predictor or independent variable is identified as X.

The simple linear regression model is described according to the equation:

$$Y = \beta_0 + \beta_1 X_1 + \varepsilon$$

Here, β_0 is the y-intercept, β_1 is the slope, and $\varepsilon istherandomerror. Therandomerrorrepresentsthedifferencebetweenthevalueadj$
Y$ but are not included in the model as predictors. The random error is also known as the residual.

In the vast majority of cases, the population values of β_0 and \beta_1$ are unknown. Therefore, from a sample, their estimations are obtained, these estimates are known as regression coefficients or least square coefficient estimates since they take values that minimize the sum of squared residuals, resulting in the line that passes closest to all points.

Ridge regression

Ridge regularization penalizes the sum of the coefficients squared. This penalty has the effect of proportionally reducing the value of all coefficients in the model without letting them reach zero. The degree of penalization is controlled by the hyperparameter λ. When $\lambda = 0$, the penalty is null, and the result is equivalent to that of a linear model by ordinary least squares (OLS). As λ increases, the penalty becomes stronger, and the values of the predictors decrease.

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = \text{residual squared sum} + \lambda\sum_{j=1}^{p}\beta_j^2$$

The main advantage of applying ridge over ordinary least squares (OLS) fitting is the reduction of variance. Generally, in situations where the relationship between the response variable and predictors is approximately linear, least squares estimates have little bias but can still suffer from high variance (small changes in the training data have a significant impact on the resulting model). This problem is accentuated as the number of predictors introduced into the model approaches the number of training observations, reaching the point where, if $p > n$, it is not possible to fit the model by ordinary least squares. By using an appropriate value of λ, the ridge method can reduce variance without significantly increasing bias, thus achieving lower total error.

The disadvantage of the ridge method is that the final model includes all predictors. This is because, although the penalty forces the coefficients to tend toward zero, they never become exactly zero (only if $\lambda = \inf$). This method minimizes the influence on the model of predictors less related to the response variable, but in the final model, they will still appear. Although this is not a problem for the accuracy of the model, it is a challenge for its interpretation

Lasso regression

Lasso regularization penalizes the sum of the absolute values of the regression coefficients. This penalty is known as l1 and has the effect of forcing the coefficients of the predictors to tend towards zero. Since a predictor with a regression coefficient of zero does not influence the model, lasso manages to exclude the less relevant predictors. Similar to ridge, the degree of penalization is controlled by the hyperparameter λ. When λ = 0, the result is equivalent to that of a linear model by ordinary least squares. As λ increases, the penalty becomes stronger, and more predictors are excluded.

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \text{residual squared sum} + \lambda\sum_{j=1}^{p}|\beta_j|$$

The main practical difference between lasso and ridge is that the former manages to make some coefficients exactly zero, thus performing predictor selection, while the latter does not exclude any. This represents a significant advantage of lasso in scenarios where not all predictors are important for the model, and it is desired that the least influential ones be excluded.

On the other hand, when there are highly correlated predictors (linearly), ridge reduces the influence of all of them simultaneously and proportionally, while lasso tends to select one of them, giving it all the weight and excluding the others. In the presence of correlations, this selection varies a lot with small perturbations (changes in the training data), so lasso solutions are very unstable if predictors are highly correlated, which is taken into account in the next steps.

In [ ]:
```
Regresion_lineal = LinearRegression()
Regresion_lineal.fit(x_entreno, y_entreno)
LR_coef = Regresion_lineal.coef_
y_pred = Regresion_lineal.predict(x_prueba)

mse = mean_squared_error(y_prueba, y_pred)   # Considerar que se debe aplicar
mape = mean_absolute_percentage_error(y_prueba, y_pred)   # COnsiderar que se
[mse, mape]
```

Out[ ]:
```
[0.002230701729399499, 0.0038343781268415497]
```

In [ ]:
```
alpha = 0.00001
ridge_model = Ridge(alpha = alpha) #aplicación regresión de ridge
ridge_model.fit(x_entreno, y_entreno) #entrenamiento regresión de ridge
ridge_coef = ridge_model.coef_ #coeficientes regresión de ridge
y_pred = ridge_model.predict(x_prueba)

mse = mean_squared_error(y_prueba, y_pred)   # COnsiderar que se debe aplicar
mape = mean_absolute_percentage_error(y_prueba, y_pred)   # COnsiderar que se
[mse, mape]
```

Out[ ]:
```
[0.002230702303162153, 0.003834869524506071]
```

In [ ]:
```
lasso_model = Lasso(alpha = alpha) #aplicación regresión de lasso
lasso_model.fit(x_entreno, y_entreno) #entrenamiento regresión de lasso
lasso_coef = lasso_model.coef_ #coeficientes regresión de lasso
y_pred = lasso_model.predict(x_prueba)

mse = mean_squared_error(y_prueba, y_pred)   # Considerar que se debe aplicar
mape = mean_absolute_percentage_error(y_prueba, y_pred)   # Considerar que se
[mse, mape]
```

Out[ ]:
```
[0.0022309333284786215, 0.003843818877160590 4]
```

In [ ]:
```
np.exp(y_pred)
```

```
Out[ ]: array([   523.69588732,   1138.91125798,   1800.60267558,   2481.25822179,
                 3197.47927039,   3891.61694156,   4565.05488313,   5164.35159533,
                 5680.89885172,   6130.        ,   2295.86135051,   4992.94037277,
                 7893.76848391,  10877.73455923,  14017.61834235,  17060.68950208,
                20013.01389388,  22640.30616852,  24904.82821343,   6338.44464237,
                13784.57638498,  21793.22133009,  30031.39467575,  38700.02770899,
                47101.37916013,  55252.19572384,  62505.6592827 ,  13208.63690653,
                28725.57459323,  45414.72929312,  62582.19649294,  80646.69538382,
                98154.20820492, 115139.6332668 ,  20832.59948126,  45305.8400049 ,
                71627.89564198,  98704.34348534, 127195.58545728, 154808.35164171,
                25717.11423791,  55928.47230123,  88422.12785315, 121847.05415536,
               157018.49424527,  38030.54940143,  82707.20070384, 130758.92070899,
               180187.806828  ,  40167.54770632,  87354.65256788, 138106.47672126,
                39993.8249981 ,  86976.84790502,  44437.        ])
```

Cross validation

Next to model implementation and checking on the similar results of the three modeling procedures, we need to train, validate and test to find the best model of IBNR. Thus, we proceed next with a Cross Validation, running on a list of 10 insurers and excluding one at a time on a loop that splits every insurer's sub-sample on training, validation and testing data.

With this in mind, the next loop estimate the three models for every company. The process is completed once every company has already trained, validated and tested each of the models.

```
In [ ]:
df_CV = input[input["GRCODE"].isin(list(input["GRCODE"].unique()[0:10]))]
lista_aseguradoras = df_CV["GRCODE"].unique()
mejore_modelos_test_full = {}
mejores_mape = {}

for i in range(len(lista_aseguradoras)): #recorre las aseguradoras de test
    print("aseguradora de test:", i)
    conj_test = lista_aseguradoras[i] #codigo de aseguradora de testeo
    datos_test = df_CV[df_CV["GRCODE"].isin([conj_test])] #datos de asegurado
    conj_entre_valid = np.delete(lista_aseguradoras, i, axis=0) #Conjunto de

    for j in range(len(conj_entre_valid)): #recorre los datos de entrenamient

        conj_vali = conj_entre_valid[j] #datos de aseguradora de validación
        conj_entre = np.delete(conj_entre_valid, j, axis=0) #aseguradoras de
        datos_train = df_CV[df_CV["GRCODE"].isin(conj_entre)] #datos de asegu
        datos_validacion = df_CV[df_CV["GRCODE"].isin([conj_vali])] #datos de

        #Se crea la clase que calculará las regresiones
        df_data        = input #pd.read_csv('medmal_pos.csv')
        df_trg_entreno = triangulo(datos_train, entreno=True, grcode=43)
        df_trg_prueba  = triangulo(datos_validacion, entreno=False, grcode=co

        Y_prueba      = matrix_y(df_trg_prueba)
```

```
        x_prueba      = Y_X[Y_X['Y'].notna()].drop(['Y'], axis=1)
        data_prueba_  = pd.merge(Y_prueba, x_prueba, on='y_ii', how='inner')
        data_prueba=data_prueba_.drop(['y_ii'], axis=1)
        y_ii = data_prueba_['y_ii']

        x_entreno = data_entreno.drop('Y', axis=1)  # Features
        y_entreno = data_entreno['Y']  # Target variable
        x_prueba  = data_prueba.drop('Y', axis=1)  # Features
        y_prueba  = data_prueba['Y']  # Target variable

        Regresion_lineal.fit(x_entreno, y_entreno)
        LR_coef = Regresion_lineal.coef_
        y_pred_1 = Regresion_lineal.predict(x_prueba)
        mape_1 = mean_squared_error(y_prueba, y_pred_1)
        Regresion_lineal_sum = [Regresion_lineal.intercept_, Regresion_lineal

        ridge_model.fit(x_entreno, y_entreno)
        ridge_coef = Regresion_lineal.coef_
        y_pred_2 = ridge_model.predict(x_prueba)
        mape_2 = mean_squared_error(y_prueba, y_pred_2)
        ridge_sum = [ridge_model.intercept_, ridge_model.coef_, ridge_model.s

        lasso_model.fit(x_entreno, y_entreno)
        lasso_coef = Regresion_lineal.coef_
        y_pred_3 = lasso_model.predict(x_prueba)
        mape_3 = mean_squared_error(y_prueba, y_pred_3)
        lasso_sum = [lasso_model.intercept_, lasso_model.coef_, lasso_model.s

        results = [mape_1, mape_2, mape_3]
        models = [Regresion_lineal_sum, ridge_sum, lasso_sum]

        if (mape_1<mape_2) & (mape_1<mape_3):
            modelo_test=Regresion_lineal
            if (mape_2<mape_1) & (mape_2<mape_3):
                modelo_test=ridge_model
            else:
                modelo_test=lasso_model

        y_pred = modelo_test.predict(x_prueba)
        mejores_mape[i,j] = mean_squared_error(y_prueba, y_pred)

        mejore_modelos_test_full["modelo_"+str(i)+"-"+str(j)] = modelo_test #
```

```
aseguradora de test: 0
aseguradora de test: 1
aseguradora de test: 2
aseguradora de test: 3
aseguradora de test: 4
aseguradora de test: 5
aseguradora de test: 6
aseguradora de test: 7
aseguradora de test: 8
aseguradora de test: 9
```

In [ ]:
```
list(dict(sorted(mejores_mape.items(), key=lambda item: item[1])).keys())[0]
```

Out[ ]:
```
(1, 0)
```

In [ ]:
```
mejore_modelos_test_full #winner is number 10
```

Out[ ]:
```
{'modelo_0-0': Lasso(alpha=1e-05),
 'modelo_0-1': Lasso(alpha=1e-05),
 'modelo_0-2': Lasso(alpha=1e-05),
 'modelo_0-3': Lasso(alpha=1e-05),
 'modelo_0-4': Lasso(alpha=1e-05),
 'modelo_0-5': Lasso(alpha=1e-05),
 'modelo_0-6': Lasso(alpha=1e-05),
 'modelo_0-7': Lasso(alpha=1e-05),
 'modelo_0-8': Lasso(alpha=1e-05),
 'modelo_1-0': Lasso(alpha=1e-05),
 'modelo_1-1': Lasso(alpha=1e-05),
 'modelo_1-2': Lasso(alpha=1e-05),
 'modelo_1-3': Lasso(alpha=1e-05),
 'modelo_1-4': Lasso(alpha=1e-05),
 'modelo_1-5': Lasso(alpha=1e-05),
 'modelo_1-6': Lasso(alpha=1e-05),
 'modelo_1-7': Lasso(alpha=1e-05),
 'modelo_1-8': Lasso(alpha=1e-05),
 'modelo_2-0': Lasso(alpha=1e-05),
 'modelo_2-1': Lasso(alpha=1e-05),
 'modelo_2-2': Lasso(alpha=1e-05),
 'modelo_2-3': Lasso(alpha=1e-05),
 'modelo_2-4': Lasso(alpha=1e-05),
 'modelo_2-5': Lasso(alpha=1e-05),
 'modelo_2-6': Lasso(alpha=1e-05),
 'modelo_2-7': Lasso(alpha=1e-05),
 'modelo_2-8': Lasso(alpha=1e-05),
 'modelo_3-0': Lasso(alpha=1e-05),
 'modelo_3-1': Lasso(alpha=1e-05),
 'modelo_3-2': Lasso(alpha=1e-05),
 'modelo_3-3': Lasso(alpha=1e-05),
 'modelo_3-4': Lasso(alpha=1e-05),
 'modelo_3-5': Lasso(alpha=1e-05),
 'modelo_3-6': Lasso(alpha=1e-05),
 'modelo_3-7': Lasso(alpha=1e-05),
 'modelo_3-8': Lasso(alpha=1e-05),
 'modelo_4-0': Lasso(alpha=1e-05),
 'modelo_4-1': Lasso(alpha=1e-05),
 'modelo_4-2': Lasso(alpha=1e-05),
 'modelo_4-3': Lasso(alpha=1e-05),
 'modelo_4-4': Lasso(alpha=1e-05),
 'modelo_4-5': Lasso(alpha=1e-05),
 'modelo_4-6': Lasso(alpha=1e-05),
 'modelo_4-7': Lasso(alpha=1e-05),
 'modelo_4-8': Lasso(alpha=1e-05),
 'modelo_5-0': Lasso(alpha=1e-05),
 'modelo_5-1': Lasso(alpha=1e-05),
```

```
 'modelo_5-2': Lasso(alpha=1e-05),
 'modelo_5-3': Lasso(alpha=1e-05),
 'modelo_5-4': Lasso(alpha=1e-05),
 'modelo_5-5': Lasso(alpha=1e-05),
 'modelo_5-6': Lasso(alpha=1e-05),
 'modelo_5-7': Lasso(alpha=1e-05),
 'modelo_5-8': Lasso(alpha=1e-05),
 'modelo_6-0': Lasso(alpha=1e-05),
 'modelo_6-1': Lasso(alpha=1e-05),
 'modelo_6-2': Lasso(alpha=1e-05),
 'modelo_6-3': Lasso(alpha=1e-05),
 'modelo_6-4': Lasso(alpha=1e-05),
 'modelo_6-5': Lasso(alpha=1e-05),
 'modelo_6-6': Lasso(alpha=1e-05),
 'modelo_6-7': Lasso(alpha=1e-05),
 'modelo_6-8': Lasso(alpha=1e-05),
 'modelo_7-0': Lasso(alpha=1e-05),
 'modelo_7-1': Lasso(alpha=1e-05),
 'modelo_7-2': Lasso(alpha=1e-05),
 'modelo_7-3': Lasso(alpha=1e-05),
 'modelo_7-4': Lasso(alpha=1e-05),
 'modelo_7-5': Lasso(alpha=1e-05),
 'modelo_7-6': Lasso(alpha=1e-05),
 'modelo_7-7': Lasso(alpha=1e-05),
 'modelo_7-8': Lasso(alpha=1e-05),
 'modelo_8-0': Lasso(alpha=1e-05),
 'modelo_8-1': Lasso(alpha=1e-05),
 'modelo_8-2': Lasso(alpha=1e-05),
 'modelo_8-3': Lasso(alpha=1e-05),
 'modelo_8-4': Lasso(alpha=1e-05),
 'modelo_8-5': Lasso(alpha=1e-05),
 'modelo_8-6': Lasso(alpha=1e-05),
 'modelo_8-7': Lasso(alpha=1e-05),
 'modelo_8-8': Lasso(alpha=1e-05),
 'modelo_9-0': Lasso(alpha=1e-05),
 'modelo_9-1': Lasso(alpha=1e-05),
 'modelo_9-2': Lasso(alpha=1e-05),
 'modelo_9-3': Lasso(alpha=1e-05),
 'modelo_9-4': Lasso(alpha=1e-05),
 'modelo_9-5': Lasso(alpha=1e-05),
 'modelo_9-6': Lasso(alpha=1e-05),
 'modelo_9-7': Lasso(alpha=1e-05),
 'modelo_9-8': Lasso(alpha=1e-05)}
```

In [ ]:

```
winner = mejore_modelos_test_full['modelo_1-0']
```

As we could see, the best model is a Lasso regression with the next parameters:

In [ ]:

```
winner.coef_
```

Out[ ]:

```
array([0.        , 1.47844251, 2.49413039, 3.22837566, 3.68398512,
       3.89455874, 4.28571065, 4.34024567, 4.33566825, 4.44036036,
       0.77618137, 1.23415607, 1.5547379 , 1.80827364, 2.00466563,
       2.16418097, 2.28739579, 2.38248208, 2.45789794])
```

In [ ]:
```
df_data         = input
df_trg_prueba   = triangulo(df_data, grcode=43, entreno=False)

Y_prueba       = matrix_y(df_trg_prueba)
x_prueba       = Y_X[Y_X['Y'].notna()].drop(['Y'], axis=1)
data_prueba_   = pd.merge(Y_prueba, x_prueba, on='y_ii', how='inner')
data_prueba=data_prueba_.drop(['y_ii'], axis=1)
y_ii = data_prueba_['y_ii']

X_test  = data_prueba.drop('Y', axis=1)  # Features
Y_test  = data_prueba['Y']  # Target variable


Y_pred = winner.predict(X_test)
```

And we recall that this model offers us a MAPE of 0.002, according to the test data:

In [ ]:
```
mejores_mape[(1,0)]
```

Out[ ]: 0.0022309333284786215

Showing the predictions, we must see that the error values are short in comparison with the other models implemented. Next to this, the model's prediction are computed in terms of $USD. Thus, model predicts that IBNR's for the largest development year must be $USD 180,109.086, compared to actual IBNR's of $USD 168,902 the difference is 0.53\%.

In [ ]:
```
pd.DataFrame({"Test" : np.exp(Y_test), "Predicted" : np.exp(Y_pred), "Percent
```

Out[ ]:

	Test	Predicted	Percentage difference
0	607.0	523.674041	2.36
1	1254.0	1138.026387	1.38
2	1836.0	1799.070652	0.27
3	2434.0	2478.992060	-0.23
4	3048.0	3194.363303	-0.58
5	3663.0	3887.552493	-0.72
6	4278.0	4559.874610	-0.76
7	4892.0	5157.798995	-0.62
8	5506.0	5672.308827	-0.34
9	6120.0	6116.634925	0.01
10	2254.0	2296.891439	-0.24

11	5113.0	4991.507813	0.28
12	8092.0	7890.920032	0.28
13	10856.0	10873.129458	-0.02
14	13682.0	14010.825728	-0.25
15	16699.0	17051.229096	-0.21
16	19689.0	20000.107205	-0.16
17	22667.0	22622.668748	0.02
18	25645.0	24879.364967	0.30
19	5843.0	6342.319410	-0.94
20	13267.0	13782.861632	-0.40
21	21574.0	21788.898873	-0.10
22	30245.0	30023.560905	0.07
23	39311.0	38687.562878	0.15
24	48237.0	47082.913640	0.23
25	57004.0	55225.539170	0.29
26	65769.0	62467.119115	0.47
27	11422.0	13216.814113	-1.54
28	27515.0	28722.224213	-0.42
29	46163.0	45406.074262	0.15
30	65258.0	62566.357483	0.38
31	83911.0	80621.345911	0.35
32	102380.0	98116.489762	0.37
33	120787.0	115084.977324	0.41
34	19933.0	20844.690772	-0.45
35	44095.0	45298.804758	-0.25
36	72834.0	71611.476796	0.15
37	101163.0	98675.547930	0.22
38	129234.0	127150.689326	0.14
39	156956.0	154742.880643	0.12
40	24604.0	25730.392726	-0.44
41	56734.0	55916.206635	0.13
42	90309.0	88396.198430	0.19

43	123078.0	121803.706684	0.09
44	156800.0	156953.020199	-0.01
45	40735.0	38047.097584	0.65
46	84679.0	82682.351296	0.21
47	127190.0	130709.967139	-0.23
48	168902.0	180109.085921	-0.53
49	43064.0	40179.616783	0.65
50	86769.0	87316.652273	-0.06
51	129678.0	138036.189953	-0.53
52	41837.0	39996.117913	0.42
53	83141.0	86917.880251	-0.39
54	44436.0	44410.437255	0.01

In [ ]:

```
print("Best Lasso estimator prediction MAPE: ",  np.round((100*(Y_test-Y_pred
```

```
Best Lasso estimator prediction MAPE:  0.006 %
```

Conclussions

As we have seen, the Lasso regression model has an excelent performance over other methodologies, even surpassing chainladder method as we estimated an error between 7% and 0.1\%, versus a Lasso that gives us a consistent 0.006\%.

We can conclude that implementing Lasso regularization methods bring IBNR's regression a better consistent estimation and a better precision for predictions. This is very useful from business perspective, as a company will seek for optimize their liabilities with respect to their assets, as claims are the larger risk source for an insurer.

However, the model could be revisited on the strenght of Lasso methdologies, it could be a matter of study the implementation of elastic net alternatives to Lasso regularization. Furthermore, this tunning was realized using a very short companies sample for meet with reasonable time processing of run-off triangles data.

Lastly, even when the Lasso model is the most accurate, we must observe that chainladder traditional procedure offers a practical and less time and resources consuming method that is pretty precise. So the trade-off between resources and precision has to be taken into account when selecting predictive strategies.

References

- Verrall, R. J. (1994). Statistical methods for the chain ladder technique. In Casualty Actuarial Society Forum (Vol. 1, pp. 393-446).