PollyVote R-Paketentwicklung

Lastenheft, Version 1.1

Inhalt

Überblick

- Github: https://github.com/pollyvote/pollyvote-r
- Email-Adressen für Nachfragen
  - Mario: mario.haim@ifkw.lmu.de
  - Andreas: graefe.andreas@gmail.com

Das Paket soll möglichst sauber und nachvollziehbar sein. Im Idealfall ist das Paket bereit, um bei CRAN initial eingereicht zu werden. Funktionen sollen ordentlich (und in englischer Sprache) dokumentiert sein. Es soll außerdem den gängigen R-Praktiken möglichst nahe kommen. Mit anderen Worten: Wenn eine der hier vorliegenden Spezifikationen komisch oder nicht sehr verbreitet, unlogisch oder unvollständig erscheint, dann bitte melden/nachfragen/mitdenken.

Ist irgendeine Struktur unlogisch oder fehlt eine Angabe oder ähnliches, soll das Paket mit den R-üblichen Warnungen arbeiten.

Datenspezifikation

Grundsätzlich soll das R-Paket einen Ansatz ähnlich dem verfolgen, der etwa von Paketen zur semantischen Textanalyse oder – ganz banal – der Regression genutzt wird. In einem ersten Schritt wird also spezifiziert und Daten eingelesen. Das zurückgegebene Objekt beinhaltet dann die Daten in einem Format, das als *Datencontainer* für alle weiteren Funktionen dient.

Das Paket soll so aufgebaut sein, dass jeweils eine Wahl (z.B. US-Präsidentschaftswahl, Bundestagswahl …) mit einem solchen Datenformat abgedeckt wird. Allerdings beziehen sich die Daten tw. auf vergangene Wahljahre, die in der Analyse berücksichtigt werden, sodass das Paket mit unterschiedlichen Wahljahren ein und derselben Wahl (z.B. 2008/2012/2016 der US-Präsidentschaftswahl oder 2013/2017 der BTW) umgehen können muss.

Dabei können durchaus Werte oder ganze Komponenten in einem Wahljahr fehlen. So ist etwa möglich, dass für die US-Präsidentschaftswahl im Jahr 2012 fünf Hauptkomponenten (Umfragen, Erwartungsfragen, statistische Modelle, Prognosemärkte, Experten), im Jahr 2016 aber sechs Hauptkomponenten (mit den statistischen Modellen aufgeteilt in ökonometrische und Index-Modelle) verfügbar sind.

Konkret müssen innerhalb des Datenformats, also innerhalb eines solchen Datencontainers, sechs Dinge spezifiziert werden (Beispieldaten sind hier teilweise fiktiv, also das Datum der BTW stimmt z.B. nicht unbedingt): Parteien, Wahljahre/-tage, Regionen, Komponenten, Prognosedaten und Wahlergebnisse.

Parteien

Dabei handelt es sich schlicht um die Namen der einzelnen Parteien sowie um die Information, ob eine Restkategorie („sonstige“) vorhanden ist.

Beispiel USA: Demokraten, Republikaner
Beispiel D: CDU/CSU, SPD, Grüne, Linke, AfD, FDP, sonstige

Wahljahre und Wahltag

Da viel mit der Größe „Tage bis zur Wahl“ gerechnet wird, ist hier das Datum je Wahl nötig. Prognostiziert wird außerdem meist die aktuelle Wahl, allerdings sind oft Daten aus früheren Daten

für Vergleiche der Prognosen nötig. Entsprechend muss spezifiziert werden, welche Wahljahre es im vorhandenen Datenpaket gibt.

Beispiel USA: 06.11.2008, 06.11.2012, 08.11.2016 (prognostiziert)
Beispiel D: 20.09.2013, 22.09.2017 (prognostiziert)

Wahlregionen und zugehörige Spezifika

Keine Pflichtangabe. Sind Regionen spezifiziert, kann (!) sich jede Komponente und jede Prognose darauf beziehen, muss es aber nicht. Es gibt in jedem Fall eine darüber liegende (meist nationale, bei Landtagswahlen kann das aber abweichen) Ebene (wenn nicht, dann wären keine Unterregionen nötig). Die Wahlregionen sind nur nötig, wenn das Wahlsystem vorsieht, nicht als Verhältniswahl, sondern z.B. als Mehrheitswahl ausgezählt zu werden. Spezifiziert werden müssen in dem Fall die Regionen, ein Gewicht pro Region, mit dem die Region in das übergeordnete (z.B. nationale) Ergebnis einfließt, sowie der Modus, mit dem je Region ausgezählt wird.

Bei einer aus Regionen zusammengestückelten Wahl ergibt sich die übergeordnete Ebene immer aus der Aggregation der Regionen (`aggregateRegions`), bei allen anderen Wahlen immer aus der Aggregation der anderen Komponenten (`aggregate`). Im US-Fall machen wir gerne beides, prognostizieren also den Popular und den Electoral Vote. In diesem Fall würden wir zwei Datencontainer instanzieren; einen, der den Popular Vote (ohne Regionen) und einen, der den Electoral Vote (über Regionen hinweg) beinhaltet.

Beispiel USA: prognostiziert wird die Präsidentschaftswahl auf nationaler Ebene, sie setzt sich aber aus dem Electoral Vote auf Staatenebene zusamme; es werden also alle Staaten als Wahlregionen spezifiziert, die jeweils mit einer spezifischen Zahl an Wahlmännern (Anzahl Wahlmänner = Gewicht) in das nationale Ergebnis einfließen, wobei der Modus in den meisten Staaten Winner-takes-all, in wenigen Staaten proportionale Verteilungen vorsieht. Hinterlegt wäre also zB Alabama (12 Wahlmänner, winner-takes-all), Texas (53, winner-takes-all), Nebraska (14, proportional) …

Beispiel Großbritannien: prognostiziert wird das House of Commons auf nationaler Ebene, das sich aber aus Einmandatswahlkreisen zusammensetzt; spezifiziert werden als Wahlregionen also alle 650 Wahlkreise, die alle das Gewicht von 1 und den Modus winner-takes-all erhalten.

Komponentenstruktur (das PollyVote-Modell)

Die PollyVote-Prognose speist sich aus mehreren Komponenten, die selbst wiederum aus Subkomponenten bestehen, die selbst wiederum aus Subkomponenten bestehen können, die selbst wiederum … nun ja, also jedenfalls liegt immer eine umgekehrte Baumstruktur (also streng hierarchisch) dem Ganzen zugrunde, wobei PollyVote immer der Ausgangspunkt (die Wurzel) darstellt. Diese Struktur ist essentiell und kann je Wahljahr variieren. Da jede Komponente gleichzeitig eine Subkomponente sein kann, sprechen wir im weiteren Verlauf nur noch von Komponenten, was alles meint, außer der PollyVote-Wurzel.

Abbildung 1: PollyVote Beispiel-Variante A

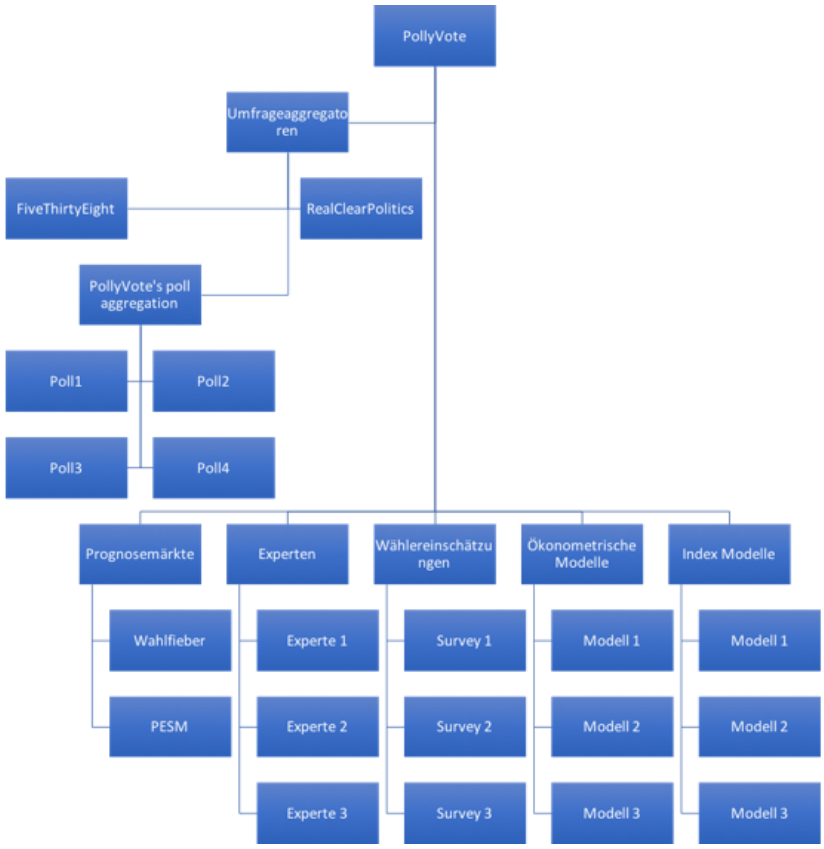
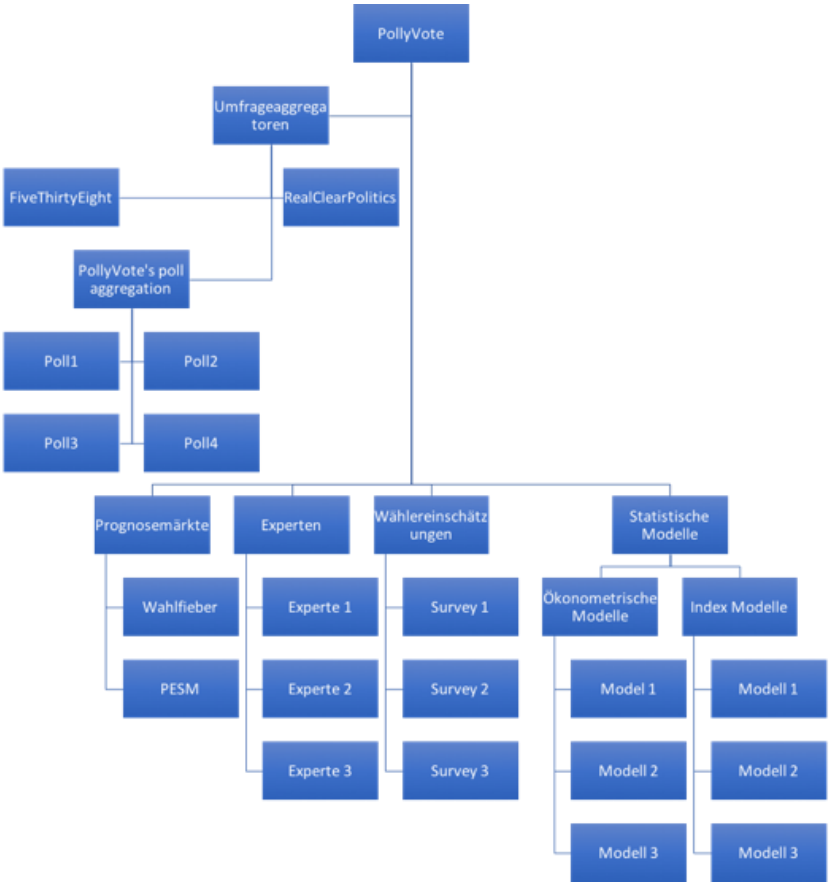


Abbildung 2: PollyVote Beispiel-Variante B



Prognosedaten (je Komponente und Wahljahr)

Für jede Komponente und für die PollyVote-Wurzel gibt es Prognosedaten im Zeitverlauf für jede Partei. Die feinste Abstufung stellt dabei der einzelne Tag dar. Prognosedaten sind also einer bestimmten Komponente (oder der PollyVote-Wurzel), einem bestimmten Datum (und damit einem bestimmten Wahljahr) und einer bestimmten Partei zugeordnet.

Beispiel USA: Experten-Komponente, 23.10.2016, Demokraten: 54%
Beispiel D: Poll2, 03.04.2017, SPD: 31%

Wahlergebnisse (je Wahljahr)

Für jedes Wahljahr (auch das aktuelle) gibt es zu einem bestimmten Zeitpunkt das amtliche Ergebnis je Partei. Das wird hier hinterlegt (in der Form von Stimmenanteilen je Partei). Zusätzlich wird hier hinterlegt, zu welchem tatsächlichen „Siegerverhältnis“ es kam, also wer die Koalition in Mehrparteiensystemen bzw. wer als tatsächlicher Gewinner (in Systemen wie den USA) hervorging. Entsprechend wird hier je Partei der tatsächliche Stimmenanteil sowie ein bool’scher Wert, ob die Partei die Regierung stellt oder nicht, hinterlegt.

Achtung, generell gilt hier (und überall im Folgenden): Die Parteienzugehörigkeit muss sich mit den Angaben bei den *Parteien* (siehe oben) decken, sodass über den Namen, wie er bei *Parteien* (oben) angegeben wurde, hier auch zugegriffen werden kann.

Beispiel USA:

- 2012: Demokraten: 52.42 (Regierung: 1), Republikaner: 47.58 (Regierung: 0)
- 2016: Demokraten: 51.20 (Reg.: 0), Republikaner: 48.80 (Reg.: 1)

Beispiel D:

- 2013: CDU/CSU: 26 (Reg.: 1), SPD: 21 (Reg.: 1), FDP: 4.9 (Reg.: 0), …

Daten importieren

Daten liegen meist als Excel- oder CSV-Dateien vor, was nicht weiter relevant für das vorliegende R-Paket sein soll. Importiert wird aus den üblichen R-Formaten, etwaige `read.table`-Befehle erfolgen außerhalb des Pakets. Da die Daten nicht in einer einzelnen Datei vorliegen, sind mehrere Import-Funktionen vonnöten, die auch das spätere Hinzufügen neuer einzelner Daten abdecken sollen.

setParties (parties, data = null, remaining = null)

Setzt alle prognostizierten Parteien. Diese Funktion überschreibt vorhergehende Parteien.

- **parties**: Vektor mit Parteinamen, die verwendet werden sollen (ohne „sonstige“).
- **data**: Der aktuelle Datencontainer. Wenn der fehlt, dann wird ein neuer Datencontainer erzeugt.
- **remaining**: Wenn nicht null (default), dann wird ein String erwartet, der den Namen der Restekategorie benennt (z.B. „sonstige“). Wird hier nichts (null) angegeben, gibt es keine Restekategorie.

Rückgabe: Aktualisierter Datencontainer.

Beispiel USA:
```
setParties(c('Demokraten', 'Republikaner'))
```

Beispiel D:
```
setParties(c('CDU/CSU', 'SPD', 'Linke', 'Grüne', 'AfD', 'FDP'), remaining='
sonstige')
```

setRegions (regions, data = null, weights = null, methods = null, election = null)

Definiert (neben der übergreifenden) Ebene die einzelnen Regionen.

- **regions**: Vektor mit den Namen der Regionen.
- **data**: Der aktuelle Datencontainer. Wenn der fehlt, dann wird ein neuer Datencontainer erzeugt.
- **weights**: Optionaler Vektor. Wenn gegeben, stehen hier in der gleichen Reihenfolge wie in regions die zugehörigen Gewichte (zB Electoral Votes). Wird weight nicht angegeben, wird für jede region ein Gewicht von 1 angenommen.
- **methods**: Optionale. Vektor oder String. Wenn angegeben, stehen hier entweder als Vektor in der gleichen Reihenfolge wie in regions oder als für alle zutreffender String die zugehörigen Methoden. Kann „wta“ (winner-takes-all) oder „vs“ (vote-share, also proportional) sein.
- **election**: Optionale Spezifikation des Wahljahres über den entsprechenden Namen (siehe `addElection`). Wenn nicht näher angegeben, wird die Regionsspezifizierung für alle Wahljahre übernommen.

Rückgabe: Aktualisierter Datencontainer.

Beispiel USA:
```
setRegions(c('AL', 'NY', 'TX'), weights=c(12, 30, 53), methods='wta')
setRegions(c('AL', 'NY', 'TX'), weights=c(12, 30, 53), methods=c('wta',
'vs', 'wta'))
```

addElection (name, date, data = null, result.vs = null, result.gov = null, result.parties = null)

Fügt dem Datencontainer ein neues Wahljahr hinzu, wobei alle Wahljahre einzeln hinzugefügt werden. Jedes Wahljahr kann (muss aber nicht) ein Wahlergebnis erhalten.

Achtung, hier wird das erste Mal in diesem Skript auf ein Datum verwiesen. Wir brauchen das immer nur tagesgenau – was ist hier für ein Format sinnvoll? PosixCT?

- **name**: Ein eindeutiger Name, mit dem das Wahljahr identifiziert wird. Meist das Jahr.
- **date**: Das tagesgenaue Datum der Wahl.
- **data**: Der aktuelle Datencontainer. Wenn der fehlt, dann wird ein neuer Datencontainer erzeugt.
- **result.vs**: Vektor mit Stimmanteilen. Entweder benannt oder in der Reihenfolge, die in result.parties mitgegeben wird.
- **result.gov**: Vektor mit bool’schen Anzeigern, ob eine Partei die Regierung im Anschluss stellt. Entweder benannt oder in der Reihenfolge, die in result.parties mitgegeben wird.
- **result.parties**: Optionaler Vektor, der die Reihenfolge der Parteien, in der result.vs und result.gov übermittelt werden, festlegt.

Rückgabe: Aktualisierter Datencontainer.

Beispiel USA:
```
addElection('POTUS16', as.POSIXct('2016-11-08'),
```

```
result.vs=c('Demokraten'=51, 'Republikaner'=49),
result.gov=c('Demokraten'=F, 'Republikaner'=T))
```

Beispiel D:

```
addElection('2017', as.POSIXct('2017-09-22'))
```

```
addElection('BTW 2013', as.POSIXct('2013-09-20'), result.vs=c(32, 30, 9, 9,
8.5, 4.9, 2.2), result.gov=c(T,T,F,F,F,F,F), result.parties=c('CDU/CSU',
'SPD', 'Linke', 'Grüne', 'AfD', 'FDP', 'sonstige'))
```

addComponent (data, component, parent = 'pollyvote', values = null, region = null)

Fügt eine einzelne Komponente (ggf. mitsamt zugehöriger Daten) in das PollyVote-Modell ein.

- **data**: Der Datencontainer.
- **component**: Der Name der neuen Komponente.
- **parent**: Die zugehörige Eltern-Komponente. Wenn nicht näher spezifiziert, wird das Wurzelelement (PollyVote) angenommen.
- **values**: Optionale Angabe von Daten. Data-Frame mit dem Datum als erste Spalte (oder row.names, was macht hier mehr Sinn?), den Parteien als Spalten (oder erste Zeile, was macht hier mehr Sinn?) und den Werten als Stimmenanteile.
- **region**: Wenn angegeben (und wenn die Region spezifiziert ist), werden Daten für diese Region eingefügt. Wenn angegeben, aber die Region ist nicht spezifiziert, wird ein Fehler erzeugt. Default ist immer die übergreifende Ebene (das bedeutet meist: national).

Rückgabe: Aktualisierter Datencontainer.

Beispiel:

```
forsa <- data.frame(CDU=c(42,41,41), SPD=c(33,34,34),
row.names=as.POSIXct(c('2017-09-19', '2017-09-20', '2017-09-21')))
pv <- setParties(c('CDU/CSU', 'SPD', 'Linke', 'Grüne', 'AfD', 'FDP'),
remaining='sonstige')
pv <- addElection('2017', as.POSIXct('2017-09-22'), data=pv)
pv <- addComponent(pv, 'Umfragen')
pv <- addComponent(pv, 'Forsa', forsa, 'Umfragen')
```

setComponents (data, ???, region = null)

Achtung, hier sind wir noch unschlüssig. Es handelt es sich hierbei sozusagen um eine formale Alternative zu `addComponent`, indem hier zunächst ein gesamtes PollyVote-Modell spezifiziert wird, bevor anschließend Daten hinzugefügt werden (über `setValues`). Entsprechend werden hier keine Werte mitgegeben. Spezifiziert werden muss nur die Baumstruktur des PollyVote-Modells, also welche Komponentennamen gibt es und in welcher Beziehung die zueinander stehen. Werte werden anschließend über einen separaten Aufruf (bzw. mehrere Aufrufe) von `setValues` hinzugefügt.

Gibt es für die Baumstruktur eine geeignete Syntax, die einerseits leicht verständlich und gut lesbar, andererseits auch gut „teilbar“ ist, also dass man zB in einem Blogpost die Syntax für das von uns vorgeschlagene PollyVote-Modell der französischen Präsidentschaftswahl für copy&paste bereitstellt?

- **data**: Der Datencontainer.
- **???:** Modellspezifikation

- **region**: Wenn angegeben (und wenn die Region spezifiziert ist), werden Daten für diese Region eingefügt. Wenn angegeben, aber die Region ist nicht spezifiziert, wird ein Fehler erzeugt. Default ist immer die übergreifende Ebene.

Rückgabe: Aktualisierter Datencontainer.

Ungefähres Beispiel, genaue Syntax nicht durchdacht:

```
pv <- setParties(c('CDU/CSU', 'SPD', 'Linke', 'Grüne', 'AfD', 'FDP'),
remaining='sonstige')
pv <- addElection('2017', as.POSIXct('2017-09-22'), data=pv)
pv <- setComponents(pv, 'pv ~ (Umfragen ~ Forsa, Emnid), (Experten ~
PollyVote-Panel)')
```

setValues (data, component, values, region = null)

Fügt keine neue Komponente hinzu, sondern füllt eine bestehende mit Daten. Besteht die Komponente nicht, wird ein Fehler erzeugt. Ansonsten wie `addComponent`, wobei values eine Pflichtangabe darstellt.

- **data**: Der Datencontainer.
- **component**: Der Name der neuen Komponente.
- **values**: Data-Frame mit dem Datum als erste Spalte (oder row.names, was macht hier mehr Sinn?), den Parteien als Spalten (oder erste Zeile, was macht hier mehr Sinn?) und den Werten als Stimmenanteile.
- **region**: Wenn angegeben (und wenn die Region spezifiziert ist), werden Daten für diese Region eingefügt. Wenn angegeben, aber die Region ist nicht spezifiziert, wird ein Fehler erzeugt. Default ist immer die übergreifende Ebene.

Rückgabe: Aktualisierter Datencontainer.

Daten aggregieren

Sind die Daten eingelesen, liegt also ein vollständiger Datencontainer vor, kann darin aggregiert werden. Aggregiert wird immer auf Komponentenebene, die sich aus mehreren (oder allen) Subkomponenten speist.

aggregate (data, method, na.handle = 'last', component = 'pollyvote', subcomponents = null, region = null, …)

Übertragen auf das PollyVote-Modell (Abbildungen 1 und 2), wird für jeden Pfad in der Baumstruktur ein aggregate-Befehl ausgeführt. Das wird manuell je Pfad ausgeführt, weil sich das auch ändern kann. Die `aggregate`-Funktion aggregiert (ausgewählte) Subkomponenten je Tag in die angegebene Komponente; dabei kann es vorkommen, dass für einzelne Subkomponenten einzelne Tageswerte einzelner Parteien fehlen.

Ein erklärendes Beispiel: Für die Kompontente "Umfragen" werden die Subkomponenten "Forsa" und "Emnid" per Mittelwert aggregiert. Das heißt, dass für alle Tage, an denen für mindestens eine der beiden Subkomponenten Werte vorliegen, für jede Partei ein Mittelwert gebildet und in den Datencontainer an die passende Stelle geschrieben wird (für die entsprechende Partei am entsprechenden Tag in der entsprechenden Komponente).

- **data**: Der Datencontainer.

- **method**: Methode, mit der die Subkomponenten in die Komponente aggregiert werden. Erwartet eine Funktion, die so aussehen soll, dass sie einige der gängigen base-Funktionen (z.B. mean, median) aber auch eigene Funktionen (z.B. trimmed_mean, weighted_mean) entgegen nehmen kann.
- **na.handle**: Verfahren, wie mit fehlenden Werten umgegangen wird (zB bieten Forsa und Emnid Werte für die AfD an, Infratest fasst die AfD aber unter „sonstige“). Möglich sind hier verschiedene als String übermittelte Varianten (ggf. kommen noch mehr):
    - ***omit:*** Zeile wird nicht berücksichtigt, als Wert wird NA aggregiert
    - ***last***: nimm den letzten verfügbaren Wert für AfD bei Infratest und rechne damit
    - ***mean_within***: nimm den Mittelwert aus allen anderen Subkomponenten innerhalb der angegebenen Komponente für den fehlenden Wert (AfD/Infratest) und rechne damit
    - ***mean_across***: berechne eine kompletten PollyVote über alle Komponenten (also auch über statistische Modelle und über Experten …) und nimm den so entstandenen Wert für die AfD als fehlenden Wert (AfD/Infratest) und rechne damit
- **component**: Name der Komponente, die aggregiert werden soll. Wenn nicht näher spezifiziert, dann wird die Wurzelkomponente (PollyVote) berechnet.
- **subcomponents**: Möglichkeit, genau zu spezifizieren, welche Subkomponenten innerhalb einer Komponente aggregiert werden sollen. Standardmäßig (also wenn null), werden alle direkten Subkomponenten verwendet. Andernfalls wird ein Vektor erwartet, der die Namen der Subkomponenten enthält, die aggregiert werden sollen. Das dient dazu, beispielsweise innerhalb von Umfragen alle außer „Infratest Dimap“ zu aggregieren.
- **region**: Wenn angegeben (und wenn die Region spezifiziert ist), werden Daten für diese Region eingefügt. Wenn angegeben, aber die Region ist nicht spezifiziert, wird ein Fehler erzeugt. Default ist immer die übergreifende Ebene.
- **…**: für method nötige weitere Parameter

Achtung, wenn als Region die übergreifende Ebene und die gewählte Komponente gleichzeitig PollyVote ist, also auf übergreifender (z.B. nationaler) Ebene aus den Regionen in die Gesamtprognose aggregiert werden soll, wird ein Fehler generiert. In diesem Fall ist die Funktion `aggregateRegions` nötig, die die Gewichte und Methoden der einzelnen Regionen berücksichtigt.

Rückgabe: Aktualisierter Datencontainer.

Beispiel:

```
forsa <- data.frame(CDU=c(42,41,41), SPD=c(33,34,34),
row.names=as.POSIXct(c('2017-09-19', '2017-09-20', '2017-09-21')))
emnid <- data.frame(CDU=c(40,40,39.6), SPD=c(34,35,34),
row.names=as.POSIXct(c('2017-09-19', '2017-09-20', '2017-09-21')))
pv <- setParties(c('CDU/CSU', 'SPD', 'Linke', 'Grüne', 'AfD', 'FDP'),
remaining='sonstige')
pv <- addElection('2017', as.POSIXct('2017-09-22'), data=pv)
pv <- addComponent(pv, 'Umfragen')
pv <- addComponent(pv, 'Forsa', forsa, 'Umfragen')
pv <- addComponent(pv, 'Emnid', emnid, 'Umfragen')

#Variante 1
pv <- aggregate(pv, 'Umfragen', mean)

#Variante 2
pv <- aggregate(pv, 'Umfragen', mean, na.handle='mean_across')
```

```
#Variante 3
rolling_average <- function( ... )
pv <- aggregate(pv, 'Umfragen', rolling_average)
```

aggregateRegions (data, regions = null, na.handle = 'last')

Dient der Aggregation der PollyVote-Werte (Wurzelknoten) innerhalb der Regionen in die übergreifende Prognose, die sich über die Gewichte (weights) und entsprechenden Methoden (methods; jeweils aus `setRegions`) ergeben.

- **data**: Der Datencontainer.
- **region**: Optionaler Vektor, der die Auswahl der einzubeziehenden Regionen einschränken kann. Wenn angegeben, werden nur die hier angegebenen Regionen berücksichtigt.
- **na.handle**: Verfahren, wie mit fehlenden Werten umgegangen wird (zB gibt es keinen PollyVote-Wert für Alabama, jedoch für alle anderen Bundesstaaten). Möglich sind hier verschiedene als String übermittelte Varianten (ggf. kommen noch mehr):
  - ***omit:*** Zeile wird nicht berücksichtigt, als Wert wird NA aggregiert
  - ***last***: nimm den letzten verfügbaren Wert für Alabama und rechne damit weiter
  - ***mean_across***: nimm den PollyVote-Mittelwert aus allen anderen Staaten für Alabama und rechne damit weiter

Rückgabe: Aktualisierter Datencontainer.

Beispiel:

```
pv <- setParties(c('D', 'R'))
pv <- setRegions(c('AL', 'NY', 'WY'), pv, c(20, 30, 15), 'wta')
pv <- addElection('2016', as.POSIXct('2016-11-08'), pv, c(51, 49), c(F, T),
c('D', 'R'))
polls_al <- data.frame(D=c(52,53,54), R=c(48,47,46),
row.names=as.POSIXct(c('2016-10-01', '2016-10-02', '2016-10-03')))
polls_ny <- data.frame(D=c(50,50,50), R=c(50,50,50),
row.names=as.POSIXct(c('2016-10-01', '2016-10-02', '2016-10-03')))
polls_wy <- data.frame(D=c(NA,49,49), R=c(NA,51,51),
row.names=as.POSIXct(c('2016-10-01', '2016-10-02', '2016-10-03')))
pv <- addComponent(pv, 'Polls', values=polls_al, region='AL')
pv <- addComponent(pv, 'Polls', values=polls_ny, region='NY')
pv <- addComponent(pv, 'Polls', values=polls_wy, region='WY')
experts_al <- data.frame(D=c(53,53,53), R=c(47,47,47),
row.names=as.POSIXct(c('2016-10-01', '2016-10-02', '2016-10-03')))
experts_ny <- data.frame(D=c(58,58,58), R=c(42,42,42),
row.names=as.POSIXct(c('2016-10-01', '2016-10-02', '2016-10-03')))
experts_wy <- data.frame(D=c(48,48,49), R=c(52,52,51),
row.names=as.POSIXct(c('2016-10-01', '2016-10-02', '2016-10-03')))
pv <- addComponent(pv, 'Experts', values=experts_al, region='AL')
pv <- addComponent(pv, 'Experts', values=experts_ny, region='NY')
pv <- addComponent(pv, 'Experts', values=experts_wy, region='WY')

pv <- aggregate(pv, mean, region='AL')
pv <- aggregate(pv, mean, region='NY')
pv <- aggregate(pv, mean, region='WY')

#Variante 1
pv <- aggregateRegions(pv)

#Variante 2
pv <- aggregateRegions(pv, c('NY', 'WY'))
```

```
#Variante 3
pv <- aggregateRegions(pv, na.handle='mean_across')
```

Daten analysieren

Diese Funktionen verändern den Datencontainer nicht. Stattdessen werden nur die analysierten/errechneten Daten zurückgegeben. Im Gegensatz zu den exportierten Daten werden aber tatsächlich neue Dinge (z.B. Fehler) berechnet.

calculateError (data, component, method, election = null, region = null, …)

Fehler beziehen sich immer auf ein Wahlergebnis und damit auf ein Wahljahr. Zu welchem Wahljahr welche Daten gehören, ergibt sich aus dem Datum einer Prognose und dem Datum eines Wahlergebnisses. Für ein noch laufendes Wahljahr können keine Fehler berechnet werden.

- **data**: Der Datencontainer.
- **component**: Komponente, für die Fehler berechnet werden sollen
- **method**: Methode, mit der gerechnet wird. Hier wird eine Funktion erwartet, die idealerweise wieder aus anderen Paketen übernommen (z.B. MAPE) oder selbst spezifiziert (z.B. MAE) werden kann.
- **election**: Das Wahljahr, für das berechnet werden soll. Wenn nicht spezifiziert, wird das aktuellste Wahljahr verwendet, für das Wahlergebnisse vorliegen (und eine Warnung wird erzeugt).
- **region**: Wenn angegeben (und wenn die Region spezifiziert ist), werden Daten für diese Region eingefügt. Wenn angegeben, aber die Region ist nicht spezifiziert, wird ein Fehler erzeugt. Default ist immer die übergreifende Ebene.
- **…**: für method nötige weitere Parameter

Rückgabe: Benannter Vektor mit dem Datum als Namen (macht das Sinn?) und dem errechneten Fehler als Wert.

Achtung, da für fehlende Werte kein Fehler berechnet werden kann, gibt es hier mitunter NA-Werte in der Rückgabe.

Beispiel:

```
forsa <- data.frame(CDU=c(42,41,41), SPD=c(33,34,34),
row.names=as.POSIXct(c('2017-09-17', '2017-09-18', '2017-09-19')))
emnid <- data.frame(CDU=c(40,40,39.6), SPD=c(34,35,34),
row.names=as.POSIXct(c('2017-09-17', '2017-09-18', '2017-09-19')))
pv <- setParties(c('CDU', 'SPD'))
pv <- addElection('2013', as.POSIXct('2013-09-20'), pv, c(CDU=41, SPD=35))
pv <- addComponent(pv, 'Umfragen')
pv <- addComponent(pv, 'Forsa', forsa, 'Umfragen')
pv <- addComponent(pv, 'Emnid', emnid, 'Umfragen')
pv <- aggregate(pv, 'Umfragen', mean)

mae <- function( ... )
calculateError(pv, 'Umfragen', mae)

#Ergebnis:
#CDU (41) 42/40=41, SPD (35) 33/34=33.5 = (0+1.5)/2
#CDU (41) 41/40=40.5, SPD (35) 34/35=34.5 = (.5+.5)/2
#CDU (41) 41/39.6=40.3, SPD (35) 34/34=34 = (.7+1)/2
```

```
c('2017-09-19'=.75, '2017-09-20'=.5, '2017-09-21'=.85)
```

calculateCI (data, component, method, elections = null, ci = .95, region = null, …)

Berechnet Konfidenzintervalle auf Basis historischer Prognosefehler. Die Berechnung erfolg dabei je Partei und Tag für die angegebene Komponente. Das Vorgehen ist dabei folgendermaßen:

1. Für jedes angegebene Wahljahr, berechne den Prognosefehler der angegebenen Komponente für alle Tage (via `calculateError(data, component, method, election, region)`).
2. Berechne das arithmetische Mittel über alle elections für jeden Tag und jede Partei.
3. Das Konfidenzintervall je Tag ergibt sich dann aus diesem arithmetischen Mittel M, dem zum CI gehörenden Z-Wert (bei 95% ist das 1.96) und der folgenden Formel:
   CI = 1.25 * M * Z

Später wird dieses Konfidenzintervall dann +/- um die Prognose gebastelt.

- **data**: Der Datencontainer.
- **component**: Komponente, für die die Konfidenzintervalle berechnet werden sollen
- **method**: Methode, mit der der Fehler berechnet wird. Hier wird eine Funktion erwartet, die idealerweise wieder aus anderen Paketen übernommen (z.B. MAPE) oder selbst spezifiziert (z.B. MAE) werden kann.
- **elections**: Optionaler Vektor, der die Wahljahre spezifiziert, die als Basis für den historischen Prognosefehler herhalten. Wenn nicht näher spezifiziert, werden alle Wahljahre berücksichtigt, die zeitlich vor der angegebenen election liegen. Diese zeitliche Ordnung ist kein Muss-Kriterium; mit anderen Worten: Durch manuelle Spezifikation ist durchaus möglich, als vergangene Wahlen auch Wahlen zu berücksichtigen, die eigentlich zeitlich nach der „election“ stattfanden.
- **ci**: Das Konfidenzintervall, das berechnet werden soll. Standard ist das 95%-Intervall.
- **region**: Wenn angegeben (und wenn die Region spezifiziert ist), werden Daten für diese Region eingefügt. Wenn angegeben, aber die Region ist nicht spezifiziert, wird ein Fehler erzeugt. Default ist immer die übergreifende Ebene.
- **…**: für method nötige weitere Parameter

Rückgabe: Benannter Vektor mit dem Datum als Namen (macht das Sinn?) und dem Konfidenzintervall als Wert.

Achtung, da für fehlende Werte kein Fehler berechnet werden kann, gibt es hier mitunter Zellen in der Rückgabe, die mit NA gefüllt sind.

Beispiel (kopiert von `calculateError`*)*:

```
forsa <- data.frame(CDU=c(42,41,41), SPD=c(33,34,34),
row.names=as.POSIXct(c('2017-09-17', '2017-09-18', '2017-09-19')))
emnid <- data.frame(CDU=c(40,40,39.6), SPD=c(34,35,34),
row.names=as.POSIXct(c('2017-09-17', '2017-09-18', '2017-09-19')))
pv <- setParties(c('CDU', 'SPD'))
pv <- addElection('2013', as.POSIXct('2013-09-20'), pv, c(CDU=41, SPD=35))
pv <- addComponent(pv, 'Umfragen')
pv <- addComponent(pv, 'Forsa', forsa, 'Umfragen')
pv <- addComponent(pv, 'Emnid', emnid, 'Umfragen')
pv <- aggregate(pv, 'Umfragen', mean)
```

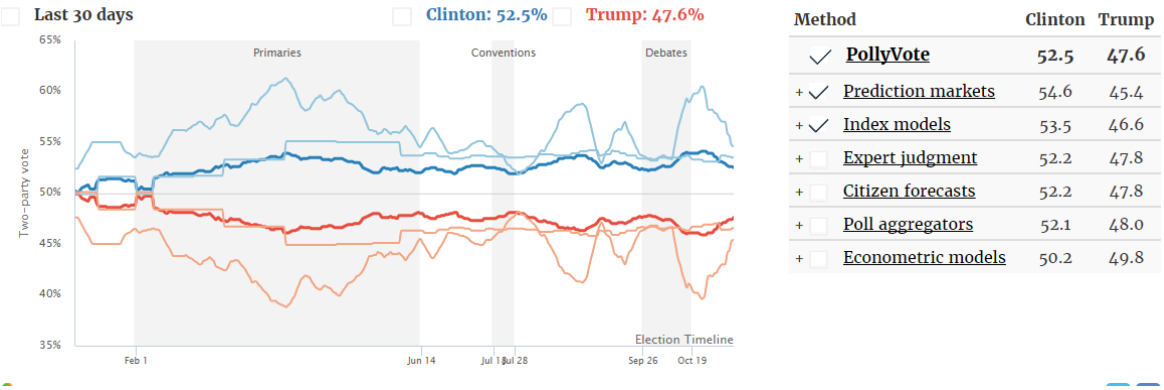
```
mae <- function( ... )
calculateCI(pv, 'Umfragen', mae)

#Ergebnis:
#Fehler wie im vorherigen Bsp.: c('2017-09-19'=.75, '2017-09-20'=.5, '2017-09-21'=.85)
#Formel: 1.25 * Fehler * 1.96

c('2017-09-19'=1.8375, '2017-09-20'=1.225, '2017-09-21'=2,0825)
```

Daten exportieren

Diese Funktionen verändern den Datencontainer nicht. Stattdessen wird nur eine neue Perspektive auf die Daten geworfen und diese Daten anschließend zurückgegeben. Die Export-Funktionen sind teilweise auch redundant; sie dienen primär der aufbereiteten Auslieferung der Daten für bestimmte Visualisierungen.

getVoteshares (data, component = 'pollyvote', election = null, party = null, region = null, limitdays = -1, for.ggplot2 = F)

Für die Standardvisualisierung der Komponenten im Zeitverlauf wird ein entsprechendes Datenformat zurückgegeben. Das Format orientiert sich an unseren standardmäßig verwendeten Tabellenformaten. Auf Wunsch wird stattdessen ein für ggplot2 vordefiniertes Format verwendet.



Method	Clinton	Trump
PollyVote	52.5	47.6
+ Prediction markets	54.6	45.4
+ Index models	53.5	46.6
+ Expert judgment	52.2	47.8
+ Citizen forecasts	52.2	47.8
+ Poll aggregators	52.1	48.0
+ Econometric models	50.2	49.8

- **data**: Der Datencontainer.
- **component**: Komponente, für die die Stimmenanteile zurückgegeben werden sollen. Die Komponentendefinition weicht hier etwas von den anderen Deklarationen ab, um dem Regelfall näher zu kommen. Standardwert ist die String-Angabe der Wurzel-Prognose, also „pollyvote“, auf übergreifender (d.h. nicht-regionaler) Ebener. Hier kann übergeben werden:
  - ***String***, also ein einzelner Komponentenname (z.B. „Umfrage“); in diesem Fall wird eine Spalte für die angegebene Komponente sowie je eine Spalte je (direkter) Subkomponente zurückgegeben
  - ***Vektor*** mit Komponentennamen; gibt genau für die angegebenen Komponenten (ohne automatische Subkomponentenergänzung) die Werte zurück
  - ***null***, eine Wildcard, bei der schlichtweg alle Komponenten zurückgegeben werden
- **election**: Das Wahljahr, für das berechnet werden soll. Wenn nicht spezifiziert, wird das aktuellste Wahljahr verwendet.
- **party**: Erlaubt die Auswahl bestimmter Parteien. Optional, wenn null, dann alle Parteien. Alternativ ist mehreres möglich:
  - ***String*** für die Angabe einer Partei

    - ***Vektor*** für die Angabe mehrerer Parteien
    - ***null*** für schlichtweg alle Parteien
- **region**: Wenn angegeben (und wenn die Region spezifiziert ist), werden Daten für diese Region eingefügt. Wenn angegeben, aber die Region ist nicht spezifiziert, wird ein Fehler erzeugt. Default ist immer die übergreifende Ebene.
- **limitdays**: Möglichkeit, in Tagen eine Grenze festzulegen, bis zu der die Daten zurückgeliefert werden sollen. Unlimitiert, wenn negativ/null (Default). So schränkt beispielsweise die Angabe von `limitdays=100` die Rückgabe auf 100 Tage vor der Wahl ein, bei einem Wahltermin am 24.09.2017 werden also nur Einträge nach dem 16.07.2017 zurückgegeben.
- **for.ggplot2**: Bool'sche Angabe, die über das Rückgabeformat entscheidet. Bei `FALSE` (Default) wird ein Data-Frame zurückgegeben, bei dem jede Zeile ein Datum widerspiegelt und der direkt als CSV exportiert werden kann. Bei `TRUE` wird indes ein Data-Frame zurückgegeben, bei dem Zeilen für Datenpunkte in der Visualisierung stehen (siehe Beispiel unten).

Rückgabe: Data-Frame mit den Stimmenanteilen je Partei je angeforderter Komponente sowie der Angabe des Datums und der Zahl der Tage bis zu Wahl. Nach DaysToElection (bzw. Date DESC) sortiert. Die Spaltenköpfe beginnen immer mit Date und DaysToElection und hängen in weiterer Folge von component ab; wenn mehrere Komponenten gefordert sind, ist der Spaltenname immer im Format Komponentenname/Partei, bei einer Komponente entfällt der erste Teil, die Partei steht immer da. Das tatsächliche Format der Tabelle hängt außerdem von `for.ggplot2` ab (siehe insbes. Beispiel 3 und 4).

Beispiel (pv angenommen):

```
getVoteshares(pv)
#Ergebnis (Spalten und Zeilen verkürzt)
#          Date DaysToElection PollyVote/CDU PollyVote/SPD Umfragen/CDU …
# 1  2017-09-24              0            41            34           41 …
# 2  2017-09-23              1            41            33           41 …
# 3  2017-09-22              2            41            32           41 …
# 4  2017-09-21              3            40            31           40 …
# 5  2017-09-20              4            41            32           41 …
# 6  2017-09-19              5            42            33           42 …
# …

getVoteshares(pv, c('pollyvote'), party='CDU')
#Ergebnis (Spalten vollständig, Zeilen verkürzt)
#          Date DaysToElection CDU
# 1  2017-09-24              0  41
# 2  2017-09-23              1  41
# 3  2017-09-22              2  41
# …

getVoteshares(pv, c('pollyvote'), limitdays=1)
#Ergebnis (vollständig)
#          Date DaysToElection CDU SPD
# 1  2017-09-24              0  41  34
# 2  2017-09-23              1  41  33

getVoteshares(pv, c('pollyvote'), limitdays=1, for.ggplot2=T)
#Ergebnis (vollständig)
#          Date DaysToElection  Component Party Voteshare
# 1  2017-09-24              0  PollyVote   CDU        41
```

```
# 2   2017-09-24              0   PollyVote   SPD         34
# 3   2017-09-23              1   PollyVote   CDU         41
# 4   2017-09-23              1   PollyVote   SPD         33
```

getSingleVoteshares (data, date, component = 'pollyvote', election = null, party = null, region = null)

Direktzugriff für `getVoteshares` an einem dezidierten Tag. Es entfällt entsprechend `limitdays` und `for.ggplot2`, stattdessen kommt der obligatorische Parameter `date` hinzu.

- **date**: Datum (Posixct?), für das der Datensatz aus `getVoteshares` zurückgegeben werden soll.

Rückgabe: Data-Frame mit den Stimmenanteilen je Partei je angeforderter Komponente. Spalten heißen immer Component, Party und Voteshare, die Aufbereitung erfolgt quasi immer `for.ggplot2`.

Beispiel (d'accord zu den Beispielen aus getVoteshares):

```
getSingleVoteshares(pv, as.POSIXct('2017-09-22'))
#Ergebnis (Zeilen verkürzt)
#    Component Party Voteshare
# 1  PollyVote   CDU        41
# 2  PollyVote   SPD        32
# 3   Umfragen   CDU        41
# …
```

```
getSingleVoteshares(pv, as.POSIXct('2017-09-22'), c('pollyvote'),
party='CDU')
#Ergebnis (vollständig)
#    Component Party Voteshare
# 1  PollyVote   CDU        41
```

```
getVoteshares(pv, as.POSIXct('2017-09-22'), c('pollyvote'))
#Ergebnis (vollständig)
#    Component Party Voteshare
# 1  PollyVote   CDU        41
# 2  PollyVote   SPD        32
```

getVoteshareErrors (data, method, component = 'pollyvote', election = null, party = null, region = null, limitdays = -1, for.ggplot2 = F)

Im Grunde wie `getVoteshares`, nur die Werte sind verschieden. Zurückgegeben werden keine Stimmenanteile, sondern berrechnete Fehler. Dazu wird `calculateError` mit der übergebenen Methode aufgerufen.

- **data**: Der Datencontainer.
- **method**: Methode, mit der gerechnet wird. Hier wird eine Funktion erwartet, die idealerweise wieder aus anderen Paketen übernommen (z.B. MAPE) oder selbst spezifiziert (z.B. MAE) werden kann.
- **component**: Komponente, für die die Stimmenanteile zurückgegeben werden sollen. Die Komponentendefinition weicht hier etwas von den anderen Deklarationen ab, um dem Regelfall näher zu kommen. Standardwert ist die String-Angabe der Wurzel-Prognose, also „pollyvote", auf übergreifender (d.h. nicht-regionaler) Ebene. Hier kann übergeben werden:

    - ***String***, also ein einzelner Komponentenname (z.B. „Umfrage“); in diesem Fall wird eine Spalte für die angegebene Komponente sowie je eine Spalte je (direkter) Subkomponente zurückgegeben
    - ***Vektor*** mit Komponentennamen; gibt genau für die angegebenen Komponenten (ohne automatische Subkomponentenergänzung) die Werte zurück
    - ***null***, eine Wildcard, bei der schlichtweg alle Komponenten zurückgegeben werden
- **election**: Das Wahljahr, für das berechnet werden soll. Wenn nicht spezifiziert, wird das aktuellste Wahljahr verwendet.
- **party**: Erlaubt die Auswahl bestimmter Parteien. Optional, wenn null, dann alle Parteien. Alternativ ist mehreres möglich:
    - ***String*** für die Angabe einer Partei
    - ***Vektor*** für die Angabe mehrerer Parteien
    - ***null*** für schlichtweg alle Parteien
- **region**: Wenn angegeben (und wenn die Region spezifiziert ist), werden Daten für diese Region eingefügt. Wenn angegeben, aber die Region ist nicht spezifiziert, wird ein Fehler erzeugt. Default ist immer die übergreifende Ebene.
- **limitdays**: Möglichkeit, in Tagen eine Grenze festzulegen, bis zu der die Daten zurückgeliefert werden sollen. Unlimitiert, wenn negativ/null (Default). So schränkt beispielsweise die Angabe von `limitdays=100` die Rückgabe auf 100 Tage vor der Wahl ein, bei einem Wahltermin am 24.09.2017 werden also nur Einträge nach dem 16.07.2017 zurückgegeben.
- **for.ggplot2**: Bool’sche Angabe, die über das Rückgabeformat entscheidet. Bei `FALSE` (Default) wird ein Data-Frame zurückgegeben, bei dem jede Zeile ein Datum widerspiegelt und der direkt als CSV exportiert werden kann. Bei `TRUE` wird indes ein Data-Frame zurückgegeben, bei dem Zeilen für Datenpunkte in der Visualisierung stehen (siehe Beispiel unten).
- **…**: für method nötige weitere Parameter

Rückgabe: Data-Frame mit den Fehlern über die Parteien hinweg je angeforderter Komponente sowie der Angabe des Datums und der Zahl der Tage bis zu Wahl. Nach DaysToElection (bzw. Date DESC) sortiert. Die Spaltenköpfe beginnen immer mit Date und DaysToElection, daran anschließend hängen die einzelnen angeforderten Komponenten. Das tatsächliche Format der Tabelle hängt außerdem von `for.ggplot2` ab (siehe Beispiele 1 und 2).

Beispiel (pv und mae angenommen):

```
getVoteshareErrors(pv, mae)
#Ergebnis (Spalten und Zeilen verkürzt)
#          Date DaysToElection PollyVote Umfragen Experten …
# 1  2017-09-24              0       2.0      3.1      2.1 …
# 2  2017-09-23              1       2.1      3.2      2.2 …
# 3  2017-09-22              2       2.0      3.1      2.1 …
# 4  2017-09-21              3       2.0      3.1      2.1 …
# 5  2017-09-20              4       2.0      3.1      2.1 …
# 6  2017-09-19              5       2.0      3.1      2.1 …
# …

getVoteshareErrors(pv, mae, for.ggplot2=T)
#Ergebnis (Zeilen verkürzt)
#          Date DaysToElection  Component Error
# 1  2017-09-24              0  PollyVote   2.0
# 2  2017-09-24              0   Umfragen   3.1
```

```
# 3  2017-09-24              0   Experten   2.1
# 1  2017-09-23              1  PollyVote   2.1
# 2  2017-09-23              1   Umfragen   3.2
# 3  2017-09-23              1   Experten   2.2
# …
```

getSingleVoteshareErrors (data, method, date, component = 'pollyvote', election = null, party = null, region = null)

Direktzugriff für `getVoteshareErrors` an einem dezidierten Tag. Es entfällt entsprechend `limitdays` und `for.ggplot2`, stattdessen kommt der obligatorische Parameter `date` hinzu.

- **date**: Datum (Posixct?), für das der Datensatz aus `getVoteshareErrors` zurückgegeben werden soll.

Rückgabe: Data-Frame mit den Fehlern je angeforderter Komponente. Spalten heißen immer Component und Error, die Aufbereitung erfolgt quasi immer `for.ggplot2`.

Beispiel (d'accord zu den Beispielen aus getVoteshareErrors):

```
getSingleVoteshareErrors(pv, mae, as.POSIXct('2017-09-22'))
#Ergebnis (Zeilen verkürzt)
#    Component Error
# 1  PollyVote   2.0
# 2   Umfragen   3.1
# 3   Experten   2.1
# …
```

getVoteshareErrorCIs (data, method, ci = .95, component = 'pollyvote', election = null, party = null, region = null, limitdays = -1, for.ggplot2 = F)

Funktioniert genau wie getVoteshareErrors, ergänzt aber je Komponentenspalte noch zwei weitere Spalten zur Angabe des unteren und oberen Konfidenzintervallgrenzwerts. Die Berechnung erfolgt über `calculateCI`, was dann zum jeweiligen Wert addiert bzw. von selbigem subtrahiert wird. Die Spaltennamen warden dabei um das Suffix "_ci-upper" und "_ci-lower" erweitert. Entsprechend sind alle Parameter identisch, hinzukommt lediglich `ci`.

- **ci**: Das Konfidenzintervall, das berechnet werden soll. Standard ist das 95%-Intervall.

Rückgabe: Genau wie bei `getVoteshareErrors`, um zwei zusätzliche Spalten je Komponente erweitert.

Beispiel (d'accord zu getVoteshareErrors):

```
getVoteshareErrorCIs(pv, mae)
#Ergebnis (Spalten und Zeilen verkürzt)
#          Date DaysToElection PollyVote PollyVote_ci-upper …
# 1  2017-09-24              0       2.0                6.9 …
# 2  2017-09-23              1       2.1                7.2 …
# 3  2017-09-22              2       2.0                6.9 …
# 4  2017-09-21              3       2.0                6.9 …
# …

getVoteshareErrors(pv, mae, for.ggplot2=T)
#Ergebnis (Zeilen verkürzt)
#          Date DaysToElection  Component      Type Value
# 1  2017-09-24              0  PollyVote     Error   2.0
```

```
# 1  2017-09-24              0  PollyVote  CI upper   6.9
# 1  2017-09-24              0  PollyVote  CI lower  -2.9
# 1  2017-09-24              0   Umfragen     Error   3.1
# 1  2017-09-24              0   Umfragen  CI upper     8
# 1  2017-09-24              0   Umfragen  CI lower  -1.8
# …
```

getSingleVoteshareErrorCIs (data, method, date, ci = .95, component = 'pollyvote', election = null, party = null, region = null)

Direktzugriff für `getVoteshareErrorCIs` an einem dezidierten Tag. Es entfällt entsprechend `limitdays` und `for.ggplot2`, stattdessen kommt der obligatorische Parameter `date` hinzu.

- **date**: Datum (Posixct?), für das der Datensatz aus `getVoteshareErrorCIs` zurückgegeben werden soll.

Rückgabe: Data-Frame mit den Fehlern und Konfidenzintervallen je angeforderter Komponente. Spalten heißen immer Component, Type und Value, die Aufbereitung erfolgt quasi immer `for.ggplot2`.

Beispiel (d'accord zu den Beispielen aus getVoteshareErrorCIs):

```
getSingleVoteshareErrors(pv, mae, as.POSIXct('2017-09-22'))
#Ergebnis (Zeilen verkürzt)
#    Component      Type Value
# 1  PollyVote     Error   2.0
# 2  PollyVote  CI upper   6.9
# 3  PollyVote  CI lower  -2.9
# 4   Umfragen     Error   3.1
# …
```

getVoteshareMeanErrorsTowardElection (data, method, component = 'pollyvote', election = null, party = null, region = null, limitdays = -1, for.ggplot2 = F)

Diese Funktion ist ein etwas stabilerer Wertevergleich. Sie ist in einem ersten Schritt absolut identisch zu `getVoteshareErrors`. In einem zweiten Schritt bestimmt die Funktion aber für jeden einzelnen Tag den Mittelwert aller Fehler einer Komponente bis zu Wahl. Die Werte aus *getVoteshareErrors* werden also sukzessive für den Zeitraum eines jeden Tages bis zum Wahltag gemittelt (z.B. für Tag 10 vor der Wahl werden alle Fehler einer Komponente der Tage 0-10, für Tag 3 alle Fehler der Komponente für die Tage 0-3 gemittelt).

getCoalitions (data, coalitions, threshold = 0, threshold.handle = 'omit', component = 'pollyvote', election = null, region = null, limitdays = -1, for.ggplot2 = F)

Vom Prinzip wie `getVoteshares`. Anstatt Parteien werden aber definierte Koalitionen berechnet. Dafür muss bei einigen Wahlsystemen ein Schwellenwert einbezogen werden, über den Parteien für den Einzug in das Parlament kommen müssen (threshold).

- **data**: Der Datencontainer.
- **coalitions**: Obligatorische Koalitions-Liste, die selbst wiederum aus String-Vektoren besteht, die Parteinamen beinhalten. Es gibt keine Wildcards, hier muss komplett definiert werden, was gewünscht ist.
- **threshold**: Wenn positiv, gibt dieser Parameter den Mindeststimmenanteil an (größer-gleich), den eine Partei erreichen muss, um an einer Koalition teilnehmen zu können.

- **threshold.handle**: Durch einen Threshold können Parteien unberücksichtigt zurückbleiben. So sind möglicherweise einige der angegebenen Koalitionen nicht möglich. Wie mit diesen Koalitionen verfahren wird, bestimmt dieser Parameter. Möglich sind zwei per String definierte Dinge:
    - ***omit*:** Standardwert. In diesem Fall verfällt diese Koalition für das gegebene Datum, stattdessen wird NA eingetragen.
    - ***ignore*:** In diesem Fall wird die Koalition aus den restlichen Parteien (und sei es nur eine) berechnet.
- **component**: Komponente, für die die Stimmenanteile zurückgegeben werden sollen. Die Komponentendefinition weicht hier etwas von den anderen Deklarationen ab, um dem Regelfall näher zu kommen. Standardwert ist die String-Angabe der Wurzel-Prognose, also „pollyvote“, auf übergreifender (d.h. nicht-regionaler) Ebene. Hier kann übergeben werden:
    - ***String***, also ein einzelner Komponentenname (z.B. „Umfrage“); in diesem Fall wird eine Spalte für die angegebene Komponente sowie je eine Spalte je (direkter) Subkomponente zurückgegeben
    - ***Vektor*** mit Komponentennamen; gibt genau für die angegebenen Komponenten (ohne automatische Subkomponentenergänzung) die Werte zurück
    - ***null***, eine Wildcard, bei der schlichtweg alle Komponenten zurückgegeben werden
- **election**: Das Wahljahr, für das berechnet werden soll. Wenn nicht spezifiziert, wird das aktuellste Wahljahr verwendet.
- **party**: Erlaubt die Auswahl bestimmter Parteien. Optional, wenn null, dann alle Parteien. Alternativ ist mehreres möglich:
    - ***String*** für die Angabe einer Partei
    - ***Vektor*** für die Angabe mehrerer Parteien
    - ***null*** für schlichtweg alle Parteien
- **region**: Wenn angegeben (und wenn die Region spezifiziert ist), werden Daten für diese Region eingefügt. Wenn angegeben, aber die Region ist nicht spezifiziert, wird ein Fehler erzeugt. Default ist immer die übergreifende Ebene.
- **limitdays**: Möglichkeit, in Tagen eine Grenze festzulegen, bis zu der die Daten zurückgeliefert werden sollen. Unlimitiert, wenn negativ/null (Default). So schränkt beispielsweise die Angabe von `limitdays=100` die Rückgabe auf 100 Tage vor der Wahl ein, bei einem Wahltermin am 24.09.2017 werden also nur Einträge nach dem 16.07.2017 zurückgegeben.
- **for.ggplot2**: Bool’sche Angabe, die über das Rückgabeformat entscheidet. Bei `FALSE` (Default) wird ein Data-Frame zurückgegeben, bei dem jede Zeile ein Datum widerspiegelt und der direkt als CSV exportiert werden kann. Bei `TRUE` wird indes ein Data-Frame zurückgegeben, bei dem Zeilen für Datenpunkte in der Visualisierung stehen (siehe Beispiel unten).

Rückgabe: D’accord zu `getVoteshares`. Der einzige Unterschied sind die Spalten der Parteien, die jetzt zu Spalten der Koalitionen werden. Koalitionsnamen werden mit Bindestrich verknüpft.

Beispiel (wie bei getVoteshares):

```
getVoteshares(pv, coalitions=list(c('CDU', 'SPD'), c('SPD', 'Linke',
'Grüne')), threshold=5, component='Experten')
#Ergebnis (Spalten und Zeilen verkürzt)
#          Date DaysToElection Experten/CDU-SPD Experten/SPD-Linke-Grüne …
# 1  2017-09-24              0               75                         42 …
# 1  2017-09-23              1               76                         42 …
```

```
# 1  2017-09-22              2               75                          41 …
# …
```

getSingleCoalitions

Kombination aus `getCoalitions` und `getSingleVoteshares`.

getCoalitionErrors

Kombination aus `getCoalitions` und `getVoteshareErrors`.

getSingleCoalitionErrors

Kombination aus `getCoalitions` und `getSingleVoteshareErrors`.

getCoalitionErrorCIs

Kombination aus `getCoalitions` und `getVoteshareErrorCIs`.

getSingleCoalitionErrorCIs

Kombination aus `getCoalitions` und `getSingleVoteshareErrorCIs`.

getCoalitionMeanErrorsTowardElection

Kombination aus `getCoalitions` und `getVoteshareMeanErrorsTowardElection`.

Sonstiges

Import/Export des Datencontainers

Ein Datencontainer, wenn er einmal formatiert ist, wäre ein super zu teilendes Konstrukt. Könnten wir alle unsere Wahlprognosen je Wahl quasi als Datencontainer mit dem Verweis auf dieses R-Paket anbieten, wäre das der Oberhammer! Dafür müsste der Datencontainer aber einerseits ein sinnvolles und von R lesbares Format aufweisen und andererseits müsste man dieses Format auch irgendwie sinnvoll speichern/exportieren sowie importieren können.

Hast du hierfür Ideen?

Variablenschubsen

Die Spezifikation einer Wahl ist so ein recht intensives Variablenspiel. Lässt sich das irgendwie gruppieren, also z.B. so ähnlich wie man das von ggplot2 kennt?

Nach aktueller Konzeption:

```
pv <- setParties(c('CDU/CSU', 'SPD', 'Linke', 'Grüne', 'AfD', 'FDP'),
remaining='sonstige')
pv <- addElection('2017', as.POSIXct('2017-09-22'), data=pv)
pv <- addComponent(pv, 'Umfragen')
pv <- addComponent(pv, 'Forsa', 'Umfragen')
pv <- addComponent(pv, 'Emnid', 'Umfragen')
```

Anstelle der vielen Zuweisungen von pv wäre es schön (und handlich und lesbar), stattdessen einzelne Befehle aneinander ketten zu können. Das würde allerdings bedeuten, das die Funktionen

auch auf den pv-Parameter ggf. verzichten können müssten bzw. ihn als Ergebnis der Verkettung annehmen können müssten. Ist also z.B. sowas möglich (man beachte, neben dem Chaining, das Weglassen des pv-Parameters):

```
pv <- setParties(c('CDU/CSU', 'SPD', 'Linke', 'Grüne', 'AfD', 'FDP'),
remaining='sonstige')
  + addElection('2017', as.POSIXct('2017-09-22'))
  + addComponent('Umfragen')
  + addComponent('Forsa', 'Umfragen')
  + addComponent('Emnid', 'Umfragen')
```

Oder alternativ:

```
pv <- setParties(c('CDU/CSU', 'SPD', 'Linke', 'Grüne', 'AfD', 'FDP'),
remaining='sonstige')
  %>% addElection('2017', as.POSIXct('2017-09-22'))
  %>% addComponent('Umfragen')
  %>% addComponent('Forsa', 'Umfragen')
  %>% addComponent('Emnid', 'Umfragen')
```

Was hältst du davon und hast du hierfür Ideen/Gedanken zur Umsetzung?