Stout_DA_case_study

Yunting Chiu

10/27/2021

Install the packages

```
library(tidyverse) # tidy data
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.6
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(corrplot) # visualize correlation
```

```
## corrplot 0.84 loaded
```

```
library(tidymodels) # data modeling
```

```
## Registered S3 method overwritten by 'tune':
##   method                    from
##   required_pkgs.model_spec parsnip
```

```
## -- Attaching packages ---------------------------------------- tidymodels 0.1.3 --
```

```
## v broom        0.7.6      v rsample      0.0.9
## v dials        0.0.9      v tune         0.1.5
## v infer        0.5.4      v workflows    0.2.2
## v modeldata    0.1.0      v workflowsets 0.0.2
## v parsnip      0.1.6      v yardstick    0.0.8
## v recipes      0.1.16
```

```
## -- Conflicts ------------------------------------------ tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## * Use tidymodels_prefer() to resolve common conflicts.
```

```
library(leaps) # model selections
library(DataExplorer) # EDA
```

```
library(tidymodels) # data modeling
library(performance)
```

```
##
## Attaching package: 'performance'
```

```
## The following objects are masked from 'package:yardstick':
##
##     mae, rmse
```

```
library(vip) # variable importance plot
```

```
##
## Attaching package: 'vip'
```

```
## The following object is masked from 'package:utils':
##
##     vi
```

```
library(yardstick) #rmse
```

Case Study 1

Read the data

- Below is a data set that represents thousands of loans made through the Lending Club platform, which is a platform that allows individuals to lend to other individuals.
  We would like you to perform the following using the language of your choice:
- Describe the dataset and any issues with it.

See the first six observations of the data set.

```
loans <- read_csv("./data/loans_full_schema.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   emp_title = col_character(),
##   state = col_character(),
##   homeownership = col_character(),
##   verified_income = col_character(),
##   verification_income_joint = col_character(),
##   loan_purpose = col_character(),
##   application_type = col_character(),
##   grade = col_character(),
##   sub_grade = col_character(),
##   issue_month = col_character(),
##   loan_status = col_character(),
##   initial_listing_status = col_character(),
##   disbursement_method = col_character()
## )
## i Use `spec()` for the full column specifications.
```

```
head(loans)
```

```
## # A tibble: 6 x 55
```

```
##   emp_title         emp_length state homeownership annual_income verified_income
##   <chr>                  <dbl> <chr> <chr>                 <dbl> <chr>
## 1 global config en~          3 NJ    MORTGAGE              90000 Verified
## 2 warehouse office~         10 HI    RENT                  40000 Not Verified
## 3 assembly                   3 WI    RENT                  40000 Source Verified
## 4 customer service           1 PA    RENT                  30000 Not Verified
## 5 security supervi~         10 CA    RENT                  35000 Verified
## 6 <NA>                      NA KY    OWN                   34000 Not Verified
## # ... with 49 more variables: debt_to_income <dbl>, annual_income_joint <dbl>,
## #   verification_income_joint <chr>, debt_to_income_joint <dbl>,
## #   delinq_2y <dbl>, months_since_last_delinq <dbl>,
## #   earliest_credit_line <dbl>, inquiries_last_12m <dbl>,
## #   total_credit_lines <dbl>, open_credit_lines <dbl>,
## #   total_credit_limit <dbl>, total_credit_utilized <dbl>,
## #   num_collections_last_12m <dbl>, num_historical_failed_to_pay <dbl>,
## #   months_since_90d_late <dbl>, current_accounts_delinq <dbl>,
## #   total_collection_amount_ever <dbl>, current_installment_accounts <dbl>,
## #   accounts_opened_24m <dbl>, months_since_last_credit_inquiry <dbl>,
## #   num_satisfactory_accounts <dbl>, num_accounts_120d_past_due <dbl>,
## #   num_accounts_30d_past_due <dbl>, num_active_debit_accounts <dbl>,
## #   total_debit_limit <dbl>, num_total_cc_accounts <dbl>,
## #   num_open_cc_accounts <dbl>, num_cc_carrying_balance <dbl>,
## #   num_mort_accounts <dbl>, account_never_delinq_percent <dbl>,
## #   tax_liens <dbl>, public_record_bankrupt <dbl>, loan_purpose <chr>,
## #   application_type <chr>, loan_amount <dbl>, term <dbl>, interest_rate <dbl>,
## #   installment <dbl>, grade <chr>, sub_grade <chr>, issue_month <chr>,
## #   loan_status <chr>, initial_listing_status <chr>, disbursement_method <chr>,
## #   balance <dbl>, paid_total <dbl>, paid_principal <dbl>, paid_interest <dbl>,
## #   paid_late_fees <dbl>
```

Exploratory Data Analysis

We reconfirm that the data frame has 10,000 observations and 55 variables based on the data description.

```
dim(loans)
```

```
## [1] 10000    55
```

Before we build models, we should consider how to deal with variables that have a large number of NAs. If we directly reduce the observations that include NA, we can see that the number of observations drops from 10000 to 201, meaning that we may have lost information in the data. We know that our target variable is `interest_rate`, so we should manually check whether theses NAs variables are relevant to the target variable(y). Also, if the variables have a large number of NAs, meaning that it is not good predictors. We should remove it.

We can see there are some variables that include NAs:

- `emp_title`: 833 - not relevant to y, remove it
- `emp_length`: 817 NAs - keep it
- `debt_to_income`: 24 NAs - keep it
- `annual_income_joint`: 8505 NAs - too many NAs remove it
- `verification_income_joint`: 8545 NAs - too many NAs remove it

- `debt_to_income_joint`: 8505 NAs - too many NAs remove it
- `months_since_last_delinq`: 5658 NAs - too many NAs remove it
- `months_since_90d_late`: 7715 NAs - too many NAs remove it
- `months_since_last_credit_inquiry`: 1271 NAs - too many NAs remove it
- `num_accounts_120d_past_due`: 318 NAs - keep it

```
loans %>%
summarise_all(funs(sum(is.na(.))))
```

```
## Warning: `funs()` was deprecated in dplyr 0.8.0.
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
```

```
## # A tibble: 1 x 55
##   emp_title emp_length state homeownership annual_income verified_income
##       <int>      <int> <int>         <int>         <int>           <int>
## 1       833        817     0             0             0               0
## # ... with 49 more variables: debt_to_income <int>, annual_income_joint <int>,
## #   verification_income_joint <int>, debt_to_income_joint <int>,
## #   delinq_2y <int>, months_since_last_delinq <int>,
## #   earliest_credit_line <int>, inquiries_last_12m <int>,
## #   total_credit_lines <int>, open_credit_lines <int>,
## #   total_credit_limit <int>, total_credit_utilized <int>,
## #   num_collections_last_12m <int>, num_historical_failed_to_pay <int>,
## #   months_since_90d_late <int>, current_accounts_delinq <int>,
## #   total_collection_amount_ever <int>, current_installment_accounts <int>,
## #   accounts_opened_24m <int>, months_since_last_credit_inquiry <int>,
## #   num_satisfactory_accounts <int>, num_accounts_120d_past_due <int>,
## #   num_accounts_30d_past_due <int>, num_active_debit_accounts <int>,
## #   total_debit_limit <int>, num_total_cc_accounts <int>,
## #   num_open_cc_accounts <int>, num_cc_carrying_balance <int>,
## #   num_mort_accounts <int>, account_never_delinq_percent <int>,
## #   tax_liens <int>, public_record_bankrupt <int>, loan_purpose <int>,
## #   application_type <int>, loan_amount <int>, term <int>, interest_rate <int>,
## #   installment <int>, grade <int>, sub_grade <int>, issue_month <int>,
## #   loan_status <int>, initial_listing_status <int>, disbursement_method <int>,
## #   balance <int>, paid_total <int>, paid_principal <int>, paid_interest <int>,
## #   paid_late_fees <int>
```

```
# original data
loans %>%
  nrow()
```

```
## [1] 10000
```

```
# droped NAs
loans %>%
  drop_na() %>%
  nrow()
```

```
## [1] 201
```

Our data now has no NA, and it still has large sample sizes (8886 observations), which is good.

```
# remove some variables that include large number of NA
loans %>%
  select(-emp_title, -annual_income_joint, -verification_income_joint, -debt_to_income_joint,
         -months_since_last_delinq, -months_since_90d_late, -months_since_last_credit_inquiry) %>%
  drop_na() -> loansNONAs

loansNONAs %>%
summarise_all(funs(sum(is.na(.))))
```

```
## # A tibble: 1 x 48
##   emp_length state homeownership annual_income verified_income debt_to_income
##        <int> <int>         <int>         <int>           <int>          <int>
## 1          0     0             0             0               0              0
## # ... with 42 more variables: delinq_2y <int>, earliest_credit_line <int>,
## #   inquiries_last_12m <int>, total_credit_lines <int>,
## #   open_credit_lines <int>, total_credit_limit <int>,
## #   total_credit_utilized <int>, num_collections_last_12m <int>,
## #   num_historical_failed_to_pay <int>, current_accounts_delinq <int>,
## #   total_collection_amount_ever <int>, current_installment_accounts <int>,
## #   accounts_opened_24m <int>, num_satisfactory_accounts <int>,
## #   num_accounts_120d_past_due <int>, num_accounts_30d_past_due <int>,
## #   num_active_debit_accounts <int>, total_debit_limit <int>,
## #   num_total_cc_accounts <int>, num_open_cc_accounts <int>,
## #   num_cc_carrying_balance <int>, num_mort_accounts <int>,
## #   account_never_delinq_percent <int>, tax_liens <int>,
## #   public_record_bankrupt <int>, loan_purpose <int>, application_type <int>,
## #   loan_amount <int>, term <int>, interest_rate <int>, installment <int>,
## #   grade <int>, sub_grade <int>, issue_month <int>, loan_status <int>,
## #   initial_listing_status <int>, disbursement_method <int>, balance <int>,
## #   paid_total <int>, paid_principal <int>, paid_interest <int>,
## #   paid_late_fees <int>
```

```
loansNONAs %>%
  nrow()
```

```
## [1] 8886
```

We can also check which numerical variables have a higher collinearity with the target variable `interest rate`. The variable with high collinearity means it is not a good predictors in a linear task.

```
loansNONAs %>%
  select_if(is.numeric) -> num_loans
reg <- lm(interest_rate ~., data = num_loans)
check_collinearity(reg)
```

```
## Warning: Model matrix is rank deficient. VIFs may not be sensible.
```

```
## # Check for Multicollinearity
```

```
##
## Low Correlation
##
##                          Term  VIF Increased SE Tolerance
##                    emp_length 1.12         1.06      0.89
##                annual_income 1.76         1.33      0.57
##               debt_to_income 1.29         1.14      0.78
##                    delinq_2y 1.34         1.16      0.75
##          earliest_credit_line 1.37         1.17      0.73
##            inquiries_last_12m 1.30         1.14      0.77
##            total_credit_limit 3.22         1.79      0.31
##         total_credit_utilized 2.31         1.52      0.43
##      num_collections_last_12m 1.06         1.03      0.94
##  total_collection_amount_ever 1.04         1.02      0.96
##           accounts_opened_24m 1.82         1.35      0.55
##     num_active_debit_accounts 4.94         2.22      0.20
##             total_debit_limit 2.46         1.57      0.41
##             num_mort_accounts 2.01         1.42      0.50
##  account_never_delinq_percent 1.39         1.18      0.72
##                     tax_liens 4.93         2.22      0.20
##                          term 4.39         2.10      0.23
##
## Moderate Correlation
##
##                          Term  VIF Increased SE Tolerance
##            total_credit_lines 5.92         2.43      0.17
##  num_historical_failed_to_pay 5.15         2.27      0.19
##         num_total_cc_accounts 7.12         2.67      0.14
##       num_cc_carrying_balance 7.00         2.65      0.14
##
## High Correlation
##
##                          Term               VIF Increased SE Tolerance
##             open_credit_lines           2283.36        47.78      0.00
##  current_installment_accounts             33.12         5.75      0.03
##     num_satisfactory_accounts           2132.57        46.18      0.00
##          num_open_cc_accounts             91.57         9.57      0.01
##                   loan_amount           1355.94        36.82      0.00
##                   installment             31.98         5.66      0.03
##                       balance           1241.25        35.23      0.00
##                    paid_total 45529910822538.88   6747585.56      0.00
##                paid_principal 43964423844709.56   6630567.38      0.00
##                 paid_interest   730460454130.95    854669.79      0.00
##                paid_late_fees        8413210.79      2900.55      0.00
```

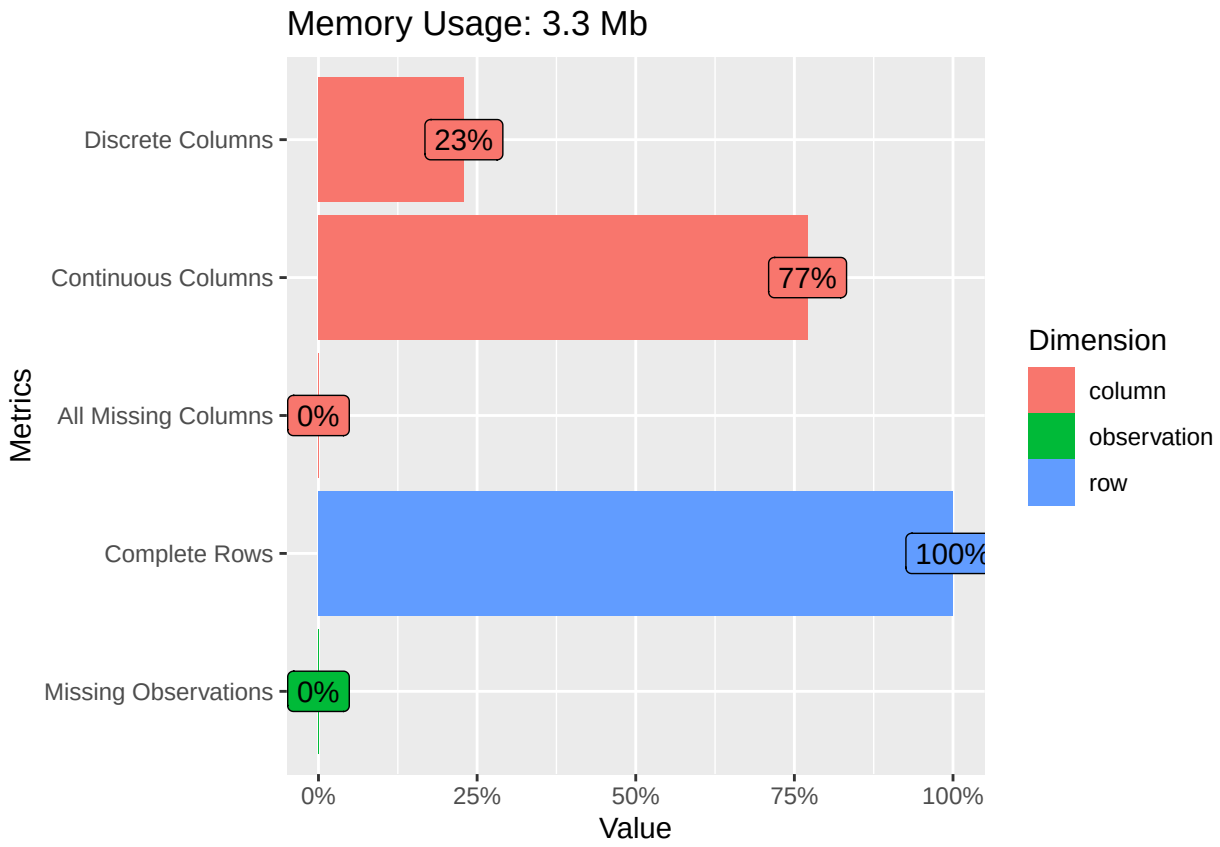
Data Visualization

- Generate a minimum of 5 unique visualizations using the data and write a brief description of your observations. Additionally, all attempts should be made to make the visualizations visually appealing.

Plot 1

The first plot from the analysis confirms the types of variables in the data. There are no missing values in the data now. Also, 23% of the variables are categorical, and 77 % of the variables are numerical.

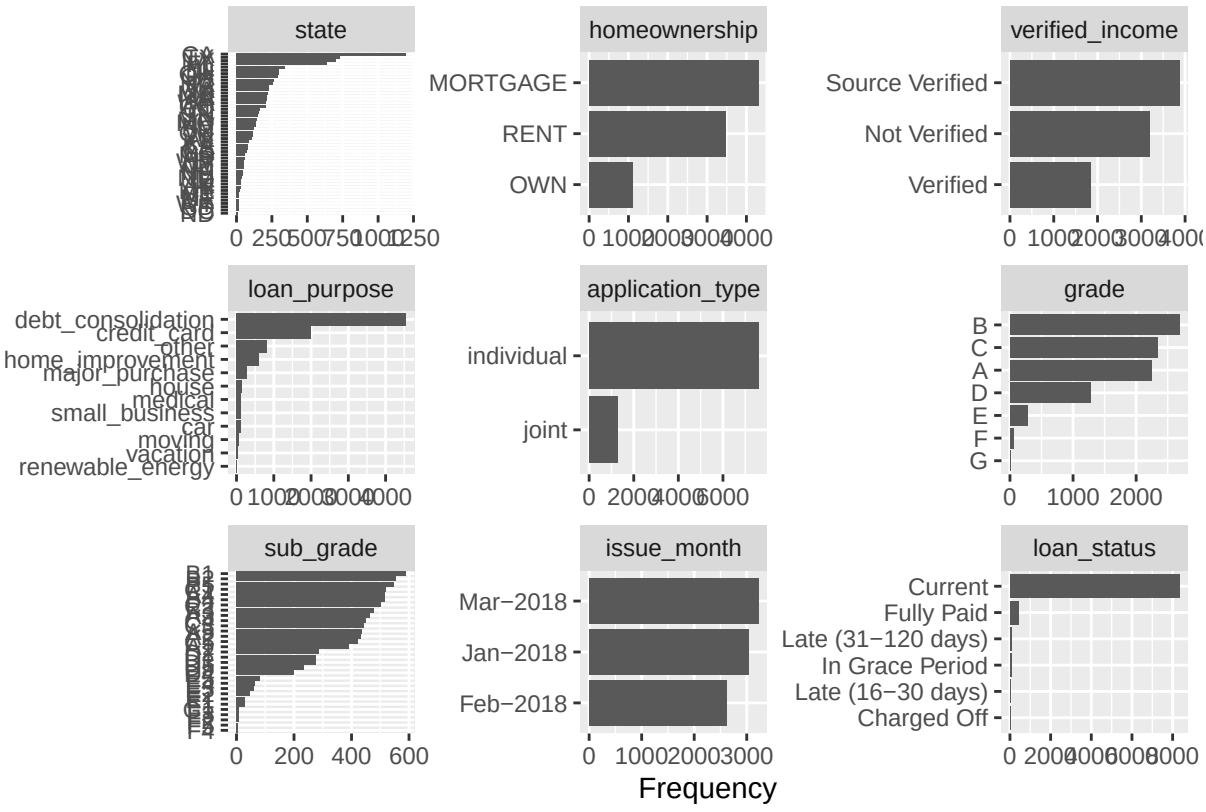
```
plot_intro(loansNONAs)
```

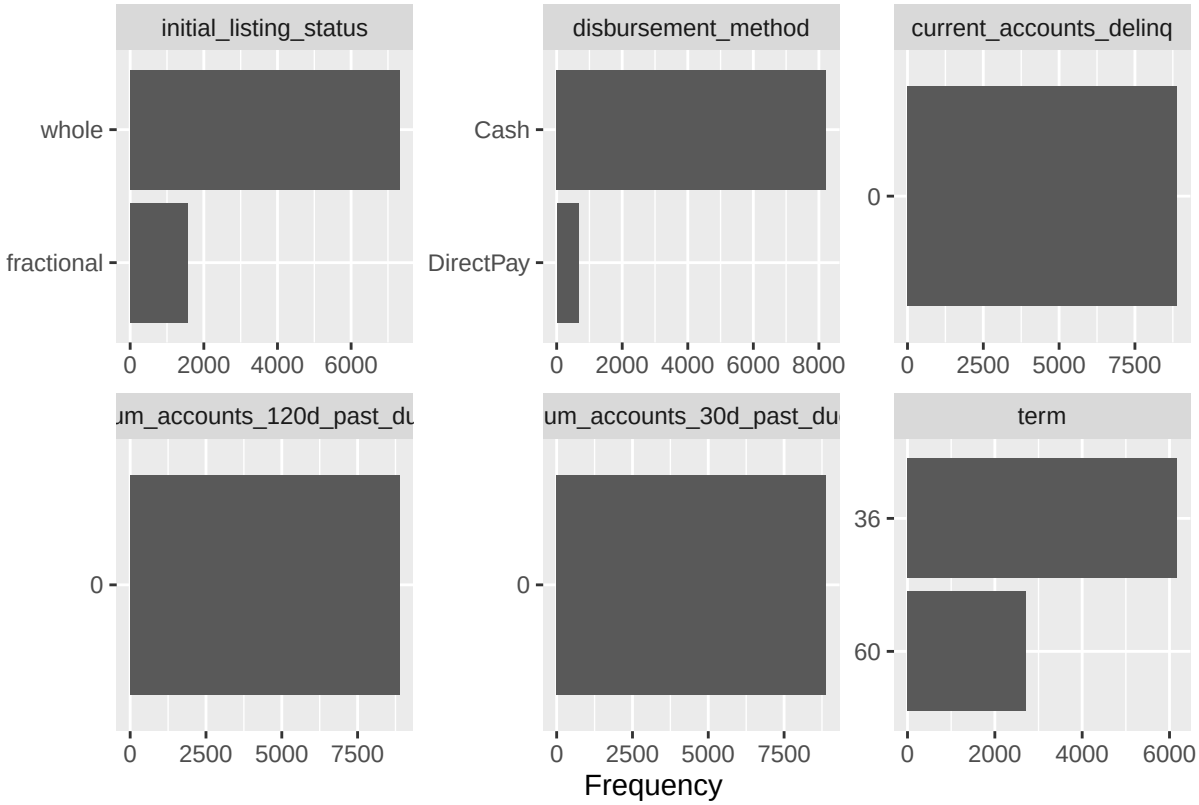


Plot 2

The bar plot shows the categorical variables in the data. According to the bar plot below, we can see some levels seem to have low frequencies, such as `joint` in the `application_type` and `Charged Off` in the `loan_status`.

```
plot_bar(loansNONAs)
```


state
homeownership
MORTGAGE
RENT
OWN
verified_income
Source Verified
Not Verified
Verified
loan_purpose
debt_consolidation
credit_card
other
home_improvement
major_purchase
house
medical
small_business
car
moving
vacation
renewable_energy
application_type
individual
joint
grade
B
C
A
D
E
F
G
sub_grade
issue_month
Mar−2018
Jan−2018
Feb−2018
loan_status
Current
Fully Paid
Late (31−120 days)
In Grace Period
Late (16−30 days)
Charged Off
Frequency


initial_listing_status
whole
fractional
disbursement_method
Cash
DirectPay
current_accounts_delinq
0
um_accounts_120d_past_du
0
um_accounts_30d_past_du
0
term
36
60
Frequency

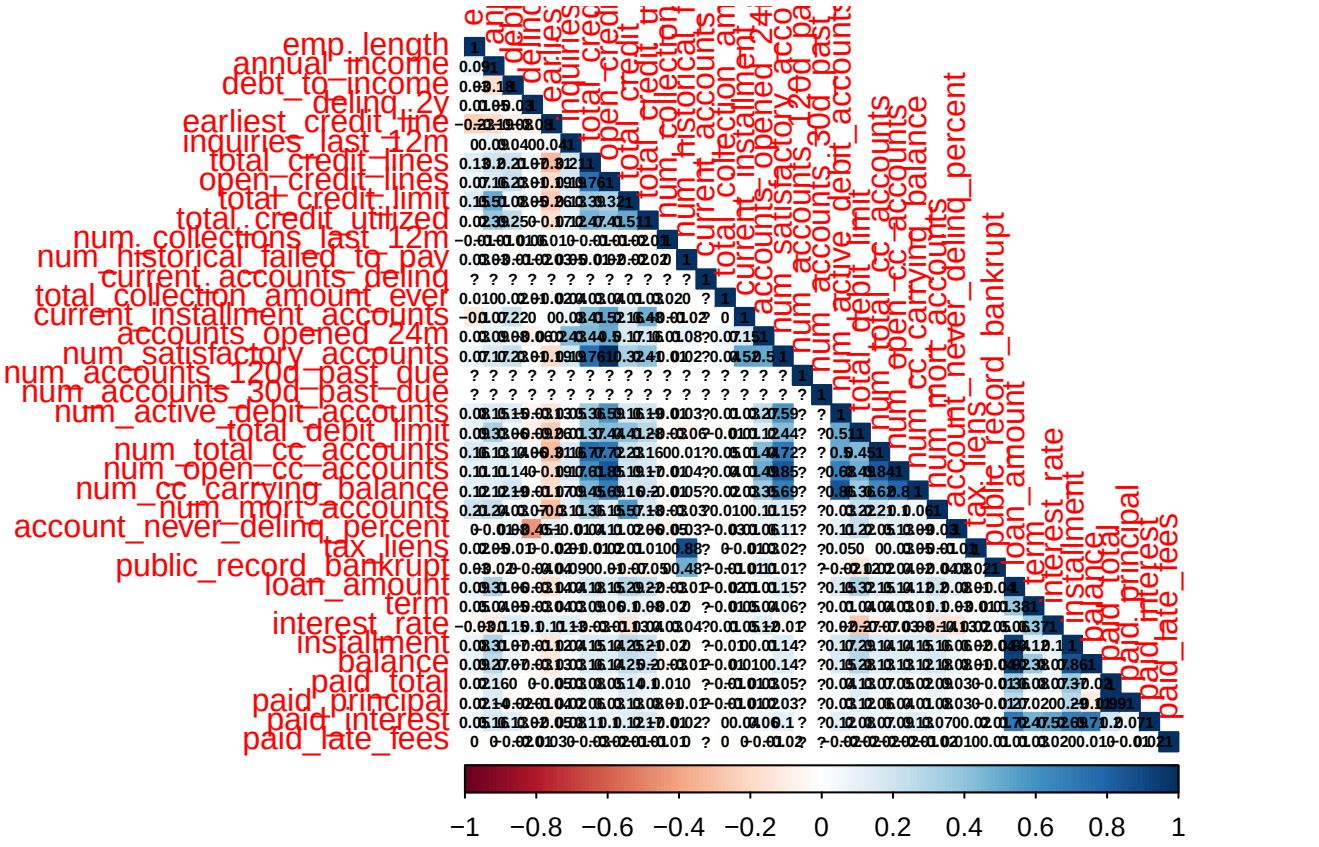
Plot 3

Now we only focus on the numerical features, we can see there are some variables have a high correlation with `interest_rate`, such as `loan_amount`, `balance`. These variables with high correlation might not be good predictors if we consider building linear models.

```
loansNONAs %>%
  select_if(is.numeric) -> num_loans
correlation <- cor(num_loans)
```

```
## Warning in cor(num_loans): the standard deviation is zero
```

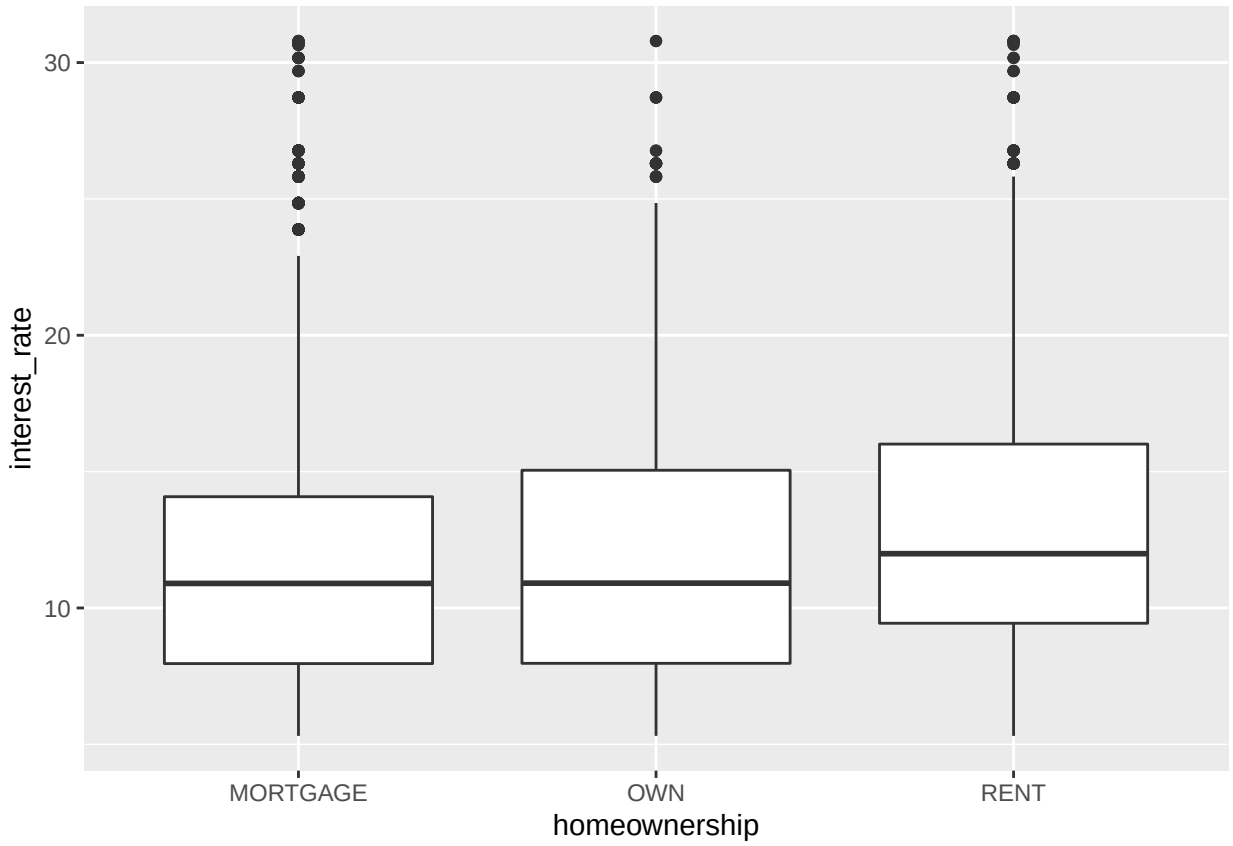
```
corrplot(correlation, method="color", addCoef.col = "black", number.cex = 0.5, type = "lower")
```



Plot 4

We can see the ownership status of the applicant's residence is not the main factor to affect the Interest rate of the loan the applicant received. The median interest rates of these three are nearly identical.

```
loansNONAs %>%
  ggplot(aes(x = homeownership, y = interest_rate)) +
  geom_boxplot()
```

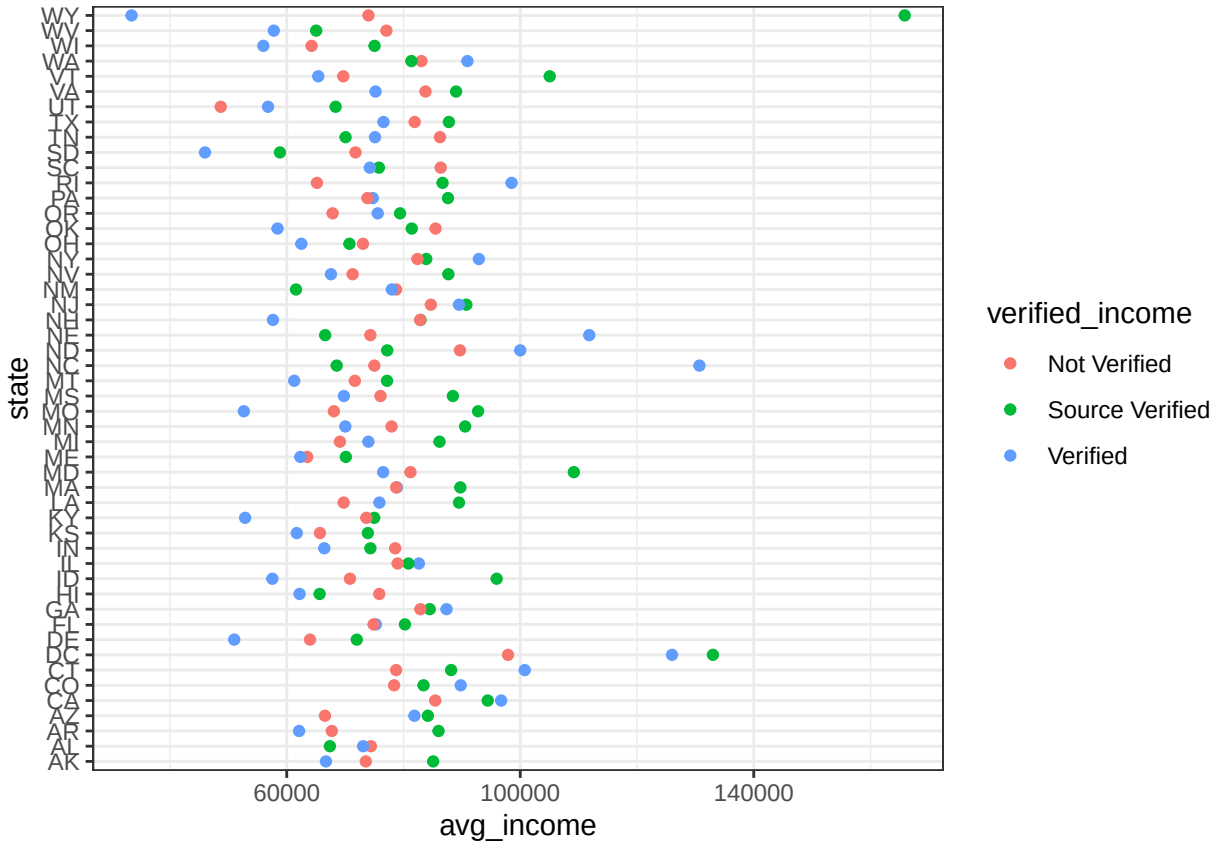


Plot 5

According to the results, compared to other states, DC and NC have the higher verified average income, meaning that despite having higher incomes, these area's people still want to apply for loans.

```
loansNONAs %>%
  #filter(verified_income == "Verified") %>%
  group_by(state, verified_income) %>%
  summarise(avg_income = mean(annual_income)) %>%
  arrange(desc(avg_income)) %>%
  ggplot(aes(x = state, y = avg_income, color = verified_income)) +
  geom_point() +
  coord_flip() +
  theme_bw()
```

```
## `summarise()` has grouped output by 'state'. You can override using the `.groups` argument.
```



Feature Selection

To make things easier, we're only focusing on the numerical features right now. In other words, there are 32 features and one response variable. Now, we'll use random forest to identify which are the most important variables in terms of `interest_rate`. The R squared (OOB) is 0.8, meaning that the algorithm performs not bad. The important score of the first five variables is greater than 10000. We will use the first five important features as predictors. That is, we will choose `paid_interest`, `paid_principal`, `paid_total`, `total_debit_limit`, and `term` as our independent variables.

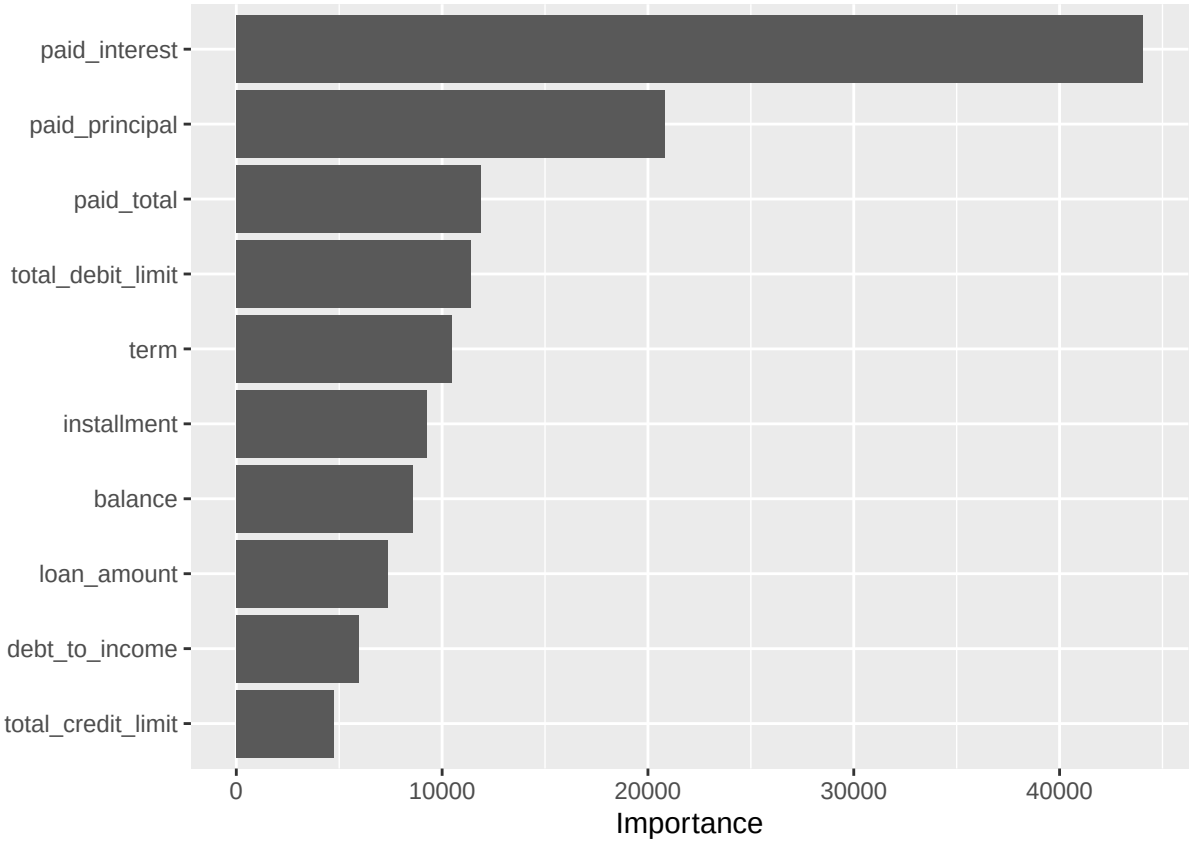
```
loans_train_split <- initial_split(num_loans, prop = 0.8)
set.seed(1234)
loans_train <- training(loans_train_split)
loans_test <- testing(loans_train_split)

rand_forest_ranger_spec <-
  rand_forest() %>%
  set_engine('ranger', importance = "impurity") %>%
  set_mode('regression')
rand_forest_ranger_spec
```

```
## Random Forest Model Specification (regression)
##
## Engine-Specific Arguments:
##   importance = impurity
##
## Computational engine: ranger
```

```
rf_fit <- fit(rand_forest_ranger_spec, interest_rate ~., data = loans_train)
rf_fit
```

```
## parsnip model object
##
## Fit time:  10s
## Ranger result
##
## Call:
##  ranger::ranger(x = maybe_data_frame(x), y = y, importance = ~"impurity",      num.threads = 1, verbo
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      7109
## Number of independent variables:  36
## Mtry:                             6
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       5.201484
## R squared (OOB):                  0.7903031
```

```
vip(rf_fit$fit)
```



Data Modeling

- Create a feature set and create a model which predicts interest rate using at least 2 algorithms. Describe any data cleansing that must be performed and analysis when examining the data.
- Visualize the test results and propose enhancements to the model, what would you do if you had more time. Also describe assumptions you made and your approach.

Firstly, we can consider using the simplest machine learning model: linear regression model. We can see `total_debit_limit`, `term` and `b0` are in the significant level. However, with the low adjusted R-squared value: 0.38%, the model doesn't follow the linear trend. We can give it a shot if we include regularization in the model.

```
# Bad result from the linear model
reg <- lm(interest_rate~ paid_interest+paid_principal+ paid_total+total_debit_limit+term,
          data = num_loans)
summary(reg)
```

```
##
## Call:
## lm(formula = interest_rate ~ paid_interest + paid_principal +
##     paid_total + total_debit_limit + term, data = num_loans)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.3970  -2.8698  -0.6614   2.1786  19.2830
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)        8.151e+00  1.759e-01  46.336   <2e-16 ***
## paid_interest     -4.051e-03  2.348e-02  -0.173    0.863
## paid_principal    -8.596e-03  2.348e-02  -0.366    0.714
## paid_total         8.594e-03  2.348e-02   0.366    0.714
## total_debit_limit -5.897e-05  1.593e-06 -37.014   <2e-16 ***
## term               7.114e-02  4.264e-03  16.685   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.91 on 8880 degrees of freedom
## Multiple R-squared:  0.3883, Adjusted R-squared:  0.3879
## F-statistic:  1127 on 5 and 8880 DF,  p-value: < 2.2e-16
```

We know that the variables `paid_interest` and `paid_principal` have a high correlations with the response variable. But if we add a regularizer, maybe we can improve the linear model. We will start to select the first five important variables above to the L1 model. That is, we will begin by attempting to use lasso regression model.

Model 1: Lasso Regression with the optimal regularization

```
num_loans %>%
  select(interest_rate, paid_interest, paid_principal, paid_total, total_debit_limit, term) -> loans_mo

set.seed(1234)
loans_split <- initial_split(loans_model_df, prop = 0.8)
loans_train <- training(loans_split)
loans_test  <- testing(loans_split)
```

```
lasso_spec <- linear_reg(mixture = 1, penalty = tune()) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

lasso_rec <- recipe(interest_rate ~., data = loans_train) %>%
  step_normalize(all_predictors()) %>%
  step_zv(all_predictors())
```

create a workflow

```
lasso_wf <- workflow() %>%
  add_model(lasso_spec) %>%
  add_recipe(lasso_rec)
```

Cross-Validation create 10-Fold Cross-Validation in the training data set.

```
set.seed(1234)
loans_folds <- vfold_cv(loans_train, strata = interest_rate, v = 10)
loans_folds$splits
```

```
## [[1]]
## <Analysis/Assess/Total>
## <6396/713/7109>
##
## [[2]]
## <Analysis/Assess/Total>
## <6397/712/7109>
##
## [[3]]
## <Analysis/Assess/Total>
## <6397/712/7109>
##
## [[4]]
## <Analysis/Assess/Total>
## <6398/711/7109>
##
## [[5]]
## <Analysis/Assess/Total>
## <6398/711/7109>
##
## [[6]]
## <Analysis/Assess/Total>
## <6399/710/7109>
##
## [[7]]
## <Analysis/Assess/Total>
## <6399/710/7109>
##
## [[8]]
## <Analysis/Assess/Total>
## <6399/710/7109>
##
## [[9]]
## <Analysis/Assess/Total>
```

```
## <6399/710/7109>
##
## [[10]]
## <Analysis/Assess/Total>
## <6399/710/7109>
```

Grid Search Regularly predict the penalty 100 times using regular grids, with the penalty range limited to 0.00001 to 1.

```
penalty_grid <- grid_regular(penalty(range = c(-5, 0)), levels = 100)
penalty_grid
```

```
## # A tibble: 100 x 1
##      penalty
##        <dbl>
##  1 0.00001
##  2 0.0000112
##  3 0.0000126
##  4 0.0000142
##  5 0.0000159
##  6 0.0000179
##  7 0.0000201
##  8 0.0000226
##  9 0.0000254
## 10 0.0000285
## # ... with 90 more rows
```

Tune

```
tune_res <- tune_grid(
  lasso_wf,
  resamples = loans_folds,
  grid = penalty_grid
)
```

Display the each penalty on rmse and rsq, respectively.

```
set.seed(1234)
tune_res %>%
  collect_metrics() %>%
  head()
```

```
## # A tibble: 6 x 7
##     penalty .metric .estimator  mean     n std_err .config
##       <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.00001   rmse    standard   3.90     10 0.0284  Preprocessor1_Model001
## 2 0.00001   rsq     standard   0.389    10 0.00634 Preprocessor1_Model001
## 3 0.0000112 rmse    standard   3.90     10 0.0284  Preprocessor1_Model002
## 4 0.0000112 rsq     standard   0.389    10 0.00634 Preprocessor1_Model002
## 5 0.0000126 rmse    standard   3.90     10 0.0284  Preprocessor1_Model003
## 6 0.0000126 rsq     standard   0.389    10 0.00634 Preprocessor1_Model003
```

Display the best rmse and rsq values of penalty

```
tune_res %>%
  show_best(metric = "rsq") %>%
```

```
  head(1)
```

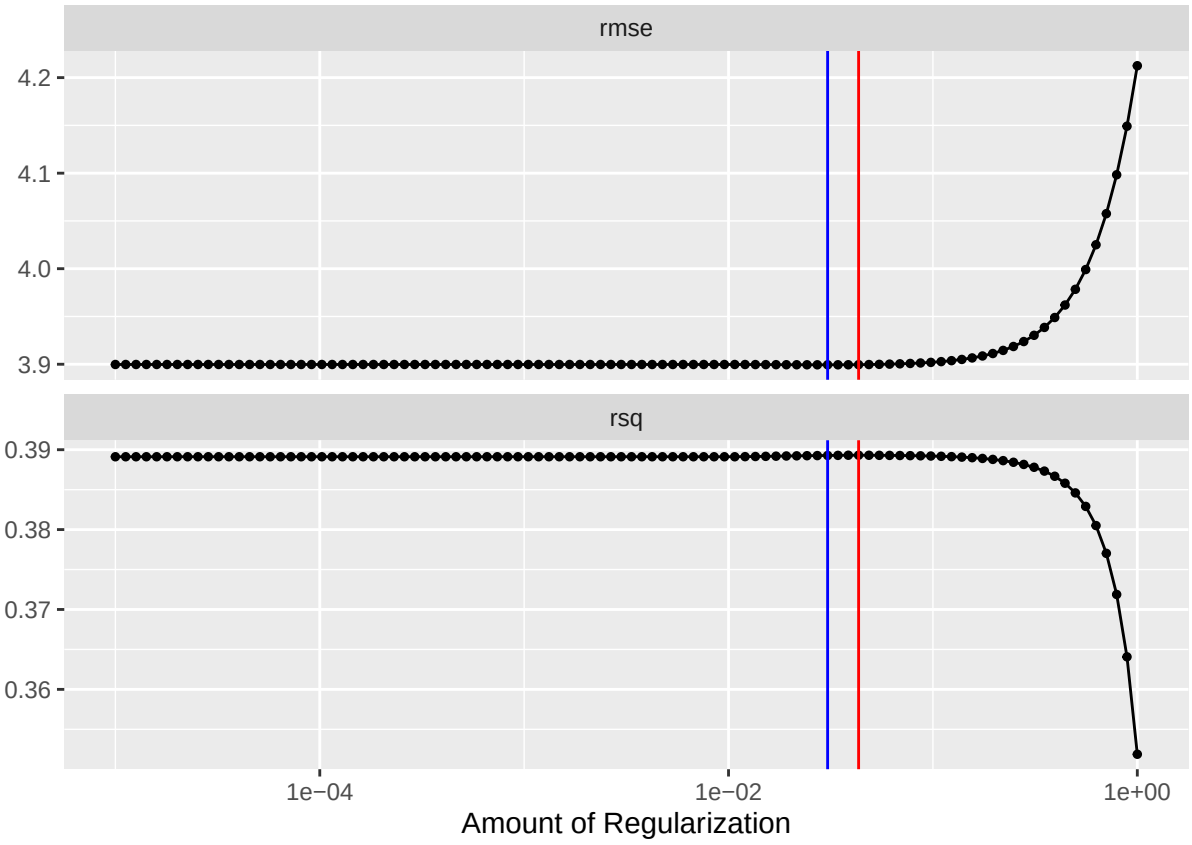
```
## # A tibble: 1 x 7
##   penalty .metric .estimator  mean     n std_err .config
##     <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1  0.0433 rsq     standard   0.389    10 0.00634 Preprocessor1_Model073
```

```
tune_res %>%
  show_best(metric = "rmse") %>%
  head(1)
```

```
## # A tibble: 1 x 7
##   penalty .metric .estimator  mean     n std_err .config
##     <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1  0.0305 rmse    standard    3.90    10  0.0286 Preprocessor1_Model070
```

We can see the best regularization value is 0.043 based on rsq, the the best regularization value is 0.030 based on rmse. We will use the smallest rmse to fit the model.

```
tune_res %>%
  autoplot() +
  geom_vline(xintercept = 0.04328761, color = "red") +
  geom_vline(xintercept = 0.03053856, color = "blue")
```




```
best_rmse <- select_best(tune_res, metric = "rmse")
best_rmse
```

The Best lambda

```
## # A tibble: 1 x 2
##   penalty .config
##     <dbl> <chr>
## 1  0.0305 Preprocessor1_Model070
```

```
lasso_final <- finalize_workflow(lasso_wf, best_rmse)

lasso_final_fit <- fit(lasso_final, data = loans_train)
```

Lasso Performance

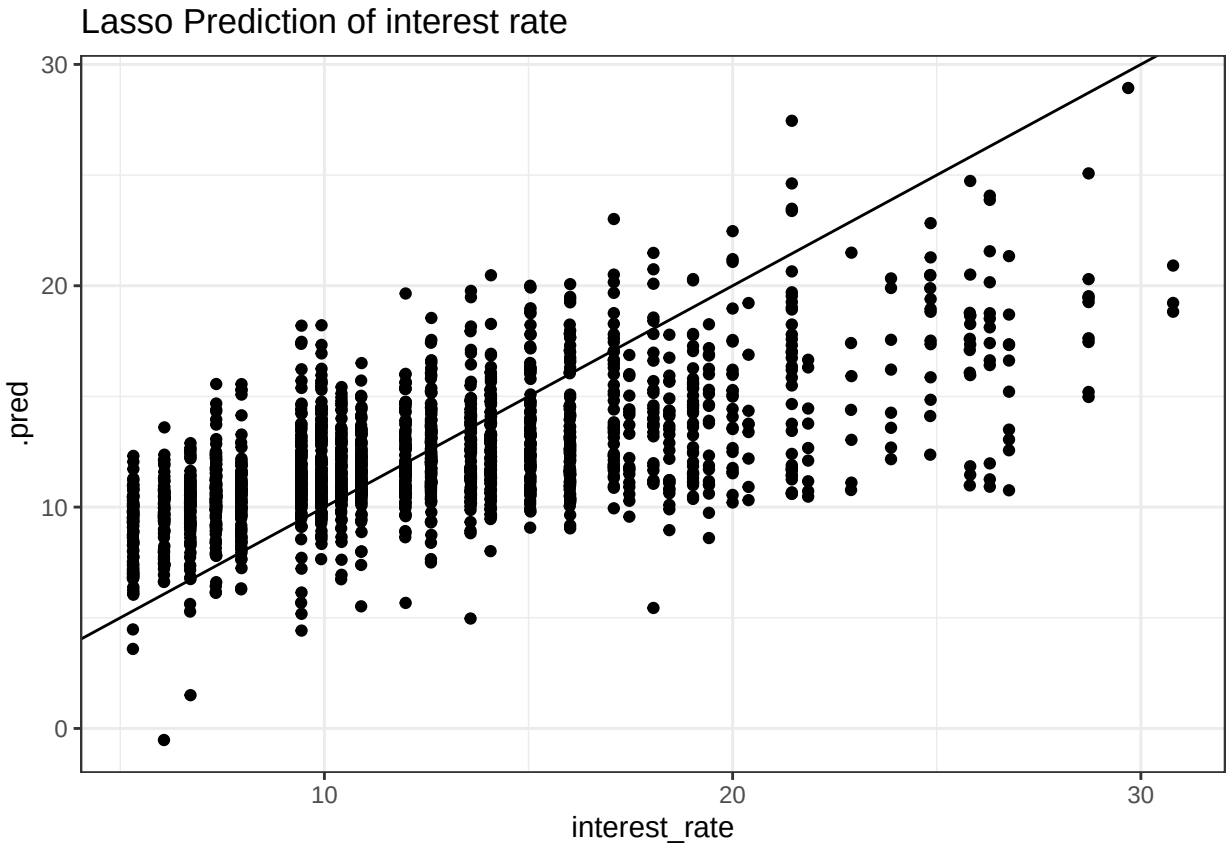
```
augment(lasso_final_fit, new_data = loans_test) %>%
  yardstick::rmse(truth = interest_rate, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        3.96
```

Visualization

The range of `interest_rate` is from 5.31 to 30.94 from the original data set. Based on the plot below, we can conclude that the model does not work well, tuning λ appears to have no obvious benefit in the model. We should try to reduce the $RMSE$.

```
augment(lasso_final_fit, new_data = loans_test) %>%
  ggplot(aes(interest_rate, .pred)) +
  geom_abline(slope = 1, intercept = 0) +
  geom_point() +
  theme_bw() +
  ggtitle("Lasso Prediction of interest rate")
```



The number of observations is greater then the number of features, so that the L1 and L2 regularization methods are not the wise choices.

Model 2: Random Forest with all numerical predictors

Next, we try to add all numerical variables as predictors. That is, 36 variables as X and 1 target variable as y. This time we will use random forest algorithm.

```
num_loans %>%
  #this is our target variable
  select(-interest_rate) %>%
  length()
```

```
## [1] 36
```

Taking 80 % observations as a training set and taking 20 % observations as a testing set.

```
loans_split2 <- initial_split(num_loans, prop = 0.8)
set.seed(1234)
loans_train2 <- training(loans_split2)
loans_test2 <- testing(loans_split2)
```

As usual, we fit the model. In comparison to the lasso regression, we discovered that the rmse of random forest is dramatically reduced from 3.96 to 1.28.

```
set.seed(1234)
# mtry = .cols(): the number of columns in the predictor matrix is used
rd_spec <- rand_forest(mtry = .cols()) %>%
```

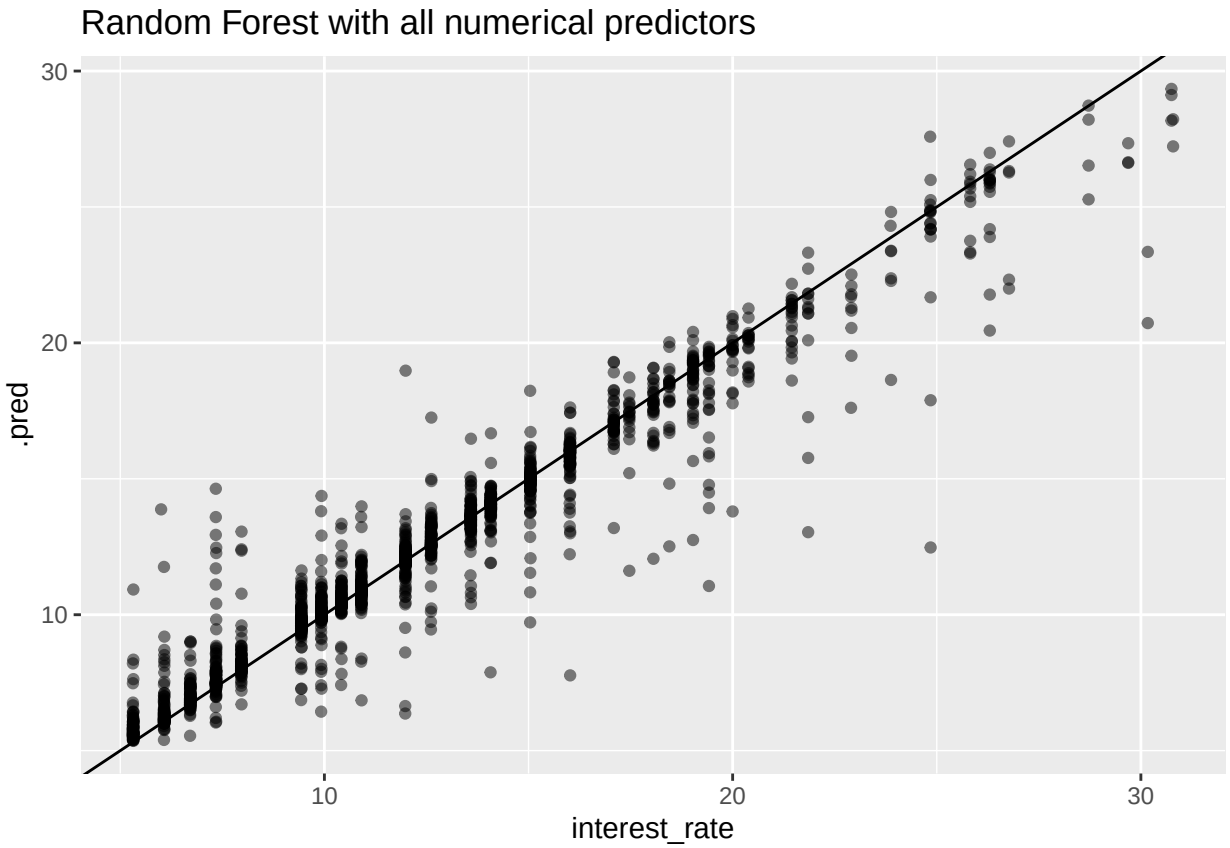
```
  set_engine("randomForest", importance = TRUE) %>%
  set_mode("regression")

rd_fit <- fit(rd_spec, interest_rate ~ ., data = loans_train2)
augment(rd_fit, new_data = loans_test2) %>%
  yardstick::rmse(truth = interest_rate, estimate = .pred)
```

Random Forest Performance

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        1.28
```

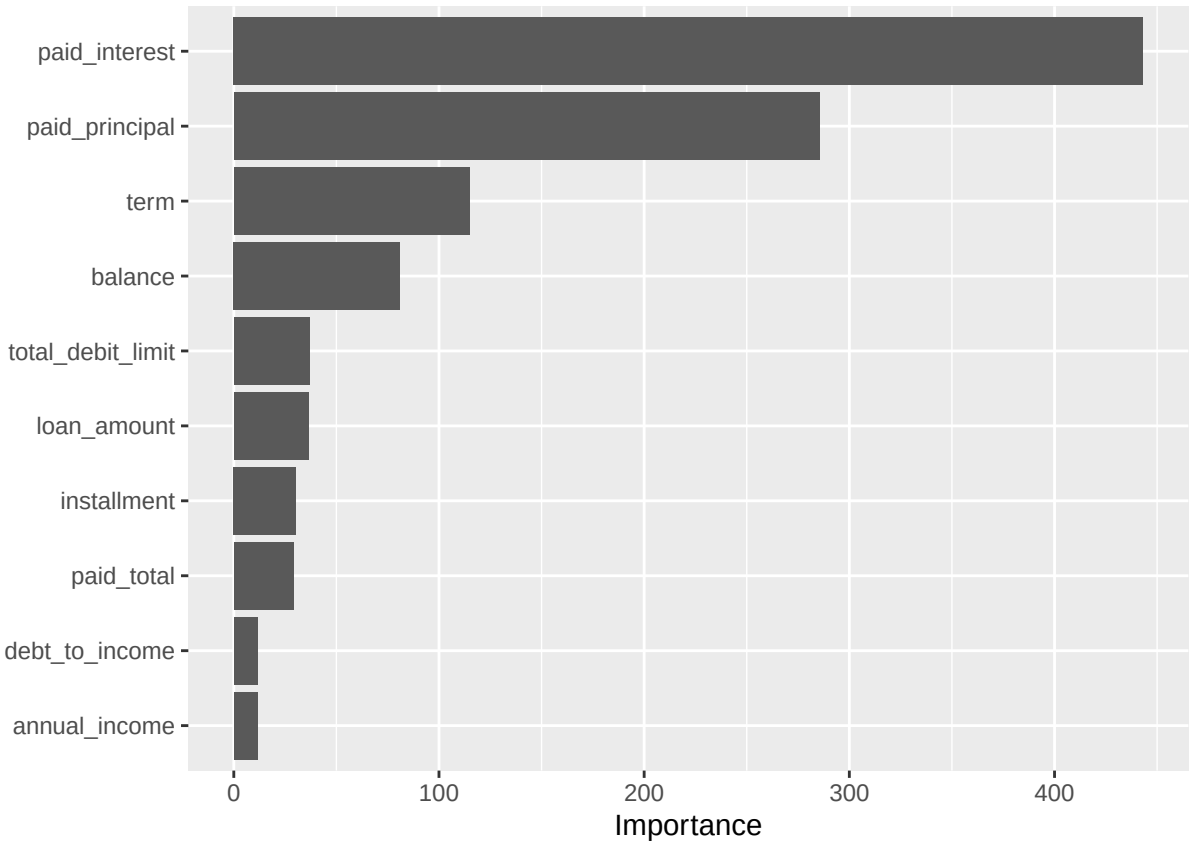
Visualization We can also create a quick scatter plot between the true and predicted value to see if we can make any diagnostics. The predicted values are almost closer to the ground truths.

```
augment(rd_fit, new_data = loans_test2) %>%
  ggplot(aes(interest_rate, .pred)) +
  geom_abline() +
  geom_point(alpha = 0.5) +
  ggtitle("Random Forest with all numerical predictors")
```



Next, let's take a look at the variable importance

```
vip(rd_fit)
```



Model 3: Random Forest with only 3 predictors

Now we only add top three importance variables in the random forest model. In reality, a smaller predictor in ML models can save time for data collection. And we can rapidly run the model and make marketing strategies more easily.

```
num_loans %>%
  select(interest_rate, paid_interest, paid_principal, term) -> loans_threeFeatures

loans_split3 <- initial_split(loans_threeFeatures, prop = 0.8)
set.seed(1234)
loans_train3 <- training(loans_split3)
loans_test3 <- testing(loans_split3)
```

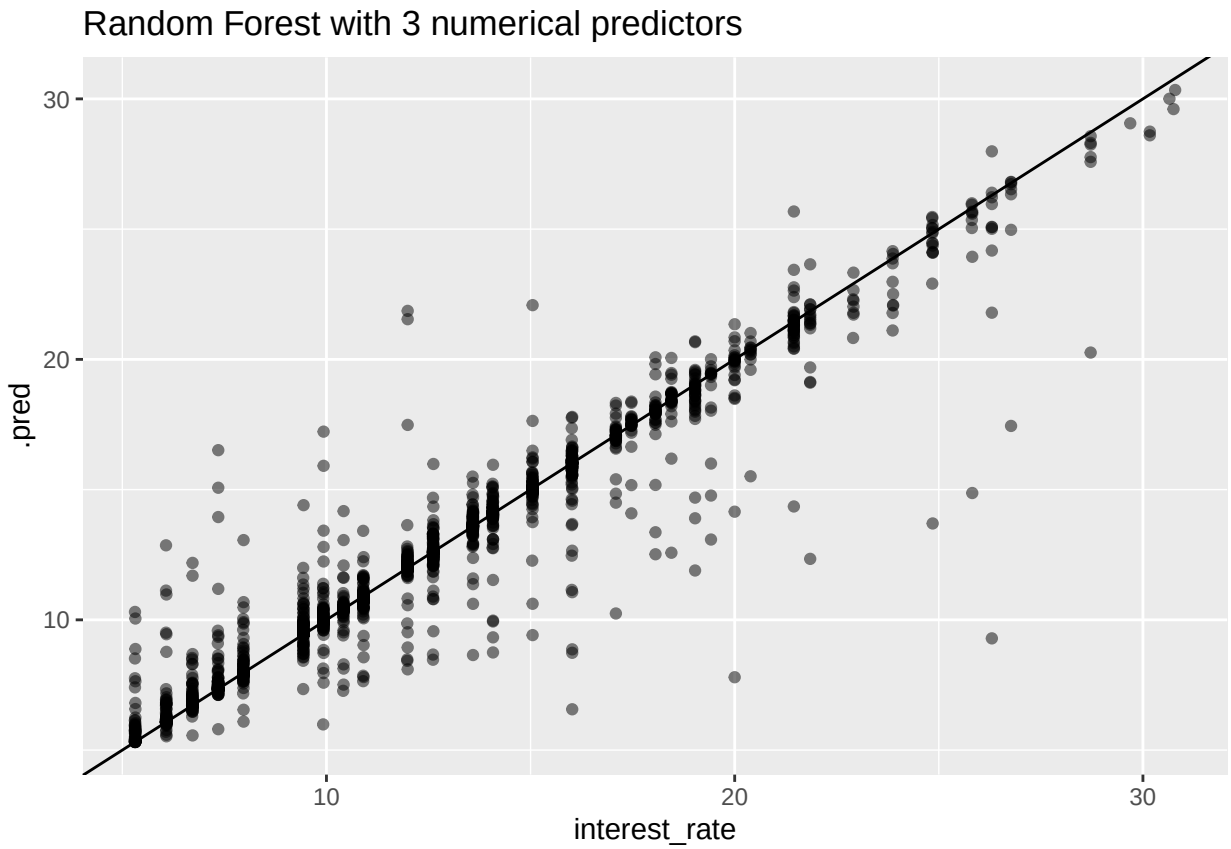
The $RMSE$ is still less than 2, indicating that the model is still performing well. The advantage is that even though we only have three predictors, we can still keep the model performing well.

```
set.seed(1234)
# mtry = .cols(): the number of columns in the predictor matrix is used
rd_spec2 <- rand_forest(mtry = .cols()) %>%
  set_engine("randomForest", importance = TRUE) %>%
  set_mode("regression")

rd_fit2 <- fit(rd_spec2, interest_rate ~ ., data = loans_train3)
augment(rd_fit2, new_data = loans_test3) %>%
  yardstick::rmse(truth = interest_rate, estimate = .pred)
```

Random Forest Performance

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        1.41
```

```
augment(rd_fit2, new_data = loans_test3) %>%
  ggplot(aes(interest_rate, .pred)) +
  geom_abline() +
  geom_point(alpha = 0.5) +
  ggtitle("Random Forest with 3 numerical predictors")
```



Visualization

Conclusion

Finally, if we need to predict the `interest_rate` based on given data, I recommend using `paid interest`, `paid principal`, and `term` by running the random forest algorithm to quickly provide business insights with limited time.

If I have more time, I'll use a recursive feature elimination algorithm and combine all numerical and categorical variables to perform feature selection. `library(caret)` is a good R package to do a feature selection. Secondly, we removed some observations due to the large number of NAs. There are some approaches that can be taken in this situation. For example, predicting missing values or replacing NAs with mean/medians mode. Finally, I believe we should try to reduce the features to make it more efficient, so if I have time, I will try to run more 3-5 feature combinations in different regression models.

Case Study 2

Read the Data

There is 1 dataset(csv) with 3 years' worth of customer orders. There are 4 columns in the csv dataset: index, CUSTOMER_EMAIL (unique identifier as hash), Net Revenue, and Year.

```
cust_orders <- read_csv("./data/customer_orders.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_double(),
##   customer_email = col_character(),
##   net_revenue = col_double(),
##   year = col_double()
## )
```

```
head(cust_orders)
```

```
## # A tibble: 6 x 4
##      X1 customer_email       net_revenue  year
##   <dbl> <chr>                      <dbl> <dbl>
## 1     0 nhknapwsbx@gmail.com       250.   2015
## 2     1 joiuzbvcpn@gmail.com        87.6  2015
## 3     2 ukkjctepxt@gmail.com       168.   2015
## 4     3 gykatilzrt@gmail.com        62.4  2015
## 5     4 mmsgsrtxah@gmail.com        43.1  2015
## 6     5 mobvusnzfr@gmail.com        39.4  2015
```

There is no NA in the dataset.

```
cust_orders %>%
summarise_all(funs(sum(is.na(.))))
```

```
## # A tibble: 1 x 4
##      X1 customer_email net_revenue  year
##   <int>          <int>       <int> <int>
## 1     0              0           0     0
```

Tidy Data

- Total revenue for the current year (2015, 2016 and 2017)

```
cust_orders %>%
  group_by(year) %>%
  summarise(total_revenue = sum(net_revenue))
```

```
## # A tibble: 3 x 2
##    year total_revenue
##   <dbl>         <dbl>
## 1  2015     29036749.
## 2  2016     25730944.
## 3  2017     31417495.
```

- New Customer Revenue e.g., **new customers not present in previous year only**

note: new customer data for 2015 is not available. + Total new customer revenue in 2016: 17,206,367 dollars. + Total new customer revenue in 2017: 16,146,519 dollars.

```
cust_orders %>%
 filter(year == 2015) -> orders_2015
#orders_2015 %>%
  #distinct(customer_email)

cust_orders %>%
 filter(year == 2016) -> orders_2016
#orders_2016 %>%
  #distinct(customer_email)

cust_orders %>%
 filter(year == 2017) -> orders_2017
#orders_2017 %>%
 #distinct(customer_email)

# New customer in 2016
orders_2016 %>%
  anti_join(orders_2015, by = "customer_email") -> tmp1
sum(tmp1$net_revenue)
```

```
## [1] 17206367
```

```
# New customer in 2017
# New customer in 2017 is defined as the new customers not present in the year of 2016, according to th
orders_2016 %>%
  anti_join(orders_2017, by = "customer_email") -> tmp2
sum(tmp2$net_revenue)
```

```
## [1] 16146519
```

- Existing Customer Growth. To calculate this, use the Revenue of existing customers for current year –(minus) Revenue of existing customers from the previous year

The existing customer is defined as the customers present in both years (2015 and 2016; 2016 and 2017) so I used inner join to find existing customers for the consecutive two years.

The revenue from existing customers increased by 39043.65 from 2015 to 2016.

```
orders_2015 %>%
  inner_join(orders_2016, by = "customer_email") %>%
  mutate(customer_growth_15_to_16 = net_revenue.y - net_revenue.x) %>%
  select(customer_email, customer_growth_15_to_16) -> growth2016
sum(growth2016$customer_growth_15_to_16)
```

```
## [1] 39043.65
```

The revenue from existing customers increased by 63857.06 from 2016 to 2017.

```
orders_2016 %>%
  inner_join(orders_2017, by = "customer_email") %>%
  mutate(customer_growth_16_to_17 = net_revenue.y - net_revenue.x) %>%
  select(customer_email, customer_growth_16_to_17) -> growth2017
sum(growth2017$customer_growth_16_to_17)
```

```
## [1] 63857.06
```

- Revenue lost from attrition Following the similar logic in question 2, I define attrition as the customers present in the previous year but not in the current year. For example, attrition for the year of 2016 would be the customers present in the year of 2015 but not in 2016.

note: there is no attrition for 2015.

In 2016, the company lost 20,551,216 in total revenue from attrition.

```
# return all rows from 2015 without a match in 2016 based on customer_email
orders_2015 %>%
  anti_join(orders_2016, by = "customer_email") -> attrition2016
sum(attrition2016$net_revenue)
```

```
## [1] 20551216
```

In 2017, the company lost 16,146,519 in total revenue from attrition.

```
# return all rows from 2016 without a match in 2017 based on customer_email
orders_2016 %>%
  anti_join(orders_2017, by = "customer_email") -> attrition2017
sum(attrition2017$net_revenue)
```

```
## [1] 16146519
```

- Existing Customer Revenue Current Year Following the same definition, existing customer is defined as the customers present in both years (2015 and 2016; 2016 and 2017).

So the total revenue of the existing customers in 2016 (current year compared to 2015) is 8,524,577 dollars.

```
# return all rows from 2016 with a match in 2015
orders_2016 %>%
  semi_join(orders_2015, by = "customer_email") -> current2016
sum(current2016$net_revenue)
```

```
## [1] 8524577
```

The total revenue of the existing customers in 2017 (current year compared to 2016) is 9,648,282 dollars.

```
# return all rows from 2017 with a match in 2016
orders_2017 %>%
  semi_join(orders_2016, by = "customer_email") -> current2017
sum(current2017$net_revenue)
```

```
## [1] 9648282
```

- Existing Customer Revenue Prior Year

Similarly total revenue of the existing customers in prior year of 2016 (2015 data) is 8,485,533 dollars.

```
orders_2016 %>%
  inner_join(orders_2015, by = "customer_email") -> prior2016
sum(prior2016$net_revenue.y)
```

```
## [1] 8485533
```

The total revenue of the existing customers in prior year of 2017 (2016 data) is 9,584,425 dollars.

```
orders_2017 %>%
  inner_join(orders_2016, by = "customer_email") -> prior2017
sum(prior2017$net_revenue.y)
```

```
## [1] 9584425
```

- Total Customers Current Year

- Total Customers Previous Year

Total customer is defined as the total number of unique emails for each year. There are 231,294 customers in 2015, 204,646 in 2016, and 249,987 in 2017.

```
orders_2015 %>%
  distinct(customer_email) %>%
  nrow()
```

```
## [1] 231294
```

```
orders_2016 %>%
  distinct(customer_email) %>%
  nrow()
```

```
## [1] 204646
```

```
orders_2017 %>%
  distinct(customer_email) %>%
  nrow()
```

```
## [1] 249987
```

- New Customers

We assume that the new customers were not on the previous year's list. There are 136891 new customers in 2016.

```
orders_2016 %>%
  anti_join(orders_2015, by = "customer_email") %>%
  distinct(customer_email) -> new2016
  nrow(new2016)
```

```
## [1] 136891
```

In 2017, there are 173449 new customers.

```
orders_2017 %>%
  anti_join(orders_2016, by = "customer_email") %>%
  distinct(customer_email) -> new2017
  nrow(new2017)
```

```
## [1] 173449
```

- Lost Customers Lost customer is also defined as attrition. There are 163,539 lost customers in 2016.

```
orders_2015 %>%
  anti_join(orders_2016, by = "customer_email") %>%
  distinct(customer_email) %>%
  nrow()
```

```
## [1] 163539
```

In 2017, there are 128,108 lost customers.

```
orders_2016 %>%
  anti_join(orders_2017, by = "customer_email") %>%
  distinct(customer_email) %>%
  nrow()
```

```
## [1] 128108
```

Data Visualization

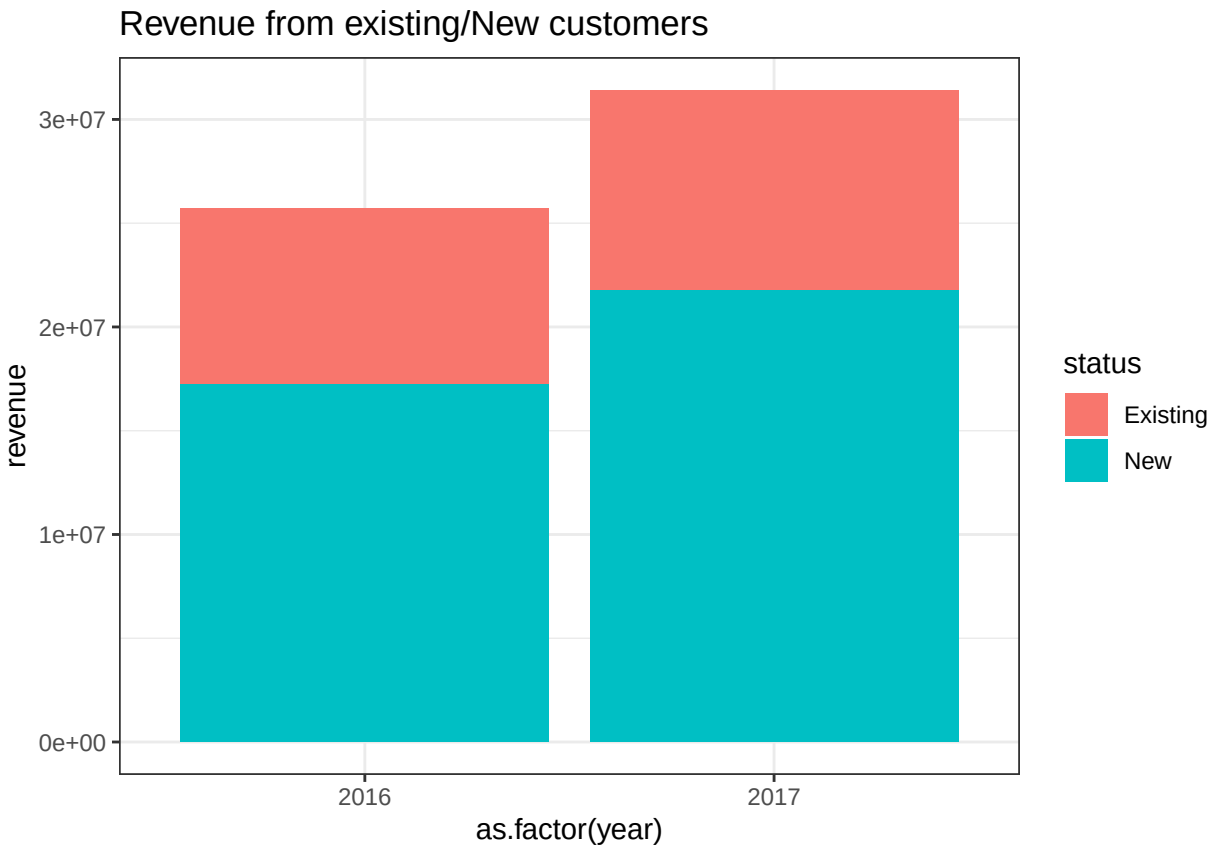
Additionally, generate a few unique plots highlighting some information from the dataset. Are there any interesting observations?

Plot1

We now have a clear understanding of the customer proportion of total revenues in 2016 and 2017. New customers generate more revenue than existing customers.

```
# New customer in 2016
orders_2016 %>%
  anti_join(orders_2015, by = "customer_email") %>%
  mutate(status = "New") -> nw2016

# New customer in 2017
orders_2017 %>%
  anti_join(orders_2016, by = "customer_email") %>%
  mutate(status = "New") -> nw2017

# Existing customer in 2016
orders_2016 %>%
  semi_join(orders_2015, by = "customer_email") %>%
  mutate(status = "Existing") -> ex2016

# Existing customer in 2017
orders_2017 %>%
  semi_join(orders_2016, by = "customer_email") %>%
  mutate(status = "Existing") -> ex2017

nw2016 %>%
  bind_rows(nw2017) %>%
  bind_rows(ex2016) %>%
  bind_rows(ex2017) -> en_20162017
en_20162017 %>%
  group_by(year, status) %>%
  summarise(revenue = sum(net_revenue)) %>%
  ggplot(aes(x = as.factor(year), y = revenue, fill = status)) +
   geom_bar(stat = "identity",
            position = "stack") +
  ggtitle("Revenue from existing/New customers") +
  theme_bw()
```

```
## `summarise()` has grouped output by 'year'. You can override using the `.groups` argument.
```

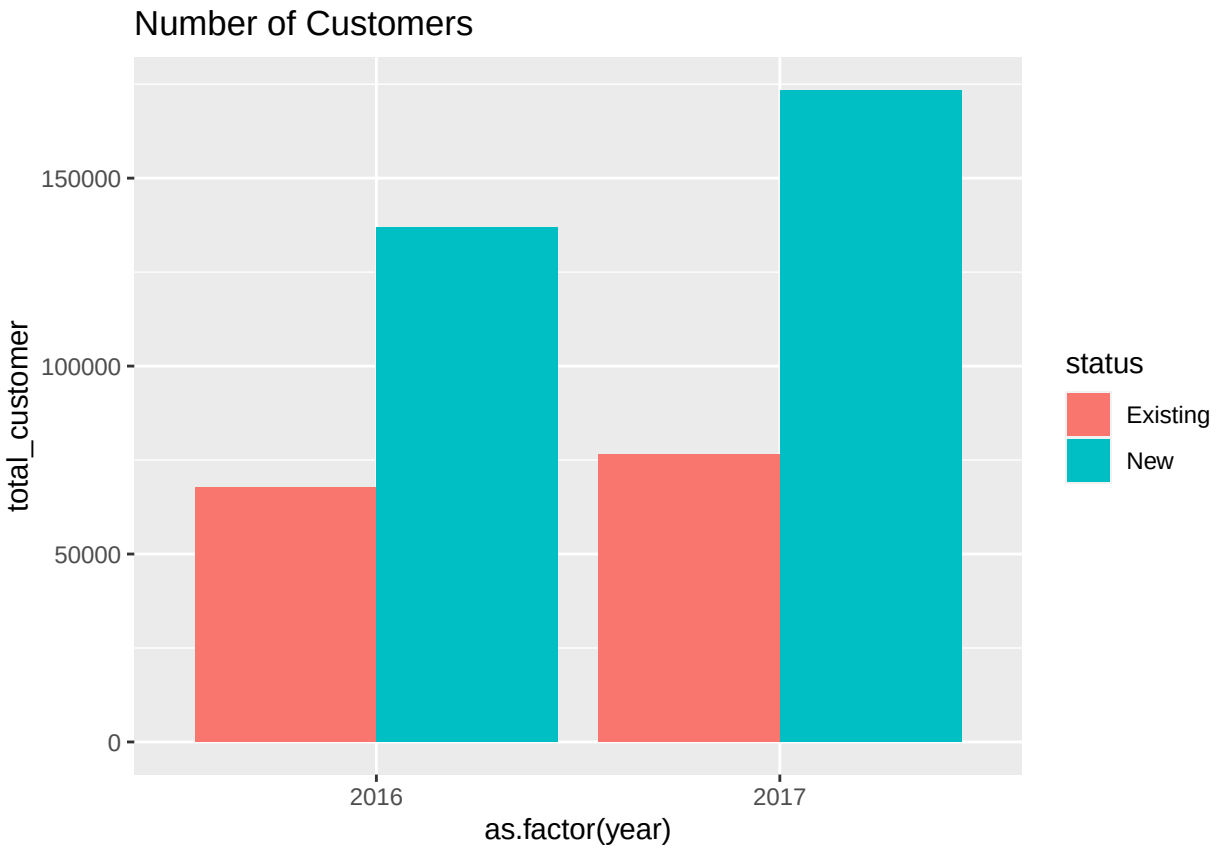


Plot 2

According to data from 2016 and 2017, the number of new customers is always greater than the number of existing customers. The company should begin to consider ways to increase customer engagement.

```
en_20162017 %>%
  group_by(year, status) %>%
  summarise(total_customer = n()) %>%
  ggplot(aes(x = as.factor(year), y = total_customer, fill = status)) +
   geom_bar(stat = "identity",
            position = "dodge") +
  ggtitle("Number of Customers")
```

```
## `summarise()` has grouped output by 'year'. You can override using the `.groups` argument.
```


Number of Customers
150000
100000
50000
0
total_customer
2016
2017
as.factor(year)
status
Existing
New