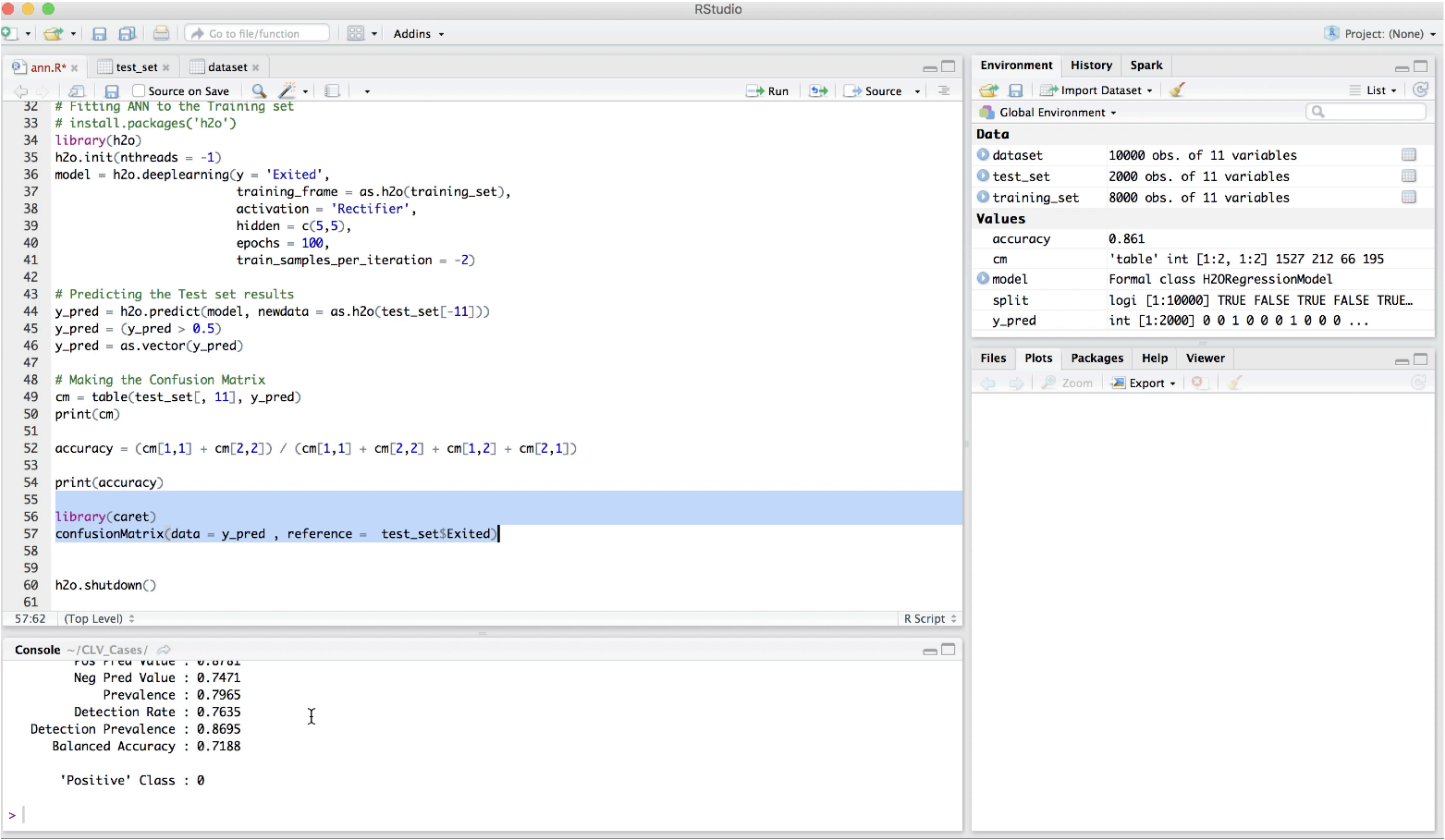
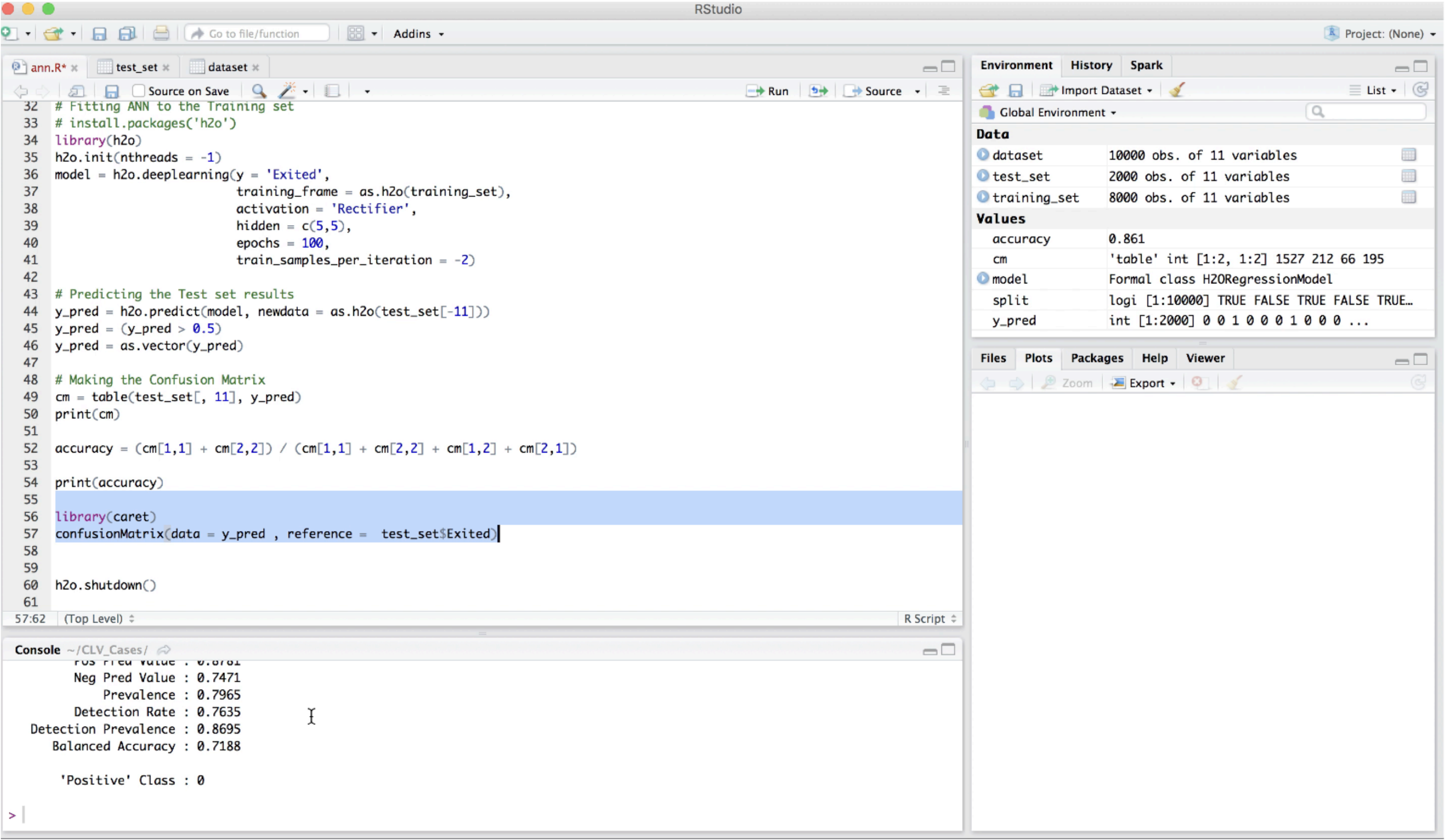
Prof. Dr. Jan Kirenz



```r
# Fitting ANN to the Training set
# install.packages('h2o')
library(h2o)
h2o.init(nthreads = -1)
model = h2o.deeplearning(y = 'Exited',
                         training_frame = as.h2o(training_set),
                         activation = 'Rectifier',
                         hidden = c(5,5),
                         epochs = 100,
                         train_samples_per_iteration = -2)

# Predicting the Test set results
y_pred = h2o.predict(model, newdata = as.h2o(test_set[-11]))
y_pred = (y_pred > 0.5)
y_pred = as.vector(y_pred)

# Making the Confusion Matrix
cm = table(test_set[, 11], y_pred)
print(cm)

accuracy = (cm[1,1] + cm[2,2]) / (cm[1,1] + cm[2,2] + cm[1,2] + cm[2,1])

print(accuracy)

library(caret)
confusionMatrix(data = y_pred , reference =  test_set$Exited)


h2o.shutdown()

```

Data		
dataset	10000 obs. of 11 variables	
test_set	2000 obs. of 11 variables	
training_set	8000 obs. of 11 variables	
Values		
accuracy	0.861	
cm	'table' int [1:2, 1:2] 1527 212 66 195	
model	Formal class H2ORegressionModel	
split	logi [1:10000] TRUE FALSE TRUE FALSE TRUE…	
y_pred	int [1:2000] 0 0 1 0 0 0 1 0 0 0 ...	

```
         Pos Pred Value : 0.8781
         Neg Pred Value : 0.7471
             Prevalence : 0.7965
         Detection Rate : 0.7635
   Detection Prevalence : 0.8695
      Balanced Accuracy : 0.7188

       'Positive' Class : 0

>
```



```r
# Fitting ANN to the Training set
# install.packages('h2o')
library(h2o)
h2o.init(nthreads = -1)
model = h2o.deeplearning(y = 'Exited',
                         training_frame = as.h2o(training_set),
                         activation = 'Rectifier',
                         hidden = c(5,5),
                         epochs = 100,
                         train_samples_per_iteration = -2)

# Predicting the Test set results
y_pred = h2o.predict(model, newdata = as.h2o(test_set[-11]))
y_pred = (y_pred > 0.5)
y_pred = as.vector(y_pred)

# Making the Confusion Matrix
cm = table(test_set[, 11], y_pred)
print(cm)

accuracy = (cm[1,1] + cm[2,2]) / (cm[1,1] + cm[2,2] + cm[1,2] + cm[2,1])

print(accuracy)

library(caret)
confusionMatrix(data = y_pred , reference =  test_set$Exited)


h2o.shutdown()

```

Environment — Global Environment

Data

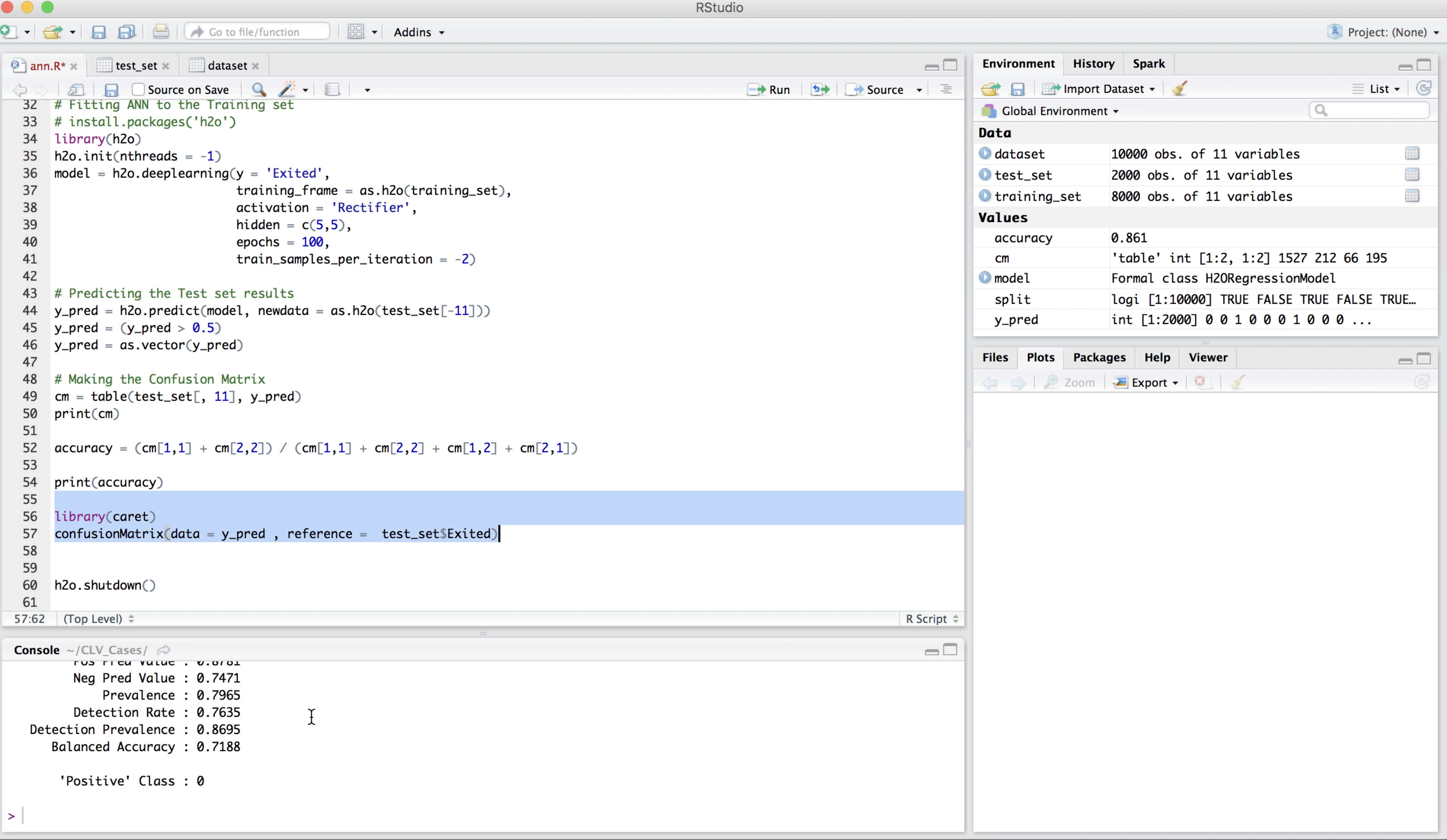
dataset	10000 obs. of 11 variables
test_set	2000 obs. of 11 variables
training_set	8000 obs. of 11 variables

Values

accuracy	0.861
cm	'table' int [1:2, 1:2] 1527 212 66 195
model	Formal class H2ORegressionModel
split	logi [1:10000] TRUE FALSE TRUE FALSE TRUE…
y_pred	int [1:2000] 0 0 1 0 0 0 1 0 0 0 ...

Console ~/CLV_Cases/

```
         Pos Pred Value : 0.8781
         Neg Pred Value : 0.7471
             Prevalence : 0.7965
         Detection Rate : 0.7635
   Detection Prevalence : 0.8695
      Balanced Accuracy : 0.7188

       'Positive' Class : 0

>
```

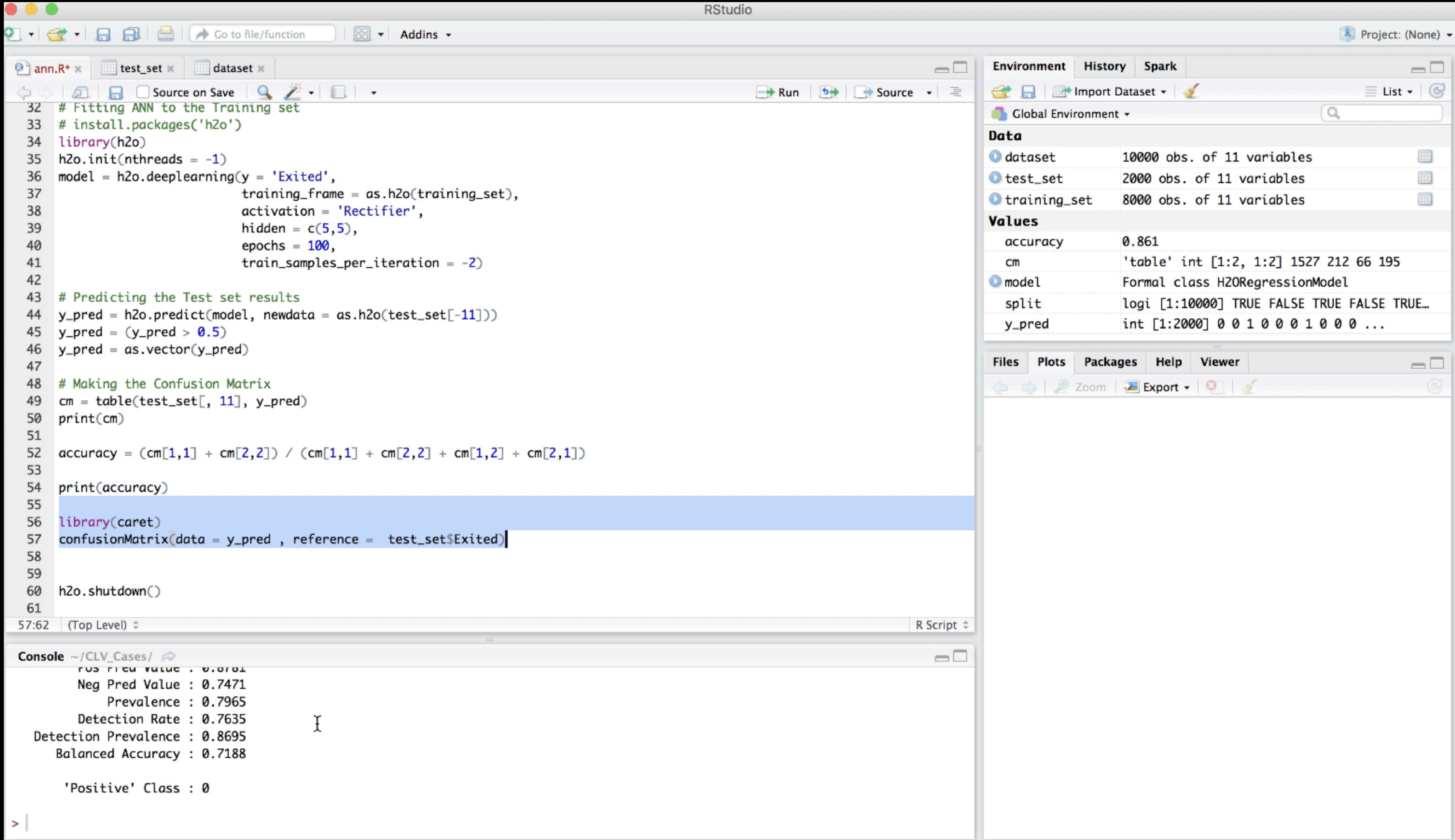

```r
# Fitting ANN to the Training set
# install.packages('h2o')
library(h2o)
h2o.init(nthreads = -1)
model = h2o.deeplearning(y = 'Exited',
                         training_frame = as.h2o(training_set),
                         activation = 'Rectifier',
                         hidden = c(5,5),
                         epochs = 100,
                         train_samples_per_iteration = -2)

# Predicting the Test set results
y_pred = h2o.predict(model, newdata = as.h2o(test_set[-11]))
y_pred = (y_pred > 0.5)
y_pred = as.vector(y_pred)

# Making the Confusion Matrix
cm = table(test_set[, 11], y_pred)
print(cm)

accuracy = (cm[1,1] + cm[2,2]) / (cm[1,1] + cm[2,2] + cm[1,2] + cm[2,1])

print(accuracy)

library(caret)
confusionMatrix(data = y_pred , reference =  test_set$Exited)


h2o.shutdown()

```

Data		
dataset	10000 obs. of 11 variables	
test_set	2000 obs. of 11 variables	
training_set	8000 obs. of 11 variables	
Values		
accuracy	0.861	
cm	'table' int [1:2, 1:2] 1527 212 66 195	
model	Formal class H2ORegressionModel	
split	logi [1:10000] TRUE FALSE TRUE FALSE TRUE…	
y_pred	int [1:2000] 0 0 1 0 0 0 1 0 0 0 ...	

```
     Neg Pred Value : 0.7471
         Prevalence : 0.7965
     Detection Rate : 0.7635
Detection Prevalence : 0.8695
   Balanced Accuracy : 0.7188

    'Positive' Class : 0

>
```

CHURN MODEL WITH DEEP LEARNING

- **Task**: Identify customers who are likely to defect
- **Model**: Neural Network (Deep Learning)



```r
# Fitting ANN to the Training set
# install.packages('h2o')
library(h2o)
h2o.init(nthreads = -1)
model = h2o.deeplearning(y = 'Exited',
                         training_frame = as.h2o(training_set),
                         activation = 'Rectifier',
                         hidden = c(5,5),
                         epochs = 100,
                         train_samples_per_iteration = -2)

# Predicting the Test set results
y_pred = h2o.predict(model, newdata = as.h2o(test_set[-11]))
y_pred = (y_pred > 0.5)
y_pred = as.vector(y_pred)

# Making the Confusion Matrix
cm = table(test_set[, 11], y_pred)
print(cm)

accuracy = (cm[1,1] + cm[2,2]) / (cm[1,1] + cm[2,2] + cm[1,2] + cm[2,1])

print(accuracy)

library(caret)
confusionMatrix(data = y_pred , reference =  test_set$Exited)


h2o.shutdown()
```

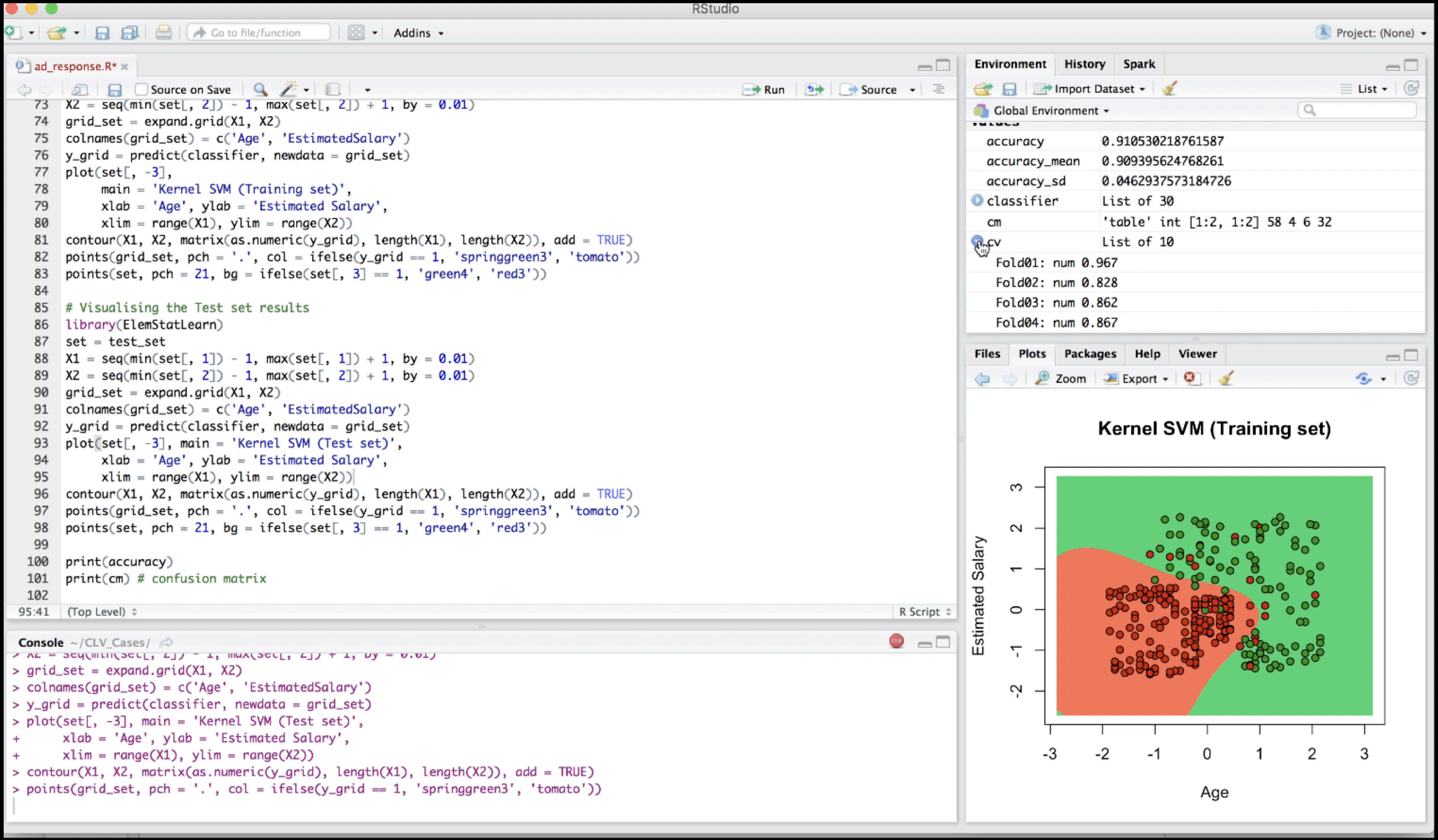
Console ~/CLV_Cases/

```
      Pos Pred Value : 0.8781
      Neg Pred Value : 0.7471
          Prevalence : 0.7965
      Detection Rate : 0.7635
Detection Prevalence : 0.8695
   Balanced Accuracy : 0.7188

    'Positive' Class : 0

>
```

Data	
dataset	10000 obs. of 11 variables
test_set	2000 obs. of 11 variables
training_set	8000 obs. of 11 variables
Values	
accuracy	0.861
cm	'table' int [1:2, 1:2] 1527 212 66 195
model	Formal class H2ORegressionModel
split	logi [1:10000] TRUE FALSE TRUE FALSE TRUE…
y_pred	int [1:2000] 0 0 1 0 0 0 1 0 0 0 ...

OFFER/CONTACT-OPTIMIZATION WITH SVM

- **Task**: Identify users who are open to an specific offer
- **Data**: Ad-click of Social Network Users
- **Model**: Kernel Support Vector Machine (SVM)
- **Validation**: k-Fold Cross Validation



```r
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3],
     main = 'Kernel SVM (Training set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))

# Visualising the Test set results
library(ElemStatLearn)
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3], main = 'Kernel SVM (Test set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))

print(accuracy)
print(cm) # confusion matrix
```

```
accuracy        0.910530218761587
accuracy_mean   0.909395624768261
accuracy_sd     0.0462937573184726
classifier      List of 30
cm              'table' int [1:2, 1:2] 58 4 6 32
cv              List of 10
  Fold01: num 0.967
  Fold02: num 0.828
  Fold03: num 0.862
  Fold04: num 0.867
```

```
> X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
> grid_set = expand.grid(X1, X2)
> colnames(grid_set) = c('Age', 'EstimatedSalary')
> y_grid = predict(classifier, newdata = grid_set)
> plot(set[, -3], main = 'Kernel SVM (Test set)',
+      xlab = 'Age', ylab = 'Estimated Salary',
+      xlim = range(X1), ylim = range(X2))
> contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
> points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
```