Exploring data 2

Regression modeling

Regression modeling

Generalized linear models serve as a powerful framework for statistically analyzing your data. They incorporate linear regression models, but also logistic regression, Poisson regression, and other regression models.

World Cup example

In the World Cup data, we may wonder if the number of tackles is associated with the time the player played. Let's start by grabbing just the variables we care about (we'll be using `Position` later, so we'll include that):

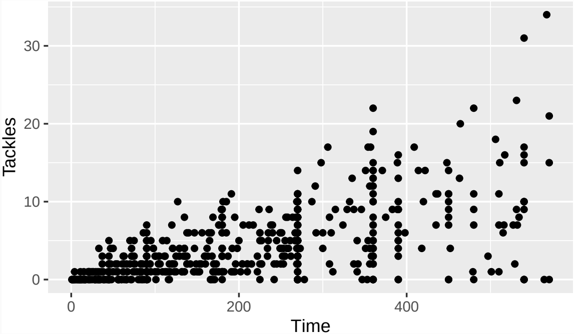
```
library(faraway)
data(worldcup)
worldcup <- worldcup %>%
  select(Time, Tackles, Position)
worldcup %>% slice(1:3)
```

```
##         Time Tackles   Position
## Abdoun    16       0 Midfielder
## Abe      351      14 Midfielder
## Abidal   180       6   Defender
```

World Cup example

We can start by plotting the relationship between the time a player played and the number of tackles they had:

```
library(ggplot2)
ggplot(worldcup, aes(Time, Tackles)) +
  geom_point()
```



World Cup example

There does indeed seem to be an association. Next, we might want to test this using some kind of statistical model or test.

Let's start by fitting a linear regression model, to see if there's evidence that tackles tend to change (increase or decrease) as the player's time played increases.

(In a bit, we'll figure out that a linear model might not be the best way to model this, since the number of tackles is a count, rather than a variable with a normal distribution, but bear with me. . .)

Formula structure

Regression models can be used to estimate how the expected value of a *dependent variable* changes as *independent variables* change.

In R, regression formulas take this structure:

```
## Generic code
[response variable] ~ [indep. var. 1] +  [indep. var. 2] + ...
```

Notice that ~ used to separate the independent and dependent variables and the + used to join independent variables. This format mimics the statistical notation:

$$Y_i \sim X_1 + X_2 + \cdots + \epsilon_i$$

You will use this type of structure in R for a lot of different function calls, including those for linear models (`lm`) and generalized linear models (`glm`).

Linear models

To fit a linear model, you can use the function `lm()`. Use the `data` option to specify the dataframe from which to get the vectors. You can save the model as an object.

```
tackle_model <- lm(Tackles ~ Time, data = worldcup)
```

This call fits the model:

$$Y_i = \beta_0 + \beta_1 X_{1,i} + \epsilon_i$$

where:

- Y_i : Number of tackles for player i (dependent variable)
- $X_{1,i}$: Minutes played by player i (independent variable)

Linear models

A few things to point out:

- By default, an intercept is fit to the model.
- If you specify a dataframe using `data` in the `lm` call, you can write the model formula using just the column names for the independent variable(s) and dependent variable you want, without quotation marks around those names.
- You can save the output of fitting the model to an R object (if you don't, a summary of the fit model will be print out at the console).

Model objects

The output from fitting a model using `lm` is a list object:

```
class(tackle_model)
```

```
## [1] "lm"
```

This list object has a lot of different information from the model, including overall model summaries, estimated coefficients, fitted values, residuals, etc.

```
names(tackle_model)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

Model objects and `broom`

This list object is not in a “tidy” format. However, you can use functions from `broom` to pull “tidy” dataframes from this model object.

For example, you can use the `glance` function to pull out a one-row tidy dataframe with model summaries.

```
library(broom)
glance(tackle_model)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.373         0.372  3.69      353. 4.31e-62     1 -1620. 3246. 3259.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

Model objects and broom

If you want to get the estimated model coefficients (and some related summaries) instead, you can use the tidy function to do that:

```
tidy(tackle_model)
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)   0.110    0.265       0.415 6.78e- 1
## 2 Time          0.0195   0.00104    18.8   4.31e-62
```

This output includes, for each model term, the **estimated coefficient** (estimate), its **standard error** (std.error), the **test statistic** (for lm output, the statistic for a test with the null hypothesis that the model coefficient is zero), and the associated **p-value** for that test (p.value).

Model objects and broom

Some of the model output have a value for each original observation (e.g., fitted values, residuals). You can use the augment function to add those elements to the original data used to fit the model:

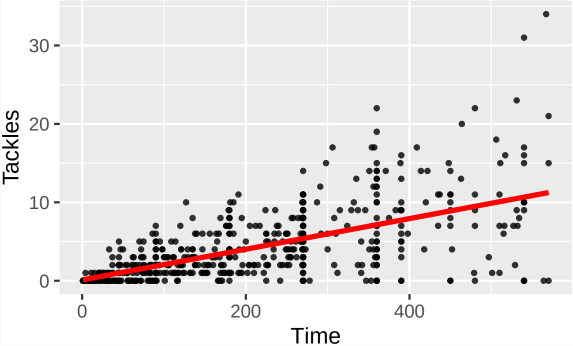
```
augment(tackle_model) %>%
  slice(1:2)
```

```
## # A tibble: 2 x 9
##   .rownames Tackles  Time .fitted .resid .std.resid    .hat .sigma   .cooksd
##   <chr>       <int> <int>   <dbl>  <dbl>      <dbl>   <dbl>  <dbl>     <dbl>
## 1 Abdoun          0    16   0.423 -0.423     -0.115 0.00464   3.69 0.0000307
## 2 Abe            14   351   6.97   7.03       1.91  0.00329   3.68 0.00601
```

Model objects and `broom`

One important use of this `augment` output is to create a plot with both the original data and a line showing the fit model (via the predictions):

```
augment(tackle_model) %>%
  ggplot(aes(x = Time, y = Tackles)) +
  geom_point(size = 0.8, alpha = 0.8) +
  geom_line(aes(y = .fitted), color = "red", size = 1.2)
```



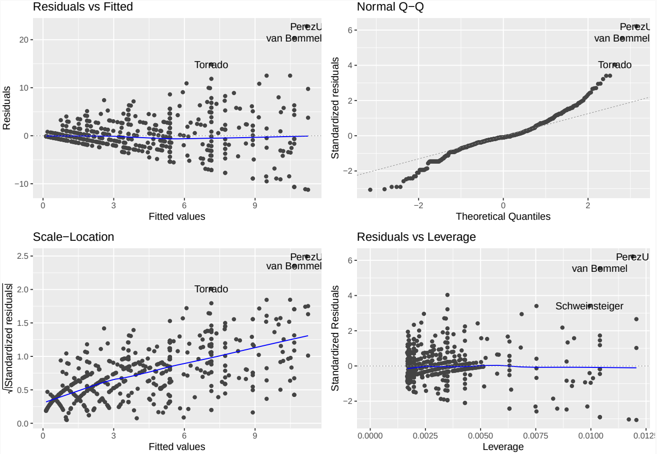
Model objects and `autoplot`

There is a function called `autoplot` in the `ggplot2` package that will check the class of an object and then create a certain default plot for that class. Although the generic `autoplot` function is in the `ggplot2` package, for `lm` and `glm` objects, you must have the `ggfortify` package installed and loaded to be able to access the methods of `autoplot` specifically for these object types.

If you have the package that includes an `autoplot` method for a specific object type, you can just run `autoplot` on the objects name and get a plot that is considered a useful default for that object type. For `lm` objects, `autoplot` gives small graphics with model diagnostic plots.

Model objects and `autoplot`

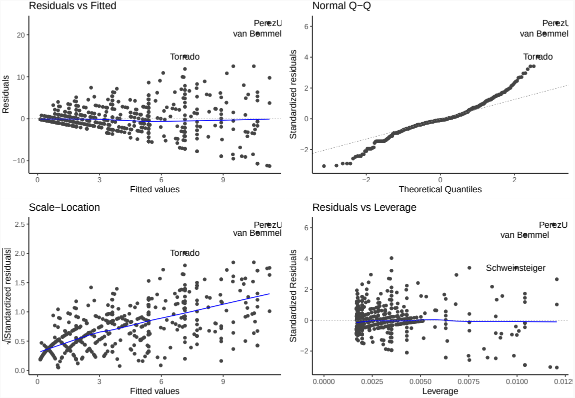
```r
library(ggfortify)
autoplot(tackle_model)
```



Model objects and `autoplot`

The output from `autoplot` is a `ggplot` object, so you can add elements to it as you would with other `ggplot` objects:

```r
autoplot(tackle_model) +
  theme_classic()
```

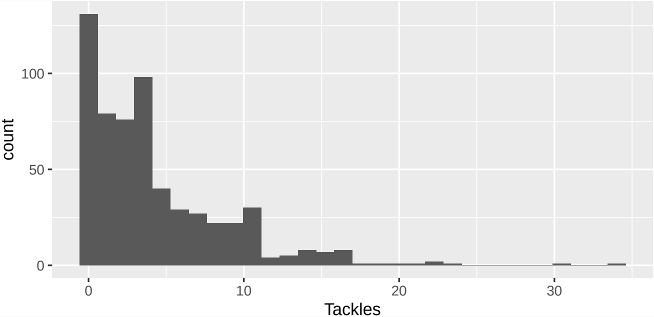


Regression models

In this case, these diagnostics clearly show that there are some problems with using a linear regression model to fit this data.

Many of these issues arise because the outcome (dependent) variable doesn't follow a normal distribution.

```
ggplot(worldcup, aes(x = Tackles)) +
  geom_histogram()
```



Regression models

A better model, therefore, might be one where we assume that `Tackles` follows a Poisson distribution, rather than a normal distribution. (For variables that represent counts, this will often be the case.)

In the a little bit, we'll look at **generalized linear models**, which let us extend the idea of a linear model to situations where the dependent variable follows a distribution other than the normal distribution.

Fitting a model with a factor

You can also use binary variables or factors (i.e., categorical variables) as independent variables in regression models:

```
tackles_model_2 <- lm(Tackles ~ Position, data = worldcup)
```

This call fits the model:

$$Y_i = \beta_0 + \beta_1 X_{1,i} + \epsilon_i$$

where $X_{1,i}$: Position of player i

Fitting a model with a factor

If there are more than one levels to the factor, then the model will fit a separate value for each level of the factor above the first level (which will serve as a baseline):

```r
levels(worldcup$Position)
```

```
## [1] "Defender"   "Forward"    "Goalkeeper" "Midfielder"
```

```r
tidy(tackles_model_2)
```

```
## # A tibble: 4 x 5
##   term               estimate std.error statistic  p.value
##   <chr>                 <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)           5.46      0.315    17.3   1.14e-54
## 2 PositionForward      -3.44      0.480    -7.17  2.20e-12
## 3 PositionGoalkeeper   -5.43      0.787    -6.91  1.27e-11
## 4 PositionMidfielder   -0.300     0.426    -0.705 4.81e- 1
```

Fitting a model with a factor

The intercept is the expected (average) value of the outcome (`Tackles`) for the first level of the factor. Each other estimate gives the expected difference between the value of the outcome for this first level of `Position` and one of the other levels of the factor.

```r
levels(worldcup$Position)
```

```
## [1] "Defender"   "Forward"    "Goalkeeper" "Midfielder"
```

```r
tidy(tackles_model_2)
```

```
## # A tibble: 4 x 5
##   term               estimate std.error statistic  p.value
##   <chr>                 <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)           5.46      0.315    17.3   1.14e-54
## 2 PositionForward      -3.44      0.480    -7.17  2.20e-12
## 3 PositionGoalkeeper   -5.43      0.787    -6.91  1.27e-11
## 4 PositionMidfielder   -0.300     0.426    -0.705 4.81e- 1
```