Credit Crad Fraud Detection

Dataset Link : https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

Dataset Description:

The dataset contains transactions made by credit cards over a two-day period in **September 2013** by European cardholders. It comprises a total of **284,807** transactions, out of which only **492** transactions are labeled as fraudulent. The dataset is highly imbalanced, with the majority of transactions being non-fraudulent.

The dataset includes the following attributes:

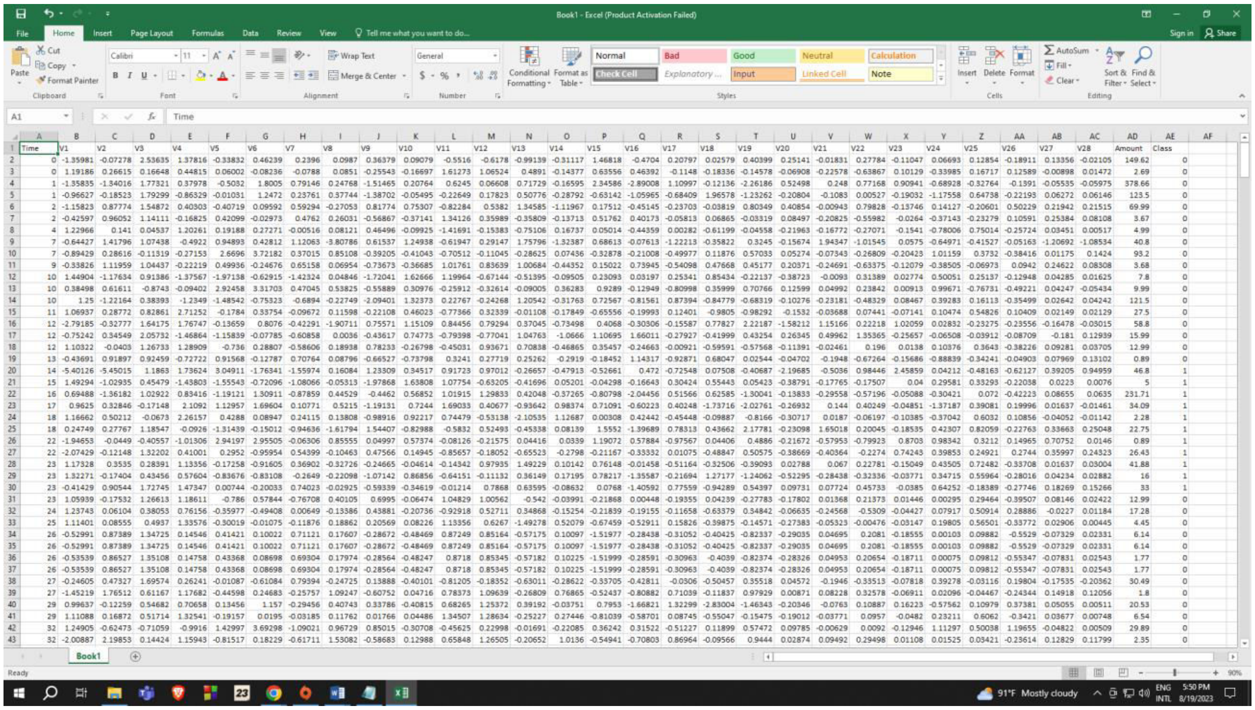
Time: The number of seconds elapsed between this transaction and the first transaction in the dataset.

V1 to V28: Features resulting from a PCA transformation for confidentiality reasons.

Amount: The transaction amount.

Class: The target variable indicating whether the transaction is fraudulent (1) or not (0).

Glimpse of the Dataset ->



Import dataset(.csv file) into Rstudio:

card=read.csv("C:/Users/O M A R/Downloads/Document/Data Science/PomPom/Book1.csv")

print(card)

```
> card=read.csv("C:/Users/O M A R/Downloads/Document/Data Science/PomPom/Book1.csv")
> print(card)
  Time         V1          V2         V3          V4           V5          V6           V7           V8         V9         V10         V11
1    0 -1.3598071 -0.07278117 2.53634674  1.37815522 -0.338320770  0.46238778  0.239598554  0.098697901  0.3637870  0.09079417 -0.55159953
2    0  1.1918571  0.26615071 0.16648011  0.44815408  0.060017649 -0.08236081 -0.078802983  0.085101655 -0.2554251 -0.16697441  1.61272666
3    1 -1.3583541 -1.34016307 1.77320934  0.37977959 -0.503198133  1.80049938  0.791460956  0.247675787 -1.5146543  0.20764287  0.62450146
4    1 -0.9662717 -0.18522601 1.79299334 -0.86329128 -0.010308880  1.24720317  0.237608940  0.377435875 -1.3870241 -0.05495192 -0.22648726
5    2 -1.1582331  0.87773676 1.54871785  0.40303393 -0.407193377  0.09592146  0.592940745 -0.270532677  0.8177393  0.75307443 -0.82284288
```

This code segment loads the credit card fraud dataset using the read.csv function and displays the first few rows of the dataset to understand its structure.

Data Processing:

```
card$Amount <- scale(card$Amount)
card$Time <- scale(card$Time)
set.seed(42)
index <- sample(1:nrow(card), 0.7 * nrow(card))
train_data <- card[index, ]
test_data <- card[-index, ]
```

Overall summary of the dataset:

summary(card)

```
> summary(card)
      Time             V1                 V2                  V3                 V4               V5                 V6                 V7
 Min.   :  0.0   Min.   :-6.0932   Min.   :-12.1142   Min.   :-5.6950   Min.   :-4.516   Min.   :-6.6320   Min.   :-2.1457   Min.   :-2.7054
 1st Qu.: 93.0   1st Qu.:-0.8976   1st Qu.: -0.1687   1st Qu.: 0.3121   1st Qu.: 1.179   1st Qu.:-1.3602   1st Qu.: 0.3757   1st Qu.:-1.1348
 Median :215.0   Median :-0.3655   Median :  0.2858   Median : 0.8922   Median : 1.810   Median :-1.3602   Median : 0.9539   Median :-1.1348
 Mean   :215.4   Mean   :-0.1702   Mean   :  0.2114   Mean   : 0.8729   Mean   : 1.312   Mean   :-0.8997   Mean   : 0.7448   Mean   :-0.6603
 3rd Qu.:326.0   3rd Qu.: 1.1108   3rd Qu.:  0.8766   3rd Qu.: 1.5138   3rd Qu.: 1.810   3rd Qu.:-0.5143   3rd Qu.: 0.9539   3rd Qu.:-0.1074
 Max.   :450.0   Max.   : 1.5861   Max.   :  5.2674   Max.   : 3.7729   Max.   : 4.076   Max.   : 3.2820   Max.   : 5.1221   Max.   : 4.8084
```

Number of Column and Row:

sprintf("Rows: %d Columns: %d",nrow(card), length(names(card)))

```
> sprintf("Rows: %d Columns: %d",nrow(card), length(names(card)))
[1] "Rows: 599 Columns: 31"
```

Count of the Missing value from each attribute:

```
> colSums(is.na(card))
  Time     V1     V2     V3     V4     V5     V6     V7     V8     V9    V10    V11    V12    V13    V14    V15    V16    V17    V18    V19 
     0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0 
   V20    V21    V22    V23    V24    V25    V26    V27    V28 Amount  Class 
     0      0      0      0      0      0      0      0      0      0      0 
```

Conversion of all Non-Numeric columns to Numeric (excluding 'Class'):

```
> numeric_cols <- sapply(card, is.numeric)
> card[, numeric_cols] <- lapply(card[, numeric_cols], as.numeric)
```

Handle missing values (replace NAs with zeros in numeric columns):

```
> numeric_cols <- sapply(card, is.numeric)
> card[, numeric_cols] <- lapply(card[, numeric_cols], function(col) {
+   col[is.na(col)] <- 0
+   return(col)
+ })
```

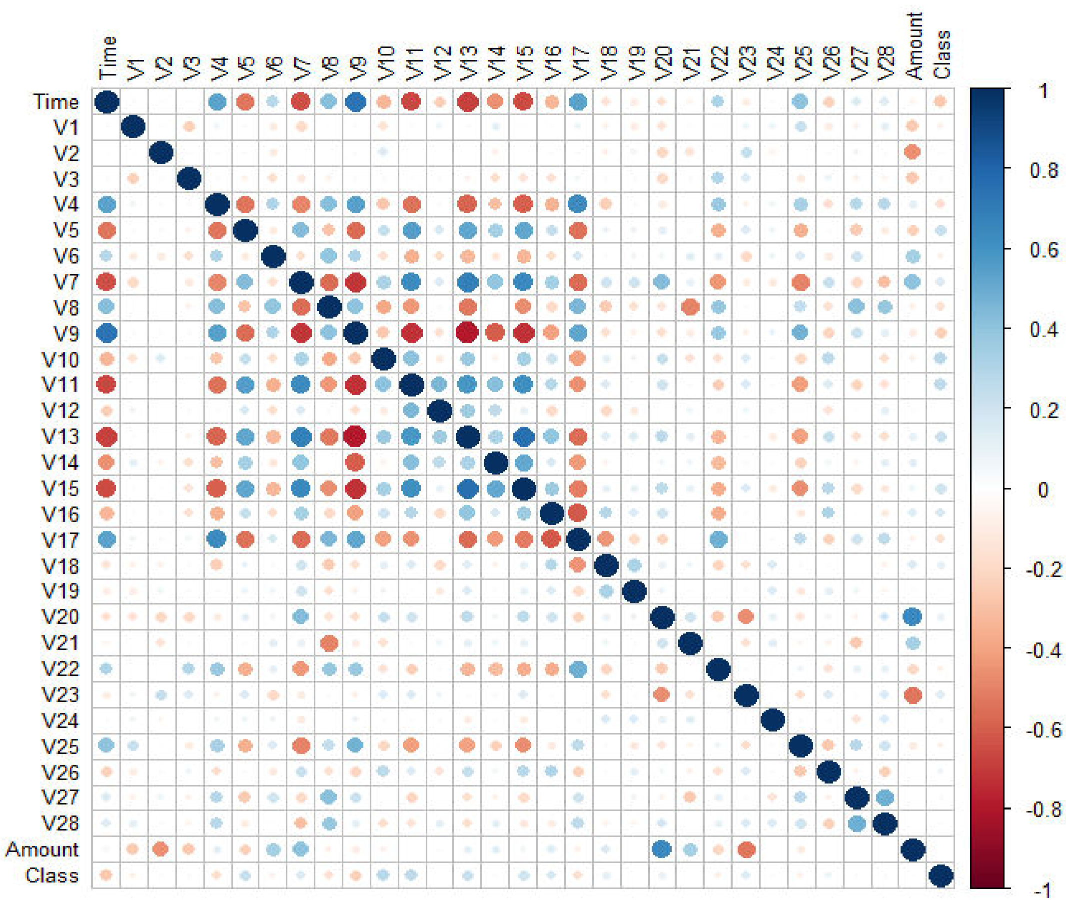
Data Normalize the features (except 'Class' and 'Time'):

```
> cols_to_normalize <- colnames(card)[!(colnames(card) %in% c("Class", "Time"))]
> card[, cols_to_normalize] <- scale(card[, cols_to_normalize])
```

Crosscheck of the Changes:

```
> head(card)
  Time         V1          V2         V3          V4        V5         V6        V7          V8         V9        V10       V11         V12
1    0 -0.8958854 -0.23701777  1.6238181  0.06285398 0.5758058 -0.3286463 1.0942915 -0.49272197 -0.8768982  0.5653416 0.4129761 -2.35926538
2    0  1.0258322  0.04570169 -0.6895607 -0.81905543 0.9844055 -0.9626256 0.7071017 -0.52016512 -1.4842311  0.1255089 2.7186744  2.65307108
3    1 -0.8947911 -1.29420227  0.8788708 -0.88389417 0.4066812  1.2286500 1.7653797 -0.19201968 -2.7193027  0.7647216 1.6658991 -0.32255370
4    1 -0.5995048 -0.33081344  0.8981832 -2.06268400 0.9122674  0.5847230 1.0918721  0.06989272 -2.5941210  0.3166538 0.7593244  0.01142972
5    2 -0.7440753  0.55585517  0.6597302 -0.86184234 0.5051591 -0.7551404 1.5239707 -1.23799017 -0.4316546  1.6953961 0.1240153  1.08346690
6    2 -0.1925879  0.62491121  0.2618374 -1.40358631 1.3546733 -0.9013709 1.3820101 -0.16650959 -1.7914679 -0.2233165 2.4294779  0.55245741
        V13        V14        V15        V16        V17       V18          V19        V20         V21        V22        V23         V24
1 0.1110417  0.4852478 2.48529475  0.1822639 -0.4419577 0.8939729  1.079741298  0.8477036 -0.01856707  0.2613796 -0.1296688 -0.02326148
2 1.7392184  0.8100350 1.52054285  1.8883624 -1.0714823 0.4751484 -0.005730137  0.1141520 -0.77944234 -2.0588876  0.6794533 -1.15111607
3 1.9901809  0.7670133 3.50226802 -4.2361663  1.3172504 0.5993067 -4.183682497  1.4738469  0.95808656  1.5116067  3.7672188 -2.11998514
4 1.7597420  0.5303494 0.05250299 -0.8937225 -2.1817909 4.7787725 -2.151572462 -0.2038883 -0.34861221 -0.4286528 -0.4347551 -3.46832255
5 2.6814443 -1.0834210 0.98703653  0.2168697 -1.3098702 0.7658429  1.868497604  1.2073443  0.01398530  1.5789459 -0.2327727  0.18285633
6 0.8075182  0.8229153 1.38388479  1.7747990 -0.9609526 0.9798048  0.216564794  0.4667445 -0.71518268 -1.8592754  0.1915785 -1.23867752
         V25         V26         V27          V28      Amount Class
1 -0.6917473 -0.19347601  0.25865658 -0.202363541  0.43034713     0
2 -0.5522222  0.80544967 -0.36548981 -0.004615734 -0.36073302     0
3 -2.3393523 -0.03486316 -0.56852880 -0.416259422  1.66351256     0
4  1.1821526 -0.29753243 -0.05151096  0.253689374  0.28971545     0
5 -1.9000490  1.99904374  0.63462865  1.103195005  0.00161432     0
6 -1.9967865  0.74209191  0.78535225  0.362147533 -0.35545664     0
```

Correlations : Pearson Correlation



This code calculates Pearson's correlation coefficients for all attributes and identifies the attributes that are highly correlated with the target 'Class'. It then selects important attributes based on correlation thresholds.

correlations <- cor(card,method="pearson")

corrplot(correlations, number.cex = .9, method = "circle", type = "full", tl.cex=0.8,tl.col = "black")

Coversion of targeted variable into Factor:

card$Class <- factor(card$Class)

card$Class

```
> card$Class <- factor(card$Class)
> card$Class
  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [71] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[141] 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[211] 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[281] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[351] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[421] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[491] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[561] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Levels: 0 1
```

KNN Classification:

In this example, we're using the “class & caret” package, specifically the knn() function, to implement the KNN classifier.

library(class)

library(caret)

card$Class <- as.factor(card$Class)

set.seed(123)

train_indices <- sample(1:nrow(card), 0.7 * nrow(card))

train_data <- card[train_indices,]

test_data <- card[-train_indices,]

knn_model <- knn(train_data[, -ncol(train_data)], test_data[, -ncol(test_data)], train_data$Class, k = 5)

head(knn_model,100)


```
> card$Class <- factor(card$Class)
> card$Class
  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [71] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[141] 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[211] 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[281] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[351] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[421] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[491] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[561] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Levels: 0 1
```


Confusion Matrix and Accuracy calculated:

```
confusion_matrix <- confusionMatrix(knn_model, test_data$Class)

print(confusion_matrix)
```

```
> confusion_matrix <- confusionMatrix(knn_model, test_data$Class)
> print(confusion_matrix)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 167   5
         1   2   6

               Accuracy : 0.9611
                 95% CI : (0.9215, 0.9842)
    No Information Rate : 0.9389
    P-Value [Acc > NIR] : 0.1351

                  Kappa : 0.6116

 Mcnemar's Test P-Value : 0.4497

            Sensitivity : 0.9882
            Specificity : 0.5455
         Pos Pred Value : 0.9709
         Neg Pred Value : 0.7500
             Prevalence : 0.9389
         Detection Rate : 0.9278
   Detection Prevalence : 0.9556
      Balanced Accuracy : 0.7668

       'Positive' Class : 0
```

10-fold cross validation:

To perform 10-fold cross-validation on dataset using the K-Nearest Neighbors (KNN) classifier, I can use the following code:

```
card$Class <- as.factor(card$Class)

num_folds <- 10

set.seed(123)

cv <- createFolds(card$Class, k = num_folds)

ctrl <- trainControl(method = "cv", number = num_folds)

knn_model <- train(Class ~ ., data = card, method = "knn",

trControl = ctrl, tuneLength = 1,

preProcess = c("center", "scale"),

metric = "Accuracy")

cat("Mean Accuracy:", knn_model$results$Accuracy, "\n")
```

This code performs 10-fold cross-validation using the KNN classifier. It creates a cross-validation object using the createFolds() function, iterates through each fold, fits the KNN model, calculates accuracy for each fold, and finally calculates the mean accuracy across all folds.

```
> cat("Mean Accuracy:", knn_model$results$Accuracy, "\n")
Mean Accuracy: 0.9332685
```

Modeling Evaluation:

Applying machine learning techniques, such as K-nearest neighbors (KNN), helps classify transactions as fraudulent or non-fraudulent. To assess model performance, we utilized accuracy, recall, and precision metrics. The challenge lies in optimizing the trade-off between recall and precision due to the imbalanced dataset.

Discussion:

The Credit Card Fraud Detection dataset provides a rich context for exploring the challenges and intricacies of detecting fraudulent transactions in financial systems. The dataset's complexity stems from its imbalanced nature, anonymized features, and the critical implications of accurate fraud detection. Let's delve into the key discussion points surrounding this dataset:

- **Key Insights:** Imbalanced nature,Feature anonymization,Importance of ‘Time’ and ‘Amount’
- **Data Preparation:** Conversion to Numeric,Handeling missing value,Feature normalization
- **Modeling and Evaluation:** K-nearest neighbors (KNN),10-fold cross validation,Confusion Matrix and predict Accuracy

Conclusion:

The Credit Card Fraud Detection dataset poses a real-world problem with significant implications for financial security. The imbalance, anonymized features, and critical nature of the task require careful preprocessing, thoughtful feature engineering, and advanced modeling techniques. Achieving high accuracy on this dataset may be misleading due to the dominant non-fraudulent class. As such, a comprehensive evaluation should involve considering recall and precision, striking a balance between detecting fraud and minimizing false positives. In practice, deploying and maintaining such models require ongoing monitoring and adjustment to adapt to evolving fraudulent behaviors.