XGBoost_Model.R

rahul

2024-12-03

```r
# Load necessary libraries for Transaction Evaluation
library(dplyr)
```

```
## Warning: package ’dplyr’ was built under R version 4.3.3
```

```
##
## Attaching package: ’dplyr’
```

```
## The following objects are masked from ’package:stats’:
##
##     filter, lag
```

```
## The following objects are masked from ’package:base’:
##
##     intersect, setdiff, setequal, union
```

```r
library(xgboost)
```

```
## Warning: package ’xgboost’ was built under R version 4.3.3
```

```
##
## Attaching package: ’xgboost’
```

```
## The following object is masked from ’package:dplyr’:
##
##     slice
```

```r
library(caret)
```

```
## Warning: package ’caret’ was built under R version 4.3.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package ’ggplot2’ was built under R version 4.3.3
```

```
## Loading required package: lattice
```

```r
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 4.3.3
```

```r
# Load dataset
data_clean <- read_excel("ACCT_Monitoring_FinalData.xlsx")

# Convert Character Columns to Factors
data_clean <- data_clean %>%
  mutate(across(c(FUEL_ONLY_PARENT_ACCT, CITY, STATE, ZIP, LOCK_CODE, LOCK_REASON, LOCK_TYPE, PORTFOLIO

# Derive TransactionPlatform
data_clean <- data_clean %>%
  mutate(TransactionPlatform = case_when(
    NONFUEL_SPEND > FUEL_SPEND ~ "Non-Fuel Focused",
    FUEL_SPEND > NONFUEL_SPEND ~ "Fuel Focused",
    TRUE ~ "Unknown"
  ))

# Derive TransactionTimeFrame
data_clean <- data_clean %>%
  mutate(TransactionTimeFrame_30 = ifelse(as.numeric(difftime(OPT_IN_DATE + 30, FLEETCOR_OPEN_DATE, uni

# Create additional features
data_clean <- data_clean %>%
  mutate(SpendUtilization = TOT_SPEND / CREDIT_LIMIT,
         AverageSpendPerTransaction = TOT_SPEND / TOT_NUM_TRX,
         DefaultRisk = as.factor(ifelse(WO_AMOUNT > 0, 1, 0)))

# Handle Missing Values
data_clean$AverageSpendPerTransaction[is.na(data_clean$AverageSpendPerTransaction)] <- 0

# Train-Test Split
set.seed(123)
train_index <- createDataPartition(data_clean$DefaultRisk, p = 0.8, list = FALSE)
train_data <- data_clean[train_index, ]
test_data <- data_clean[-train_index, ]

# Convert date columns to numeric (e.g., as timestamps)
train_data <- train_data %>%
  mutate(OPT_IN_DATE = as.numeric(OPT_IN_DATE),
         FLEETCOR_OPEN_DATE = as.numeric(FLEETCOR_OPEN_DATE))

test_data <- test_data %>%
  mutate(OPT_IN_DATE = as.numeric(OPT_IN_DATE),
         FLEETCOR_OPEN_DATE = as.numeric(FLEETCOR_OPEN_DATE))

# Convert factors to numeric, including derived columns
train_data_numeric <- train_data %>%
  mutate(across(where(is.factor), ~ as.integer(as.factor(.)))) %>%
  mutate(across(where(is.character), ~ as.integer(as.factor(.))))  # This line is mainly for safety

test_data_numeric <- test_data %>%
  mutate(across(where(is.factor), ~ as.integer(as.factor(.)))) %>%
  mutate(across(where(is.character), ~ as.integer(as.factor(.))))  # This line is mainly for safety

# Check the structure of the numeric data
str(train_data_numeric)
```

```
## tibble [401 x 41] (S3: tbl_df/tbl/data.frame)
##  $ FAKE_ACCTCODE              : num [1:401] 111114 111117 111120 111126 111127 ...
##  $ OPT_IN_DATE                : num [1:401] 1.66e+09 1.66e+09 1.68e+09 1.65e+09 1.62e+09 ...
##  $ FUEL_ONLY_PARENT_ACCT      : int [1:401] 1 1 1 1 1 1 1 1 1 1 ...
##  $ CLI_AMOUNT                 : num [1:401] 6000 62000 58000 60000 5100 11500 5000 4000 4500 7500 ...
##  $ FLEETCOR_OPEN_DATE         : num [1:401] 1.57e+09 1.57e+09 1.57e+09 1.57e+09 1.57e+09 ...
##  $ CITY                       : int [1:401] 67 343 159 359 138 106 57 257 232 13 ...
##  $ STATE                      : int [1:401] 42 10 42 42 27 35 14 14 6 11 ...
##  $ ZIP                        : int [1:401] 373 181 381 372 88 470 321 317 455 140 ...
##  $ TERM_DAYS                  : num [1:401] 7 7 15 30 30 30 15 15 15 7 ...
##  $ DUE_DAYS                   : num [1:401] 7 7 13 14 14 15 15 15 14 7 ...
##  $ LOCK_DAYS                  : num [1:401] 3 18 18 5 10 5 10 10 7 3 ...
##  $ CREDIT_LIMIT               : num [1:401] 7200 62000 58000 75000 5100 ...
##  $ LOCK_CODE                  : int [1:401] 5 12 12 12 7 12 12 7 3 7 ...
##  $ LOCK_REASON                : int [1:401] 9 12 12 12 2 12 12 2 8 2 ...
##  $ LOCK_TYPE                  : int [1:401] 1 3 3 3 1 3 3 1 1 1 ...
##  $ DEPOSIT_FLAG               : num [1:401] 0 0 0 0 1 0 0 1 0 0 ...
##  $ PORTFOLIO                  : int [1:401] 4 4 4 2 2 5 6 6 1 4 ...
##  $ LOB_REPORTING              : int [1:401] 1 1 1 2 2 2 2 2 2 1 ...
##  $ FUEL_SPEND                 : num [1:401] 54262 5527 154562 105351 0 ...
##  $ NONFUEL_SPEND              : num [1:401] 1865 6971 232 100544 0 ...
##  $ TOT_SPEND                  : num [1:401] 56127 12498 154794 205895 0 ...
##  $ PAYMENT_AMOUNT             : num [1:401] 64416 10220 160157 108386 0 ...
##  $ NO_OF_PAYMENT              : num [1:401] 28 45 27 14 0 14 28 8 11 0 ...
##  $ NSF_AMT                    : num [1:401] 0 0 0 0 0 0 0 0 0 0 ...
##  $ NSF_PMTS                   : num [1:401] 0 0 0 0 0 0 0 0 0 0 ...
##  $ PAYDEX                     : num [1:401] 0 68.6 79.5 58.9 78 ...
##  $ VANTAGE_SCORE              : num [1:401] 721 825 797 0 0 ...
##  $ FUEL_NUM_TRX               : num [1:401] 148 115 2421 1526 0 ...
##  $ NONFUEL_NUM_TRX            : num [1:401] 6 30 7 673 0 190 3 5 12 4 ...
##  $ FUEL_TRX_AMT               : num [1:401] 54262 5527 154562 105351 0 ...
##  $ NONFUEL_TRX_AMT            : num [1:401] 1865 6971 232 100544 0 ...
##  $ SEGMENT_SCORE              : num [1:401] 45.1 33.4 3 42.1 0 ...
##  $ TOT_NET_REV                : num [1:401] 5711 1964 3995 15659 0 ...
##  $ WO_AMOUNT                  : num [1:401] 0 0 0 0 0 ...
##  $ TOT_TRX_AMT                : num [1:401] 56127 12498 154794 205895 0 ...
##  $ TOT_NUM_TRX                : num [1:401] 154 145 2428 2199 0 ...
##  $ TransactionPlatform        : int [1:401] 1 2 1 1 3 2 1 1 1 2 ...
##  $ TransactionTimeFrame_30    : int [1:401] 1 1 1 1 1 1 1 1 1 1 ...
##  $ SpendUtilization           : num [1:401] 7.795 0.202 2.669 2.745 0 ...
##  $ AverageSpendPerTransaction: num [1:401] 364.5 86.2 63.8 93.6 0 ...
##  $ DefaultRisk                : int [1:401] 1 1 1 1 1 1 1 2 2 2 ...
```

```
# Create the DMatrix, including derived columns
train_matrix <- xgb.DMatrix(data = as.matrix(train_data_numeric), label = as.numeric(train_data$Default
test_matrix <- xgb.DMatrix(data = as.matrix(test_data_numeric), label = as.numeric(test_data$DefaultRis
```

```r
# Set Parameters and Train the Model
params <- list(
  objective = "binary:logistic",
  eval_metric = "auc",
  max_depth = 6,
  eta = 0.3
)

# Train the XGBoost Model
xgb_model <- xgb.train(
  params = params,
  data = train_matrix,
  nrounds = 100,
  watchlist = list(train = train_matrix, test = test_matrix),
  verbose = 0
)

summary(xgb_model)
```

```
##                 Length Class              Mode
## handle              1  xgb.Booster.handle externalptr
## raw             74477  -none-             raw
## niter               1  -none-             numeric
## evaluation_log      3  data.table         list
## call                6  -none-             call
## params              5  -none-             list
## callbacks           1  -none-             list
## feature_names      41  -none-             character
## nfeatures           1  -none-             numeric
```

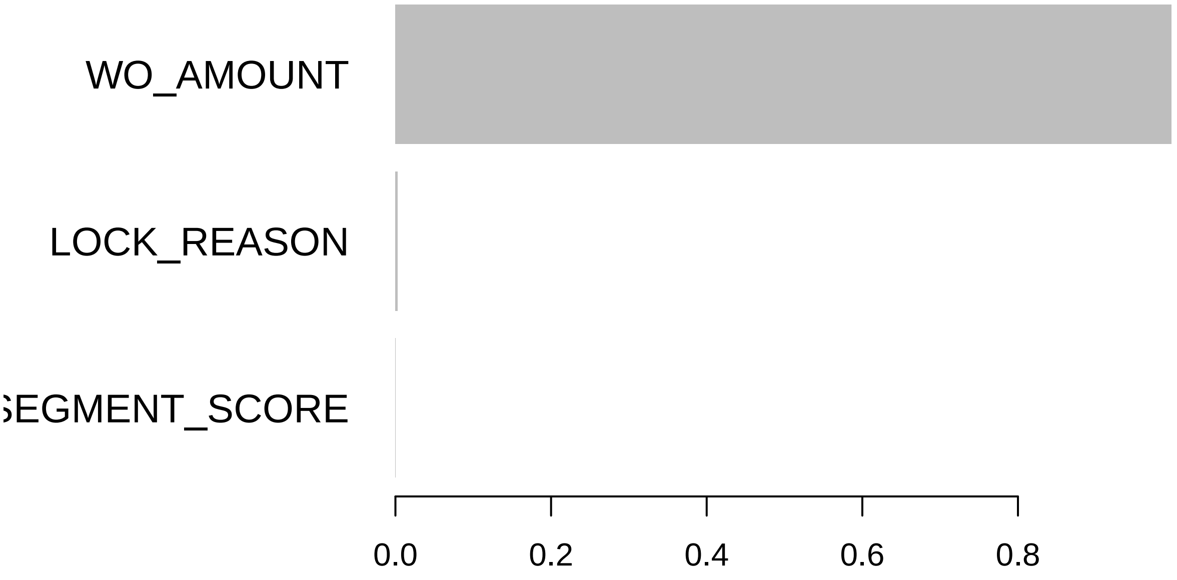
```r
# Evaluate Model Performance
pred_probs <- predict(xgb_model, test_matrix)
pred_labels <- ifelse(pred_probs > 0.5, 1, 0)  # Convert probabilities to binary labels

# Create a confusion matrix to evaluate the model
conf_matrix <- confusionMatrix(as.factor(pred_labels), as.factor(as.numeric(test_data$DefaultRisk) - 1))
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 79  0
##          1  0 20
##
##                Accuracy : 1
##                  95% CI : (0.9634, 1)
##     No Information Rate : 0.798
##     P-Value [Acc > NIR] : 1.982e-10
```

```
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.000
##             Specificity : 1.000
##          Pos Pred Value : 1.000
##          Neg Pred Value : 1.000
##              Prevalence : 0.798
##          Detection Rate : 0.798
##    Detection Prevalence : 0.798
##       Balanced Accuracy : 1.000
##
##        'Positive' Class : 0
##
```

```
# Feature Importance
importance <- xgb.importance(feature_names = colnames(train_data_numeric), model = xgb_model)
xgb.plot.importance(importance)
```



```
# Display the feature importance values
importance_values <- importance %>%
  arrange(desc(Gain)) # Arrange by Gain or any other metric (e.g., Cover, Frequency)

print(importance_values)
```

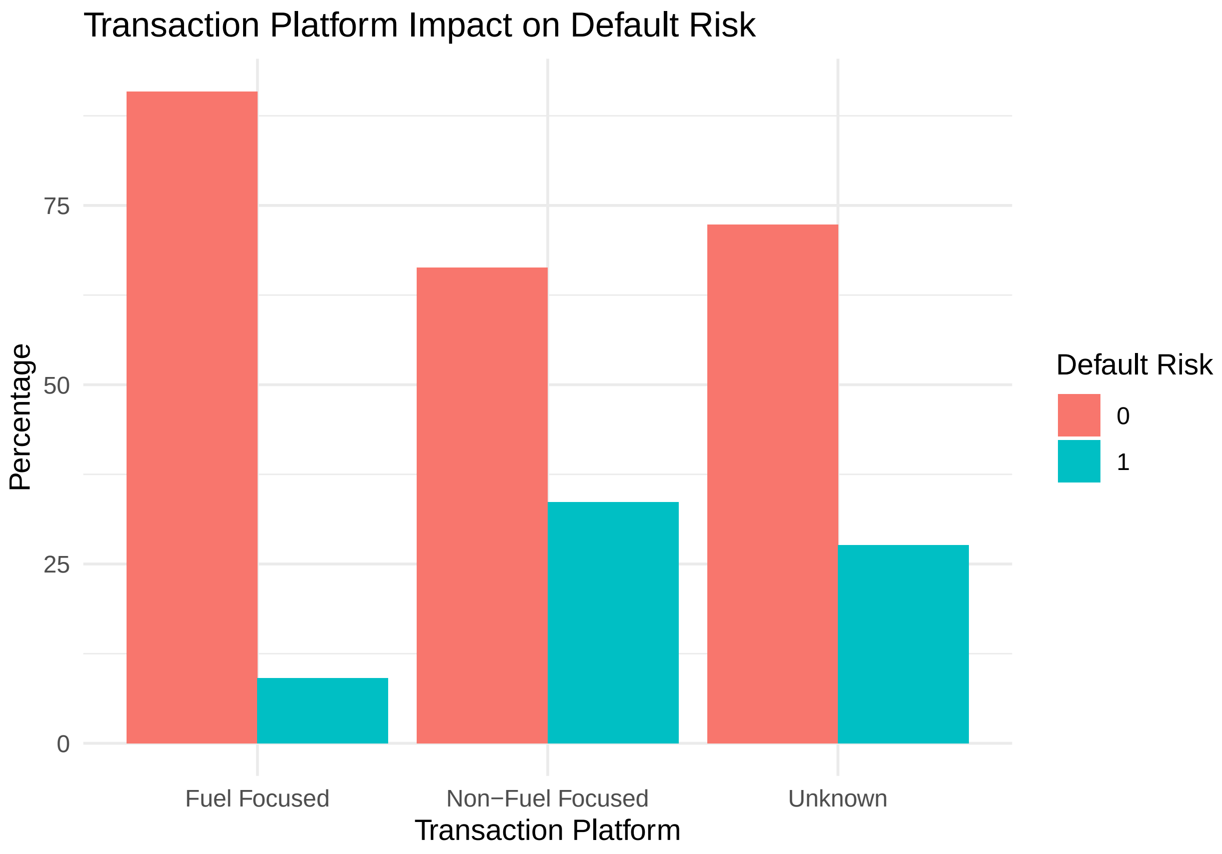
```
##           Feature         Gain      Cover  Frequency   Importance
##            <char>        <num>      <num>      <num>        <num>
## 1:      WO_AMOUNT 0.9971426080 0.97877916 0.76190476 0.9971426080
## 2:    LOCK_REASON 0.0027021478 0.01751866 0.19047619 0.0027021478
## 3: SEGMENT_SCORE 0.0001552442 0.00370218 0.04761905 0.0001552442
```

```
# Load necessary libraries for visualization
library(ggplot2)

# Analyze TransactionPlatform
transaction_platform_analysis <- data_clean %>%
  group_by(TransactionPlatform, DefaultRisk) %>%
  summarise(Count = n()) %>%
  mutate(Percentage = Count / sum(Count) * 100)
```

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

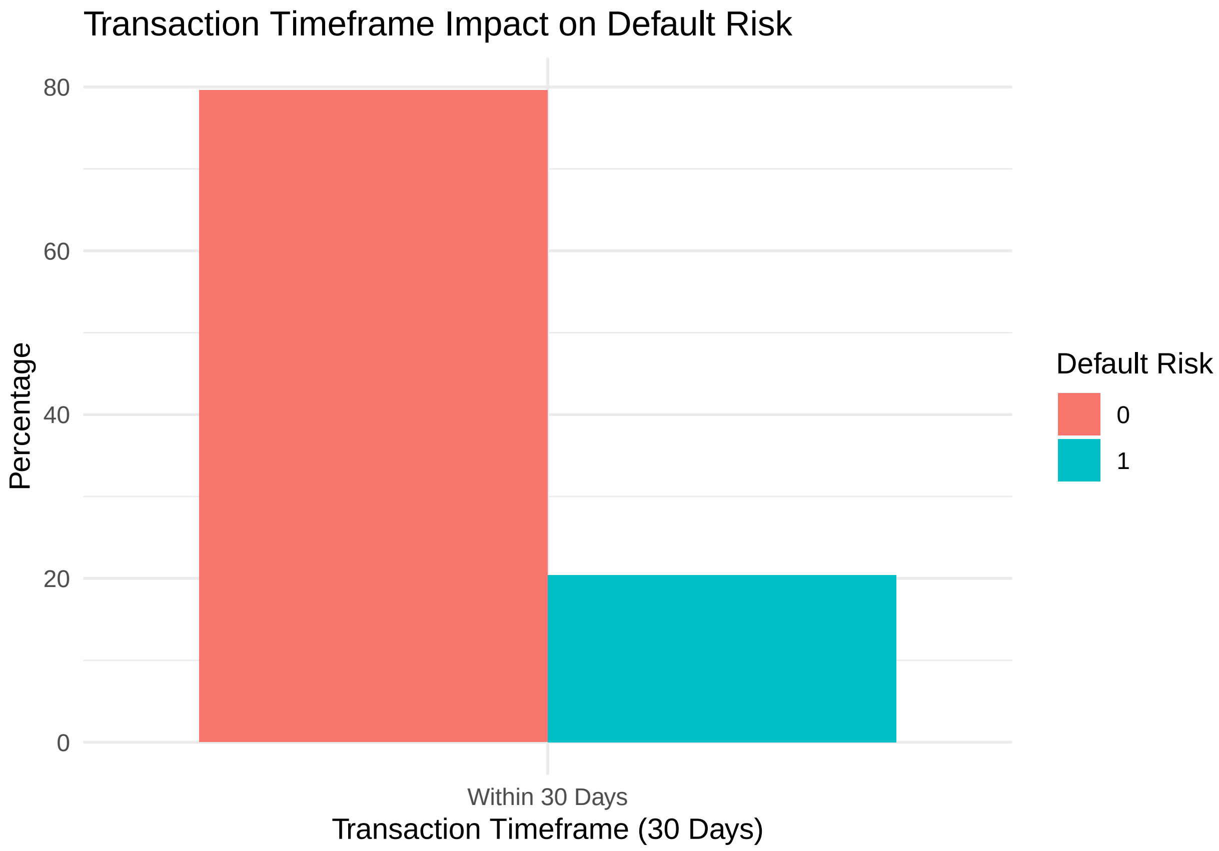
```
ggplot(transaction_platform_analysis, aes(x = TransactionPlatform, y = Percentage, fill = as.factor(Def
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Transaction Platform Impact on Default Risk",
       x = "Transaction Platform",
       y = "Percentage",
       fill = "Default Risk") +
  theme_minimal()
```



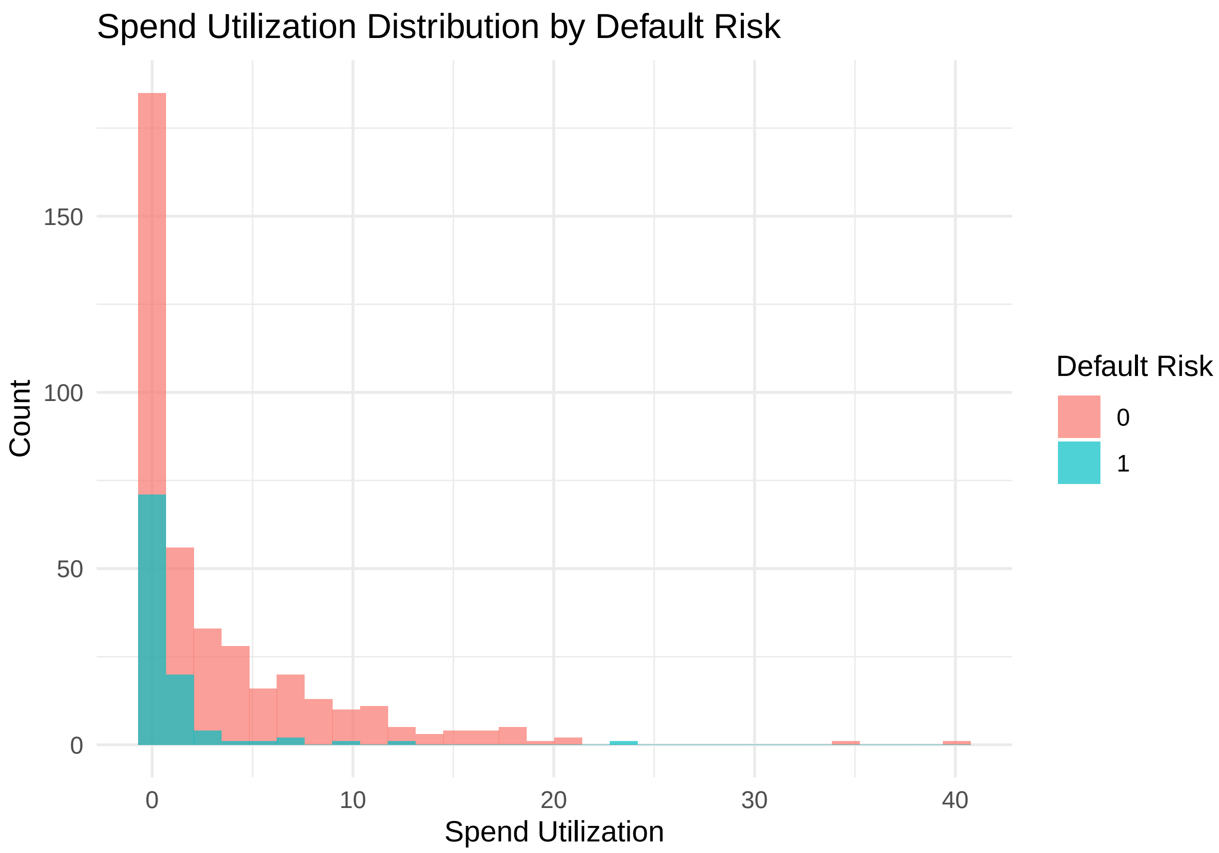
```r
# Analyze TransactionTimeFrame
transaction_timeframe_analysis <- data_clean %>%
  group_by(TransactionTimeFrame_30, DefaultRisk) %>%
  summarise(Count = n()) %>%
  mutate(Percentage = Count / sum(Count) * 100)
```

```
## `summarise()` has grouped output by 'TransactionTimeFrame_30'. You can override
## using the `.groups` argument.
```

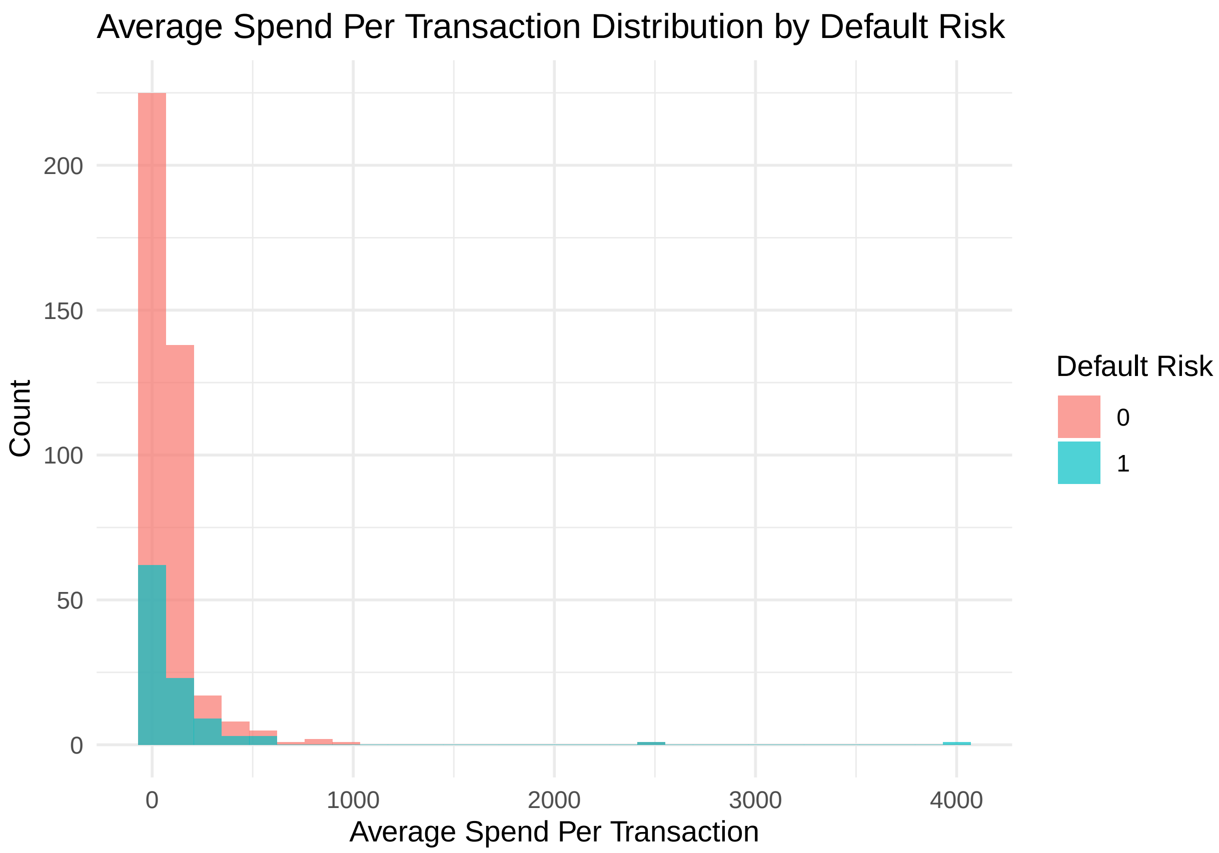
```r
ggplot(transaction_timeframe_analysis, aes(x = TransactionTimeFrame_30, y = Percentage, fill = as.facto
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Transaction Timeframe Impact on Default Risk",
       x = "Transaction Timeframe (30 Days)",
       y = "Percentage",
       fill = "Default Risk") +
  theme_minimal()
```



```r
# Analyze SpendUtilization
ggplot(data_clean, aes(x = SpendUtilization, fill = as.factor(DefaultRisk))) +
  geom_histogram(bins = 30, alpha = 0.7, position = "identity") +
  labs(title = "Spend Utilization Distribution by Default Risk",
       x = "Spend Utilization",
       y = "Count",
       fill = "Default Risk") +
  theme_minimal()
```



```r
# Analyze AverageSpendPerTransaction
ggplot(data_clean, aes(x = AverageSpendPerTransaction, fill = as.factor(DefaultRisk))) +
  geom_histogram(bins = 30, alpha = 0.7, position = "identity") +
  labs(title = "Average Spend Per Transaction Distribution by Default Risk",
       x = "Average Spend Per Transaction",
       y = "Count",
       fill = "Default Risk") +
  theme_minimal()
```


Average Spend Per Transaction Distribution by Default Risk
Count
200
150
100
50
0
0
1000
2000
3000
4000
Average Spend Per Transaction
Default Risk
0
1