Assignment 1: Predict 454

Rahul Sangole

2018-10-15

Overview of Methodology

The approach used for this assignment, modeling the "Two month's Salary" dataset is as follows:

1. Data are split into train and test datasets before further consumption.
2. Exploratory Data Analysis (EDA) was conducted at a univariate, bivariate and multivariate level using numerical and graphical summarizations.
3. The EDA influenced the data preparation stage in terms of transforming the response variable, and adding new features to the predictor matrix.
4. Models are built on two sets of data preparation: (i) with no interaction terms, and (ii) with interaction terms. The types of models investigated are linear regressions, variable selection procedures on regression (best subsets, forward & backward selection, lasso & ridge), tree models (recursive partition trees & conditional inference trees) and random forest models.
5. Two staged of model building are conducted. In the first stage, models are built using individual R commands using each model's respective package (for example, `lm`, `regsubsets`, `party` and `partyKit` etc). This allowed a deep dive into the residuals and diagnostics for each model, which influenced the modeling strategy. Once the strategy is finalized, the `caret` package is used to build all models again using the `caret::train` interface. `caret::trainControl` is used to perform 10-fold cross validation to perform hyperparameter tuning and to determine the variation of performance for each model.
6. RMSE, MAE and RSquared are the metrics used to judge the performance of each model.
7. Once all the models are built using `caret`, each model is applied to the test dataset.
8. Model performance is compared on both the training and testing data, which reveals which model has generalized the best.

Exploratory Data Analysis

Univariate and multivariate EDA is performed on the data. Two of the more interesting plots are presented here. The first plot is a ternary plot showing the relationship between 4 variables, `price`, `cut`, `carat` and `clarity`. `Price` is plotted as a color. Right away, we can see that `price` increases with `carat`, but seems independent of `color` and `clarity`. Secondly, there are two curious structures in the data. At (i) and (ii), there seem to be unusually linear

relationships in the data, which could indicate some imputation or data entry process during the data gathering process.

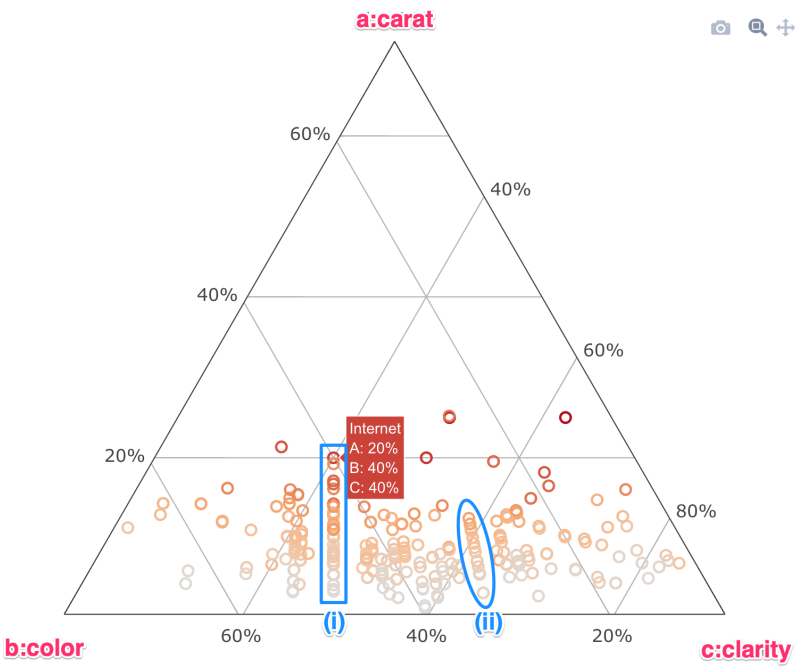

Figure 1: Ternary plot showing the relationship between price, carat, color and clarity. Price is colored from low (in gray) to high (in red)

Another very useful method for data exploration is utilizing interactive parallel plots using `plotly`. Several key insights are obtained. For example, the plot below, `carat` values *exactly* equal to 2.0 are selected. We can see that all these data points are from the Blue Nile store only. Also, though `carat` and `price` are linearly related, oddly this relationship breaks down for `price` values above $20,000.

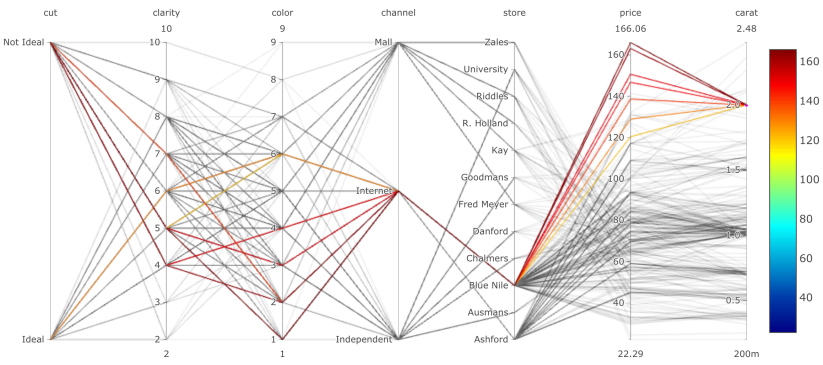

Figure 2: Interactive parallel lines plot created using plotly, with lines highlighted for carat value exactly equal to 2.0

The false color level plot attached in the appendix also shows that there are very few stores which sell diamonds at a very high price point. This could be

useful to improve model performance.

Data Preparation

The original data are split into a `train` and `test` split using a stratified 70:30 split. The stratification is performed on the `price` response variable, to ensure that the resulting distributions are similar in nature, as shown to the right. The splitting is carried out using `caret::createDataPartition()`.

The response variable is quite right skewed with a small number of observations (81) beyond the 80th percentile. This combined with residual analysis for the initial linear regression models showed the need to transform the response. Box-cox analysis is conducted which recommends a `lambda` value of 0.2. This is applied to `price` which makes the variable a lot more normally distributed, as shown the figure to the right.

Two additional feature variables are created: (i) Binary variable indicating if `carat` is exactly equal to 2.0, and (ii) Binary variable indicating if `carat` is greater than 2.0. Model-matrix dataframes are created using `model.matrix()` to convert the factor variables into 0/1 indicator variables. Second-order interaction terms are then included for all possible combinations and stored separately. Thus, there are two training dataframe available: No interactions (282, 19), and 2nd-order interactions (282, 134). While modeling, partial 2nd-order interactions are dynamically tested using the formula interface as well.

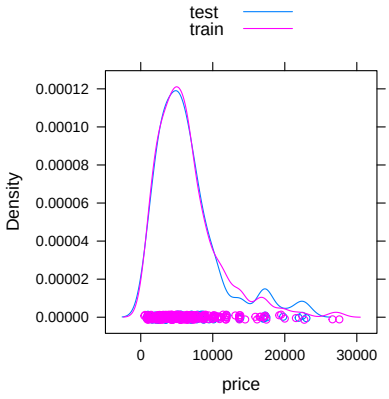

Figure 3: Density plots for the Train-Test split

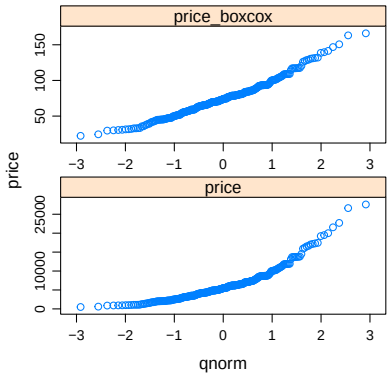

Figure 4: QQ plot showing the improvement of normality upon box-cox transformation of the pricing variable

Modeling

The overview of the modeling approach is as follows:

1. Linear regression models
    1. Simple multilinear regression
    2. Variable selection procedures: . Best subsets regression . Forward selection . Backward selection
    3. Shrinkage methods: . Lasso . Ridge
2. Tree models: . Recursive partition trees . Conditional inference trees
3. Random forest models

Details of each model are summarized below.

Linear models

Linear regression models using `lm()` are fit on the data with and without the predictor transformation. Both models are statistically significant with RSquared values of 0.92 and 0.93 respectively. There are two `store` levels which do not have a fit estimate because of a linear combination which exists

after dummy variable creation. This is addressed later in the modeling. The raw price linear model's residuals show a pattern which does get rectified upon box-cox transformation of the price variable. The appendix shows the summary of the linear model which shows all the major variables are statistically significant (except a few stores_ indicator variables), including the two new feature variables.

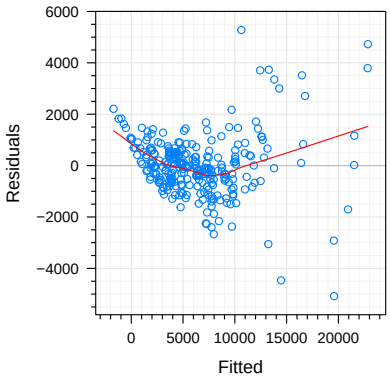

Figure 5: Summary of lm on raw price value

Variable selection using `bsr`

Various methods of variable selection are run on both the styles of the dataset, including backward and forward using the `leaps` package, and best subsets using `regsubsets`. For the backwards selection on the data without interaction terms, the variables selected are shown below. This model results in an RSquared value of 0.90, slightly lower than the linear model. The best RMSE for backwards selection is 0.239, with an MAE of 0.182.

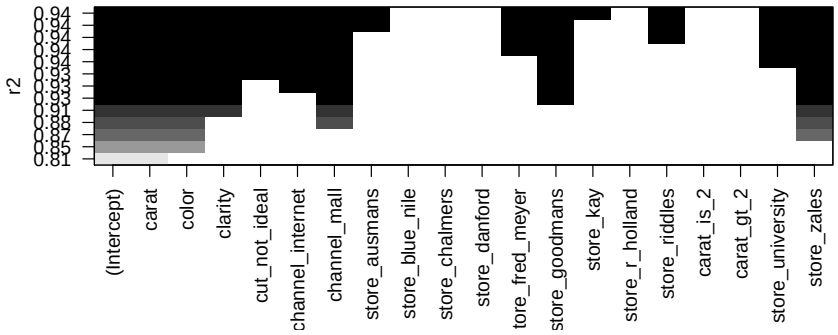

Figure 6: Variable selection using RSquared, backward selection

Forward selection gives a better RSquared than backward, given it includes a higher number of predictor variables. The best RMSE for forwards selection is 0.205, with an MAE of 0.167, which is much better than the backwards selection model.

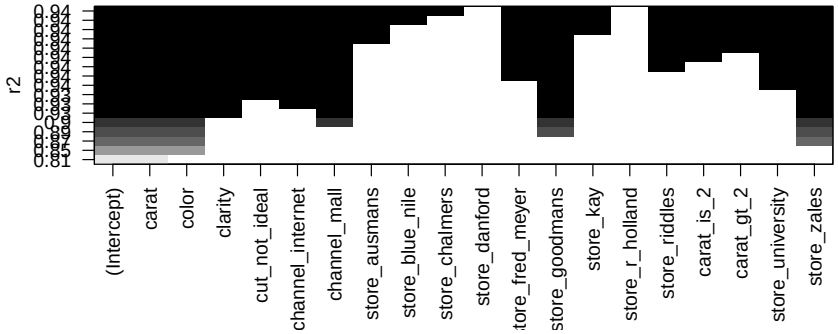

Figure 7: Variable selection using RSquared, forward selection

Shrinkage models

For lasso and ridge models, the dataset with interactions is used, since for the dataset without interactions, upon cross validation, no terms were eliminated for the dataset without interactions. The plot below shows the tranceplot for the variation in coefficients as the strength of regularization is changed for the ridge regression model. As expected `carat` has the strongest coefficient and is retained the longest, given it's strong correlation with the predictor. A large percentage of the remaining 100s of variables turn near zero. Cross validation is used to decide the hyperparameter.

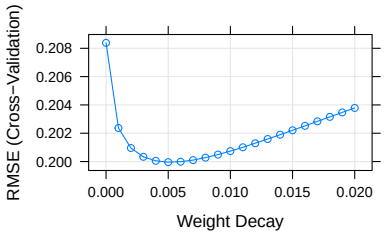

Figure 8: Cross validation to fine tune the ridge regression parameter

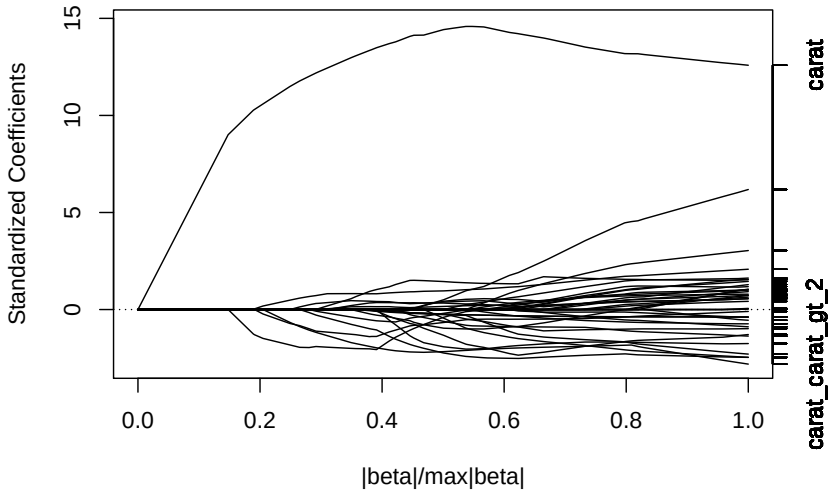

Figure 9: Ridge regression trace plot

Tree models

Two models are investigated in this section: a decision tree using `rpart`, and a conditional inference tree using `ctree`. For the `rpart` model, the decision tree obtained is shown here. As expected, `carat` is the dominant predictor, followed by `color` and `store`. The conditional trees are much more difficult to visually depict, though they performed much better than a simpler decision tree, as expected, since the simpler trees tend to have low bias and very high variance.

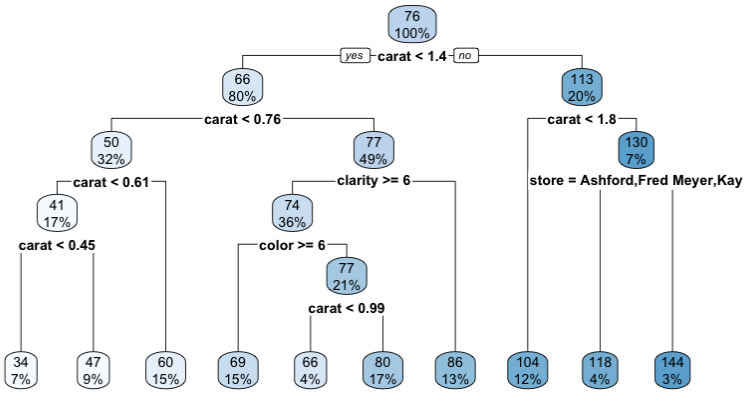

Figure 10: rpart tree

Random Forest models

RF models are built using `caret` and the `randomForest` package. Although both datasets are used for RF models, the dataset with explicit interactions performed much worse. Since trees can perform interactions without having to explicitely define them, the explicit definitions were simply adding unnecessary variables for the model to search through.

Hyperparameters were tuned via 10 fold cross validation. `mtry` which selects how many variables to randomly pick at each split was selected to be 16 (out of 19), while the number of random trees built in each model stablizes after ~150 trees. The CV search result is shown here which shows 16 parameters.

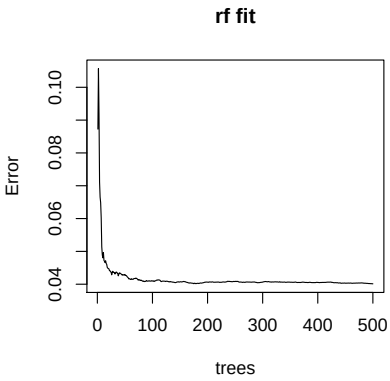

Figure 11: RF error as a function of number of trees

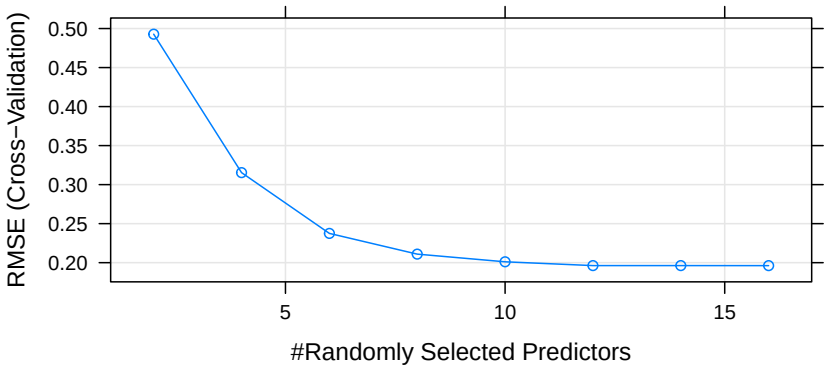

Figure 12: Cross validation to fine tune the ridge regression parameter

Comparison

In total, 10 models are built, each with 10-fold cross validation performed on the training dataset, for a total of 100 individual models. `caret` package is used to build these models and also post process them. This processing is described below.

Training set performance

The advantage of running all the models using `caret` is that this package allows the user to easily compare the performance of all the models using various procedures. For example, the plot below shows the comparison of the two metrics MAE and RMSE for each model, for each of the 10 validation sets in a 10-fold cross validation. This is still on the training dataset, but *should* be indicative of performance on the test dataset. MAE and RMSE show very similar performance for roughly all the models, except the Random Forest models which show a higher RMSE than linear regression, vs RF models show a lower MAE than linear regression. Since MAE is less sensitive to outliers, the high RMSE is probably driven by some of the outliers identified earlier. The dotplot shows the point estimates and 95% error bars for the RSquared values. Here, Lasso and Backward selection models with 2nd order interactions score the highest.

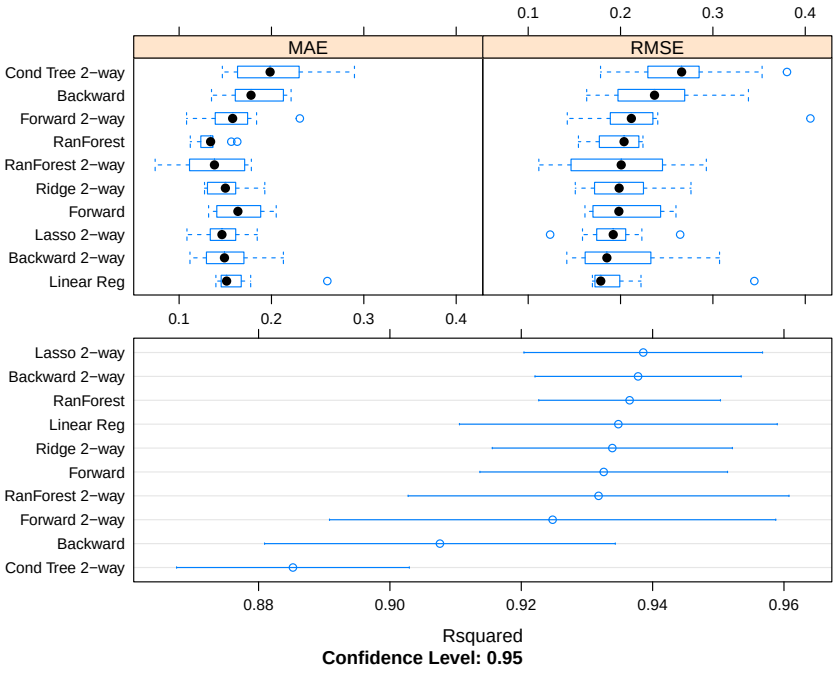

Figure 13: Comparison of the model performance for each of the 10 resamples in the 10-fold CV

Test set performance

The table below compares the performance of the models when applied on the test set, which so far has remained untouched throughout the modeling process. Now, we can see that the RF model is stored at the very top, though only by a very slim margin. Lasso and Ridge are in the lead as well, while linear regression (lowest training RMSE model) is now at position 6.

Table 1: Results Compared

modelNames	R2_train	R2_test	RMSE_train	RMSE_test	MAE_train	MAE_test
RanForest	0.9872	0.9488	0.0886	0.1813	0.0611	0.1245
Lasso 2-way	0.9499	0.9419	0.1718	0.1877	0.1298	0.1326
RanForest 2-way	0.9886	0.9412	0.0857	0.1948	0.0579	0.1312
Ridge 2-way	0.9526	0.9407	0.1671	0.1900	0.1303	0.1358
Forward	0.9393	0.9371	0.1890	0.2004	0.1528	0.1390
Linear Reg	0.9393	0.9371	0.1890	0.2004	0.1528	0.1390

We can compare the relative performance of the models on point estimates of all three metrics for the training and testing runs. This will allow us to understand how much each model has tended to overfit to the training data. As expected, in all cases, test set performance is worse than the training set. In terms of the least gap between the training and test set performance, the Lasso model with 2nd order interaction has done the best. in terms of the absolute best predictive performance, the RF model (without explicit interaction terms included) has performed the best.

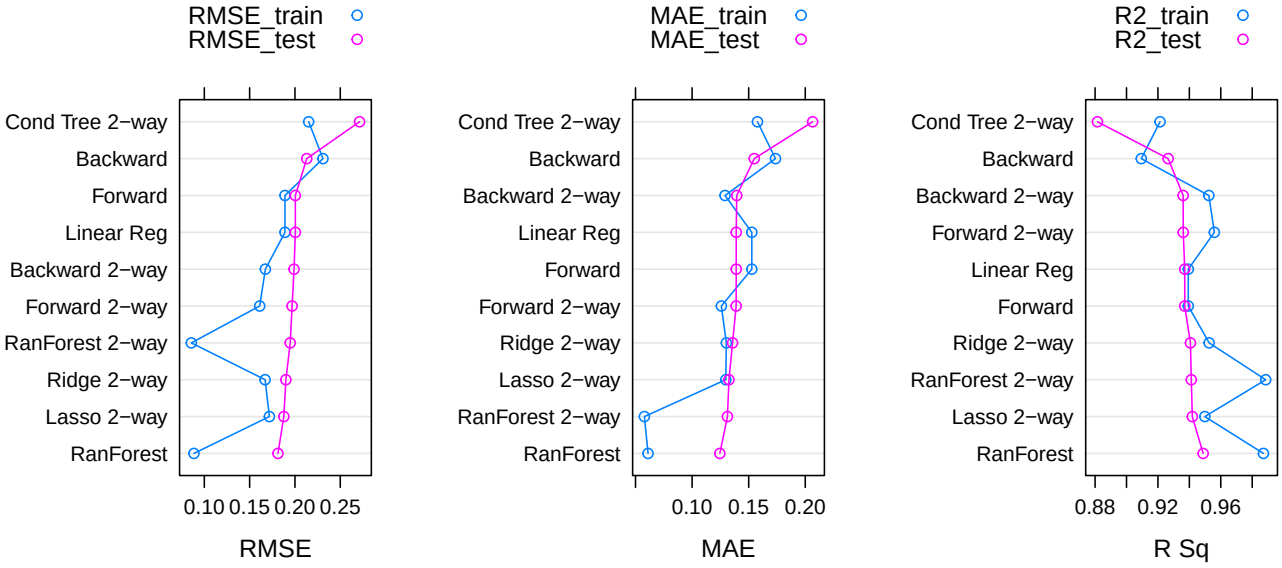

Figure 14: Comparison of the final models on train and test datasets

The scatter plot of the price prediction against the actual for both the datasets are shown here, which shows quite a strong perdictive performance. The model tends to deviate from truth at the extreme ends of the price spec-

trum, with stronger performance in the mid-section of price, where there are maximum number of points.

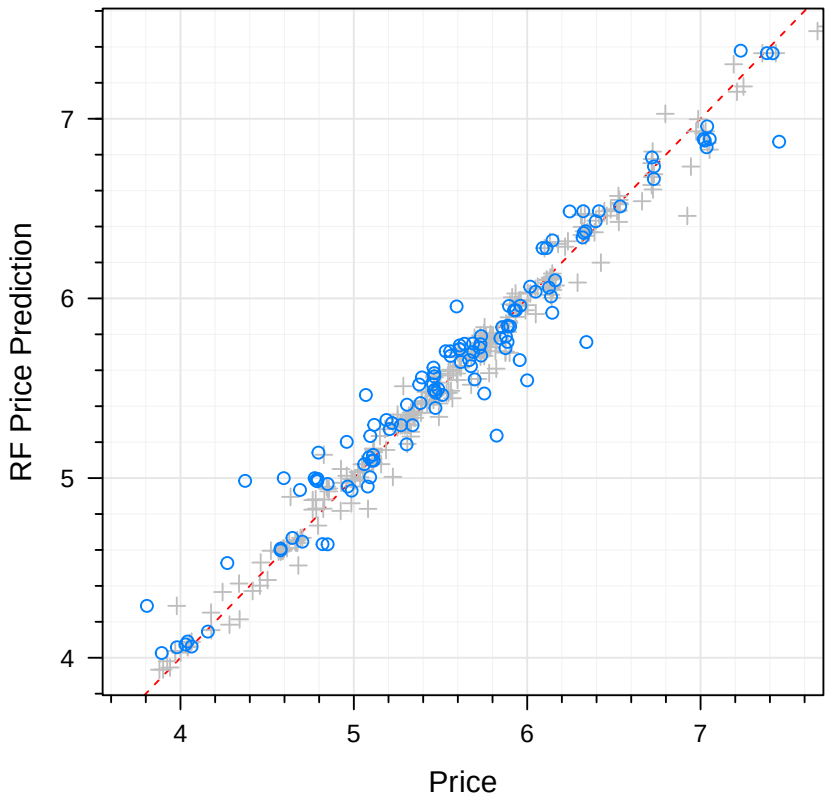

Figure 15: Random Forest performance. Gray points show train data, blue points show test data.

Since the objective of this assignment is purely predictive in nature, I would select the Random Forest model (with no explicit interactions).

The entire codebase is stored on github.

Appendix

Level plot

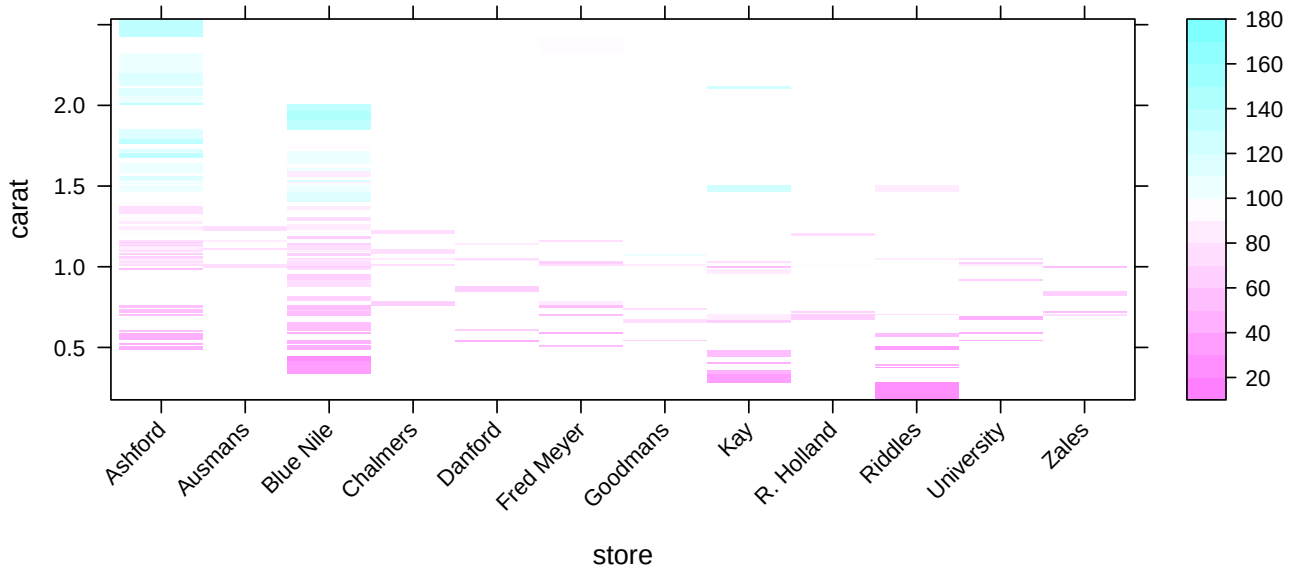

Figure 16: False color level plot showing that the few stores sell diamonds with a very high price

Linear model summary

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5080.1  -595.5   -52.2   524.2  5277.1
##
## Coefficients: (2 not defined because of singularities)
##                   Estimate Std. Error t value
## (Intercept)       3119.76     592.42   5.266
## carat             9817.32     236.58  41.496
## color             -653.08      44.54 -14.664
## clarity           -632.54      53.16 -11.899
## cut_not_ideal     -609.62     167.88  -3.631
## channel_internet  -442.40     448.16  -0.987
## channel_mall      2275.36     656.28   3.467
## store_ausmans      274.20     646.97   0.424
## store_blue_nile    -42.77     193.94  -0.221
```

```
## store_chalmers    -165.69     645.67  -0.257
## store_danford      207.22     616.08   0.336
## store_fred_meyer -1227.03     615.49  -1.994
## store_goodmans    2869.23     652.33   4.398
## store_kay          646.22     601.71   1.074
## store_r_holland   -291.25     711.85  -0.409
## store_riddles     1147.69     632.73   1.814
## store_university       NA         NA      NA
## store_zales            NA         NA      NA
## carat_is_2        5005.64     537.03   9.321
## carat_gt_2       -2095.43     492.76  -4.252
##                  Pr(>|t|)
## (Intercept)      2.89e-07 ***
## carat             < 2e-16 ***
## color             < 2e-16 ***
## clarity           < 2e-16 ***
## cut_not_ideal    0.000339 ***
## channel_internet 0.324471
## channel_mall     0.000614 ***
## store_ausmans    0.672047
## store_blue_nile  0.825624
## store_chalmers   0.797681
## store_danford    0.736877
## store_fred_meyer 0.047228 *
## store_goodmans   1.58e-05 ***
## store_kay        0.283817
## store_r_holland  0.682769
## store_riddles    0.070835 .
## store_university       NA
## store_zales            NA
## carat_is_2        < 2e-16 ***
## carat_gt_2       2.94e-05 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 1220 on 264 degrees of freedom
## Multiple R-squared:  0.9283, Adjusted R-squared:  0.9236
## F-statistic: 200.9 on 17 and 264 DF,  p-value: < 2.2e-16
```

Important caret code

This code shows the power of utilizing `caret` models with the `tidyverse` particularly list-columns and `purrr` to quickly investigate results.

```
results_df <- tibble(
    modelNames = c(
        'Backward',
        'Backward 2-way',
        'Forward',
        'Forward 2-way',
        'Lasso 2-way',
        'Ridge 2-way',
        'Linear Reg',
        'RanForest',
        'RanForest 2-way',
        'Cond Tree 2-way'
    ),
    yhat_train = list(
        pred_train_bwd,
        pred_train_bwd_2w,
        pred_train_fwd,
        pred_train_fwd_2w,
        pred_train_lasso,
        pred_train_ridge,
        pred_train_lm_sqrt,
        pred_train_rf,
        pred_train_rf_2w,
        pred_train_tree
    ),
    yhat_test = list(
        pred_test_bwd,
        pred_test_bwd_2w,
        pred_test_fwd,
        pred_test_fwd_2w,
        pred_test_lasso,
        pred_test_ridge,
        pred_test_lm_sqrt,
        pred_test_rf,
        pred_test_rf_2w,
        pred_test_tree
    )
) %>%
    mutate(
        train_residuals = purrr::map(yhat_train, ~ (.x - train_y_sqrt)),
        test_residuals =  purrr::map(yhat_test, ~ (.x - test_y_sqrt)),
        R2_train = purrr::map_dbl(yhat_train, ~ caret::R2(.x, train_y_sqrt)),
        R2_test = purrr::map_dbl(yhat_test, ~ caret::R2(.x, test_y_sqrt)),
```

```
        RMSE_train = purrr::map_dbl(yhat_train, ~ caret::RMSE(.x, train_y_sqrt)),
        RMSE_test = purrr::map_dbl(yhat_test, ~ caret::RMSE(.x, test_y_sqrt)),
        MAE_train = purrr::map_dbl(yhat_train, ~ caret::MAE(.x, train_y_sqrt)),
        MAE_test = purrr::map_dbl(yhat_test, ~ caret::MAE(.x, test_y_sqrt))
    )
results_df
p1 <- results_df %>%
    arrange(RMSE_test) %>%
    mutate(modelNames = factor(modelNames,levels = modelNames)) %>%
    dotplot(modelNames~RMSE_train+RMSE_test,.,auto.key=T, type='b', xlab='RMSE')
p2 <- results_df %>%
    arrange(MAE_test) %>%
    mutate(modelNames = factor(modelNames,levels = modelNames)) %>%
    dotplot(modelNames~MAE_train+MAE_test,.,auto.key=T, type='b',xlab='MAE')
p3 <- results_df %>%
    arrange(-R2_test) %>%
    mutate(modelNames = factor(modelNames,levels = modelNames)) %>%
    dotplot(modelNames~R2_train+R2_test,.,auto.key=T, type='b',xlab='R Sq')
(final_result_dotplot <- gridExtra::grid.arrange(p1,p2,p3,ncol=3)
```