# MT7049 project 4

Max Brehmer, Florian John

07/01/2024

## 1 Introduction

Elements of statistical learning [3] presents multiple machine learning methods that can be used to perform classification and regression on data where different methods are more suitable depending on data appearance. In this project we analyse data collected from the UCI Machine Learning Repository and make a comparison between the support vector machine and random forest techniques on a binary classification problem on a data-set describing whether a person will accept a coupon recommended to him in different driving scenarios. Toward the end of the report we present results comparing performance of the random forest with different parameter choices such as number of trees, tree depth and removal of correlating features the model.

## 2 Data

### 2.1 Description of data-set

We decided to work with a data-set from the UCI Machine learning repo called "in-vehicle coupon recommendation" [1]. This data-set contains 26 features providing information about the driver, such as their destination, gender or age, as well as descriptions of the occasion such as the weather and temperature, or the type of coupon on offer.

Three out of the 26 features are removed from the data-set. These features are "car" describing the brand of car being driven, "direction_opp" which is the inverse feature to "direction_same", making it an obsolete and unnecessary parameter. Finally we removed "toCoupon_GEQ5min" which tells us whether or not the location of the coupon usage is less than 5 minutes away by car. This feature contains almost exclusively zeroes and very few ones, potentially leadning to overfitting and an imbalanced feature influence.

The remaining 23 features used in this project are described in table 1.

Table 1: Description of data-set parameters

| **Variable Name** | **Type** | **Description** |
|---|---|---|
| Destination | CHR | No Urgent Place, Home, Work |
| Passanger | CHR | Alone, Friend(s), Kid(s), Partner |
| Weather | CHR | Sunny, Rainy, Snowy |
| Temperature | INT | Temperature in degrees Farenheit (30, 55, 80) |
| Time | CHR | Morning, Noon, Afternoon, Evening, Night |
| Coupon | CHR | Type of coupon: Bar, Coffee House, Take-away, Restaurant($\leq$\$20), Restaurant(\$20-\$50) |
| Expiration | CHR | Expiration time (1d, 2h) |
| Gender | CHR | Female, Male |
| Age | CHR | Below 21, 21 to 50, Above 50 |
| MaritalStatus | CHR | Unmarried partner, Single, Married partner, Divorced, Widowed |
| has_children | INT | Parent to at least one child? (No = 0, Yes = 1) |
| Education | CHR | Some High School, High School Graduate, Some college, Bachelors degree, Associates degree, Graduate degree (MSc or PhD) |
| Occupation | CHR | List of 25 occupations (incl. Unemployed, Student, Retired, Legal, Healthcare support etc.) |
| Income | INT | 9 Levels of income ranked on a numeric scale between 0-8. |
| bar_gt1 | INT | Visits Bar more than once per month (No = 0, Yes = 1) |
| coffee_gt1 | INT | Visits Coffee House more than once per month (No = 0, Yes = 1) |
| takeaway_gt1 | INT | Gets Take Away food more than once per month (No = 0, Yes = 1) |
| restaurantless20_gt1 | INT | Visits Restaurant with avg expense $\leq$\$20 more than once per month (No = 0, Yes = 1) |
| restaurant20plus_gt1 | INT | Visits Restaurant with avg expense \$20-\$50 more than once per month (No = 0, Yes = 1) |
| toCoupon_GEQ15min | INT | Driving distance to the restaurant/bar for using the coupon is 15< minutes (No = 0, Yes = 1) |
| toCoupon_GEQ25min | INT | Driving distance to the restaurant/bar for using the coupon is 25< minutes (No = 0, Yes = 1) |
| direction_same | INT | Restaurant/bar same direction as destination (No = 0, Yes = 1) |
| accept_coupon | INT | Coupon accepted? (No = 0, Yes = 1) |

## 2.2 Adaptation of dataset

Some of the 23 remaining parameters may be significantly correlated, leading to multicollinearity, which can induce problems for any machine learning algorithm, such as overfitting, an imbalanced feature influence and additional computation times using redundant features. In section 4 we discuss the cor-

relation and influence of all parameters in the data-set in order to maximize accuracy and arrive at a final model.

A transformation of some parameters was also necessary.

- The **Temperature** variable remains a three way outcome variable, but is transformed from a categorical variable where each category is independent to a numeric scale where it is assumed that the names of the categories represent the midpoint of a certain temperature category, thus all temperatures are treated as the numeric value (30, 55, 80) which the given observation is closest to.
- The times listed in the original data-set are similarly to the temperature variable, categorized by the time of day (by selected hours) of which an observation is closest. Since **Time** is not on a numeric (lower $->$ higher) scale, we cannot make a linear numeric tranformation. Instead the categories are named by the approximate time of day.
- **Age** remains a categorical category, but was reduced from 8 categories to 3, in which we now look at whether the driver is below 21 (student), above 50 (retired or close to retired) or somewhere in between (working age).
- We transformed the **income** variable from a categorical variable with 9 levels of income into a linear integer variable where each income level is assigned a numeric value in order of where on the income ladder the driver is placed.
- **bar_gt1** and all the other variables determining habits of eating/drinking out were transformed from 5 categories to binary integer variables. Any driver going to a bar (and subsequently coffee houses, take-aways etc.) more than once a month, are considered regular customers, while people who go once a month or less are not considered regular customers.

# 3 Analysation methods

## 3.1 Choice of methods

### 3.1.1 Support vector machines

Support vector classifiers is a method that uses linear boundaries to separate classes. It can also be generalized to include non linear boundaries which is a method named Support vector machines, that is used for data where categorization can't easily be determined by a linear boundary and where it might be more appropriate with non-linearity. [3, p. 423]

In order to select a margin that allows separation of our categorical data, we want to solve an optimization problem where we want to find the hyperplane in

feature space which allows the biggest margin possible between points. For this purpose we want to perform the kernel trick which is a method that is used to compute inner product in a transformed space $K$ which then is used to measure the distance between points in $K$. Using a suitable kernel we can perform this optimization and for choices of kernel we could select linear, polynomial and radial depending on the appearance of our data. [3, p. 423-424]

For the choice of cost parameter we note that a high value of C (cost parameter) might cause over-fitting while a small c such as $C = 0.1$ will have smooth boundaries. The value chosen for $C$ affects the margin size for the decision border which results in a higher focus on correctly classified points near the border when $C$ is larger and smaller $C$ involves data further away from that border. In order to Find an optimal $C$ we may use cross validation. [3, p. 424]

Support Vector Machines are a simple and efficient model when working with data that can be quantified. However, if the data is categorical, the method doesn't work so well. This is because finding a boundary vector necessitates that all values can be placed on a numerical hyperplane. With just one observation in the "wrong" category, the boundary width is guaranteed to be zero, thus we may look for a method more suitable for data with both numeric and categorical parameters.

### 3.1.2 Random Forests

A more suitable method when working with mixed parameter types (categorical & numeric) are decision trees and more specifically the Random Forest algorithm. This method is not affected by the issue of placing categorical values on a numeric hyperplane. The random forest algorithm is a machine learning algorithm based on decision trees. A single decision tree is simple and, given enough depth, a relatively unbiased algorithm. Yet, there are limitations since the process of classifying data based on a sequence of parameters can lead to poor predictions due to sub-optimal variance and flexibility.

In order to capture more complex interactions, an alternative method based on the same principles as decision trees is useful. The random forest algorithm uses a process called bagging (bootstrapping + aggregation) to lower the variance of the classification model. The main idea behind bagging is to average many noisy but approximately biased models in order to reduce variance. A great benefit of using bagging is that the expectation of an average of $B$ trees is the same as the expectation of any individual tree. This means that the bias of bagged trees is same as the individual (bootstrapped) trees.

Bootstrapping is a random sampling process using replacement. An observation in the data-set is sampled at random and placed first in the sampled list. This process is repeated as many times as the amount of observations in the data-set. Replacement is necessary in order to ensure diversity among the

different trees. This means that some observations are selected more than once and some (roughly 1/3 of observations) are not selected at all. These observations are called out-of-bag (OOB) samples and function as a validation set for the constructed decision tree.

When all **[ntrees]** decision trees have been sampled. The OOB samples are used to validate each tree. Each tree produces a class (0 or 1) as output. The next step is aggregation, in which majority voting takes place.

The most important parameters in the random forest model are

- The number of trees ('ntrees' = 500 default)
- Number of features considered at each split ('mtry' $\approx \sqrt{\text{nr of features}}$)

The most common parameter values are shown in brackets. There are of course more parameters but these have the biggest impact on the models' accuracy.
[3, p. 587-588]

### 3.2 Cramer's V

Cramer's V is a method that uses Pearson's chi squared test in order to measure correlation between categorical features [2, p. 288]. The Cramer's V for two features is

$$V = \sqrt{\frac{\chi^2/n}{\min(c-1, r-1)}}$$

where $\chi^2$ is The Chi-square statistic, $n$ is the total sample size and $c$ and $r$ are the number of columns and rows respectively. Now $0 \leq V \leq 1$ indicates level of correlation. The scale of measurability is therefore between 0 and 1, where 0 is no correlation and 1 indicates perfect correlation [4].

## 4 Methods chosen for data

We choose to incorporate the random forest method on our data since it's suitable for mixed data types, and since our data contains both categorical, numbered and binomial data we opted to reject use of the SVM since it's mostly suitable for numeric data. The random forest method however is susceptible to bias due to correlation between features in data since the random choice of tree might have an increased choice of choosing one features influence over others if two features are correlated.

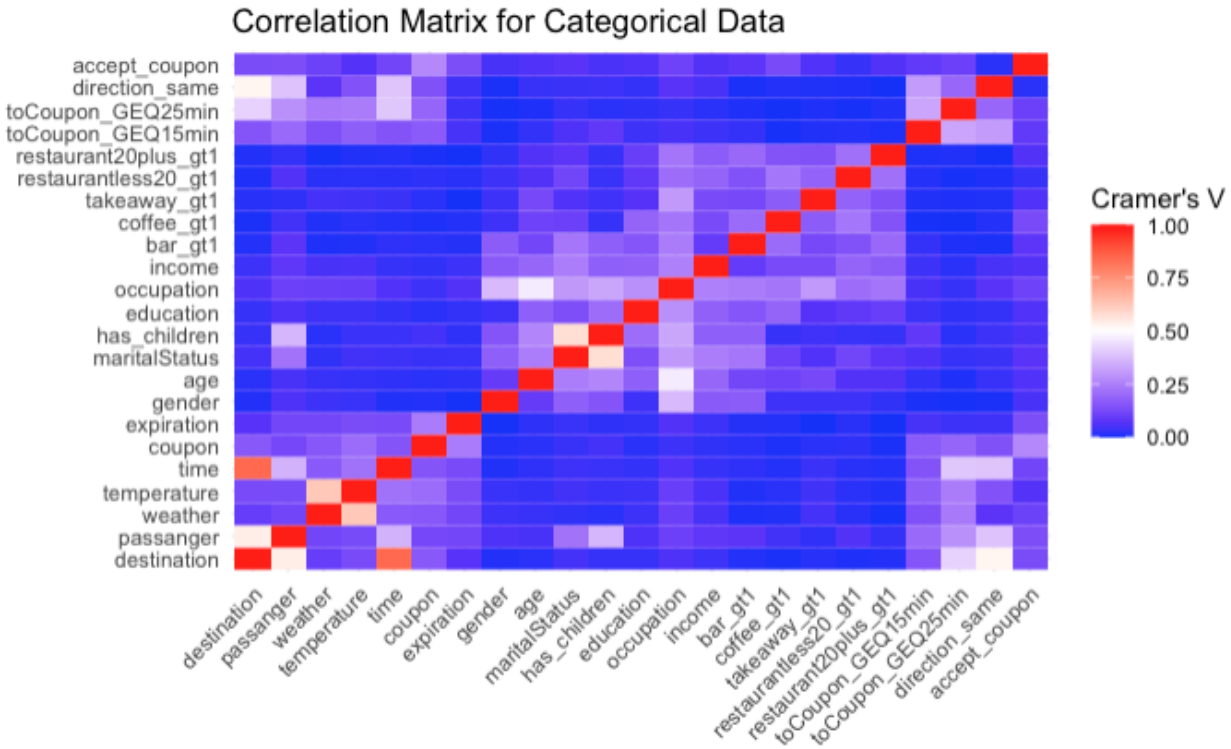


Figure 1: Plot of Cramer's V between features

## 4.1 Correlation between features

In order to avoid bias when performing random forest we evaluate correlation between our categorical data using a plot containing Cramer's V.

From figure 1 we observe that we have five correlated pairs with Cramer's V $\geq 0.5$ most notably Time $\sim$ destination, temperature $\sim$ weather and has_children $\sim$ maritalStatus. Principal Component Analysis (PCA) [3, p. 547-548] is in this case conventionally used in relation to random forests in-order to eliminate bias from correlating features. But due to it's incompatibility with non-continuous data we chose to instead perform cross validation where we remove features with $V \geq 0.5$ as a dimensionality reduction technique instead.

## 4.2 Implementation of Random Forests

After dimensionality reduction and parameter selection we have removed the **Temperature**, **Time**, **Age**, **has_children** features and performed the random forest algorithm for 200 trees with 4 splits per node.
Now describing how the random forest algorithm was performed for our choice

| Features removed | Avg accuracy | Avg kappa |
|---|---|---|
| base_model | 0.5833 | 0.0941 |
| passanger+temperature+time+age+has_children | 0.6036 | 0.1379 |
| temperature+time+age+has_children | 0.6202 | 0.1776 |
| time+age+has_children | 0.5893 | 0.0990 |
| age+has_children | 0.6262 | 0.1749 |
| has_children | 0.6036 | 0.1269 |
| full_model | 0.6065 | 0.1432 |

Table 2: Model performance metrics

of parameters we have the following. The algorithm involves creating $B = 200$ decision trees based of bootstrapped data. We start by drawing a Bootstrap a sample $\mathbf{Z}^b$ of size $N = 12079$ by randomly sampling with replacement from our coupon data. We then grow a decision tree $T_b$ to $\mathbf{Z}^b$ for $b = 1, 2, .., B$. Now for each terminal node of the tree we Select $\mathbf{[Mtry = 4]}$ features from our $p = 22$ available features. Then we select the best feature from $m$ and the create two daughter nodes and repeat the process until tree of maximum node depth of 4 is reached. This process is repeated by creating new decision trees for each bootstrap sample.
The algorithm is in our case performed on a subset that is 1% of our full data-set in order to reduce complexity. After we have $\{T_1, .., T_B\}$ trees the algorithm is is complete [3, p. 588].

# 5 Results

In this section we present results for accuracy tests conducted from varying values of [**ntrees**] and [**Mtry**] in random forest implementations. We also show tables depicting test accuracy for reductions in dimensionality.

In order to measure performance one can make predictions using new data. For our new data we separate data where 20% is dedicated for test data and the rest for training.

## 5.1 Choice of parameter

Results for different numbers of trees show that model accuracy improvement stagnates for values greater than 200 [3, p. 591]. And the standard choice of [**Mtry**] is the square root of the number of features $p$ so that $\mathbf{[Mtry]} \approx \sqrt{p}$ [3, p. 589]. Figure 2 shows an accuracy plot for number of trees sequenced in values of 10 from 0 to 300 and tree depth from 2 to 5. We observe that values greater than 200 for number of trees seem to result in stagnation in terms of improvement and in the same way we observe stagnation for values greater than 4 for [**Mtry**] value. We also note that higher values for [**Mtry**] was tested and resulted in the same observation. The optimal parameter choice for our data

was therefore concluded to be 200 trees and [**Mtry = 4**] and no higher due to complexity.

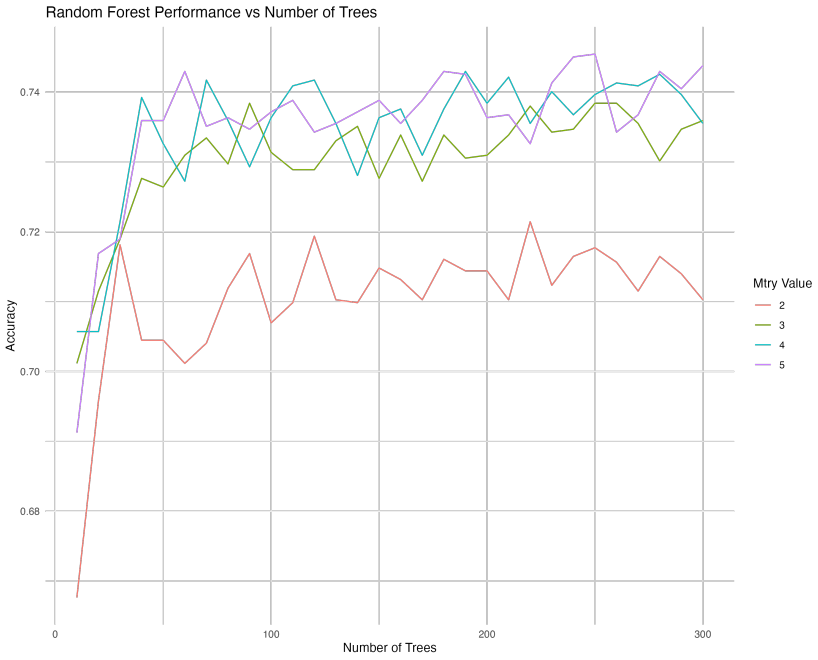


Figure 2: Accuracy Plot for Number of trees by Mtry

## 5.2 Reduction in dimensionality

As mentioned in section 4.1 the use of PCA was not chosen due to data being non-continuous and instead other methods for dimensionality reduction were conducted. Firstly candidates for features which might increase model accuracy by removal were features with a high Cramers' V value in relation to another feature. From this pool we first performed feature elimination and observed difference in model accuracy and iteratively kept features which outperformed others. This method however yielded different results on similar tests and instead we opted to use an average of multiple tests with different [**Mtry**] values where we instead conducted inclusion tests (similar to forward selection).

When evaluating performance, a subset of our data was used in order to decrease loading times and instead focus on inter-model performance with different features. Observing table 2 we notice higher performances among certain model with sizes between the Base model and the Full model. In particular, the model which shows the highest accuracy is the model with features **Age** and **has_children** removed.

Accuracy however is not always completely reliable. Some of the model's correct classifications are purely by chance. In order to better understand how the model responds to new information, we can use a metric know as Cohen's Kappa, which utilizes differences in observed vs expected accuracy to produce a value between -1 and 1. The model with the highest Kappa value is the one where two more parameters have been removed, **Temperature** and **Time**. These two models together with the Full model form the final 3 models to be considered.

### 5.3 Performance of final model

To perform a selection of our final model, we conducted a random forest classification on the full data-set with parameters **[ntrees] = 200** and **[Mtry] = 4**. We selected 18 parameters as predictor variables and one, **accept_coupon** as response. In the final model the following parameters we removed: **Temperature**, **Time**, **Age**, **has_children**.

As seen in table 3, this model had the highest Accuracy and Kappa value among the remaining candidate models. An accuracy of (71.83-75.38%) suggests that the final model achieves a moderately low miss-classification rate, miss-classifying around 1 in every 4 coupon decisions. A Cohen's kappa value of 0.459 suggests that there is a moderate agreement between the predicted and observed classifications and that this model is significantly better than a selection by random chance.

| **Features removed** | **Accuracy** | **95% CI for accuracy** | **Kappa** |
|---|---|---|---|
| temp.+time+age+has_children | 0.7363 | (0.7183, 0.7538) | 0.4590 |
| age+has_children | 0.7322 | (0.7141, 0.7498) | 0.4489 |
| full_model | 0.7272 | (0.7090, 0.7449) | 0.4358 |

Table 3: Final model performance

## 6 Discussion

For the final section we reflect on the process for method selection, model selection, the models performance and what further steps could have been taken in order to achieve a better final model.

### 6.1 Model selection

After selecting a dataset we opted to model after a binomial classification problem. Our initial chosen method for this problem was to use SVM. However after comparing the method to other methods we opted for the random forest instead

because it was more suitable for modeling our data. We then proceeded to compare different parameter and feature choices for the model. We are confident our final selection of model parameters are optimal due to empirical evidence presented in section 5, but there still might exist a better feature selection for our model. We opted to choose a model with removed features for time, temperature, age and has_children but alternatively one might have achieved a better model by performing a more extensive feature selection. We note that both forward elimination and selection have been performed but yielded similar results for features. Ultimately an accuracy of approximately 0.7363 which can be observed in Table 3 is satisfactory for the chosen data and difference in feature selection mostly yield marginally different results.

### 6.2 Further improvements

Due to time constraints a smaller data-set was used in deparametrisation which could have resulted in inaccuracies when selecting features. Furthermore use of a more extensive backward selection which compares all 22 features might have yielded a better final model. Use of SVM could have been conducted and presented if data had been converted into dummy variables but likely would not have led to a better result.

## References


[1] in-vehicle coupon recommendation. UCI Machine Learning Repository, 2020. DOI: https://doi.org/10.24432/C5GS4P.

[2] Harald Cramér. *Mathematical Methods of Statistics*. Princeton University Press, Princeton, 1946.

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2 edition, 2009.

[4] Statology. How to interpret cramer's v, Year of publication or last update. Accessed on: Date of access.


# 7 Appendix

### 7.1 Code

```
```{r, message=FALSE, warning=FALSE}
# Load libraries
library(tidyverse)
library(ggplot2)
library(vcd)
library(reshape2)
```
```

```
library(randomForest)
library(caret)
```

```{r}
# Load dataframe
coupon_df <- read.csv("data/vehicle_data.csv",sep = ",")

#number of non-missing observations in the "car" column
sum(coupon_df$car != "")

# number of missing observations not in the "car" column
sum(coupon_df == "") - sum(coupon_df$car == "")

# number of missing obs in other columns
sum(coupon_df$Bar == "")
sum(coupon_df$CoffeeHouse == "")
sum(coupon_df$CarryAway == "")
sum(coupon_df$RestaurantLessThan20 == "")
sum(coupon_df$Restaurant20To50 == "")

# all obs with missing values in some column except "car"
nocar_coupon_df <- coupon_df[rowSums(coupon_df[,-15] == "") > 0,]

# make copy of the original data frame and replace "" with NA values
coupon_df1 <- coupon_df
coupon_df1[coupon_df1 == ""] <- NA

# remove "car" column
coupon_df1 <- coupon_df1[,-15]

# remove all observations with missing values
coupon_df1 <- na.omit(coupon_df1)
```

```{r}
# removing toCoupon_GEQ5min and direction_opp
coupon_df1 <- coupon_df1[,-c(24,20)]

# renaming columns
coupon_df1 <- coupon_df1 %>%
  rename(bar_gt1 = Bar, coffee_gt1 = CoffeeHouse,
  takeaway_gt1 = CarryAway, restaurantless20_gt1 = RestaurantLessThan20,
  restaurant20plus_gt1 = Restaurant20To50, accept_coupon = Y)
```

```
# renaming and tranforming columns
coupon_df1 <- coupon_df1 %>%
  mutate(time = case_when(
    time == "2PM"  ~ "Afternoon",
    time == "10AM" ~ "Noon",
    time == "6PM"  ~ "Evening",
    time == "7AM"  ~ "Morning",
    time == "10PM" ~ "Night",
    TRUE ~ time)) %>%
  mutate(age = case_when(
    age %in% c("21", "26", "31", "36", "41", "46") ~ "21to50",
    TRUE ~ age)) %>%
  mutate(income = case_when(
    income == "Less than $12500"  ~ 0,
    income == "$12500 — $24999" ~ 1,
    income == "$25000 — $37499" ~ 2,
    income == "$37500 — $49999" ~ 3,
    income == "$50000 — $62499" ~ 4,
    income == "$62500 — $74999" ~ 5,
    income == "$75000 — $87499" ~ 6,
    income == "$87500 — $99999" ~ 7,
    income == "$100000 or More" ~ 8)) %>%
  mutate(bar_gt1 = case_when(
    bar_gt1 %in% c("never", "less1") ~ 0,
    bar_gt1 %in% c("1~3", "4~8", "gt8") ~ 1)) %>%
  mutate(coffee_gt1 = case_when(
    coffee_gt1 %in% c("never", "less1") ~ 0,
    coffee_gt1 %in% c("1~3", "4~8", "gt8") ~ 1)) %>%
  mutate(takeaway_gt1 = case_when(
    takeaway_gt1 %in% c("never", "less1") ~ 0,
    takeaway_gt1 %in% c("1~3", "4~8", "gt8") ~ 1)) %>%
  mutate(restaurantless20_gt1 = case_when(
    restaurantless20_gt1 %in% c("never", "less1") ~ 0,
    restaurantless20_gt1 %in% c("1~3", "4~8", "gt8") ~ 1)) %>%
  mutate(restaurant20plus_gt1 = case_when(
    restaurant20plus_gt1 %in% c("never", "less1") ~ 0,
    restaurant20plus_gt1 %in% c("1~3", "4~8", "gt8") ~ 1)) %>%
  mutate_if(is.numeric, as.integer)
```

```{r}
# Get the variable names (column names of coupon_df1)
varnames <- colnames(coupon_df1)

# Get the data types for each column
data_types <- sapply(coupon_df1, class)
```

```
# Create a vector for the outcomes
outcomes <- c("No Urgent Place, Home, Work",
              "Alone, Friend(s), Kid(s), Partner",
              "Sunny, Rainy, Snowy",
              "30, 55, 80",
              "Morning, Noon, Afternoon, Evening, Night",
              "Bar, Coffee House, Take away,
--------------Restaurant(<$20), Restaurant($20--$50)",
              "1d, 2h",
              "Female, Male",
              "Below 21, 21 to 50, Above 50",
              "Unmarried partner, Single, Married partner,
--------------Divorced, Widowed",
              "No=0, Yes=1",
              "Some High School, High School Graduate, Some college,
--------------Bachelors degree, Associates degree,
--------------Graduate degree (Msc or PHD)",
              "List of 25 occupations (incl. Unemployed, Student,
--------------Retired, Legal, Healthcare support etc.)",
              "Less than $12500=0, $12500--$24999=1,
--------------$25000--$37499=2, $37500--$49999=3,
--------------$50000--$62499=4, $62500--$74999=5,
--------------$75000--$87499=6, $87500--$99999=7,
--------------$100000 or More=8",
              "Visits Bar more than once per month
--------------(No=0, Yes=1)",
              "Visits Coffee House more than once per month
--------------(No=0, Yes=1)",
              "Gets Take Away food more than once per month
--------------(No=0, Yes=1)",
              "Visits Restaurant with avg expense <$20
--------------more than once per month (No=0, Yes=1)",
              "Visits Restaurant with avg expense $20--$50
--------------more than once per month (No=0, Yes=1)",
              "Driving distance to the restaurant/bar for
--------------using the coupon is 15< minutes (No=0, Yes=1)",
              "Driving distance to the restaurant/bar for
--------------using the coupon is 25< minutes (No=0, Yes=1)",
              "restaurant/bar same direction as destination
--------------(No=0, Yes=1)",
              "No=0, Yes=1")

# Combine everything into a new data frame
table_df <- data.frame(Variable = varnames,
Type = data_types, Outcome = outcomes, row.names = NULL)
```

```
# Display the table
knitr::kable(table_df)
```

```{r, warning=FALSE}
# define function for cramers V
cramersV <- function(x, y) {
  tbl <- table(x, y)
  chi2 <- chisq.test(tbl, correct = FALSE)$statistic
  n <- sum(tbl)
  phi2 <- chi2 / n
  minDim <- min(nrow(tbl)-1, ncol(tbl)-1)
  sqrt(phi2 / minDim)
}

# creating correlation matrix (using cramers V)
cor_matrix <- matrix(nrow = ncol(coupon_df1), ncol = ncol(coupon_df1))
for (i in 1:ncol(coupon_df1)) {
  for (j in 1:ncol(coupon_df1)) {
    cor_matrix[i, j] <- cramersV(coupon_df1[[i]], coupon_df1[[j]])
  }
}
rownames(cor_matrix) <- colnames(coupon_df1)
colnames(cor_matrix) <- colnames(coupon_df1)
cor_melted <- melt(cor_matrix)

# plotting correlation matrix
ggplot(cor_melted, aes(Var1, Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                       midpoint = 0.5, limit = c(0,1), space = "Lab",
                       name="Cramer's V") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(x = '', y = '', title = 'Correlation Matrix for Categorical Data')
```

```{r}
coupon_df1$accept_coupon <- as.factor(coupon_df1$accept_coupon)
# Convert the target variable to a factor if it's a classification problem

# Splitting the data into training and test sets
set.seed(2024)  # for reproducibility
train_indices <- sample(1:nrow(coupon_df1), size = 0.8*nrow(coupon_df1))
# 80% for training
```

```
train_data <- coupon_df1[train_indices, ]
test_data <- coupon_df1[-train_indices, ]

# Train the model
rf_model <- randomForest(accept_coupon ~ .,
data = train_data, ntree = 200, mtry = 4)

# Predict on test data
predictions <- predict(rf_model, test_data)

# Evaluate the model
confusionMatrix(predictions, test_data$accept_coupon)
```

```{r}
set.seed(2024)  # For reproducibility

# setting tuning parameters
ntree_values = seq(10, 300, by = 10)
mtry_values = c(2, 3, 4, 5)
results = data.frame(ntree = integer(),
mtry = integer(), accuracy = numeric())

# iterating over models with various tuning
# parameters and calculating accuracy
for (m in mtry_values) {
  for (n in ntree_values) {
    model = randomForest(accept_coupon ~ .,
    data = train_data, ntree = n, mtry = m)
    predictions = predict(model, test_data)
    accuracy =
    sum(predictions == test_data$accept_coupon) / nrow(test_data)
    results =
    rbind(results, data.frame(ntree = n, mtry = m, accuracy = accuracy))
  }
}

# visualizing accuracy for various models
ggplot(results, aes(x = ntree, y = accuracy, color = as.factor(mtry))) +
  geom_line() +
  labs(title = "Random Forest Performance vs Number of Trees",
       x = "Number of Trees",
       y = "Accuracy",
       color = "Mtry Value") +
  theme_minimal()
```

```r
# Forward feature selection

set.seed(2024)  # For reproducibility
sample_size <- floor(0.01 * nrow(coupon_df1))
data_subset <- coupon_df1[sample(1:nrow(coupon_df1), sample_size), ]

# Define base, additional and target variables
all_vars <- names(data_subset)[1:22]  # All predictor variables
base_vars <- names(data_subset)[c(3, 6:8, 10, 12:22)]
additional_vars <- names(data_subset)[c(1, 2, 4, 5, 9, 11)]
target_var <- names(data_subset)[23]

# Define a 10-fold cross-validation
control <- trainControl(method="cv", number=10)
tuneGrid <- expand.grid(.mtry = c(2, 3, 4, 5, 6, 7, 8))
performances <- data.frame(variable=character(),
accuracy=numeric(), kappa=numeric(), stringsAsFactors=FALSE)

# Train base model with Random Forest
base_formula <- as.formula(paste(target_var, "~",
paste(base_vars, collapse="+")))
base_model <- train(base_formula, data=data_subset,
method="rf", trControl=control, tuneGrid=tuneGrid)
base_acc <- max(base_model$results$Accuracy)
base_kappa <- max(base_model$results$Kappa)
performances <- rbind(performances,
data.frame(model_name="base_model", accuracy=base_acc, kappa = base_kappa))

# Train full model with Random Forest
full_model <- train(as.formula(paste(target_var, "~ ."),
data=data_subset, method="rf", trControl=control, tuneGrid=tuneGrid)
full_acc <- max(full_model$results$Accuracy)
full_kappa <- max(full_model$results$Kappa)
performances <- rbind(performances,
data.frame(model_name="full_model", accuracy=full_acc, kappa = full_kappa))

# Incremental feature addition
used_vars <- base_vars
for (var in additional_vars) {
  used_vars <- c(used_vars, var)  # Add variable to current set
  updated_formula <- as.formula(paste(target_var, "~",
  paste(used_vars, collapse="+")))
  model <- train(updated_formula, data=data_subset,
  method="rf", trControl=control, tuneGrid=tuneGrid)
```

```
  mean_accuracy <- mean(model$results$Accuracy)
  mean_kappa <- mean(model$results$Kappa)

  removed_vars <- setdiff(all_vars, used_vars)
  model_name <- ifelse(length(removed_vars) == 0,
  "full_model", paste(removed_vars, collapse="+"))

  performances <- rbind(performances, data.frame(model_name=model_name,
  accuracy=mean_accuracy, kappa = mean_kappa))
}

# display table
print(performances)
```

```{r}
set.seed(2024)  # for reproducibility

# Final model
final_model <- coupon_df1[,-c(9, 11)]

final_model$accept_coupon <- as.factor(final_model$accept_coupon)
# Convert the target variable to a factor if it's a classification problem

# Splitting the data into training and test sets
train_indices <- sample(1:nrow(final_model), size = 0.8*nrow(final_model))
# 80% for training
train_data <- final_model[train_indices, ]
test_data <- final_model[-train_indices, ]

# Train the model
rf_model <- randomForest(accept_coupon ~ .,
data = train_data, ntree = 200, mtry = 4)

# Predict on test data
predictions <- predict(rf_model, test_data)

# Evaluate the model
confusionMatrix(predictions, test_data$accept_coupon)
```