Task 1: LinReg Oktoberfest

Tolga Tabanli

2025-09-21

Task 1: Regression - Oktoberfest

The managers of Oktoberfest are curious: Do people drink more beer when they eat more chicken – or is it the other way around? Since every liter of beer and every chicken sold is carefully recorded, they have an idea: Perhaps beer consumption can be estimated based on chicken sales, without having to count every single liter. Or maybe even based on other parameters like visitors and the prices.

Your task is to use regression to find out if there is a correlation between the beer consumption and other Oktoberfest data, and how accurate beer consumption can be predicted from them. This will give the managers a feel for how eating, drinking, prices and visits are related. So you can play a bit of Oktoberfest detective!

We will use the packages from the `tidyverse`:

```
library(tidyverse)
```

I've split the data in advance using the following code to create a random training and test data.

```
library(rsample)

set.seed(100)
splitter <- initial_split(oktoberfest, prop = 0.8)
test_oktoberfest <- training(splitter)
train_oktoberfest <- testing(splitter)
write_csv(test_oktoberfest, "oktoberfest_test.csv", col_names = TRUE)
write_csv(train_oktoberfest, "oktoberfest_train.csv", col_names = TRUE)
```

Tip: Quick guide

Here is a short list of steps to consider, not all might apply:

1. Data exploration
2. Missingness
3. Imputation
4. Feature filtering
5. Feature engineering (transformations / standardization)
6. Model fitting - cross-validation
7. Check errors (residual distribution)

Data exploration

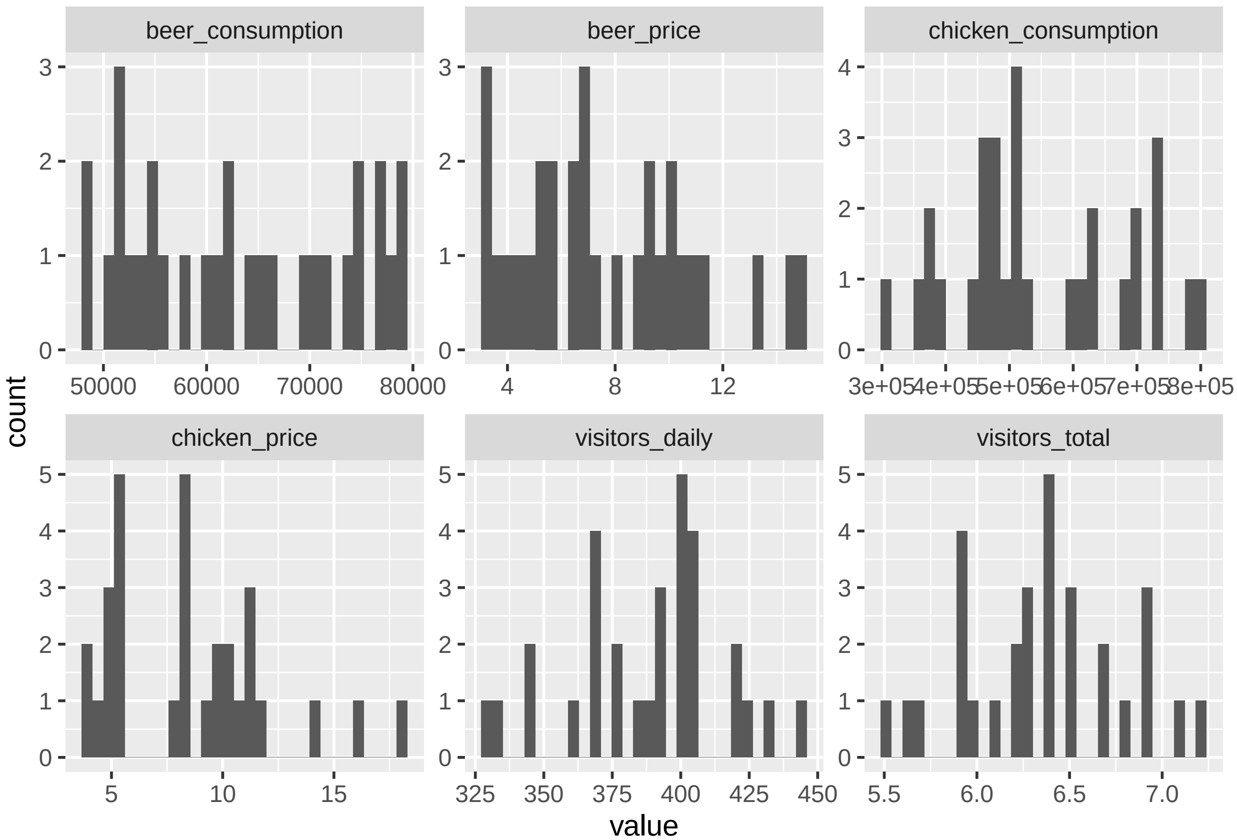
Some notes on exploration tools (Tierney, 2024)[1]: summary(), str(), skimr, dplyr::glimpse(), visdat… We can also employ histograms for all variables.

```
oktoberfest_train <- read_csv("data/oktoberfest_train.csv")
oktoberfest_train
```

```
## # A tibble: 30 x 8
##     year duration visitors_total visitors_daily beer_price beer_consumption
##    <dbl>    <dbl>          <dbl>          <dbl>      <dbl>            <dbl>
##  1  1985       16            7.1            444       3.2             54541
##  2  1986       16            6.7            419       3.3             53807
##  3  1987       16            6.5            406       3.37            51842
##  4  1989       16            6.2            388       3.6             51241
##  5  1991       16            6.4            400       4.21            54686
##  6  1992       16            5.9            369       4.42            48888
##  7  1993       16            6.5            406       4.71            51933
##  8  1995       16            6.7            419       5.15            50162
##  9  1996       16            6.9            431       5.24            52622
## 10  1997       16            6.4            400       5.45            55891
## # i 20 more rows
## # i 2 more variables: chicken_price <dbl>, chicken_consumption <dbl>
```

```
oktoberfest_train %>%
  pivot_longer(!matches(c("year", "duration")), names_to = "variable", values_to = "value") %>%
  ggplot(aes(x = value)) +
  geom_histogram(bins = 30) +
  facet_wrap(~ variable, scales = "free")
```

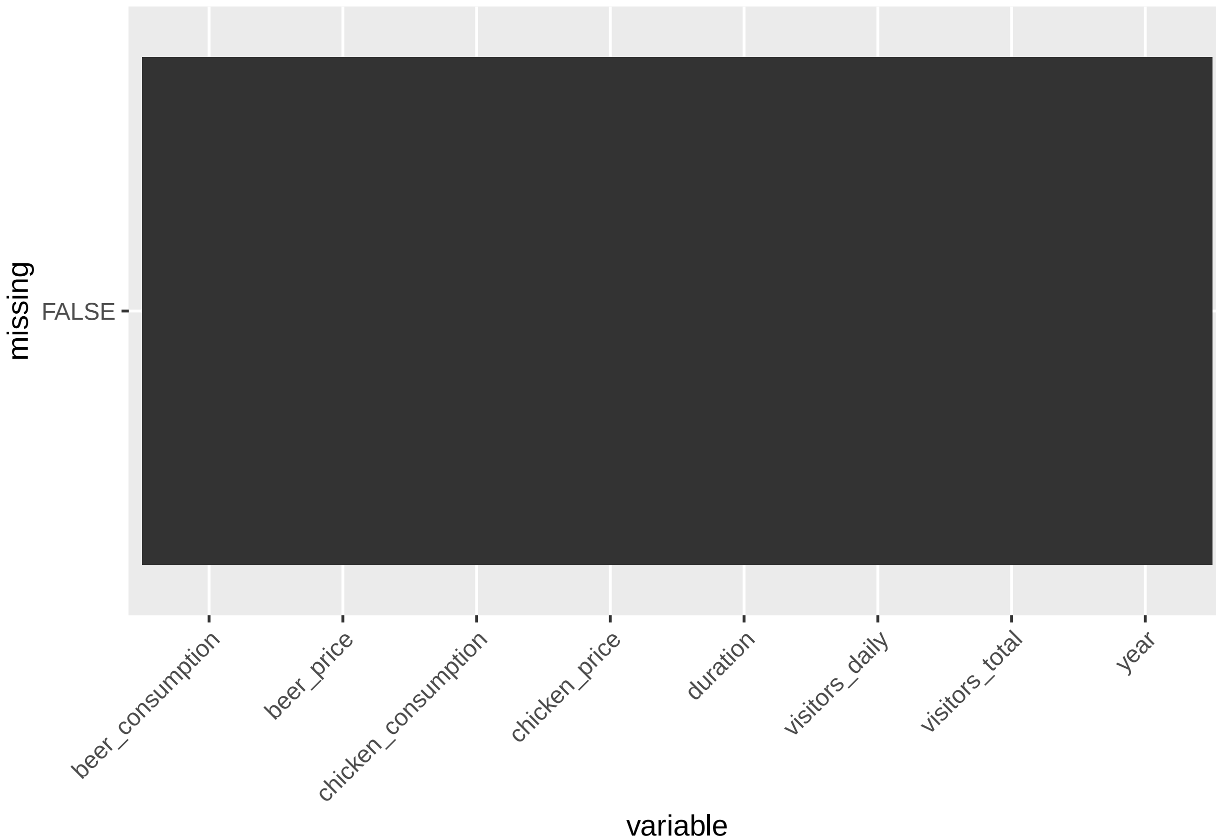
[1] https://cran.r-project.org/web/packages/naniar/vignettes/getting-started-w-naniar.html



For simplicity of the interpretation of the variables, we're not going to undertake any transformations for now. But for future, log-transformation for right skewed data and Box-Cox transformation for more general applications could be considered.

Missingness

```
oktoberfest_train %>%
  # mark missing values
  is.na() %>%
  # cast as data frame again
  as.data.frame() %>%
  pivot_longer(cols =  everything(), names_to = "variable", values_to = "missing") %>%
  ggplot(aes(x = variable, y = missing)) +
    geom_raster() +
    theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1))
```



There are no missing values! No imputation is required.

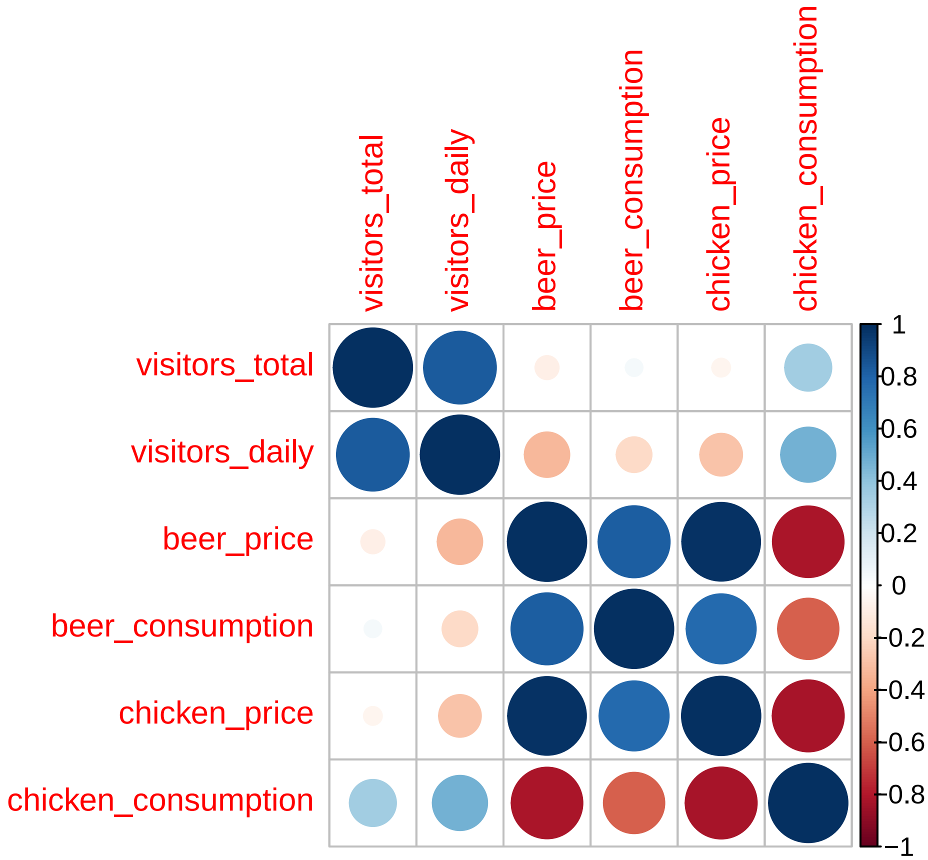
Any multi-collinearity? We can easily compare them in a correlation plot or look at the variance inflation factor (VIF) in advanced and more robust analyses. Multi-collinearity is an important problem: It affects the explainability, interpretability and performance of our (particularly parametric) models. The coefficients fit during regression might become unreliable, and tiny disturbances in training data might change the direction and strength of the relationship. It makes it hard to infer which variables are real predictors. Although it does not affect the prediction performance much, variance inflation increases the risk of overfitting. The variance of a coefficient (j) is given by the following formula[2], where R_j^2 denotes its coefficient of determination for X_j regressed on all other predictors:

$$\mathrm{Var}(\hat{\beta}_j) = \frac{s^2}{(n-1)\,\mathrm{Var}(X_j)} \cdot \frac{1}{1-R_j^2}$$

As the correlation among the predictors increases, the Variance Inflation Factor (VIF), the second term, also increases, hence the coefficient's variance. This situation makes it difficult to infer the significance of the coefficients, even if they are true, and the extrapolation of the results.

```
oktoberfest_train %>%
  # choose all predictor variables
  select(!matches(c("year", "duration"))) %>%
  # create the correlation matrix (values should be between -1 and 1)
  cor() %>%
  # visualize using correlation plot
  corrplot::corrplot()
```

[2] https://en.wikipedia.org/wiki/Variance_inflation_factor



Feature filtering

Are *more features* = *more information*? Not always. Apart from computational burden, we have seen one of them: Multicollinearity. Other problem is the "Curse of dimensionality": When the feature space gets larger, when more features join, surpassing the number of observations, which generally correspond to the rows in our data sets, it becomes what is known as **sparse data**. This deprives the model of learning meaningful patterns and trends from the data set. This could cause the model to overfit the training test.

One approach is filtering out features with only one value or very low unique values. One can do so-called zero variance or near-zero variance filtering.

Another advanced method would be dimension reduction techniques.

We'll just remove the year and the duration, since it obviously does not make sense to use them as explanatory variables.

```r
oktoberfest_train <- oktoberfest_train %>%
  select(-year, -duration)
```

Standardization

Let's continue with the standardization of the predictors so that their contributions to predicting the target value becomes more comparable. We will center and scale using the function `scale()`.

```r
# Since data frames behave as lists, we can iterate by lapply()
scaled_oktoberfest <- oktoberfest_train %>%
  mutate(across(!beer_consumption, ~ as.numeric(scale(.))))
```

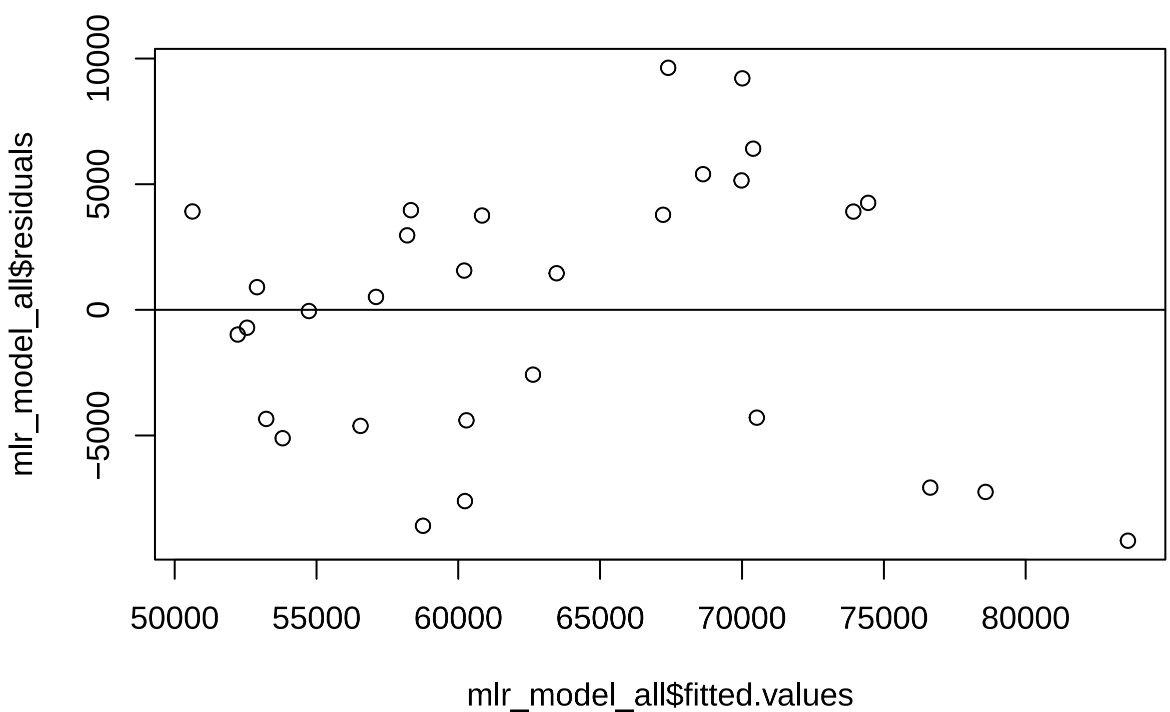
Training / Fitting

We'll train our model using all the variables and all the observation in the training data set. The target variable is beer consumption.

```r
mlr_model_all <- lm(beer_consumption ~ ., data = scaled_oktoberfest)
```

Let's see the distribution of residuals against predicted y-values.

```r
plot(mlr_model_all$fitted.values, mlr_model_all$residuals)
abline(h = 0)
```



We can evaluate the performance now on the test data! We'll use Root Mean Squared Error for this purpose. We should **filter** and **scale** the predictor similarly. Basically all the things we did to the training data set. But there is a crucial catch! We should avoid treatments that would cause **data leakage**, for example using mean of test set column, if we had done any imputation by taking column means or scaling test set with its own mean and standard deviation.

```r
# get mean and standard deviation of the training set
mean_sd <- oktoberfest_train %>%
  select(-beer_consumption) %>%
  mutate(across(everything(), as.numeric)) %>%
  summarise(across(everything(),
                   list(mean = mean, sd = sd)))
```

```r
oktoberfest_test <- read_csv("data/oktoberfest_test.csv")

test_processed <- oktoberfest_test %>%
  select(-c(year, duration, beer_consumption)) %>%
  mutate(across(everything(),
    ~ (.- mean_sd[[paste0(cur_column(), "_mean")]]) /
       mean_sd[[paste0(cur_column(), "_sd")]]
  ))

predictions <- predict(mlr_model_all, newdata = test_processed)
rmse = sqrt(mean((oktoberfest_test$beer_consumption - predictions)^2))
rmse
```

```
## [1] 4564.646
```

Can you spot on a crucial mistake that we've made in the

Demo: Summarizing using tidymodels!

That was a long journey so far. We should always keep in mind to do the same preprocessing steps done on the training set also on the test set, and avoid data leakage in doing so. Wouldn't it much easier to apply these steps in a more structured and organized way? Here, let us introduce the `tidymodels` framework, which follows the conventions of `tidyverse`. Similarly, it also consists of multiple packages that help in different aspects of modelling: * rsample | data partition * recipes | preprocessing * parsnip | training models * workflows | step-wise pipelining * tune | optimizing hyperparameters * yardstick | evaluate performance * broom | formatting * dials | tuning parameter grids

```r
library(tidymodels)

train_set <- read_csv("data/oktoberfest_train.csv")
test_set <- read_csv("data/oktoberfest_test.csv")

          # determine your target and predictor(s)
recipe <- recipe(beer_consumption ~ ., data = train_set) %>%
  update_role(year, new_role = "ID") %>%          # year can be used as an ID
  update_role(duration, new_role = "ignore") %>%  # do not use duration as a predictor
  step_normalize(all_predictors())                # scale and center

# determine modelling "machine"
linear_modeller <- linear_reg() %>%
  set_engine("lm")

wf <- workflow() %>%
  add_model(linear_modeller) %>%
  add_recipe(recipe)

# Fitting on training set
fit <- wf %>% fit(data = train_set)

# Prediction on test_set
preds <- predict(fit, new_data = test_set) %>%
  bind_cols(test_set)
```

```r
# evaluation
metrics <- preds %>%
  metrics(truth = beer_consumption, estimate = .pred)

metrics
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    4565.
## 2 rsq     standard       0.797
## 3 mae     standard    3921.
```

Bonus: How good is your model in predicting the chicken consumption of this year?