BlackFriday_Pravat

Pravat

23 April 2019

GitHub Source

https://github.com/pasayatpravat/Black_Friday_R.git (https://github.com/pasayatpravat/Black_Friday_R.git)

Global setup and Library import

Introduction

The dataset here is a sample of the transactions made in a retail store. The store wants to know better the customer purchase behaviour against different products. Specifically, here the problem is a regression problem where we are trying to predict the dependent variable (the amount of purchase) with the help of the information contained in the other variables.

Classification problem can also be settled in this dataset since several variables are categorical, and some other approaches could be “Predicting the age of the consumer” or even “Predict the category of goods bought”. This dataset is also particularly convenient for clustering and maybe find different clusters of consumers within it.

CONTENT:

Both Dataset have the following fields, except Purchase - which is not available in BlackFriday_test.csv.

1. User_ID: Unique ID of the customer
2. Product_ID: Unique ID of the product sold/bought
3. Gender: Gender of the Customer
4. Age: Age group of the customer
5. Occupation: Customer occupation category
6. City_Category: Category of the city
7. Stay_In_Current_City_Years: Number of years the Customer has been staying in the city
8. Marital_Status: Customer’s marital status
9. Product_Category_1: Parent category of the product
10. Product_Category_2: Sub-category on the Product_Category_1
11. Product_Category_3: Sub-category on the Product_Category_2
12. Purchase: Target variable. Monitary amount of purchase

Useful Functions

```
# Function to split the dataset randomly with a given propert
splitdt <- function(dt, test_proportion=0.2, seed=NULL){
  if(!is.null(seed)) set.seed(seed)

  train_index <- sample(nrow(dt), floor(nrow(dt)*(1-test_proportion)), replace = FALSE)
  trainset<-dt[train_index]
  testset<-dt[-train_index]

  return(list(train=trainset, test=testset))
}

# MAPE metric
mape<-function(real,predicted){
  return(mean(abs((real-predicted)/real)))
}
```

Data Reading and preparation

The dataset is offered in two separated fields, one for the training and another one for the test set.

```
original_training_data = fread(file = file.path("BlackFriday_train.csv"))
original_test_data = read.csv(file = file.path("BlackFriday_test.csv"))
```

To avoid applying the Feature Engineering process two times (once for training and once for test), you can just join both datasets (using the `rbind` function), apply your FE and then split the datasets again. However, if we try to do join the two dataframes as they are, we will get an error because they do not have the same columns: `test_data` does not have a column `Purchase`. Therefore, we first create this column in the test set and then we join the data

```
# Create the column Purchase in test set and assign the value to 0
original_test_data$Purchase <- 0

# Create the column dataType in both train and test set and assign the value 'train' & 'tes
t'. This will help us to split the dataset from the dataType, not by position
original_training_data$dataType <- "train"
original_test_data$dataType <- "test"

# Join the two datasets
dataset <- rbind(original_training_data, original_test_data)
```

Let's now visualize the dataset to see where to begin

```
summary(dataset)
```

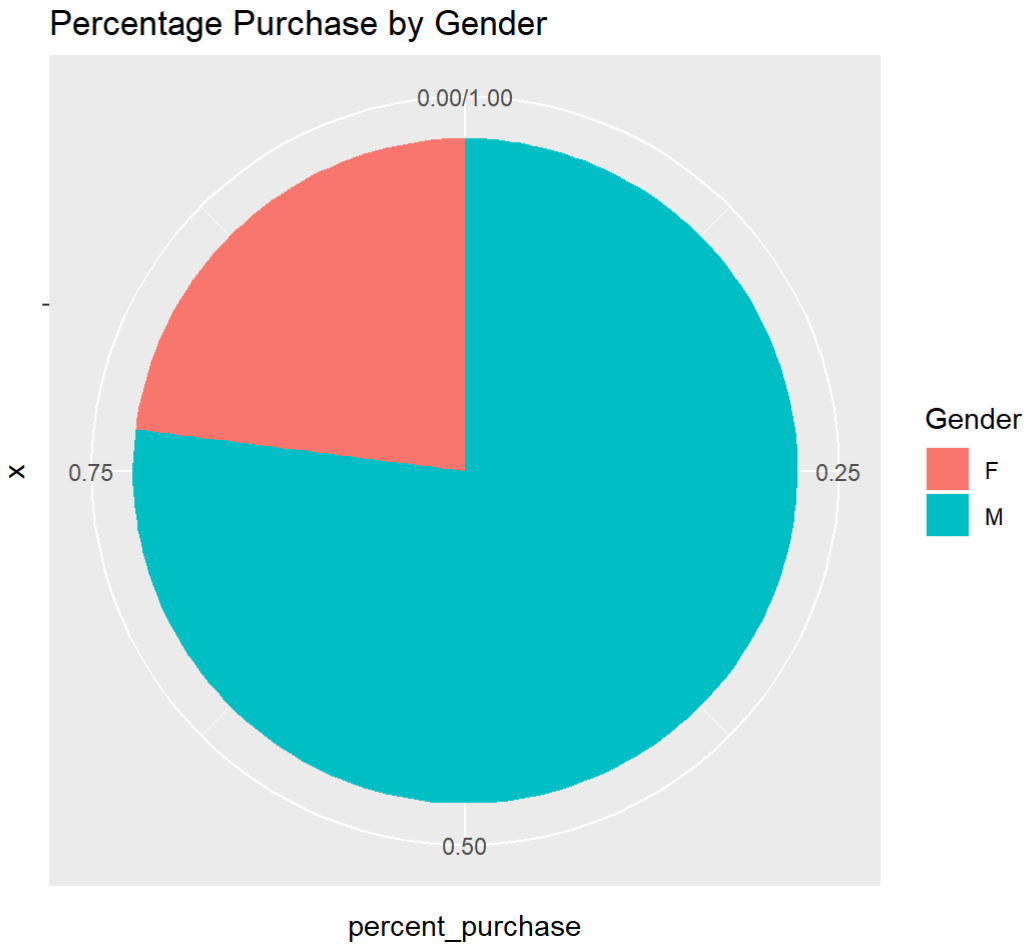
```
##     User_ID            Product_ID     Gender        Age
##  Min.   :1000001   P00265242:  1858   F:132197   36-45:107499
##  1st Qu.:1001495   P00110742:  1591   M:405380   26-35:214690
##  Median :1003031   P00025442:  1586              18-25: 97634
##  Mean   :1002992   P00112142:  1539              0-17 : 14707
##  3rd Qu.:1004417   P00057642:  1430              46-50: 44526
##  Max.   :1006040   P00184942:  1424              51-55: 37618
##                    (Other)  :528149              55+  : 20903
##    Occupation     City_Category Stay_In_Current_City_Years
##  Min.   : 0.000   C:166446      1 :189192
##  1st Qu.: 2.000   B:226493      3 : 93312
##  Median : 7.000   A:144638      0 : 72725
##  Mean   : 8.083                 2 : 99459
##  3rd Qu.:14.000                 4+: 82889
##  Max.   :20.000
##

##  Marital_Status   Product_Category_1 Product_Category_2 Product_Category_3
##  Min.   :0.0000   Min.   : 1.000     Min.   : 2.00      Min.   : 3.0
##  1st Qu.:0.0000   1st Qu.: 1.000     1st Qu.: 5.00      1st Qu.: 9.0
##  Median :0.0000   Median : 5.000     Median : 9.00      Median :14.0
##  Mean   :0.4088   Mean   : 5.296     Mean   : 9.84      Mean   :12.7
##  3rd Qu.:1.0000   3rd Qu.: 8.000     3rd Qu.:15.00      3rd Qu.:16.0
##  Max.   :1.0000   Max.   :18.000     Max.   :18.00      Max.   :18.0
##                                      NA's   :166986     NA's   :373299
##     Purchase       dataType
##  Min.   :    0   Length:537577
##  1st Qu.: 5139   Class :character
##  Median : 7847   Mode  :character
##  Mean   : 8399
##  3rd Qu.:11790
##  Max.   :23961
##
```

We can see some problems just by taking a look to the summary: the dataset has missing values, there are some categorical columns codified as numeric, it has different scales for the feature values. In addition, we will take a deeper look to the data to detect more subtle issues: correlation between features, skewness in the feature values.

EDA - Exploratory Data Analysis

Let’s now visualize the dataset to get some insights that we can use during feature engineering.

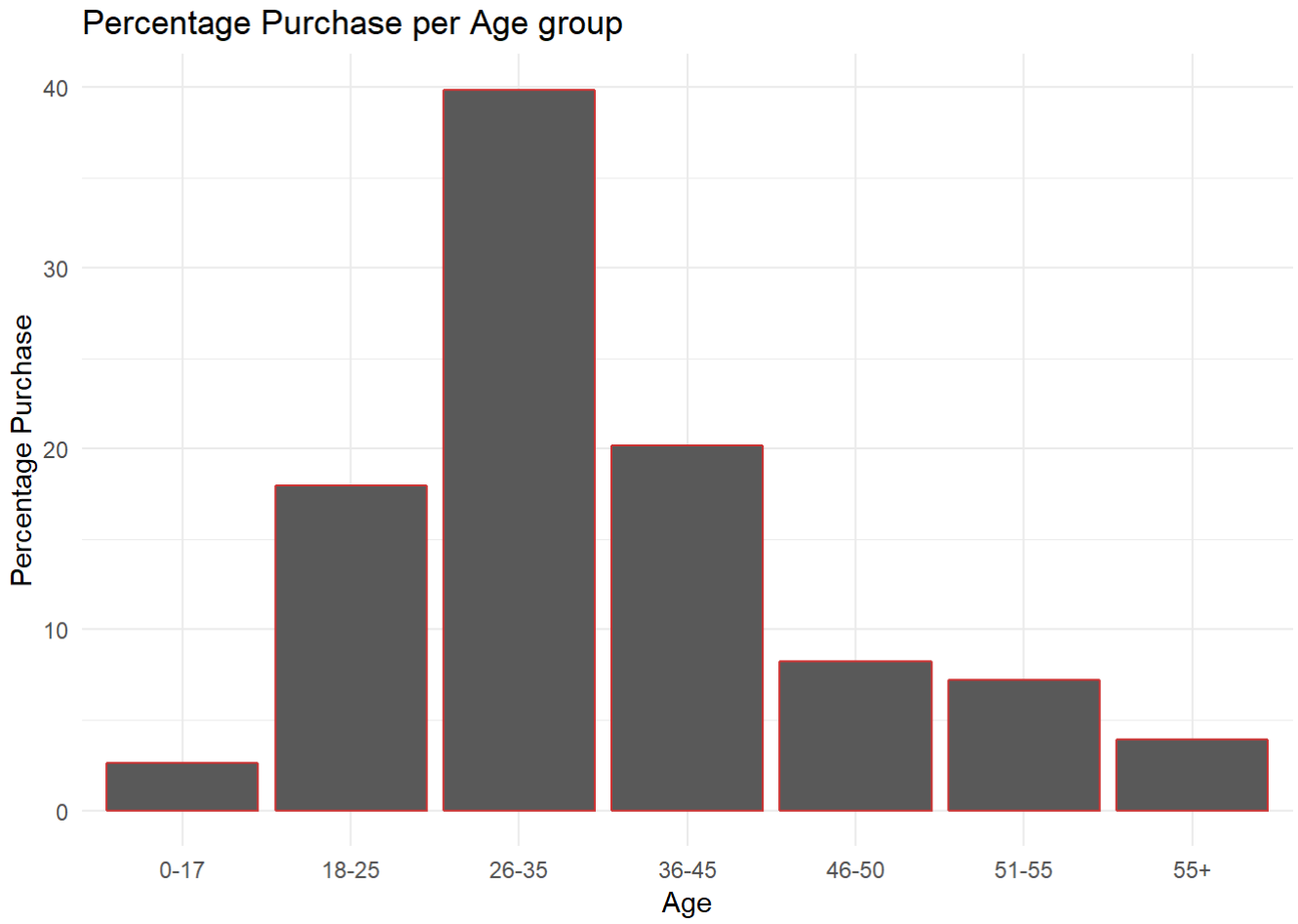
Distribution of Purchase by Gender



This shows that purchases done by `Male` are more than 3 times of that done by `Female` .

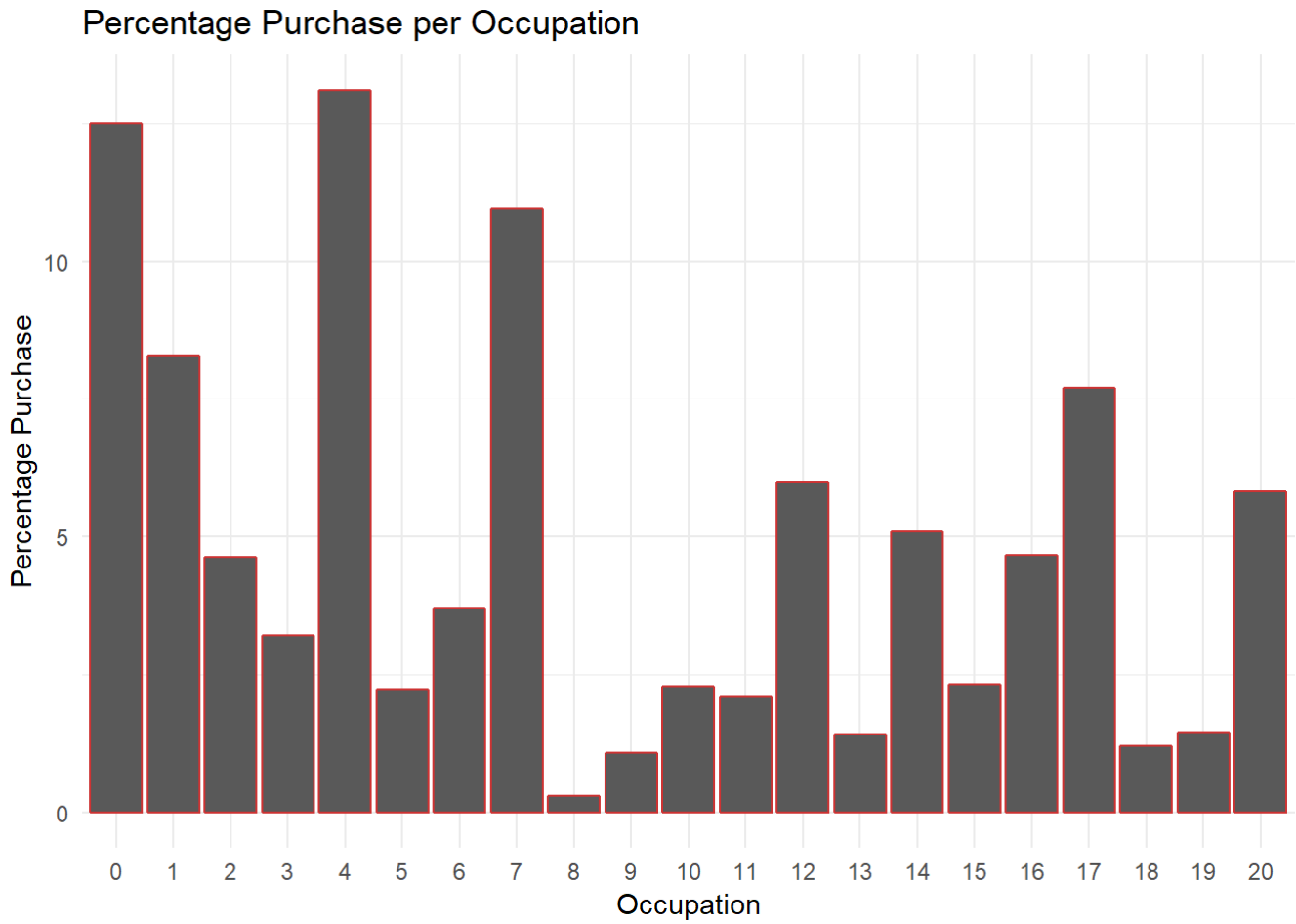
Purchase by Age group

```
##      Age percentage_purchase_age
## 1: 26-35                39.867035
## 2: 36-45                20.171584
## 3: 18-25                17.978423
## 4: 46-50                 8.222417
## 5: 51-55                 7.206122
## 6:   55+                 3.909795
## 7:  0-17                 2.644624
```



Purchase by Occupation

```
##     Occupation percentage_purchase_occupation
##  1:          4                     13.1088520
##  2:          0                     12.4904135
##  3:          7                     10.9463330
##  4:          1                      8.2842291
##  5:         17                      7.7051263
##  6:         12                      6.0013196
##  7:         20                      5.8226906
##  8:         14                      5.0945044
##  9:         16                      4.6633342
## 10:          2                      4.6367285
## 11:          6                      3.7014668
## 12:          3                      3.2039858
## 13:         15                      2.3182674
## 14:         10                      2.2771789
## 15:          5                      2.2367778
## 16:         11                      2.0884370
## 17:         19                      1.4511526
## 18:         13                      1.4080468
## 19:         18                      1.2020001
## 20:          9                      1.0681342
## 21:          8                      0.2910214
##     Occupation percentage_purchase_occupation
```

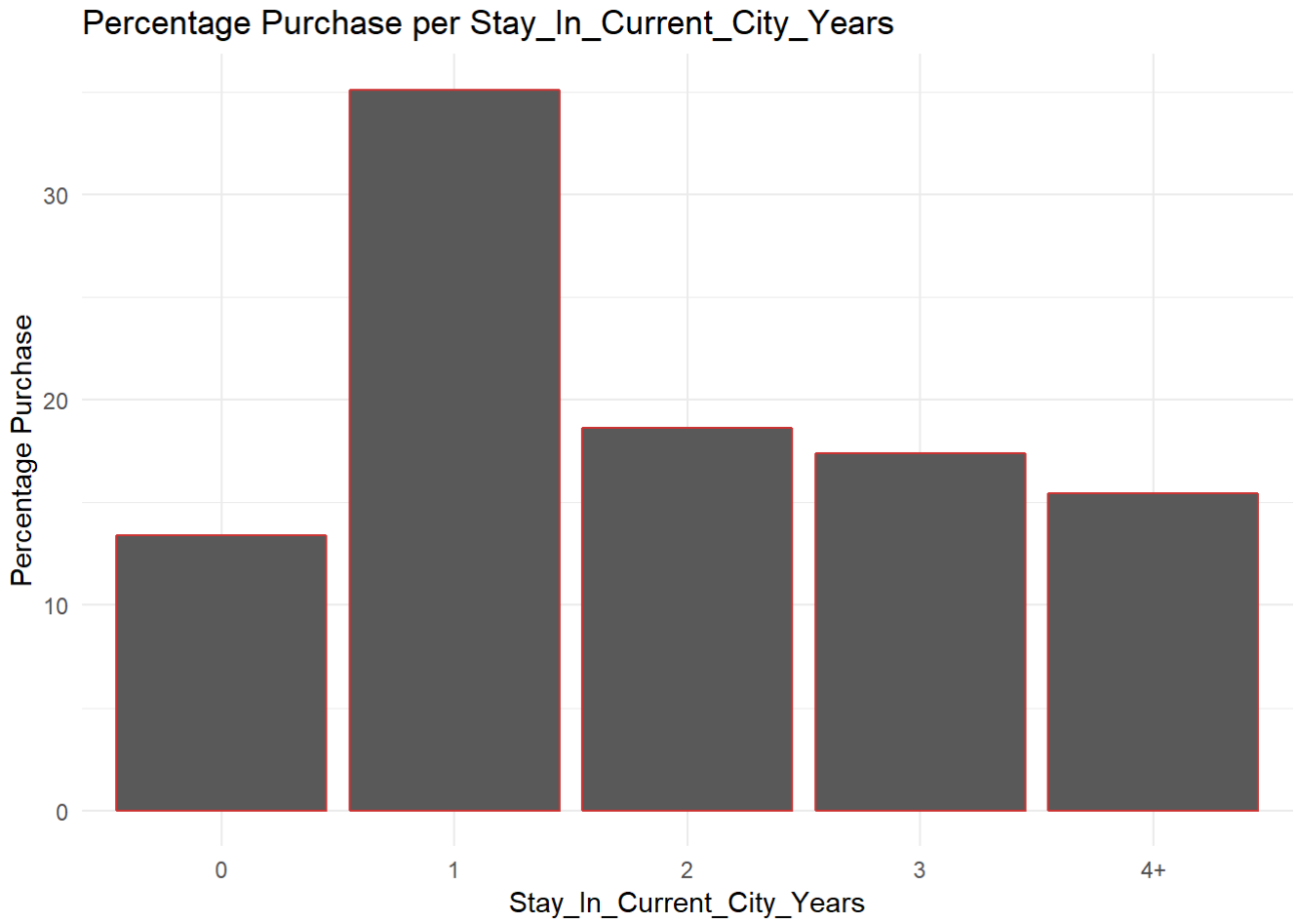


Purchase by City Category

```
##    City_Category percentage_purchase_city
## 1:             B                 41.53787
## 2:             C                 32.64197
## 3:             A                 25.82016
```

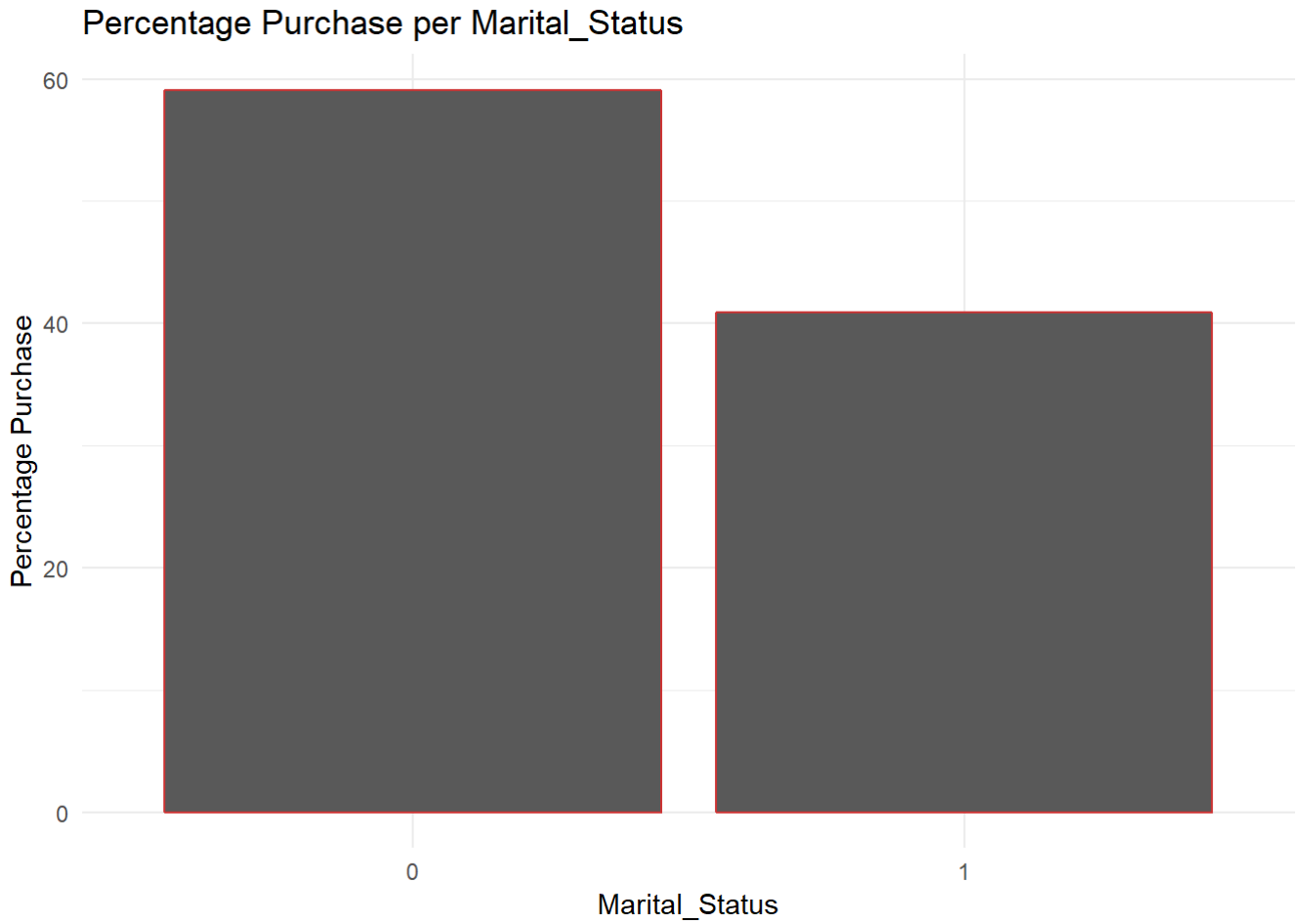
Purchase by Stay_In_Current_City_Years

```
##    Stay_In_Current_City_Years percentage_purchase_stay
## 1:                          1                 35.13130
## 2:                          2                 18.63829
## 3:                          3                 17.39102
## 4:                         4+                 15.44723
## 5:                          0                 13.39216
```



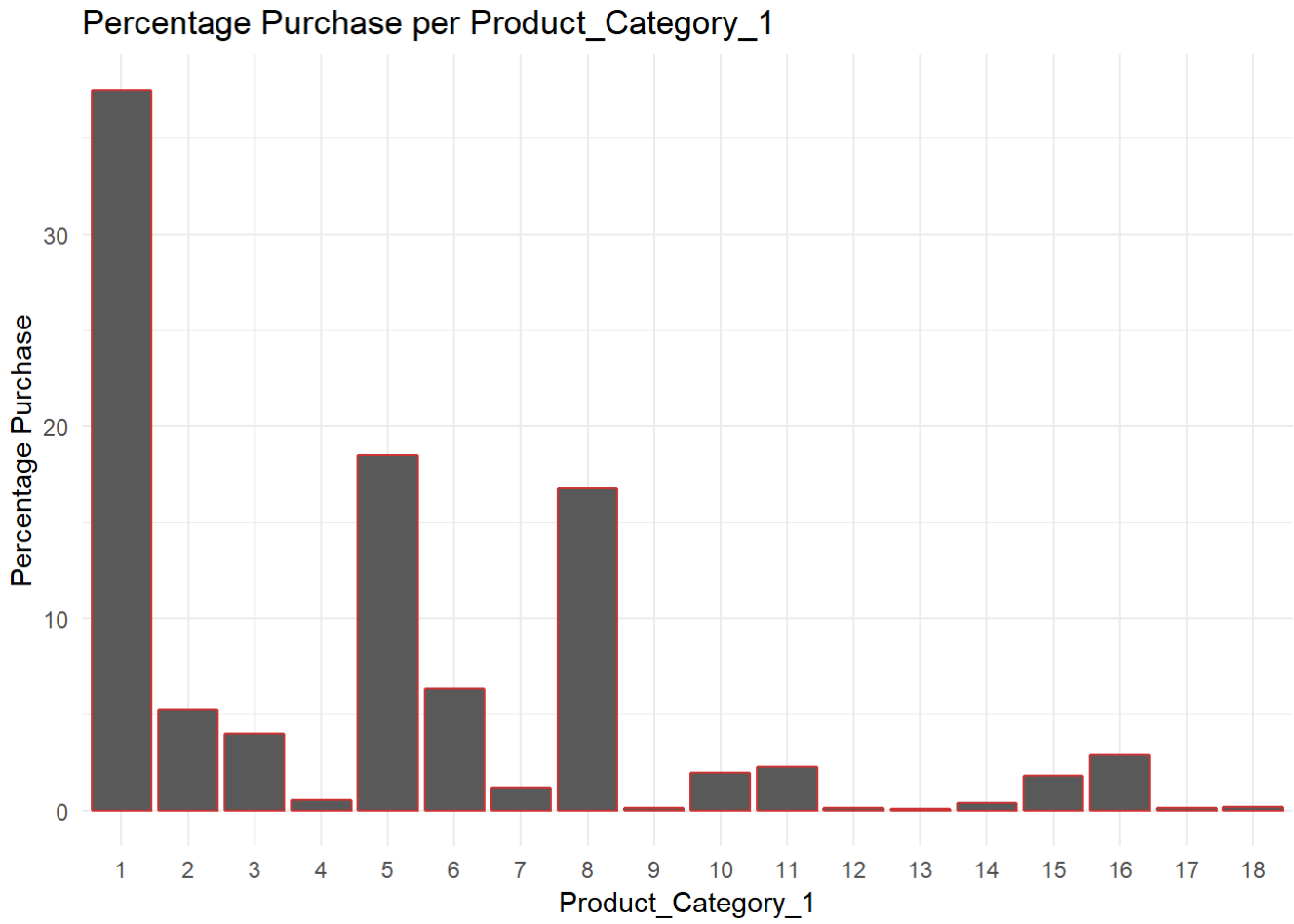
Purchase by Marital_Status

```
##    Marital_Status percentage_purchase_mstatus
## 1:              0                    59.11513
## 2:              1                    40.88487
```



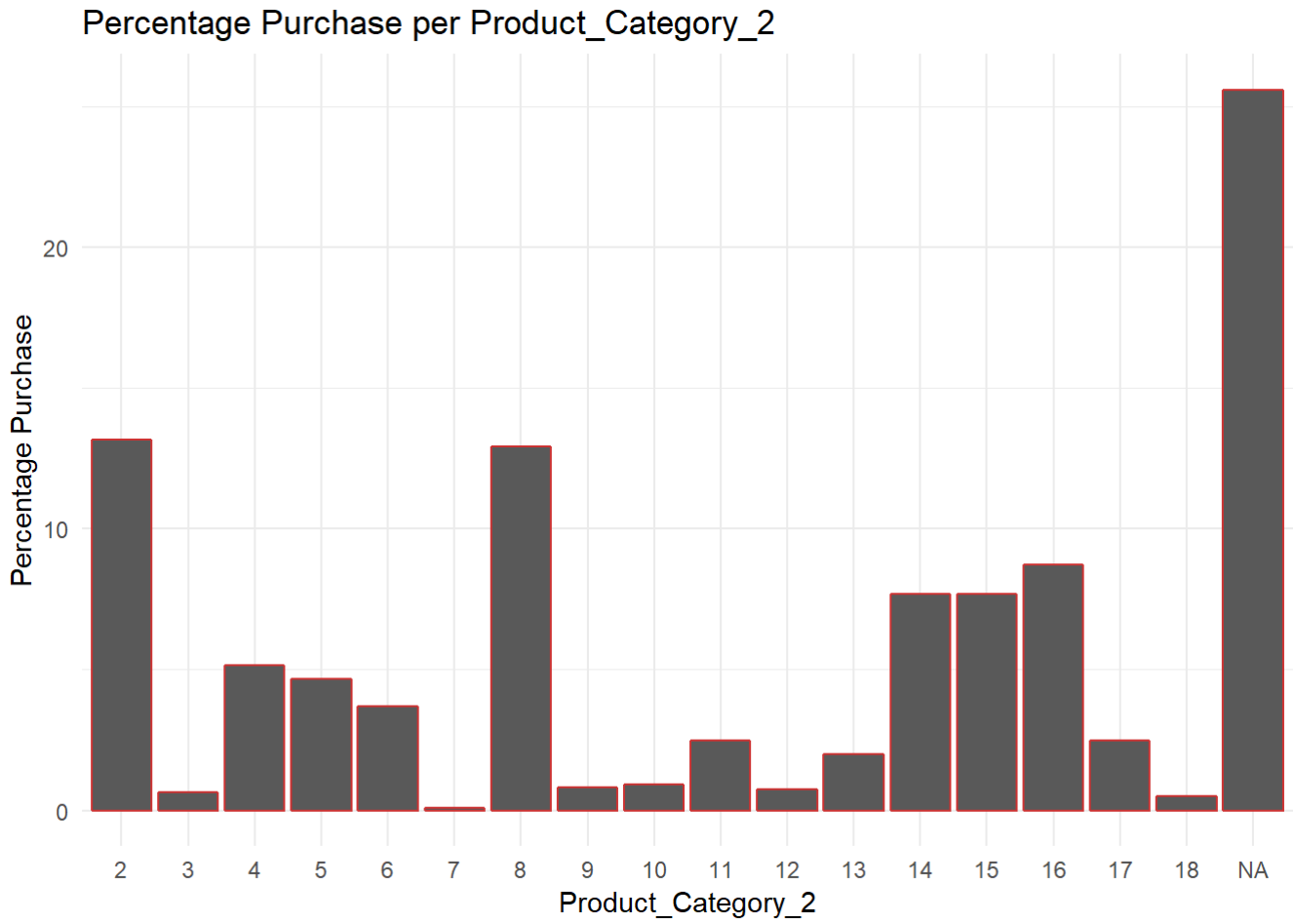
Purchase by Product_Category_1

```
##     Product_Category_1 percentage_purchase_pcat1
##  1:                  1                37.5266900
##  2:                  5                18.4872636
##  3:                  8                16.7377647
##  4:                  6                 6.3525061
##  5:                  2                 5.2628853
##  6:                  3                 4.0093946
##  7:                 16                 2.8525300
##  8:                 11                 2.2425923
##  9:                 10                 1.9796092
## 10:                 15                 1.8214109
## 11:                  7                 1.1970121
## 12:                  4                 0.5379949
## 13:                 14                 0.3886245
## 14:                 18                 0.1833623
## 15:                  9                 0.1243825
## 16:                 17                 0.1135406
## 17:                 12                 0.1041113
## 18:                 13                 0.0783249
```



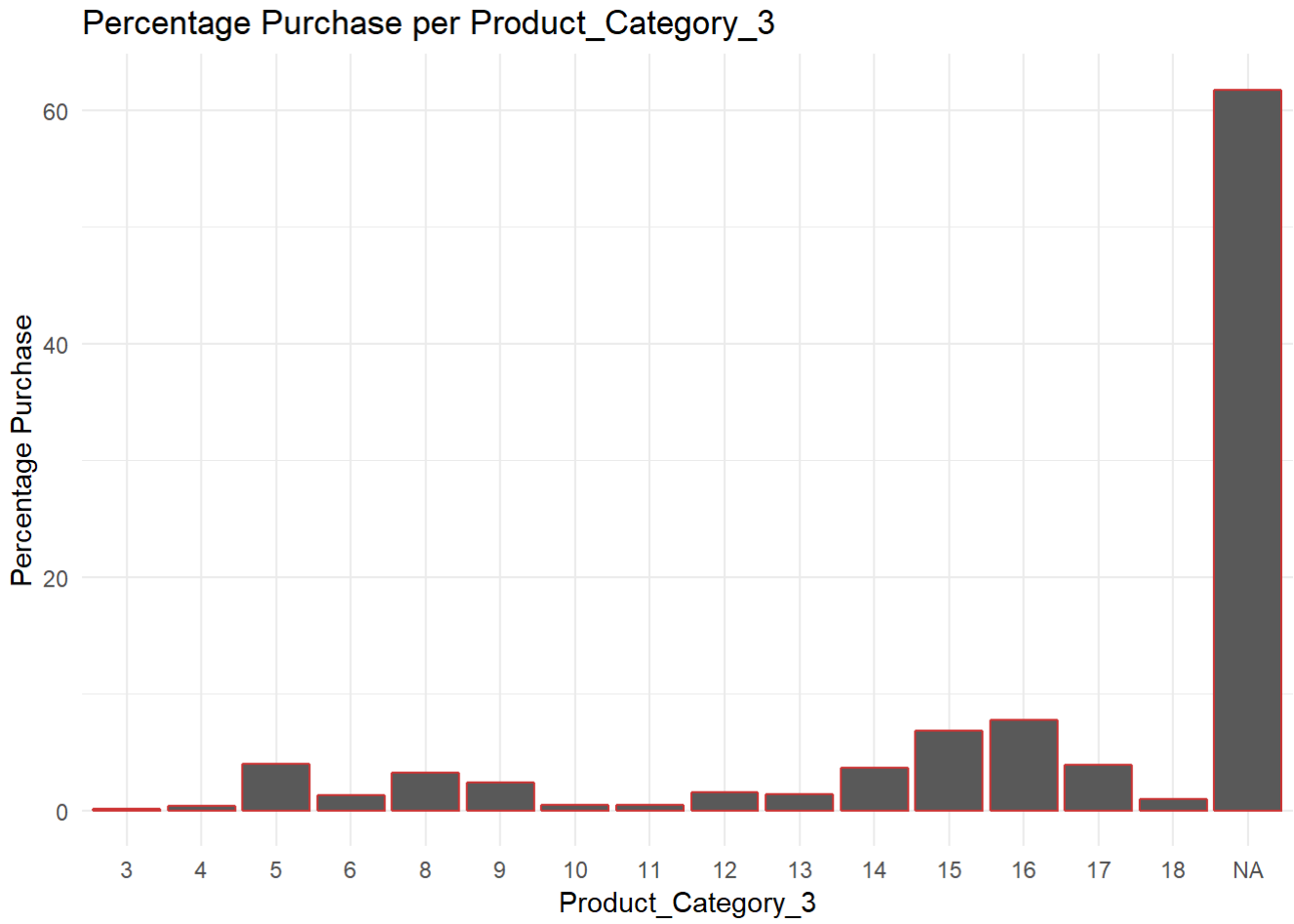
Purchase by Product_Category_2

```
##     Product_Category_2 percentage_purchase_pcat2
##  1:                 NA                25.60979891
##  2:                  2                13.16173751
##  3:                  8                12.93354160
##  4:                 16                 8.74045081
##  5:                 15                 7.69376926
##  6:                 14                 7.67777333
##  7:                  4                 5.14916531
##  8:                  5                 4.65813439
##  9:                  6                 3.71360278
## 10:                 11                 2.46868131
## 11:                 17                 2.46858984
## 12:                 13                 2.00260362
## 13:                 10                 0.92665424
## 14:                  9                 0.81063548
## 15:                 12                 0.74967182
## 16:                  3                 0.63861899
## 17:                 18                 0.51201116
## 18:                  7                 0.08455964
```



Purchase by Product_Category_3

```
##     Product_Category_3 percentage_purchase_pcat3
##  1:                 NA                61.7900792
##  2:                 16                 7.7074322
##  3:                 15                 6.8007718
##  4:                  5                 3.9673267
##  5:                 17                 3.8584091
##  6:                 14                 3.6311635
##  7:                  8                 3.2119316
##  8:                  9                 2.3646467
##  9:                 12                 1.5817642
## 10:                 13                 1.4005028
## 11:                  6                 1.2747808
## 12:                 18                 0.9940927
## 13:                 10                 0.4607817
## 14:                 11                 0.4270236
## 15:                  4                 0.3598829
## 16:                  3                 0.1694105
```



Data Cleaning and Correction

Factorize features

If we go back to the summary of the dataset we can identify some numerical features that are actually categories: `Occupation`, `Marital_Status`, `Product_Category_1`, `Product_Category_2` and `Product_Category_3`. What we have to do is to convert them to the proper 'class' or 'type' using the `as.factor` command.

```
# Declare the columns that needs to be converted to categorical variable
convert_to_cat_columns <- c("User_ID","Product_ID", "Occupation", "Marital_Status", "Product_
Category_1", "Product_Category_2", "Product_Category_3", "dataType")

# Do the conversion
dataset[ , (convert_to_cat_columns) := lapply(.SD, as.factor), .SDcols = convert_to_cat_colum
ns]
```

Hunting NAs

Our dataset is filled with missing values, therefore, before we can build any predictive model we'll clean our data by filling in all NA's with more appropriate values. As another option, we could just remove the entries with null values (i.e., remove rows). However, in this situation (and in many other that you will face) this is out of the question: we have to provide a prediction for each and every record. Similarly, we could discard the features with null values (i.e., remove columns), but it would mean the removal of many features (and the information they provide).

As a rule of thumb, if we are allowed to discard some of your data and we do not have many null values (or you do not have a clear idea of how to impute them) we can safely delete them. If this is not the case, we must find a way to impute them (either by applying some knowledge of the addressed domain or by using some more advanced imputation method.

Counting columns with null values.

```
na_cols <- names(which(colSums(is.na(dataset)) > 0))
paste('There are', length(na_cols), 'columns with missing values')
```

```
## [1] "There are 2 columns with missing values"
```

```
# Number of missing values in descending order
sort(
  colSums(
    sapply(
      dataset[, .SD, .SDcols = na_cols],
      is.na)
  ),
  decreasing = T)
```

```
## Product_Category_3 Product_Category_2
##             373299             166986
```

```
# Missing values in percentages
sort(
  sapply(
    dataset[, .SD, .SDcols = na_cols],
    function(x){100*sum(is.na(x))/length(x)}
  ),
  decreasing = T);
```

```
## Product_Category_3 Product_Category_2
##           69.44103           31.06271
```

Here we observe that `Product_Category_2` and `Product_Category_3` columns only have missing values. `Product_Category_3` is a sub-category of `Product_Category_2` , which inturn is a sub-category of `Product_Category_1` . So `Product_Category_1` sits at highest level, while `Product_Category_3` is the most ganular level.

As of now, let's impute the missing values of these two columns with `UKN` i.e. Unknown.

```
# Create a new level 'UKN' before we impute missing values with 'UKN'
dataset$Product_Category_2 = factor(dataset$Product_Category_2,
                                    levels=c(levels(dataset$Product_Category_2), "UKN"))
dataset$Product_Category_3 = factor(dataset$Product_Category_3,
                                    levels=c(levels(dataset$Product_Category_3), "UKN"))

# Let's impute missing value now
dataset$Product_Category_2[is.na(dataset$Product_Category_2)] <- "UKN";
dataset$Product_Category_3[is.na(dataset$Product_Category_3)] <- "UKN";
```

Outliers

We will now focus on numerical values. The main problem with numerical values are outliers (values which largely differ from the rest). Outliers can mislead the training of our models resulting in less accurate models and ultimately worse results.

In this dataset, we have `Purchase` i.e. the target variable as the only numeric column.

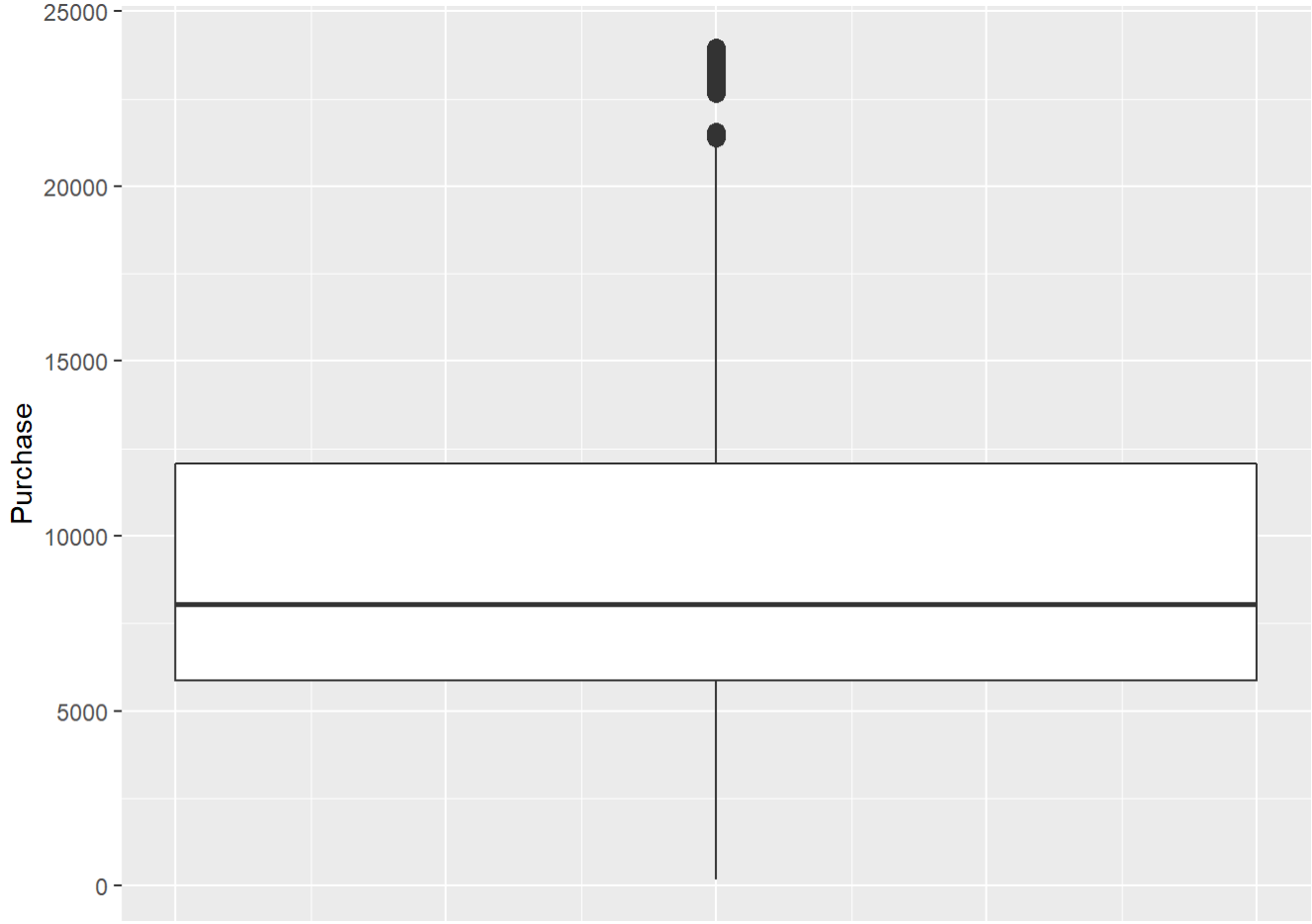
In this section we seek to identify outliers and then properly deal with them. If we summarize the dataset, we can see variables which “Max.” is much larger than the rest of values. These features are susceptible of containing outliers. Nevetheless, the easiest way to detect outliers is visualizing the numerical values; for instance, by `boxploting` the column values.

The `boxplot` function can eliminate the outliers. However, if you apply it with the default values it is going to eliminate too much of them. We can adapt its working with the `outlier.size` param with a recommended value of at least 3.

```
# Split the dataset into train and test, so that we can remove outlier from training set
training_data <- dataset[dataset$dataType == "train",]
test_data <- dataset[dataset$dataType == "test",]

sprintf("Number of training records before outlier removal: %i", nrow(training_data))
```

```
## [1] "Number of training records before outlier removal: 483819"
```

```
par(mfrow=c(2,5))
for (col in names(training_data)) {
  if (!is.factor(training_data[[col]])){
    print(ggplot(training_data, aes_string(y=col)) +
            geom_boxplot(width = 0.1, outlier.size = 3) +
            theme(axis.line.x = element_blank(), axis.title.x = element_blank(),
                  axis.ticks.x = element_blank(), axis.text.x = element_blank(),
                  legend.position="none"))

    #to_remove <- boxplot.stats(training_data[[col]])$out

    # I decided not to remove outliers as performance of the models went down
    #training_data <- training_data[!training_data[[col]] %in% to_remove, ]
  }
}
```




```
sprintf("Number of training records after outlier removal: %i", nrow(training_data))
```

```
## [1] "Number of training records after outlier removal: 483819"
```

```
# Merge the train and test
dataset <- rbind(training_data, test_data)
```

So, we see that `2403` number of records are identified as outliers and are removed from the training set. Once the outliers are removed from the training set, we again merge the training and test dataset so that we can apply feature engineering process on the whole dataset.

Feature Engineering

This is the section to give free rein to our imagination and create/modify all the features that might improve the final result.

Feature creation

```
training_data <- dataset[dataset$dataType == "train",]
test_data <- dataset[dataset$dataType == "test",]

# feature representing the count of each user
userIDCount <- as.data.table(table(training_data$User_ID))
colnames(userIDCount) <- c("User_ID","User_ID_Count")

training_data <- merge(x = training_data, y = userIDCount, by = "User_ID", all.x = TRUE)
test_data <- merge(x = test_data, y = userIDCount, by = "User_ID", all.x = TRUE)

sum(is.na(test_data$User_ID_Count))
```

```
## [1] 0
```

```
# feature representing the count of each product
productIDCount <- as.data.table(table(training_data$Product_ID))
colnames(productIDCount) <- c("Product_ID","Product_ID_Count")

training_data <- merge(x = training_data, y = productIDCount, by = "Product_ID", all.x = TRUE
)
test_data <- merge(x = test_data, y = productIDCount, by = "Product_ID", all.x = TRUE)

sum(is.na(test_data$Product_ID_Count))
```

```
## [1] 0
```

```
# feature representing the average Purchase of each product
productIDMean <- ddply(training_data, .(Product_ID), summarize, Product_Mean=mean(Purchase))

training_data <- merge(x = training_data, y = productIDMean, by = "Product_ID", all.x = TRUE)
test_data <- merge(x = test_data, y = productIDMean, by = "Product_ID", all.x = TRUE)

sum(is.na(test_data$Product_Mean))
```

```
## [1] 14
```

```
test_data$Product_Mean[is.na(test_data$Product_Mean)] <- mean(training_data$Purchase)

# feature representing the average Purchase by each user
userIDMean <- ddply(training_data, .(User_ID), summarize, User_Mean=mean(Purchase))

training_data <- merge(x = training_data, y = userIDMean, by = "User_ID", all.x = TRUE)
test_data <- merge(x = test_data, y = userIDMean, by = "User_ID", all.x = TRUE)

sum(is.na(test_data$Product_Mean))
```

```
## [1] 0
```

```
# feature representing the proportion of times the user purchases the product more than the p
roduct's average
training_data$flag_high_product <- ifelse(training_data$Purchase > training_data$Product_Mea
n, 1, 0)
user_high_product <- ddply(training_data, .(User_ID), summarize, User_High_Product = mean(fla
g_high_product))

training_data <- merge(training_data, user_high_product, by="User_ID", all.x = TRUE)
test_data <- merge(test_data, user_high_product, by="User_ID", all.x = TRUE)

sum(is.na(test_data$User_High_Product))
```

```
## [1] 0
```

```
training_data[, c("flag_high_product") := NULL]

# feature representing the proportion of times the user purchases more than the his/her own a
verage
training_data$flag_high_user <- ifelse(training_data$Purchase > training_data$User_Mean, 1, 0
)
user_high_self <- ddply(training_data, .(User_ID), summarize, User_High_Self = mean(flag_high
_user))

training_data <- merge(training_data, user_high_self, by="User_ID", all.x = TRUE)
test_data <- merge(test_data, user_high_self, by="User_ID", all.x = TRUE)

sum(is.na(test_data$User_High_Self))
```

```
## [1] 0
```

```
training_data[, c("flag_high_user") := NULL]

# Merge the train and test
dataset <- rbind(training_data, test_data)
```

Feature transformation

```
# Fill the unknown values in product category 2 and 3
na_idx_pcat_2 <- dataset$Product_Category_2 == "UKN"
dataset[na_idx_pcat_2,]$Product_Category_2 <- dataset[na_idx_pcat_2,]$Product_Category_1
dataset$Product_Category_2 <- as.factor(as.character(dataset$Product_Category_2))

na_idx_pcat_3 <- dataset$Product_Category_3 == "UKN"
dataset[na_idx_pcat_3,]$Product_Category_3 <- dataset[na_idx_pcat_3,]$Product_Category_2
dataset$Product_Category_3 <- as.factor(as.character(dataset$Product_Category_3))
```

Skewness

While building predictive models we often see skewness in the target variable. Then we generally take transformations to make it more normal. We generally do it for linear models and not for tree based models. This actually means that our distribution is not normal, we are deliberately making it normal for prediction.

The way of getting rid of the skewness is to use the `log` (or the `log1p`) of the values of that feature, to flatten it. To reduce right skewness, take roots or logarithms or reciprocals (x to 1/x). This is the commonest problem in practice. To reduce left skewness, take squares or cubes or higher powers.

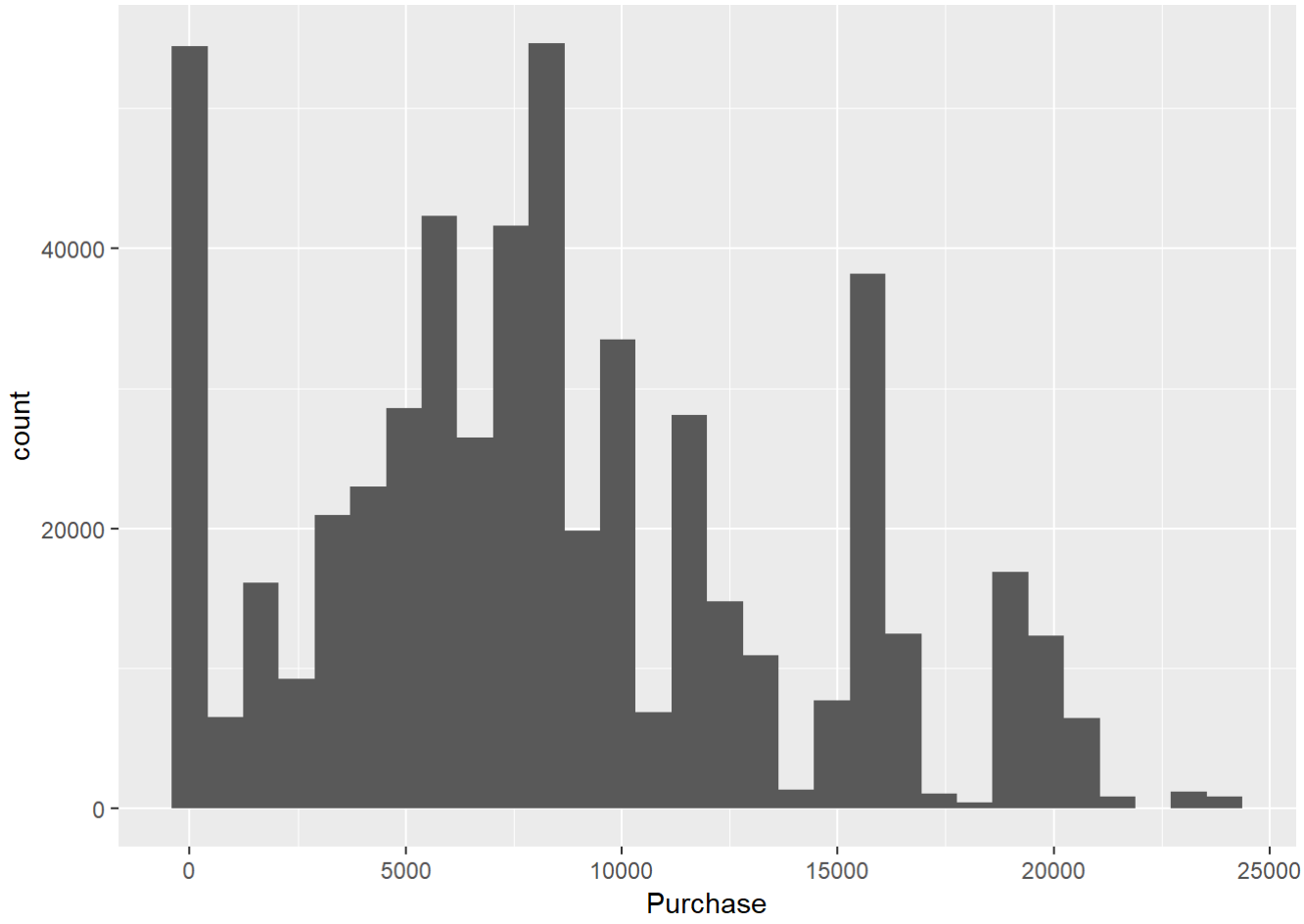
We now need to detect skewness in the Target value.

```
skewness(dataset$Purchase) #0.4091141 (Positive or right skewed, moderately skewed)
```

```
## [1] 0.4433281
```

```
ggplot(dataset, aes(x = Purchase)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We can see that the traget variable has a skewness of around 0.4, which means that the data is moderately skewed. In such case, let's not take any action for skewness transformation.

Train, Test and Validation Spliting

To facilitate the data cleaning and feature engineering we merged train and test datasets. We now split them again to create our final model.

```
training_data <- dataset[dataset$dataType == "train",]
test_data <- dataset[dataset$dataType == "test",]

training_data$dataType <- NULL
test_data$dataType <- NULL

training_data_split <- splitdt(training_data, 0.1, 123)
training_data_subsample <- training_data_split$test
```

Model creation and evaluation

In this section we will explore different types of models.

Before starting model building, let’s set the model training control parameters.

```
train_control<- trainControl(method="cv",
                             number=5,
                             search="grid",
                             savePredictions = T,
                             verboseIter = T)
```

Random Forest

Let’s build a Random Forest model using differnet grid parameters and training control parameters, mentioned previously.

```
# I am not executing the RF as this is not my best model and it takes a lot of time for compu
tation.

if(F){
  set.seed(1234)

  cv_rf<- caret::train(Purchase~.,
                       data=training_data_subsample[, -c("User_ID", "Product_ID")],
                       trControl=train_control,
                       method="rf",
                       metric="RMSE",
                       tuneGrid= expand.grid(.mtry=c(6,8,10,12,14)),
                       verbose=T,
                       importance=T)

  # Best tunning parameters obtained
  cv_rf$bestTune # .mtry = 10

  # Random Forest (No. Of Trees Vs Error)
  plot(cv_rf$finalModel, main = "Random Forest (No. Of Trees Vs Error)") #ntree = 100

  # Random Forest (No. Of Predictors Vs Accuracy
  plot(cv_rf, main = "Random Forest (No. Of Predictors Vs Accuracy)")

  # Top 20 important variable
  plot(varImp(cv_rf), main = "Variance Importance RandomForest", top = 20)

  # Predict for the rest of the training data
  train_pred_remaining <- predict(cv_rf, training_data_split$train)

  mape(training_data_split$train$Purchase,train_pred_remaining) #35%
}
```

XG BOOST

Let’s build a XG Boost model using differnet grid parameters and training control parameters, mentioned previously.

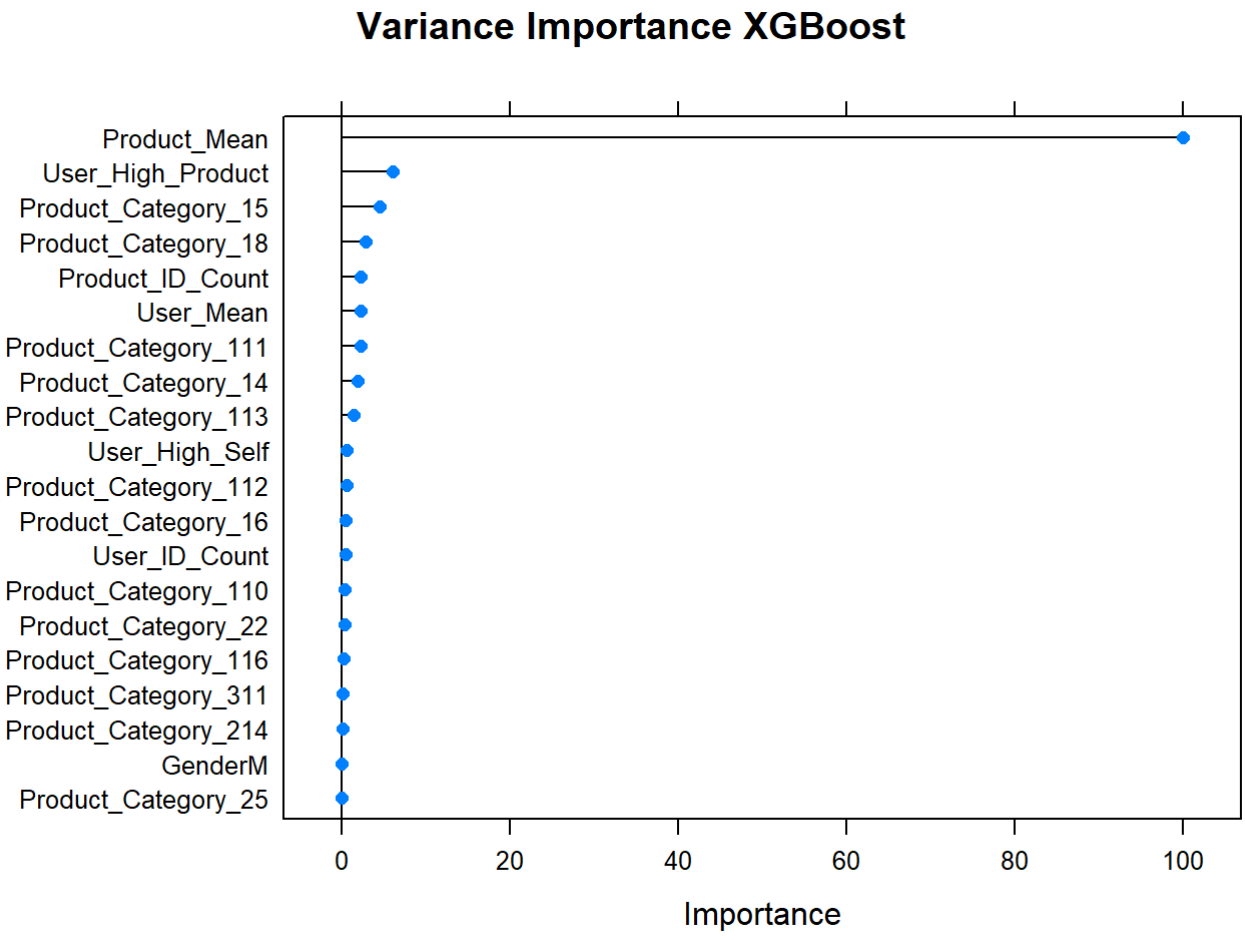
```
set.seed(12345)
if(F){
  tuneGridXGB <- expand.grid(
    eta = c(0.01, 0.1),
    max_depth = c(6, 8, 10),
    gamma = c(0.1, 1, 5),
    colsample_bytree = c(0.5, 0.75),
    min_child_weight = c(2, 5),
    subsample = c(0.50, 0.75),
    nrounds=c(150, 500)
  )
}

# For quick run: with optimize grid parameters
tuneGridXGB <- expand.grid(
  eta = c(0.01),
  max_depth = c(6),
  gamma = c(1),
  colsample_bytree = c(0.75),
  min_child_weight = c(5),
  subsample = c(0.5),
  nrounds=c(500)
)

# train the xgboost learner
cv_xgboost <- caret::train(Purchase~.,
                           data = training_data_subsample[, -c("User_ID", "Product_ID")],
                           method = 'xgbTree',
                           metric = 'RMSE',
                           trControl = train_control,
                           tuneGrid = tuneGridXGB,
                           verbose = T,
                           importance = T)
```

```
## + Fold1: eta=0.01, max_depth=6, gamma=1, colsample_bytree=0.75, min_child_weight=5, subsam
ple=0.5, nrounds=500
## - Fold1: eta=0.01, max_depth=6, gamma=1, colsample_bytree=0.75, min_child_weight=5, subsam
ple=0.5, nrounds=500
## + Fold2: eta=0.01, max_depth=6, gamma=1, colsample_bytree=0.75, min_child_weight=5, subsam
ple=0.5, nrounds=500
## - Fold2: eta=0.01, max_depth=6, gamma=1, colsample_bytree=0.75, min_child_weight=5, subsam
ple=0.5, nrounds=500
## + Fold3: eta=0.01, max_depth=6, gamma=1, colsample_bytree=0.75, min_child_weight=5, subsam
ple=0.5, nrounds=500
## - Fold3: eta=0.01, max_depth=6, gamma=1, colsample_bytree=0.75, min_child_weight=5, subsam
ple=0.5, nrounds=500
## + Fold4: eta=0.01, max_depth=6, gamma=1, colsample_bytree=0.75, min_child_weight=5, subsam
ple=0.5, nrounds=500
## - Fold4: eta=0.01, max_depth=6, gamma=1, colsample_bytree=0.75, min_child_weight=5, subsam
ple=0.5, nrounds=500
## + Fold5: eta=0.01, max_depth=6, gamma=1, colsample_bytree=0.75, min_child_weight=5, subsam
ple=0.5, nrounds=500
## - Fold5: eta=0.01, max_depth=6, gamma=1, colsample_bytree=0.75, min_child_weight=5, subsam
ple=0.5, nrounds=500
## Aggregating results
## Fitting final model on full training set
```

```
# Best tunning parameters obtained
cv_xgboost$bestTune
```

```
##   nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
## 1     500         6 0.01     1             0.75                5       0.5
```

```
#nrounds  max_depth  eta   gamma  colsample_bytree  min_child_weight   subsample
#500         6       0.01    1         0.75                5             0.5

# Let's check the model summary
# plot(cv_xgboost, main = "XGBoost Summary")

# Top 20 important variable
plot(varImp(cv_xgboost), main = "Variance Importance XGBoost", top = 20)
```




```
# Predict for the rest of the training data
train_pred_remaining <- predict(cv_xgboost, training_data_split$train)

# Calculate MAPE for the predicted values
mape(training_data_split$train$Purchase,train_pred_remaining) #26.63568%
```

```
## [1] 0.2663568
```

Model Prediction

In this section, we will predict the Target variable of the TEST dataset using previously created ML models such as

1. Random Forest
2. XG Boost

As XG Boost performed better that Random Forest, let's just predict the target for TEST dataset.

```
# Predict the Target variable for the TEST data using Random Forest
#final_pred_rf <- predict(cv_rf, test_data)

# Predict the Target variable for the TEST data using XG Boost
final_pred_xgboost <- predict(cv_xgboost, test_data)
```

Final Submission

In this section, we try to create CSV file for the submission.

As XGBoost performed the beest, we will do submission for XGBoost.

```
# Assign the output of either XGBoost or RF
final_pred_sub <- final_pred_xgboost

# Create the DataTable with Submission file format
submission <- data.table(User_ID = test_data$User_ID,
                         Product_ID = test_data$Product_ID,
                         Purchase= final_pred_sub)

# Assign the strings to the according integer
write.csv(submission, file = "Black_Friday_Final_Submission_v3.csv", row.names = FALSE)
```