Lecture 6

Residuals and outliers

The **residuals** of a regression model are the difference between the predicted values and the observed values. There are a number of ways to assess residuals to optimize our regression models. Some examples of regression metrics are listed below. This is not an exhaustive list. Recall the similarity of these metrics to variation metrics from lecture one.

Regression Metrics:

- *Mean Absolute Error* (MAE): The average of the absolute differences between predicted and actual values.
- *Mean Squared Error* (MSE): The average of the squared differences between predicted and actual values.
- *Root Mean Squared Error* (RMSE): The square root of the MSE, providing an interpretable measure in the same units as the target variable.
- *R-squared* (R^2): The proportion of the variance in the dependent variable that is predictable from the independent variables.

These are just a few examples of commonly used metrics for evaluating machine learning models. The choice of metric depends on the specific problem, the type of model being evaluated, and the goals of the analysis. It's essential to select metrics that are relevant to the problem domain and provide insights into the model's performance. Additionally, it's often useful to consider multiple metrics to get a comprehensive understanding of a model's strengths and weaknesses.

Here's how you can calculate various regression model metrics in R from predicted values and actual values. These are general cases. We'll look at specific examples later.
Bias: Bias is the difference between the predicted values and the actual values.

```
# Assuming 'predicted_values' contains the predicted values and 'actual_values' contains the actual values
bias <- mean(predicted_values - actual_values)
print(paste("Bias:", bias))
```

R-squared (R^2): R-squared measures the proportion of the variance in the dependent variable that is predictable from the independent variables.

```
# Assuming 'predicted_values' contains the predicted values and 'actual_values' contains the actual values
rsquared <- cor(predicted_values, actual_values)^2
print(paste("R-squared:", rsquared))
```

Adjusted R-squared: Adjusted R-squared adjusts the R-squared value based on the number of predictors in the model.

```
# Assuming 'predicted_values' contains the predicted values, 'actual_values' contains the actual values, and 'n' is the number of observations
n <- length(actual_values)
p <- ncol(predictors) # number of predictors
adjusted_rsquared <- 1 - (1 - rsquared) * ((n - 1) / (n - p - 1))
print(paste("Adjusted R-squared:", adjusted_rsquared))
```

Mean Square Error (MSE): MSE is the average of the squared differences between predicted and actual values.

```
# Assuming 'predicted_values' contains the predicted values and 'actual_values' contains the
actual values
mse <- mean((predicted_values - actual_values)^2)
print(paste("Mean Square Error (MSE):", mse))
```

Root Mean Square Error (RMSE): RMSE is the square root of MSE, providing an interpretable measure in the same units as the target variable.

```
# Assuming 'predicted_values' contains the predicted values and 'actual_values' contains the
actual values
rmse <- sqrt(mse)
print(paste("Root Mean Square Error (RMSE):", rmse))
```

Residual Standard Error (RSE): RSE is the standard deviation of the residuals (differences between predicted and actual values).

```
# Assuming 'predicted_values' contains the predicted values and 'actual_values' contains the
actual values, the -2 is for simple regression since there are 2 parameters, change  the value to -k
for k parameters in a multiple regression model
rse <- sqrt(sum((predicted_values - actual_values)^2) / (length(actual_values) - 2))
print(paste("Residual Standard Error (RSE):", rse))
```

Mean Absolute Error (MAE): MAE is the average of the absolute differences between predicted and actual values.

```
# Assuming 'predicted_values' contains the predicted values and 'actual_values' contains the
actual values
mae <- mean(abs(predicted_values - actual_values))
print(paste("Mean Absolute Error (MAE):", mae))
```

Mean Absolute Percentage Error (MAPE): MAPE measures the average relative error as a percentage of the actual values.

```
# Assuming 'predicted_values' contains the predicted values and 'actual_values' contains the
actual values
mape <- mean(abs((actual_values - predicted_values) / actual_values)) * 100
print(paste("Mean Absolute Percentage Error (MAPE):", mape))
```

Symmetric Mean Absolute Percentage Error (SMAPE): SMAPE is a symmetric version of MAPE.

```
# Assuming 'predicted_values' contains the predicted values and 'actual_values' contains the
actual values
smape <- mean(2 * abs(actual_values - predicted_values) / (abs(actual_values) +
abs(predicted_values))) * 100
print(paste("Symmetric Mean Absolute Percentage Error (SMAPE):", smape))
```

Mean Squared Logarithmic Error (MSLE): MSLE is the average of the squared differences between the natural logarithms of predicted and actual values.

```
# Assuming 'predicted_values' contains the predicted values and 'actual_values' contains the
actual values
```

```
msle <- mean((log1p(predicted_values) - log1p(actual_values))^2)
print(paste("Mean Squared Logarithmic Error (MSLE):", msle))
```

Root Mean Squared Logarithmic Error (RMSLE): RMSLE is the square root of MSLE.

```
# Assuming 'predicted_values' contains the predicted values and 'actual_values' contains the
actual values
rmsle <- sqrt(msle)
print(paste("Root Mean Squared Logarithmic Error (RMSLE):", rmsle))
```

Akaike Information Criterion (AIC): AIC measures the quality of a model relative to other models.

```
# Assuming 'predicted_values' contains the predicted values, 'actual_values' contains the actual
values, and 'k' is the number of parameters in the model
k <- 1 # Example: number of parameters
aic <- length(actual_values) * log(mse) + 2 * k
print(paste("Akaike Information Criterion (AIC):", aic))
```

Bayesian Information Criterion (BIC): BIC is similar to AIC but penalizes models with more parameters more heavily.

```
# Assuming 'predicted_values' contains the predicted values, 'actual_values' contains the actual
values, and 'k' is the number of parameters in the model
bic <- length(actual_values) * log(mse) + k * log(length(actual_values))
print(paste("Bayesian Information Criterion (BIC):", bic))
```

Mallow's Cp is a metric used to compare the performance of regression models, particularly in the context of model selection. It is used to assess the goodness-of-fit of a model while penalizing for the number of predictors included in the model. Lower values of Cp indicate better model fit.
To calculate Mallow's Cp for a LOESS (locally estimated scatterplot smoothing) regression model by hand in R, you typically need to follow these steps:

- Fit the LOESS regression model.
- Obtain the predicted values from the LOESS model.
- Calculate the residual sum of squares (RSS) of the model.
- Calculate the degrees of freedom of the model.
- Calculate Mallow's Cp using the formula:

$$C_p = \left(\frac{1}{n}\right) * \left(\frac{RSS}{\sigma^2}\right) + (2p - n)$$

where:
n is the number of observations.
p is the number of predictors in the model.
σ^2 is the estimated error variance.

Here's how you can implement this in R:

```
# Fit the LOESS regression model
loess_model <- loess(y ~ x, data = your_data)
# Obtain predicted values
predicted_values <- predict(loess_model)
# Calculate residual sum of squares (RSS)
RSS <- sum((your_data$y - predicted_values)^2)
# Calculate degrees of freedom
```

```
df <- length(your_data$y) - length(loess_model$coefficients)
# Calculate error variance
sigma_sq <- RSS / df
# Calculate Mallow's Cp
Cp <- (1/length(your_data$y)) * (RSS / sigma_sq) + (2 * length(loess_model$coefficients) -
length(your_data$y))
print(paste("Mallow's Cp:", Cp))
```

In this code:

- your_data represents your dataset with variables x and y.
- loess() is used to fit the LOESS regression model.
- predict() is used to obtain the predicted values.
- Residual sum of squares (RSS) is calculated.
- Degrees of freedom (df) are calculated as the difference between the number of observations and the number of coefficients in the model.
- Error variance (σ^2) is estimated as RSS divided by df.

Mallow's Cp is calculated using the provided formula. Make sure to replace your_data, x, and y with your actual data and variable names. Additionally, ensure that the LOESS model is appropriate for your data and research question.

These metrics have different strengths and weaknesses, but not all regression models are set up in the same way, so it may not be possible to rely on built-in functions for these metrics in order to compare regression models built with different function types. We may also want to optimize a model using metrics other than standard residual square errors.

Let's look at some specific examples from the mtcars dataset using a variety of regression models to calculate their residuals, and then calculate a sample of metrics. The example metrics focus on the simple linear model, but the code can be updated to look at other model residuals as well.

```
# Load the mtcars dataset
data(mtcars)

# Simple Linear Regression: mpg ~ hp
model_lm <- lm(mpg ~ hp, data = mtcars)

# Multiple Linear Regression: mpg ~ hp + wt + qsec
model_lm_multi <- lm(mpg ~ hp + wt + qsec, data = mtcars)

# Generalized Linear Model: mpg ~ hp + wt + qsec (using Gaussian family)
model_glm <- glm(mpg ~ hp + wt + qsec, family = gaussian, data = mtcars)

# LOESS (Locally Estimated Scatterplot Smoothing): mpg ~ hp
model_loess <- loess(mpg ~ hp, data = mtcars)

# Penalized Regression: Ridge Regression (L2) (using glmnet package)
# install.packages("glmnet")
```

```
library(glmnet)
x <- as.matrix(mtcars[, c("hp", "wt", "qsec")])
y <- mtcars$mpg
model_ridge <- glmnet(x, y, alpha = 0, lambda = 0.1) # Alpha=0 for Ridge, lambda controls
regularization

# Spline Regression: mpg ~ hp (using splines)
library(splines)
model_spline <- lm(mpg ~ ns(hp, df = 4), data = mtcars)

# Gaussian Process Regression (using kernlab package)
# install.packages("kernlab")
library(kernlab)
model_gp <- gausspr(mpg ~ hp + wt + qsec, data = mtcars, kernel = "rbfdot")

# Get predicted values from the model
predicted_lm <- predict(model_lm)
predicted_multi <- predict(model_lm_multi)

# Calculate residuals by hand: Residual = Observed - Predicted
residuals_lm <- mtcars$mpg - predicted_lm
residuals_multi <- mtcars$mpg - predicted_multi

# Calculate Residual Sum of Squares (RSS)
RSS_lm <- sum(residuals_lm^2)

# Calculate Total Sum of Squares (TSS)
TSS <- sum((mtcars$mpg - mean(mtcars$mpg))^2)

# Calculate R-squared
R_squared_lm <- 1 - RSS_lm / TSS
R_squared_lm

# Number of observations
n <- length(mtcars$mpg)

# Calculate RMSE
RMSE_lm <- sqrt(RSS_lm / n)
RMSE_lm

# Calculate MAPE
MAPE_lm <- mean(abs((mtcars$mpg - predicted_lm) / mtcars$mpg)) * 100
MAPE_lm

# Number of predictors in the model
p_lm <- length(coefficients(model_lm))

# Calculate AIC
```

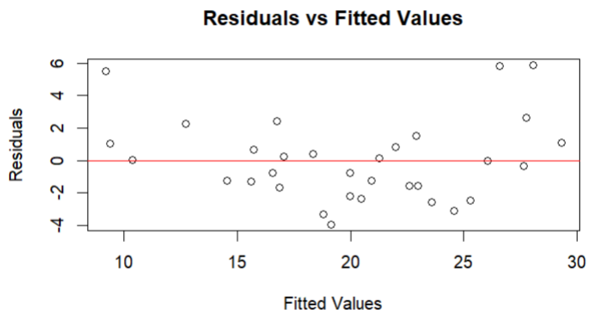
```
AIC_lm <- n * log(RSS_lm / n) + 2 * p_lm
AIC_lm

# Calculate BIC
BIC_lm <- n * log(RSS_lm / n) + p_lm * log(n)
BIC_lm

# Calculate error variance (sigma^2) from the full model (multiple regression)
sigma2_full <- sum(residuals_multi^2) / (n - p_lm)

# Calculate Mallow's Cp
Cp <- RSS_lm / sigma2_full + 2 * p_lm - n
Cp

# R-squared
summary(model_lm)$r.squared

# RMSE
sqrt(mean(residuals_lm^2))

# MAPE (not built-in, but you can verify)
mean(abs((mtcars$mpg - predicted_lm) / mtcars$mpg)) * 100

# AIC
AIC(model_lm)

# BIC
BIC(model_lm)
```

As noted, some of these metrics have built-in functions that are designed to work with models that produce output structured similarly to the lm() function output. However, if the model output is not structured that way, the built-in function will not work. And some metrics don't have built-in functions for them at all. Because we can build the residuals from the predict() function also means that if we have an equation for the model (for parametric models), we can also calculate residuals and all these metrics from the equation.

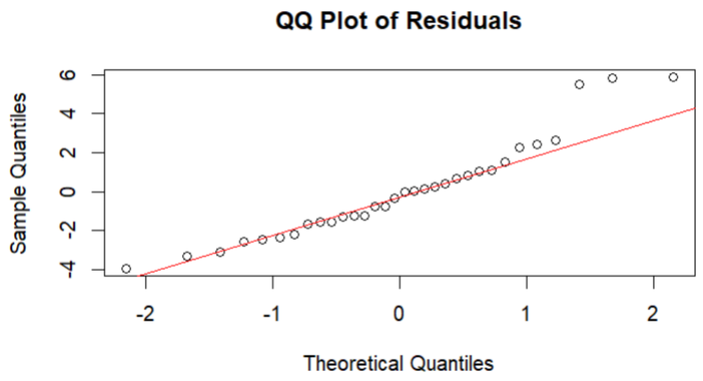
Outlier Identification and Removal
Identifying outliers in residuals is an essential part of regression diagnostics. Outliers can have a significant impact on the model's fit and the interpretation of results. Below are examples of how you can identify outliers in residuals using both graphical and statistical methods, as well as hypothesis tests, with R code examples.

```
# Load the dataset
data(mtcars)

# Fit a simple linear regression model
model <- lm(mpg ~ wt + hp, data = mtcars)
```

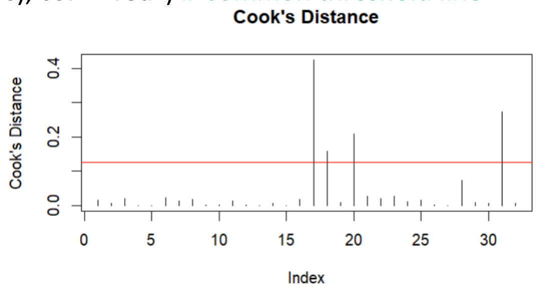
```
# Calculate residuals
residuals <- resid(model)

# Plot Residuals vs Fitted Values
plot(fitted(model), residuals,
    main = "Residuals vs Fitted Values",
    xlab = "Fitted Values",
    ylab = "Residuals")
abline(h = 0, col = "red")
```



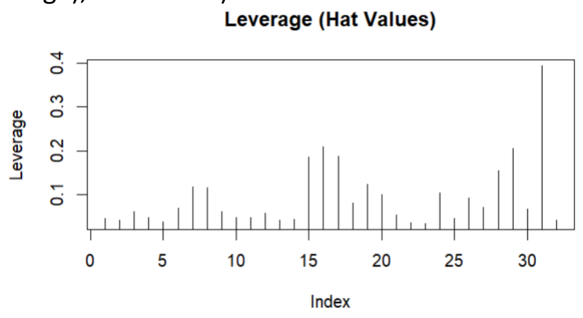
```
# QQ Plot of Residuals
qqnorm(residuals, main = "QQ Plot of Residuals")
qqline(residuals, col = "red")
```



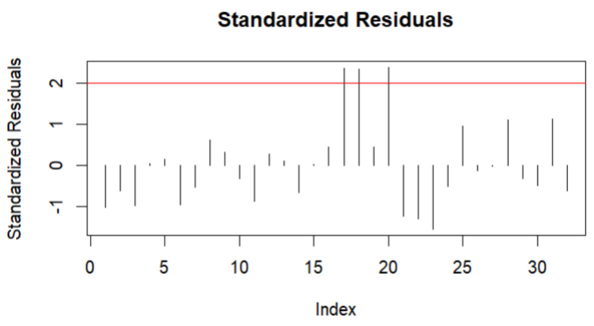
```
# Calculate Cook's Distance
cooks_distance <- cooks.distance(model)

# Plot Cook's Distance
plot(cooks_distance, type = "h", main = "Cook's Distance",
    ylab = "Cook's Distance")
abline(h = 4 / nrow(mtcars), col = "red") # Common threshold line
```



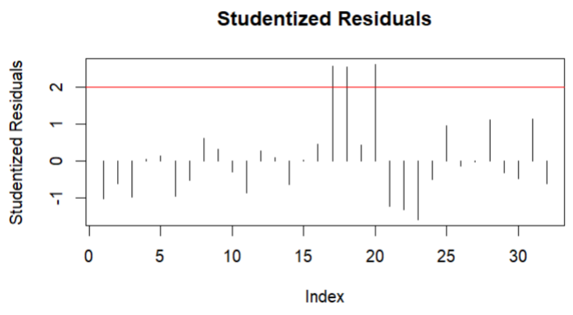
```
# Calculate Leverage (Hat Values)
leverage <- hatvalues(model)

# Plot Leverage
plot(leverage, type = "h", main = "Leverage (Hat Values)",
    ylab = "Leverage")
abline(h = 2 * mean(leverage), col = "red") # Common threshold
```



```
# Calculate Standardized Residuals
standardized_residuals <- rstandard(model)

# Calculate Studentized Residuals
studentized_residuals <- rstudent(model)

# Plot Standardized Residuals
plot(standardized_residuals, type = "h", main = "Standardized Residuals",
    ylab = "Standardized Residuals")
abline(h = c(-2, 2), col = "red") # Common threshold lines
```



```
# Plot Studentized Residuals
plot(studentized_residuals, type = "h", main = "Studentized Residuals",
    ylab = "Studentized Residuals")
abline(h = c(-2, 2), col = "red") # Common threshold lines
```



```
# Load the car package
library(car)

# Perform the Bonferroni Outlier Test
outlier_test <- outlierTest(model)

# Display results
print(outlier_test)

# Perform Shapiro-Wilk Test for Normality
shapiro_test <- shapiro.test(residuals)

# Display results
print(shapiro_test)
```

While identifying outliers is critical, handling them requires careful consideration. Simply removing outliers without understanding the cause can lead to biased results. Instead, outliers should be investigated to understand why they exist and whether they should be addressed or if they offer valuable information about the data.

Another test you can use is the Rosner Test.

```
# Install the EnvStats package if you haven't already
install.packages("EnvStats")

# Load the package
library(EnvStats)

# Example data
data <- c(10.1, 10.5, 10.2, 10.6, 10.8, 11.2, 10.4, 10.7, 11.4, 15.2, 16.8, 17.3)

# Perform Rosner's test
# The `k` parameter specifies the maximum number of potential outliers to test for.
rosner_test_result <- rosnerTest(data, k = 3)

# Display the results
print(rosner_test_result)
```

Sample Output:

```
Rosner Test for Outliers

Data: data

Number of Observations: 12
Maximum Number of Outliers (k): 3
Number of Outliers Detected: 2

  Obs.  Value   R      p-value  Outlier?
```

11	16.8	13.44	< 0.05	Yes
12	17.3	13.99	< 0.05	Yes
10	15.2	9.87	0.07	No

```
# Perform Rosner's test on the 'mpg' column of the mtcars dataset
rosner_test_mtcars <- rosnerTest(mtcars$mpg, k = 3)

# Display the results
print(rosner_test_mtcars)
```

```
Results of Outlier Test
-------------------------

Test Method:                     Rosner's Test for Outliers

Hypothesized Distribution:       Normal

Data:                            mtcars$mpg

Sample Size:                     32

Test Statistics:                 R.1 = 2.291272
                                 R.2 = 2.291827
                                 R.3 = 2.182345

Test Statistic Parameter:        k = 3

Alternative Hypothesis:          Up to 3 observations are not
                                 from the same Distribution.

Type I Error:                    5%

Number of Outliers Detected:     0

  i   Mean.i     SD.i Value Obs.Num    R.i+1 lambda.i+1 Outlier
1 0 20.09062 6.026948  33.9      20 2.291272   2.938048   FALSE
2 1 19.64516 5.565359  32.4      18 2.291827   2.923571   FALSE
3 2 19.22000 5.122930  30.4      19 2.182345   2.908473   FALSE
```

The Rosner test is a robust method to identify multiple outliers in a dataset, especially when you suspect there are more than one. This method is particularly valuable in quality control and environmental data analysis, where outliers might indicate significant deviations or anomalies. It also conveniently identifies the observation number in the output so that the values can be removed. For example, since observation 20 is the most extreme value, I could remove it like this:

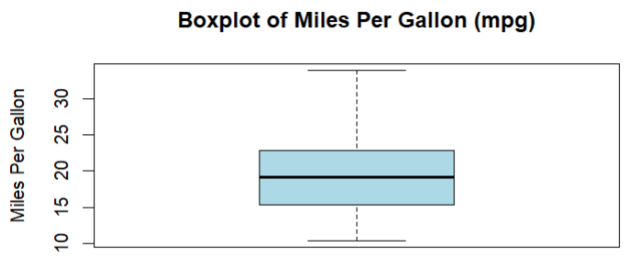
```
# Load the mtcars dataset
data(mtcars)

# Remove row 20
mtcars <- mtcars[-20, ]

# Display the modified dataset to verify
print(mtcars)
```

If you have several values to remove, you can remove them with a filter if they meet a particular condition.

```
# Example: Remove rows where mpg is less than 15
mtcars <- mtcars[mtcars$mpg >= 15, ]

# Display the modified dataset
print(mtcars)
```

Boxplots can also be used to identify specific observations as outliers.

```
# Load the mtcars dataset
data(mtcars)

# Create a boxplot for the mpg variable
boxplot(mtcars$mpg,
      main = "Boxplot of Miles Per Gallon (mpg)",
      ylab = "Miles Per Gallon",
      col = "lightblue")
```



```
# Create a boxplot with labels for outliers
boxplot(mtcars$mpg,
      main = "Boxplot of Miles Per Gallon (mpg)",
      ylab = "Miles Per Gallon",
      col = "lightblue",
      outline = TRUE)

# Identify and label the outliers
outliers <- boxplot.stats(mtcars$mpg)$out
outlier_indices <- which(mtcars$mpg %in% outliers)

# Add text labels for outliers
text(x = rep(1, length(outliers)), y = outliers,
    labels = rownames(mtcars)[outlier_indices],
    pos = 4, cex = 0.8, col = "red")

# Extract outliers using boxplot.stats
outliers <- boxplot.stats(mtcars$mpg)$out
print(outliers)
```

If you run these examples, mpg has no outliers, but if you update the code to test the model residuals from the prior example, you can print which vehicles are producing the outliers in that model as shown below.

```
Chrysler Imperial          Fiat 128    Toyota Corolla
         5.507513          5.800972          5.853791
```

Keep in mind that outliers are to be expected when you have a lot of data. Small datasets are the most sensitive to outliers, but larger datasets, just by chance, will have some. If you do choose to remove the outliers, rerun all models and statistics and be prepared to justify the removal. For largish datasets, the default should be to identify them but not necessarily remove them unless they have a significant impact on the model.

Resources:

1. https://stattrek.com/regression/residual-analysis
2. https://online.stat.psu.edu/stat462/node/172/
3. https://cran.r-project.org/web/packages/metrica/vignettes/available_metrics_regression.html