Inclass Example

Mei Maddox

2022-10-13

```
## Warning: Unknown levels in `f`: RP
```

```
## Warning: Unknown levels in `f`: RP
```

```
## Warning: Unknown levels in `f`: RRNe
```

```
## Warning: Unknown levels in `f`: RRNn, RRAn, RRNe, RRAe
```

```
## Warning: Unknown levels in `f`: 2.5Fin
```

```
## Warning: Unknown levels in `f`: Mix
```

```
## Warning in `[<-.factor`(`*tmp*`, is.na(electrical), value = "Other"): invalid
## factor level, NA generated
```

```
## Warning: Unknown levels in `f`: ImStucc, Stone
```

```
## Warning: Unknown levels in `f`: Ex
```

```
## Warning in xtfrm.data.frame(x): cannot xtfrm data frames
```

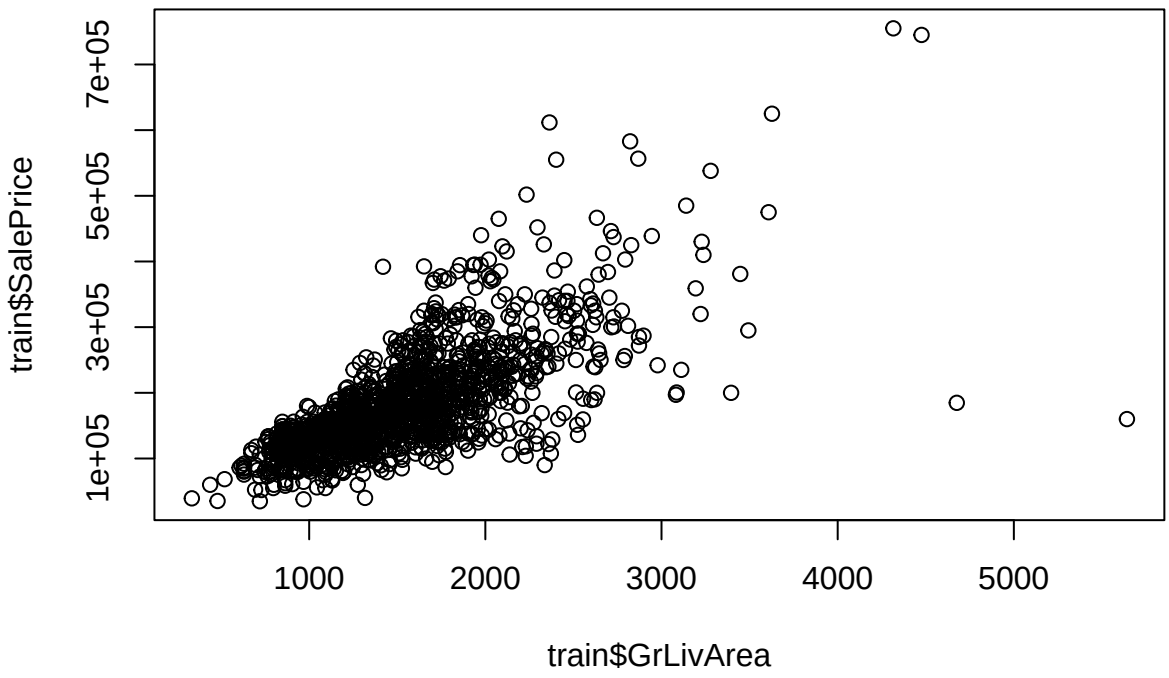
```
## Warning in xtfrm.data.frame(x): cannot xtfrm data frames
```

```
## Warning in xtfrm.data.frame(x): cannot xtfrm data frames
```

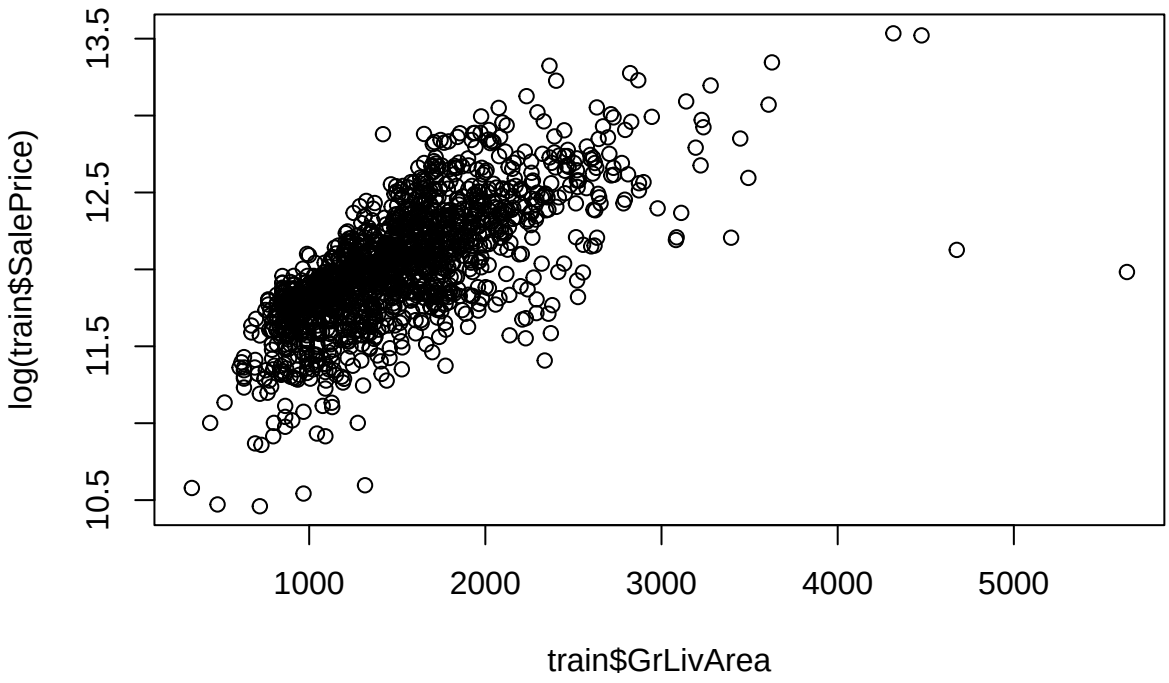
```
## Warning in xtfrm.data.frame(x): cannot xtfrm data frames
```

First Try for building a predictive model, using just one variable, but as a smooth function:

```
plot(train$SalePrice ~ train$GrLivArea)
```

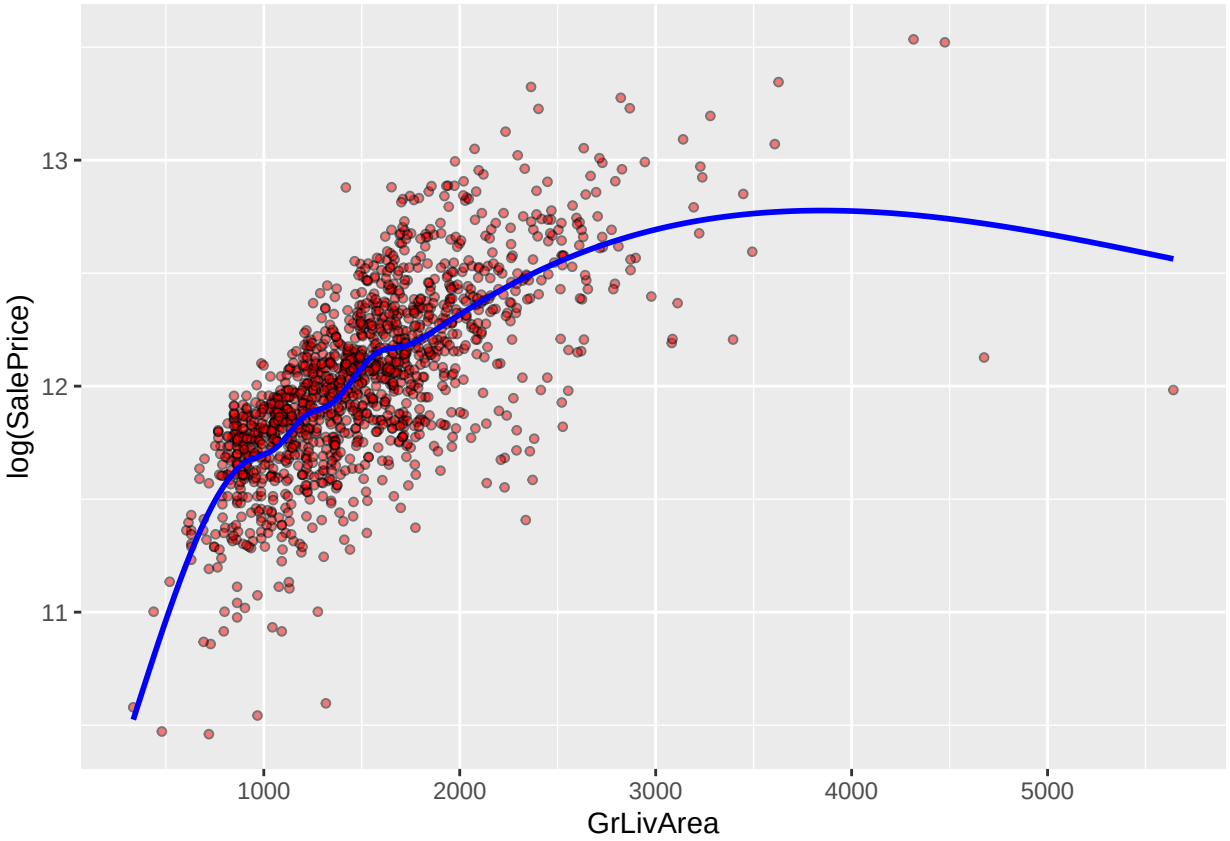



```
plot(log(train$SalePrice) ~ train$GrLivArea)
```



Better to log-transform response

```
fit1 = lm(log(SalePrice) ~ ns(GrLivArea, df=10), data=train)
seqGrLivArea = seq(min(train$GrLivArea), max(train$GrLivArea), length.out=200)
predictedSpline = predict(fit1, newdata = data.frame(GrLivArea=seqGrLivArea))
ggplot(data=train,
       aes(x=GrLivArea,
           y=log(SalePrice)
       )
  ) +
  geom_point(pch=21, fill="red", size=1.2, alpha=0.5) +
  geom_line(
    data = data.frame(
      x = seqGrLivArea,
      y = predictedSpline
    ),
    aes(
      x=x,
      y=y
    ),
    color = "blue", size = 1
  )
```



Preparing the dataset with the predicted sale prices for the test data:

```
predicted.SalePrice = exp(predict(fit1, newdata=data.frame(GrLivArea = test$GrLivArea)))
SubmitDF = data.frame(Id=test$Id, SalePrice=predicted.SalePrice)
write.csv(file='../submissions/inclass1.csv', SubmitDF, row.names = FALSE)
```

Submitting this file to the Kaggle competition, I obtained a "Prediction error", measures as

$$\sum(\log(\hat{y}_i) - \log(y_i))^2$$

of 0.28857, where $\hat{y}_i$ is my prediction of the sale price of the ith house in the test data, and y_i is the actual sale price only known to Kaggle.

Second Try, including all predictors!

If we include all predictors, one issue is that a few predictors might have a lot of NA values, and then the corresponding observation is not used in the fit. (You find this out when you try to fit the full model.) Let's see which variables have the most NA's.

```
train %>%
  summarize(across(everything(), ~sum(is.na(.)))) %>%
  sort(decreasing=TRUE)
```

```
## Warning in xtfrm.data.frame(x): cannot xtfrm data frames
```

```
## # A tibble: 1 x 79
##   LotFrontage GarageYrBlt    Id MSSubClass MSZoning LotArea Street Alley
##         <int>       <int> <int>      <int>    <int>   <int>  <int> <int>
## 1         259          81     0          0        0       0      0     0
```

```
## # ... with 71 more variables: LotShape <int>, LandContour <int>,
## #   LotConfig <int>, LandSlope <int>, Neighborhood <int>, Condition1 <int>,
## #   Condition2 <int>, BldgType <int>, HouseStyle <int>, OverallQual <int>,
## #   OverallCond <int>, YearBuilt <int>, YearRemodAdd <int>, RoofStyle <int>,
## #   Exterior1st <int>, Exterior2nd <int>, MasVnrType <int>, MasVnrArea <int>,
## #   ExterQual <int>, ExterCond <int>, Foundation <int>, BsmtQual <int>,
## #   BsmtCond <int>, BsmtExposure <int>, BsmtFinType1 <int>, ...
```

```
dim(train)
```

```
## [1] 1460   79
```

Removing/Replacing Missing Values

It is best to get the training dataset that has no missing values (since they will be discarded in the fitting process of the full model anyway), and then check if any other predictors are constant. Which variables still have a lot of missing values:

```
train %>%
  summarize(across(everything(), ~sum(is.na(.x)))) %>%
  sort(decreasing = TRUE)
```

```
## Warning in xtfrm.data.frame(x): cannot xtfrm data frames
```

```
## # A tibble: 1 x 79
##   LotFrontage GarageYrBlt    Id MSSubClass MSZoning LotArea Street Alley
##         <int>       <int> <int>      <int>    <int>   <int>  <int> <int>
## 1         259          81     0          0        0       0      0     0
## # ... with 71 more variables: LotShape <int>, LandContour <int>,
## #   LotConfig <int>, LandSlope <int>, Neighborhood <int>, Condition1 <int>,
## #   Condition2 <int>, BldgType <int>, HouseStyle <int>, OverallQual <int>,
## #   OverallCond <int>, YearBuilt <int>, YearRemodAdd <int>, RoofStyle <int>,
## #   Exterior1st <int>, Exterior2nd <int>, MasVnrType <int>, MasVnrArea <int>,
## #   ExterQual <int>, ExterCond <int>, Foundation <int>, BsmtQual <int>,
## #   BsmtCond <int>, BsmtExposure <int>, BsmtFinType1 <int>, ...
```

For now, I'm going to drop `LotFrontage` from consideration, although we could impute values. I'm also going to drop `GarageYrBlt` from consideration, because it has around 80 missing values for those garages where there is no information. Since we have info on the garage from other variables, I rather keep 81 observations in the dataset, but not include `GarageYrBlt`. So, I'm going to drop `GarageYrBlt` from the list of predictors:

```
train = train %>% select(-LotFrontage, -GarageYrBlt)
test = test %>% select(-LotFrontage, -GarageYrBlt)
```

Fitting the model with almost all variables

We can now fit the full model:

```
SalePrice = train$SalePrice
HouseId = train$Id #just in case we need it
train = train %>% select(-Id, -SalePrice)
fit2 = lm(log(SalePrice) ~ . , data=train)
```

We can now try to predict the sales price based on the variables in the test data, since we have addressed all missing values in th test data:

```
predicted.SalePrice2 = exp(predict(fit2, newdata=test))
```

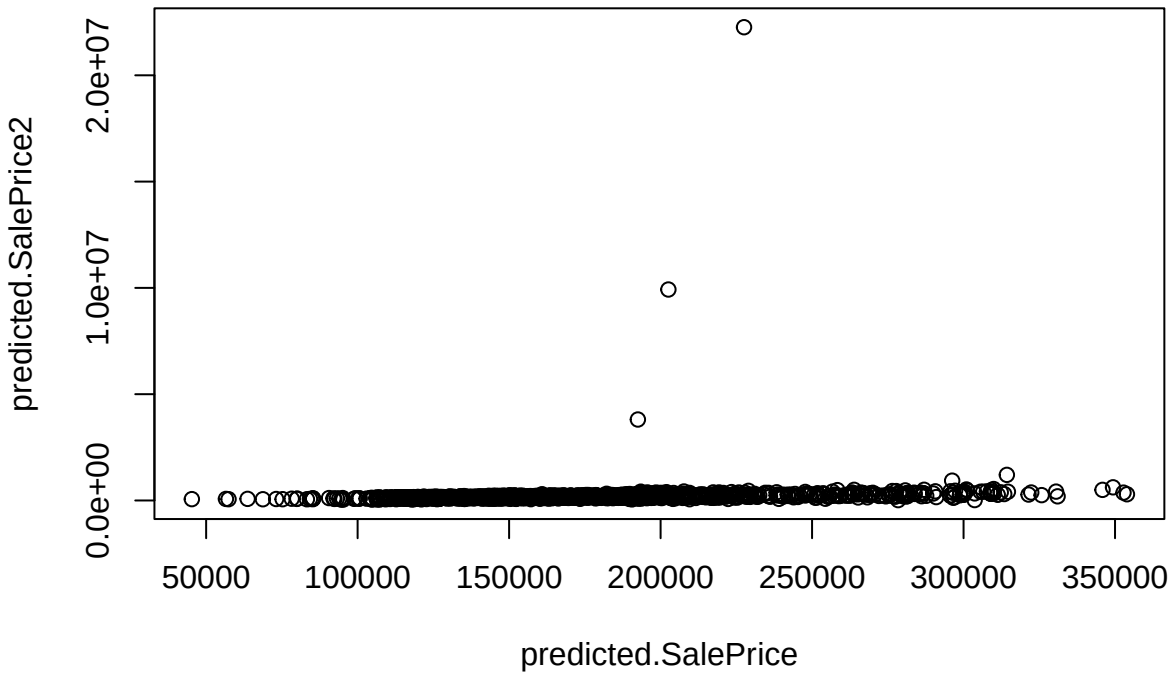
```
## Warning in predict.lm(fit2, newdata = test): prediction from a rank-deficient
## fit may be misleading
```

Preparing the dataset with the predicted sale prices for the test data:

```
SubmitDF = data.frame(Id=test$Id, SalePrice=predicted.SalePrice2)
write.csv(file='../submissions/inclass2.csv', SubmitDF, row.names = FALSE)
```

Interestingly, using all these variables, the prediction score did not go down by much. It is now 0.26450. What is the relationship between our predictions based on the two models:

```
plot(predicted.SalePrice2 ~ predicted.SalePrice)
```



This is pretty telling. Just for a few houses (three), we predicted a much higher price with the second model compared to the first. Which houses are these:

```
SubmitDF %>% slice_max(SalePrice,n=8)
```

```
##        Id  SalePrice
## 1140 2600 22261168.2
## 1044 2504  9924031.0
## 961  2421  3808022.8
## 1090 2550  1204635.6
## 1251 2711   932354.8
## 1223 2683   618442.0
## 1168 2628   538556.1
## 20   1480   516352.8
```

For the house with ID 2600 in the test data, we predicted a sales price of over 22 million! The error alone in this prediction could be huge! To find out, I'm replacing just the prediction for the 5 most expensive predicted prices with the maximum sales price found in the training data.

```
SubmitDF$SalePrice[SubmitDF$Id %in% c(2600, 2504, 2421, 2550, 2711)] = max(SalePrice)
write.csv(file='../submissions/inclass3.csv', SubmitDF, row.names = FALSE)
```

Yes, the prediction error went down to 0.19609!