```
{
"Session Name": "Semi-structured data – best or worst of both worlds?",
"cast":[
        { "speaker":"Jim Weiss","designation":["FCAS","CPCU"] ,"social": "linkedin.com/in/jimtheactuary"},
        {"speaker": "Khanh Luu","designation":"ACAS" ,"social": "linkedin.com/in/khanh-luu-acas-14929630"},
],
"venue":"Casualty Actuaries of the Northwest Annual Meeting",
"session start":"2022-03-16 07:15:00PM GMT" ,
"session end":"2022-03-16 08:30:00PM GMT" ,
"number slides": 13
}
```

Session (semi) structure

- Warm-up | *Live polling questions*
- Background | *Problem statement and refresher on data forms*
- Application | *Cases where semi-structured data may enhance actuarial analysis*
- Tips | *Exemplary techniques and software*

Poll #1 - Autonomy

What is the typical level of IT involvement in a new actuarial analysis?

a. Ambient | *Infrastructure and security only*
b. Back | *Required for extraction and custom reporting*
c. Front | *Required to "productionize" analysis results*
d. Back and front | *Involved at multiple phases of typical project*

Poll #2 - Accessibility

What portion of realizable insights contained in insurance databases are actuaries typically able to unlock?

a. 76-100% | *We are the great liberators of insights*

b. 51-75% | *There may be more hiding under the covers*

c. 0-50% | *The insights are trapped and must be rescued*

d. Don't know | *No idea what this data is capable of*

Problem statements

- Dark data
  - "Unmatchables"
  - "Burial at sea"
- Imperfect governance
  - Lacking documentation
  - Ambiguous definitions
- Innovation frictions
  - Software development cycle
  - Compatibility

Could using a different data-interchange approach help solve all these problems?

Data forms

Structured

pre-defined model
tabular format
SQL tables, Excel files

Metadata

Data about data
Not a separate data structure
Infoschema

Semi-structured

No pre-defined model
Some hierarchy, contains tag/ text
JSON data, XML data

Unstructured

No pre-defined model
Documents, Image,
Video, Audio

As we go from Structured to Semi-structured to Unstructured, data generally becomes more text heavy.

Triangle example

	Incurral.Year	012	024	036	048	060	072	084	096	108	120
1	1995	17674	32062	38619	42035	43829	44723	45162	45375	45483	45540
2	1996	18315	32791	39271	42933	44950	45917	46392	46600	46753	NA
3	1997	18606	32942	39634	43411	45428	46357	46681	46921	NA	NA
4	1998	18816	33667	40575	44446	46476	47350	47809	NA	NA	NA
5	1999	20649	36515	43724	47684	49753	50716	NA	NA	NA	NA
6	2000	22327	39312	46848	51065	53242	NA	NA	NA	NA	NA
7	2001	23141	40527	48284	52661	NA	NA	NA	NA	NA	NA
8	2002	24301	42168	50356	NA	NA	NA	NA	NA	NA	NA
9	2003	24210	41640	NA	NA	NA	NA	NA	NA	NA	NA
10	2004	24468	NA	NA	NA	NA	NA	NA	NA	NA	NA

See appendix for how we brought this down into R

Source: R package 'insuranceData'


```
[
    {
        "report": "Schedule P",
        "subreport": "Part 3b",
        "subcoverage": "Liability",
        "reportDate": "2004-12-31",
        "Triangle": [
            {
                "Incurral.Year": 1995,
                "012": 17674,
                "024": 32062,
                "036": 38619,
                "048": 42035,
                "060": 43829,
                "072": 44723,
                "084": 45162,
                "096": 45375,
                "108": 45483,
                "120": 45540
            },
            {
                "Incurral.Year": 1996,
                "012": 18315,
                "024": 32791,
                "036": 39271,
                "048": 42933,
                "060": 44950,
                "072": 45917,
                "084": 46392,
                "096": 46600,
                "108": 46753
            },
            {
                "Incurral.Year": 1997,
                "012": 18606,
                "024": 32942,
```

Pros and cons

- Implicitly documented
- Lightweight
  - Grouping of similar entries
  - No need to store empties
- Web friendly
- Pre-joined
- Useful conduit

We will address above in "Application"

- Difficult to enforce controls
- Heavyweight
  - Redundant tags
  - Inability to index
- Ugly
- Lack of common keys
- Generally requires rendering

We will address above in "Tips"

Application: Rating a policy

Driver	Fav Color
Jim	Green
Khanh	Red
Kiki	Blue
Mikey	Orange

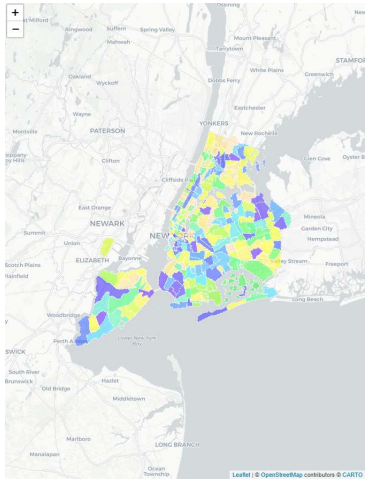
Vehicle	Model Year	MSRP	Territory
Prius	2022	30,000	CA 001
Ferrari	2015	225,000	WA 003

Coverage	Limit	Deductible
BI	100/300	
PD	25	
UM	100/300	
OTC		$500

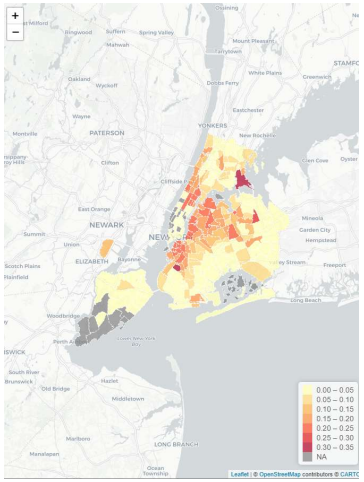
Effective	11/19/2021
Expiration?	11/19/2022
Payment	Installments
Card on File	**** **** 1234

- The input (user experience) should be structured (e.g. text boxes, dropdowns)
- The premium calculation (thought process, abstraction) is unstructured (e.g. model executable)
- The output (e.g. acceptance, premium) is structured
- How much additional data structure is (un)necessary to connect the endpoints to the underlying process?

Application: Describing spatial relationships




```
{
"type" : "FeatureCollection",
"features" : [
{
"type" : "Feature",
"id" : 0,
"geometry" : {
"type" : "Polygon",
"coordinates" : [
[
[
-74.184454504536305,
40.695004246630056
],
[
-74.185633504709287,
40.691656245932521
],
[
-74.185912504786472,
40.691458245883872
],
[
-73.952187654924685,
40.77302811508185
]
]
]
},
"properties" : {
"FID" : 262,
"OBJECTID" : 263,
"Shape_Leng" : 0.037016625299400002,
"Shape_Area" : 6.5769766416899996e-05,
"zone" : "Yorkville West",
"LocationID" : 263,
"borough" : "Manhattan"
}
}
]
}
```

- The input (elaborate polygons) is essentially unstructured (e.g. jpeg)
- The context (a fare for each polygon) is structured (e.g. data table)
- The output (choropleth map) is unstructured (e.g. jpeg)
- How many clicks, scrolls, uploads etc. are typically (un)necessary to represent the context in an unstructured way?

Analysis from 2021 CAS Data Visualization Workshop @ RPM for more info on Leaflets
For related data visit: https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc

Tip: Parsing semi-structured data

JSON from earlier


```
> Triangle3
[{"report":"Schedule P","subreport":"Part 3b","subcoverage":"Liability","reportDate":"2004-12-31","Triangle":[{"Incurral.Year":1995,"012":17674,"024":32062,"036":38619,"048":42035,"060":43829,"072":44723,"084":45162,"096":45375,"108":45483,"120":45540},{"Incurral.Year":1996,"012":18315,"024":32791,"036":39271,"048":42933,"060":44950,"072":45917,"084":46392,"096":46600,"108":46753},{"Incurral.Year":1997,"012":18606,"024":32942,"036":39634,"048":43411,"060":45428,"072":46357,"084":46681,"096":46921},{"Incurral.Year":1998,"012":18816,"024":33667,"036":40575,"048":44446,"060":46476,"072":47350,"084":47809},{"Incurral.Year":1999,"012":20649,"024":36515,"036":43724,"048":47684,"060":49753,"072":50716},{"Incurral.Year":2000,"012":22327,"024":39312,"036":46848,"048":51065,"060":53242},{"Incurral.Year":2001,"012":23141,"024":40527,"036":48284,"048":52661},{"Incurral.Year":2002,"012":24301,"024":42168,"036":50356},{"Incurral.Year":2003,"012":24210,"024":41640},{"Incurral.Year":2004,"012":24468}]}]
```

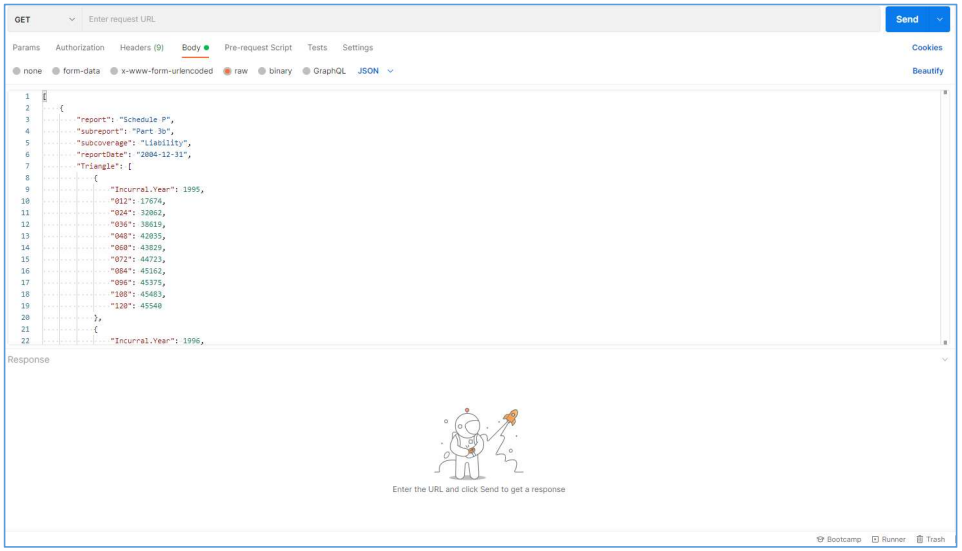

fromJSON in jsonlite R package creates dataframe

	report	subreport	subcoverage	reportDate	Triangle
1	Schedule P	Part 3b	Liability	2004-12-31	*11 variables*

Unpack nested information with list commands

	Incurral.Year	X012	X024	X036	X048	X060	X072	X084	X096	X108	X120
1	1995	17674	32062	38619	42035	43829	44723	45162	45375	45483	45540
2	1996	18315	32791	39271	42933	44950	45917	46392	46600	46753	*NA*
3	1997	18606	32942	39634	43411	45428	46357	46681	46921	*NA*	*NA*
4	1998	18816	33667	40575	44446	46476	47350	47809	*NA*	*NA*	*NA*
5	1999	20649	36515	43724	47684	49753	50716	*NA*	*NA*	*NA*	*NA*
6	2000	22327	39312	46848	51065	53242	*NA*	*NA*	*NA*	*NA*	*NA*
7	2001	23141	40527	48284	52661	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
8	2002	24301	42168	50356	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
9	2003	24210	41640	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
10	2004	24468	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*

Call an API with Postman or HTTR



Tip: making associations

- ++ Edit distance/ Levenshtein distance >> Approximate matching.
- Example: Business name
  - Both Python/ SAS have built in package (e.g. Python has fuzzywuzzy, SAS has compged)
  - Matched if partial elements contained within string
- Example: Person name
  - Python package allows matching when order of elements are different
  - [Last name], [First Name] vs. [First Name] [Last Name]

Conclusion

- Head in the clouds | Can't see snowflakes and sparks from inside cave
- Cuts both ways | Information loss can occur when (un)structuring
- Rosetta stone | Semi-structure provides multi-useful go-between
- Deceptively simple | Easy to start, tougher to scale
- Keys to success | Better leverage partnerships and open-source

Appendix – Downloading the triangle

```
# load libraries and data sets
library(dplyr)
library(insuranceData)
library(data.table)
library(jsonlite)
library(httr)

# render Schedule P
data(IndustryAuto)
for (ctr in 1:max(IndustryAuto$Development.Year)) {
 Triangle1<-IndustryAuto%>%
   filter(Development.Year==ctr)%>%
   group_by(Incurral.Year)%>%
    summarize_at(c('Claim'),sum)%>%
   setnames(old=c('Claim'),
          new=c(paste(sprintf('%03d',ctr*12),sep='')))
 if (ctr==1) {Triangle<-Triangle1} else {Triangle<-Triangle%>%left_join(Triangle1)}
 rm(Triangle1)
}
```

Appendix – Different JSON commands

```
# create a JSON
report<-'Schedule P'
subreport <-'Part 3b'
coverage<-'PPA'
subcoverage<-'Liability'
reportDate<-'2004-12-31'

myJSON<-as.data.frame(t(as.data.frame(c(report,subreport,subcoverage,reportDate))))%>%
  rename(report=1,subreport=2,subcoverage=3,reportDate=4)
rownames(myJSON)<-NULL
myJSON$Triangle<-list(Triangle)
TriangleJ<-toJSON(myJSON)

# render Schedule P
TriangleJ_parse<-fromJSON(TriangleJ)
TriangleJ_parse_SchedP<-as.data.frame(TriangleJ_parse[['Triangle']])
```