Credit Card Fraud Detection

A4 Project Report

Group_A4

2025-06-25

Table of contents

0.1 Load the Dataset . . . 1
0.2 EDA . . . 6
0.3 Resampling . . . 11
0.4 Basic Model Fitting with Original Data Set . . . 13
0.5 Model with Regularization (LASSO) . . . 15
0.6 Autoencoders for Anomaly Detection . . . 17
0.7 XGBoost Model Original Data . . . 19
0.8 XGBoost with Resampling . . . 23
0.9 Support Vector Machines . . . 26
0.10 Random Forest . . . 28
0.11 Evaluating all models on Confusion Matrix . . . 29

0.1 Load the Dataset

```
# Load packages
suppressPackageStartupMessages({
library(tidyverse)
library(skimr)
library(reshape2)
library(corrplot)
library(scales)  # for nice axis labels
library(caret)
library(MASS)
library(car)
library(h2o)
library(xgboost)
library(pROC)
library(e1071)
library(randomForest)
library(ROSE)
library(DMwR)
```

```r
# install.packages("remotes")
# remotes::install_github("cran/DMwR")

# Load DMwR and convert target to factor

})
```

```
Warning: package 'dplyr' was built under R version 4.4.3

Warning: package 'skimr' was built under R version 4.4.2

Warning: package 'reshape2' was built under R version 4.4.3

Warning: package 'corrplot' was built under R version 4.4.2

Warning: package 'caret' was built under R version 4.4.3

Warning: package 'car' was built under R version 4.4.2

Warning: package 'carData' was built under R version 4.4.2

Warning: package 'h2o' was built under R version 4.4.3

Warning: package 'xgboost' was built under R version 4.4.3

Warning: package 'pROC' was built under R version 4.4.3

Warning: package 'e1071' was built under R version 4.4.3

Warning: package 'randomForest' was built under R version 4.4.3

Warning: package 'ROSE' was built under R version 4.4.3
```

```r
# Read the dataset
df <- read.csv("creditcard.csv")

summary(df)
```

```
      Time              V1                  V2                  V3
 Min.   :     0   Min.   :-56.40751   Min.   :-72.71573   Min.   :-48.3256
 1st Qu.: 54202   1st Qu.: -0.92037   1st Qu.: -0.59855   1st Qu.: -0.8904
 Median : 84692   Median :  0.01811   Median :  0.06549   Median :  0.1799
 Mean   : 94814   Mean   :  0.00000   Mean   :  0.00000   Mean   :  0.0000
 3rd Qu.:139321   3rd Qu.:  1.31564   3rd Qu.:  0.80372   3rd Qu.:  1.0272
 Max.   :172792   Max.   :  2.45493   Max.   : 22.05773   Max.   :  9.3826
       V4                 V5                   V6                 V7
 Min.   :-5.68317   Min.   :-113.74331   Min.   :-26.1605   Min.   :-43.5572
 1st Qu.:-0.84864   1st Qu.:  -0.69160   1st Qu.: -0.7683   1st Qu.: -0.5541
 Median :-0.01985   Median :  -0.05434   Median : -0.2742   Median :  0.0401
 Mean   : 0.00000   Mean   :   0.00000   Mean   :  0.0000   Mean   :  0.0000
 3rd Qu.: 0.74334   3rd Qu.:   0.61193   3rd Qu.:  0.3986   3rd Qu.:  0.5704
 Max.   :16.87534   Max.   :  34.80167   Max.   : 73.3016   Max.   :120.5895
       V8                  V9                 V10                 V11
 Min.   :-73.21672   Min.   :-13.43407   Min.   :-24.58826   Min.   :-4.79747
 1st Qu.: -0.20863   1st Qu.: -0.64310   1st Qu.: -0.53543   1st Qu.:-0.76249
 Median :  0.02236   Median : -0.05143   Median : -0.09292   Median :-0.03276
 Mean   :  0.00000   Mean   :  0.00000   Mean   :  0.00000   Mean   : 0.00000
 3rd Qu.:  0.32735   3rd Qu.:  0.59714   3rd Qu.:  0.45392   3rd Qu.: 0.73959
 Max.   : 20.00721   Max.   : 15.59500   Max.   : 23.74514   Max.   :12.01891
      V12                V13                V14                V15
 Min.   :-18.6837   Min.   :-5.79188   Min.   :-19.2143   Min.   :-4.49894
 1st Qu.: -0.4056   1st Qu.:-0.64854   1st Qu.: -0.4256   1st Qu.:-0.58288
 Median :  0.1400   Median :-0.01357   Median :  0.0506   Median : 0.04807
 Mean   :  0.0000   Mean   : 0.00000   Mean   :  0.0000   Mean   : 0.00000
 3rd Qu.:  0.6182   3rd Qu.: 0.66251   3rd Qu.:  0.4931   3rd Qu.: 0.64882
 Max.   :  7.8484   Max.   : 7.12688   Max.   : 10.5268   Max.   : 8.87774
      V16                 V17                 V18
 Min.   :-14.12985   Min.   :-25.16280   Min.   :-9.498746
 1st Qu.: -0.46804   1st Qu.: -0.48375   1st Qu.:-0.498850
 Median :  0.06641   Median : -0.06568   Median :-0.003636
 Mean   :  0.00000   Mean   :  0.00000   Mean   : 0.000000
 3rd Qu.:  0.52330   3rd Qu.:  0.39968   3rd Qu.: 0.500807
 Max.   : 17.31511   Max.   :  9.25353   Max.   : 5.041069
      V19                 V20                 V21
 Min.   :-7.213527   Min.   :-54.49772   Min.   :-34.83038
 1st Qu.:-0.456299   1st Qu.: -0.21172   1st Qu.: -0.22839
 Median : 0.003735   Median : -0.06248   Median : -0.02945
 Mean   : 0.000000   Mean   :  0.00000   Mean   :  0.00000
 3rd Qu.: 0.458949   3rd Qu.:  0.13304   3rd Qu.:  0.18638
 Max.   : 5.591971   Max.   : 39.42090   Max.   : 27.20284
      V22                  V23                 V24
 Min.   :-10.933144   Min.   :-44.80774   Min.   :-2.83663
 1st Qu.: -0.542350   1st Qu.: -0.16185   1st Qu.:-0.35459
 Median :  0.006782   Median : -0.01119   Median : 0.04098
 Mean   :  0.000000   Mean   :  0.00000   Mean   : 0.00000
 3rd Qu.:  0.528554   3rd Qu.:  0.14764   3rd Qu.: 0.43953
```

```
 Max.   : 10.503090   Max.   : 22.52841   Max.   : 4.58455
      V25                 V26                 V27
 Min.   :-10.29540   Min.   :-2.60455   Min.   :-22.565679
 1st Qu.: -0.31715   1st Qu.:-0.32698   1st Qu.: -0.070840
 Median :  0.01659   Median :-0.05214   Median :  0.001342
 Mean   :  0.00000   Mean   : 0.00000   Mean   :  0.000000
 3rd Qu.:  0.35072   3rd Qu.: 0.24095   3rd Qu.:  0.091045
 Max.   :  7.51959   Max.   : 3.51735   Max.   : 31.612198
      V28                Amount              Class
 Min.   :-15.43008   Min.   :    0.00   Min.   :0.000000
 1st Qu.: -0.05296   1st Qu.:    5.60   1st Qu.:0.000000
 Median :  0.01124   Median :   22.00   Median :0.000000
 Mean   :  0.00000   Mean   :   88.35   Mean   :0.001728
 3rd Qu.:  0.07828   3rd Qu.:   77.17   3rd Qu.:0.000000
 Max.   : 33.84781   Max.   :25691.16   Max.   :1.000000
```

```r
df$Hour <- (df$Time %% (60*60*24)) / 3600  # convert to hour in day
dplyr::select(df, !"Time") -> df

skim(df)
```

Table 1: Data summary

Name	df
Number of rows	284807
Number of columns	31
Column type frequency:	
numeric	31
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
V1	0	1	0.00	1.96	-56.41	-0.92	0.02	1.32	2.45	
V2	0	1	0.00	1.65	-72.72	-0.60	0.07	0.80	22.06	
V3	0	1	0.00	1.52	-48.33	-0.89	0.18	1.03	9.38	
V4	0	1	0.00	1.42	-5.68	-0.85	-0.02	0.74	16.88	
V5	0	1	0.00	1.38	-113.74	-0.69	-0.05	0.61	34.80	
V6	0	1	0.00	1.33	-26.16	-0.77	-0.27	0.40	73.30	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
V7	0	1	0.00	1.24	-43.56	-0.55	0.04	0.57	120.59	
V8	0	1	0.00	1.19	-73.22	-0.21	0.02	0.33	20.01	
V9	0	1	0.00	1.10	-13.43	-0.64	-0.05	0.60	15.59	
V10	0	1	0.00	1.09	-24.59	-0.54	-0.09	0.45	23.75	
V11	0	1	0.00	1.02	-4.80	-0.76	-0.03	0.74	12.02	
V12	0	1	0.00	1.00	-18.68	-0.41	0.14	0.62	7.85	
V13	0	1	0.00	1.00	-5.79	-0.65	-0.01	0.66	7.13	
V14	0	1	0.00	0.96	-19.21	-0.43	0.05	0.49	10.53	
V15	0	1	0.00	0.92	-4.50	-0.58	0.05	0.65	8.88	
V16	0	1	0.00	0.88	-14.13	-0.47	0.07	0.52	17.32	
V17	0	1	0.00	0.85	-25.16	-0.48	-0.07	0.40	9.25	
V18	0	1	0.00	0.84	-9.50	-0.50	0.00	0.50	5.04	
V19	0	1	0.00	0.81	-7.21	-0.46	0.00	0.46	5.59	
V20	0	1	0.00	0.77	-54.50	-0.21	-0.06	0.13	39.42	
V21	0	1	0.00	0.73	-34.83	-0.23	-0.03	0.19	27.20	
V22	0	1	0.00	0.73	-10.93	-0.54	0.01	0.53	10.50	
V23	0	1	0.00	0.62	-44.81	-0.16	-0.01	0.15	22.53	
V24	0	1	0.00	0.61	-2.84	-0.35	0.04	0.44	4.58	
V25	0	1	0.00	0.52	-10.30	-0.32	0.02	0.35	7.52	
V26	0	1	0.00	0.48	-2.60	-0.33	-0.05	0.24	3.52	
V27	0	1	0.00	0.40	-22.57	-0.07	0.00	0.09	31.61	
V28	0	1	0.00	0.33	-15.43	-0.05	0.01	0.08	33.85	
Amount	0	1	88.35	250.12	0.00	5.60	22.00	77.16	25691.16	
Class	0	1	0.00	0.04	0.00	0.00	0.00	0.00	1.00	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Hour	0	1	14.54	5.85	0.00	10.60	15.01	19.33	24.00	

All predictors are numeric.

Class is extremely imbalanced, so we must handle this before modeling.

Many PCA variables have non-normal, high-variance distributions → visual EDA (boxplots, density plots) will help us decide if some features are especially important.

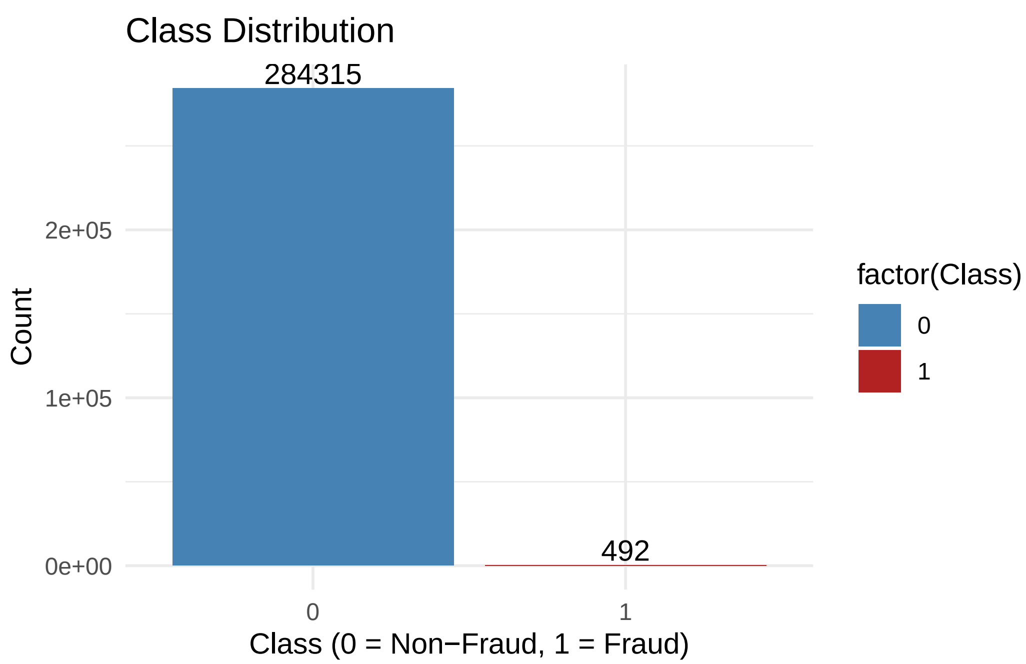
Amount and Time are not standardized — these need scaling or transformation.

0.2 EDA

1. Visualize Class Imbalance

```
ggplot(df, aes(x = factor(Class))) +
  geom_bar(aes(fill = factor(Class))) +
  geom_text(stat = "count", aes(label = ..count..), vjust = -0.2) +
  scale_fill_manual(values = c("steelblue", "firebrick")) +
  labs(title = "Class Distribution", x = "Class (0 = Non-Fraud, 1 = Fraud)", y = "Count") +
  theme_minimal()
```

```
Warning: The dot-dot notation (`..count..`) was deprecated in ggplot2 3.4.0.
i Please use `after_stat(count)` instead.
```



This bar chart shows the number of transactions for each class in our dataset. We clearly see a massive imbalance:

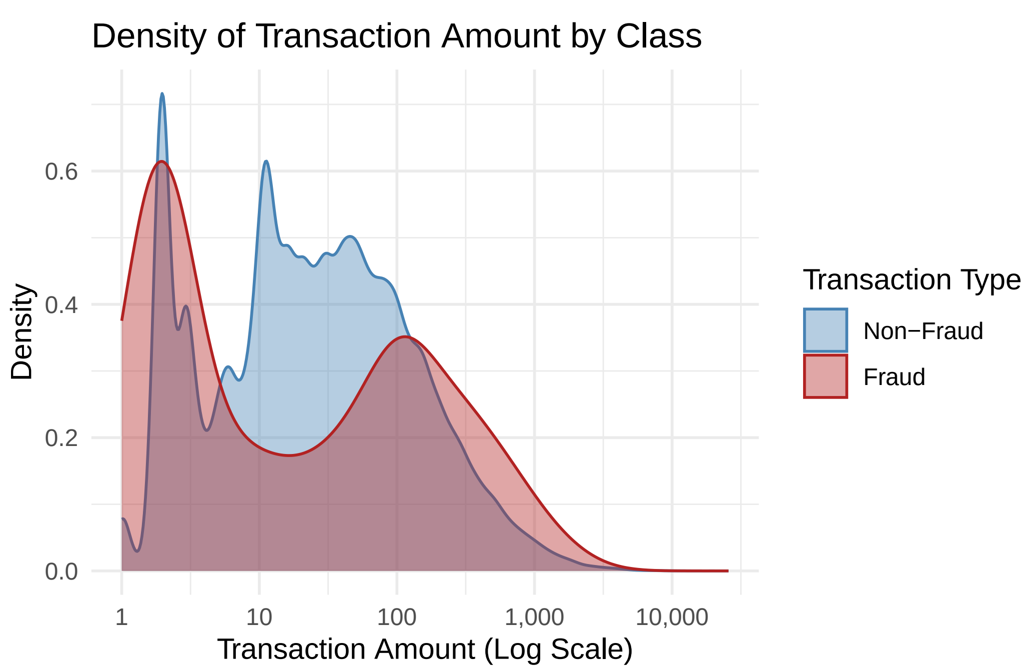
There are 284,315 non-fraudulent transactions (class 0), making up nearly 99.83% of the data.

In contrast, there are only 492 fraudulent transactions (class 1), which is about 0.17% of the total.

If we trained a model without addressing this imbalance, it might just predict "non-fraud" for everything and still appear 99.8% "accurate" — but it would fail to catch real fraud. This makes it essential to use resampling methods (like SMOTE or ROSE).

2. Visualize Transaction Amount by Class

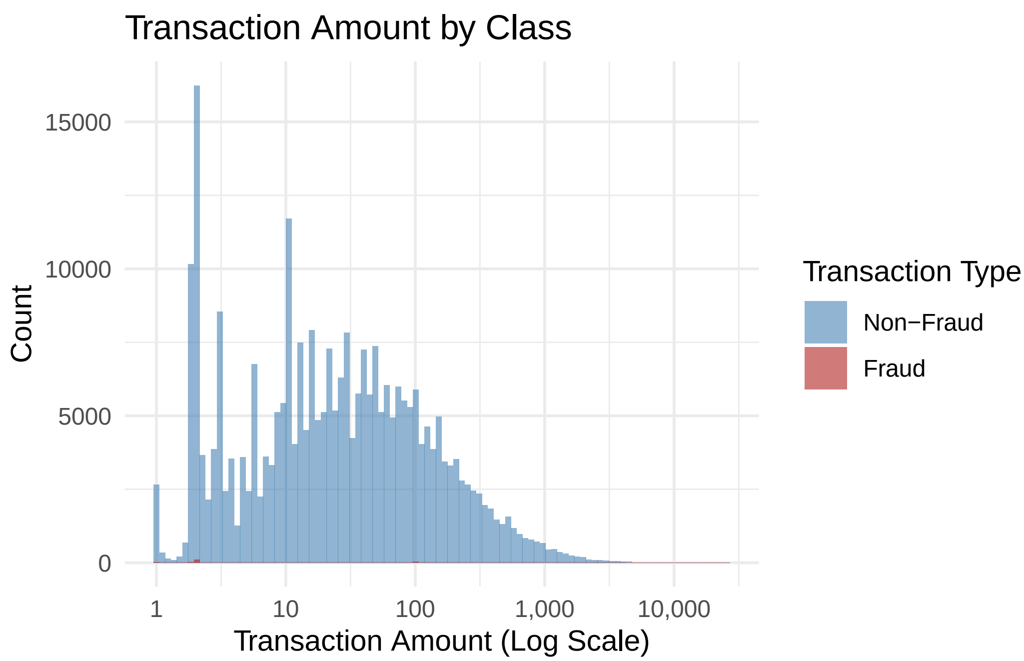
```
ggplot(df, aes(x = Amount + 1, color = factor(Class), fill = factor(Class))) +
  geom_density(alpha = 0.4) +
  scale_x_log10(labels = comma) +
  scale_fill_manual(values = c("steelblue", "firebrick"), labels = c("Non-Fraud", "Fraud")) +
  scale_color_manual(values = c("steelblue", "firebrick"), labels = c("Non-Fraud", "Fraud")) +
  labs(
    title = "Density of Transaction Amount by Class",
    x = "Transaction Amount (Log Scale)",
    y = "Density",
    fill = "Transaction Type",
    color = "Transaction Type"
  ) +
  theme_minimal()
```



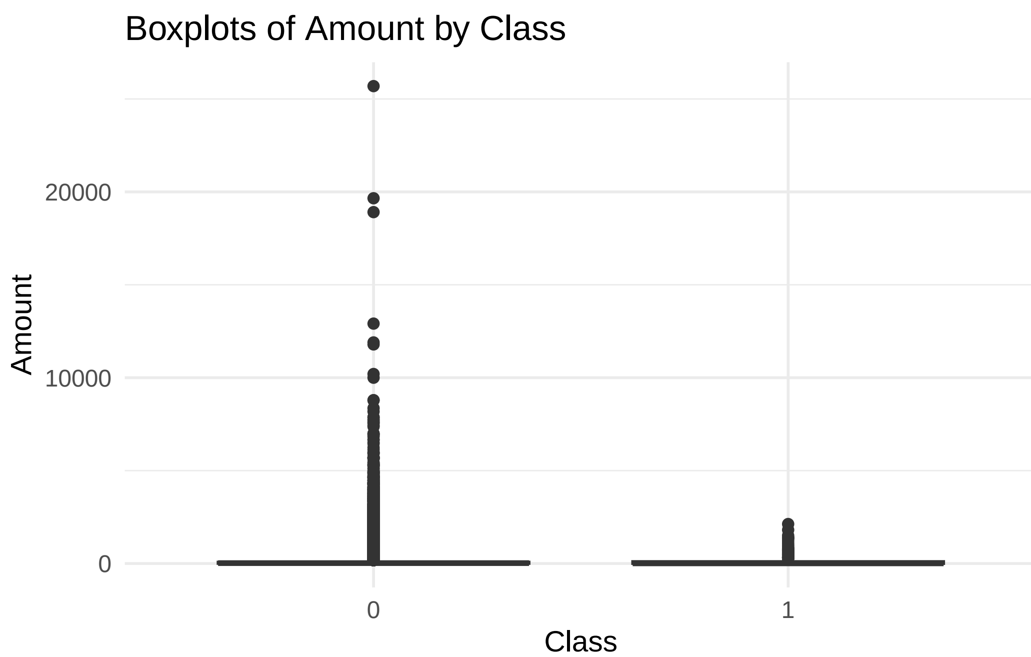
This plot reveals that fraudulent transactions tend to cluster around lower amounts, while non-fraudulent transactions are spread across a broader range. Fraud shows higher density below 100 units, hinting at a preference for small-value fraudulent actions.

```
ggplot(df, aes(x = Amount + 1, fill = factor(Class))) +
  geom_histogram(bins = 100, position = "identity", alpha = 0.6) +
  scale_x_log10(labels = comma) +
```

```r
  scale_fill_manual(values = c("steelblue", "firebrick"), labels = c("Non-Fraud", "Fraud")) +
  labs(
    title = "Transaction Amount by Class",
    x = "Transaction Amount (Log Scale)",
    y = "Count",
    fill = "Transaction Type"
  ) +
  theme_minimal()
```

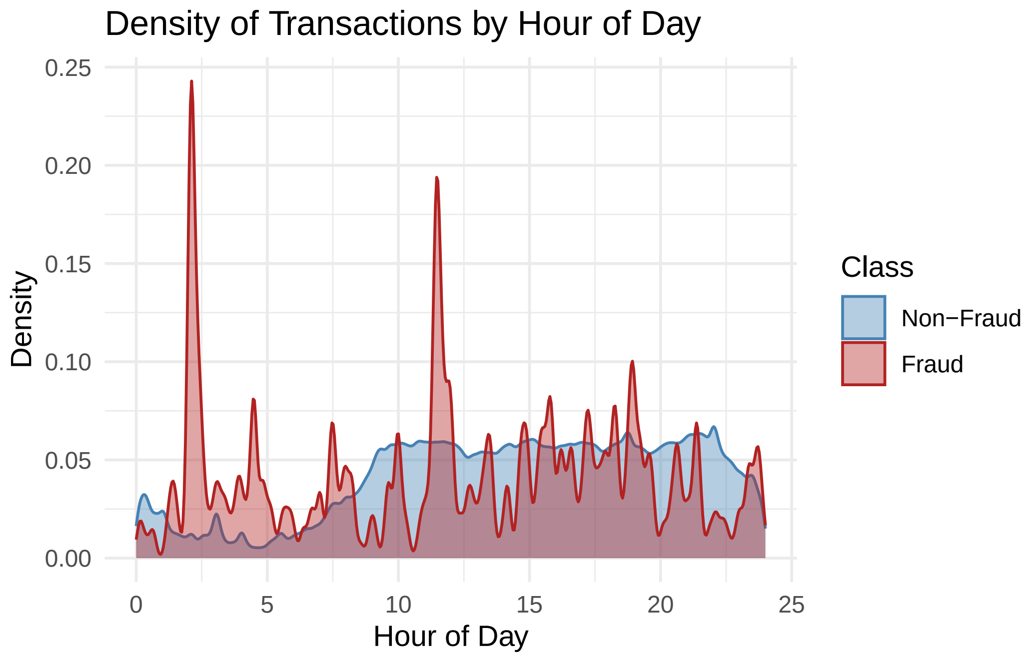



```r
ggplot(df, aes(x = factor(Class), y = Amount)) +
  geom_boxplot() +
  labs(title = "Boxplots of Amount by Class",
       x = "Class",
       y = "Amount") +
  theme_minimal()
```



3. Transaction Time by Class

```
ggplot(df, aes(x = Hour, fill = factor(Class), color = factor(Class))) +
  geom_density(alpha = 0.4, adjust = 1.2, bw = 0.1) +
  scale_fill_manual(values = c("steelblue", "firebrick"), labels = c("Non-Fraud", "Fraud")) +
  scale_color_manual(values = c("steelblue", "firebrick"), labels = c("Non-Fraud", "Fraud")) +
  labs(
    title = "Density of Transactions by Hour of Day",
    x = "Hour of Day",
    y = "Density",
    fill = "Class",
    color = "Class"
  ) +
  theme_minimal()
```



This plot shows when during the day fraudulent vs. non-fraudulent transactions are most likely to occur. Although the dataset spans two days, we compress both days into a 24-hour cycle to capture daily patterns.

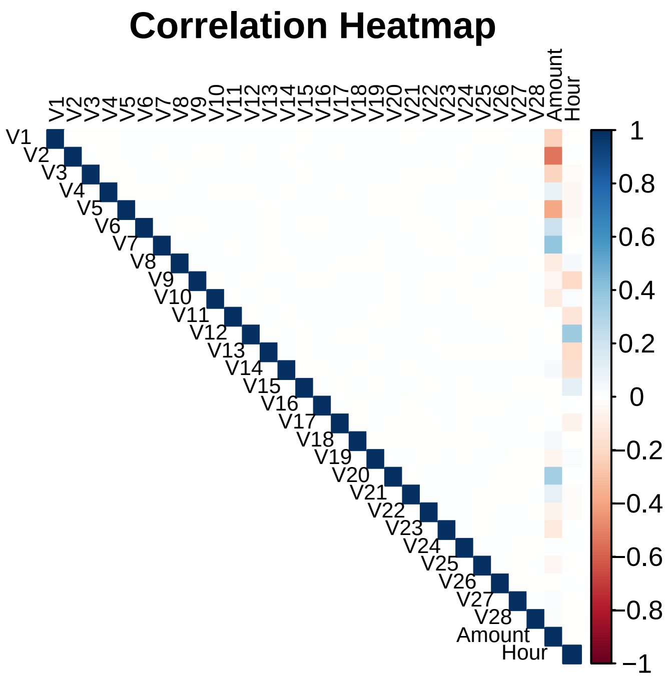
Non-fraudulent transactions are fairly evenly distributed throughout the day, with a peak during business hours.

Fraudulent transactions, however, appear slightly more concentrated in the early morning (around 1–6 AM), when regular activity is lower.

This could suggest that fraud attempts are more likely to occur when users or bank systems are less active, possibly to avoid detection.

4. Correlation Matrix of Features

```
# Compute correlation matrix
cor_matrix <- cor(df[, -which(names(df) == "Class")])  # Exclude 'Class'

# Base R heatmap using corrplot
corrplot(cor_matrix, method = "color", type = "upper",
         tl.cex = 0.7, tl.col = "black", title = "Correlation Heatmap", mar = c(0,0,1,0))
```



5. T-test on Amount for Fraud vs. Non-Fraud

```
t_test_result <- t.test(Amount ~ Class, data = df)
print(t_test_result)
```

```

	Welch Two Sample t-test

data:  Amount by Class
t = -2.9288, df = 492.61, p-value = 0.003561
alternative hypothesis: true difference in means between group 0 and group 1 is not equal
95 percent confidence interval:
 -56.67588 -11.16472
sample estimates:
mean in group 0 mean in group 1
       88.29102       122.21132
```

0.3 Resampling

```
# Check class imbalance
table(df$Class)
```

```

     0      1
284315    492
```

```
prop.table(table(df$Class))  # Percent fraud vs. normal
```

```
          0           1 
0.998272514 0.001727486 
```

We'll keep all the fraud cases (Class = 1), generate synthetic ones, and reduce the number of non-fraud cases (Class = 0) to create a balanced training set.

```r
df$Class <- as.factor(df$Class)

# Apply SMOTE with Undersampling
set.seed(1)

df_smote_under <- SMOTE(Class ~ ., data = df, perc.over = 600, perc.under = 100)

table(df_smote_under$Class)
```

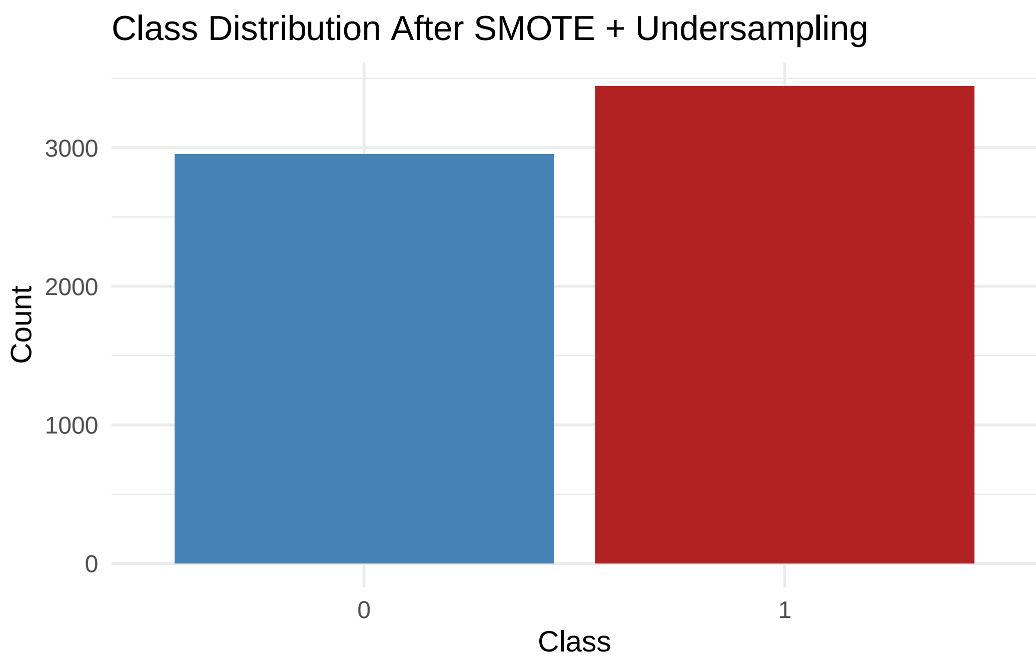
```
   0    1 
2952 3444 
```

perc.over = 600 → SMOTE created 6 synthetic cases per real fraud → 492 × 6 = 2952 synthetic frauds

Total frauds after SMOTE = 492 original + 2952 synthetic = 3444

perc.under = 100 → You keep 1 non-fraud for each fraud → So, 2952 non-frauds were selected from the original 284,315

```r
ggplot(df_smote_under, aes(x = Class)) +
  geom_bar(fill = c("steelblue", "firebrick")) +
  labs(title = "Class Distribution After SMOTE + Undersampling", x = "Class", y = "Count") +
  theme_minimal()
```



After applying SMOTE with 600% oversampling and 1:1 undersampling, we generated 3444 fraud cases (492 real + 2952 synthetic) and kept 2952 non-fraud cases. This gives us a nearly balanced dataset (54% fraud vs. 46% non-fraud) suitable for training without being overwhelmed by majority class bias.

0.4 Basic Model Fitting with Original Data Set

1. Scale the Features

```
features <- df[, setdiff(names(df), "Class")]

# Scale features
scaled_features <- as.data.frame(scale(features))

# Combine with target column
df_scaled <- cbind(scaled_features, Class = df$Class)
```

2. Create Train/Test Split

```
set.seed(123)
df_scaled$Class <- as.factor(df_scaled$Class)
train_index <- createDataPartition(df_scaled$Class, p = 0.7, list = FALSE)

train_data <- df_scaled[train_index, ]
test_data  <- df_scaled[-train_index, ]
```

3. Fit a Logistic Regression Model using whole data

```r
# Fit initial logistic model on all predictors
initial_model <- glm(Class ~ ., data = train_data, family = binomial)

summary(initial_model)
```

```

Call:
glm(formula = Class ~ ., family = binomial, data = train_data)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)    
(Intercept) -8.75739    0.18354 -47.715  < 2e-16 ***
V1           0.20826    0.10623   1.960 0.049949 *  
V2           0.02583    0.11885   0.217 0.827943    
V3           0.08172    0.08769   0.932 0.351361    
V4           1.03880    0.13335   7.790 6.71e-15 ***
V5           0.16737    0.11071   1.512 0.130566    
V6          -0.10918    0.12198  -0.895 0.370747    
V7          -0.15976    0.10538  -1.516 0.129498    
V8          -0.21197    0.04763  -4.451 8.57e-06 ***
V9          -0.23356    0.15428  -1.514 0.130069    
V10         -0.98031    0.13784  -7.112 1.14e-12 ***
V11          0.12477    0.09806   1.272 0.203212    
V12         -0.05018    0.11509  -0.436 0.662815    
V13         -0.18810    0.10259  -1.833 0.066738 .  
V14         -0.46986    0.07399  -6.350 2.15e-10 ***
V15         -0.14419    0.09721  -1.483 0.138017    
V16         -0.20490    0.13518  -1.516 0.129565    
V17         -0.01208    0.07458  -0.162 0.871283    
V18         -0.01229    0.13267  -0.093 0.926180    
V19          0.02439    0.09881   0.247 0.805006    
V20         -0.36865    0.08875  -4.154 3.27e-05 ***
V21          0.20600    0.05594   3.683 0.000231 ***
V22          0.29387    0.11593   2.535 0.011250 *  
V23         -0.05825    0.04267  -1.365 0.172191    
V24          0.09914    0.11312   0.876 0.380804    
V25         -0.05208    0.08487  -0.614 0.539512    
V26         -0.15603    0.12010  -1.299 0.193882    
V27         -0.30696    0.07084  -4.333 1.47e-05 ***
V28         -0.08484    0.03757  -2.258 0.023941 *  
Amount       0.21982    0.11568   1.900 0.057401 .  
Hour         0.11945    0.11962   0.999 0.317999    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 5077.4  on 199365  degrees of freedom
```

```
Residual deviance: 1403.3  on 199335  degrees of freedom
AIC: 1465.3

Number of Fisher Scoring iterations: 12
```

4. Check Variance Inflation Factor for Multicollinearity

```
vif(initial_model)
```

```
       V1        V2        V3        V4        V5        V6        V7        V8 
 7.280154 13.952606  5.258596  4.322648  8.255510  3.841172  9.450485  2.396360 
       V9       V10       V11       V12       V13       V14       V15       V16 
 5.380269  8.718316  2.552058  6.447101  1.240800  4.992102  1.336216  9.008872 
      V17       V18       V19       V20       V21       V22       V23       V24 
 8.834502  6.644131  2.399256  9.621105  2.577871  3.231270  2.610319  1.464988 
      V25       V26       V27       V28    Amount      Hour 
 1.995120  1.431975  6.821556  1.962003 20.757140  1.527513 
```

Rule of thumb –> Vif >10 is a sign of multicollinearity –> we have many values >10

```
log_prob <- predict(initial_model, newdata = test_data, type = "response")
```

0.5 Model with Regularization (LASSO)

```
h2o.init(nthreads = -1)
```

```

 Connection successful!

R is connected to the H2O cluster: 
    H2O cluster uptime:         7 minutes 304 milliseconds 
    H2O cluster timezone:       Europe/Vienna 
    H2O data parsing timezone:  UTC 
    H2O cluster version:        3.44.0.3 
    H2O cluster version age:    1 year, 6 months and 4 days 
    H2O cluster name:           H2O_started_from_R_sarp_afb950 
    H2O cluster total nodes:    1 
    H2O cluster total memory:   7.69 GB 
    H2O cluster total cores:    20 
    H2O cluster allowed cores:  20 
    H2O cluster healthy:        TRUE 
    H2O Connection ip:          localhost 
    H2O Connection port:        54321 
    H2O Connection proxy:       NA 
    H2O Internal Security:      FALSE 
    R Version:                  R version 4.4.1 (2024-06-14 ucrt) 
```

```
Warning in h2o.clusterInfo():
Your H2O cluster version is (1 year, 6 months and 4 days) old. There may be a newer version ava
Please download and install the latest version from: https://h2o-release.s3.amazonaws.com/h2o/l
```

```r
# Convert to H2O frame
train_h2o <- as.h2o(train_data)
```

```
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```r
test_h2o <- as.h2o(test_data)
```

```
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

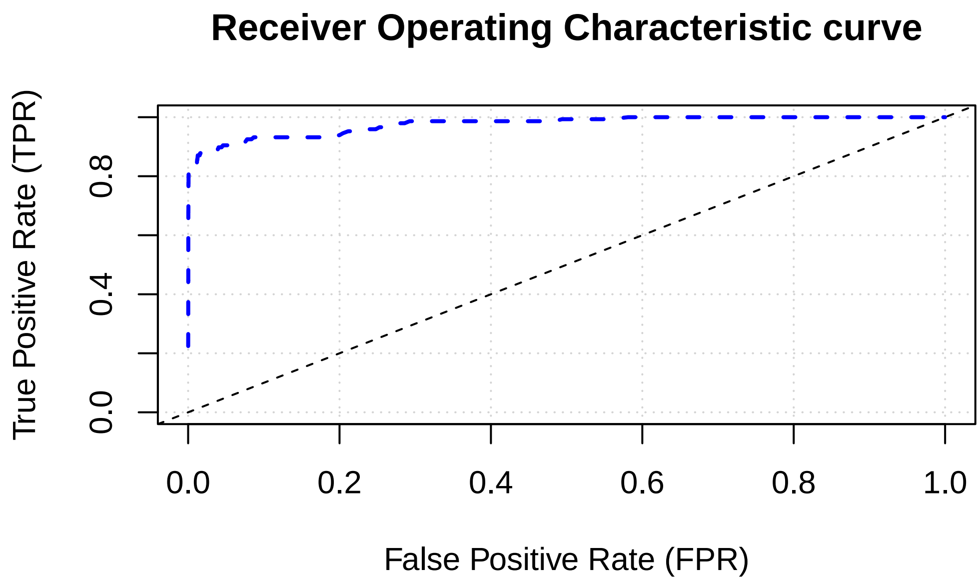
```r
# Train lasso (lambda search enabled)
model <- h2o.glm(x = setdiff(names(train_data), "Class"),
                 y = "Class",
                 training_frame = train_h2o,
                 family = "binomial",
                 alpha = 1,
                 lambda_search = TRUE)
```

```
  |
  |                                                                      |   0%
  |
  |======================================================                |  77%
  |
  |======================================================================| 100%
```

```r
# Predict
lasso_pred <- h2o.predict(model, test_h2o)
```

```
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```r
lasso_perf <- h2o.performance(model, newdata = test_h2o)

# Plot ROC curve
plot(lasso_perf, type = "roc")
```



0.6 Autoencoders for Anomaly Detection

1. Apply autoencoder model

```r
autoencoder <- h2o.deeplearning(
  x = names(scaled_features),
  training_frame = train_h2o,
  autoencoder = TRUE,
  hidden = c(10, 2, 10),  # symmetrical bottleneck
  epochs = 50,
  activation = "Tanh",
  seed = 123
)
```

```
  |
  |                                                                      |   0%
  |
  |====                                                                  |   6%
  |
  |===========                                                           |  16%
  |
```

```
|===================                                                      |  27%
|
|==========================                                               |  37%
|
|==================================                                       |  48%
|
|=========================================                                |  59%
|
|================================================                         |  69%
|
|========================================================                 |  80%
|
|===============================================================          |  90%
|
|=========================================================================| 100%
```

The small hidden layer (2 in the center) forces the model to compress information — anomalies will reconstruct poorly.

3. Get Reconstruction Error (Anomaly Score)

```r
recon_error <- h2o.anomaly(autoencoder, train_h2o, per_feature = FALSE)
recon_error_df <- as.data.frame(recon_error)
colnames(recon_error_df) <- "MSE"
```

4. Add Class Labels Back for Evaluation

```r
recon_error_df$Class <- train_data$Class
```

5. Confusion Matrix

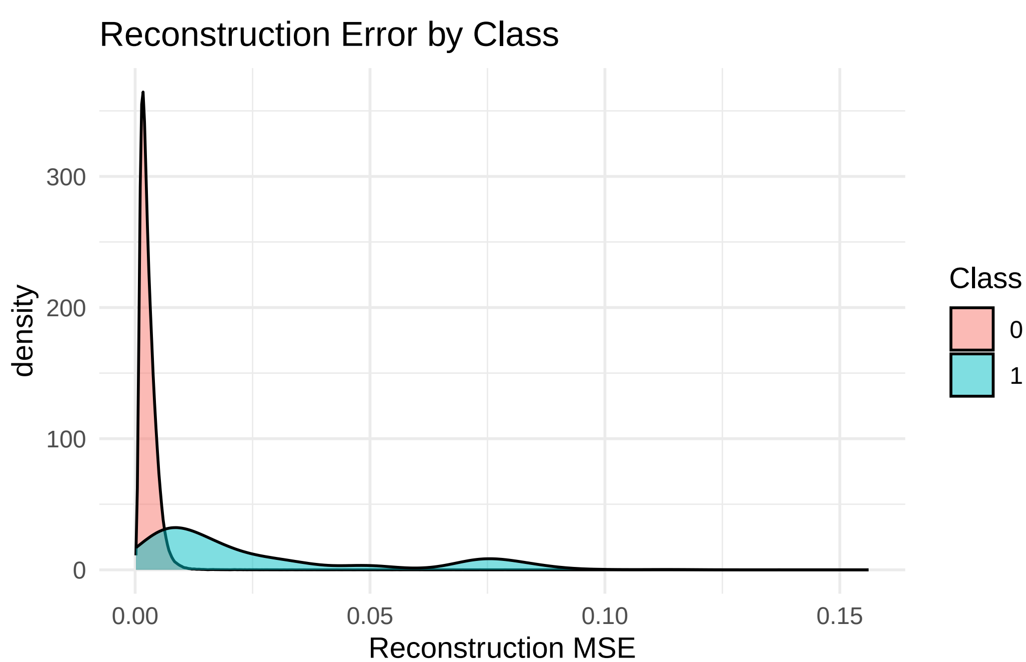
```r
threshold <- quantile(recon_error_df$MSE, 0.98)

# Predict anomalies
recon_error_df$pred <- ifelse(recon_error_df$MSE > threshold, 1, 0)

confusionMatrix(factor(recon_error_df$pred), factor(recon_error_df$Class), positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 195286     92
         1   3735    253

               Accuracy : 0.9808
                 95% CI : (0.9802, 0.9814)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : 1
```

```
               Kappa : 0.114

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.733333
            Specificity : 0.981233
         Pos Pred Value : 0.063440
         Neg Pred Value : 0.999529
             Prevalence : 0.001730
         Detection Rate : 0.001269
   Detection Prevalence : 0.020003
      Balanced Accuracy : 0.857283

       'Positive' Class : 1
```

6. Plot the reconstruction error

```r
ggplot(recon_error_df, aes(x = MSE, fill = factor(Class))) +
  geom_density(alpha = 0.5) +
  labs(title = "Reconstruction Error by Class", x = "Reconstruction MSE", fill = "Class")
  theme_minimal()
```



0.7 XGBoost Model Original Data

We proceeded with applying the XGBoost- Extreme Gradient Boosting algorithm to the data. Given the highly imbalanced credit card fraud dataset, we aim to inspect the algorithms ability to capture complex interactions and decision boundaries.

3. Convert data to DMatrix

```
y_train_bin <- as.numeric(train_data$Class) - 1  # 1 for "pos", 0 for "neg"
y_test_bin  <- as.numeric(test_data$Class) - 1

dtrain <- xgb.DMatrix(
  data  = as.matrix(train_data[ , setdiff(names(train_data), "Class")]),
  label = y_train_bin
)
dtest <- xgb.DMatrix(
  data  = as.matrix(test_data[  , setdiff(names(test_data),  "Class")]),
  label = y_test_bin
)
```

Both test and training data was converted into a special format xgb.DMatrix. Reasoning:

- optimized for speed and memory efficiency
- supports the weighting of individual rows, which can be used to manage class imbalance

2. Define Hyperparamters

```
params <- list(
  objective = "binary:logistic",
  eval_metric = "auc",
  eta = 0.1,
  max_depth = 6
)
```

The baseline hyperparameters of the XGBoost model are the following

- learning rate of 0.1
- max tree depth of 6 - moderate depth, enough to detect important interactions without overly complexity
- early stopping rounds of 10 - stop training if performance on the validation set doesn't improve
- auc evaluation metric - wellsuited for imbalanced classification problems

3. Train the Model using df_scaled training data

```
xgb_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = 100,
  watchlist = list(train = dtrain, test = dtest),
  early_stopping_rounds = 10,
  verbose = 0
)

# Predict probabilities and labels
```

```r
xgb_pred_prob <- predict(xgb_model, dtest)
xgb_pred_label <- ifelse(xgb_pred_prob > 0.5, 1, 0)
```

The trained XGBoost model generates probabilites on the test set. Hard class labels are applied to the predicated probability. Within the thresholds:

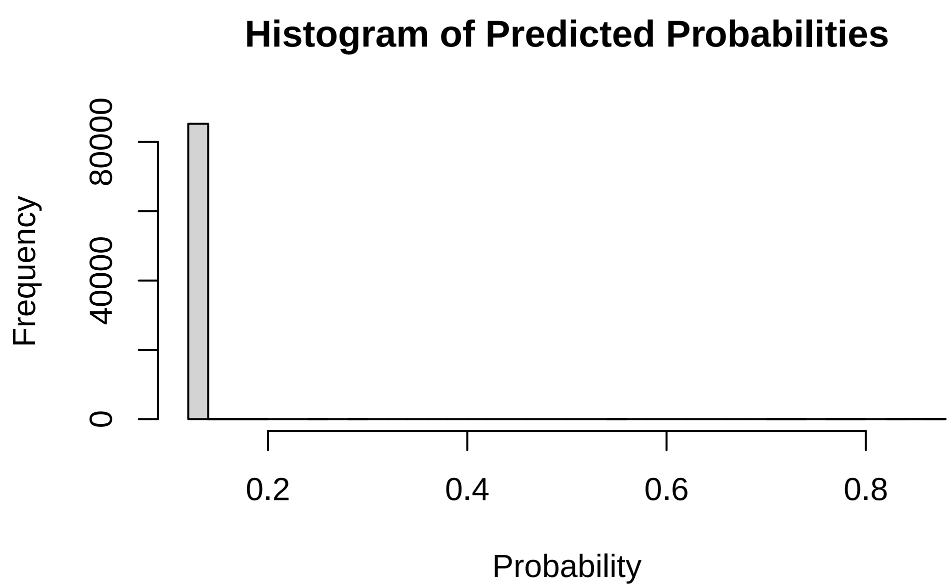
- > .50 it is labeled as fraud (1)
- <= .50 it is labeled as non-fraud (0)

4. Inspect Prediction Probability Distribution

```r
summary(xgb_pred_prob)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1326  0.1326  0.1326  0.1337  0.1326  0.8627
```

```r
hist(xgb_pred_prob, breaks = 50, main = "Histogram of Predicted Probabilities", xlab = "Probabi
```



5. Confusion Matrix & AUC Curve

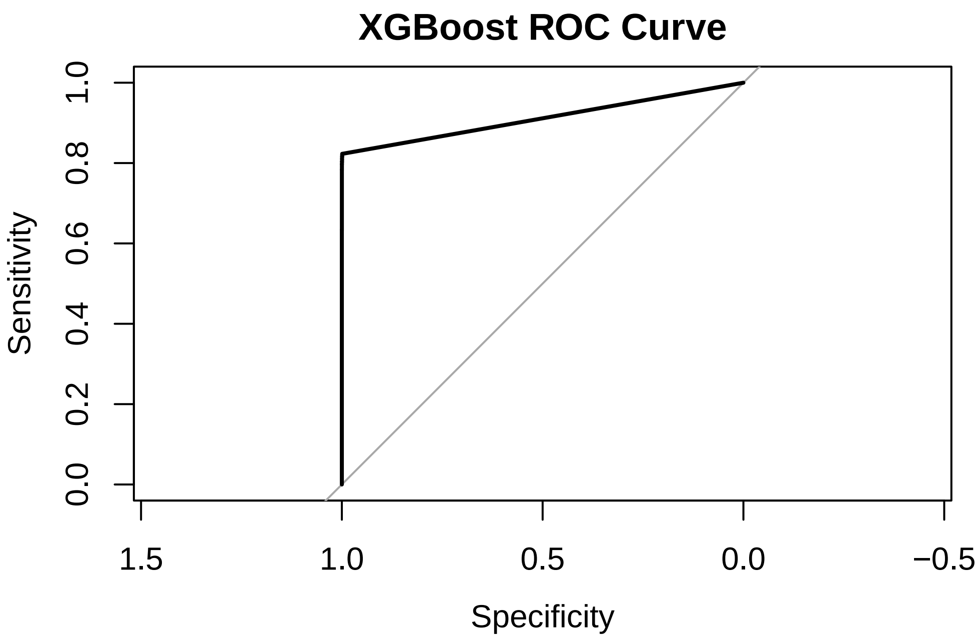
```r
conf_matrix <- confusionMatrix(
  factor(xgb_pred_label),
  factor(getinfo(dtest, "label")),
  positive = "1"
)
print(conf_matrix)
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 85285    32
         1     9   115

               Accuracy : 0.9995
                 95% CI : (0.9993, 0.9997)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8485

 Mcnemar's Test P-Value : 0.0005908

            Sensitivity : 0.782313
            Specificity : 0.999894
         Pos Pred Value : 0.927419
         Neg Pred Value : 0.999625
             Prevalence : 0.001720
         Detection Rate : 0.001346
   Detection Prevalence : 0.001451
      Balanced Accuracy : 0.891104

       'Positive' Class : 1
```

```r
# ROC and AUC
roc_obj <- roc(getinfo(dtest, "label"), xgb_pred_prob)
```

```
Setting levels: control = 0, case = 1

Setting direction: controls < cases
```

```r
plot(roc_obj, main = "XGBoost ROC Curve")
```




```
cat("AUC:", auc(roc_obj), "\n")
```

```
AUC: 0.9113808
```

When using the original imbalanced dataset, the ROC curve shows a less steep shape, and the model is less confident in distinguishing fraud due to the extreme class imbalance. This limits sensitivity and causes the ROC curve to underperform. We try to address this with a Resampling method ROSE.

0.8 XGBoost with Resampling

Resample the Dataset by oversampling the minority

2. Apply ROSE to the scaled data to balance it

```
set.seed(123)
train_index <- createDataPartition(df_scaled$Class, p = 0.7, list = FALSE)
train_raw <- df_scaled[train_index, ]
test_data_xg <- df_scaled[-train_index, ]

# Now apply ROSE only to the training set

train_data_xg <- ROSE(Class ~ ., data = train_raw, seed = 1, N = nrow(train_raw), p = 0.2)$data
```

4. Convert to DMatrix

```r
dtrain_xg <- xgb.DMatrix(
  data = as.matrix(train_data_xg[, -ncol(train_data_xg)]),
  label = as.numeric(as.character(train_data_xg$Class))
)

dtest_xg <- xgb.DMatrix(
  data = as.matrix(test_data_xg[, -ncol(test_data_xg)]),
  label = as.numeric(as.character(test_data_xg$Class))
)
```

5. Define Hyperparameters & create model

```r
params <- list(
  objective = "binary:logistic",
  eval_metric = "auc",
  eta = 0.1,
  max_depth = 6
)

xgb_model_rose <- xgb.train(
  params = params,
  data = dtrain_xg,
  nrounds = 100,
  watchlist = list(train = dtrain_xg, test = dtest_xg),
  early_stopping_rounds = 10,
  verbose = 0
)
```

6. Predict Probabilities & Define Threshold

```r
xgb_rose_pred_prob <- predict(xgb_model, dtest_xg)
xgb_rose_pred_label <- ifelse(xgb_rose_pred_prob > 0.8, 1, 0)
```

7. Confusion Matrix

```r
conf_matrix <- confusionMatrix(
  factor(xgb_rose_pred_label),
  factor(getinfo(dtest_xg, "label")),
  positive = "1"
)
print(conf_matrix)
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 85293    72
         1     1    75
```

```
               Accuracy : 0.9991
                 95% CI : (0.9989, 0.9993)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : 1.002e-11

                  Kappa : 0.6723

 Mcnemar's Test P-Value : 2.550e-16

            Sensitivity : 0.5102041
            Specificity : 0.9999883
         Pos Pred Value : 0.9868421
         Neg Pred Value : 0.9991566
             Prevalence : 0.0017205
         Detection Rate : 0.0008778
   Detection Prevalence : 0.0008895
      Balanced Accuracy : 0.7550962

       'Positive' Class : 1
```

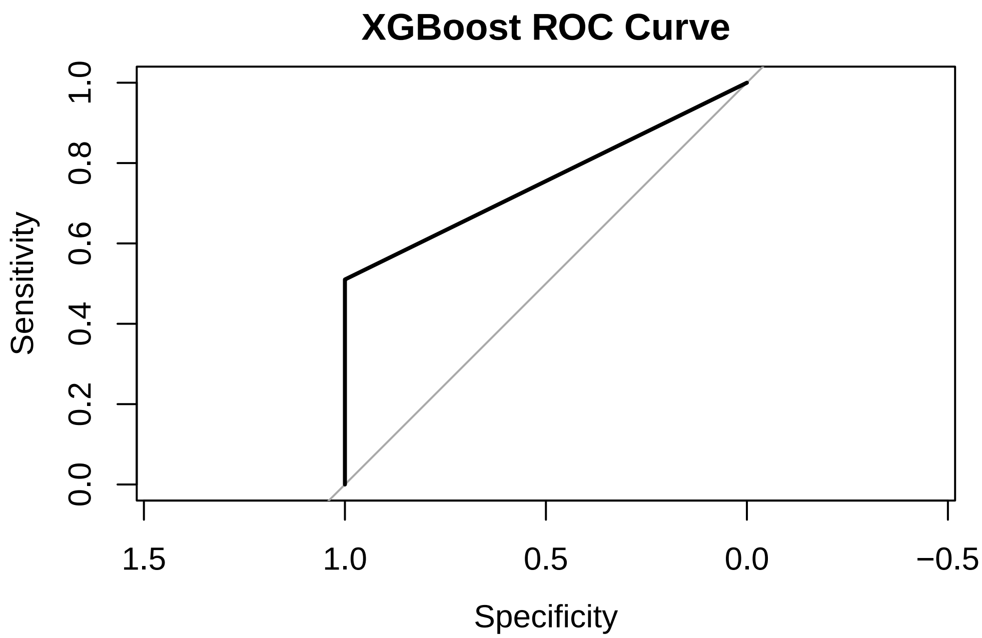
8. AUC, ROC Curve

```r
# ROC and AUC
roc_obj <- roc(getinfo(dtest_xg, "label"), xgb_rose_pred_label)
```

```
Setting levels: control = 0, case = 1
```

```
Setting direction: controls < cases
```

```r
plot(roc_obj, main = "XGBoost ROC Curve")
```




```
cat("AUC:", auc(roc_obj), "\n")
```

```
AUC: 0.7550962
```

After applying ROSE resampling with a 20% fraud rate (`p = 0.2`), the ROC curve becomes significantly sharper and more optimistic. This happens because:

- The model now sees more fraud cases during training
- It learns a clearer decision boundary between classes
- However, this also introduces some synthetic data artifacts, and the results can overestimate real-world performance

0.9 Support Vector Machines

```
#--- Caret trainControl ---
ctrl <- trainControl(
  method          = "cv",
  number          = 5,
  summaryFunction = twoClassSummary,
  classProbs      = TRUE,
  verboseIter     = TRUE
)

# rename levels for twoClassSummary ("pos" must be the second level)
levels(train_data$Class) <- c("neg","pos")
levels(test_data$Class)  <- c("neg","pos")
```

```r
# SVM Grid Search
svm_grid <- expand.grid(
  sigma = c(0.001, 0.01),
  C     = c(0.1, 1, 10)
)

set.seed(123)
svm_tuned <- train(
  Class ~ .,
  data      = train_data,
  method    = "svmRadial",
  metric    = "ROC",
  trControl= ctrl,
  tuneGrid = svm_grid
)

saveRDS(rf_tuned, "rf_tuned.RDS")

svm_tuned = readRDS("svm_tuned.RDS")


print(svm_tuned)
```

```
Support Vector Machines with Radial Basis Function Kernel

199366 samples
    30 predictor
     2 classes: 'neg', 'pos'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 159493, 159493, 159492, 159493, 159493
Resampling results across tuning parameters:

  sigma  C     ROC        Sens       Spec
  0.001   0.1  0.9407281  0.9998241  0.7333333
  0.001   1.0  0.9588456  0.9998342  0.7623188
  0.001  10.0  0.9548649  0.9998643  0.7710145
  0.010   0.1  0.9597058  0.9998342  0.5971014
  0.010   1.0  0.9594919  0.9998995  0.7478261
  0.010  10.0  0.9447723  0.9998995  0.7797101

ROC was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.01 and C = 0.1.
```

```r
# best parameters:
svm_tuned$bestTune
```

```
  sigma   C
4  0.01 0.1
```

0.10 Random Forest

```r
#Random Forest Grid Search
rf_grid <- expand.grid(
  mtry = c(2, 4, 6, 8)
)

set.seed(123)
rf_tuned <- train(
  Class ~ .,
  data      = train_data,
  method    = "rf",
  metric    = "ROC",
  trControl = ctrl,
  tuneGrid  = rf_grid,
  ntree     = 100
)

saveRDS(rf_tuned, "rf_tuned.RDS")

rf_tuned = readRDS("rf_tuned.RDS")

print(rf_tuned)
```

```
Random Forest

199366 samples
    30 predictor
     2 classes: 'neg', 'pos'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 159493, 159493, 159492, 159493, 159493
Resampling results across tuning parameters:

  mtry  ROC        Sens       Spec
  2     0.9534074  0.9998995  0.7275362
  4     0.9492509  0.9998945  0.7855072
  6     0.9523820  0.9998995  0.8000000
  8     0.9537972  0.9998895  0.8000000

ROC was used to select the optimal model using the largest value.
The final value used for the model was mtry = 8.
```

```r
rf_tuned$bestTune
```

```
  mtry
4    8
```

0.11 Evaluating all models on Confusion Matrix

```r
# find threshold maximizing F1
find_best_thr <- function(probs, truth, pos_label="pos") {
  # truth: factor with levels c("neg","pos")
  thresholds <- seq(0, 1, by = 0.01)
  f1_scores  <- sapply(thresholds, function(th) {
    preds <- factor(ifelse(probs > th, pos_label,
                           setdiff(levels(truth), pos_label)),
                    levels = levels(truth))
    cm    <- confusionMatrix(preds, truth, positive = pos_label)
    as.numeric(cm$byClass["F1"])
  })
  best_idx <- which.max(f1_scores)
  list(threshold = thresholds[best_idx], f1 = f1_scores[best_idx])
}

#Re-define eval to take an arbitrary threshold
evaluate_at_thr <- function(probs, truth, thr, model_name) {
  preds <- factor(ifelse(probs > thr, "pos", "neg"), levels = c("neg","pos"))
  cm    <- confusionMatrix(preds, truth, positive = "pos")
  auc   <- roc(response = as.numeric(truth=="pos"), predictor = probs)$auc
  cat("\n=== ", model_name, " (thr=", round(thr,2), ") ===\n", sep = "")
  print(cm)
  cat("AUC:", round(auc,4), "\n",
      "F1:", round(cm$byClass["F1"], 4), "\n")
}



# 3) Apply to each model

## (a) Logistic regression
log_prob <- predict(initial_model, newdata = test_data, type = "response")
# truth factor
truth <- factor(ifelse(test_data$Class == 1, "pos", "neg"))
best <- find_best_thr(log_prob, truth)
evaluate_at_thr(log_prob, truth, best$threshold, "Logistic Regression")
```

```
=== Logistic Regression (thr=0.2) ===
```

```
Confusion Matrix and Statistics

          Reference
Prediction   neg   pos
       neg 85262    36
       pos    32   111

               Accuracy : 0.9992
                 95% CI : (0.999, 0.9994)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : 2.46e-13

                  Kappa : 0.7651

 Mcnemar's Test P-Value : 0.716

            Sensitivity : 0.755102
            Specificity : 0.999625
         Pos Pred Value : 0.776224
         Neg Pred Value : 0.999578
             Prevalence : 0.001720
         Detection Rate : 0.001299
   Detection Prevalence : 0.001674
      Balanced Accuracy : 0.877363

       'Positive' Class : pos

AUC: 0.9761
 F1: 0.7655
```

```r
## (b) LASSO (H2O)
lasso_h2o  <- h2o.predict(model, test_h2o)
```

```
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```r
lasso_prob <- as.vector(lasso_h2o$p1)
best <- find_best_thr(lasso_prob, truth)
evaluate_at_thr(lasso_prob, truth, best$threshold, "LASSO (H2O)")
```

```
=== LASSO (H2O) (thr=0.18) ===
Confusion Matrix and Statistics
```

```
          Reference
Prediction   neg   pos
       neg 85261    35
       pos    33   112

               Accuracy : 0.9992
                 95% CI : (0.999, 0.9994)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : 2.46e-13

                  Kappa : 0.7667

 Mcnemar's Test P-Value : 0.9035

            Sensitivity : 0.761905
            Specificity : 0.999613
         Pos Pred Value : 0.772414
         Neg Pred Value : 0.999590
             Prevalence : 0.001720
         Detection Rate : 0.001311
   Detection Prevalence : 0.001697
      Balanced Accuracy : 0.880759

       'Positive' Class : pos

AUC: 0.9763
 F1: 0.7671
```

```r
## (c) Autoencoder anomaly (use MSE as "prob")

recon_error <- h2o.anomaly(autoencoder, test_h2o, per_feature = FALSE)
recon_error_df <- as.data.frame(recon_error)
colnames(recon_error_df) <- "MSE"


ae_prob <- recon_error_df$MSE

# map levels to neg/pos
best <- find_best_thr(ae_prob, truth)
evaluate_at_thr(ae_prob, truth, best$threshold, "Autoencoder")
```

```
=== Autoencoder (thr=0.02) ===
Confusion Matrix and Statistics

          Reference
Prediction   neg   pos
       neg 85204    87
```

```
       pos    90    60

               Accuracy : 0.9979
                 95% CI : (0.9976, 0.9982)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : 0.9929

                  Kappa : 0.403

 Mcnemar's Test P-Value : 0.8805

            Sensitivity : 0.4081633
            Specificity : 0.9989448
         Pos Pred Value : 0.4000000
         Neg Pred Value : 0.9989800
             Prevalence : 0.0017205
         Detection Rate : 0.0007022
   Detection Prevalence : 0.0017556
      Balanced Accuracy : 0.7035540

       'Positive' Class : pos

AUC: 0.9287
 F1: 0.404
```

```r
## (d) XGBoost final
xgb_prob <- predict(xgb_model, dtest)
best     <- find_best_thr(xgb_prob, truth)
evaluate_at_thr(xgb_prob, truth, best$threshold, "XGBoost")
```

```
=== XGBoost (thr=0.47) ===
Confusion Matrix and Statistics

          Reference
Prediction   neg   pos
       neg 85285    32
       pos     9   115

               Accuracy : 0.9995
                 95% CI : (0.9993, 0.9997)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8485

 Mcnemar's Test P-Value : 0.0005908
```

```
            Sensitivity : 0.782313
            Specificity : 0.999894
         Pos Pred Value : 0.927419
         Neg Pred Value : 0.999625
             Prevalence : 0.001720
         Detection Rate : 0.001346
   Detection Prevalence : 0.001451
      Balanced Accuracy : 0.891104

       'Positive' Class : pos

AUC: 0.9114
 F1: 0.8487
```

```r
## (e) XGBoost + ROSE (if available)

xgbr_prob <- predict(xgb_model_rose, dtest_xg)
truth_rose <- factor(test_data_xg$Class, levels=c("0","1"))
truth_rose <- factor(ifelse(truth_rose=="1","pos","neg"), levels=c("neg","pos"))
best <- find_best_thr(xgbr_prob, truth_rose)
evaluate_at_thr(xgbr_prob, truth_rose, best$threshold, "XGBoost (ROSE)")
```

```
=== XGBoost (ROSE) (thr=0.85) ===
Confusion Matrix and Statistics

          Reference
Prediction   neg   pos
       neg 85259    30
       pos    35   117

               Accuracy : 0.9992
                 95% CI : (0.999, 0.9994)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : 2.24e-14

                  Kappa : 0.7822

 Mcnemar's Test P-Value : 0.6198

            Sensitivity : 0.795918
            Specificity : 0.999590
         Pos Pred Value : 0.769737
         Neg Pred Value : 0.999648
             Prevalence : 0.001720
         Detection Rate : 0.001369
   Detection Prevalence : 0.001779
      Balanced Accuracy : 0.897754
```

```
       'Positive' Class : pos

AUC: 0.9593
 F1: 0.7826
```

```r
## (f) SVM (caret)
svm_prob <- predict(svm_tuned, test_data, type = "prob")[, "pos"]
best     <- find_best_thr(svm_prob, truth)
evaluate_at_thr(svm_prob, truth, best$threshold, "SVM")
```

```
=== SVM (thr=0.01) ===
Confusion Matrix and Statistics

          Reference
Prediction   neg   pos
       neg 85263    35
       pos    31   112

               Accuracy : 0.9992
                 95% CI : (0.999, 0.9994)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : 5.053e-14

                  Kappa : 0.772

 Mcnemar's Test P-Value : 0.7119

            Sensitivity : 0.761905
            Specificity : 0.999637
         Pos Pred Value : 0.783217
         Neg Pred Value : 0.999590
             Prevalence : 0.001720
         Detection Rate : 0.001311
   Detection Prevalence : 0.001674
      Balanced Accuracy : 0.880771

       'Positive' Class : pos

AUC: 0.9582
 F1: 0.7724
```

```r
## (g) Random Forest (caret)
rf_prob <- predict(rf_tuned, test_data, type = "prob")[, "pos"]
best    <- find_best_thr(rf_prob, truth)
evaluate_at_thr(rf_prob, truth, best$threshold, "Random Forest")
```

```
=== Random Forest (thr=0.41) ===
Confusion Matrix and Statistics

          Reference
Prediction   neg   pos
       neg 85283    30
       pos    11   117

               Accuracy : 0.9995
                 95% CI : (0.9993, 0.9997)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8507

 Mcnemar's Test P-Value : 0.004937

            Sensitivity : 0.795918
            Specificity : 0.999871
         Pos Pred Value : 0.914062
         Neg Pred Value : 0.999648
             Prevalence : 0.001720
         Detection Rate : 0.001369
   Detection Prevalence : 0.001498
      Balanced Accuracy : 0.897895

       'Positive' Class : pos

AUC: 0.9305
 F1: 0.8509
```