Homework #3

Michael Munguia and Karim Hammoud

10/24/2021

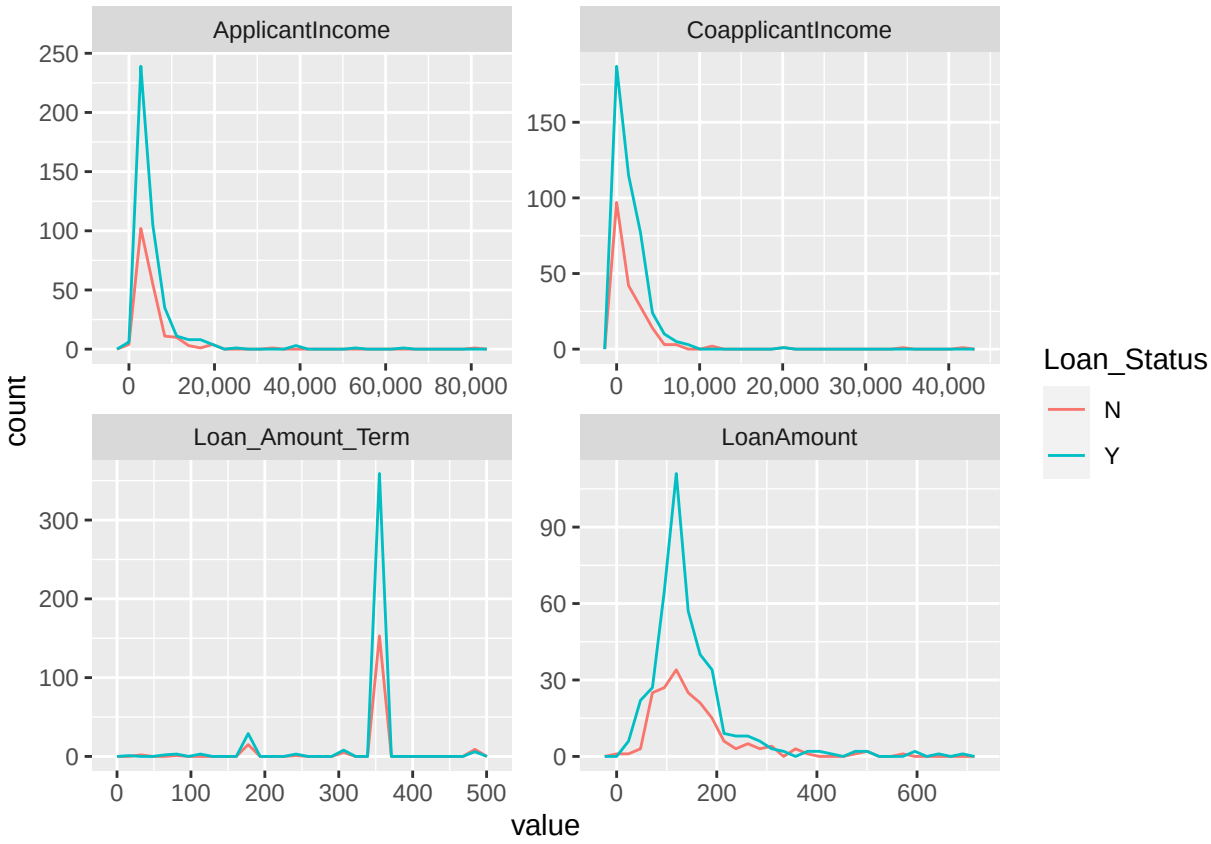
Section 1: Exploratory Data Analysis (EDA) & Data Preparation

Exploratory Analysis (EDA)

Given that our target variable is an approved loan status (`Loan_Status = "Y"`), the very first observation we might make is that about 69% of the applicants represented in the data set were applicants with an approved loan. Surveying the continuous variables available, we see that applicant/co-applicants' incomes and loan amounts share similarly right-skewed distributions, with the two income variables being more extreme in their skew. These three variables share a sense of scale (i.e. they refer to monetary amounts and two are incomes) whereas the loan term is on a different scale and measurement (months) along with its displaying a left-skewed distribution.

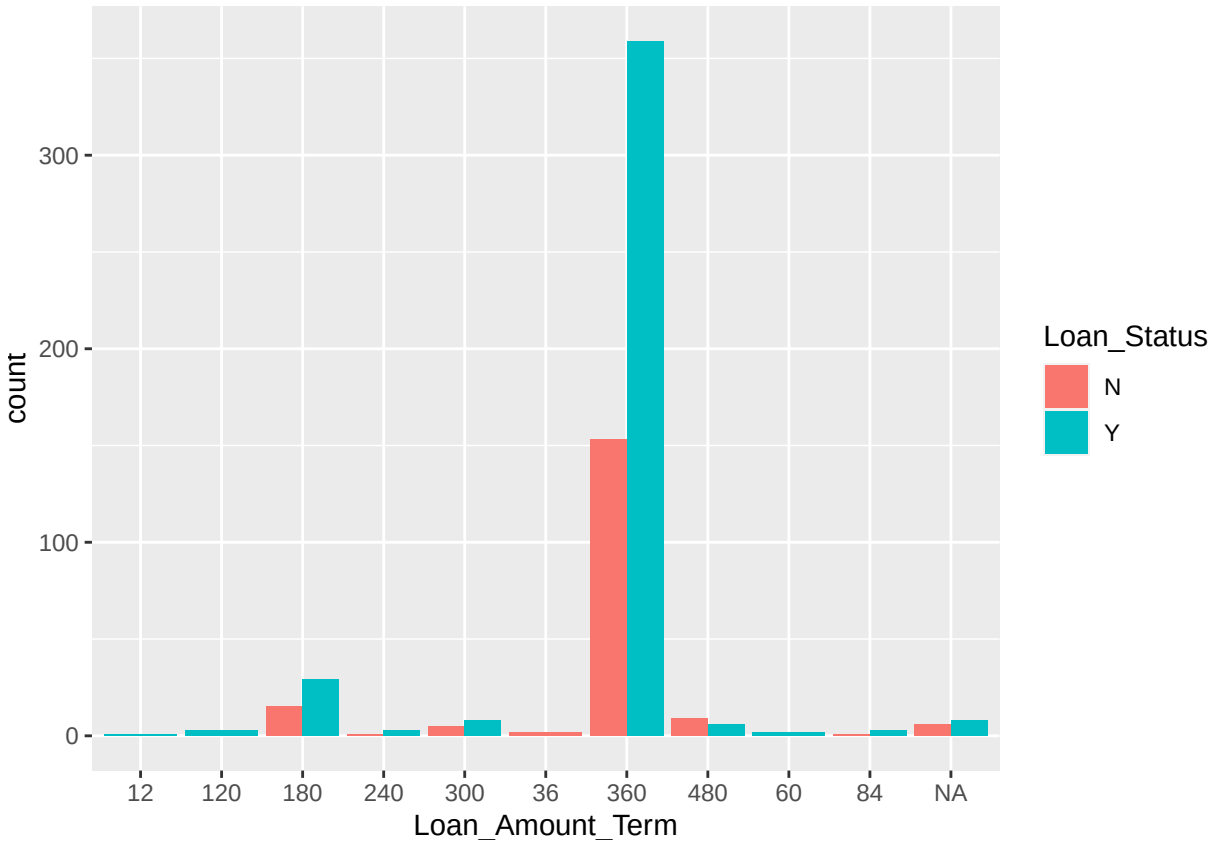
It is notable that the variable representing the income of a co-applicant, `CoapplicantIncome`, is an untidy variable - it represents both whether a co-applicant was involved in the process *and* their income. There is insufficient context to understand how to best utilize this variable. For example, a co-applicant may be a married spouse, blood relative or even a domestic partner. We may be able to intuit and better engineer features based on this information, but without further context we cannot know for certain how to handle unmarried applicants vis-a-vis their `CoapplicantIncome` value.

```
cont_vars <- c(
  "ApplicantIncome", "CoapplicantIncome", "LoanAmount", "Loan_Amount_Term"
  )

disc_vars <- c(
  "Gender", "Married", "Dependents", "Education", "Self_Employed",
  "Property_Area", "Credit_History"
  )

show_dist <- function(df) {
  df %>%
    ggplot(aes(value, color = Loan_Status)) +
    geom_freqpoly(bins = 30) +
    facet_wrap("name", scales = "free") +
    scale_x_continuous(labels = scales::comma_format())
}

pivot_longer(loans, all_of(cont_vars)) %>% show_dist()
```



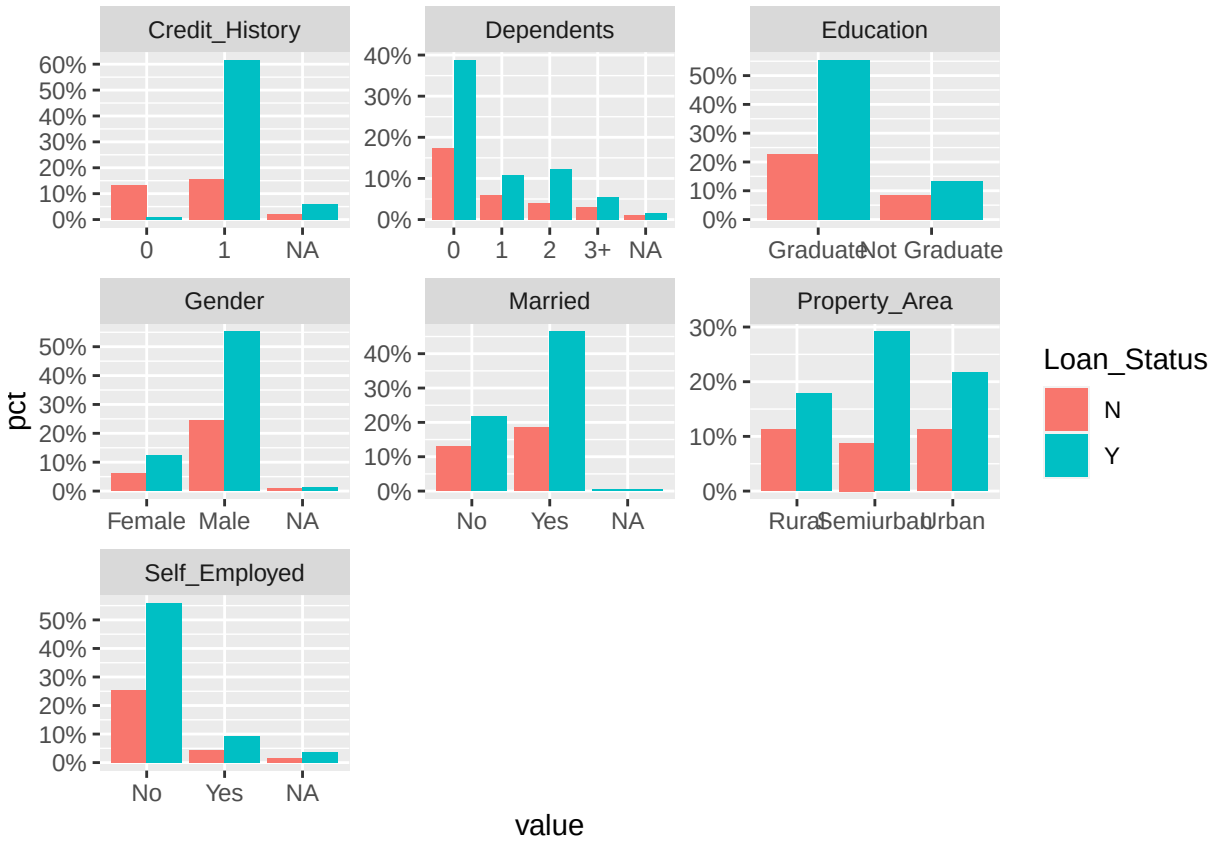
Knowing that we will be utilizing linear discriminant analysis downstream, it's important to identify that, with the exception of `Loan_Amount_Term`, we may be able to apply a transformation to force these variables into a more Gaussian shape. In fact, `Loan_Amount_Term` should truly be considered a discrete variable as it seems to follow fixed 12-day increments. When visualized, we see that nearly all the applications were for 360 day loans.

```
loans <- mutate(loans, across("Loan_Amount_Term", as.character))

loans %>%
  ggplot(aes(x = Loan_Amount_Term, fill = Loan_Status)) +
  geom_bar(position = "dodge")
```



Taking the remaining data set and visualizing loan status for each discrete variable, we can see some interesting trends. Credit history seems to play a crucial role, with applicants meeting guidelines (`Credit_History = 1`) showing only 7 denied-loans across a data set of 614 applicants. A majority proportion of the approved-loan group have no dependents, graduated from college and are male. That last feature initially gave some pause as it may have been indicative of a discriminatory bias, however, it appears that an overwhelming 80% of applicants identify as male.

```
long_disc <- loans %>%
  mutate(across("Credit_History", as.character)) %>%
  select(all_of(c(disc_vars, "Loan_Status"))) %>%
  pivot_longer(all_of(disc_vars)) %>%
  count(name, value, Loan_Status) %>%
  group_by(name) %>%
  mutate(total = sum(n)) %>%
  ungroup() %>%
  mutate(pct = n / total)


long_disc %>%
  ggplot(aes(x = value, y = pct, fill = Loan_Status)) +
  geom_col(position = "dodge") +
  scale_y_continuous(breaks = seq(0, 1, 0.1), labels = scales::percent_format(1)) +
  facet_wrap("name", scales = "free")
```



Married applicants make up a majority in both groups, and across the whole data set. While different types of property area are well represented across both groups, there is some differentiation in how they are represented within each group. The self-employed are less represented in the approved-loan group, though this may be an artifact of the lack of representation of many self-employed applicants to begin with - they make up only 13% of applicants (or a count of 82).

Data Preparation

The transformations will not be directly applied to the data set itself as that would make potential changes more cumbersome to deal with down the road. Instead, the data will be divided into randomly selected training, validation and test data sets. While pre-processing/transformations specific to the given choice of model will be part of downstream modeling, we will remove `NA` values we know will interrupt the process at this present stage. The training set is 70% of the data with validation and test sets evenly split at 15% over the overall total of 541 observations used.

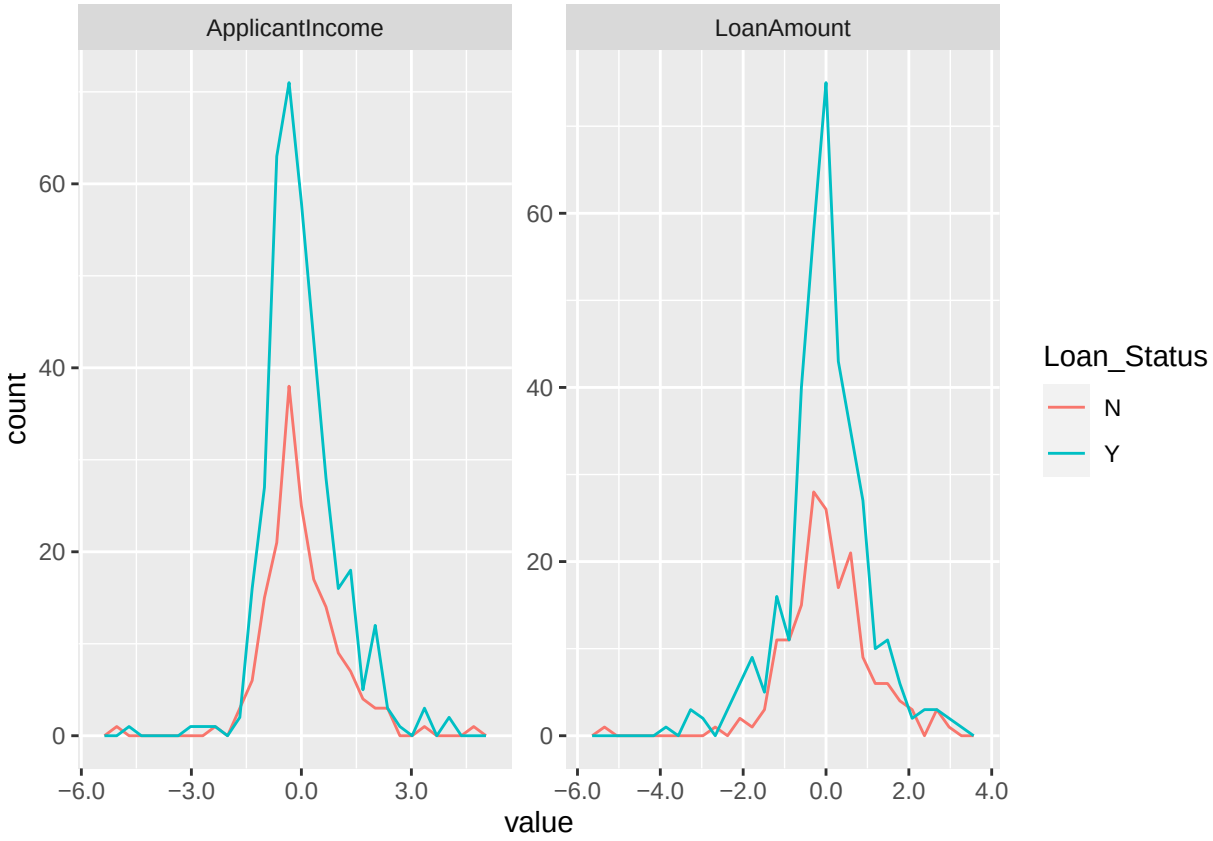
```
loans_na <- drop_na(
  loans,
  ApplicantIncome, LoanAmount, Credit_History, Married, Property_Area, Education
  )

training <- slice_sample(loans_na, prop = 0.70)
remaining <- anti_join(loans_na, training, by = "Loan_ID")

validation <- slice_sample(remaining, prop = 0.5)
test <- anti_join(remaining, validation, by = "Loan_ID")
```

Section 2: Linear Discriminant Analysis

Our first goal is to create a model using Linear Discriminant Analysis (LDA). In order to achieve this, we will need to assume a Gaussian distribution across predictors. This implicitly guides us towards the continuous variables that showed skewed distributions during EDA.

As such, we will conduct some transformations on these variables to help us achieve a (closer) to Gaussian distribution for these continuous variables. The variable `CoapplicantIncome` was not included in this work, due to the untidy aspects of its recording - which would only become exacerbated as the distribution becomes bimodal when employing `log` transformations. As discussed, the categorical variable `LoanAmount` will not be used in this scenario either. The new distributions can be seen below - while imperfect, they are far closer to a normal/Gaussian distribution than before.

```
lda_cont <- c("ApplicantIncome", "LoanAmount")

loans_na %>%
  mutate(across(lda_cont, ~ scale(log(.))[ , 1])
    ) %>%
  pivot_longer(lda_cont) %>%
  show_dist()
```



Based on the above, we can create a LDA-based model as below:

```
lda_mod <- MASS::lda(
  Loan_Status ~ log(ApplicantIncome) + log(LoanAmount), data = training
  )
```

Due to the unbalanced target classes, the LDA model above would always classify our data as `Loan_Status = "Y"`, which is not remotely useful. As such, through experimentation (for lack of domain knowledge), a threshold was settled upon to help established more reliable prediction. In this way, any prediction where the posterior probability was greater than to 65% for an approval was classed as such.

```
validation_preds <- predict(lda_mod, newdata = validation)
test_preds <- predict(lda_mod, newdata = test)
threshold <- 0.65

lda_validation <- validation %>%
  mutate(
    Predicted = if_else(validation_preds$posterior[, "Y"] > threshold, "Y", "N")
  )

lda_test <- test %>%
  mutate(
    Predicted = if_else(test_preds$posterior[, "Y"] > threshold, "Y", "N")
  )
```

Even so, we can see that the performance is not spectacular using a confusion matrix like the one below. In terms of accuracy, the model only achieves about 57% - slightly better than the null model.

```
conf_mat <- function(df) {
  table("expected" = df$Loan_Status, "predicted" = df$Predicted)
}

conf_mat(lda_validation)
```

```
##          predicted
## expected  N  Y
##        N  4 20
##        Y 15 42
```

This is unsurprising - while we were able to create *more Gaussian* distributions of the variables with some pre-processing, they were never fully Gaussian and thus one of the main assumptions is suspect. While we glossed over this detail, the continuous variables could not be linearly separated, so any boundaries created by LDA are somewhat suspect - we did not anticipate this model to be fruitful for this data set.

Section 3: K-nearest Neighbors (KNN)

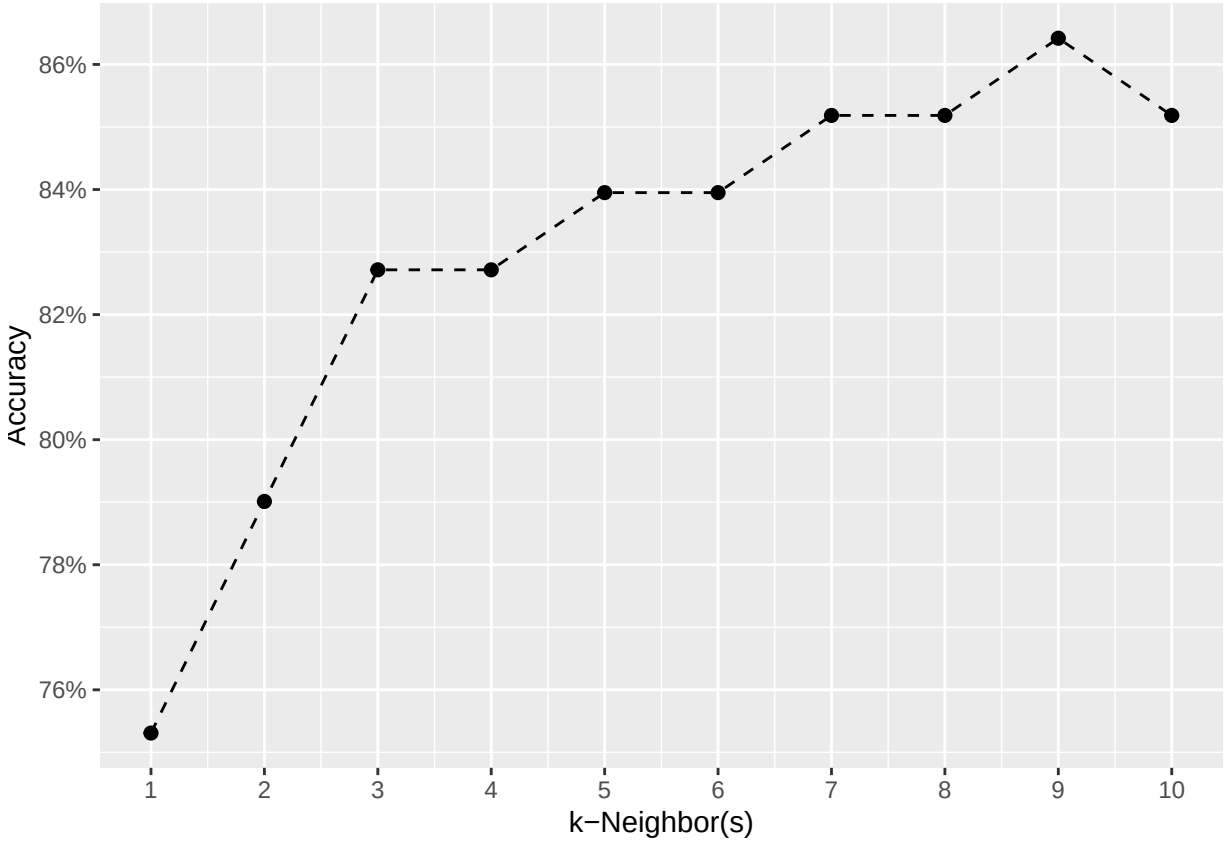
Utilizing KNN requires a different setup in how the data is provided to `class::KNN` than we saw with `MASS::lda`. Recycling the training, validation and test split from before we simply modify the structure to facilitate using our next method. Additionally - the continuous predictors are scaled and centered in order to avoid differences of scale causing any of them to dominate the results.

```
knn_data <- map(
  list("train" = training, "val" = validation, "test" = test),
  ~ .x %>%
    select(all_of(
      c("ApplicantIncome", "LoanAmount", "Credit_History", "Loan_Status")
      )) %>%
    mutate(across(c("ApplicantIncome", "LoanAmount"), ~ scale(.)[ ,1]))
)
```

We'll want to land on a value for k and doing so is much easier by mapping our function and visualizing the results.

```
do_knn <- function(test_set, k_val) {
  class::knn(
    train = select(knn_data[["train"]], -Loan_Status),
    cl = knn_data[["train"]]$Loan_Status,
    test = select(test_set, -Loan_Status),
    k = k_val
  )
}

neighbors <- seq(1, 10)

val_scores <- map_dbl(
  neighbors,
  ~ mean(do_knn(knn_data[["val"]], .x) == knn_data[["val"]]$Loan_Status)
  )
```

```
tibble(k = neighbors, accuracy = val_scores) %>%
  ggplot(aes(k, accuracy)) +
  geom_point(size = 2) +
  geom_line(linetype = 2) +
  labs(x = "k-Neighbor(s)", y = "Accuracy") +
  scale_x_continuous(breaks = neighbors) +
  scale_y_continuous(n.breaks = 8, labels = scales::percent_format(1))
```



Based on this output, it appears that the best choice as far this KNN model goes is to utilize $k = 5$. While the accuracy gain from $k = 4$ to $k = 5$ is not massive, this should provide decent performance. Doing so with the test data set achieves an accuracy of about 84% on the validation data set.

```
knn_validation <- validation %>% mutate(
  Predicted = do_knn(knn_data[["val"]], 5)
  )

knn_test <- test %>% mutate(
  Predicted = do_knn(knn_data[["test"]], 5)
)

conf_mat(knn_validation)
```

```
##          predicted
## expected  N  Y
##        N 13 11
##        Y  2 55
```

Looking at the confusion matrix, this model has a moderate tendency to falsely approve applications. So while we may have a nice accuracy score, this alone may make the model unappealing in the realm of loan approvals.

Section 4: Decision trees

Utilizing the same training/validation data sets as before, in this section we will fit a decision tree and, subsequently, a random forest model. We'll begin by loading the `caret` library and fitting a tree-based model using the CART method provided by the `rpart` library.

```
library(caret)
library(rpart)

dt <- train(
  Loan_Status ~ Married + Property_Area + Credit_History + Education,
  data = training,
  method = "rpart"
  )

dt
```

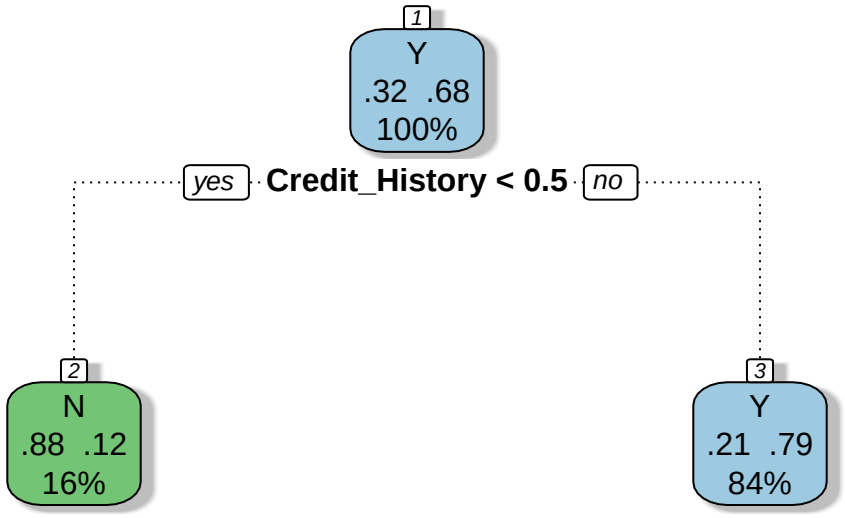
```
## CART
##
## 378 samples
##   4 predictor
##   2 classes: 'N', 'Y'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 378, 378, 378, 378, 378, 378, ...
## Resampling results across tuning parameters:
##
```

```
##   cp          Accuracy   Kappa
##   0.0000000  0.7905741  0.4532298
##   0.1900826  0.8079579  0.4841196
##   0.3801653  0.7360069  0.2118370
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.1900826.
```

As you see the optimal complexity parameter value used is about 0.19, leading to an accuracy of just about 80%. This is comparable to the KNN model we saw earlier and certainly an improvement upon the LDA model.

One of the benefits of the Decision Tree, however, is that it is a non-parameteric model that uses simple, branched decision rules to optimize classification. This makes our own interpreting the model, and communicating how it works to non-technical parties, highly intuitive.

```
rattle::fancyRpartPlot(dt$finalModel, sub = "")
```



As we had seen during EDA, `Credit_History` looks to play an important role in which applicants are rejected/approved for their loan. In running through the above process, our Decision Tree uses it alone to classify applicants. This is both advantageous for its predictive ability as much as its simplicity.

Now let's review the confusion matrix for the decision tree.

```
dt_validation <- validation %>%
  mutate(Predicted = predict(dt, newdata = validation))
```

```
dt_test <- test %>%
  mutate(Predicted = predict(dt, newdata = test))

conf_mat(dt_validation)
```

```
##          predicted
## expected  N  Y
##        N 14 10
##        Y  0 57
```

As you see from the matrix above, just more than half of expected rejections were classified as approvals. On the other hand, all approvals were correctly classified. This is very comparable in its performance to the earlier KNN model, though with an improvement in accuracy (84% vs 87%).

Section 5: Random forests

Finally, we get to the Random Forest section of the assignment. This is an ensemble method that combines multiple Decision Trees to optimize classification. While all predictors are at play initially, their utilization is randomized and so we leverage the power of multiple weak learners to achieve better predictions.

To do this, we'll load the `randomForest` library and fit a model using `caret` again.

```
library(randomForest)

rf <- train(
  Loan_Status ~ Married + Property_Area + Credit_History + Education,
  data = training,
  method = 'rf'
  )

rf
```

```
## Random Forest
##
## 378 samples
##   4 predictor
##   2 classes: 'N', 'Y'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 378, 378, 378, 378, 378, 378, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.8014852  0.4698364
##   3     0.7932920  0.4508483
##   5     0.7863187  0.4393751
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

Given the above output from the model, we can conclude that the training accuracy is about 79% - just slightly below the earlier KNN and Decision Tree models. Let's review the confusion matrix against the validation data for this model:

```
rf_validation <- validation %>%
  mutate(Predicted = predict(rf, newdata = validation))

rf_test <- test %>%
  mutate(Predicted = predict(rf, newdata = test))

conf_mat(rf_validation)
```

```
##          predicted
## expected  N  Y
##        N 14 10
##        Y  0 57
```

We again have a similar scenario as before: no false predictions for rejections, but we do have false approvals at a comparable degree to the earlier three models.

Section 6: Model performance

Based on the validation matrices and accuracy scores, we are essentially left with a decision between the KNN, Decision Tree and Random Forest models. Given the predictive ability of `Credit_History` on its own, both the Decision Tree and Random Forest models appear to end up in the same place in terms of accuracy and performance. The major difference being, however, the overall simplicity of the Decision Tree versus Random Forest. Additionally, the Random Forest can be computationally expensive - though perhaps in the case of this particular data set and chosen predictors that may not be an issue.

Comparing the confusion matrices for both the KNN and Decision Tree models below, we see the end result of our modeling efforts.

```
conf_mat(knn_test)
```

```
##          predicted
## expected  N  Y
##        N 10 14
##        Y  4 54
```

```
conf_mat(dt_test)
```

```
##          predicted
## expected  N  Y
##        N 11 13
##        Y  0 58
```

Both models have false approvals at similar rates as they did during their assessment against the validation data. It seems we cannot avoid this imperfection with the models we've developed here, however, we can see that the accuracy for the Decision Tree model lands at about 84% versus the KNN model's 78%. Additionally, the Decision Tree can be demonstrated and explained to the wider organization without necessitating their coming to terms with feature scaling and clustering. For these reasons, we've selected the Decision Tree model as our final choice.