R Notebook

```
survey_nps_tot = df_clean[df_clean$survey %in% c( "4", "16", "18", "52"),]
```

Variable aggregation

```
survey_nps_tot <- survey_nps_tot %>%
  mutate(direzione_aggregata = case_when(
    direzione %in% c(1, 2, 3, 4, 5, 7, 6, 8, 9) ~ as.factor(direzione),
    TRUE ~ "Altro"
  ))
survey_nps_tot$direzione_aggregata = as.factor((survey_nps_tot$direzione_aggregata))
```

Nas Imputation

```
survey_nps_tot <- kNN(survey_nps_tot, variable = "risk_rating_comm")
```

Dual NPS Variable

```
survey_nps_tot <- survey_nps_tot %>%
  mutate(nps_class = case_when(
      nps < 9 ~ 0,
    nps %in% c(9, 10) ~ 1

  ))
```

Data cleaning

```
survey_nps_tot_clean <- survey_nps_tot[complete.cases(survey_nps_tot[, c("nps_class", "segmento_des_com
                          c("nps_class", "segmento_des_comm", "regione_aggregata" , "direzione_aggrega
```

Random Forest

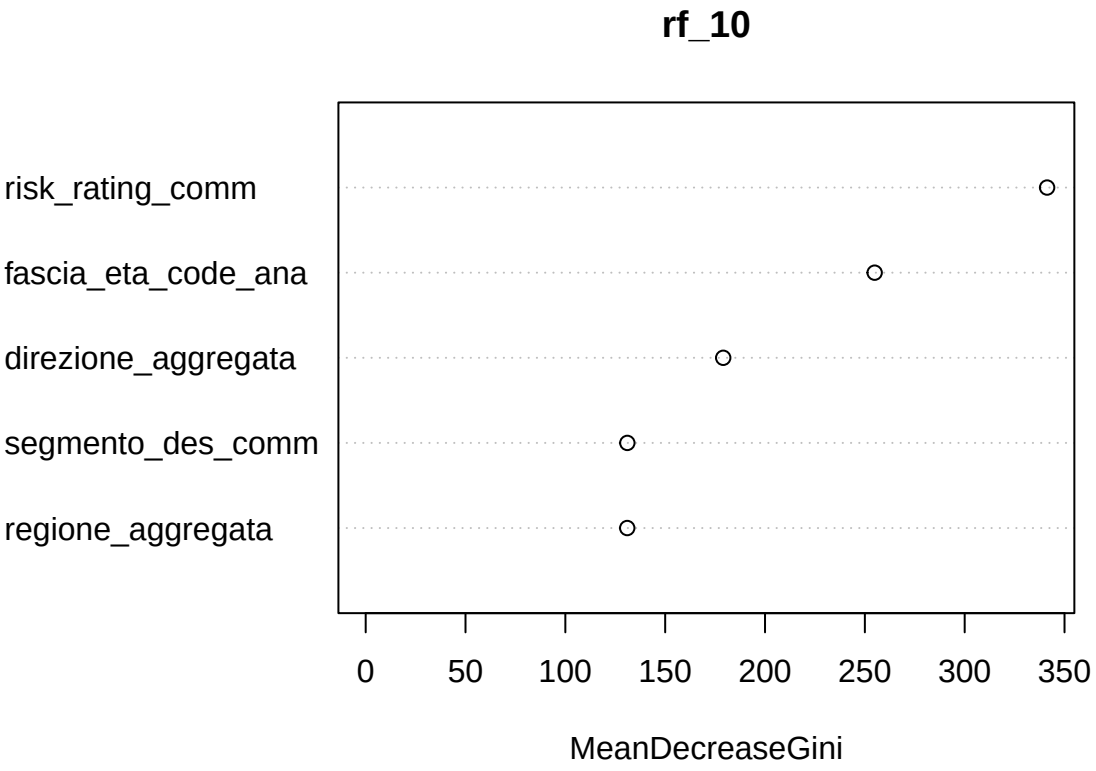
```r
survey_nps_tot_clean$nps_class <- as.factor(survey_nps_tot_clean$nps_class)


# Costruisci la Random Forest con tutte le variabili
set.seed(123)  # Per riproducibilità
rf_10 <- randomForest(nps_class ~ segmento_des_comm +  regione_aggregata +  direzione_aggregata +  fasc

# Visualizza i risultati
print(rf_10)
```

```
##
## Call:
##  randomForest(formula = nps_class ~ segmento_des_comm + regione_aggregata +      direzione_aggregata
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 33.1%
## Confusion matrix:
##     0    1 class.error
## 0 805 2665   0.7680115
## 1 717 6029   0.1062852
```

```r
# Visualizza l'importanza delle variabili
importance(rf_10)
```

```
##                     MeanDecreaseGini
## segmento_des_comm           131.0446
## regione_aggregata           130.9893
## direzione_aggregata         179.0630
## fascia_eta_code_ana         254.9113
## risk_rating_comm            341.2730
```

```r
varImpPlot(rf_10)
```



Predictions

```
set.seed(123)  # Per riproducibilità
train_index <- sample(1:nrow(survey_nps_tot_clean), 0.75 * nrow(survey_nps_tot_clean))
train_set <- survey_nps_tot_clean[train_index, ]
test_set <- survey_nps_tot_clean[-train_index, ]

# Assicurati che nps_class sia un fattore con livelli coerenti
train_set$nps_class <- factor(train_set$nps_class, levels = c("0", "1"))
test_set$nps_class <- factor(test_set$nps_class, levels = c("0", "1"))

# Predizioni basate su soglie
thresholds <- seq(0, 1, by = 0.01)
results <- data.frame(threshold = thresholds, accuracy = NA)

library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following object is masked from 'package:colorspace':
```

```
##
##     coords

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
# Prevedi le probabilità per la classe positiva
predictions_prob <- predict(rf_10, newdata = train_set, type = "prob")[, 2]

# Calcola la curva ROC
roc_curve <- roc(train_set$nps_class, predictions_prob)
```

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

## Setting direction: controls < cases
```

```
# Trova la soglia che massimizza la Youden's J statistic (TPR - FPR)
best_threshold <- coords(roc_curve, "best", ret = "threshold", best.method = "youden")

# Stampa la soglia ottimale
print(paste("Soglia che massimizza la ROC:", best_threshold))
```

```
## [1] "Soglia che massimizza la ROC: 0.779"
```

```
# Calcola e stampa l'AUC
auc_value <- auc(roc_curve)
print(paste("AUC:", auc_value))
```

```
## [1] "AUC: 0.736588894120718"
```

```
# Prevedi le probabilità per la classe positiva nel test set
predictions_prob_test <- predict(rf_7, newdata = test_set, type = "prob")[, 2]

# Usa la soglia ottimale per convertire le probabilità in predizioni binarie
predictions_test <- ifelse(predictions_prob_test > 0.761, "1", "0")

# Calcola la matrice di confusione per il test set
cm_test <- confusionMatrix(as.factor(predictions_test), test_set$nps_class)

# Stampa i risultati della matrice di confusione
print(cm_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0  570  279
##          1  281 1424
##
##                Accuracy : 0.7807
```

```
##                  95% CI : (0.7642, 0.7966)
##     No Information Rate : 0.6668
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5063
##
##  Mcnemar's Test P-Value : 0.9663
##
##             Sensitivity : 0.6698
##             Specificity : 0.8362
##          Pos Pred Value : 0.6714
##          Neg Pred Value : 0.8352
##              Prevalence : 0.3332
##          Detection Rate : 0.2232
##    Detection Prevalence : 0.3324
##       Balanced Accuracy : 0.7530
##
##        'Positive' Class : 0
##
```

```r
# Calcola l'accuratezza, precisione, richiamo e F1-score sul test set
accuracy <- cm_test$overall['Accuracy']
precision <- cm_test$byClass['Pos Pred Value']
recall <- cm_test$byClass['Sensitivity']
f1_score <- 2 * (precision * recall) / (precision + recall)
specificity <- cm_test$byClass["Specificity"]

cat("Model Evaluation Metrics on Test Set:\n",
    "Accuracy   :", round(accuracy, 4), "\n",
    "Precision  :", round(precision, 4), "\n",
    "Recall     :", round(recall, 4), "\n",
    "F1-Score   :", round(f1_score, 4), "\n",
    "Specificity:", round(specificity, 4), "\n")
```

```
## Model Evaluation Metrics on Test Set:
##  Accuracy   : 0.7807
##  Precision  : 0.6714
##  Recall     : 0.6698
##  F1-Score   : 0.6706
##  Specificity: 0.8362
```