Linear Regression

Data Set

h2o

```r
ames <- AmesHousing::make_ames()
ames.h2o <- as.h2o(ames)
```

stratified (*Sale_Price*) training sample

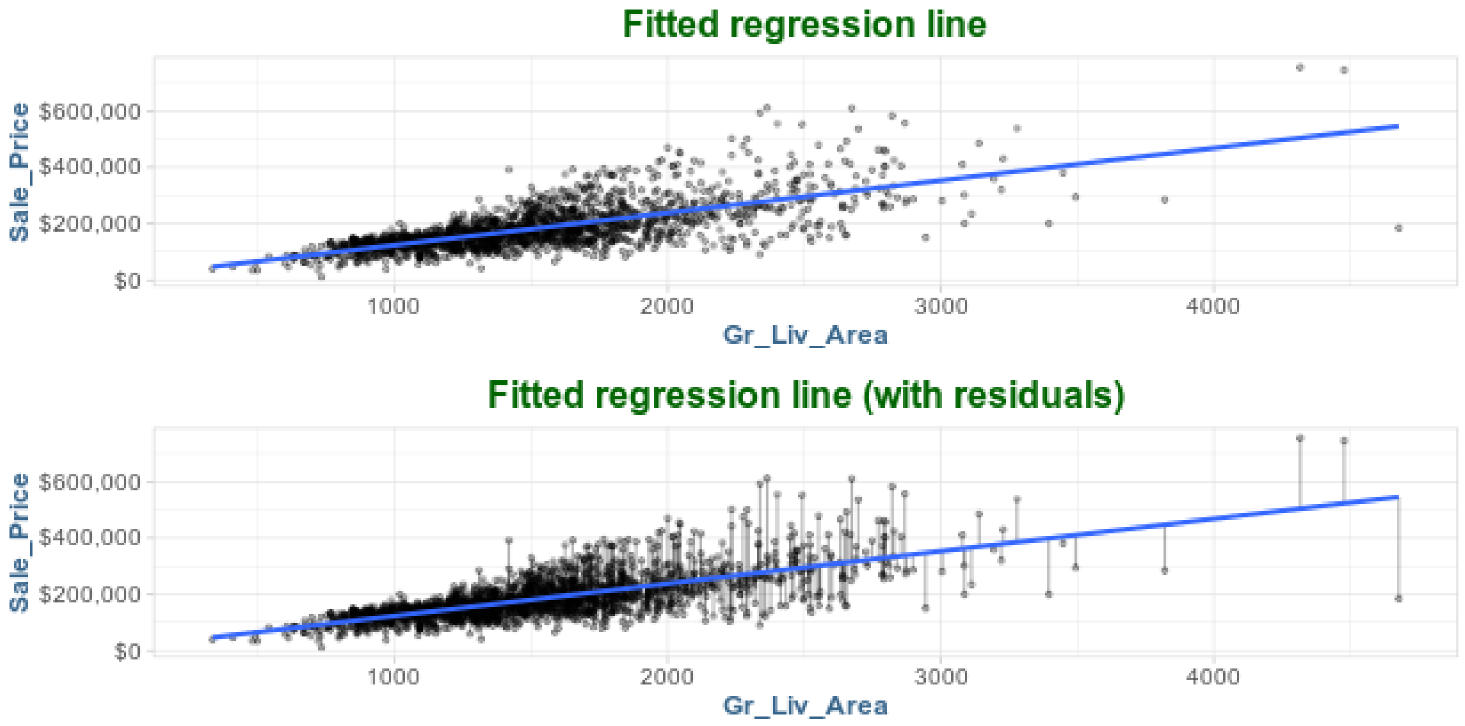
```r
set.seed(123)

split <- initial_split(ames, prop = 0.7,
                       strata = "Sale_Price")

ames_train <- training(split)
ames_test <- testing(split)
```

Simple Linear Model

```r
model1 <- lm(Sale_Price ~ Gr_Liv_Area, data = ames_train)
```

```r
# Fitted regression line (full training)
p1 <- model1 %>%
   broom::augment() %>%
   ggplot(aes(Gr_Liv_Area, Sale_Price)) +
   geom_point(size = 1, alpha = 0.3) +
   geom_smooth(se = F, method = "lm") +
   scale_y_continuous(labels = scales::dollar) +
   ggtitle("Fitted regression line")

# Fitted regression line (restricted range)
p2 <- model1 %>%
   broom::augment() %>%
   ggplot(aes(Gr_Liv_Area, Sale_Price)) +
   geom_segment(aes(x = Gr_Liv_Area, y = Sale_Price,
                    xend = Gr_Liv_Area, yend = .fitted),
                alpha = .3) +
   geom_point(size = 1, alpha = 0.3) +
   geom_smooth(se = F, method = "lm") +
   scale_y_continuous(labels = scales::dollar) +
   ggtitle("Fitted regression line (with residuals)")
```

```r
grid.arrange(p1, p2, nrow = 2)
```



```r
summary(model1)
```

```

Call:
lm(formula = Sale_Price ~ Gr_Liv_Area, data = ames_train)

Residuals:
    Min      1Q  Median      3Q     Max 
-361143  -30668   -2449   22838  331357 

Coefficients:
            Estimate Std. Error t value            Pr(>|t|)    
(Intercept) 8732.938   3996.613   2.185               0.029 *  
Gr_Liv_Area  114.876      2.531  45.385 <0.0000000000000002 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 56700 on 2051 degrees of freedom
Multiple R-squared:  0.5011,    Adjusted R-squared:  0.5008 
F-statistic:  2060 on 1 and 2051 DF,  p-value: < 0.00000000000000022
```

```r
sigma(model1) # RMSE, also Residual Standard Error in summary()
```

```
[1] 56704.78
```

```r
sigma(model1)^2 # MSE
```

```
[1] 3215432370
```

Inference

The variability of an estimate is its *standard error (SE)*, the square root of its variance.

t-test for the coefficents are simply the estimated coefficent divided by the standard error (t value = Estimate / Std. Error)

t-test measure the number of standard deviations each coefficent is away from zero (basically abs(T) > 2 is significant at 95% conf)

The confidence interval for coefficents is:

$$\hat{\beta}_j \pm t_{1-\alpha/2,n-p} \hat{SE}(\hat{\beta}_j)$$

```r
confint(model1, level = .95)
```

```
                2.5 %     97.5 %
(Intercept) 895.0961 16570.7805
Gr_Liv_Area 109.9121   119.8399
```

Interpretation: We are 95% confident that each one unit increase in Gr_Liv_Area adds between 109.9 and 119.8 dollars to the sale price.

Linear Regression Assumptions:

- 1.) Independent observations
- 2.) The random errors have mean zero, and constant variance
- 3.) The random errors are normally distributed

Multiple Linear Regression

```r
(model2 <- lm(Sale_Price ~ Gr_Liv_Area + Year_Built, data = ames_train))
```

```

Call:
lm(formula = Sale_Price ~ Gr_Liv_Area + Year_Built, data = ames_train)

Coefficients:
```

```
(Intercept)  Gr_Liv_Area   Year_Built 
-2123054.21        99.18      1093.48 
```

```
# Equivalent
(model2 <- update(model1, . ~ . + Year_Built))
```

```

Call:
lm(formula = Sale_Price ~ Gr_Liv_Area + Year_Built, data = ames_train)

Coefficients:
(Intercept)  Gr_Liv_Area   Year_Built  
-2123054.21        99.18      1093.48  
```

```
round(coef(model2), 3)
```

```
 (Intercept)  Gr_Liv_Area   Year_Built 
-2123054.207       99.176     1093.485 
```

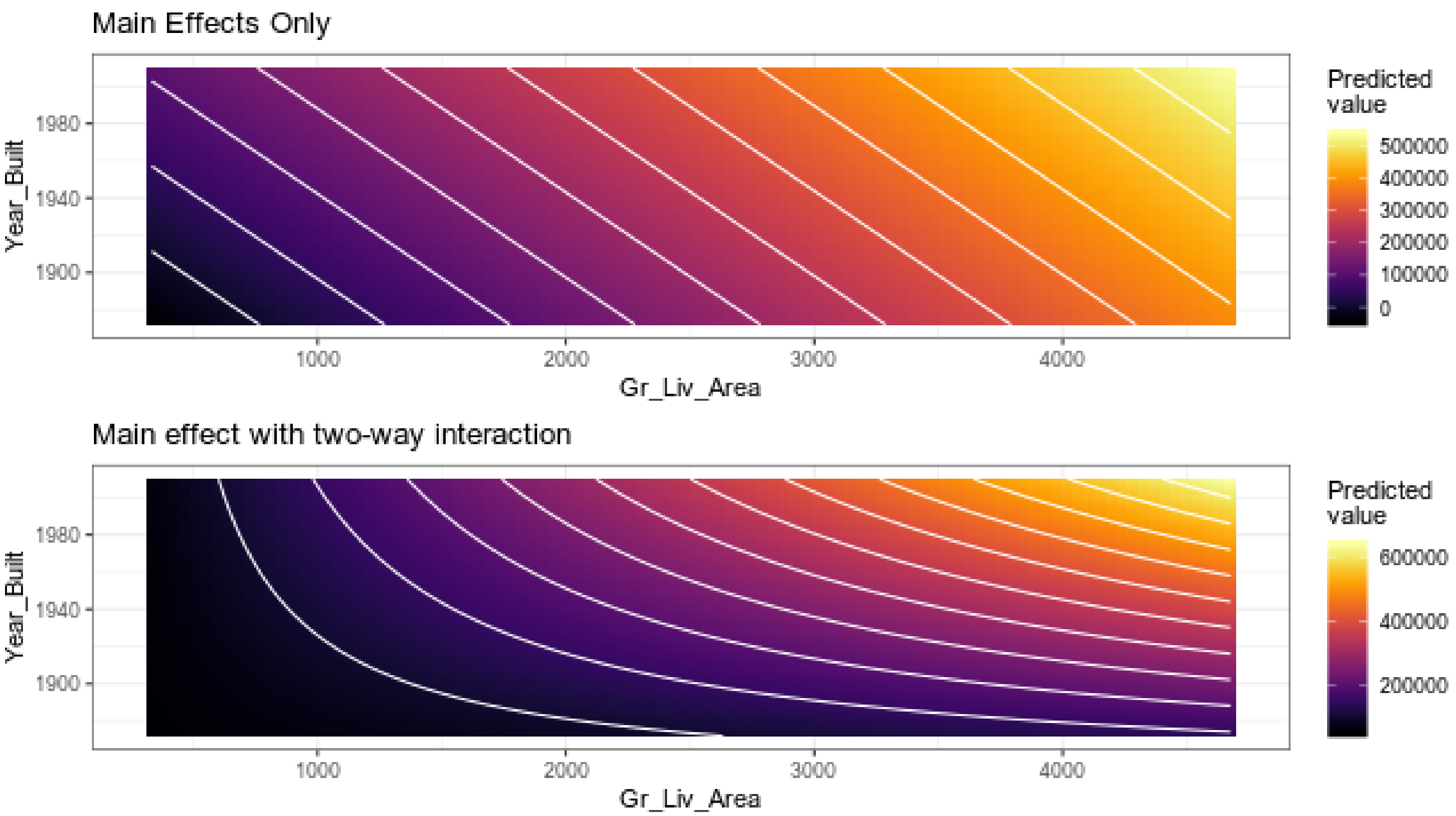
```
summary(model3 <- lm(Sale_Price ~ Gr_Liv_Area + Year_Built + Gr_Liv_Area:Year_Built, data = ame
```

```

Call:
lm(formula = Sale_Price ~ Gr_Liv_Area + Year_Built + Gr_Liv_Area:Year_Built, 
    data = ames_train)

Residuals:
    Min      1Q  Median      3Q     Max 
-440543  -25191   -1896   17599  281542 

Coefficients:
                           Estimate   Std. Error t value             Pr(>|t|)
(Intercept)            382194.30149 209192.64043   1.827               0.0678
Gr_Liv_Area             -1483.88103    125.65052 -11.810 <0.0000000000000002
Year_Built               -179.79795    106.40443  -1.690               0.0912
Gr_Liv_Area:Year_Built      0.80371      0.06378  12.601 <0.0000000000000002
                          
(Intercept)            .  
Gr_Liv_Area            ***
Year_Built             .  
Gr_Liv_Area:Year_Built ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 44810 on 2049 degrees of freedom
Multiple R-squared:  0.6887,    Adjusted R-squared:  0.6883 
F-statistic:  1511 on 3 and 2049 DF,  p-value: < 0.00000000000000022
```

```
round(coef(model3), 3)
```

```
           (Intercept)            Gr_Liv_Area             Year_Built
            382194.301              -1483.881               -179.798
Gr_Liv_Area:Year_Built
                 0.804
```

```
fit1 <- lm(Sale_Price ~ Gr_Liv_Area + Year_Built, data = ames_train)
fit2 <- lm(Sale_Price ~ Gr_Liv_Area * Year_Built, data = ames_train)

# Regression plane
plot_grid <- expand.grid(
   Gr_Liv_Area = seq(from = min(ames_train$Gr_Liv_Area), to = max(ames_train$Gr_Liv_Area),
                     length = 100),
   Year_Built = seq(from = min(ames_train$Year_Built), to = max(ames_train$Year_Built),
                     length = 100)
)

plot_grid$y1 <- predict(fit1, newdata = plot_grid)
plot_grid$y2 <- predict(fit2, newdata = plot_grid)

# Level plots
p1 <- ggplot(plot_grid, aes(x = Gr_Liv_Area, y = Year_Built,
                      z = y1, fill = y1)) +
   geom_tile() +
   geom_contour(color = "white") +
   viridis::scale_fill_viridis(name = "Predicted\nvalue", option = "inferno") +
   theme_bw() +
   ggtitle("Main Effects Only")

p2 <- ggplot(plot_grid, aes(x = Gr_Liv_Area, y = Year_Built,
                      z = y2, fill = y2)) +
   geom_tile() +
   geom_contour(color = "white") +
   viridis::scale_fill_viridis(name = "Predicted\nvalue", option = "inferno") +
   theme_bw() +
   ggtitle("Main effect with two-way interaction")

gridExtra::grid.arrange(p1, p2, nrow = 2)
```



Full Model

```
model3 <- lm(Sale_Price ~ ., data = ames_train)
broom::tidy(model3)
```

```
# A tibble: 283 x 5
   term                                  estimate std.error statistic p.value
   <chr>                                    <dbl>     <dbl>     <dbl>   <dbl>
 1 (Intercept)                           -5.61e6 11261881.    -0.498   0.618
 2 MS_SubClassOne_Story_1945_and_Older    3.56e3     3843.     0.926   0.355
 3 MS_SubClassOne_Story_with_Finished_Atti…  1.28e4    12834.     0.997   0.319
 4 MS_SubClassOne_and_Half_Story_Unfinishe…  8.73e3    12871.     0.678   0.498
 5 MS_SubClassOne_and_Half_Story_Finished_…  4.11e3     6226.     0.660   0.509
 6 MS_SubClassTwo_Story_1946_and_Newer   -1.09e3     5790.    -0.189   0.850
 7 MS_SubClassTwo_Story_1945_and_Older    7.14e3     6349.     1.12    0.261
 8 MS_SubClassTwo_and_Half_Story_All_Ages -1.39e4    11003.    -1.27    0.206
 9 MS_SubClassSplit_or_Multilevel        -1.15e4    10512.    -1.09    0.276
10 MS_SubClassSplit_Foyer                -4.39e3     8057.    -0.545   0.586
# … with 273 more rows
```

Assessing Model Accuracy

Models 1/2/3:

```r
# Train model using 10-fold cross-validation

set.seed(123) # for reproducibility

(cv_model <- train(
   form = Sale_Price ~ Gr_Liv_Area,
   data = ames_train,
   method = "lm",
   trControl = trainControl(method = "cv", number = 10)
))
```

```
Linear Regression

2053 samples
   1 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1846, 1848, 1848, 1848, 1848, 1848, ...
Resampling results:

  RMSE      Rsquared   MAE
  56410.89  0.5069425  39169.09

Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
set.seed(123)

cv_model2 <- train(
   Sale_Price ~ Gr_Liv_Area + Year_Built,
   data = ames_train,
   method = "lm",
   trControl = trainControl(method = "cv", number = 10)
)

set.seed(123)

suppressWarnings({
   cv_model3 <- train(
      Sale_Price ~ .,
      data = ames_train,
      method = "lm",
      trControl = trainControl(method = "cv", number = 10)
   )
})
```

Accuracy:

```
summary(resamples(list(
   model1 = cv_model,
   model2 = cv_model2,
   model3 = cv_model3
)))
```

```

Call:
summary.resamples(object = resamples(list(model1 = cv_model, model2
 = cv_model2, model3 = cv_model3)))

Models: model1, model2, model3
Number of resamples: 10

MAE
           Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
model1 34457.58 36323.74 38943.81 39169.09 41660.81 45005.17    0
model2 28094.79 30594.47 31959.30 32246.86 34210.70 37441.82    0
model3 12458.27 15420.10 16484.77 16258.84 17262.39 19029.29    0

RMSE
           Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
model1 47211.34 52363.41 54948.96 56410.89 60672.31 67679.05    0
model2 37698.17 42607.11 45407.14 46292.38 49668.59 54692.06    0
model3 20844.33 22581.04 24947.45 26098.00 27695.65 39521.49    0

Rsquared
            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
model1 0.3598237 0.4550791 0.5289068 0.5069425 0.5619841 0.5965793    0
model2 0.5714665 0.6392504 0.6800818 0.6703298 0.7067458 0.7348562    0
model3 0.7869022 0.9018567 0.9104351 0.8949642 0.9166564 0.9303504    0
```

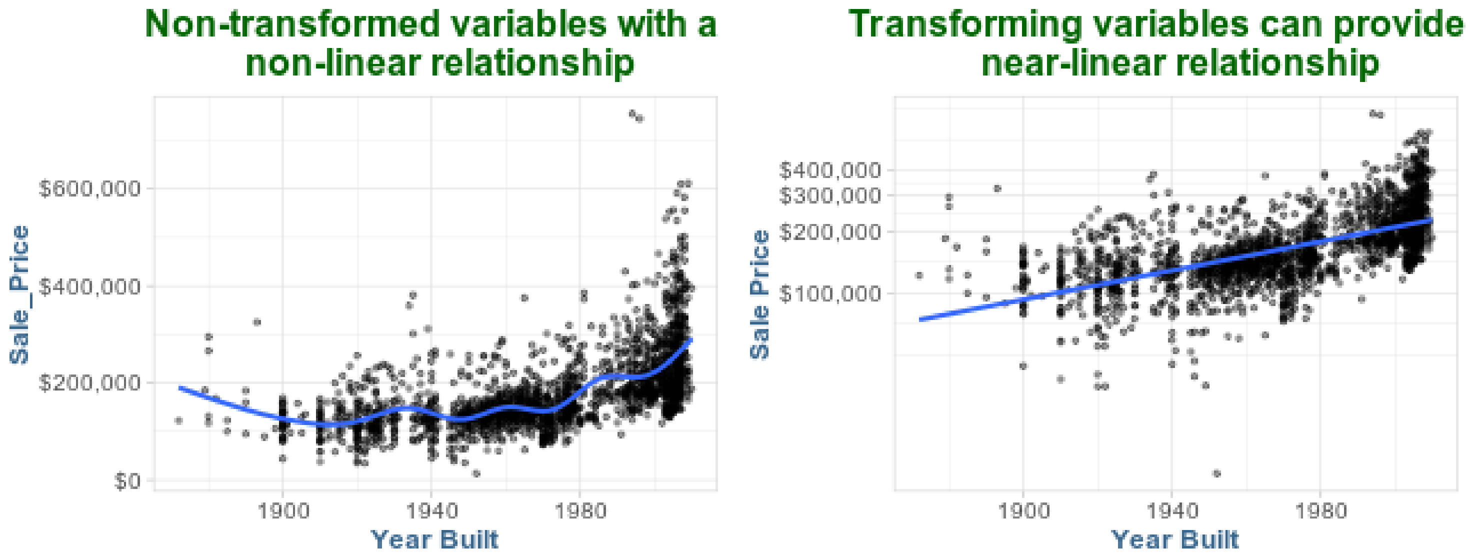
Model Concerns / Assumptions

1.) Linear Relationships

Possible solution to non-linear relationship is by variable transformations:

```
p1 <- ggplot(ames_train, aes(Year_Built, Sale_Price)) +
   geom_point(size = 1, alpha = .4) +
   geom_smooth(se = F) +
   scale_y_continuous(labels = scales::dollar) +
   xlab("Year Built") +
   ggtitle(paste("Non-transformed variables with a \n", "non-linear relationship"))

p2 <- ggplot(ames_train, aes(Year_Built, Sale_Price)) +
   geom_point(size = 1, alpha = .4) +
   geom_smooth(se = F, method = "lm") +
   scale_y_log10("Sale Price", labels = scales::dollar, breaks = seq(0, 400000, by = 100000))
   xlab("Year Built") +
   ggtitle(paste("Transforming variables can provide a \n", "near-linear relationship"))

gridExtra::grid.arrange(p1, p2, nrow = 1)
```

```
`geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



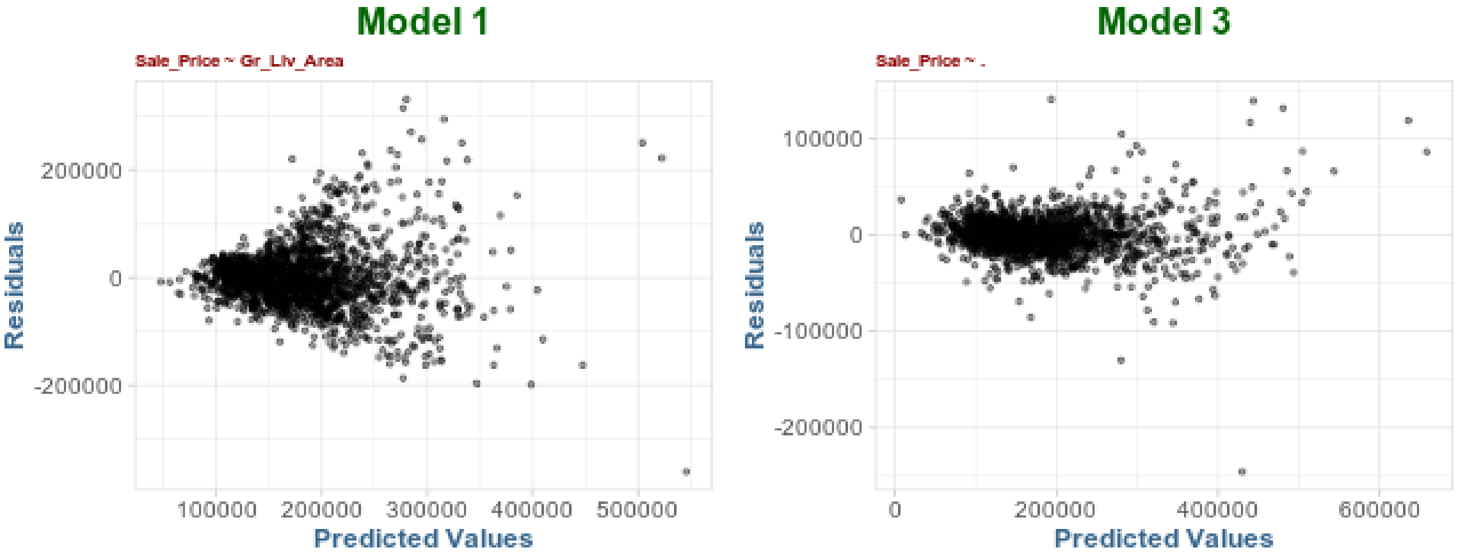
2.) Constant variance among residuals

Linear models assume constant variance amoung error terms (*homoscedasticity*).

Notice the cone shape in Model 1:

```
df1 <- broom::augment(cv_model$finalModel, data = ames_train)

p1 <- ggplot(df1, aes(.fitted, .resid)) +
   geom_point(size = 1, alpha = .4) +
   xlab("Predicted Values") +
   ylab("Residuals") +
   ggtitle("Model 1", subtitle = "Sale_Price ~ Gr_Liv_Area")

df2 <- broom::augment(cv_model3$finalModel, data = ames_train)
```

```
p2 <- ggplot(df2, aes(.fitted, .resid)) +
   geom_point(size = 1, alpha = .4) +
   xlab("Predicted Values") +
   ylab("Residuals") +
   ggtitle("Model 3", subtitle = "Sale_Price ~ .")

gridExtra::grid.arrange(p1, p2, nrow =1)
```



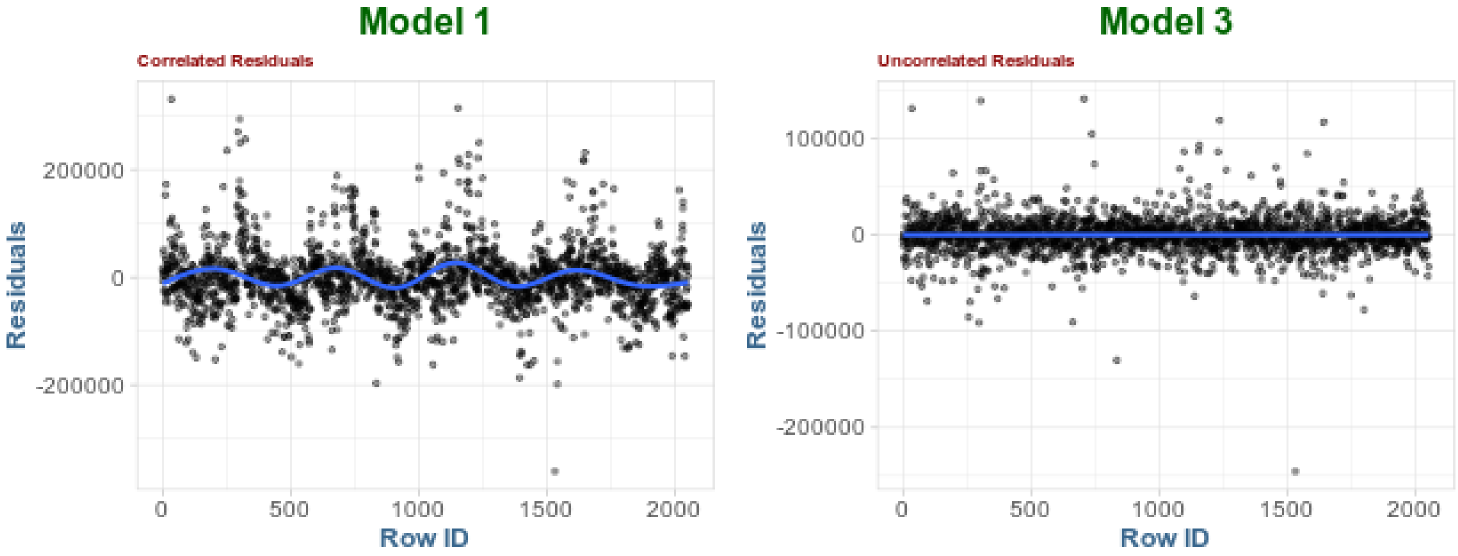
3.) No autocorrelation

Residuals should be uncorrelated and i.i.d.

```
df1 <- mutate(df1, id = row_number())
df2 <- mutate(df2, id = row_number())

p1 <- ggplot(df1, aes(id, .resid)) +
   geom_point(size = 1, alpha = .4) +
   geom_smooth(se = F) +
   xlab("Row ID") +
   ylab("Residuals") +
   ggtitle("Model 1", subtitle = "Correlated Residuals")

p2 <- ggplot(df2, aes(id, .resid)) +
   geom_point(size = 1, alpha = .4) +
   geom_smooth(se = F) +
   xlab("Row ID") +
   ylab("Residuals") +
   ggtitle("Model 3", subtitle = "Uncorrelated Residuals")
```

```
gridExtra::grid.arrange(p1, p2, nrow = 1)
```

```
`geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
`geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



Model 1 has a distict pattern to the residuals (due to being ordered by neighborhood, which is unaccounted for in the model)

4.) More observations than predictors

The number of features cannot exceed the number of observations.

(not a problem in this example)

5.)

No or little multicollinearity.

Collinearity refers to the situation where two or more predictor variables are closely related to one another.

```
cor(ames_train$Garage_Area, ames_train$Garage_Cars)
```

```
[1] 0.8906198
```

Example: *Garage_Area* and *Garage_Cars* are highly correlated.

```
summary(cv_model3) %>%
   broom::tidy() %>%
   filter(term %in% c("Garage_Area", "Garage_Cars"))
```

```
# A tibble: 2 x 5
```

```
  term        estimate std.error statistic p.value
  <chr>          <dbl>     <dbl>     <dbl>   <dbl>
1 Garage_Cars   3021.     1771.       1.71 0.0882
2 Garage_Area     19.7       6.03      3.26 0.00112
```

Only *Garage_Area* is significant (p < 0.05),

However, refit w/o *Garage_Area* term,

```
# model without Garage Area
set.seed(123)

mod_wo_Garage_Cars <- train(
   Sale_Price ~ .,
   data = select(ames_train, -Garage_Area),
   method = "lm",
   trControl = trainControl(method = "cv", number = 10)
)
```

```
Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
may be misleading

Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
may be misleading

Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
may be misleading

Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
may be misleading

Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
may be misleading

Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
may be misleading

Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
may be misleading

Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
may be misleading

Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
may be misleading

Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
```

```
may be misleading
```

```
summary(mod_wo_Garage_Cars) %>%
   broom::tidy() %>%
   filter(term == "Garage_Cars")
```

```
# A tibble: 1 x 5
  term        estimate std.error statistic       p.value
  <chr>          <dbl>     <dbl>     <dbl>         <dbl>
1 Garage_Cars    7161.     1239.      5.78 0.00000000881
```

Garage_Cars becomes significant (p < 0.001)

Best to check VIF scores (variance inflation factor)

Principal Component Analysis

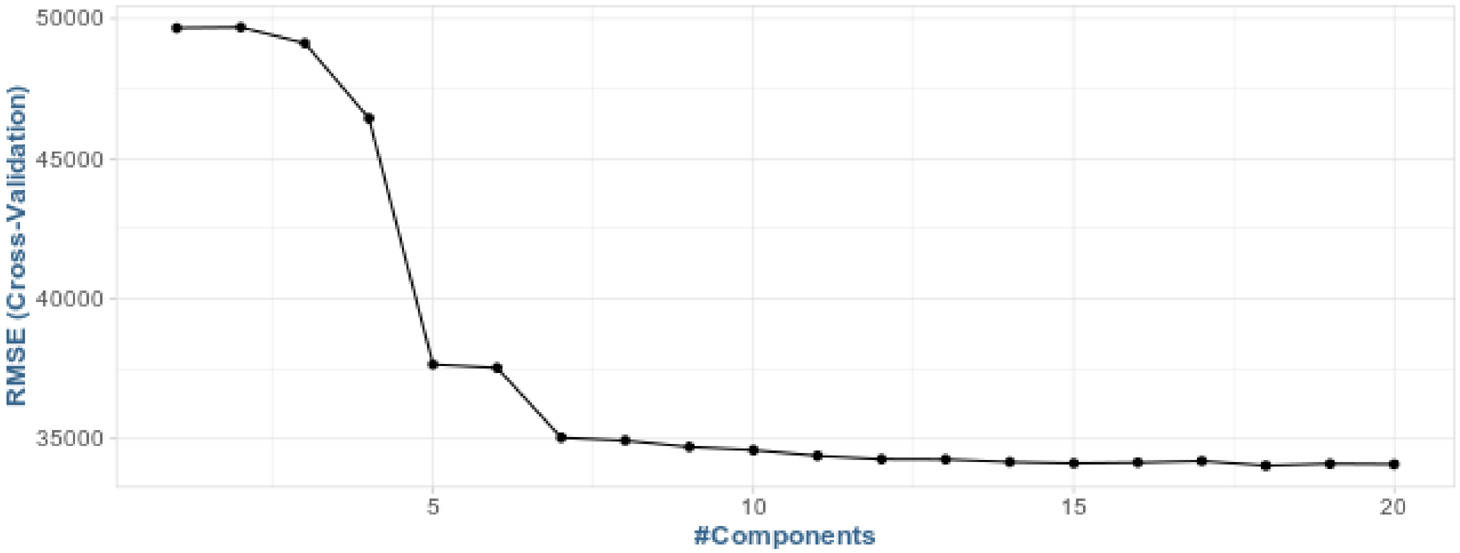
PCA -> Linear factorization of features

PCR -> Principal Component Regression

```
# performs 10-fold cross validation on a PCR model tuning the
# number of principal components to use as predictors from 1-20

set.seed(123)

cv_model_pcr <- train(
   Sale_Price ~ .,
   data = ames_train,
   method = "pcr",
   trControl = trainControl(method = "cv", number = 10),
   preProcess = c("zv", "center", "scale"),
   tuneLength = 20
)

cv_model_pcr$bestTune
```

```
   ncomp
18    18
```

```r
ggplot(cv_model_pcr)
```



```r
summary(cv_model_pcr)
```

```
Data:   X dimension: 2053 294
    Y dimension: 2053 1
Fit method: svdpc
Number of components considered: 18
TRAINING: % variance explained
          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
X           5.987    8.766    11.29    13.39    15.36    17.14    18.58
.outcome   61.142   61.167    62.09    66.28    78.06    78.09    80.87
          8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
X           19.96    21.12     22.23     23.31     24.39     25.41     26.41
.outcome    80.89    81.33     81.39     81.68     81.74     81.89     81.98
          15 comps  16 comps  17 comps  18 comps
X            27.37     28.30     29.23     30.14
.outcome     82.01     82.01     82.12     82.28
```

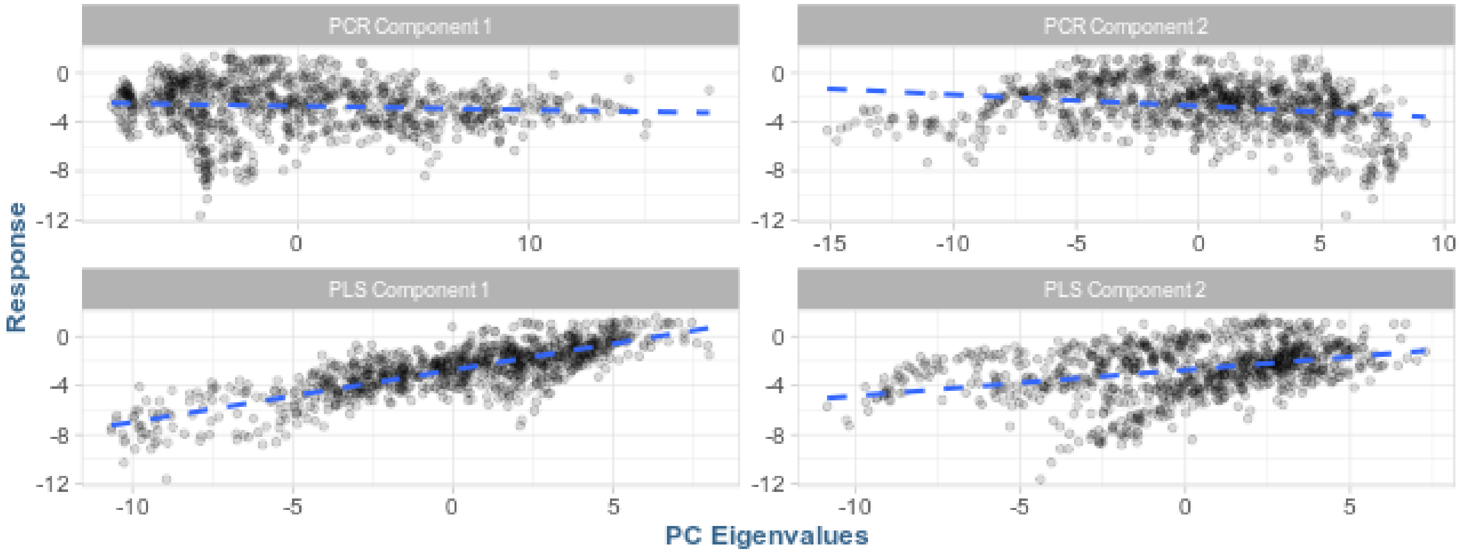
Partial least squares

PLS can be viewed as a supervised dimension reduction procedure.

Below is a comparision between PCR and PLS:

```r
data(solubility)

df <- cbind(solTrainX, solTrainY)
```

```
pca_df <- recipe(solTrainY ~ ., data = df) %>%
   step_center(all_predictors()) %>%
   step_scale(all_predictors()) %>%
   step_pca(all_predictors()) %>%
   prep(training = df, retain = T) %>%
   juice() %>%
   select(PC1, PC2, solTrainY) %>%
   rename(`PCR Component 1` = "PC1", `PCR Component 2` = "PC2") %>%
   gather(component, value, -solTrainY)

pls_df <- recipe(solTrainY ~., data = df) %>%
   step_center(all_predictors()) %>%
   step_scale(all_predictors()) %>%
   step_pls(all_predictors(), outcome = "solTrainY") %>%
   prep(training = df, retain = T) %>%
   juice() %>%
   rename(`PLS Component 1` = "PLS1", `PLS Component 2` = "PLS2") %>%
   gather(component, value, -solTrainY)

pca_df %>%
   bind_rows(pls_df) %>%
   ggplot(aes(value, solTrainY)) +
   geom_point(alpha = .15) +
   geom_smooth(method = "lm", se = F, lty = "dashed") +
   facet_wrap(~ component, scales = "free") +
   labs(x = "PC Eigenvalues", y = "Response")
```

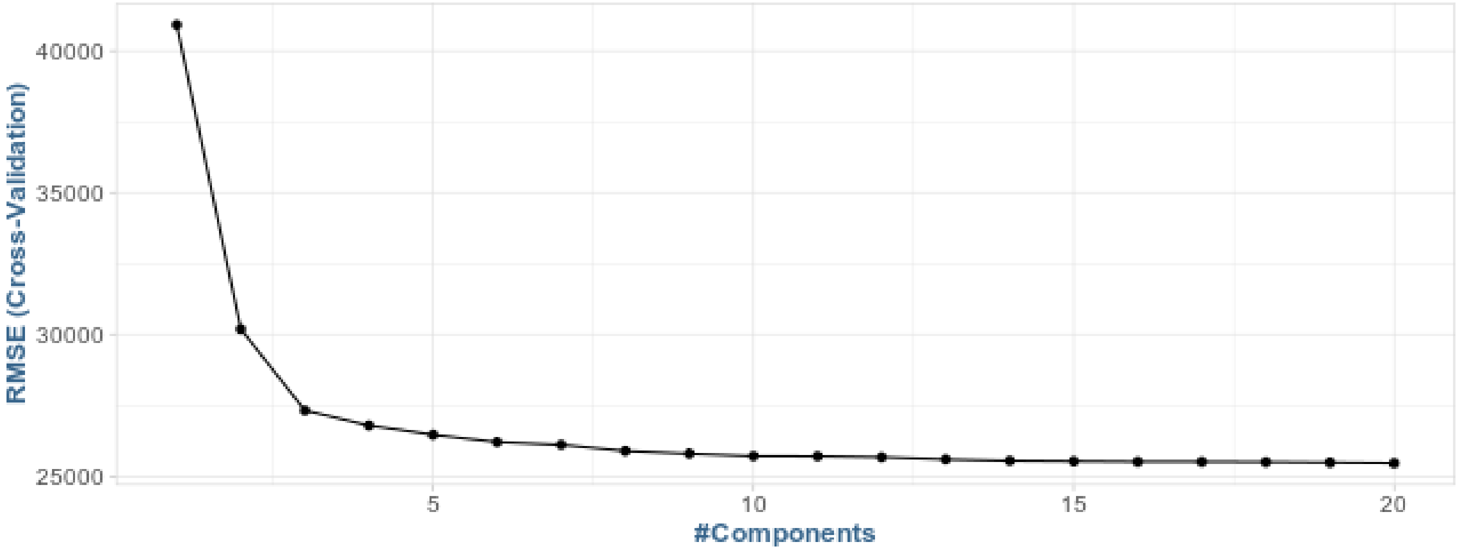


Fit PLS to ames:

```
# perform 10-fold cross validation on a PLS model tuning the
# number of principal compoents to use as predictors from 1-20

set.seed(123)

cv_model_pls <- train(
   Sale_Price ~ .,
   data = ames_train,
   method = "pls",
   trControl = trainControl(method = "cv", number = 10),
   preProcess = c("zv", "center", "scale"),
   tuneLength = 20
)

# model with lowest RMSE

cv_model_pls$bestTune
```

```
   ncomp
20    20
```

```
ggplot(cv_model_pls)
```



Feature interpretation

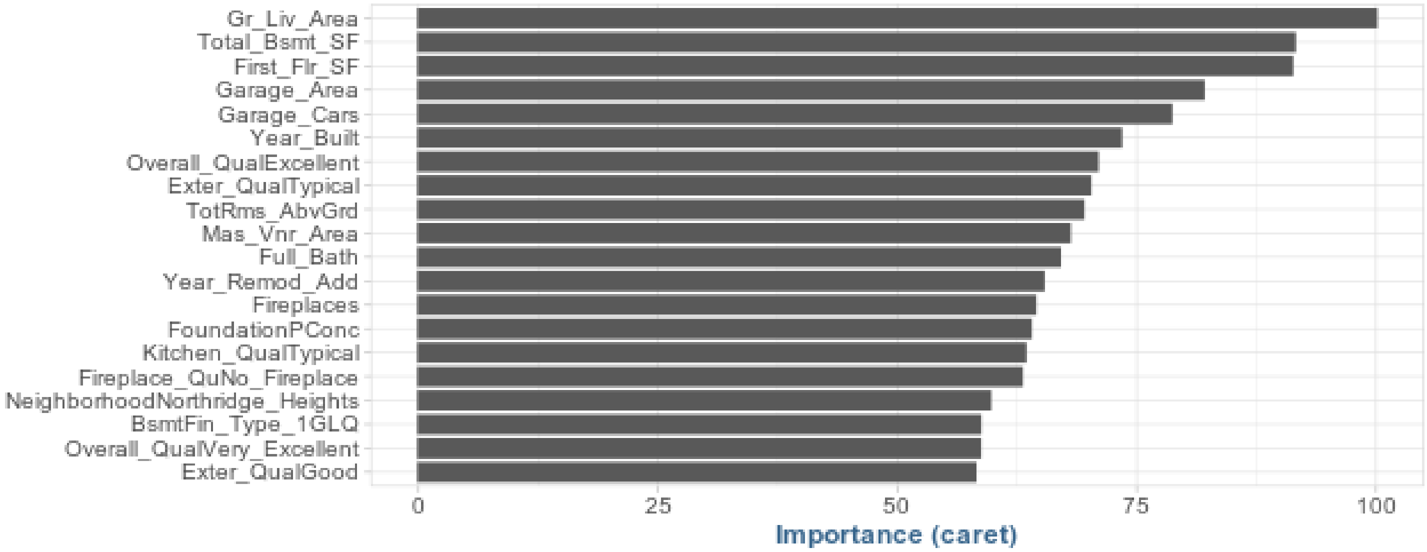
Linear models are monotonic, meaning 1 unit change in X means a constant change in Y.

In PLS we can measure the importance of features by an absolute weighted sum of the regression coefficents.

We can use vip to extract and plot the most important variables:

```
vip(cv_model_pls, num_features = 20, method = "model")
```

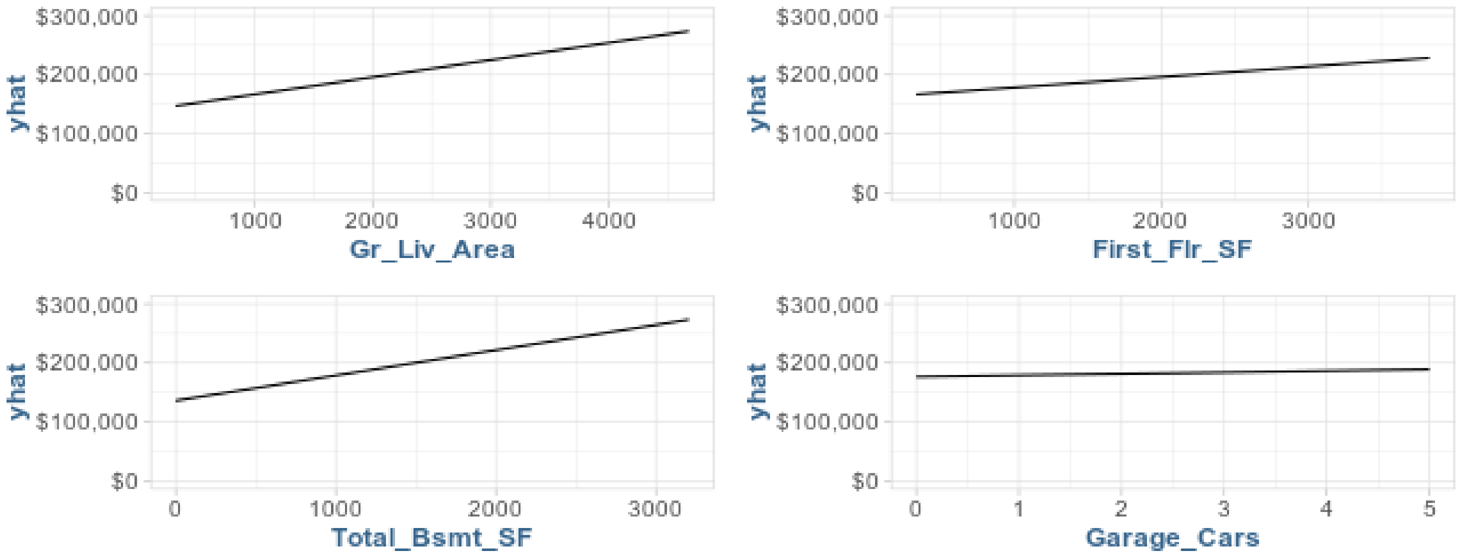
```

Attaching package: 'pls'

The following object is masked from 'package:caret':

    R2

The following object is masked from 'package:stats':

    loadings
```



Alternatively, we can use PDP (partial dependence plots) to summarize the relationship.

```
p1 <- pdp::partial(cv_model_pls, pred.var = "Gr_Liv_Area", grid.resolution = 20) %>%
  autoplot() +
  scale_y_continuous(limits = c(0, 300000), labels = scales::dollar)

p2 <- pdp::partial(cv_model_pls, pred.var = "First_Flr_SF", grid.resolution = 20) %>%
  autoplot() +
  scale_y_continuous(limits = c(0, 300000), labels = scales::dollar)

p3 <- pdp::partial(cv_model_pls, pred.var = "Total_Bsmt_SF", grid.resolution = 20) %>%
  autoplot() +
  scale_y_continuous(limits = c(0, 300000), labels = scales::dollar)

p4 <- pdp::partial(cv_model_pls, pred.var = "Garage_Cars", grid.resolution = 4) %>%
  autoplot() +
  scale_y_continuous(limits = c(0, 300000), labels = scales::dollar)
```

```
grid.arrange(p1, p2, p3, p4, nrow = 2)
```



We see all 4 of the top predictive features have a positive linear relationship.

```
# clean up
rm(list = ls())
```