

Programmieren in Anwendungen

Annette Bieniusa

Technische Universität Kaiserslautern

bieniusa@cs.uni-kl.de

12.06.2014

Überblick

Fallstudie 2: UFO-Sichtungen

Programmieren in R (Teil 2)

Datenvisualisierung mit ggplot2

Fallstudie 2: UFO-Sichtungen (Fortsetzung)

Fallstudie 2: UFO-Sichtungen

Aus: Drew Conway, John Myles White: Machine Learning for Hackers - Case Studies and Algorithms to Get You Started.
O'Reilly Media, Februar 2012.

Daten zu UFO-Sichtungen

- ▶ Rund 60.000 Datensätze gesammelt durch das National UFO Reporting Center in den letzten Jahrhunderten
- ▶ Daten zum Download:
https://github.com/johnmyleswhite/ML_for_Hackers/tree/master/01-Introduction
- ▶ Aufbau des Datensatzes

DateOccured	Zeitpunkt der Beobachtung
DateReported	Zeitpunkt der Meldung
Location	Ort
ShortDescription	Kurzbeschreibung
Duration	Dauer
LongDescription	Ausführliche Beschreibung

Import des Datensatzes

- ▶ Einträge sind durch Tabulator voneinander getrennt (.tsv - Datei)
- ▶ Datentypen (Zahlen, Strings, Faktoren, etc.) werden heuristisch ermittelt
- ▶ Hier ist die Umwandlung von Strings in Faktoren unerwünscht
- ▶ Leerer String soll durch NA ersetzt werden

```
ufo <- read.delim(file="ufo_awesome.tsv", sep="\t",  
  na.strings="", header=FALSE,  
  stringsAsFactors=FALSE)
```

Struktur von Datensätzen

- ▶ Mit der Funktion `str()` erhält man einen Überblick zur Struktur eines Datensatzes.

```
> str(ufo)
'data.frame': 61870 obs. of 6 variables:
 $ V1: chr  "19951009" "19951010" "19950101" "19950510" ...
 $ V2: chr  "19951009" "19951011" "19950103" "19950510" ...
 $ V3: chr  " Iowa City, IA" " Milwaukee, WI" " Shelton, WA" ...
 $ V4: chr  NA NA NA NA ...
 $ V5: chr  NA "2 min." NA "2 min." ...
 $ V6: chr  "Man repts. witnessing flash ..."
```

Schritt 1: Benennung der Spalten

- ▶ Benannte Spalten vereinfachen den Umgang mit den Daten, da weniger Verwechslungen und Vertippen möglich

```
names(ufo) <- c("DateOccurred", "DateReported",  
"Location", "ShortDescr", "Duration", "LongDescr")
```

Schritt 2: Datumskonvertierung

- ▶ Datumsformat scheint "YYYYMMDD" (year-month-day)

```
> ufo$DateOccurred <- as.Date(ufo$DateOccurred,  
                             format="%Y%m%d")
```

```
Fehler in strptime(x, format, tz = "GMT") :
```

```
  Eingabe-Zeichenkette ist zu lang
```

- ▶ Offensichtlich haben einige Eingaben das falsche Format!
- ▶ Ausweg: Ausfiltern der fehlerhaften Datensätze oder Ausbessern von Hand mit `fix()` (häufig nicht möglich)

```
good.rows <- ifelse(nchar(ufo$DateOccurred) == 8 &  
                  nchar(ufo$DateReported) == 8,  
                  TRUE,FALSE)
```

```
ufo_clean <- ufo[good.row,]
```

- ▶ Dann Datumseinträge `DateOccurred` und `DateReported` konvertieren mit `as.Date()`, wie oben

Filtern von Datensätzen

- ▶ Einfache Filterkriterien
 - == gleich
 - != ungleich
 - >, >= größer (gleich)
 - <, <= kleiner (gleich)
- ▶ Bei kategoriellen Daten durch %in%
haar %in% c("blond", "braun")
- ▶ Kombination von Filterkriterien
 - & Und
 - | Oder
 - ! Negation

Schritt 3: Bearbeitung der Ortsinformation

- ▶ Ortsformat ist scheinbar "Stadt, Staat"

```
get.location <- function(l) {  
  split.location <- tryCatch(strsplit(l, ",")[[1]],  
                             error = function(e) return(c(NA, NA)))  
  clean.location <- gsub("^ ", "", split.location)  
  if (length(clean.location) > 2) {  
    return(c(NA, NA))  
  } else {  
    return(clean.location)  
  }  
}
```

- ▶ `strsplit(x, ",")`
trennt die Zeichenkette bei Komma, das folgende `[[1]]`
unquoted den Ergebnisvektor
- ▶ `gsub("^ ", "", x)`
entfernt alle Leerzeichen am Anfang der Zeichenkette x

Schritt 3: Bearbeitung der Ortsinformation

- ▶ Erstellen einer Matrix mit der Stadt-Staat-Information durch Anwenden von `get.location` mittels `lapply()` und Zusammenfassen der resultierenden Liste durch `do.call()` und `rbind()`
- ▶ Hinzufügen dieser Matrix zum Datensatz

```
city.state <- lapply(ufo$Location, get.location)
location.matrix <- do.call(rbind, city.state)
ufo <- transform(ufo,
                 USCity = location.matrix[, 1],
                 USState = location.matrix[, 2],
                 stringsAsFactors = FALSE)
```

Schritt 4: Filtern nach US-Staaten

- ▶ `state.abb` ist ein Vektor mit Abkürzungen der US-Staatennamen
- ▶ `match()` liefert den Index in `state.abb`, der die passende Abkürzung enthält
- ▶ Falls keine passt, wird `NA` gesetzt
- ▶ Schließlich wird nach den Einträgen mit korrekten US-Staaten gefiltert

```
ufo$USState <- state.abb[match(ufo$USState, state.abb)]  
ufo.us <- subset(ufo, !is.na(USState))
```

Ergebnis: Bearbeiteter Datensatz

```
> head(ufo.us)
  DateOccurred DateReported      Location ShortDescription Duration
1  1995-10-09  1995-10-09  Iowa City, IA          <NA>      <NA>
2  1995-10-10  1995-10-11  Milwaukee, WI          <NA>      2 min.
3  1995-01-01  1995-01-03    Shelton, WA          <NA>      <NA>
4  1995-05-10  1995-05-10    Columbia, MO          <NA>      2 min.
5  1995-06-11  1995-06-14    Seattle, WA          <NA>      <NA>
6  1995-10-25  1995-10-24 Brunswick County, ND  <NA>      30
min.
```

```
                                LongDescription
1 Man repts. witnessing "flash, followed by a classic UFO, ...
2 Man on Hwy 43 SW of Milwaukee sees large, bright blue light ...
3 Telephoned Report:CA woman visiting daughter witness discs and ...
4 Man repts. son's bizarre sighting of small humanoid creature ...
5 Anonymous caller repts. sighting 4 ufo's in NNE sky, 45 deg. ...
6 Sheriff's office calls to rept. that deputy, 20 mi. SSE of ...
```

```
      USCity USState
1      Iowa City    IA
2      Milwaukee    WI
3      Shelton     WA
4      Columbia    MO
5      Seattle     WA
6 Brunswick County ND
```

Fortsetzung folgt...

Programmieren in R (Teil 2)

Eigenheiten von R

- ▶ Variablen müssen nicht deklariert werden, sondern werden durch die erste Zuweisung verwendbar
- ▶ Indizierung startet mit 1, nicht mit 0
- ▶ Zuweisung zu einem nicht existenten Index eines Vektors erweitert den Vektor (ähnlich bei Matrix, Listen, ...)

```
> z <- 1:5  
> z[7] <- 9  
> z  
[1] 1 2 3 4 5 NA 9
```

- ▶ Kommentare beginnen mit #

Typische Fehlerquellen

- ▶ Verwechslung von Groß/Kleinschreibung: `length()` und `Length()` sind verschiedene Funktionen!
- ▶ Verwenden von Funktionen ohne das notwendige Package geladen zu haben
- ▶ Vergessen von Anführungszeichen beim Laden, Installieren etc. von Packages
- ▶ Jeder Funktionsaufruf braucht Klammern, auch ohne Parameter!

Factors

- ▶ Datenwerte entstammen oft Kategorien, diese können eine Ordnung implizieren
- ▶ Beispiel: Geschlecht (m/w/sonstiges) [ungeordnet], Schulnoten (sehr gut, gut, befriedigend, ausreichend, mangelhaft, ungenügend) [geordnet]
- ▶ In R werden diese Datenwerte `factors` genannt
- ▶ Eine Ordnung kann mit `ordered()` spezifiziert werden, standardmässig wird die alphabetische Ordnung der Kategorien genommen
- ▶ `str()` gibt einen Überblick über die Struktur
- ▶ Intern werden die Factor-Datenwerte als Integer gespeichert, die den einzelnen Werten zugeordnet sind
→ wesentlich effizienter als Strings

Factors mit Ordnung

```
> noten <- c("sehr gut", "mangelhaft", "gut", "gut")
> noten <- factor(noten, order = TRUE)
> noten
[1] sehr gut    mangelhaft gut
Levels: gut < mangelhaft < sehr gut
> noten <- factor(noten, order = TRUE,
                  levels=c("sehr gut", "gut", "mangelhaft"))
> noten
[1] sehr gut    mangelhaft gut
Levels: sehr gut < gut < mangelhaft
> str(noten)
Ord.factor w/ 3 levels "sehr gut"<"gut"<...: 1 3 2
```

Umkodieren von Variablen

- ▶ Beispiel: Gegeben ein Datensatz mit den Teilnehmern einer Veranstaltung
- ▶ Spalte `teilnehmer$alter` bezeichnet das Alter
- ▶ Gewünschte neue Variable im Datensatz:
Welcher der Teilnehmer ist minderjährig?

```
status <- (teilnehmer$alter < 18) * 1  
          + (teilnehmer$alter >= 18) * 2
```

- ▶ Umwandlung von numerischem Wert in Faktor

```
status_factor <- factor(status,  
  labels = c("minderjaehrig", "volljaehrig"))
```
- ▶ Häufigkeit der einzelnen Faktoren/Kategorien

```
table(status_factor)
```

Datenvisualisierung mit ggplot2

Was ist ggplot2?

- ▶ ggplot2 ist ein R-Package für Datenvisualisierung
- ▶ Unkomplizierte, intuitive Erstellung von modernen Grafiken
- ▶ ggplot2 basiert auf der "grammar of graphics"
- ▶ Modifikation der Plotkomponenten auf hoher Abstraktionsebene durch Layern (Schichten)
- ▶ Installation: `install.packages('ggplot2')`
- ▶ Einbindung in den Workspace: `library(ggplot2)`

Anwendungsbeispiel: Datensatz diamonds

- ▶ Standardmäßig im ggplot2-Package enthalten
- ▶ Beinhaltet Eigenschaften zu fast 54.000 Diamanten

```
> head(diamonds)
  carat      cut color clarity depth table price ...
1  0.23    Ideal   E     SI2   61.5    55   326 ...
2  0.21  Premium   E     SI1   59.8    61   326 ...
3  0.23     Good   E     VS1   56.9    65   327 ...
4  0.29  Premium   I     VS2   62.4    58   334 ...
5  0.31     Good   J     SI2   63.3    58   335 ...
6  0.24 Very Good   J     VVS2   62.8    57   336 ...
```

Eigenschaften:

- ▶ carat: Gewicht des Diamanten
- ▶ cut: Qualität des Schliffs
- ▶ color: Farbe (von J bis D) etc. (siehe `help(diamonds)`)

Struktur von diamonds

```
> str(diamonds)
'data.frame': 53940 obs. of 10 variables:
 $ carat   : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22...
 $ cut     : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 ...
 $ color   : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 ...
 $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5...
 $ depth   : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1...
 $ table   : num  55 61 65 58 58 57 57 55 61 61 ...
 $ price   : int  326 326 327 334 335 336 336 337 337 338 ...
 $ x       : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 ...
 $ y       : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78...
 $ z       : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49...
```

Datenvisualisierung mit ggplot2

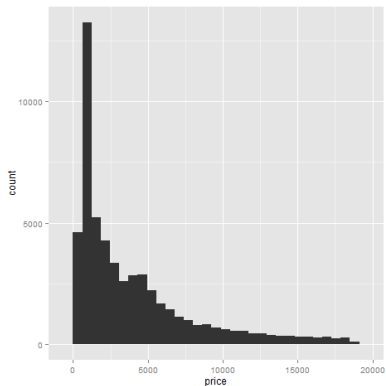
qplot - Quick plot

```
qplot(data, x, y = NULL, ..., facets = NULL,  
      margins = FALSE, geom = "auto", stat = list(NULL),  
      position = list(NULL), xlim = c(NA, NA),  
      ylim = c(NA, NA), log = "", main = NULL,  
      xlab = deparse(substitute(x)),  
      ylab = deparse(substitute(y)), asp = NA)
```

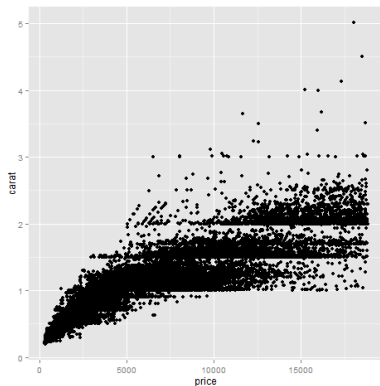
Wichtigste Parameter in rot!

- ▶ data: Datensatz mit x,y,... Spalten
- ▶ geom: siehe nächste Seite
- ▶ xlim, ylim: Achsenabschnitte
- ▶ xlab, ylab: Achsenbeschriftung
- ▶ main: Überschrift

Einfache Geometrien



(a) `qplot(data=diamonds,price)`



(b) `qplot(data=diamonds,price,carat)`

Standardwerte für Geometrien

Eindimensionale Geometrien (falls nur x spezifiziert ist)

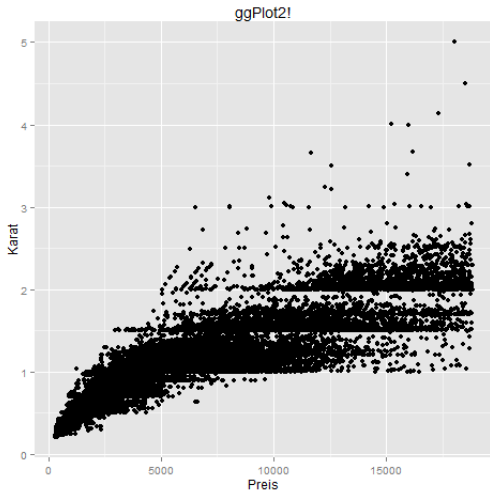
- ▶ geom = "histogram" Histogramm (Standard), binwidth = ... gibt die Größe des Bins an
- ▶ geom = "freqpoly" und geom="density" Frequenz- und Dichteplot

Zweidimensionale Geometrien (falls x und y spezifiziert sind)

- ▶ geom = "point" Scatterplot (Standard)
- ▶ geom = "smooth" Glättung
- ▶ geom = "boxplot" Box-and-whisker Plot, liefert Informationen zur Verteilung der Daten
- ▶ geom="path" und geom="line" Verbindungslinien zwischen den Datenpunkten, oft genutzt für zeitliche Entwicklungen

Plot mit Beschriftung

```
ggplot(data=diamonds, price, carat, main="ggPlot2!",  
       xlab="Preis", ylab="Karat")
```



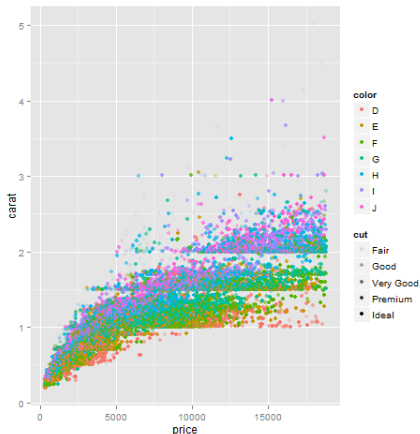
Weitere Aspekte: Farbe

```
qplot(data=diamonds, price, carat, color=color)
```



Weitere Aspekte: Alpha-Wert

```
qplot(data=diamonds, price, carat, color=color,  
      alpha=cut)
```



Weitere Information: <http://ggplot2.org/book/qplot.pdf>

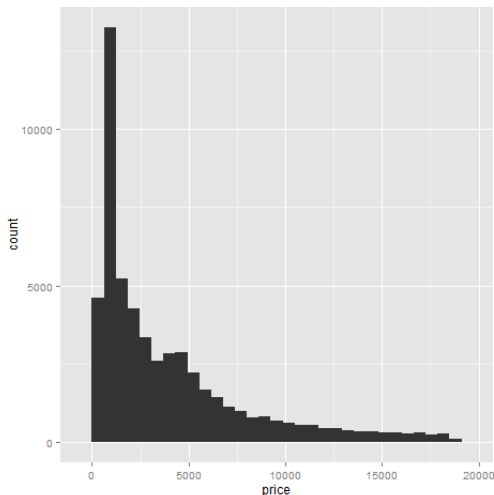
Datenvisualisierung mit ggplot2

ggplot - Grafik für Fortgeschrittene

- ▶ Grundanweisung: `ggplot(data, aes)`
- ▶ Die Ästhetik definiert, wie Datenvariablen auf visuelle Eigenschaften der benutzten Geometrie übertragen werden: `aes(x, y, color, alpha, size,...)`
- ▶ Eine Ästhetik ist für alle folgenden Layer gültig
- ▶ Inkrementelle Ploterstellung durch Addition von Anweisungen mithilfe von `+`

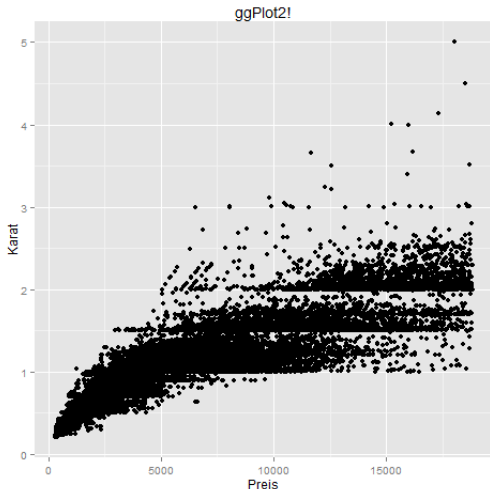
Histogramm

```
myPlot <- ggplot(data=diamonds, aes(x=price))  
myPlot <- myPlot + geom_bar()
```



Scatterplot

```
myPlot <- ggplot(data=diamonds, aes(x=price, y=carat))  
myPlot + geom_point() + ggtitle("ggPlot2")  
+ xlab("Preis") + ylab("Karat")
```



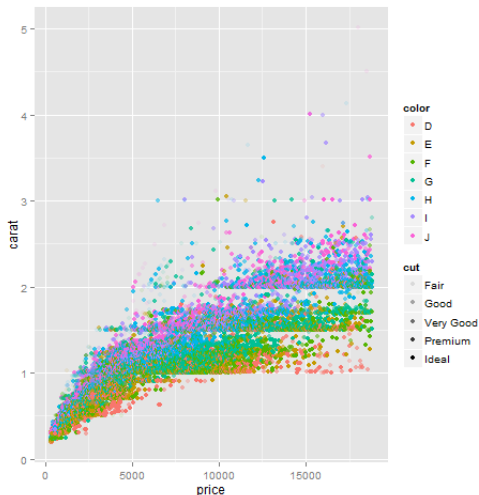
Scatterplot mit Farbe

```
myPlot + geom_point(aes(color=color))
```



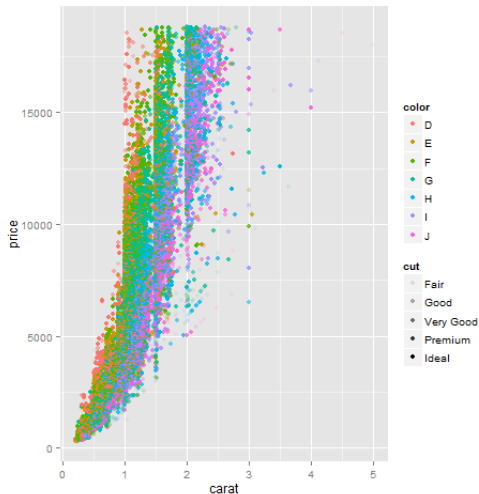
Scatterplot mit Farbe und Alphawert

```
myPlot + geom_point(aes(color=color, alpha=cut))
```



Wechsel der Koordinaten

```
myPlot + geom_point(aes(color=color, alpha=cut))  
+ coord_flip()
```



Einschub - Facetten

Facetten dienen der Aufsplitterung von Diagrammen in die einzelnen Kategorien eines Parameters:

+ **facet_grid(horizontal ~ vertical)**

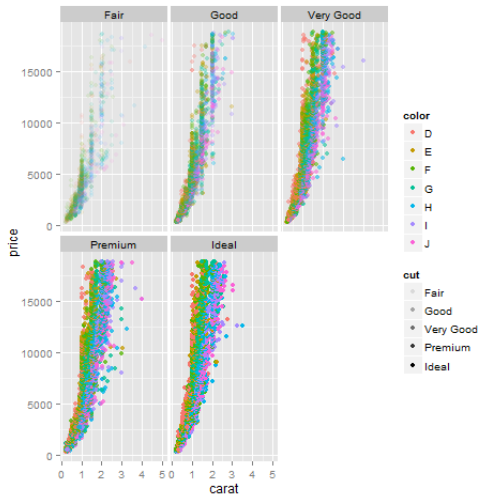
- ▶ Gruppiere in der Horizontalen nach den Werten von cut
`facet_grid(cut ~ .)`
- ▶ Gruppiere in der Vertikalen nach den Werten von cut
`facet_grid(. ~ cut)`

+ **facet_wrap(~ vertical)**

- ▶ Automatische Platzierung der Facetten neben- und untereinander
- ▶ Parameter `ncol=x` begrenzt das Wrapping horizontal

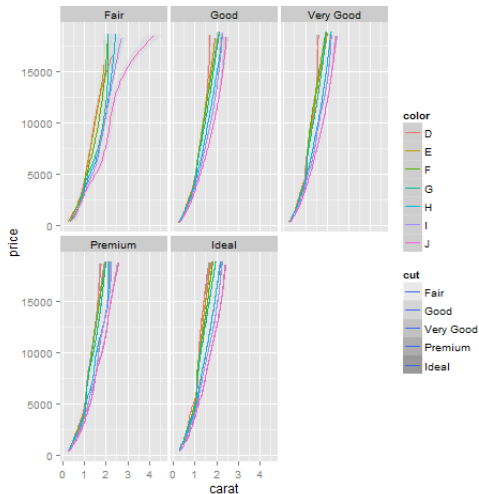
Aufteilung mit Facette cut

```
myPlot + geom_point(aes(color=color, alpha=cut))  
+ coord_flip() + facet_wrap(~ cut)
```



Glättung zur Visualisierung von Trends

```
myPlot + geom_smooth(aes(color=color, alpha=cut))  
+ coord_flip() + facet_wrap(~ cut)
```



Säulendiagramm

```
myPlot2 <- ggplot(data=diamonds, aes(x=clarity))
```

```
+ geom_bar(
```

```
  data=w, #Definition eigener Data frame
```

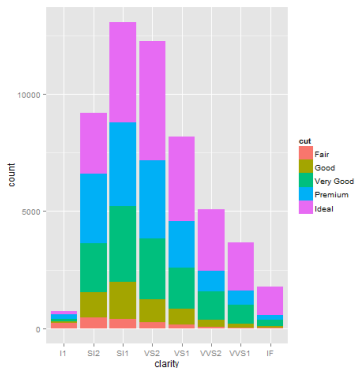
```
  width=x, #Einstellung der Breite
```

```
  aes(fill=y), #Füllfarbe der Balken abhängig von y
```

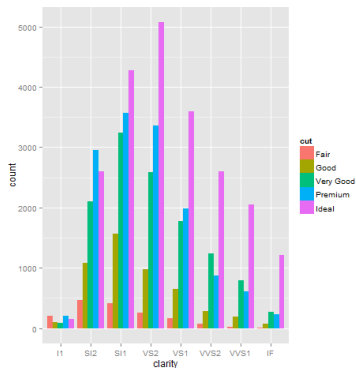
```
  position="z" #Anordnung der gefärbten Balken
```

```
)
```

Beispiel: Säulendiagramm



(c) `aes(fill=cut)`, `position="stack"`

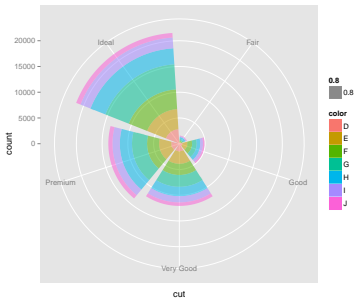


(d) `aes(fill=cut)`, `position="dodge"`

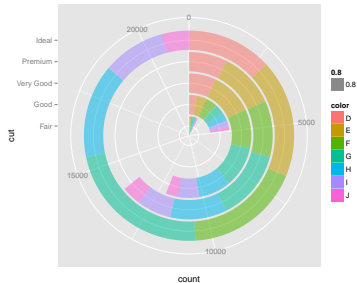
Kreisdiagramme als spezielle Säulendiagramme

+ `geom_bar()` + `coord_polar()`

`theta="x"` #welche Achse bildet den Kreisumfang, "x"/"y"
)



(e) `theta="x"`



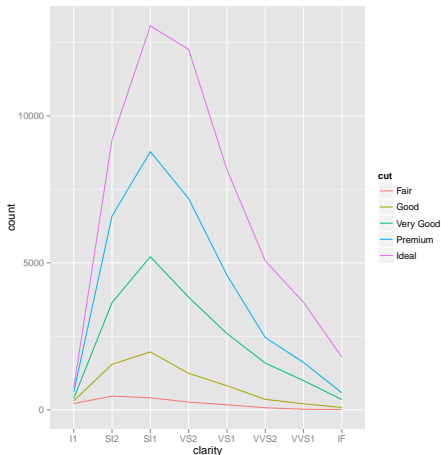
(f) `theta="y"`

Frequency polygon

```
geom_freqpoly(  
  data=w, #Definition eigener Data frame  
  aes(group=x,color=x),  
  #Linien nach Kriterium x, Farben der Kriterien  
  position="z" #Anordnung der Linien  
)
```

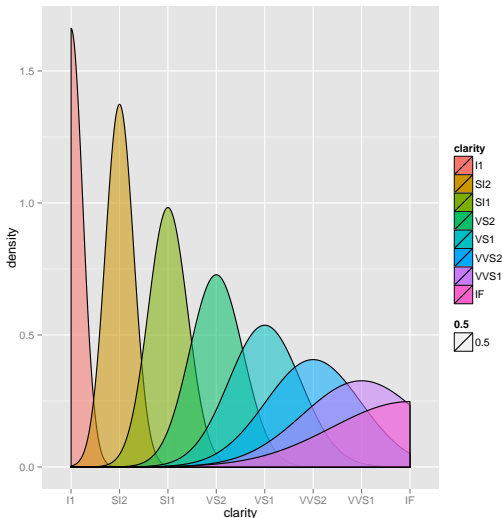
Beispiel: Frequency polygon

```
myPlot2 + geom_freqpoly(aes(group=cut, color=cut),  
                        position="stack")
```

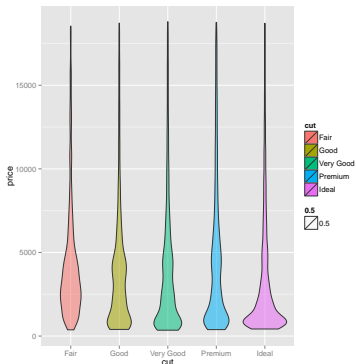


Dichteauswertung "density"

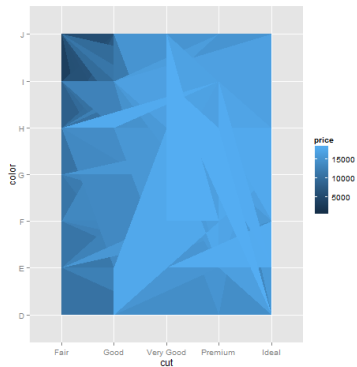
```
myPlot2 + geom_density(aes(fill=clarity, alpha=0.5))
```



...und viele mehr



(g) `geom_violin()`



(h) `geom_polygon()`

Weitere Ideen und Möglichkeiten:
<http://docs.ggplot2.org/current/>

Fallstudie 2: UFO-Sichtungen (Fortsetzung)

Visualisierung der Daten

- ▶ Ziel: Zeitlicher Verlauf der Sichtungen pro US-Staat
- ▶ Beschränkung auf die Zeit nach 1990, da davor nur sporadische Sichtungen

```
> summary(ufo.us$DateOccurred)
      Min.      1st Qu.      Median      Mean      ...
"1400-06-30" "1999-09-07" "2004-01-10" "2001-02-13" ...
ufo.us <- subset(ufo.us, DateOccurred >= as.Date("1990-01-01"))
```

Schritt 5: Aggregieren nach Monat und Jahr

- ▶ Hinzufügen eines weiteren Aspekts, nämlich Jahr und Monat der Beobachtung
- ▶ `strftime()` formatiert das Datum nach dem angegebenen Muster
- ▶ `ddply()` nimmt den Datensatz, splittet ihn in die verschiedenen Untergruppen (hier: Staat und Jahr-Monat) und wendet die Funktion `nrow` auf die Untergruppen an
- ▶ Der resultierende Vektor ist geordnet nach Staaten (alphabetisch) und Datum (chronologisch)

```
> ufo.us$YearMonth <- strftime(ufo.us$DateOccurred, format = "%Y-%m")
> sightings.counts <- ddply(ufo.us, .(USState,YearMonth), nrow)
> head(sightings.counts)
```

	USState	YearMonth	V1
1	AK	1936-10	1
2	AK	1943-04	1
3	AK	1949-06	1
4	AK	1954-02	1
5	AK	1955-06	1

Schritt 6: Ergänzung fehlender Einträge

- ▶ Erstellen eines (geordneten) Vektors mit allen Jahr-Monat-Variationen und Konvertierung in passendes Stringformat
- ▶ Kombinieren mit den US-Staaten
- ▶ Zusammenfassen der resultierenden Liste zu einem Vektor

```
date.range <- seq.Date(from = as.Date(min(ufo.us$DateOccurred)),  
                      to = as.Date(max(ufo.us$DateOccurred)),  
                      by = "month")  
date.strings <- strftime(date.range, "%Y-%m")  
states.dates <- lapply(state.abb, function(s) cbind(s, date.strings))  
states.dates <- data.frame(do.call(rbind, states.dates),  
                          stringsAsFactors = FALSE)
```

Schritt 6: Ergänzung fehlender Einträge

- ▶ Zusammenfügen der beiden (geordneten) Datensätze
- ▶ `all=TRUE` bewirkt, dass fehlende Einträge in `sightings.counts` mit `NA` ergänzt werden

```
all.sightings <- merge(states.dates,  
                        sightings.counts,  
                        by.x = c("s", "date.strings"),  
                        by.y = c("USState", "YearMonth"),  
                        all = TRUE)
```

Schritt 7: Konvertieren der Datentypen

- ▶ Benennung der Spalten
- ▶ Ersetzen von NA durch 0
- ▶ Umwandeln der Datum-Strings in Datum-Objekte
- ▶ Umwandeln der Staatenkürzel in Factors

```
names(all.sightings) <- c("State", "YearMonth", "Sightings")
all.sightings$Sightings[is.na(all.sightings$Sightings)] <- 0
all.sightings$YearMonth <- as.Date(rep(date.range, length(state.abb)))
all.sightings$State <- as.factor(all.sightings$State)
```

Schritt 8: Generieren der Graphik

- ▶ `facet_wrap()` liefert Plots für jeden Staat in einem 10x5 Raster
- ▶ `theme_bw()` ändert den Standard-Style zu einem Style mit weißem Hintergrund
- ▶ `scale_color_manual()` setzt dunkelblau als Linienfarbe
- ▶ `scale_x_date()` rastert die x-Achse als Datumformat im Abstand von 5 Jahren

```
state.plot <-  
  ggplot(all.sightings, aes(x=YearMonth,y=Sightings)) +  
  geom_line(aes(color="darkblue")) +  
  facet_wrap(~State, nrow=10, ncol=5) +  
  theme_bw() +  
  scale_color_manual(values=c("darkblue"="darkblue"),guide="none") +  
  scale_x_date(breaks="5 years", labels=date_format('%Y')) +  
  xlab("Years") +  
  ylab("Number of Sightings") +  
  ggtitle("Number of UFO sightings by U.S. State (1990-2010)")
```

UFO-Sichtungen in den einzelnen US-Staaten (1990-2010)

