exam 1 - stats 0218

Exam 1

Question 1.

At a simple glance, we can see that electric-type predictions (65,442) are much lower compared to the other three types. Fire is almost double, Grass is triple, and Water is about six times as many as Electric, so those three types overshadow Electric on the map. But that is not the only reason; there is a caveat. k1 table shows a huge number of Grass predictions overall, but the heatmap itself does not reflect that!

The heatmap is only showing two dimensions (Attack and Defense), but our kNN model is actually using four (HP, Attack, Defense, Speed). The model considers every combination of these four features and, because we didn't specify k, the class package seems to default to k = 1, so it picks the single closest neighbor in 4D space. However, when we collapse everything into a 2D heatmap, we're essentially grouping all HP–Speed combinations under each (Attack, Defense) pair. The plot then displays only the most frequent predicted type for each (Attack, Defense). For instance, if Attack = 50 and Defense = 50, the model checks all possible HP and Speed values at that coordinate and usually finds Water is predicted most often, hence the tile is labeled Water. Because Electric never comes out on top for any (Attack, Defense) bin, it gets overwritten in the 2D view, even though it's predicted many times overall in absolute terms. And the same logic explains why Grass doesn't appear as often as its raw count suggests because it's simply outvoted in each tile, but less often than Electric.

Just a side note, aren't we supposed to split the initial dataset into training and testing? It looks like the code given to us uses the whole dataset as training data. Or is it doing that in the background?

Question 2.

a. Absolutely Terrible

Let's say Bank of Amuna uses a kNN model to predict whether a loan applicant will default. 1 = Yes default (5% of observations) 0 = No default (95% of observations)

The model uses variables like credit score, income, and employment status to compare applicants and assigns them a group based on their 'k' nearest neighbors. The model achieves 50% overall accuracy, which might sound reasonable at first glance, but is actually worse than doing nothing. Since 95% of applicants don't default, simply predicting "no default" for everyone would already give 95% accuracy. But obviously, what's the point if we just predict no default for everyone? That doesn't help the bank make decisions or manage risk. For the kNN model to drop all the way to 50%, it must be misclassifying a huge number of both reliable and risky borrowers. This also suggests that the variables we're using don't help the model group the defaulters together. Instead, they're scattered and mixed in with non-defaulters throughout the feature space. If the k value is very low, like k = 1, the model is extremely sensitive to noise. Setting k = 50 could help smooth out some of that noise, but it still doesn't solve the core problem. The model can't distinguish defaulters well because they're rare and spread out. Even with 50 neighbors, a defaulter is likely to be surrounded by mostly non-defaulters, so they get

misclassified. On the other side, some reliable borrowers might get flagged as risky just because of a few nearby outliers from Group 1.

As a bank, one of my main objectives is to lend to reliable borrowers, so I need a model that helps clearly identify safe applicants. But this model misclassifies many defaulters as safe, which will make my bank suffer financial loss (the amount depends on the loan amount). The other objective is to accurately find good borrowers since my bank is a business that needs to make money. If we falsely deny reliable applicants, we lose business and lose reputation in the market.

Even with the most optimal chosen k, kNN performs poorly in this setting because of the imbalance data and the lack of clear separation between the groups. Therefore, this model is absolutely terrible for my bank's objectives in real life.

b. Very Mediocre

Let's say Amuna's Animal Shelter uses an LDA model to predict whether a visitor will adopt a dog. 1 = Yes adopt (5% of the observations) 0 = No adopt (95% of the observations)

The model uses variables like time spent in the shelter, number of previous visits, and age group to make predictions. LDA assumes the data for both groups are normally distributed, and it tries to maximize the distance between the means of the two groups (adopters and non-adopters), while also minimizing the variation within each group. LDA can be considered okay in this situation because it uses prior probabilities when making predictions, so it's aware of the 95–5 imbalance and factors that into the classification. Even with only 50% accuracy, it's at least making somewhat informed decisions. And even if the data isn't perfectly normal, LDA might still capture the general trend of each group and build a clean linear rule, which isn't completely useless. That said, the model still has issues. Its linear boundary isn't flexible enough to capture the real complexity of adoption behavior. And with only 5% of the data belonging to adopters, it's still heavily biased toward predicting non-adoption. So even when someone's features resemble an adopter, they might still get classified as a non-adopter simply because the model expects almost everyone to be one. Adjusting the threshold from its default 0.5 to something lower (like 0.1 or 0.2) might improve the model's ability to detect actual adopters, but would also increase false positives. Therefore, this model is mediocre.

c. Quite Impressive

I just read some news about Luigi Mangione.

Let's say a team of Middlebury students uses a tree-based model to predict whether a defendant will be sentenced to death or not in capital punishment cases. 1 = Death sentence (5% of the cases) 0 = No death sentence (95% of the cases)

The model uses features like the type of crime, state of trial, prior convictions, defendant and victim race, and jury composition. Our goal is to understand how structural factors affect the decisions and the relationship between these factors. What makes the tree-based model impressive is how well it handles the complexity of this data. There are essentially two key components to building a good decision tree model: - deciding which features to split on, and - deciding when to stop splitting. The model chooses splits based on which variables best separate the outcomes. For example, it may learn that in certain

states, a specific combination of crime type, jury makeup, and race variables strongly predicts a death sentence. These interactions are hard to detect with simpler models like LDA, which assumes linear boundaries, or kNN, which struggles when the rare group is overwhelmed by the majority. In our case, only 5% of defendants receive the death sentence, meaning the model has to learn from a very small number of positive cases. With such a large imbalance (95-5), the tree is at high risk of overfitting because it might keep splitting the data until it creates extremely small, overly specific branches just to fit the few death sentence cases. But by pruning the tree, it discourages the model from building fragile rules around a very small number of cases, and instead prioritizes patterns that generalize well and highlight differences between the groups. Even if the overall accuracy is just 50%, the tree may still do a much better job of identifying actual death sentence cases than other models. And because our purpose isn't just to make perfect predictions, the model also lets us see clearly what factors are driving those predictions. That said, we also recognize that the cost of misclassifying a death sentence case is extremely high, which is why this model is not intended for use in making real-time legal decisions. Instead, its value lies in its transparency/audit, where the tree allows us to clearly see which features and combinations are associated with the most serious sentencing outcomes. In that sense, the model is quite 'impressive' in informing us about the fairness, bias, and inequality.

(Overall, the quality of these models seem very subjective and depend on what the person building this model prioritizes)

Question 3.

Predict a stroke.

```
library(tidyverse)
```

```
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ dplyr     1.1.4     ✔ readr     2.1.5
✔ forcats   1.0.0     ✔ stringr   1.5.1
✔ ggplot2   3.5.1     ✔ tibble    3.2.1
✔ lubridate 1.9.4     ✔ tidyr     1.3.1
✔ purrr     1.0.2
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to
become errors
```

```
library(rpart)
library(rpart.plot)
stroke_data <- read_csv("~/Downloads/stroke_data.csv")
```

```
Rows: 5110 Columns: 12
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (6): sex, ever_married, work_type, Residence_type, bmi, smoking_status
dbl (6): id, age, hypertension, heart_disease, avg_glucose_level, stroke
```

```
ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Let's explore the variables in our dataset especially with respect to how much each variable is correlated with incidence of stroke . There are 8 categorical and 3 continuous variables in addition to the patient identified ID.

Categorical:

```
stroke_binary <- stroke_data |>
  mutate(stroke_binary = case_when(stroke == 1 ~ 'yes',
         stroke == 0 ~ 'no',
         TRUE ~ NA))

table(stroke_binary$sex, stroke_binary$stroke_binary)
```

```
          no  yes
  Female 2853  141
  Male   2007  108
  Other     1    0
```

```
#sex
stroke_binary |>
  group_by(sex) |>
  summarise(
    stroke_yes = sum(stroke_binary == "yes", na.rm = TRUE),
    stroke_no = sum(stroke_binary == "no", na.rm = TRUE),
    total = stroke_yes + stroke_no,
    stroke_rate = stroke_yes / total
  )
```

```
# A tibble: 3 × 5
  sex    stroke_yes stroke_no total stroke_rate
  <chr>       <int>     <int> <int>       <dbl>
1 Female        141      2853  2994      0.0471
2 Male          108      2007  2115      0.0511
3 Other           0         1     1      0
```

```
#hypertension
stroke_binary |>
  group_by(hypertension) |>
  summarise(
    stroke_yes = sum(stroke_binary == "yes", na.rm = TRUE),
    stroke_no = sum(stroke_binary == "no", na.rm = TRUE),
    total = stroke_yes + stroke_no,
    stroke_rate = stroke_yes / total
  )
```

```
# A tibble: 2 × 5
  hypertension stroke_yes stroke_no total stroke_rate
         <dbl>      <int>     <int> <int>       <dbl>
1            0        183      4429  4612      0.0397
2            1         66       432   498      0.133
```

```
#heart disease
stroke_binary |>
  group_by(heart_disease) |>
  summarise(
    stroke_yes = sum(stroke_binary == "yes", na.rm = TRUE),
    stroke_no = sum(stroke_binary == "no", na.rm = TRUE),
    total = stroke_yes + stroke_no,
    stroke_rate = stroke_yes / total
  )
```

```
# A tibble: 2 × 5
  heart_disease stroke_yes stroke_no total stroke_rate
          <dbl>      <int>     <int> <int>       <dbl>
1             0        202      4632  4834      0.0418
2             1         47       229   276      0.170
```

```
#ever married
stroke_binary |>
  group_by(ever_married) |>
  summarise(
    stroke_yes = sum(stroke_binary == "yes", na.rm = TRUE),
    stroke_no = sum(stroke_binary == "no", na.rm = TRUE),
    total = stroke_yes + stroke_no,
    stroke_rate = stroke_yes / total
  )
```

```
# A tibble: 2 × 5
  ever_married stroke_yes stroke_no total stroke_rate
  <chr>             <int>     <int> <int>       <dbl>
1 No                   29      1728  1757      0.0165
2 Yes                 220      3133  3353      0.0656
```

```
#work type
stroke_binary |>
  group_by(work_type) |>
  summarise(
    stroke_yes = sum(stroke_binary == "yes", na.rm = TRUE),
    stroke_no = sum(stroke_binary == "no", na.rm = TRUE),
    total = stroke_yes + stroke_no,
    stroke_rate = stroke_yes / total
  )
```

```
# A tibble: 5 × 5
  work_type     stroke_yes stroke_no total stroke_rate
  <chr>              <int>     <int> <int>       <dbl>
1 Govt_job              33       624   657     0.0502
2 Never_worked           0        22    22     0
3 Private              149      2776  2925     0.0509
4 Self-employed         65       754   819     0.0794
5 children               2       685   687     0.00291
```

```
#residence
stroke_binary |>
  group_by(Residence_type) |>
  summarise(
    stroke_yes = sum(stroke_binary == "yes", na.rm = TRUE),
    stroke_no = sum(stroke_binary == "no", na.rm = TRUE),
    total = stroke_yes + stroke_no,
    stroke_rate = stroke_yes / total
  )
```

```
# A tibble: 2 × 5
  Residence_type stroke_yes stroke_no total stroke_rate
  <chr>              <int>     <int> <int>       <dbl>
1 Rural                114      2400  2514      0.0453
2 Urban                135      2461  2596      0.0520
```

```
#smoking status
stroke_binary |>
  group_by(smoking_status) |>
  summarise(
    stroke_yes = sum(stroke_binary == "yes", na.rm = TRUE),
    stroke_no = sum(stroke_binary == "no", na.rm = TRUE),
    total = stroke_yes + stroke_no,
    stroke_rate = stroke_yes / total
  )
```

```
# A tibble: 4 × 5
  smoking_status  stroke_yes stroke_no total stroke_rate
  <chr>                <int>     <int> <int>       <dbl>
1 Unknown                 47      1497  1544      0.0304
2 formerly smoked         70       815   885      0.0791
3 never smoked            90      1802  1892      0.0476
4 smokes                  42       747   789      0.0532
```

I know this is such an inefficient way to do this as there must be a function (for-loop) to call each categorical variable to compute the same thing essentially, but I will come for office hours for that, but this crude way should give me enough ideas about the relationship between stroke rate and each categorical variable.

From the tables, we can see that hypertension, heart disease, and whether someone previously smoked or smokes have higher rate of stroke.

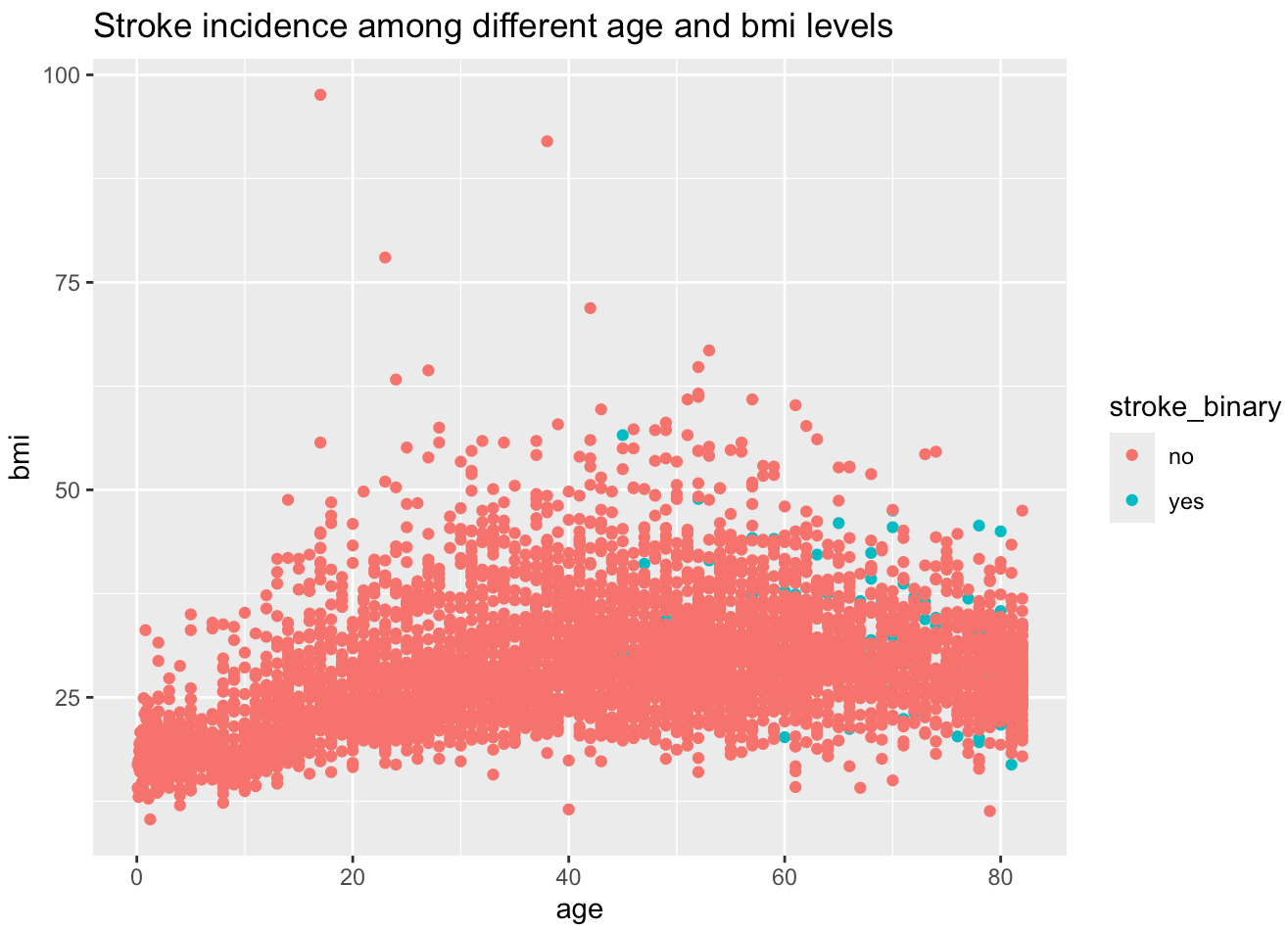
Numerical:

```
stroke_binary |>
  ggplot() +
  geom_point(aes(y=avg_glucose_level, x=age, color = stroke_binary)) +
  labs(title = "Stroke incidence among different age and glucose levels",
       x = "age",
       y = "average glucose level")
```


```
stroke_binary$bmi <- as.numeric(stroke_binary$bmi)
```

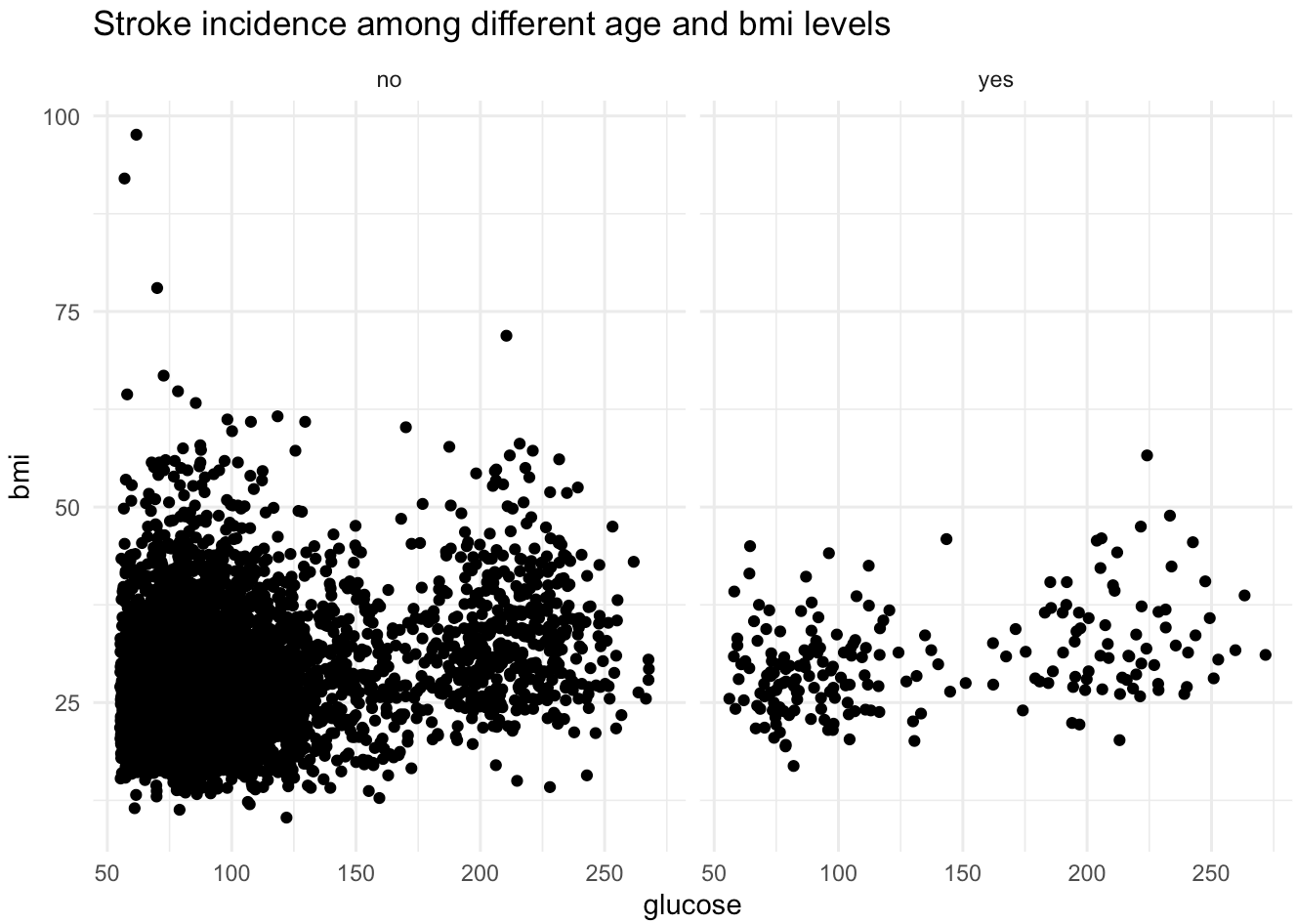
```
Warning: NAs introduced by coercion
```

```
stroke_binary |>
  filter(!is.na(bmi)) |>
  ggplot() +
  geom_point(aes(y=bmi, x=age, color = stroke_binary)) +
  labs(title = "Stroke incidence among different age and bmi levels",
       x = "age",
       y = "bmi")
```




```
stroke_binary |>
  ggplot() +
  geom_point(aes(x = avg_glucose_level, y = bmi)) +
  facet_wrap(~ stroke_binary) +
  labs(
    title = "Stroke incidence among different age and bmi levels",
    x = "glucose",
    y = "bmi"
  ) +
  theme_minimal()
```

```
Warning: Removed 201 rows containing missing values or values outside the scale range
(`geom_point()`).
```



So from these visualizations, we can see that older age (age>40) has a high correlation with stroke rate.

Therefore, I chose age, hypertension, heart disease, and smoking habits as my explanatory variables because they showed clear differences in stroke rates during my earlier summaries, and they also make sense medically. To me, the 'best' model is the one that gives the highest accuracy, so I went with a random forest. It builds lots of decision trees using random samples of the data, and at each split, it randomly picks which features to consider. That randomness makes the trees more diverse, which helps the model perform better overall. It's more flexible than just using one tree, and it's better at picking up on complicated patterns. So for this dataset, random forest felt like the strongest option.

```
library(randomForest)
```

```
randomForest 4.7-1.2

Type rfNews() to see new features/changes/bug fixes.


Attaching package: 'randomForest'

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

    combine
```

```
The following object is masked from 'package:ggplot2':

    margin
```

```
library(ranger)
```

```
Attaching package: 'ranger'

The following object is masked from 'package:randomForest':

    importance
```

```
library(janitor)
```

```
Attaching package: 'janitor'

The following objects are masked from 'package:stats':

    chisq.test, fisher.test
```

```
set.seed(1)
training_data_rows <- sample(1:nrow(stroke_binary),
                             size = nrow(stroke_binary)/2)

training_stroke <- stroke_binary[training_data_rows, ]
testing_stroke <- stroke_binary[-training_data_rows, ]

#train our dataset
rf_1 <- randomForest(factor(stroke_binary) ~ age + hypertension + `heart_disease` + `smok
                     data = training_stroke,
                     mtry = 4,
                     importance = TRUE,
                     classwt = c("no" = 1, "yes" = 10)
                     )

rf_1
```

```
Call:
 randomForest(formula = factor(stroke_binary) ~ age + hypertension +      heart_disease +
smoking_status, data = training_stroke, mtry = 4,      importance = TRUE, classwt = c(no
= 1, yes = 10))
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4
```

```
        OOB estimate of  error rate: 16.79%
Confusion matrix:
      no yes class.error
no  2072 352   0.1452145
yes   77  54   0.5877863
```

```
#make predictions
predictions <- predict(rf_1, testing_stroke)

confusion_matrix <- table(Predicted = predictions, Actual = testing_stroke$stroke_binary)
confusion_matrix
```

```
         Actual
Predicted   no  yes
      no  2094   68
      yes  343   50
```

```
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix) #I asked chatgpt to compu
accuracy
```

```
[1] 0.8391389
```

My initial model had an overall accuracy of about 95%, but it only correctly identified 7 people who had strokes and missed 111 actual stroke cases. Not good! Even though stroke isn't extremely rare, it's still uncommon enough that the model leaned heavily toward predicting the majority class(no stroke). Since random forests prioritize overall accuracy, they tend to play it safe and predict the more frequent outcome. To address this, I increased the weight for stroke cases by 10 times, which helped realize what's more valuable. As a result, the updated model caught 50 stroke cases, which is an improvement of 43 more correct predictions compared to the original model. Most importantly, that means we were able to identify many people who might have otherwise been completely missed; however, the new model accuracy is lower at about 84%.

Although higher numbers of people were wrongly predicted to have a stroke (343), which certainly have caused them emotional and financial distress, it is better to live knowing that you thought you would have one, but end up not having one. I am extremely sorry about the 68 people who were missed by this model.

Question 4.

Following the same logic as Question 3, I will use my random forest model to answer Question 4. In the context of stroke prediction, missing someone who actually has a stroke (false negative) is 10 times more costly than incorrectly warning someone (false positive). It's better for the model to be overly cautious and raise some false alarms if it means we catch more true stroke cases early, which is consistent with my model from Question 3. However, I further want to reduce the number of false negatives, so I will set a threshold to make the predictions more accurate to lower costs.

By adjusting the threshold to 0.001, my model now catches 92 out of 118 stroke cases, which is again a massive improvement. Of course, it does this by flagging many more people who didn't end up having a stroke, but in a C/10C scenario where missing a true stroke is far more costly than a false alarm, this trade-off makes sense. We want to be as cautious as possible.

The question 3 model cost is 343C + 10C*68=1023C This threshold model cost is 716C + 10C*26= 976C

This shows that our question 4 threshold-adjusted model here is less costlier.

```
probs <- predict(rf_1, testing_stroke, type = "prob")

threshold <- 0.001

preds <- ifelse(probs[, "yes"] > threshold, "yes", "no") #chatgpt helped me here

table(Predicted = preds, Actual = testing_stroke$stroke_binary)
```

```
         Actual
Predicted   no  yes
      no  1721   26
      yes  716   92
```

```
mean(preds == testing_stroke$stroke_binary)
```

```
[1] 0.709589
```