Polynomial Regression Demo, Winter 2024

Zhiwei Tan, Katrin Jõgi

Introduction

When the relationship between the output variable and the predictor variable is not linear, nonlinear regression can be used to fit the data to a model. Types of nonlinear regression are polynomial regression, stepwise regression and splines. The current demo focuses on the polynomial regression.

Polynomial regression offers flexibility in capturing non-linear relationships by increasing the polynomial order. This allows for a closer fit to the data, potentially revealing complex patterns. However, this flexibility also increases the risk of overfitting, where the model memorizes noise instead of learning generalizable trends.

Context

For the demo, we are using the Boston housing data set from the MASS library. This dataset contains information collected by the U.S Census Service concerning housing in the area of Boston Mass. The Boston data frame has 506 observations on 14 variables.

dataset columns can be seen using help(Boston)

- CRIM - per capita crime rate by town
- ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS - proportion of non-retail business acres per town.
- CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- NOX - nitric oxides concentration (parts per 10 million)
- RM - average number of rooms per dwelling
- AGE - proportion of owner-occupied units built prior to 1940
- DIS - weighted distances to five Boston employment centres
- RAD - index of accessibility to radial highways
- TAX - full-value property-tax rate per $10,000
- PTRATIO - pupil-teacher ratio by town
- B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT - % lower status of the population
- MEDV - Median value of owner-occupied homes in $1000's

For the purposes of the current demo, the outcome variable is MEDV - Median value of owner-occupied homes in $1000's and the predictor variable is LSTAT - % lower status of the population.

We set out to explore whether polynomial regression is appropriate when modeling the effect of % lower status of the population on the housing values.

##Load data and data processing

Load data Boston housing data from MASS package. Check for missing values and data types. Split the data for training and test.

```
data('Boston') #Load data

sum(is.na(Boston)) #check for missing values

## [1] 0

str(Boston) #view format of the data

## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524
0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...

head(Boston) #view top six observations

##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black
lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90
4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90
9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83
4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63
2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90
5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12
5.21
##   medv
## 1 24.0
```

```
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

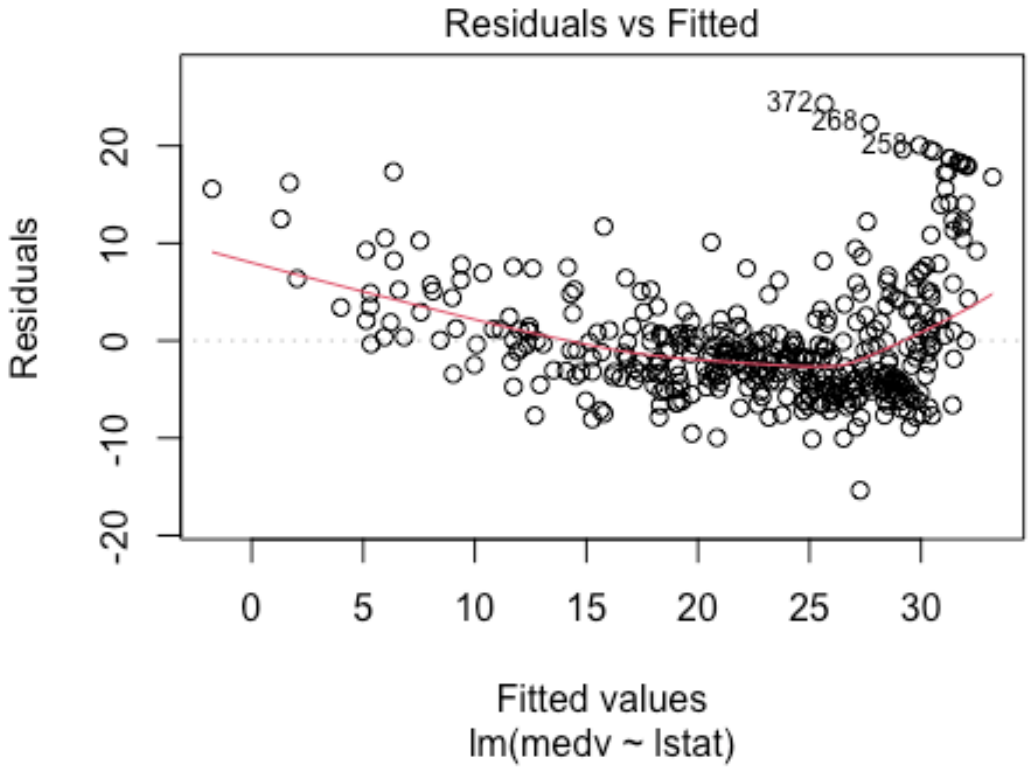
In order to be able to test the generalizability of the selected model, we split our data set into the train and test set with 80/20 split.

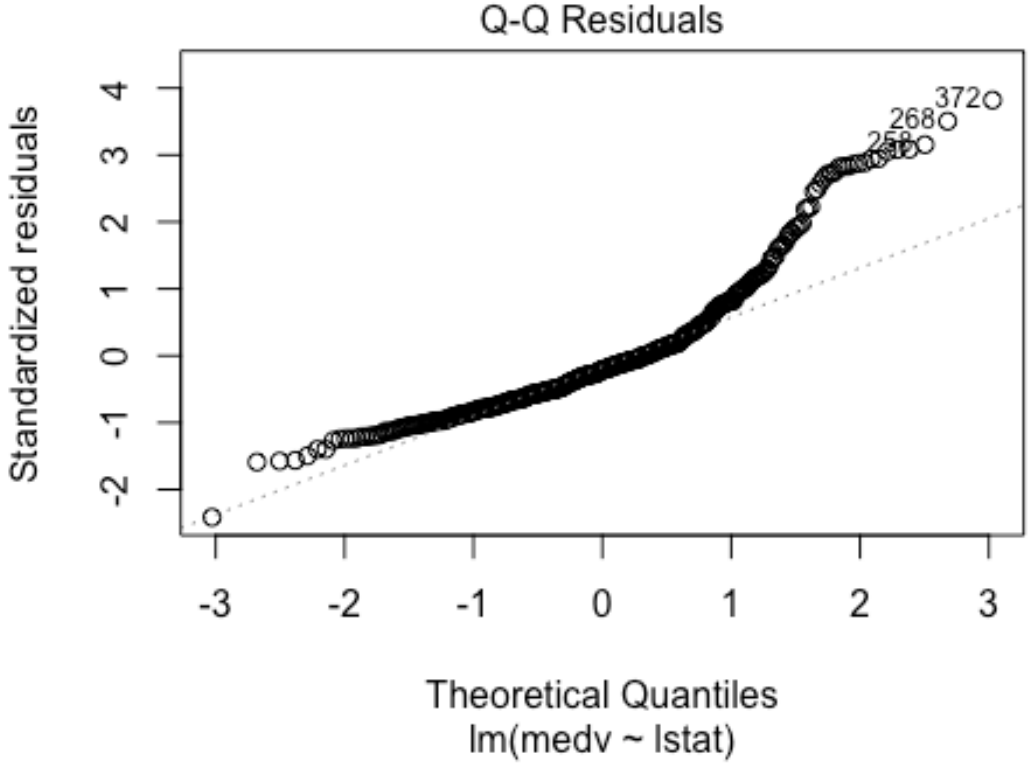
```
set.seed(1234)

training.samples <- Boston$medv %>% createDataPartition(p = 0.8, list =
FALSE)

train.data  <- Boston[training.samples, ]

test.data <- Boston[-training.samples, ]
```

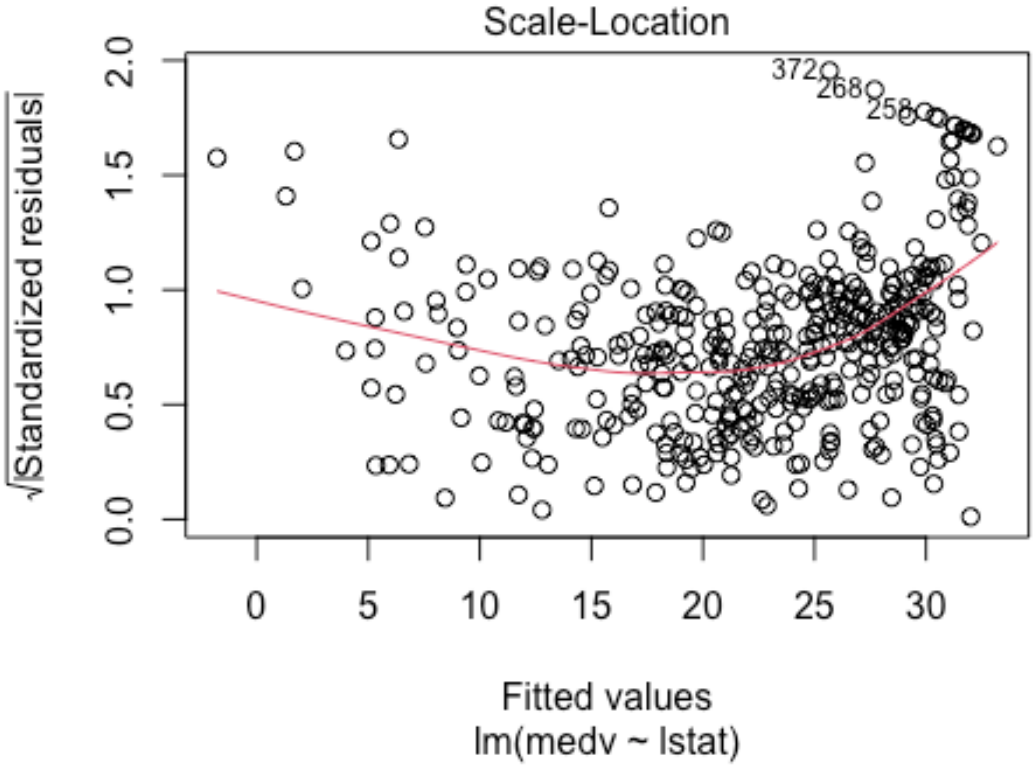
Linear Regression

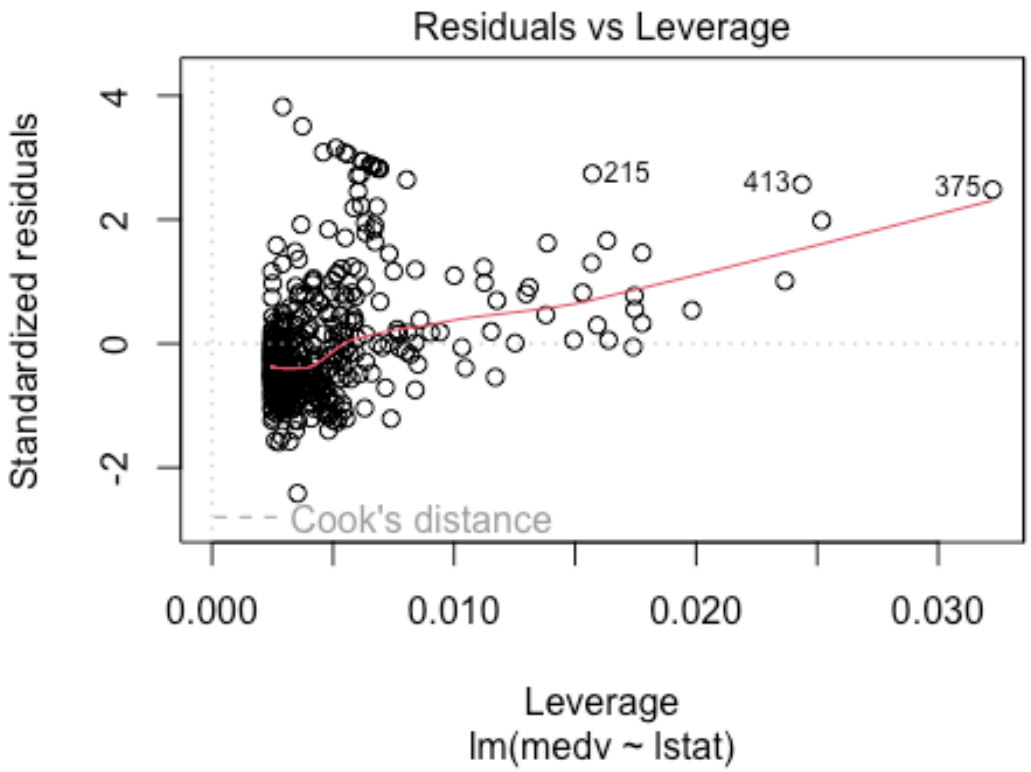
We want to compare a linear regression on a polynomial regression. First, we check the assumptions of the linear model and look for any outliers.

```
Boston.reg <- lm(medv ~ lstat, data = train.data)

plot(Boston.reg)
```

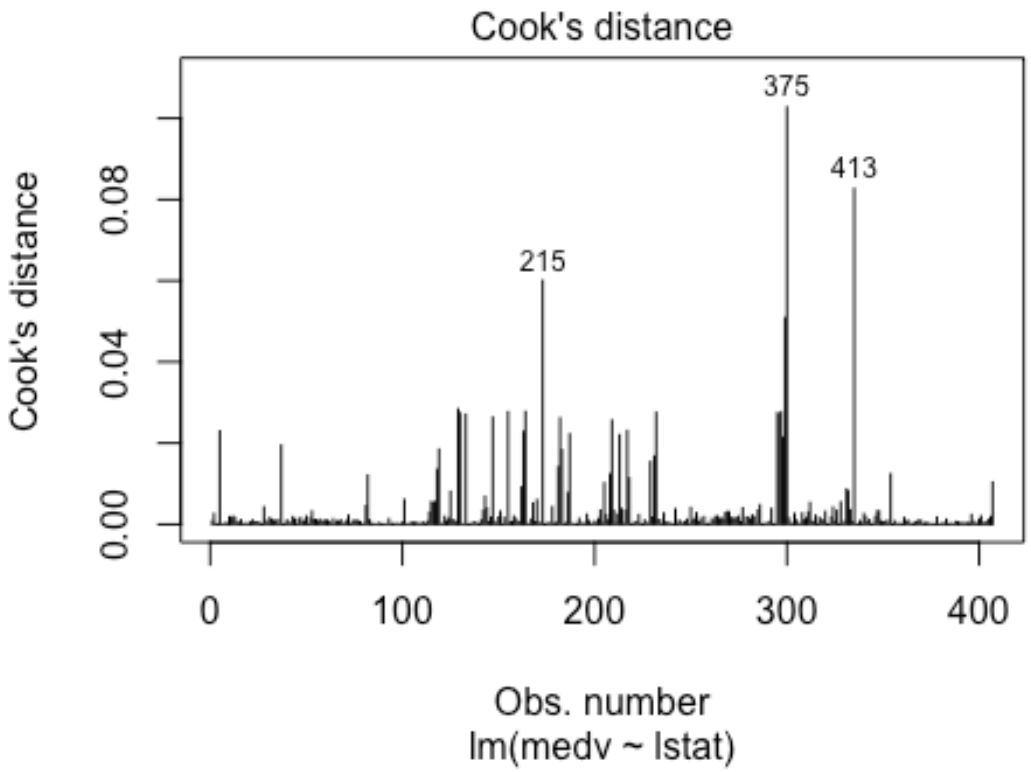

Residuals vs Fitted
372
268
258
Residuals
20
10
0
-10
-20
0
5
10
15
20
25
30
Fitted values
lm(medv ~ lstat)


Q-Q Residuals
Standardized residuals
-2
-1
0
1
2
3
4
372
268
-3
-2
-1
0
1
2
3
Theoretical Quantiles
lm(medv ~ lstat)


Scale-Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
2.0
372
268
258
0
5
10
15
20
25
30
Fitted values
lm(medv ~ lstat)




```r
plot(Boston.reg, 4) #Cook's distance
```



Polynomial regression model is sensitive to outliers, even one or two outliers can badly affect its performance. We do not observe any potentially influential points on the Residuals vs Leverage plot.

The Scale-Location plot does not display any specific shape, thus there is not enough evidence against the assumption of equality of residual variance across the range of the fitted values.

The Residuals vs Fitted plot suggests that the the assumption of linearity is not necessarily met across the range of the fitted values.

The Normal Q-Q plot indicates that, though not definitive, there is not enough evidence against the assumption of normality across the range of the fitted values.

#Summary and visual of the linear model

```
summary(Boston.reg)

##
## Call:
## lm(formula = medv ~ lstat, data = train.data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -15.369  -4.201  -1.336   2.126  24.324
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.87586    0.63594   54.84   <2e-16 ***
## lstat       -0.96532    0.04352  -22.18   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.376 on 405 degrees of freedom
## Multiple R-squared:  0.5485, Adjusted R-squared:  0.5474
## F-statistic: 492.1 on 1 and 405 DF,  p-value: < 2.2e-16
```

```
ggplot(train.data, aes(x = lstat, y = medv)) +  geom_point(alpha = 0.2) +
geom_smooth(method = "lm", se = FALSE) + labs(title = "Linear regression", x
= "% lower status population", y = "median home values")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

R-squared for the linear model is 0.5485, which means 54.85% of the variance in the housing prices (outcome variable) is explained by the % of lower status population (predictor variable)

It can be observed visually that the observations do not spread uniformly around the linear line of best fit. Curvature in the data can be observed, suggesting that a polynomial regression line could better explain the data.

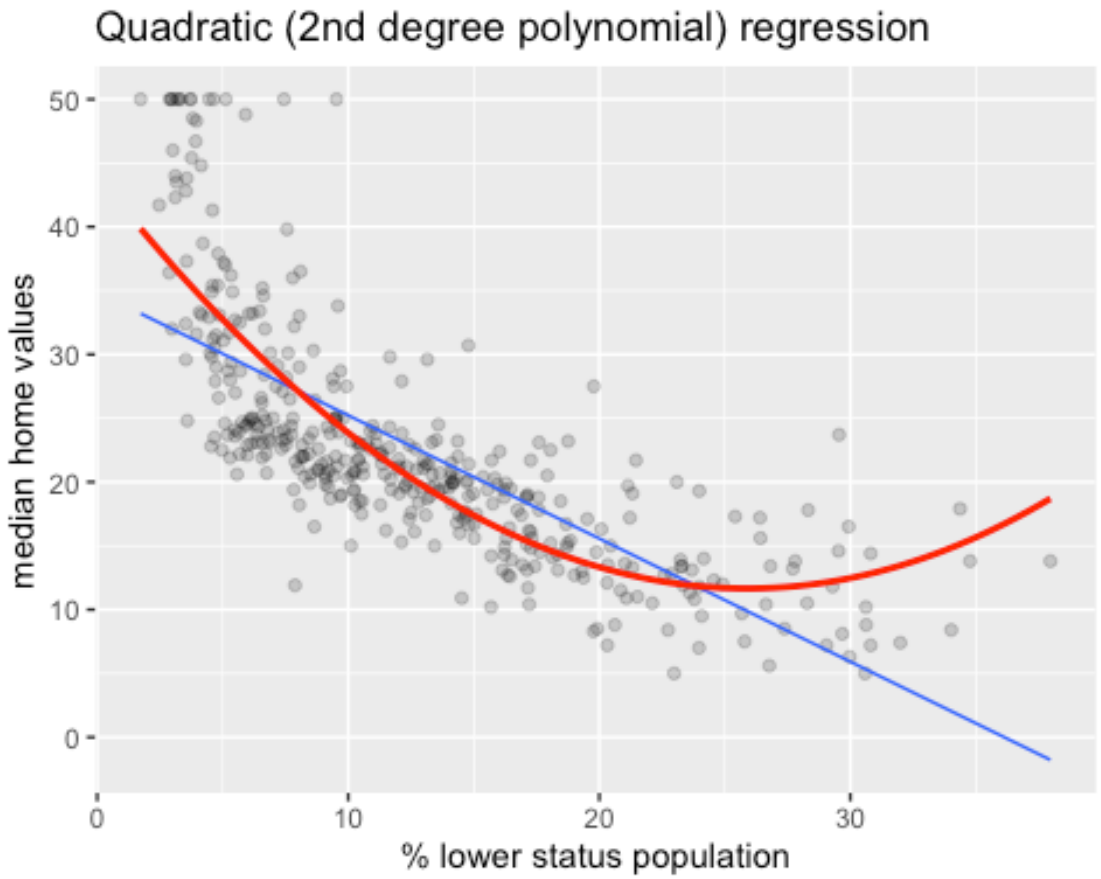
Polynomial regression

Polynomial regression is a type of linear regression that can capture nonlinear relationships between predictor and outcome variables. It achieves this by fitting smooth curves (e.g., quadratic, cubic) to the data using least squares. Similar to linear regression, R-squared helps evaluate the model's fit, with higher values indicating better performance.

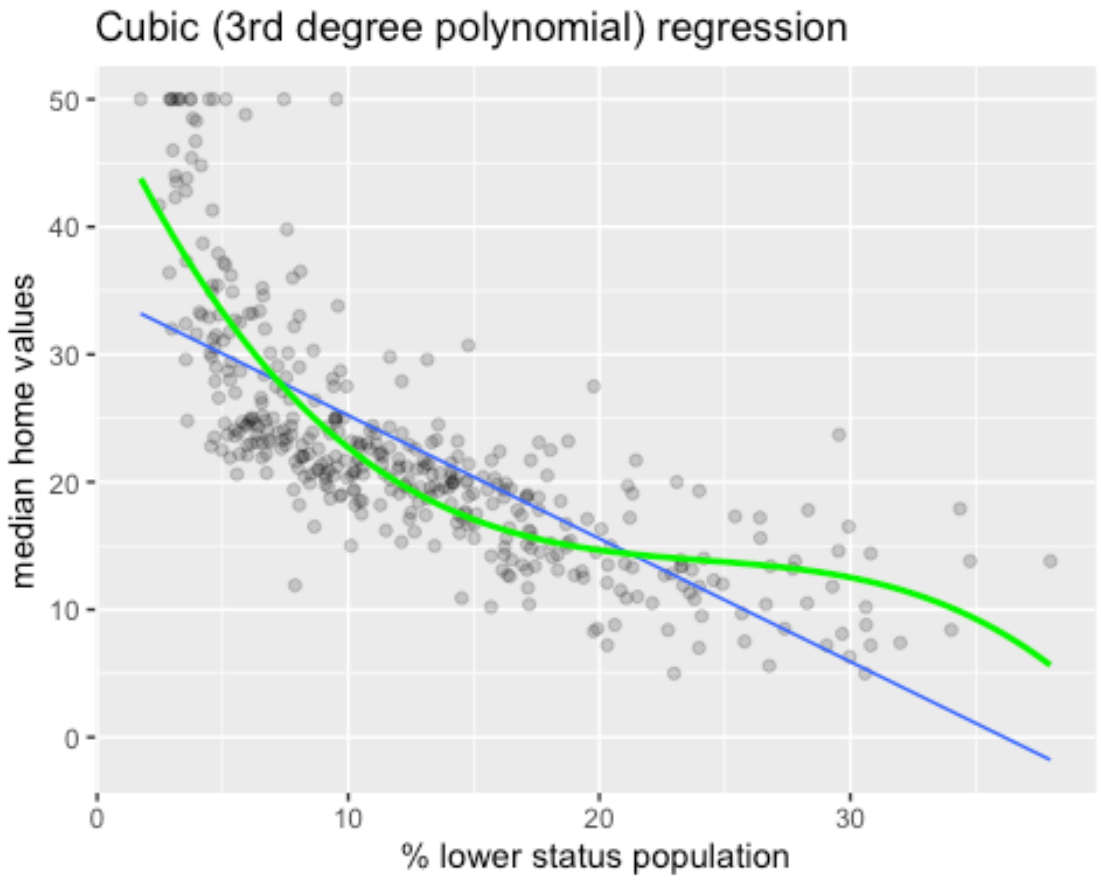
Quadratic and Cubic regression

Lets visualize the relationship between the outcome and the predictor variable with 2nd degree (quadratic) and 3rd degree (cubic) polynomial fits to see if we identify potential non-linearity. The observation values are set to 20% opacity so we can see the spread, and linear and polynomial lines have been overlapped.

```
ggplot(train.data, aes(x = lstat, y = medv)) + geom_point(alpha = 0.2) +
geom_smooth(method = "lm", se = FALSE, size = 0.5) + geom_smooth(method =
"lm", formula = y ~ poly(x, 2), se = FALSE, color = "red") + labs(title =
"Quadratic (2nd degree polynomial) regression", x = "% lower status
population", y = "median home values")

## `geom_smooth()` using formula = 'y ~ x'
```




```
ggplot(train.data, aes(x = lstat, y = medv)) +  geom_point(alpha = 0.2) +
geom_smooth(method = "lm", se = FALSE, size = 0.5) + geom_smooth(method =
"lm", formula = y ~ poly(x, 3), se = FALSE, color = "green") + labs(title =
"Cubic (3rd degree polynomial) regression", x = "% lower status population",
y = "median home values")

## `geom_smooth()` using formula = 'y ~ x'
```



Based on visual inspection, the polynomial regression lines seem to follow the data more closely, but it is not obvious which is performing better of the two.

Polynomial code

Although polynomial regression allows for a nonlinear relationship between predictor and outcome variables, it is still considered linear regression since it is linear in the regression coefficients. Thus, standard linear regression software can be used. The linear model is extended by adding extra predictors, obtained by raising the original predictors to a power. Since pure U-shaped or S-shaped relationships are rare, polynomials are usually specified in addition to a linear term in order to pick up deviations from a linear relationship.

Why not simply add lstat^2 directly? Standard linear regression software doesn't recognize terms like lstat^2 on their own.

```
Boston.poly.wrong.1 <- lm(medv ~ lstat + lstat^2, data=train.data)

summary(Boston.poly.wrong.1)

##
## Call:
## lm(formula = medv ~ lstat + lstat^2, data = train.data)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -15.369  -4.201  -1.336   2.126  24.324
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.87586    0.63594   54.84   <2e-16 ***
## lstat       -0.96532    0.04352  -22.18   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.376 on 405 degrees of freedom
## Multiple R-squared:  0.5485, Adjusted R-squared:  0.5474
## F-statistic: 492.1 on 1 and 405 DF,  p-value: < 2.2e-16
```

Polynomial regression requires transforming the predictor variable into its polynomial form.

- poly(x, degree, raw = TRUE): This function takes the predictor x, the desired polynomial degree (e.g., 2 for squared), and sets raw = TRUE to ensure the coefficients align with the I() function coefficients.
- I(x^power): This function takes the predictor x raised to the desired power.

Note: While raw = TRUE in poly() might lead to slightly different coefficients compared to without it, interpreting polynomial coefficients can be challenging anyway. Both methods ultimately fit the same curve.

```
Boston.poly.2.P <- lm(medv ~ lstat + poly(lstat, 2, raw = TRUE),
data=train.data) #poly()

summary(Boston.poly.2.P)

##
## Call:
## lm(formula = medv ~ lstat + poly(lstat, 2, raw = TRUE), data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.4293  -3.9840  -0.4439   2.5059  25.4111
##
## Coefficients: (1 not defined because of singularities)
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  44.044838   0.986032   44.67   <2e-16 ***
## lstat                        -2.501980   0.141721  -17.65   <2e-16 ***
## poly(lstat, 2, raw = TRUE)1        NA         NA      NA       NA
## poly(lstat, 2, raw = TRUE)2  0.048314   0.004293   11.26   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.57 on 404 degrees of freedom
```

```
## Multiple R-squared:  0.6563, Adjusted R-squared:  0.6546
## F-statistic: 385.7 on 2 and 404 DF,  p-value: < 2.2e-16
```

```
Boston.poly.2.I <- lm(medv ~ lstat + I(lstat^2), data=train.data) #I()
```

```
summary(Boston.poly.2.I)
```

```
##
## Call:
## lm(formula = medv ~ lstat + I(lstat^2), data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.4293  -3.9840  -0.4439   2.5059  25.4111
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 44.044838   0.986032   44.67   <2e-16 ***
## lstat       -2.501980   0.141721  -17.65   <2e-16 ***
## I(lstat^2)   0.048314   0.004293   11.26   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.57 on 404 degrees of freedom
## Multiple R-squared:  0.6563, Adjusted R-squared:  0.6546
## F-statistic: 385.7 on 2 and 404 DF,  p-value: < 2.2e-16
```

#Comparison of the linear and quadratic model Coefficients in polynomial regression represent the change in the outcome variable for a one-unit increase in the corresponding polynomial term of the predictor variable. While interpreting these coefficients can be complex, the overall R-squared increase (e.g., from 54.85% to 65.63%) suggests the polynomial model is a better fit in this case.

#Higher order polynomial models Since the quadratic model was an improvement over the linear model, we decided to explore higher order polynomial models, to see if we can further improve our model.

The I() function requires including all polynomial terms up to the desired power. For example, including only I(lstat^3) misses the essential second-order term.

```
Boston.poly.wrong.2 <- lm(medv ~ lstat + I(lstat^3), data=train.data)
```

```
summary(Boston.poly.wrong.2)
```

```
##
## Call:
## lm(formula = medv ~ lstat + I(lstat^3), data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.7250  -3.9524  -0.5647   2.5254  24.8943
```

```
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.075e+01  8.186e-01  49.777   <2e-16 ***
## lstat       -1.717e+00  8.474e-02 -20.266   <2e-16 ***
## I(lstat^3)   8.353e-04  8.356e-05   9.997   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.716 on 404 degrees of freedom
## Multiple R-squared:  0.6381, Adjusted R-squared:  0.6363
## F-statistic: 356.1 on 2 and 404 DF,  p-value: < 2.2e-16
```

```
Boston.poly.3 <- lm(medv ~ lstat + I(lstat^2) + I(lstat^3), data=train.data)
```

```
summary(Boston.poly.3)
```

```
##
## Call:
## lm(formula = medv ~ lstat + I(lstat^2) + I(lstat^3), data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.5826  -3.5112  -0.3461   2.8153  26.5501
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 50.6720734  1.6263515  31.157  < 2e-16 ***
## lstat       -4.2641712  0.3756696 -11.351  < 2e-16 ***
## I(lstat^2)   0.1700436  0.0245037   6.940 1.58e-11 ***
## I(lstat^3)  -0.0023432  0.0004648  -5.041 7.00e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.409 on 403 degrees of freedom
## Multiple R-squared:  0.6767, Adjusted R-squared:  0.6743
## F-statistic: 281.2 on 3 and 403 DF,  p-value: < 2.2e-16
```

```
Boston.poly.4 <- lm(medv ~ poly(lstat, 4, raw = TRUE), data=train.data)
```

```
summary(Boston.poly.4)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, 4, raw = TRUE), data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.1715  -3.3178  -0.5482   2.4121  27.5123
##
## Coefficients:
```

```
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   6.176e+01  2.585e+00  23.892  < 2e-16 ***
## poly(lstat, 4, raw = TRUE)1 -8.283e+00  8.275e-01 -10.010  < 2e-16 ***
## poly(lstat, 4, raw = TRUE)2  6.091e-01  8.461e-02   7.199 3.01e-12 ***
## poly(lstat, 4, raw = TRUE)3 -2.044e-02  3.378e-03  -6.051 3.30e-09 ***
## poly(lstat, 4, raw = TRUE)4  2.472e-04  4.573e-05   5.405 1.11e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.229 on 402 degrees of freedom
## Multiple R-squared:  0.6986, Adjusted R-squared:  0.6956
## F-statistic: 232.9 on 4 and 402 DF,  p-value: < 2.2e-16
```

```
Boston.poly.5 <- lm(medv ~ poly(lstat, 5, raw = TRUE), data=train.data)
```

```
summary(Boston.poly.5)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, 5, raw = TRUE), data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.8402  -3.1241  -0.4857   2.2791  27.4777
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   7.446e+01  4.153e+00  17.932  < 2e-16 ***
## poly(lstat, 5, raw = TRUE)1 -1.423e+01  1.739e+00  -8.181 3.76e-15 ***
## poly(lstat, 5, raw = TRUE)2  1.527e+00  2.515e-01   6.071 2.96e-09 ***
## poly(lstat, 5, raw = TRUE)3 -8.118e-02  1.605e-02  -5.056 6.52e-07 ***
## poly(lstat, 5, raw = TRUE)4  2.023e-03  4.616e-04   4.384 1.49e-05 ***
## poly(lstat, 5, raw = TRUE)5 -1.883e-05  4.870e-06  -3.867 0.000129 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.14 on 401 degrees of freedom
## Multiple R-squared:  0.7094, Adjusted R-squared:  0.7058
## F-statistic: 195.8 on 5 and 401 DF,  p-value: < 2.2e-16
```

```
Boston.poly.6 <- lm(medv ~ poly(lstat, 6, raw = TRUE), data=train.data)
```

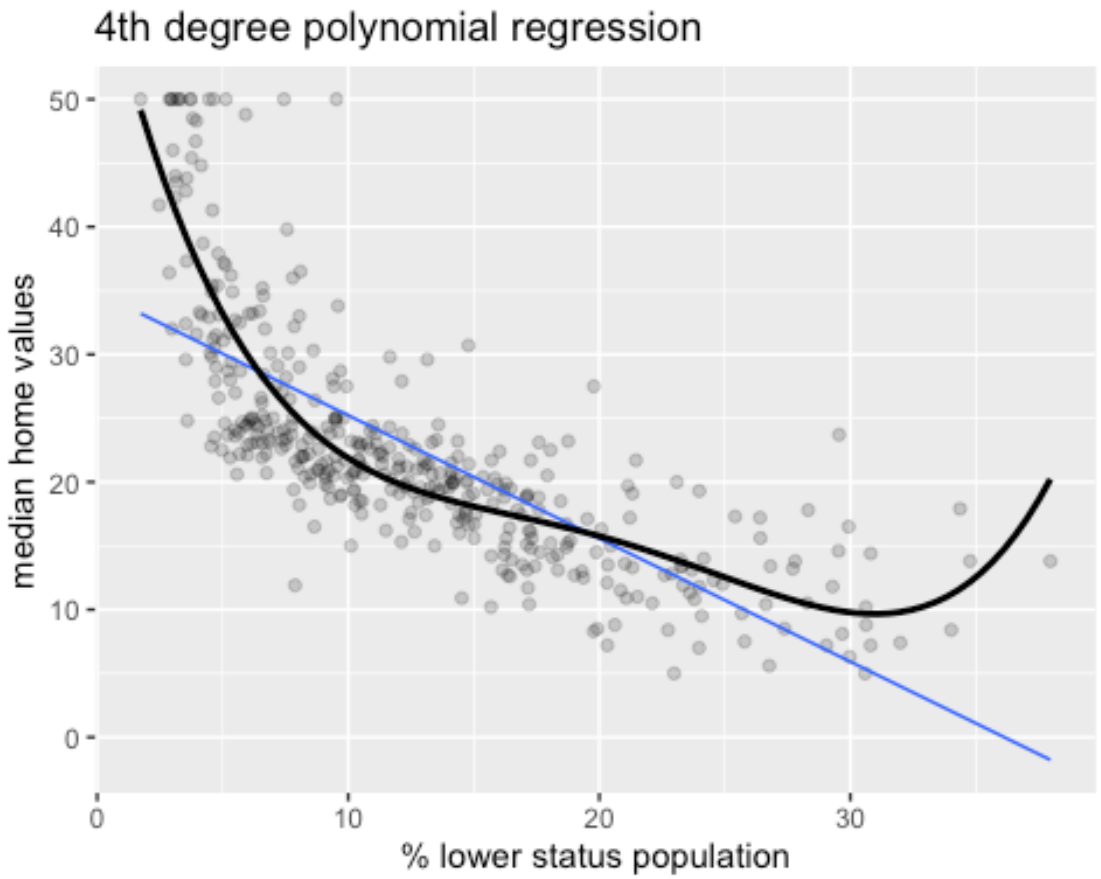
```
summary(Boston.poly.6)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, 6, raw = TRUE), data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.9000  -3.0530  -0.6403   2.1256  27.1289
```

```
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  8.412e+01  6.662e+00  12.626  < 2e-16 ***
## poly(lstat, 6, raw = TRUE)1 -1.981e+01  3.484e+00  -5.688 2.49e-08 ***
## poly(lstat, 6, raw = TRUE)2  2.662e+00  6.632e-01   4.015 7.11e-05 ***
## poly(lstat, 6, raw = TRUE)3 -1.877e-01  5.979e-02  -3.140  0.00182 **
## poly(lstat, 6, raw = TRUE)4  7.023e-03  2.742e-03   2.561  0.01080 *
## poly(lstat, 6, raw = TRUE)5 -1.324e-04  6.162e-05  -2.149  0.03222 *
## poly(lstat, 6, raw = TRUE)6  9.930e-07  5.369e-07   1.849  0.06514 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.125 on 400 degrees of freedom
## Multiple R-squared:  0.7119, Adjusted R-squared:  0.7076
## F-statistic: 164.7 on 6 and 400 DF,  p-value: < 2.2e-16
```

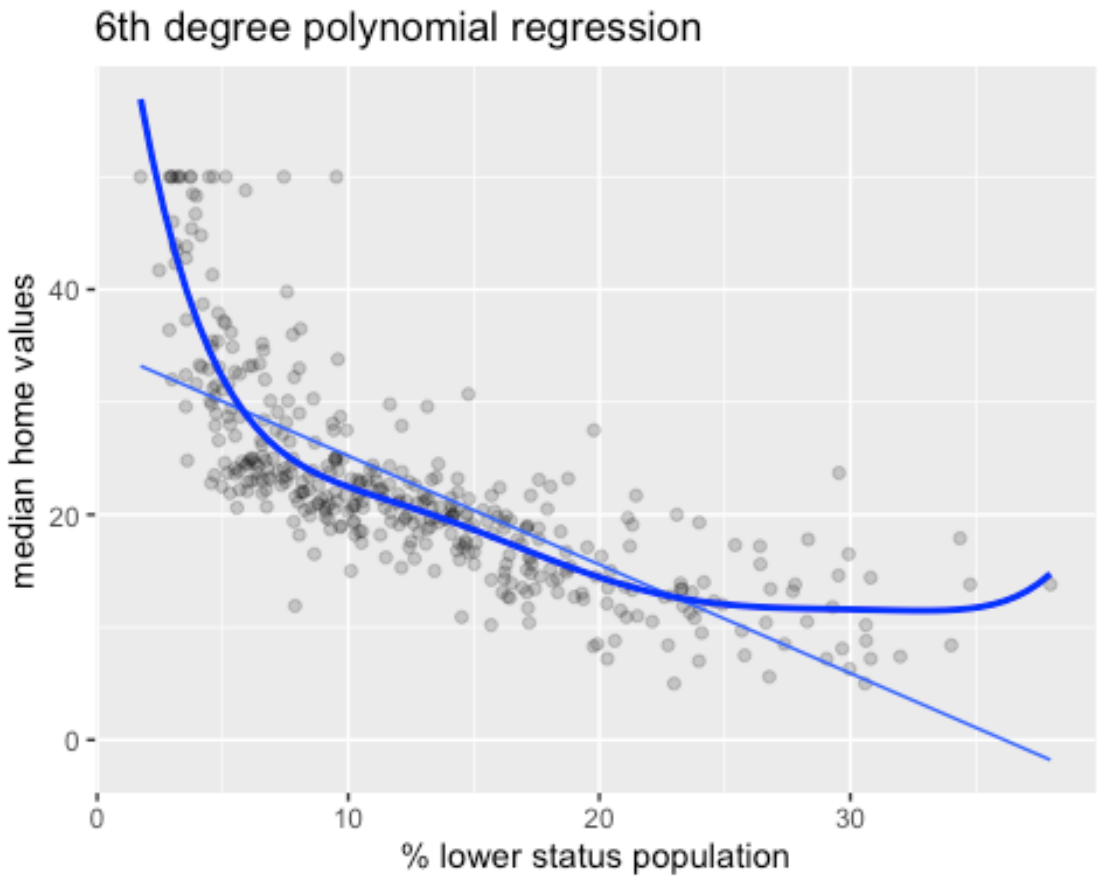
```
par(mfrow = c(1,3))
ggplot(train.data, aes(x = lstat, y = medv)) + geom_point(alpha = 0.2) +
geom_smooth(method = "lm", se = FALSE, size = 0.5) + geom_smooth(method =
"lm", formula = y ~ poly(x, 4), se = FALSE, color = "black") + labs(title =
"4th degree polynomial regression", x = "% lower status population", y =
"median home values")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```




```
ggplot(train.data, aes(x = lstat, y = medv)) +  geom_point(alpha = 0.2) +
geom_smooth(method = "lm", se = FALSE, size = 0.5) + geom_smooth(method =
"lm", formula = y ~ poly(x, 5), se = FALSE, color = "purple") + labs(title =
"5th degree polynomial regression", x = "% lower status population", y =
"median home values")

## `geom_smooth()` using formula = 'y ~ x'
```


```
ggplot(train.data, aes(x = lstat, y = medv)) +  geom_point(alpha = 0.2) +
geom_smooth(method = "lm", se = FALSE, size = 0.5) + geom_smooth(method =
"lm", formula = y ~ poly(x, 6), se = FALSE, color = "blue") + labs(title =
"6th degree polynomial regression", x = "% lower status population", y =
"median home values")

## `geom_smooth()` using formula = 'y ~ x'
```



Comparison of the R-squared

```
summary(Boston.reg)$r.squared #linear
```

```
## [1] 0.5485369
```

```
summary(Boston.poly.2.P)$r.squared #quadratic
```

```
## [1] 0.6563094
```

```
summary(Boston.poly.3)$r.squared #cubic
```

```
## [1] 0.6766978
```

```
summary(Boston.poly.4)$r.squared #quartic
```

```
## [1] 0.6986023
```

```
summary(Boston.poly.5)$r.squared #quintic
```

```
## [1] 0.7094364
```

```
summary(Boston.poly.6)$r.squared #sextic
```

```
## [1] 0.7118998
```

R-squared values suggest higher-order polynomial models explain more variance in the outcome variable (increasing R-squared), 71.19% for the 6th order polynomial compared to 54.85% for the linear model. However, visually, the differences are subtle.

Model Comparison

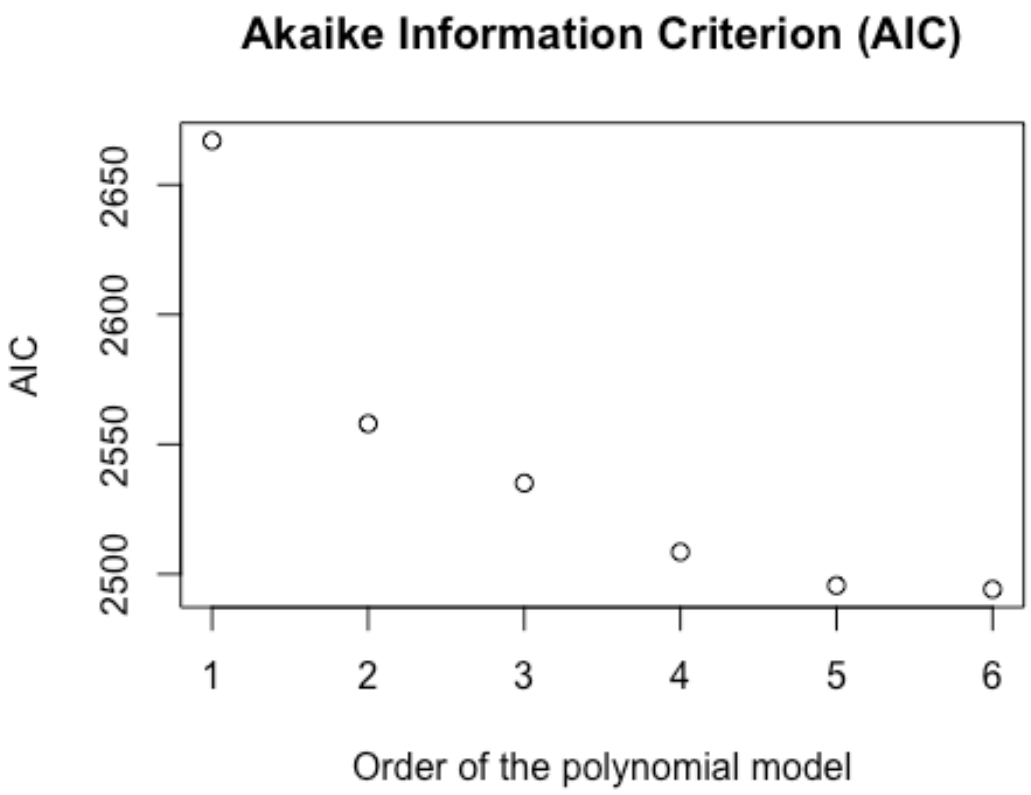
To move beyond visual inspection, we employed formal techniques:

- F-tests (anova): These tests helped compare models and identify diminishing improvements with higher polynomial degrees.
- AIC: This metric considers both model fit and complexity. A lower AIC value indicates a better balance between explaining the data and avoiding overfitting.

```
anova(Boston.reg, Boston.poly.2.P, Boston.poly.3, Boston.poly.4,
Boston.poly.5, Boston.poly.6) #F-test
```

```
## Analysis of Variance Table
##
## Model 1: medv ~ lstat
## Model 2: medv ~ lstat + poly(lstat, 2, raw = TRUE)
## Model 3: medv ~ lstat + I(lstat^2) + I(lstat^3)
## Model 4: medv ~ poly(lstat, 4, raw = TRUE)
## Model 5: medv ~ poly(lstat, 5, raw = TRUE)
## Model 6: medv ~ poly(lstat, 6, raw = TRUE)
##   Res.Df   RSS Df Sum of Sq        F    Pr(>F)
## 1    405 16464
## 2    404 12533  1    3930.2 149.6320 < 2.2e-16 ***
## 3    403 11790  1     743.5  28.3074 1.727e-07 ***
## 4    402 10991  1     798.8  30.4124 6.279e-08 ***
## 5    401 10596  1     395.1  15.0421 0.0001229 ***
## 6    400 10506  1      89.8   3.4202 0.0651399 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
aic <- c(AIC(Boston.reg), AIC(Boston.poly.2.P), AIC(Boston.poly.3),
AIC(Boston.poly.4), AIC(Boston.poly.5), AIC(Boston.poly.6))
df <- data.frame(aic) #AIC

plot(aic, data = aic, main = "Akaike Information Criterion (AIC)", xlab =
"Order of the polynomial model", ylab = "AIC")
```



The hypothesis that is checked at each step is that the decrease in RSS is not significant. The hypothesis is rejected if the p-value is smaller than a given significance level (say 0.05).

The F-test and AIC comparison suggests that the higher order models are increasingly better fit, compared to the linear or lower order models, up to and including the 5th order polynomial.

Over-Fitting / Generalization Limitations

#Test model on test data While the F-tests and AIC suggest a 5th order model might fit best, relying solely on these metrics can be misleading. To validate the model's generalizability, we used the trained model on our test data. This ensures the chosen model performs well on unseen data, not just the training data used to fit it.

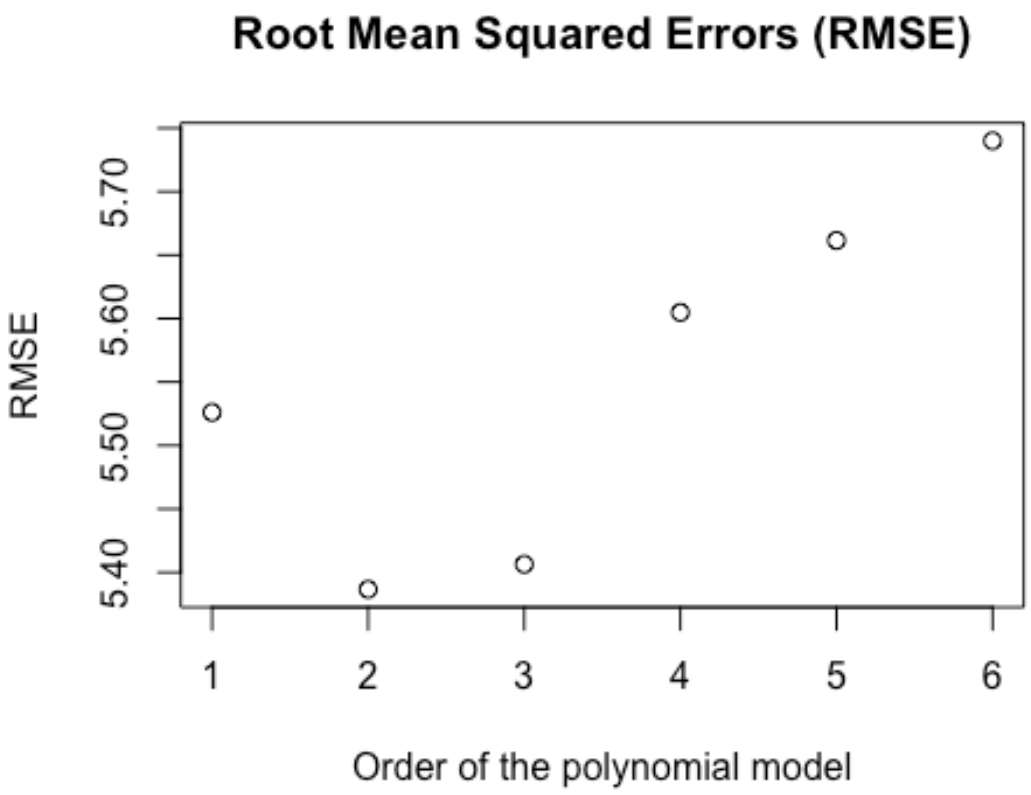
We compared the Root Mean Squared Errors (RMSE) and the adjusted R-squared (R2) values of the trained model on the test data.

```
print('--- Linear ---')
```

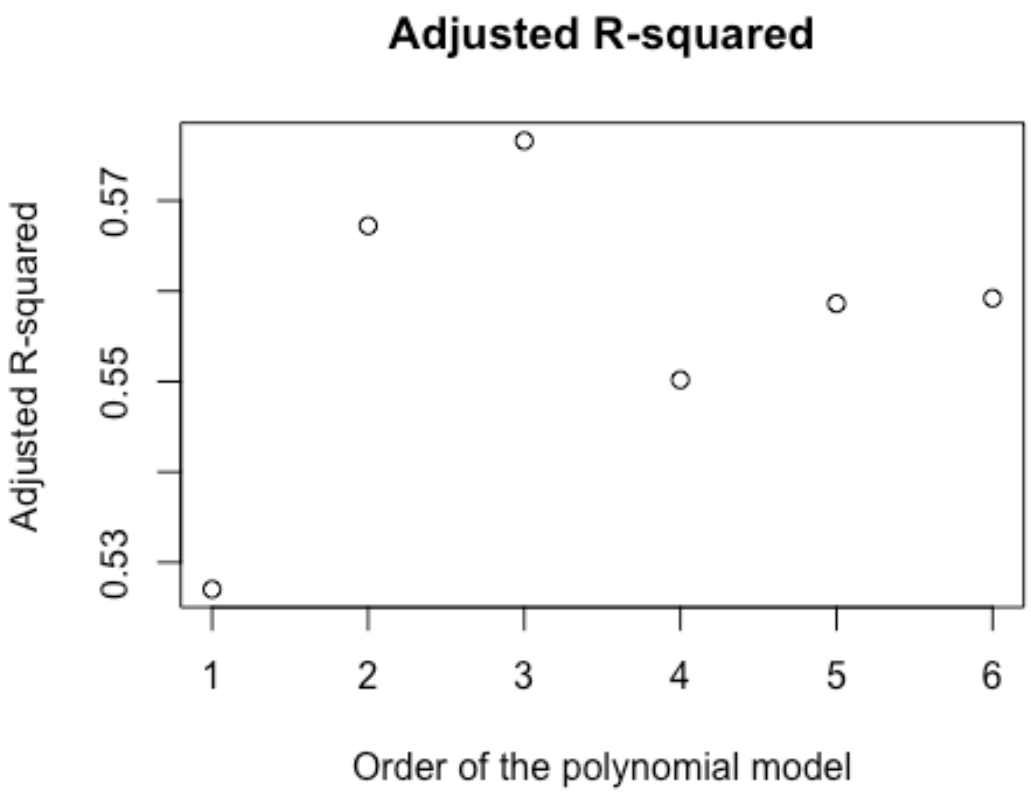
```
## [1] "--- Linear ---"
```

```r
predictions.l <- Boston.reg %>% predict(test.data)
RMSE(predictions.l, test.data$medv)
## [1] 5.526096
R2(predictions.l, test.data$medv)
## [1] 0.5270256
print('--- 2nd order ---')
## [1] "--- 2nd order ---"
predictions.2 <- Boston.poly.2.P %>% predict(test.data)
RMSE(predictions.2, test.data$medv)
## [1] 5.386672
R2(predictions.2, test.data$medv)
## [1] 0.5672324
print('--- 3rd order ---')
## [1] "--- 3rd order ---"
predictions.3 <- Boston.poly.3 %>% predict(test.data)
RMSE(predictions.3, test.data$medv)
## [1] 5.406411
R2(predictions.3, test.data$medv)
## [1] 0.5766242
print('--- 4th order ---')
## [1] "--- 4th order ---"
predictions.4 <- Boston.poly.4 %>% predict(test.data)
RMSE(predictions.4, test.data$medv)
## [1] 5.604825
R2(predictions.4, test.data$medv)
## [1] 0.5501867
print('--- 5th order ---')
## [1] "--- 5th order ---"
```

```r
predictions.5 <- Boston.poly.5 %>% predict(test.data)

RMSE(predictions.5, test.data$medv)

## [1] 5.661506

R2(predictions.5, test.data$medv)

## [1] 0.5586259

print('--- 6th order ---')

## [1] "--- 6th order ---"

predictions.6 <- Boston.poly.6 %>% predict(test.data)

RMSE(predictions.6, test.data$medv)

## [1] 5.74002

R2(predictions.6, test.data$medv)

## [1] 0.5592157

rmse <- c(RMSE(predictions.l, test.data$medv), RMSE(predictions.2,
test.data$medv), RMSE(predictions.3, test.data$medv), RMSE(predictions.4,
test.data$medv), RMSE(predictions.5, test.data$medv), RMSE(predictions.6,
test.data$medv))
r2 <- c(R2(predictions.l, test.data$medv), R2(predictions.2, test.data$medv),
R2(predictions.3, test.data$medv), R2(predictions.4, test.data$medv),
R2(predictions.5, test.data$medv), R2(predictions.6, test.data$medv))
df1 <- data.frame(rmse)
df2 <- data.frame(r2)
plot(rmse, data = df1, main = "Root Mean Squared Errors (RMSE)", xlab =
"Order of the polynomial model", ylab = "RMSE")
```




```r
plot(r2, data = df2, main = "Adjusted R-squared", xlab = "Order of the
polynomial model", ylab = "Adjusted R-squared")
```



Our evaluation confirms that polynomial models are an improvement over the linear model, indicated by higher adjusted R-squared and lower RMSE values. However, this improvement plateaus and even reverses at some point. This is a telltale sign of overfitting, where the model memorizes noise in the training data and loses its ability to generalize to unseen data. Thus, the simpler models perform better on new data.

#Conclusion In conclusion, fitting higher order polynomial terms to the data suggested that the 5th order polynomial model might explain the data the best. After testing the model on the test data, we conclude that a polynomial model is an improvement over a linear model, but since higher order polynomial models exhibit signs of overfitting, we select the quadratic model.

#Reading list 1:
http://ndl.ethernet.edu.et/bitstream/123456789/26799/1/122.Henning%20Best%2C%20Christof%20Wolf-.pdf

2: https://online.stat.psu.edu/stat501/lesson/9/9.8

3: https://www.statology.org/polynomial-regression/

4: https://home.iitk.ac.in/~shalab/regression/Chapter12-Regression-PolynomialRegression.pdf

5: https://towardsdatascience.com/polynomial-regression-bbe8b9d97491

6: https://statisticsbyjim.com/regression/choose-linear-nonlinear-regression/

7: https://www.analyticsvidhya.com/blog/2021/07/all-you-need-to-know-about-polynomial-regression/

8: (R reference): https://bookdown.org/ssjackson300/Machine-Learning-Lecture-Notes/polynomial-regression.html

9: (R reference): https://www.statology.org/polynomial-regression-r/

10: (R reference): https://medium.com/analytics-vidhya/machine-learning-project-3-predict-salary-using-polynomial-regression-7024c7bace4f