Final Group project

Group 11: Jacqui Cole, Brent Lewis, Saagar Hemrajani, Vaani Kohli - <https://github.com/shemrajani94/dsb2023_group_11>

2023-06-08

Table of Contents

# The problem: predicting credit card fraud

The goal of the project is to predict fraudulent credit card transactions.

We will be using a dataset with credit card transactions containing legitimate and fraud transactions. Fraud is typically well below 1% of all transactions, so a naive model that predicts that all transactions are legitimate and not fraudulent would have an accuracy of well over 99%– pretty good, no?

You can read more on credit card fraud on [Credit Card Fraud Detection Using Weighted Support Vector Machine](https://www.scirp.org/journal/paperinformation.aspx?paperid=105944)

The dataset we will use consists of credit card transactions and it includes information about each transaction including customer details, the merchant and category of purchase, and whether or not the transaction was a fraud.

## Obtain the data

The dataset is too large to be hosted on Canvas or Github, so please download it from dropbox <https://www.dropbox.com/sh/q1yk8mmnbbrzavl/AAAxzRtIhag9Nc_hODafGV2ka?dl=0> and save it in your dsb repo, under the data folder.

As we will be building a classifier model using tidymodels, there’s two things we need to do:

1. Define the outcome variable is\_fraud as a factor, or categorical, variable, instead of the numerical 0-1 varaibles.
2. In tidymodels, the first level is the event of interest. If we leave our data as is, 0 is the first level, but we want to find out when we actually did (1) have a fraudulent transaction

## Rows: 671,028  
## Columns: 14  
## $ trans\_date\_trans\_time <dttm> 2019-02-22 07:32:58, 2019-02-16 15:07:20, 2019-…  
## $ trans\_year <dbl> 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2020, …  
## $ category <chr> "entertainment", "kids\_pets", "personal\_care", "…  
## $ amt <dbl> 7.79, 3.89, 8.43, 40.00, 54.04, 95.61, 64.95, 3.…  
## $ city <chr> "Veedersburg", "Holloway", "Arnold", "Apison", "…  
## $ state <chr> "IN", "OH", "MO", "TN", "CO", "GA", "MN", "AL", …  
## $ lat <dbl> 40.1186, 40.0113, 38.4305, 35.0149, 39.4584, 32.…  
## $ long <dbl> -87.2602, -80.9701, -90.3870, -85.0164, -106.385…  
## $ city\_pop <dbl> 4049, 128, 35439, 3730, 277, 1841, 136, 190178, …  
## $ job <chr> "Development worker, community", "Child psychoth…  
## $ dob <date> 1959-10-19, 1946-04-03, 1985-03-31, 1991-01-28,…  
## $ merch\_lat <dbl> 39.41679, 39.74585, 37.73078, 34.53277, 39.95244…  
## $ merch\_long <dbl> -87.52619, -81.52477, -91.36875, -84.10676, -106…  
## $ is\_fraud <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …

The data dictionary is as follows

| column(variable) | description |
| --- | --- |
| trans\_date\_trans\_time | Transaction DateTime |
| trans\_year | Transaction year |
| category | category of merchant |
| amt | amount of transaction |
| city | City of card holder |
| state | State of card holder |
| lat | Latitude location of purchase |
| long | Longitude location of purchase |
| city\_pop | card holder’s city population |
| job | job of card holder |
| dob | date of birth of card holder |
| merch\_lat | Latitude Location of Merchant |
| merch\_long | Longitude Location of Merchant |
| is\_fraud | Whether Transaction is Fraud (1) or Not (0) |

We also add some of the variables we considered in our EDA for this dataset during homework 2.

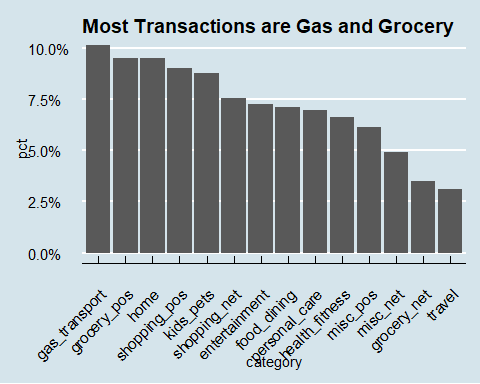
card\_fraud <- card\_fraud %>%   
 mutate( hour = hour(trans\_date\_trans\_time),  
 wday = wday(trans\_date\_trans\_time, label = TRUE),  
 month\_name = month(trans\_date\_trans\_time, label = TRUE),  
 age = interval(dob, trans\_date\_trans\_time) / years(1)  
) %>%   
 rename(year = trans\_year) %>%   
   
 mutate(  
   
 # convert latitude/longitude to radians  
 lat1\_radians = lat / 57.29577951,  
 lat2\_radians = merch\_lat / 57.29577951,  
 long1\_radians = long / 57.29577951,  
 long2\_radians = merch\_long / 57.29577951,  
   
 # calculate distance in miles  
 distance\_miles = 3963.0 \* acos((sin(lat1\_radians) \* sin(lat2\_radians)) + cos(lat1\_radians) \* cos(lat2\_radians) \* cos(long2\_radians - long1\_radians)),  
  
 # calculate distance in km  
 distance\_km = 6377.830272 \* acos((sin(lat1\_radians) \* sin(lat2\_radians)) + cos(lat1\_radians) \* cos(lat2\_radians) \* cos(long2\_radians - long1\_radians))  
  
 )

## Exploratory Data Analysis (EDA)

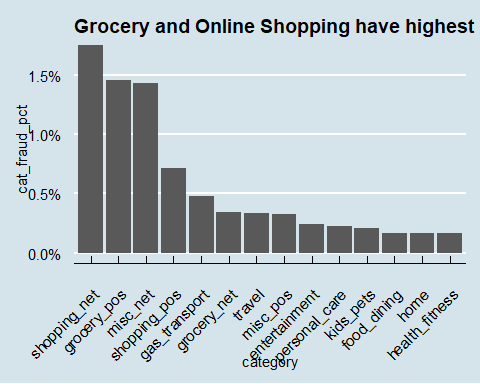
You have done some EDA and you can pool together your group’s expertise in which variables to use as features. You can reuse your EDA from earlier, but we expect at least a few visualisations and/or tables to explore teh dataset and identify any useful features.

Group all variables by type and examine each variable class by class. The dataset has the following types of variables:

card\_fraud %>%   
 group\_by(category) %>%  
 summarise(count = n()) %>%  
 arrange(desc(count)) %>%   
 mutate(pct = count/nrow(card\_fraud)) %>%   
 mutate(category = fct\_rev(fct\_reorder(category, pct, max))) %>%  
 ggplot(aes(x =category, y=pct))+  
 geom\_bar(position = 'dodge', stat='identity')+  
 theme\_minimal(base\_size=6) +  
 ggtitle("Most Transactions are Gas and Grocery") +  
 scale\_y\_continuous(labels = scales::percent) +  
 ggthemes::theme\_economist() +  
 theme(plot.title = element\_text(size = 14),  
 axis.text.x = element\_text(angle = 45, hjust = 1, vjust = 0.85)) # Adjust the size value as needed

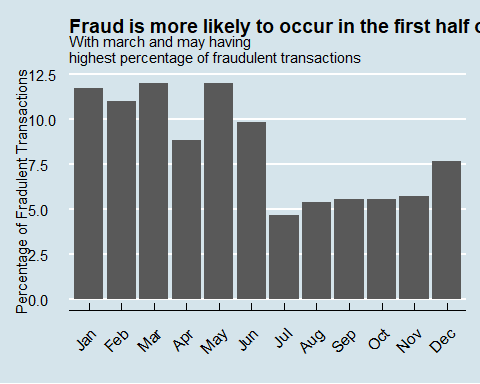


num\_fraud <- card\_fraud %>%  
 filter(is\_fraud == 1) %>%  
 count() %>%   
 pull(n)  
  
num\_category <- card\_fraud %>%  
 group\_by(category) %>%  
 summarise(category\_count = n())  
  
  
rate\_by\_cat <- card\_fraud %>%  
 filter(is\_fraud == 1 & !is.na(category)) %>%   
 group\_by(category) %>%   
 summarise(category\_fraud\_count = n()) %>%   
 left\_join(num\_category, by='category') %>%   
 mutate(cat\_fraud\_pct = category\_fraud\_count/category\_count) %>%   
 arrange(desc(cat\_fraud\_pct))   
   
   
rate\_by\_cat %>%   
 mutate(category = fct\_rev(fct\_reorder(category, cat\_fraud\_pct, max))) %>%  
 ggplot(aes(x =category, y=cat\_fraud\_pct))+  
 geom\_bar(position = 'dodge', stat='identity')+  
 theme\_minimal(base\_size=6) +  
 ggtitle("Grocery and Online Shopping have highest percentage of fraud") +  
 scale\_y\_continuous(labels = scales::percent) +  
 ggthemes::theme\_economist() +  
 theme(plot.title = element\_text(size = 14),  
 axis.text.x = element\_text(angle = 45, hjust = 1, vjust = 0.85)) # Adjust the size value as needed

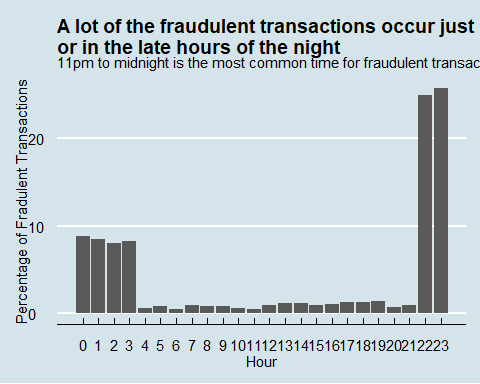


card\_fraud\_date <- card\_fraud %>%   
 mutate(  
 date\_only = lubridate::date(trans\_date\_trans\_time),  
 month\_name = lubridate::month(trans\_date\_trans\_time, label=TRUE),  
 month\_number = month(date(trans\_date\_trans\_time)),  
 hour = lubridate::hour(trans\_date\_trans\_time),  
 weekday = lubridate::wday(trans\_date\_trans\_time, label = TRUE),  
 weekday\_num = lubridate::wday(trans\_date\_trans\_time, label = FALSE)  
 )  
  
card\_fraud\_date %>%  
 filter(is\_fraud == 1 & !is.na(month\_name)) %>%  
 mutate(month\_name = fct\_reorder(month\_name, month\_number, max)) %>%  
 group\_by(month\_name, month\_number) %>%  
 summarize(pct\_fraud\_transactions = n() / num\_fraud \* 100) %>%   
 ggplot(aes(x = month\_name, y = pct\_fraud\_transactions)) +  
 geom\_bar(stat = "identity", position = "dodge") +  
 labs(title = "Fraud is more likely to occur in the first half of the year than the second",  
 x = "",  
 y = "Percentage of Fradulent Transactions",  
 subtitle = "With march and may having\nhighest percentage of fraudulent transactions") +  
 ggthemes::theme\_economist() +   
 theme(plot.title = element\_text(size = 14),  
 axis.text.x = element\_text(angle = 45, hjust = 1, vjust = 0.85))

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

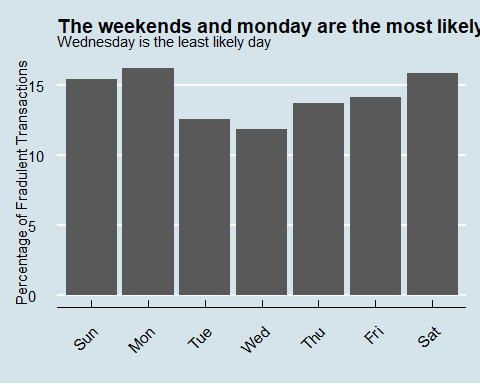


card\_fraud\_date %>%  
 filter(is\_fraud == 1 & !is.na(hour)) %>%  
 arrange(hour) %>%   
 group\_by(hour) %>%  
 summarize(pct\_fraud\_transactions = n() / num\_fraud \* 100) %>%   
 ggplot(aes(x = hour, y = pct\_fraud\_transactions)) +  
 geom\_bar(stat = "identity", position = "dodge") +  
 scale\_x\_continuous(breaks = seq(0, 23, by = 1)) +  
 labs(title = "A lot of the fraudulent transactions occur just before midnight\nor in the late hours of the night",  
 x = "Hour",  
 y = "Percentage of Fradulent Transactions",  
 subtitle = "11pm to midnight is the most common time for fraudulent transactions to occur") +  
 ggthemes::theme\_economist() +   
 theme(plot.title = element\_text(size = 14),  
 axis.text.x = element\_text(hjust = 0.5, vjust = 0.85))



card\_fraud\_date %>%  
 filter(is\_fraud == 1 & !is.na(weekday)) %>%  
 mutate(weekday = fct\_reorder(weekday, weekday\_num, max)) %>%  
 group\_by(weekday, weekday\_num) %>%  
 summarize(pct\_fraud\_transactions = n() / num\_fraud \* 100) %>%   
 ggplot(aes(x = weekday, y = pct\_fraud\_transactions)) +  
 geom\_bar(stat = "identity", position = "dodge") +  
 labs(title = "The weekends and monday are the most likely days for fraud",  
 x = "",  
 y = "Percentage of Fradulent Transactions",  
 subtitle = "Wednesday is the least likely day") +  
 ggthemes::theme\_economist() +   
 theme(plot.title = element\_text(size = 14),  
 axis.text.x = element\_text(angle = 45, hjust = 1, vjust = 0.85))

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



1. Strings
2. Geospatial Data
3. Dates
4. Date/Times
5. Numerical

Strings are usually not a useful format for classification problems. The strings should be converted to factors, dropped, or otherwise transformed.

***Strings to Factors***

* category, Category of Merchant
* job, Job of Credit Card Holder

***Strings to Geospatial Data***

We have plenty of geospatial data as lat/long pairs, so I want to convert city/state to lat/long so I can compare to the other geospatial variables. This will also make it easier to compute new variables like the distance the transaction is from the home location.

* city, City of Credit Card Holder
* state, State of Credit Card Holder

## Exploring factors: how is the compactness of categories?

* Do we have excessive number of categories? Do we want to combine some?

card\_fraud %>%   
 count(category, sort=TRUE)%>%   
 mutate(perc = n/sum(n))

## # A tibble: 14 × 3  
## category n perc  
## <chr> <int> <dbl>  
## 1 gas\_transport 68046 0.101   
## 2 grocery\_pos 63791 0.0951  
## 3 home 63597 0.0948  
## 4 shopping\_pos 60416 0.0900  
## 5 kids\_pets 58772 0.0876  
## 6 shopping\_net 50743 0.0756  
## 7 entertainment 48521 0.0723  
## 8 food\_dining 47527 0.0708  
## 9 personal\_care 46843 0.0698  
## 10 health\_fitness 44341 0.0661  
## 11 misc\_pos 41244 0.0615  
## 12 misc\_net 32829 0.0489  
## 13 grocery\_net 23485 0.0350  
## 14 travel 20873 0.0311

card\_fraud %>%   
 count(job, sort=TRUE) %>%   
 mutate(perc = n/sum(n))

## # A tibble: 494 × 3  
## job n perc  
## <chr> <int> <dbl>  
## 1 Film/video editor 5106 0.00761  
## 2 Exhibition designer 4728 0.00705  
## 3 Naval architect 4546 0.00677  
## 4 Surveyor, land/geomatics 4448 0.00663  
## 5 Materials engineer 4292 0.00640  
## 6 Designer, ceramics/pottery 4262 0.00635  
## 7 IT trainer 4014 0.00598  
## 8 Financial adviser 3959 0.00590  
## 9 Systems developer 3948 0.00588  
## 10 Environmental consultant 3831 0.00571  
## # ℹ 484 more rows

The predictors category and job are transformed into factors.

card\_fraud <- card\_fraud %>%   
 mutate(category = factor(category),  
 job = factor(job),  
 hour = factor(hour))

category has 14 unique values, and job has 494 unique values. The dataset is quite large, with over 670K records, so these variables don’t have an excessive number of levels at first glance. However, it is worth seeing if we can compact the levels to a smaller number.

### Why do we care about the number of categories and whether they are “excessive”?

Consider the extreme case where a dataset had categories that only contained one record each. There is simply insufficient data to make correct predictions using category as a predictor on new data with that category label. Additionally, if your modeling uses dummy variables, having an extremely large number of categories will lead to the production of a huge number of predictors, which can slow down the fitting. This is fine if all the predictors are useful, but if they aren’t useful (as in the case of having only one record for a category), trimming them will improve the speed and quality of the data fitting.

If I had subject matter expertise, I could manually combine categories. If you don’t have subject matter expertise, or if performing this task would be too labor intensive, then you can use cutoffs based on the amount of data in a category. If the majority of the data exists in only a few categories, then it might be reasonable to keep those categories and lump everything else in an “other” category or perhaps even drop the data points in smaller categories.

## Do all variables have sensible types?

Consider each variable and decide whether to keep, transform, or drop it. This is a mixture of Exploratory Data Analysis and Feature Engineering, but it’s helpful to do some simple feature engineering as you explore the data. In this project, we have all data to begin with, so any transformations will be performed on the entire dataset. Ideally, do the transformations as a recipe\_step() in the tidymodels framework. Then the transformations would be applied to any data the recipe was used on as part of the modeling workflow. There is less chance of data leakage or missing a step when you perform the feature engineering in the recipe.

## Which variables to keep in your model?

You have a number of variables and you have to decide which ones to use in your model. For instance, you have the latitude/lognitude of the customer, that of the merchant, the same data in radians, as well as the distance\_km and distance\_miles. Do you need them all?

card\_fraud %>%  
 glimpse

## Rows: 671,028  
## Columns: 24  
## $ trans\_date\_trans\_time <dttm> 2019-02-22 07:32:58, 2019-02-16 15:07:20, 2019-…  
## $ year <dbl> 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2020, …  
## $ category <fct> entertainment, kids\_pets, personal\_care, grocery…  
## $ amt <dbl> 7.79, 3.89, 8.43, 40.00, 54.04, 95.61, 64.95, 3.…  
## $ city <chr> "Veedersburg", "Holloway", "Arnold", "Apison", "…  
## $ state <chr> "IN", "OH", "MO", "TN", "CO", "GA", "MN", "AL", …  
## $ lat <dbl> 40.1186, 40.0113, 38.4305, 35.0149, 39.4584, 32.…  
## $ long <dbl> -87.2602, -80.9701, -90.3870, -85.0164, -106.385…  
## $ city\_pop <dbl> 4049, 128, 35439, 3730, 277, 1841, 136, 190178, …  
## $ job <fct> "Development worker, community", "Child psychoth…  
## $ dob <date> 1959-10-19, 1946-04-03, 1985-03-31, 1991-01-28,…  
## $ merch\_lat <dbl> 39.41679, 39.74585, 37.73078, 34.53277, 39.95244…  
## $ merch\_long <dbl> -87.52619, -81.52477, -91.36875, -84.10676, -106…  
## $ is\_fraud <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …  
## $ hour <fct> 7, 15, 22, 10, 17, 1, 9, 17, 16, 12, 10, 16, 23,…  
## $ wday <ord> Fri, Sat, Fri, Sun, Sat, Mon, Sun, Tue, Sat, Sun…  
## $ month\_name <ord> Feb, Feb, Dec, Mar, Feb, Sep, Dec, Jun, Dec, Dec…  
## $ age <dbl> 59.34607, 72.87570, 34.74299, 28.09431, 33.85676…  
## $ lat1\_radians <dbl> 0.7002017, 0.6983289, 0.6707388, 0.6111253, 0.68…  
## $ lat2\_radians <dbl> 0.6879527, 0.6936959, 0.6585264, 0.6027105, 0.69…  
## $ long1\_radians <dbl> -1.522978, -1.413195, -1.577551, -1.483816, -1.8…  
## $ long2\_radians <dbl> -1.527620, -1.422876, -1.594686, -1.467940, -1.8…  
## $ distance\_miles <dbl> 50.56023, 34.69750, 72.10554, 61.50534, 43.21622…  
## $ distance\_km <dbl> 81.36880, 55.84021, 116.04262, 98.98324, 69.5497…

## Fit your workflows in smaller sample

You will be running a series of different models, along the lines of the California housing example we have seen in class. However, this dataset has 670K rows and if you try various models and run cross validation on them, your computer may slow down or crash.

Thus, we will work with a smaller sample of 10% of the values the original dataset to identify the best model, and once we have the best model we can use the full dataset to train- test our best model.

# RATIONALE BEHIND OUR DECISION-MAKING PROCESS TO DETERMINE WHICH FEATURES TO CHOOSE FOR THE CREATION OF OUR CLASSIFICATION-BASED MACHINE-LEARNING MODEL THAT PREDICTS IF A TRANSACTION IS FRAUDULENT OR NOT.  
#  
# Considering each categories  
#  
# Strings  
#  
# category - we saw from our homework2 that fraud is linked to the category so we need to include this as a feature  
#  
# So, we short list: category  
#  
# job - we considered this but can see that there are far too many jobs listed that this data category is not useful as it is too fine  
# grained for our purposes. Hence, we drop this variable.  
#   
# city, states and city\_pop are likely correlated to each other. We are also not sure that these variables will impact significantly upon the likelihood of fraud. We decide that it is possible but we do not include any of these variables as a feature for the model for the time being, since we wish to see how the training and testing of our model fares with the other features that we have already chosen. Indeed, we don't wish to employ too many features because that could overfit the model. We could come back to considering the addition of one of these city/country variables later if need be, once we see if our model works or not with our other features.  
#  
# Geospatial data  
#  
# A range of these type of data are given which we realise are highly correlated, i.e. latitude and longitude of the transaction (lat, long), the latitude and longitude of the merchant (merch\_lat, merch\_long), the radius of one and the other latitude and longitude (lat1\_radius and lat2\_radius, and long1\_radius, and long2\_radius), distance between the merchant and card holder that is given in both km and miles (distance\_miles and distance\_km).  
#  
# It only feels sensible to choose distance\_km since its calculation is based on the latitude and longitude of both merchant and card holder, and the radius of these variables. Meanwhile, we have to choose between a distance as a km or miles unit, so we simply decide on the metric option: distance\_km.  
#  
# So, we short list: distance\_km  
#  
# Dates and Times  
#  
# The dob variable is dropped because we intuit that the day or month of someone's birth will not impact on whether they are a target of fraud or not. However, their year of birth could impact on the likelihood of being a target of fraud or otherwise. One of the other variables, age, already exists as a proxy to the year of birth and so we decide that we can simply drop the dob, and use age for our purposes.  
#  
# So we short list: age  
#  
# We intuit that trans\_date\_trans\_time will affect the likelihood of a fraudulent transaction (see homework 2). The trans\_date\_trans\_time will need to be split up into hour, day, month and year, and whether or not the date was a weekday or not, but this has already been done for us, given the following variables are given: hour, wday, and month\_name. So trans\_date\_trans\_time is considered to be redundant. Homework2 also showed that the evidence was unclear for any correlation between year and fraud likelihood.  
#  
# So, we short list: hour, wday, month\_name  
#  
# Numericals  
#  
# We intuit that the amount (amt) of money in the card transaction is surely a feature because it is more attractive to commit fraud when there is a great 'return on (fraud) investment'. Indeed, we saw in Homework2 that the amt variable is significantly correlated to the likelihood of fraudulent behaviour or otherwise.  
#  
# In summary, the above text has considered all variables in the card\_fraud dataset in terms of whether or not they should be selected as any of our features for creating our machine-learning (classification) model which predicts whether nor not fraud occurs or not (cf the is\_fraud variable). We concluded that the following variables should go forward as features:  
#  
# category, amt, hour, wday, month\_name, age, distance\_km  
#

# select a smaller subset  
my\_card\_fraud <- card\_fraud %>%   
 select(is\_fraud, category, amt, hour, wday, month\_name, age, distance\_km) %>%   
 slice\_sample(prop = 0.05) %>%   
 mutate(hour = as.factor(hour))

## Split the data in training - testing

# \*\*Split the data\*\*  
  
set.seed(123)  
  
data\_split <- initial\_split(my\_card\_fraud, # updated data  
 prop = 0.8,   
 strata = is\_fraud)  
  
card\_fraud\_train <- training(data\_split)   
card\_fraud\_test <- testing(data\_split)

## Cross Validation

Start with 3 CV folds to quickly get an estimate for the best model and you can increase the number of folds to 5 or 10 later.

set.seed(123)  
cv\_folds <- vfold\_cv(data = card\_fraud\_train,   
 v = 3,   
 strata = is\_fraud)  
cv\_folds

## # 3-fold cross-validation using stratification   
## # A tibble: 3 × 2  
## splits id   
## <list> <chr>  
## 1 <split [17893/8947]> Fold1  
## 2 <split [17893/8947]> Fold2  
## 3 <split [17894/8946]> Fold3

## Define a tidymodels recipe

What steps are you going to add to your recipe? Do you need to do any log transformations?

fraud\_rec <- recipe(is\_fraud ~ .,   
 data = card\_fraud\_train) %>%  
 step\_log(amt, distance\_km) %>%   
 step\_naomit(everything(), skip = TRUE) %>%   
 step\_novel(all\_nominal(), -all\_outcomes()) %>%  
 step\_normalize(all\_numeric(), -all\_outcomes()) %>%  
 step\_dummy(all\_nominal(), -all\_outcomes()) %>%  
 step\_zv(all\_numeric(), -all\_outcomes())  
  
# - `step\_novel()` converts all nominal variables to factors and takes care of other issues related to categorical variables.  
#   
# - `step\_log()` will log transform data (since some of our numerical variables are right-skewed). Note that this step can not be performed on negative numbers.  
#   
# - `step\_normalize()` normalizes (center and scales) the numeric variables to have a standard deviation of one and a mean of zero. (i.e., z-standardization).  
#   
# - `step\_dummy()` converts our factor column ocean\_proximity into numeric binary (0 and 1) variables.  
#   
# `step\_zv()` removes any numeric variables that have zero variance.  
#   
# `step\_corr()` will remove predictor variables with high correlations with other predictor variables.

Once you have your recipe, you can check the pre-processed dataframe

prepped\_data <-   
 fraud\_rec %>% # use the recipe object  
 prep() %>% # perform the recipe on training data  
 juice() # extract only the preprocessed dataframe   
  
glimpse(prepped\_data)

## Rows: 26,840  
## Columns: 59  
## $ amt <dbl> 0.94257260, 0.41894969, -2.28514393, 0.4953878…  
## $ age <dbl> -1.02932975, -1.54964789, -0.71398546, -0.4812…  
## $ distance\_km <dbl> 0.92831452, 0.81452441, -0.94743592, 0.9216576…  
## $ is\_fraud <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_food\_dining <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0…  
## $ category\_gas\_transport <dbl> 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_grocery\_net <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_grocery\_pos <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_health\_fitness <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_home <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_kids\_pets <dbl> 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_misc\_net <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0…  
## $ category\_misc\_pos <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_personal\_care <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0…  
## $ category\_shopping\_net <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0…  
## $ category\_shopping\_pos <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1…  
## $ category\_travel <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0…  
## $ hour\_X1 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ hour\_X2 <dbl> 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ hour\_X3 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ hour\_X4 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ hour\_X5 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0…  
## $ hour\_X6 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ hour\_X7 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0…  
## $ hour\_X8 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0…  
## $ hour\_X9 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ hour\_X10 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ hour\_X11 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ hour\_X12 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0…  
## $ hour\_X13 <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ hour\_X14 <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0…  
## $ hour\_X15 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ hour\_X16 <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0…  
## $ hour\_X17 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ hour\_X18 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1…  
## $ hour\_X19 <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ hour\_X20 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0…  
## $ hour\_X21 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ hour\_X22 <dbl> 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ hour\_X23 <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ wday\_1 <dbl> -0.54006172, 0.23145502, 0.38575837, 0.2314550…  
## $ wday\_2 <dbl> 0.54006172, -0.23145502, 0.07715167, -0.231455…  
## $ wday\_3 <dbl> -0.4308202, -0.4308202, -0.3077287, -0.4308202…  
## $ wday\_4 <dbl> 0.2820380, -0.1208734, -0.5237849, -0.1208734,…  
## $ wday\_5 <dbl> -0.1497862, 0.3637664, -0.4921546, 0.3637664, …  
## $ wday\_6 <dbl> 0.06154575, 0.55391171, -0.30772873, 0.5539117…  
## $ wday\_7 <dbl> -0.01706972, 0.35846409, -0.11948803, 0.358464…  
## $ month\_name\_01 <dbl> 0.22237479, -0.22237479, -0.29649973, 0.222374…  
## $ month\_name\_02 <dbl> -0.11174754, -0.11174754, 0.04469902, -0.11174…  
## $ month\_name\_03 <dbl> -3.344968e-01, 3.344968e-01, 2.508726e-01, -3.…  
## $ month\_name\_04 <dbl> -0.2069771, -0.2069771, -0.3679593, -0.2069771…  
## $ month\_name\_05 <dbl> 0.1398355, -0.1398355, 0.2288218, 0.1398355, 0…  
## $ month\_name\_06 <dbl> 0.36069561, 0.36069561, 0.06710616, 0.36069561…  
## $ month\_name\_07 <dbl> 0.2467613, -0.2467613, -0.3388855, 0.2467613, …  
## $ month\_name\_08 <dbl> -0.09741925, -0.09741925, 0.45633226, -0.09741…  
## $ month\_name\_09 <dbl> -0.39408195, 0.39408195, -0.40679427, -0.39408…  
## $ month\_name\_10 <dbl> -0.458540826, -0.458540826, 0.267079920, -0.45…  
## $ month\_name\_11 <dbl> -0.320804707, 0.320804707, -0.128321883, -0.32…  
## $ month\_name\_12 <dbl> -0.1337848111, -0.1337848111, 0.0401354433, -0…

## Define various models

You should define the following classification models:

1. Logistic regression, using the glm engine
2. Decision tree, using the C5.0 engine
3. Random Forest, using the ranger engine and setting importance = "impurity")
4. A boosted tree using Extreme Gradient Boosting, and the xgboost engine
5. A k-nearest neighbours, using 4 nearest\_neighbors and the kknn engine

## Model Building   
  
# 1. Pick a `model type`  
# 2. set the `engine`  
# 3. Set the `mode`: classification  
  
# Logistic regression  
log\_spec <- logistic\_reg() %>% # model type  
 set\_engine(engine = "glm") %>% # model engine  
 set\_mode("classification") # model mode  
  
# Show your model specification  
log\_spec

## Logistic Regression Model Specification (classification)  
##   
## Computational engine: glm

# Decision Tree  
tree\_spec <- decision\_tree() %>%  
 set\_engine(engine = "C5.0") %>%  
 set\_mode("classification")  
  
tree\_spec

## Decision Tree Model Specification (classification)  
##   
## Computational engine: C5.0

# Random Forest  
library(ranger)  
  
rf\_spec <-   
 rand\_forest() %>%   
 set\_engine("ranger", importance = "impurity") %>%   
 set\_mode("classification")  
  
  
# Boosted tree (XGBoost)  
library(xgboost)

##   
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':  
##   
## slice

xgb\_spec <-   
 boost\_tree() %>%   
 set\_engine("xgboost") %>%   
 set\_mode("classification")   
  
# K-nearest neighbour (k-NN)  
knn\_spec <-   
 nearest\_neighbor(neighbors = 4) %>% # we can adjust the number of neighbors   
 set\_engine("kknn") %>%   
 set\_mode("classification")

## Bundle recipe and model with workflows

## Bundle recipe and model with `workflows`  
  
  
log\_wflow <- # new workflow object  
 workflow() %>% # use workflow function  
 add\_recipe(fraud\_rec) %>% # use the new recipe  
 add\_model(log\_spec) # add your model spec  
  
tree\_wflow <-  
 workflow() %>%  
 add\_recipe(fraud\_rec) %>%   
 add\_model(tree\_spec)   
  
rf\_wflow <-  
 workflow() %>%  
 add\_recipe(fraud\_rec) %>%   
 add\_model(rf\_spec)   
  
xgb\_wflow <-  
 workflow() %>%  
 add\_recipe(fraud\_rec) %>%   
 add\_model(xgb\_spec)  
  
knn\_wflow <-  
 workflow() %>%  
 add\_recipe(fraud\_rec) %>%   
 add\_model(knn\_spec)  
  
  
  
# show object  
knn\_wflow

## ══ Workflow ════════════════════════════════════════════════════════════════════  
## Preprocessor: Recipe  
## Model: nearest\_neighbor()  
##   
## ── Preprocessor ────────────────────────────────────────────────────────────────  
## 6 Recipe Steps  
##   
## • step\_log()  
## • step\_naomit()  
## • step\_novel()  
## • step\_normalize()  
## • step\_dummy()  
## • step\_zv()  
##   
## ── Model ───────────────────────────────────────────────────────────────────────  
## K-Nearest Neighbor Model Specification (classification)  
##   
## Main Arguments:  
## neighbors = 4  
##   
## Computational engine: kknn

## Fit models

You may want to compare the time it takes to fit each model. tic() starts a simple timer and toc() stops it

tic()  
log\_res <- log\_wflow %>%   
 fit\_resamples(  
 resamples = cv\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas, accuracy,  
 kap, roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE))

## → A | warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## There were issues with some computations A: x1 → B | warning: prediction from rank-deficient fit; attr(\*, "non-estim") has doubtful cases  
## There were issues with some computations A: x1There were issues with some computations A: x1 B: x1There were issues with some computations A: x2 B: x1There were issues with some computations A: x2 B: x2There were issues with some computations A: x3 B: x2There were issues with some computations A: x3 B: x3There were issues with some computations A: x3 B: x3

time <- toc()

## 17.12 sec elapsed

log\_time <- time[[4]]  
  
  
tic()  
tree\_res <- tree\_wflow %>%   
 fit\_resamples(  
 resamples = cv\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas, accuracy,  
 kap, roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE))   
time <- toc()

## 34.53 sec elapsed

tree\_time <- time[[4]]  
  
tic()  
rf\_res <- rf\_wflow %>%   
 fit\_resamples(  
 resamples = cv\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas, accuracy,  
 kap, roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE))

## → A | warning: While computing binary `precision()`, no predicted events were detected (i.e. `true\_positive + false\_positive = 0`).   
## Precision is undefined in this case, and `NA` will be returned.  
## Note that 56 true event(s) actually occured for the problematic event level, '1'.  
## There were issues with some computations A: x1 → B | warning: While computing binary `precision()`, no predicted events were detected (i.e. `true\_positive + false\_positive = 0`).   
## Precision is undefined in this case, and `NA` will be returned.  
## Note that 63 true event(s) actually occured for the problematic event level, '1'.  
## There were issues with some computations A: x1There were issues with some computations A: x1 B: x1There were issues with some computations A: x1 B: x1

time <- toc()

## 119.36 sec elapsed

rf\_time <- time[[4]]  
  
tic()  
xgb\_res <- xgb\_wflow %>%   
 fit\_resamples(  
 resamples = cv\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas, accuracy,  
 kap, roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE))   
time <- toc()

## 19.65 sec elapsed

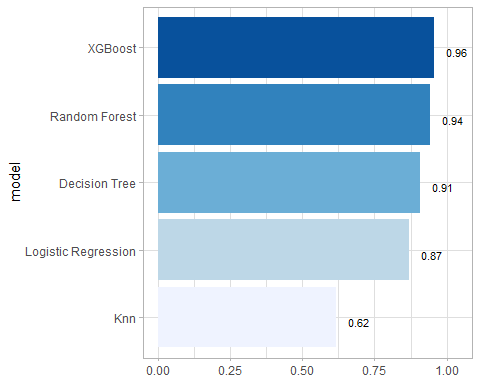
xgb\_time <- time[[4]]  
  
tic()  
knn\_res <- knn\_wflow %>%   
 fit\_resamples(  
 resamples = cv\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas, accuracy,  
 kap, roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE))   
time <- toc()

## 336.47 sec elapsed

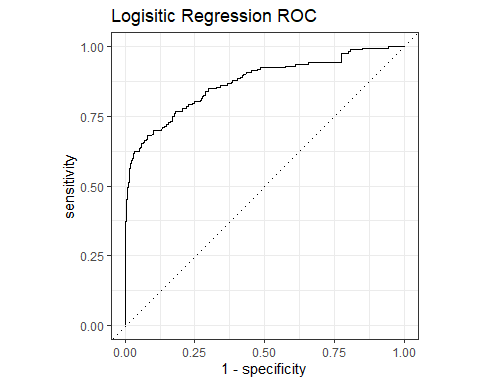
knn\_time <- time[[4]]

## Compare models

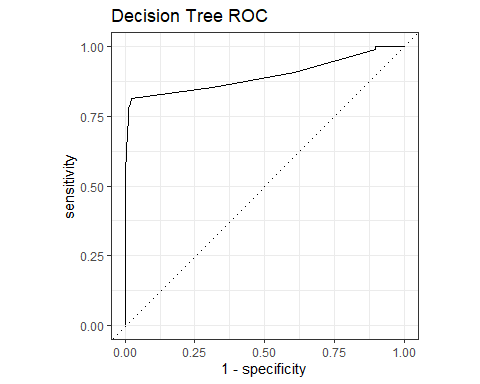
## Model Comparison  
  
log\_metrics <-   
 log\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 # add the name of the model to every row  
 mutate(model = "Logistic Regression",  
 time = log\_time)  
  
tree\_metrics <-   
 tree\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "Decision Tree",  
 time = tree\_time)  
  
rf\_metrics <-   
 rf\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "Random Forest",  
 time = rf\_time)  
  
xgb\_metrics <-   
 xgb\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "XGBoost",  
 time = xgb\_time)  
  
knn\_metrics <-   
 knn\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "Knn",  
 time = knn\_time)  
  
# add more models here  
  
# create dataframe with all models  
model\_compare <- bind\_rows(log\_metrics,  
 tree\_metrics,  
 rf\_metrics,  
 xgb\_metrics,  
 knn\_metrics  
 ) %>%   
 # get rid of 'sec elapsed' and turn it into a number  
 mutate(time = str\_sub(time, end = -13) %>%   
 as.double()  
 )  
  
#Pivot wider to create barplot  
 model\_comp <- model\_compare %>%   
 select(model, .metric, mean, std\_err) %>%   
 pivot\_wider(names\_from = .metric, values\_from = c(mean, std\_err))   
  
# show mean are under the curve (ROC-AUC) for every model  
model\_comp %>%   
 arrange(mean\_roc\_auc) %>%   
 mutate(model = fct\_reorder(model, mean\_roc\_auc)) %>% # order results  
 ggplot(aes(model, mean\_roc\_auc, fill=model)) +  
 geom\_col() +  
 coord\_flip() +  
 scale\_fill\_brewer(palette = "Blues") +  
 geom\_text(  
 size = 3,  
 aes(label = round(mean\_roc\_auc, 2),   
 y = mean\_roc\_auc + 0.08),  
 vjust = 1  
 )+  
 theme\_light()+  
 theme(legend.position = "none")+  
 labs(y = NULL)



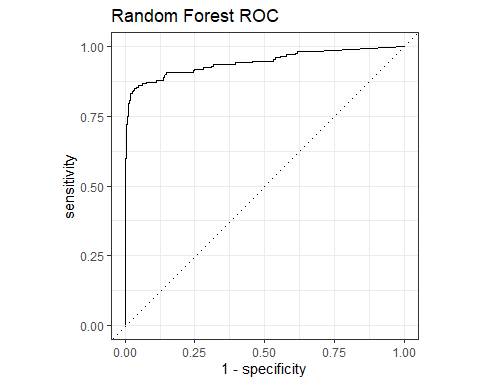
log\_pred <- log\_res %>% collect\_predictions()  
log\_pred %>%   
 roc\_curve(is\_fraud, .pred\_1) %>%   
 autoplot() +  
 labs(title="Logisitic Regression ROC")



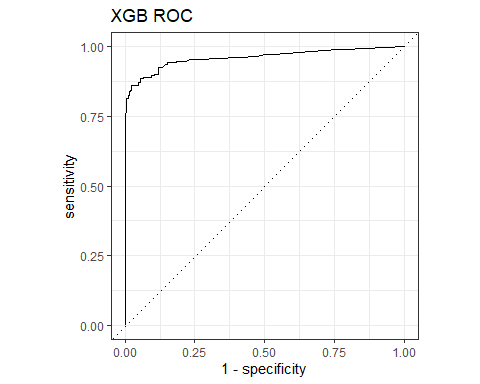
tree\_pred <- tree\_res %>% collect\_predictions()  
tree\_pred %>%   
 roc\_curve(is\_fraud, .pred\_1) %>%   
 autoplot() +  
 labs(title="Decision Tree ROC")



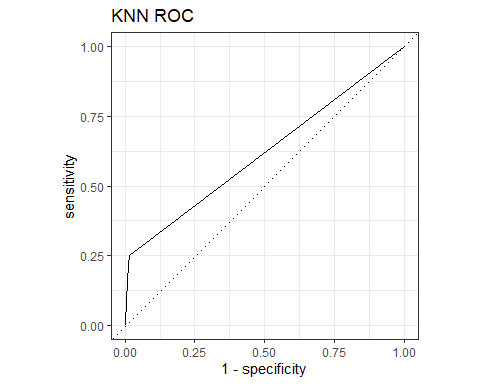
rf\_pred <- rf\_res %>% collect\_predictions()  
rf\_pred %>%   
 roc\_curve(is\_fraud, .pred\_1) %>%   
 autoplot() +  
 labs(title="Random Forest ROC")



xgb\_pred <- xgb\_res %>% collect\_predictions()  
xgb\_pred %>%   
 roc\_curve(is\_fraud, .pred\_1) %>%   
 autoplot() +  
 labs(title="XGB ROC")



knn\_pred <- knn\_res %>% collect\_predictions()  
knn\_pred %>%   
 roc\_curve(is\_fraud, .pred\_1) %>%   
 autoplot() +  
 labs(title="KNN ROC")



## Which metric to use

This is a highly imbalanced data set, as roughly 99.5% of all transactions are ok, and it’s only 0.5% of transactions that are fraudulent. A naive model, which classifies everything as ok and not-fraud, would have an accuracy of 99.5%, but what about the sensitivity, specificity, the AUC, etc?

## last\_fit()

last\_fit\_xgb <- last\_fit(xgb\_wflow,   
 split = data\_split,  
 metrics = metric\_set(  
 accuracy, f\_meas, kap, precision,  
 recall, roc\_auc, sens, spec))  
  
last\_fit\_xgb %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 4  
## .metric .estimator .estimate .config   
## <chr> <chr> <dbl> <chr>   
## 1 accuracy binary 0.997 Preprocessor1\_Model1  
## 2 f\_meas binary 0.676 Preprocessor1\_Model1  
## 3 kap binary 0.675 Preprocessor1\_Model1  
## 4 precision binary 0.793 Preprocessor1\_Model1  
## 5 recall binary 0.590 Preprocessor1\_Model1  
## 6 sens binary 0.590 Preprocessor1\_Model1  
## 7 spec binary 0.999 Preprocessor1\_Model1  
## 8 roc\_auc binary 0.975 Preprocessor1\_Model1

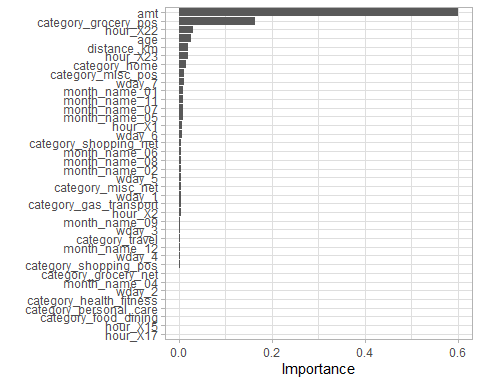
#Compare to training  
xgb\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.996 3 0.000582 Preprocessor1\_Model1  
## 2 f\_meas binary 0.663 3 0.0354 Preprocessor1\_Model1  
## 3 kap binary 0.661 3 0.0357 Preprocessor1\_Model1  
## 4 precision binary 0.792 3 0.0644 Preprocessor1\_Model1  
## 5 recall binary 0.571 3 0.0196 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.957 3 0.00518 Preprocessor1\_Model1  
## 7 sens binary 0.571 3 0.0196 Preprocessor1\_Model1  
## 8 spec binary 0.999 3 0.000362 Preprocessor1\_Model1

## Get variable importance using vip package

library(vip)  
  
last\_fit\_xgb %>%   
 pluck(".workflow", 1) %>%   
 pull\_workflow\_fit() %>%   
 vip(num\_features = 100) +  
 theme\_light()

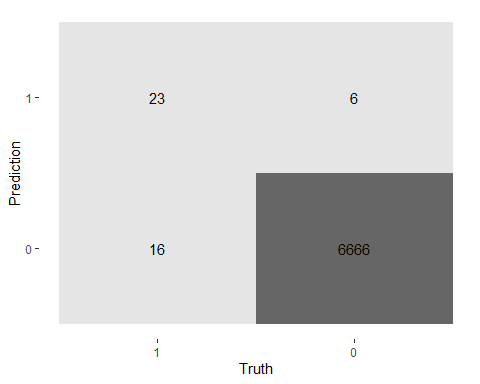
## Warning: `pull\_workflow\_fit()` was deprecated in workflows 0.2.3.  
## ℹ Please use `extract\_fit\_parsnip()` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.



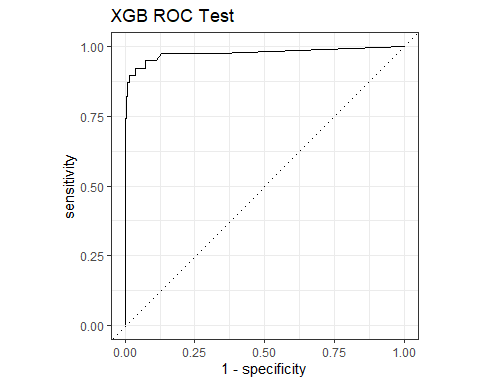
## Amount and category\_grocery are the most important features,   
## followed by age and whether the transaction was made between 10-11pm

## Plot Final Confusion matrix and ROC curve

## Final Confusion Matrix  
  
last\_fit\_xgb %>%  
 collect\_predictions() %>%   
 conf\_mat(is\_fraud, .pred\_class) %>%   
 autoplot(type = "heatmap")



## Final ROC curve  
last\_fit\_xgb %>%   
 collect\_predictions() %>%   
 roc\_curve(is\_fraud, .pred\_1) %>%   
 autoplot() +  
 labs(title="XGB ROC Test")



#Assessing which model affords the best classification from these results:  
#  
# First, considering the results from each model in turn:   
#  
# The logistic regression (LR) performs reasonably, as judged by the fact that the ROC is decent (0.83) although the model has low sensitivity while having high specificity. Meanwhile, the precision and recall (and thus F-score) metrics are very low (in the 30% territory); so there is little discrimination between any elements in the confusion matrix; no particular fold is far better than any other. So, this is not a good model.   
#  
# The decision tree model fares slightly worse in the ROC metric (0.78) but it has better sensitivity than the LR model. The evaluation metrics are also better, e.g. precision (0.65) and recall (0.47). This suggests that the model does fit some of the data with a priori information better.  
#  
# The random forest results compare more favourably, being the best of all models tested in terms of the ROC result (0.94). The model has perfect precision but almost zero recall and thus a poor F-score. It also has poor sensitivity.  
#  
# The bootstrapping decision-tree option, gradient boosting, fares similarly to that of the random forest model (ROC = 0.93), but overall evaluation metrics look a lot better and are more balanced. For example, the precision and recall are 69 and 46% respectively. The sensitivity (46%) and specificity (100%) are also much better and more balanced. So the Gradient boosting method results look particularly good.  
#  
# the k-nearest neighbour classification model does not perform so well (ROC = 0.71); it is derived from good specificity (100% = perfect) but low sensitivity (21%). Meanwhile, the statistical figure-of-merits, precision (30%) and recall (21%) are very low. So, this is not a good model.  
#  
# In summary, the gradient boosting model is clearly the best model. Others suffer in various regards, in particularly some are very imbalanced between precision versus recall, and/or sensitivity versus specificity. It makes sense that the gradient boosting (bootstrapping) model offers the best fit to the training data. This is because the random forest model will train many trees independently while a gradient boosting model will train many trees subsequently (correcting the errors in a given tree from that of its forerunner tree). Thus, the gradient boosting model has bootstrapping functionality, and its results are easier to interpret than that of a random forest model because gradient boosting models are afforded from one final tree while the result of a random forest model are the ensemble of many trees that have been modelled in parallel. Given these fundamental model considerations, and the similar ROC of these two model options, the gradient boosting model was deemed to be the best model for being taken forward in classifying the test (unseen) movie data.   
#  
#The results of applying these test data to the gradient boosting model are good, albeit less good than the results from the training data. This comparison stands to reason since unseen data will naturally fit less well than the training data which were used to create (i.e. were tailored to the fit of) the model. Besides, the results of testing and training data still fare well in having fairly small overall differences.  
#

## Calculating the cost of fraud to the company

* How much money (in US$ terms) are fraudulent transactions costing the company? Generate a table that summarizes the total amount of legitimate and fraudulent transactions per year and calculate the % of fraudulent transactions, in US$ terms. Compare your model vs the naive classification that we do not have any fraudulent transactions.

best\_model\_wflow <- xgb\_wflow  
  
best\_model\_preds <-   
 best\_model\_wflow %>%   
 fit(data = card\_fraud\_train) %>%   
   
 ## Use `augment()` to get predictions for entire data set  
 augment(new\_data = card\_fraud)  
  
best\_model\_preds %>%   
 conf\_mat(truth = is\_fraud, estimate = .pred\_class)

## Truth  
## Prediction 1 0  
## 1 2518 472  
## 0 1418 666620

cost <- best\_model\_preds %>%  
 select(is\_fraud, amt, pred = .pred\_class)   
  
cost <- cost %>%  
 mutate(  
 false\_naives = if\_else((is\_fraud ==1),amt,0),   
 # naive false-- we think every single transaction is ok and not fraud. We lose the amount of the transaction if its fraud  
  
 false\_negatives=if\_else((is\_fraud ==1)&(pred ==0),amt,0),  
 # false negatives-- we thought they were not fraud, but they were. We lose the amount of transaction if its fraud  
   
 false\_positives=if\_else((is\_fraud ==0)&(pred ==1),amt \*0.02,0),  
 # false positives-- we thought they were fraud, but they were not. We lose the amount of the transaction \* 0.02 (our margin)  
  
 true\_positives=0,  
 # true positives-- we thought they were fraud, and they were   
   
 true\_negatives=0  
 # true negatives-- we thought they were ok, and they were   
)  
   
# Summarising  
cost\_summary <- cost %>%   
 summarise(across(starts\_with(c("false","true", "amt")),   
 ~ sum(.x, na.rm = TRUE)))  
  
cost\_summary

## # A tibble: 1 × 6  
## false\_naives false\_negatives false\_positives true\_positives true\_negatives  
## <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 2075089. 617323. 7578. 0 0  
## # ℹ 1 more variable: amt <dbl>

print(scales::dollar(cost\_summary$false\_naives - cost\_summary$false\_negatives - cost\_summary$false\_positives))

## [1] "$1,450,188"

#Implementing our model will save money  
# Calculated by doing False Naive cost - (False negative cost + false positive cost)

* If we use a naive classifier thinking that all transactions are legitimate and not fraudulent, the cost to the company is $2,075,089.
* With our best model, the total cost of false negatives, namely transactions our classifier thinks are legitimate but which turned out to be fraud, is $617,323.
* Our classifier also has some false positives, $7,578.15, namely flagging transactions as fraudulent, but which were legitimate. Assuming the card company makes around 2% for each transaction (source: <https://startups.co.uk/payment-processing/credit-card-processing-fees/>), the amount of money lost due to these false positives is $151.56
* The $ improvement over the naive policy is $1,457,614.