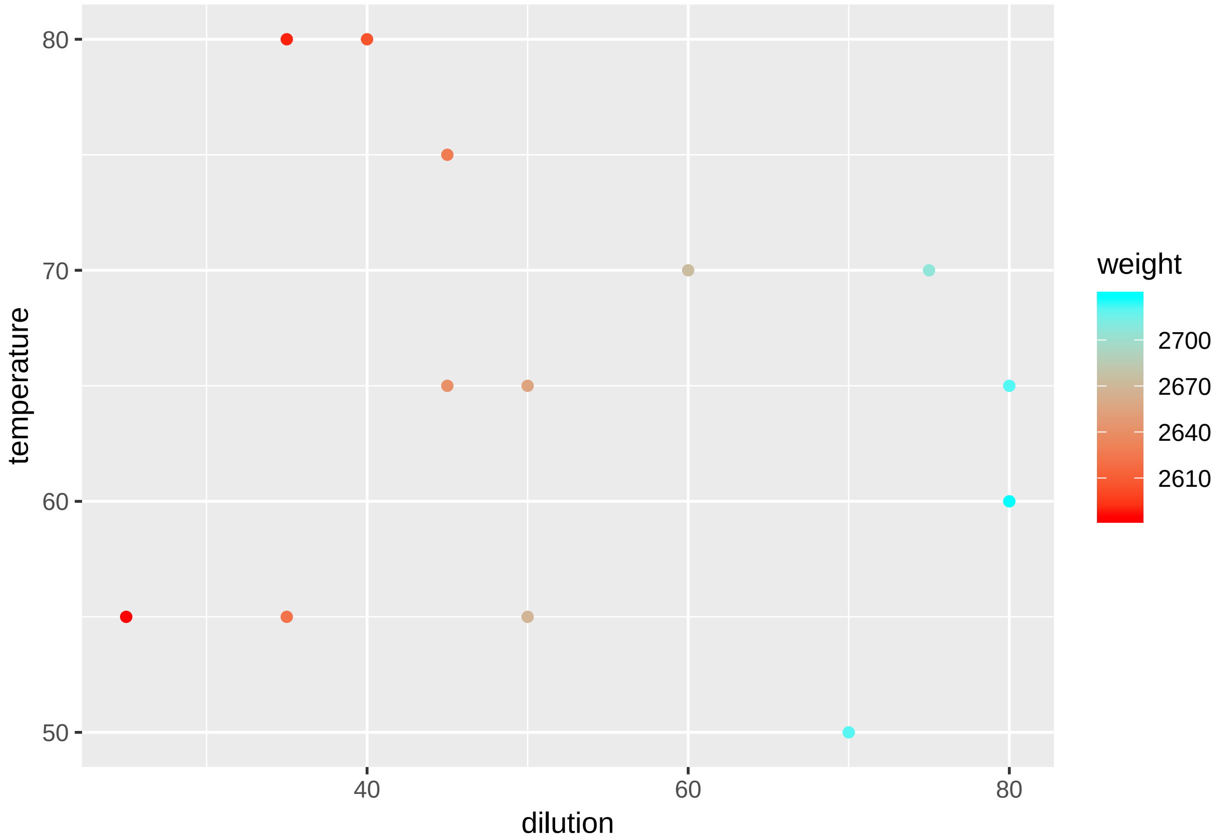
assignment_hoftydom

Dominik Hoftych

December 9, 2018

Visualize the data

```
#ggplot(orig_data, aes(x = dilution, y=temperature, color=weight)) + geom_point() + scale_colour_gradie
ggplot(train_data, aes(x = dilution, y=temperature, color=weight)) + geom_point() + scale_colour_gradie
```




```
#ggplot(test_data, aes(x = dilution, y=temperature, color=weight)) + geom_point() + scale_colour_gradie
#ggplot(outer_data, aes(x = dilution, y=temperature, color=weight)) + geom_point() + scale_colour_gradi

temperature_axis <- seq(30, 100, 5)
dilution_axis <- seq(0, 100, 5)

#tbl <- reshape(data, idvar="temperature", timevar="dilution", direction="wide")[,2:16]
```

Task 1:

In the summary of the polynomial regression with degreess 6,

we can see the fields t-value and p-value [Pr(>|t|)].

The null hypothesis of the statistic is that the true value

of the coefficient is 0.

Use this information to determine to determine what are

degrees of the polynomial best explains the data with

confidence 99%.

Write code that proves (on it's output) that the degrees are really the best.

```
fit_dil1_temp1 <- lm(weight~dilution + temperature, data=train_data)
fit_dil2_temp2  <- lm(weight~poly(dilution, 2)+poly(temperature, 2), data=train_data)
fit_dil2_temp3  <- lm(weight~poly(dilution, 2)+poly(temperature, 3), data=train_data)
fit_dil3_temp2  <- lm(weight~poly(dilution, 3)+poly(temperature, 2), data=train_data)
fit_dil3_temp3  <- lm(weight~poly(dilution, 3)+poly(temperature, 3), data=train_data)
fit_dil3_temp4  <- lm(weight~poly(dilution, 3)+poly(temperature, 4), data=train_data)
fit_dil4_temp3  <- lm(weight~poly(dilution, 4)+poly(temperature, 3), data=train_data)
fit_dil4_temp4  <- lm(weight~poly(dilution, 4)+poly(temperature, 4), data=train_data)

# summary(fit_dil1_temp1)
 summary(fit_dil2_temp2)
```

```
##
## Call:
## lm(formula = weight ~ poly(dilution, 2) + poly(temperature, 2),
##     data = train_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.9100 -0.1977  0.0334  0.2779  0.6649
##
## Coefficients:
##                        Estimate Std. Error  t value Pr(>|t|)
## (Intercept)           2656.9869     0.1524 17430.23  < 2e-16 ***
## poly(dilution, 2)1     163.6901     0.5999   272.88  < 2e-16 ***
## poly(dilution, 2)2     -18.7804     0.5639   -33.30 7.21e-10 ***
## poly(temperature, 2)1  -43.2651     0.5698   -75.94 1.01e-12 ***
```

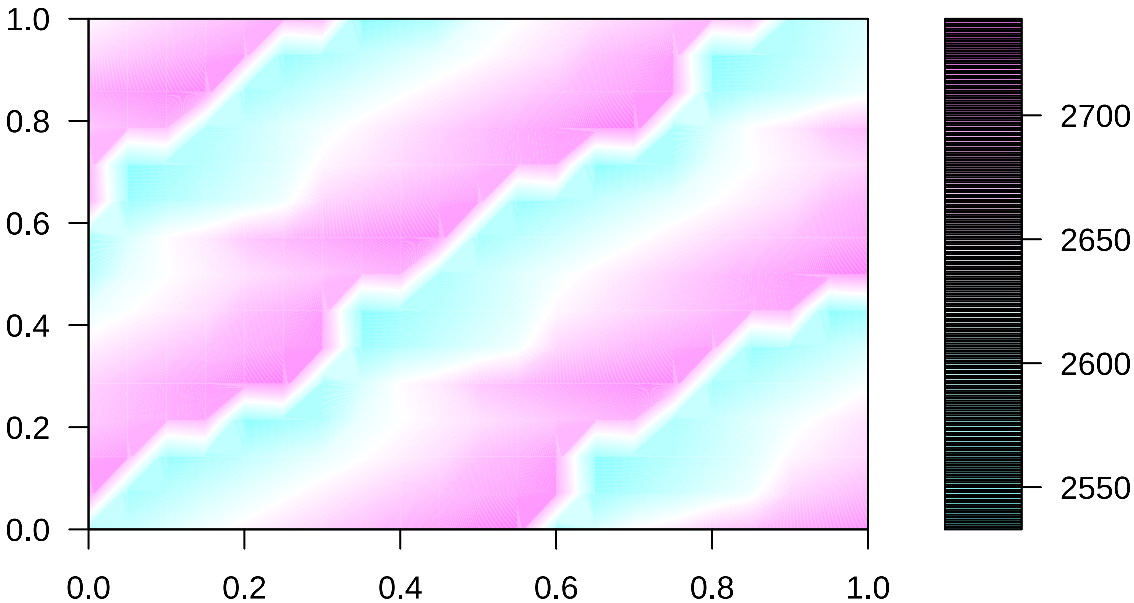
```
## poly(temperature, 2)2   -1.7057     0.5943    -2.87   0.0208 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5496 on 8 degrees of freedom
## Multiple R-squared:  0.9999, Adjusted R-squared:  0.9999
## F-statistic: 2.553e+04 on 4 and 8 DF,  p-value: < 2.2e-16
```

```
# summary(fit_dil2_temp3)
 summary(fit_dil3_temp2)
```

```
##
## Call:
## lm(formula = weight ~ poly(dilution, 3) + poly(temperature, 2),
##     data = train_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.20737 -0.17024  0.01024  0.10294  0.29445
##
## Coefficients:
##                        Estimate Std. Error   t value Pr(>|t|)
## (Intercept)          2656.98692    0.05975 44469.221  < 2e-16 ***
## poly(dilution, 3)1     163.65519    0.23518   695.879  < 2e-16 ***
## poly(dilution, 3)2     -18.93732    0.22226   -85.203 8.08e-12 ***
## poly(dilution, 3)3       1.60042    0.23839     6.714 0.000274 ***
## poly(temperature, 2)1  -43.95023    0.24554  -178.996 4.48e-14 ***
## poly(temperature, 2)2   -1.54432    0.23418    -6.595 0.000306 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2154 on 7 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 1.33e+05 on 5 and 7 DF,  p-value: < 2.2e-16
```

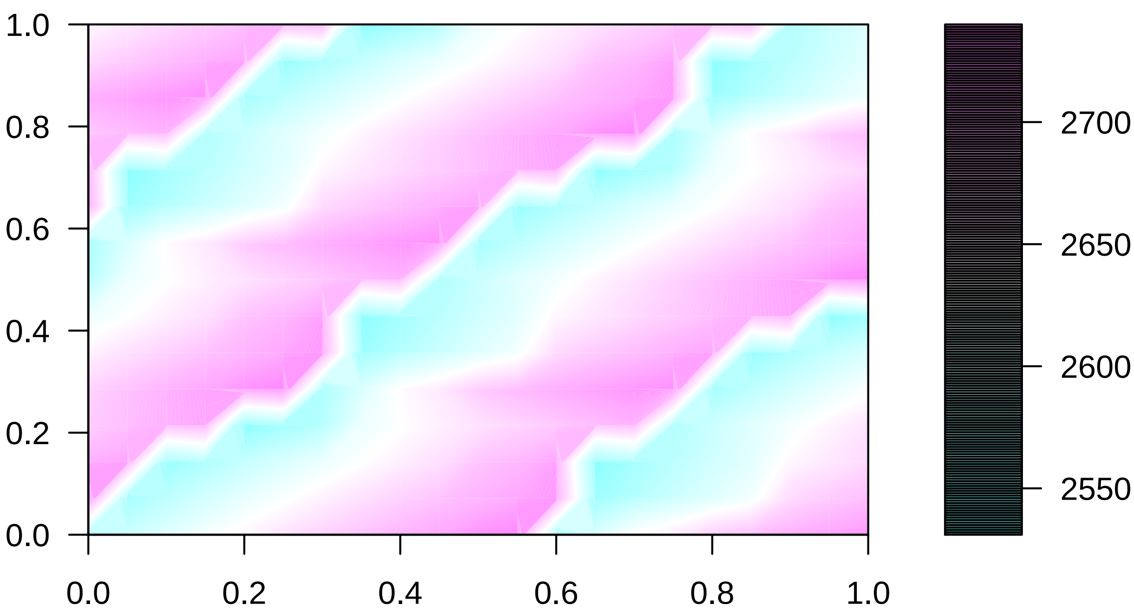
```
 optimal_poly <- fit_dil3_temp2
# summary(fit_dil3_temp3)
# summary(fit_dil3_temp4)
# summary(fit_dil4_temp3)
# summary(fit_dil4_temp4)


# Visualize the regressor
prediction22 <- predict(fit_dil2_temp2, test_data)
filled.contour(matrix(prediction22, nrow=21, ncol=15), nlevels=255)
```

```
## Warning in matrix(prediction22, nrow = 21, ncol = 15): data length [78] is
## not a sub-multiple or multiple of the number of rows [21]
```




```r
prediction32 <- predict(fit_dil3_temp2, test_data)
filled.contour(matrix(prediction32, nrow=21, ncol=15), nlevels=255)
```

```
## Warning in matrix(prediction32, nrow = 21, ncol = 15): data length [78] is
## not a sub-multiple or multiple of the number of rows [21]
```




```r
# with this approach, we increase the degree of the polynomial and
# call summary for each fit, in which we can see p-values for
# corresponding degrees of the polynomials. we are looking for the
# lowest p-value possible (which also satisfies <0.01 condition),
# which is 2.2e-16 for both 2-2 and 3-2 degrees of polynomials of
#dilution and temperature respectively.
confint(fit_dil3_temp2, level=0.99)
```

```
##                        0.5 %      99.5 %
## (Intercept)         2656.7778328 2657.1960133
```

```
## poly(dilution, 3)1     162.8321863  164.4781878
## poly(dilution, 3)2     -19.7151157  -18.1595247
## poly(dilution, 3)3       0.7661888    2.4346425
## poly(temperature, 2)1  -44.8094803  -43.0909715
## poly(temperature, 2)2   -2.3638165   -0.7248305
```

```
# confint function can be used to see the CI of our model
```

Task 2:

If we we would like to use natural spline, resp. smoothing spline

instead of polynomial, what would number of degrees of freedom would be required

for each predictor variable, again best explaining the data

with confidence 99%? Write code that proves the answer.

Note: Instead of summary(), which is not applicable here,

you need to use the anova() statistic.

For smoothing spline, assume that degrees of freedom is a natural number.

Use the functions `gam` and `s` instead of `lm` resp. `ns`.

```
# Natural spline
fit_ns22 <- lm(weight~ns(dilution, 2)+ns(temperature, 2), data=train_data)
fit_ns23 <- lm(weight~ns(dilution, 2)+ns(temperature, 3), data=train_data)
fit_ns32 <- lm(weight~ns(dilution, 3)+ns(temperature, 2), data=train_data)

anova(fit_ns22, fit_ns23, fit_ns32)
```

```
## Analysis of Variance Table
##
## Model 1: weight ~ ns(dilution, 2) + ns(temperature, 2)
## Model 2: weight ~ ns(dilution, 2) + ns(temperature, 3)
## Model 3: weight ~ ns(dilution, 3) + ns(temperature, 2)
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1      8 2.8062
## 2      7 2.7604  1   0.04584 0.1162 0.7432
## 3      7 1.5258  0   1.23452
```

```
optimal_ns <- fit_ns22

# The optimal natural spline is fit_ns22 (2 degrees of freedom in each variable).
#If we increase the degrees of freedom
# in either variable, we do not get anything better explaining the data.
```

```
# Smoothing splines
fit_s32 <- gam(weight~s(dilution, 3)+s(temperature, 2), data=train_data)
fit_s33 <- gam(weight~s(dilution, 3)+s(temperature, 3), data=train_data)
fit_s42 <- gam(weight~s(dilution, 4)+s(temperature, 2), data=train_data)

anova(fit_s32, fit_s33, fit_s42)
```

```
## Analysis of Deviance Table
##
## Model 1: weight ~ s(dilution, 3) + s(temperature, 2)
## Model 2: weight ~ s(dilution, 3) + s(temperature, 3)
## Model 3: weight ~ s(dilution, 4) + s(temperature, 2)
##   Resid. Df Resid. Dev          Df Deviance Pr(>Chi)
## 1         7     4.0677
## 2         6     3.9607  1.0000e+00  0.10707   0.6871
## 3         6     1.2539 -5.9347e-05  2.70679
```

```
optimal_s <- fit_s32
# analogously as for natural splines, fit_s32 is the best one.
```

Task 3:

Compute Residual Sum of Squares (RSS) for each of the found models on testing data and outer data.

compare which model performs the best according to this criterion.

Discuss the interpolating and extrapolating capabilities of the models.

Note: On testing data, this evaluates the interpolating capabilities, wheter on the outer

data, this evaluates the extrapolating capabilities of the learned model.

```
poly.test <- predict(optimal_poly, newdata = list(temperature = test_data$temperature, dilution = test_d
poly.outer <- predict(optimal_poly, newdata = list(temperature = outer_data$temperature, dilution = oute

ns.test <- predict(optimal_ns, newdata = list(temperature = test_data$temperature, dilution = test_data$
```

```r
ns.outer <- predict(optimal_ns, newdata = list(temperature = outer_data$temperature, dilution = outer_da

s.test <- predict(optimal_s, newdata = list(temperature = test_data$temperature, dilution = test_data$di
s.outer <- predict(optimal_s, newdata = list(temperature = outer_data$temperature, dilution = outer_data

mse.poly.test <- mean((poly.test - test_data$weight)^2)
mse.poly.outer <- mean((poly.outer - outer_data$weight)^2)
cat("polynomnial interpolation error", mse.poly.test)
```

```
## polynomnial interpolation error 0.4225099
```

```r
cat("\npolynomial extrapoliation error", mse.poly.outer)
```

```
##
## polynomial extrapoliation error 19.06468
```

```r
mse.ns.test <- mean((ns.test - test_data$weight)^2)
mse.ns.outer <- mean((ns.outer - outer_data$weight)^2)
cat("\nnatural spline interpolation error", mse.ns.test)
```

```
##
## natural spline interpolation error 2.583389
```

```r
cat("\nnatural spline extrapoliation error",mse.ns.outer)
```

```
##
## natural spline extrapoliation error 224.7038
```

```r
mse.s.test <- mean((s.test - test_data$weight)^2)
mse.s.outer <- mean((s.outer - outer_data$weight)^2)
cat("\nsmoothing spline interpolation error", mse.s.test)
```

```
##
## smoothing spline interpolation error 3.418082
```

```r
cat("\nsmoothing spline extrapoliation error",mse.s.outer)
```

```
##
## smoothing spline extrapoliation error 243.0891
```

```r
# polynomials extrapolate the best, which is quite surprising as one would expect
#the smoothing splines to perform the best.
```

Task 4:

Plot each of those models on `orig_data`. Discuss.

Then, for each of the methods (polynomial, natural spline, smoothing spline)

also learn and plot a model with:

*** 5 degrees of freedom for both regressor variables for polynomial.**

*** 6 degrees of freedom for both regressor variables for natural spline.**

*** 6 degrees of freedom for both regressor variables for smoothing spline.**

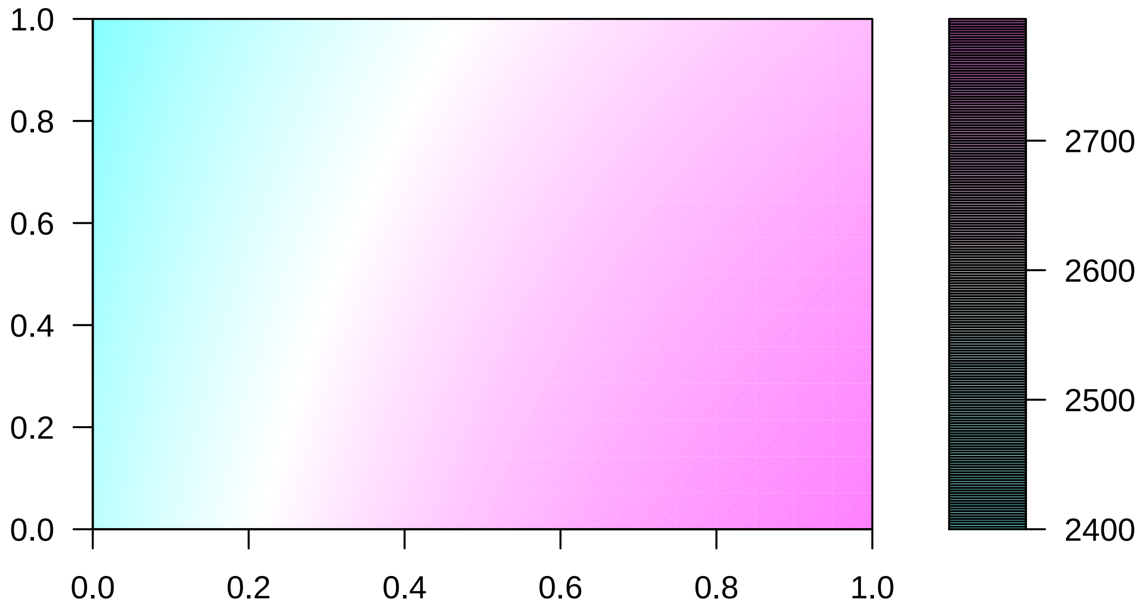
(Note: That is signifianctly more than optimal.)

Do you visually see a difference compared to the models with best numbers of DoF?

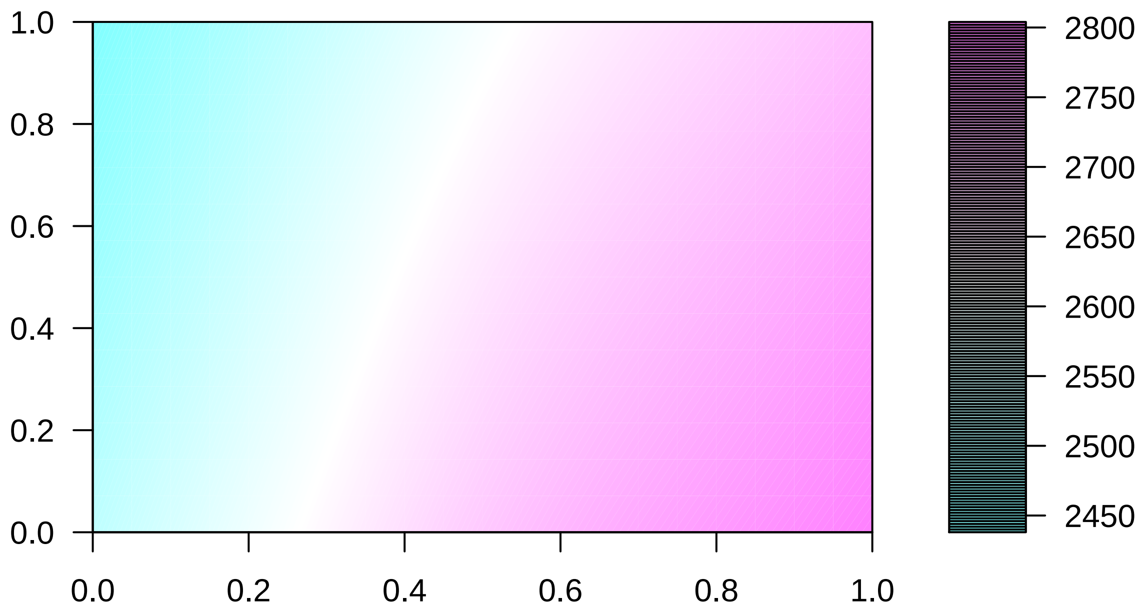
Discuss the interpolating (test_data)

and extrapolating (outer_data) capabilities.

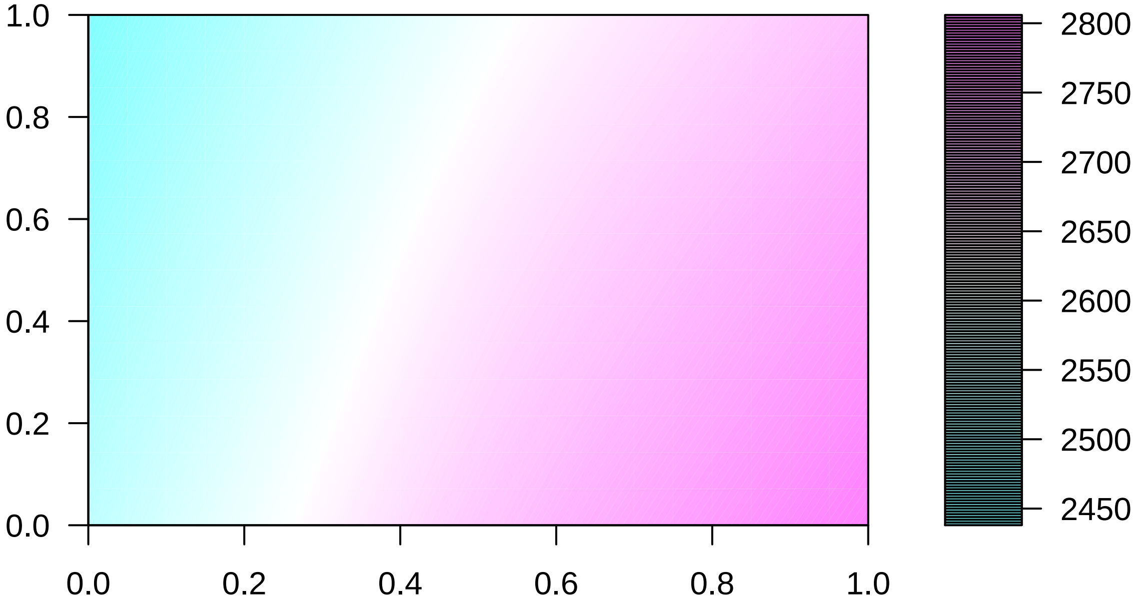
```
filled.contour(matrix(predict(optimal_poly, orig_data), nrow=21, ncol=15), nlevels=255)
```



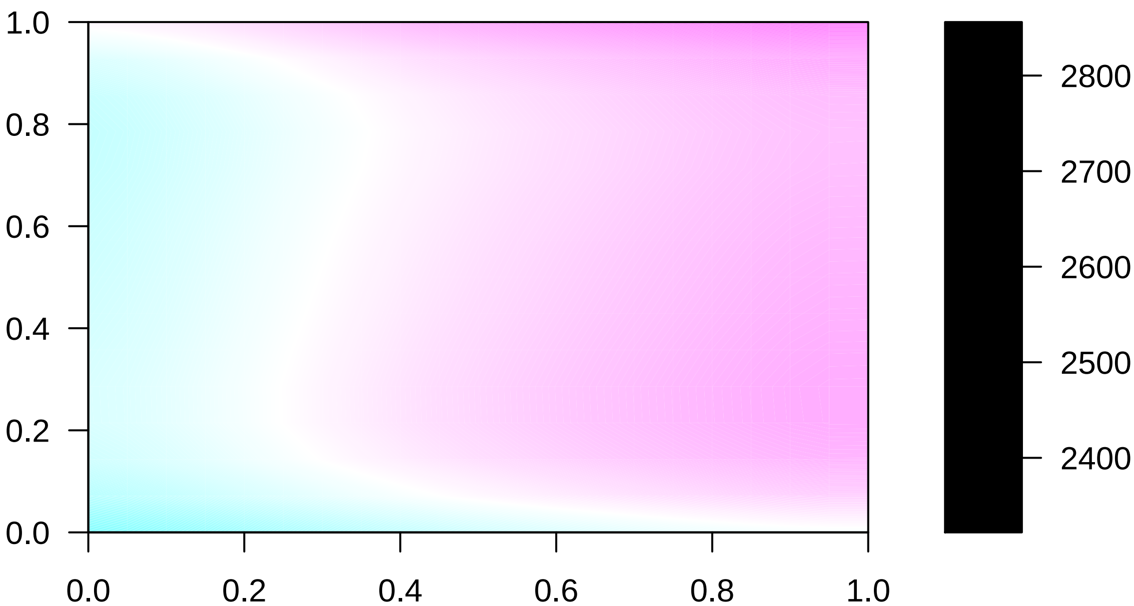

```r
filled.contour(matrix(predict(optimal_ns, orig_data), nrow=21, ncol=15), nlevels=255)
```



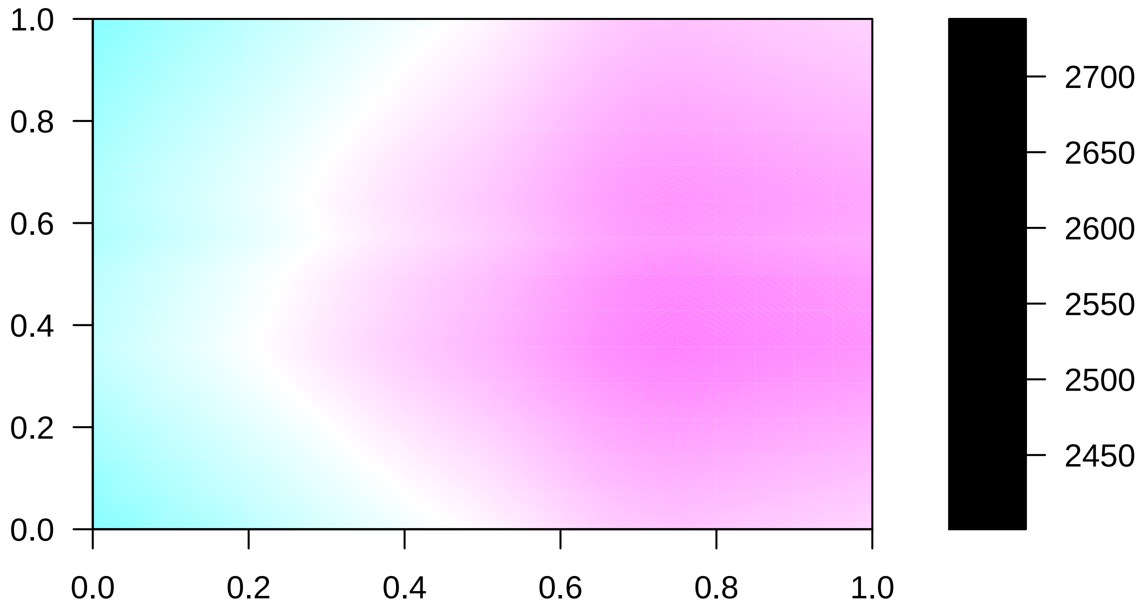

```r
filled.contour(matrix(predict(optimal_s, orig_data), nrow=21, ncol=15), nlevels=255)
```



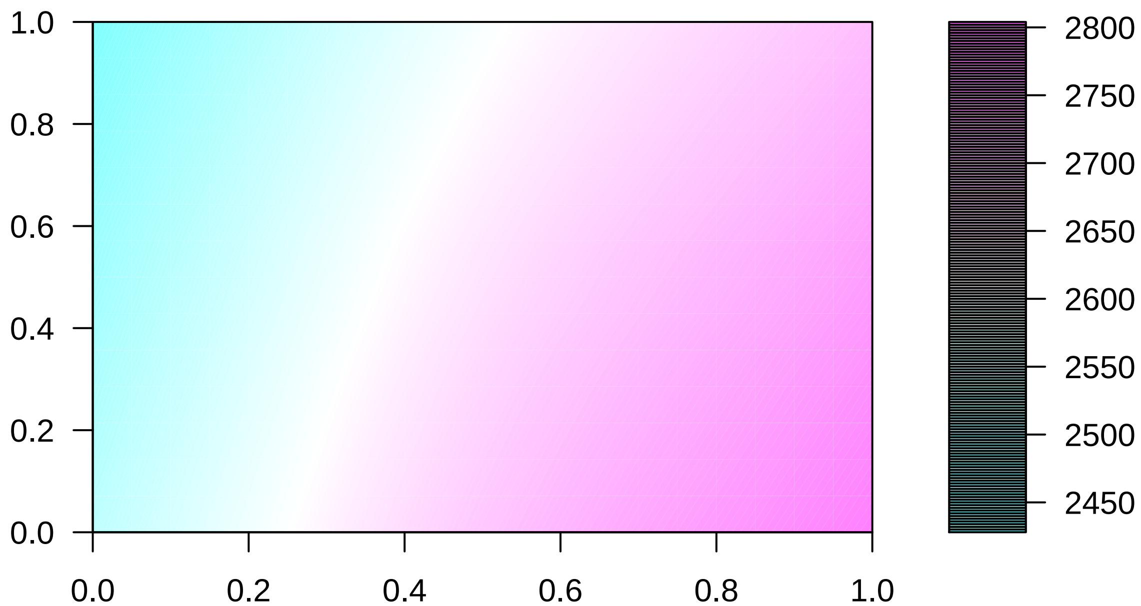

```
# between models above, there is no much difference. They have quite similar degrees
#of freedom which might be an explanation of it.

poly_55<- lm(weight~poly(dilution, 5)+poly(temperature, 5), data=train_data)
ns_66 <- lm(weight~ns(dilution, 6)+ns(temperature, 6), data=train_data)
s_66<- gam(weight~s(dilution, 6)+s(temperature, 6), data=train_data)

filled.contour(matrix(predict(poly_55, orig_data), nrow=21, ncol=15), nlevels=255)
```




```
filled.contour(matrix(predict(ns_66, orig_data), nrow=21, ncol=15), nlevels=255)
```




```
filled.contour(matrix(predict(s_66, orig_data), nrow=21, ncol=15), nlevels=255)
```




```
# models above are clearly overfitted. smoothing splines seem to work well as they are the
#only one looking 'nonlinearly' compared to polynomials and natural splines or the optimal models discu
```