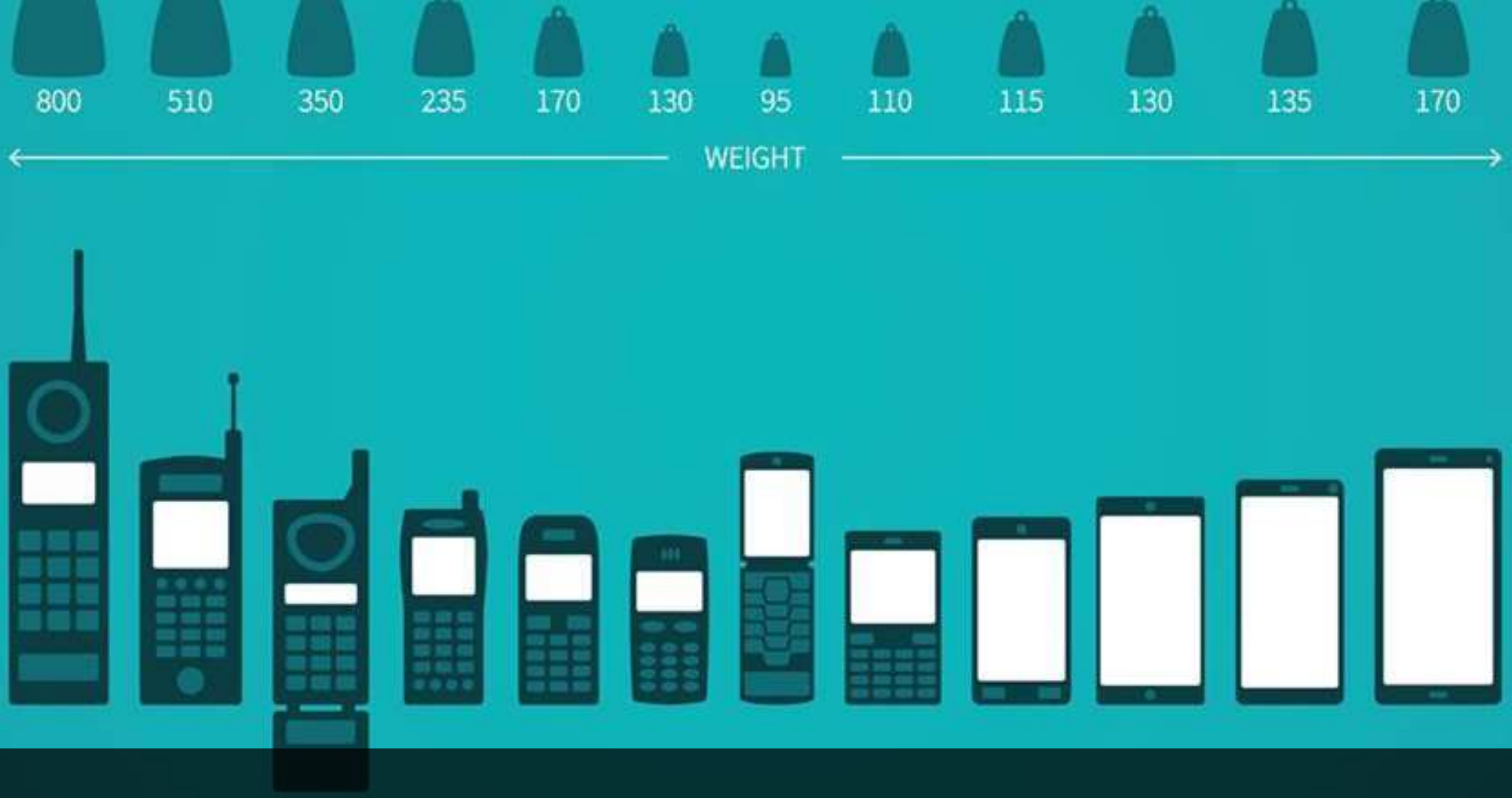


Mobile Price Classification Dataset

Tesina di Metodologie Statistiche per i Big Data

A cura di:

Lorenzo Galeazzi

Marco Belardinelli

Obiettivi della presentazione

- Provare quale tra i vari metodi a nostra disposizione è il migliore per fare previsioni sul nostro dataset
- Analisi delle variabili che più influiscono sulla classificazione
- Differenze tra metodi supervisionati e non supervisionati

Pulizia dataset

```
> dim(mobile_train_df)
[1] 2000   21
```

Nel dataset troviamo 21 variabili e 2000 osservazioni.

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height
1	842	0	2.2	0	1	0	7	0.6	188	2	2	20
2	1021	1	0.5	1	0	1	53	0.7	136	3	6	905
3	563	1	0.5	1	2	1	41	0.9	145	5	6	1263
4	615	1	2.5	0	0	0	10	0.8	131	6	9	1216
5	1821	1	1.2	0	13	1	44	0.6	141	2	14	1208
6	1859	0	0.5	1	3	0	22	0.7	164	1	7	1004

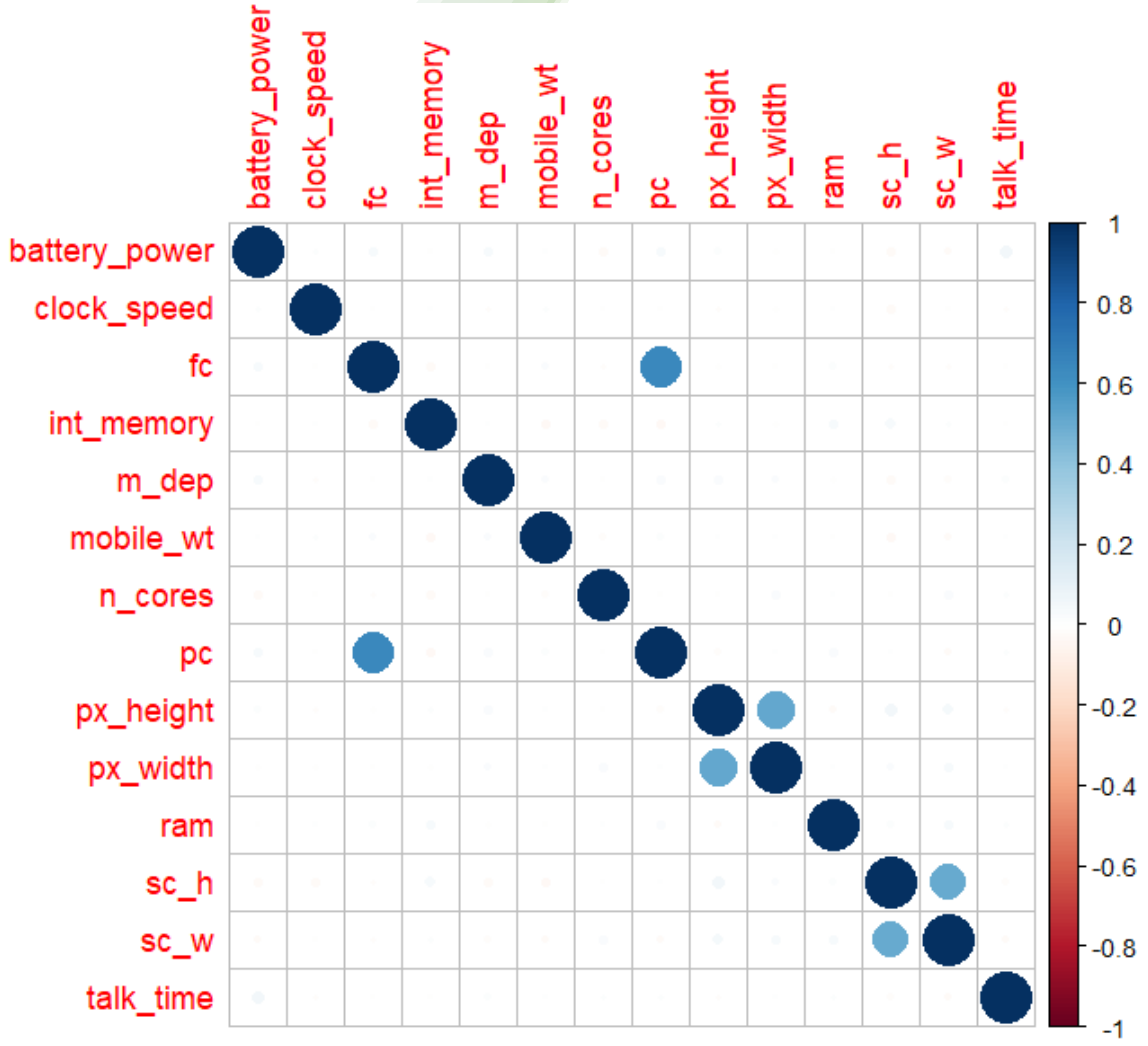
	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range
1	756	2549	9	7	19	0	0	1	1
2	1988	2631	17	3	7	1	1	0	2
3	1716	2603	11	2	9	1	1	0	2
4	1786	2769	16	8	11	1	0	0	2
5	1212	1411	8	2	15	1	1	0	1
6	1654	1067	17	1	10	1	0	0	1

Alcune variabili sono numeriche altre binomiali.

```
mobile_train_df$blue = as.factor(mobile_train_df$blue)
mobile_train_df$dual_sim = as.factor(mobile_train_df$dual_sim)
mobile_train_df$four_g = as.factor(mobile_train_df$four_g)
mobile_train_df$three_g = as.factor(mobile_train_df$three_g)
mobile_train_df$touch_screen = as.factor(mobile_train_df$touch_screen)
mobile_train_df$wifi = as.factor(mobile_train_df$wifi)
mobile_train_df$price_range = as.factor(mobile_train_df$price_range)
```

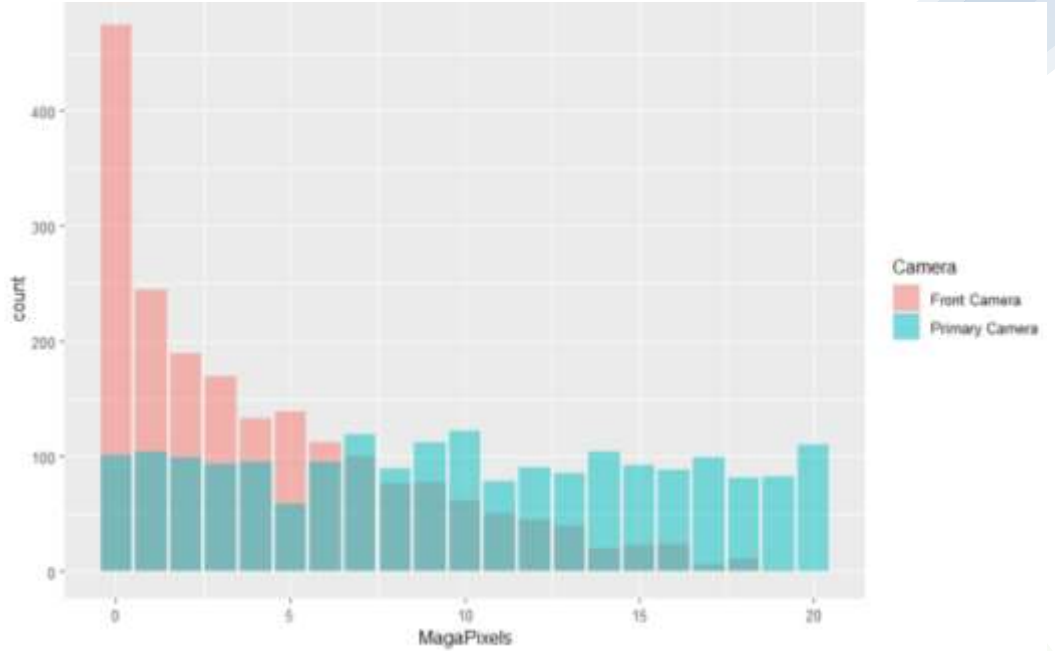
Non sono presenti osservazioni doppie o con valori mancanti procediamo quindi a trasformare le variabili binomiali in fattori.

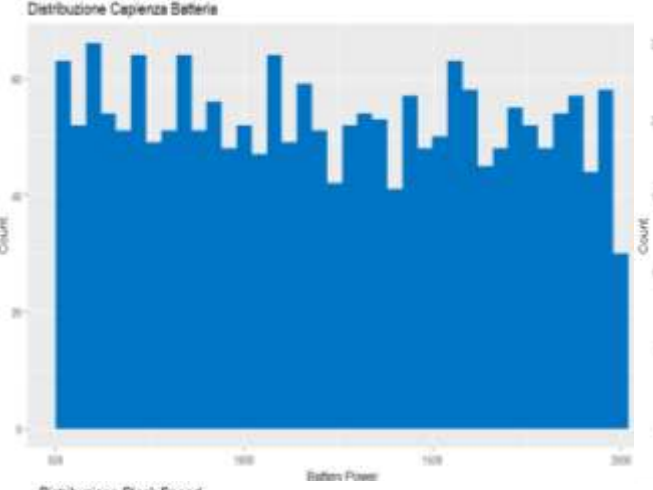
Correlazioni

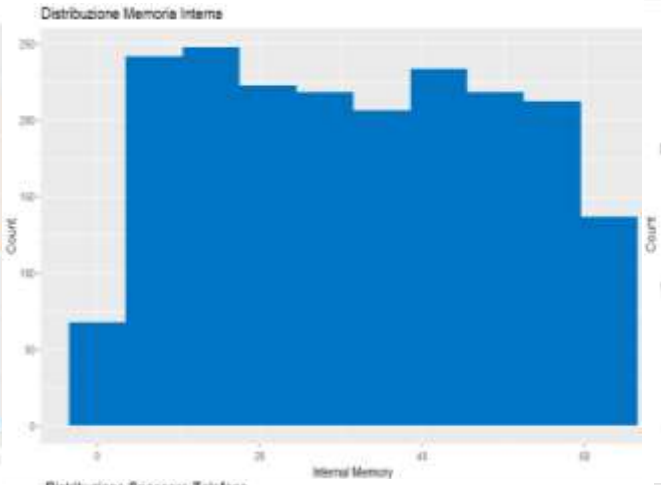


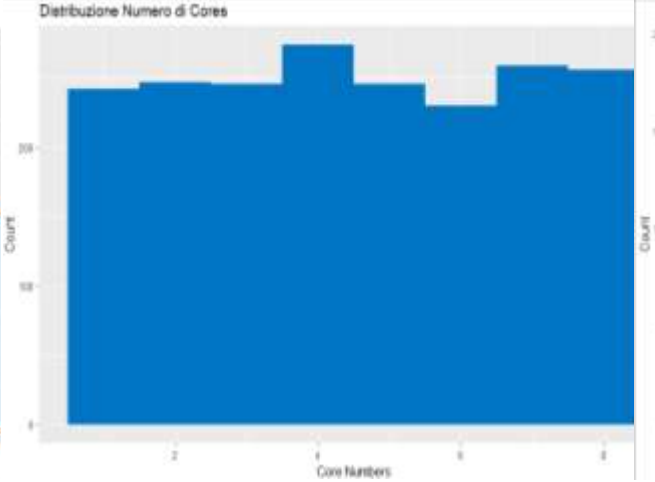
Con questa tabella cerchiamo eventuali correlazioni tra le variabili, notiamo:

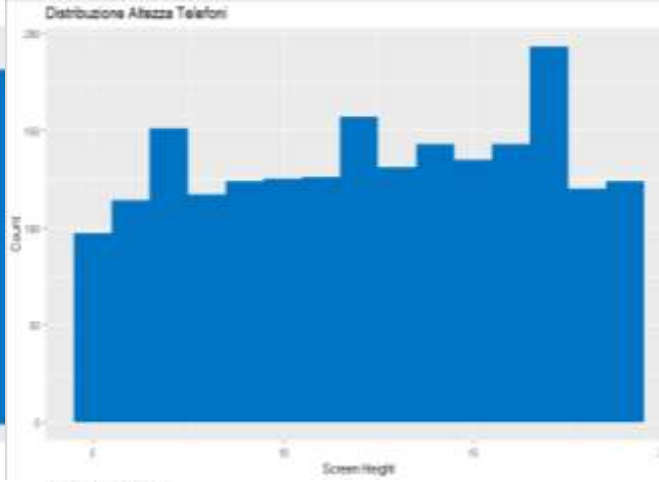
- risoluzione pixel in altezza - risoluzione pixel in larghezza (0.51)
- altezza schermo - larghezza schermo (0.51)
- fotocamera interna - fotocamera principale (0.64)

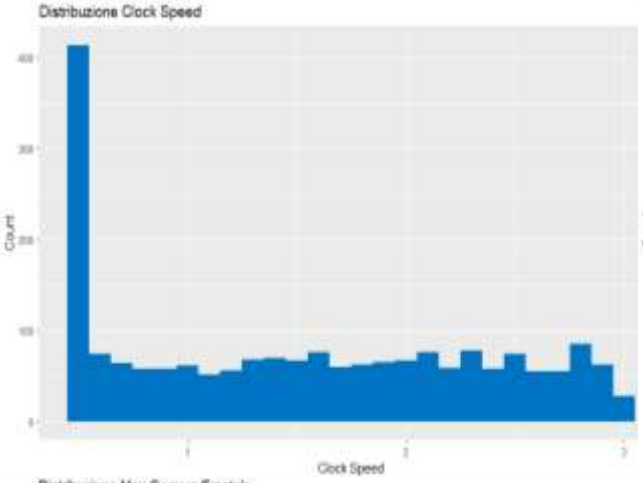


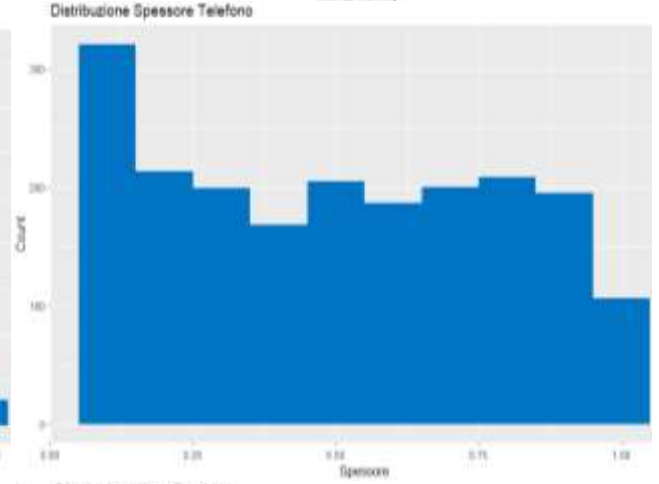

Distribuzione Capienza Batteria
Count
Battery Power

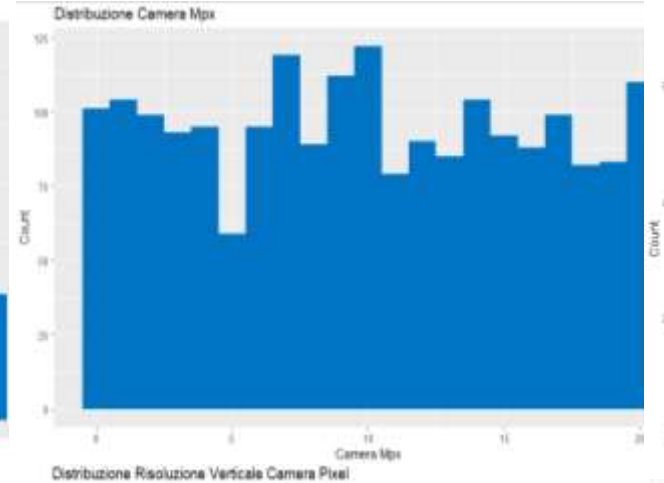

Distribuzione Memoria Interna
Count
Internal Memory

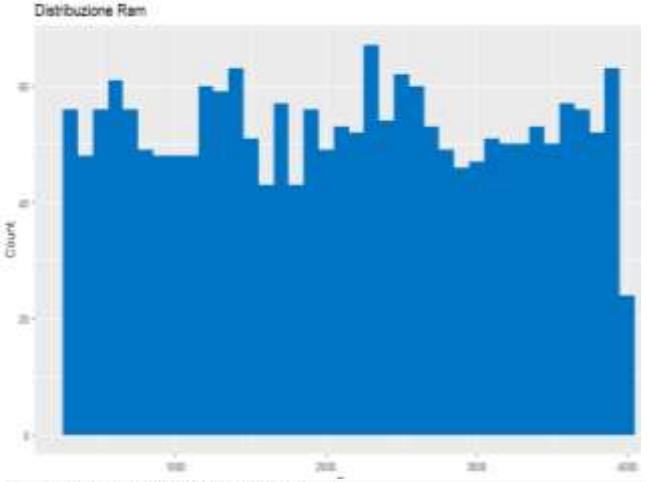

Distribuzione Numero di Cores
Count
Core Numbers

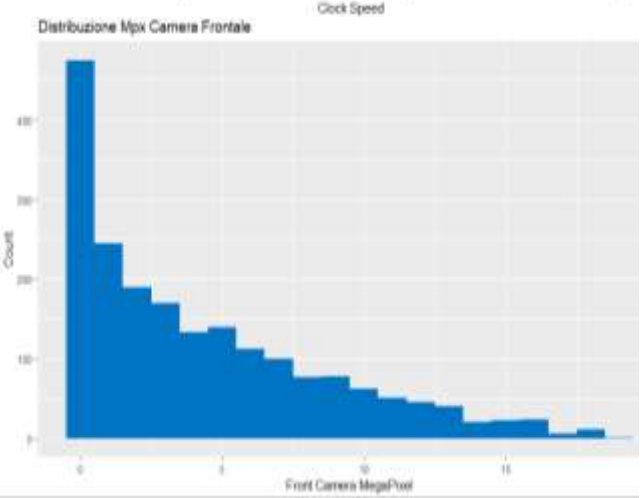

Distribuzione Altezza Telefoni
Count
Screen Height

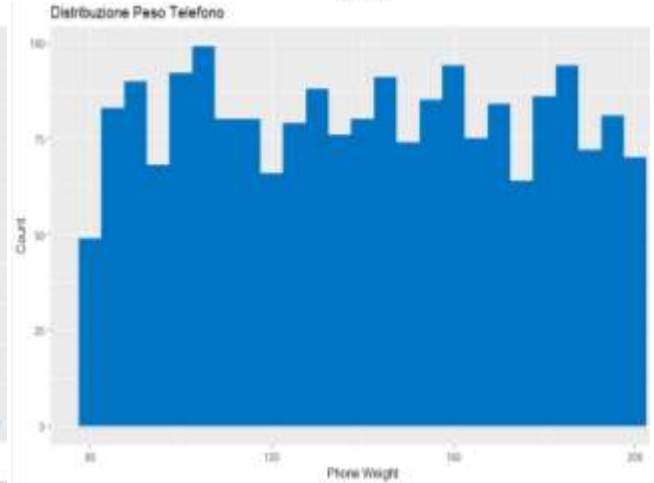

Distribuzione Clock Speed
Count
Clock Speed

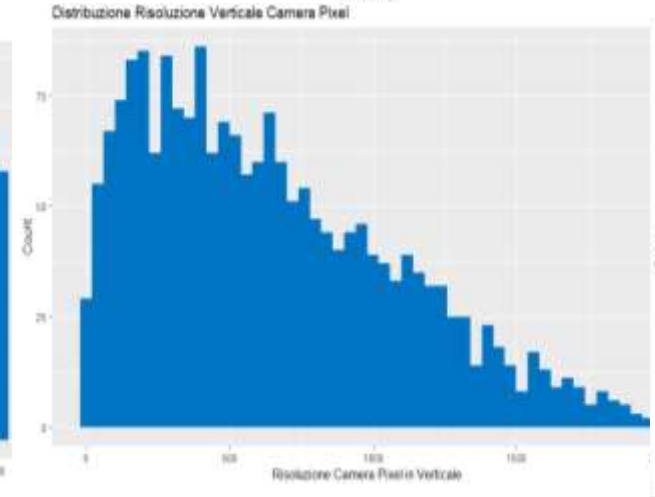

Distribuzione Spessore Telefono
Count
Spessore

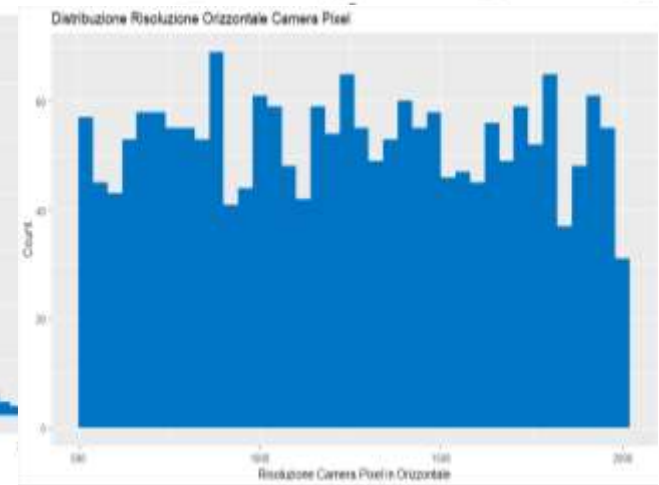

Distribuzione Camera Mpx
Count
Camera Mpx


Distribuzione Ram
Count


Distribuzione Mpx Camera Frontale
Count
Front Camera MegaPixel


Distribuzione Peso Telefono
Count
Phone Weight


Distribuzione Risoluzione Verticale Camera Pixel
Count
Risoluzione Camera Pixel in Verticale


Distribuzione Risoluzione Orizzontale Camera Pixel
Count
Risoluzione Camera Pixel in Orizzontale

Standardizzazione Variabili e Divisione Train-Test

Al fine di procedere con le previsioni standardizziamo le nostre variabili

```
> numeric_data = mobile_train_df[,c(1,3,5,7:17)]
> fun_norm = function(x){
+     return((x-mean(x))/sd(x))
+ }
> data_std = as.data.frame(lapply(numeric_data, fun_norm))
> data = data.frame(data_std, mobile_train_df[,c(2,4,6,18:21)])
```

… per poi dividerle in TRAIN & TEST.

```
> set.seed(123)
> sample_data<- sample.split(data,SplitRatio = 0.75)
> sample_data
 [1]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE
[17]  TRUE  TRUE  TRUE FALSE FALSE
> train_data<-subset(data, sample_data==TRUE)
> dim(train_data)
[1] 1429   21
> test_data<-subset(data, sample_data==FALSE)
> dim(test_data)
[1] 571  21
> prop.table(table(train_data$price_range))

        0         1         2         3
0.2477257 0.2540238 0.2533240 0.2449265
> prop.table(table(test_data$price_range))

        0         1         2         3
0.2556918 0.2399299 0.2416813 0.2626970
```

Confrontando i due subset vediamo che la variabile che andremo a studiare si distribuisce equamente in entrambi.

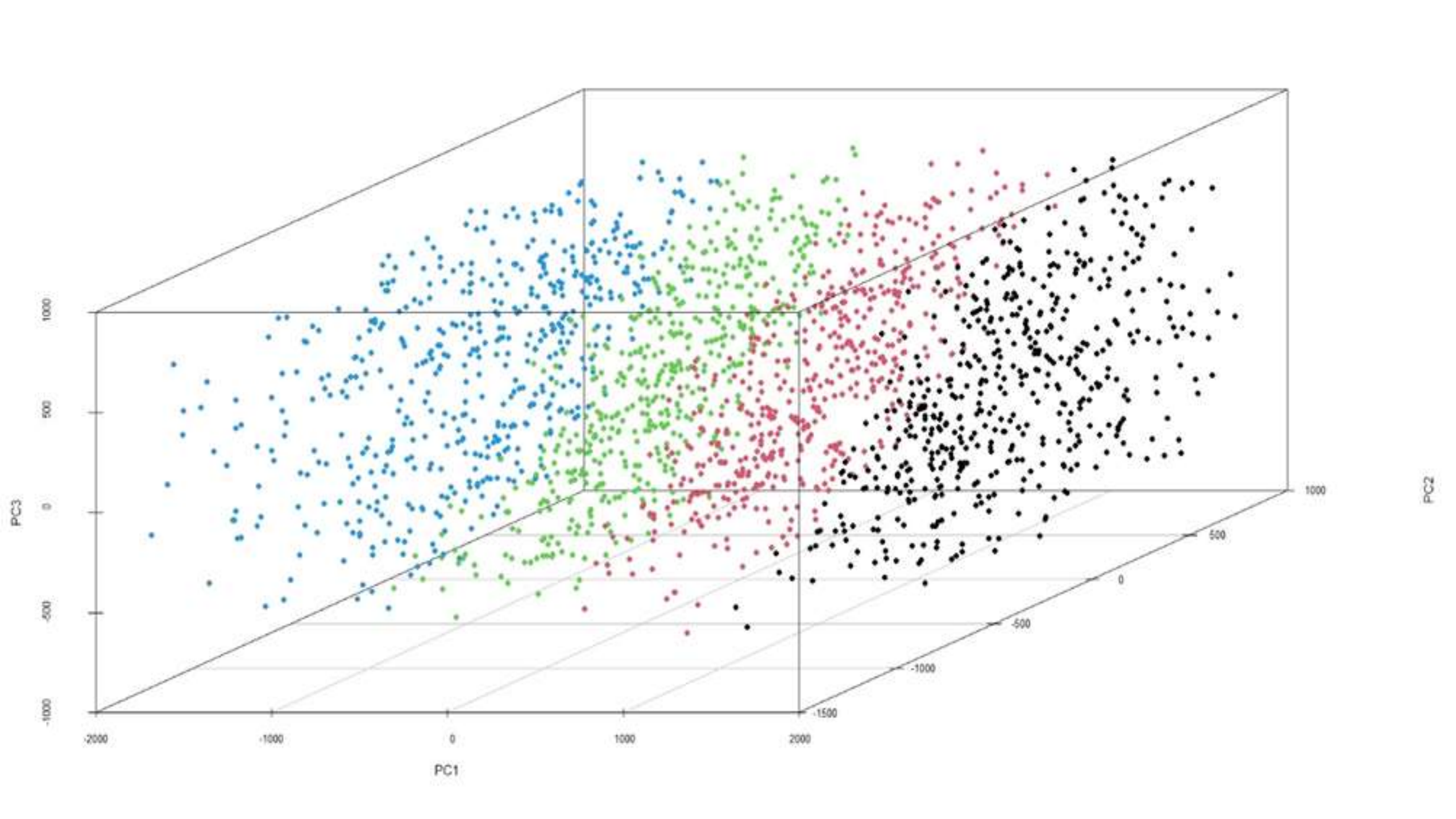
Principal Component Analysis (PCA)

```
Importance of components:
                          PC1      PC2      PC3      PC4      PC5      PC6     PC7     PC8
Standard deviation     1084.7779 538.4416 439.5203 305.99036 35.40763 18.12040 6.84271 5.46913
Proportion of Variance    0.6704   0.1652   0.1101   0.05335  0.00071  0.00019 0.00003 0.00002
Cumulative Proportion     0.6704   0.8356   0.9457   0.99903  0.99974  0.99993 0.99995 0.99997
                           PC9    PC10   PC11   PC12   PC13    PC14
Standard deviation     5.22756 3.02570 2.914  2.281  0.815   .2877
Proportion of Variance 0.00002 0.00001 0.000  0.000  0.0000  0.0000
Cumulative Proportion  0.99999 0.99999 1.000  1.000  1.0000  1.0000
```

Sommando le prime quattro componenti principali arriviamo al 99% varianza totale.

Avevamo già visto quali variabili erano le più significative, gli autovettori delle prime quattro PC ci confermano la loro utilità.

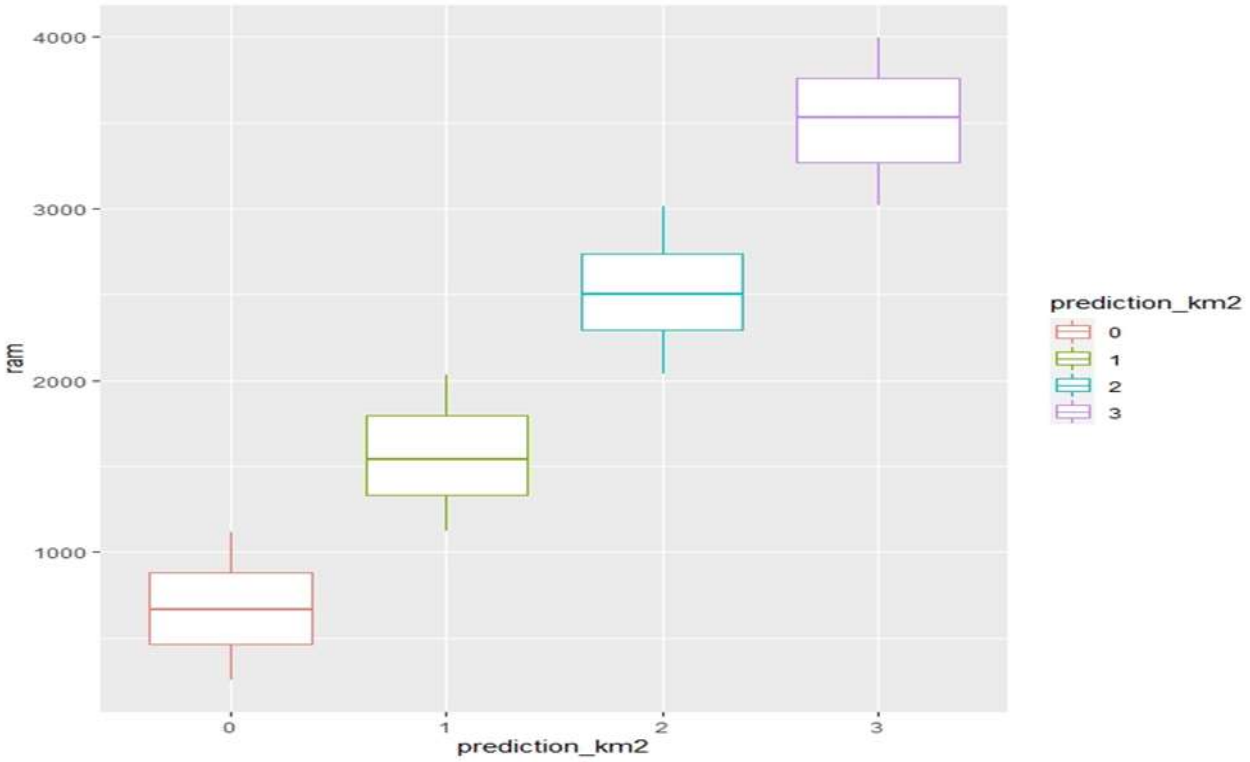
```
> pcav$rotation
                         PC1           PC2           PC3           PC4
battery_power  3.471979e-04 -1.038840e-02  9.994425e-01 -3.171326e-02
clock_speed   -2.630029e-06  2.087359e-05  2.144006e-05 -1.396166e-05
fc            -6.055997e-05  6.733209e-05  3.293617e-04 -8.234845e-05
int_memory    -5.482375e-04 -6.847613e-05 -1.472883e-04  1.190878e-03
m_dep          2.534142e-06 -1.515199e-05  2.212233e-05 -6.542344e-07
mobile_wt      8.443485e-05 -3.911582e-05  1.500664e-04  8.096725e-05
n_cores       -1.035355e-05 -3.924499e-05 -1.588837e-04 -2.275485e-04
pc            -1.624202e-04  9.088395e-05  4.286954e-04 -4.638556e-04
px_height      9.903074e-03 -7.250362e-01  1.430758e-02  6.884899e-01
px_width      -9.647755e-04 -6.886014e-01 -3.014695e-02 -7.245119e-01
ram           -9.999503e-01 -6.519582e-03  5.178224e-04  7.505923e-03
sc_h          -6.123714e-05 -3.718628e-04 -2.846868e-04  5.401018e-04
sc_w          -1.422252e-04 -3.656035e-04 -2.157749e-04  1.337468e-04
talk_time     -5.468870e-05  2.090449e-05  6.481607e-04 -3.502860e-04
```


PC3
1000
500
0
-500
-1000
-2000
-1000
0
1000
2000
PC1
1000
500
0
-500
-1000
-1500
PC2

K-Means

```
km2 = kmeans(scale(numeric_data[,11]), centers = 4, nstart = 100)
prediction_km2 = as.factor(km2$cluster -1)
cluster_km2 = data.frame(mobile_train_df, prediction_km2)
conf_km_ram = table(cluster_km2$price_range,cluster_km2$prediction_km2)
names(dimnames(conf_km_ram)) = c( "observed", "predicted")
cluster_km2$prediction_km2[cluster_km2$prediction_km2 == "2"] = "5"
cluster_km2$prediction_km2[cluster_km2$prediction_km2 == "3"] = "2"
cluster_km2$prediction_km2[is.na(cluster_km2$prediction_km2) ] = "3"
accuracy_km_ram = sum(diag(conf_km_ram))/2000
```

```
          predicted
observed   0   1   2   3
       0 403  97   0   0
       1  54 321 125   0
       2   0  65 347  88
       3   0   0  69 431

> accuracy_km_ram
[1] 0.751
```



Nearest Neighbor

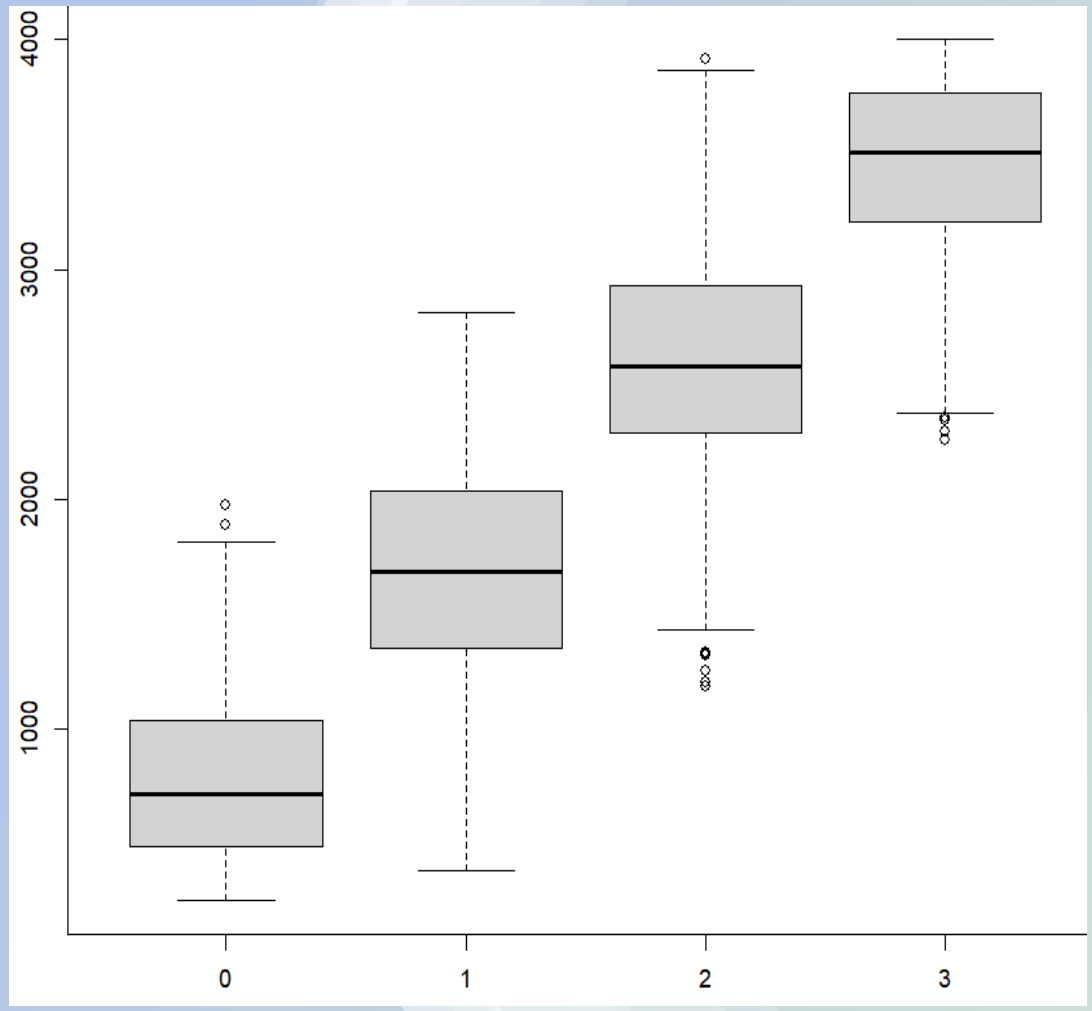
Con K=20 ovvero la radice del numero di osservazioni del nostro dataset abbiamo ottenuto un'accuracy elevata.

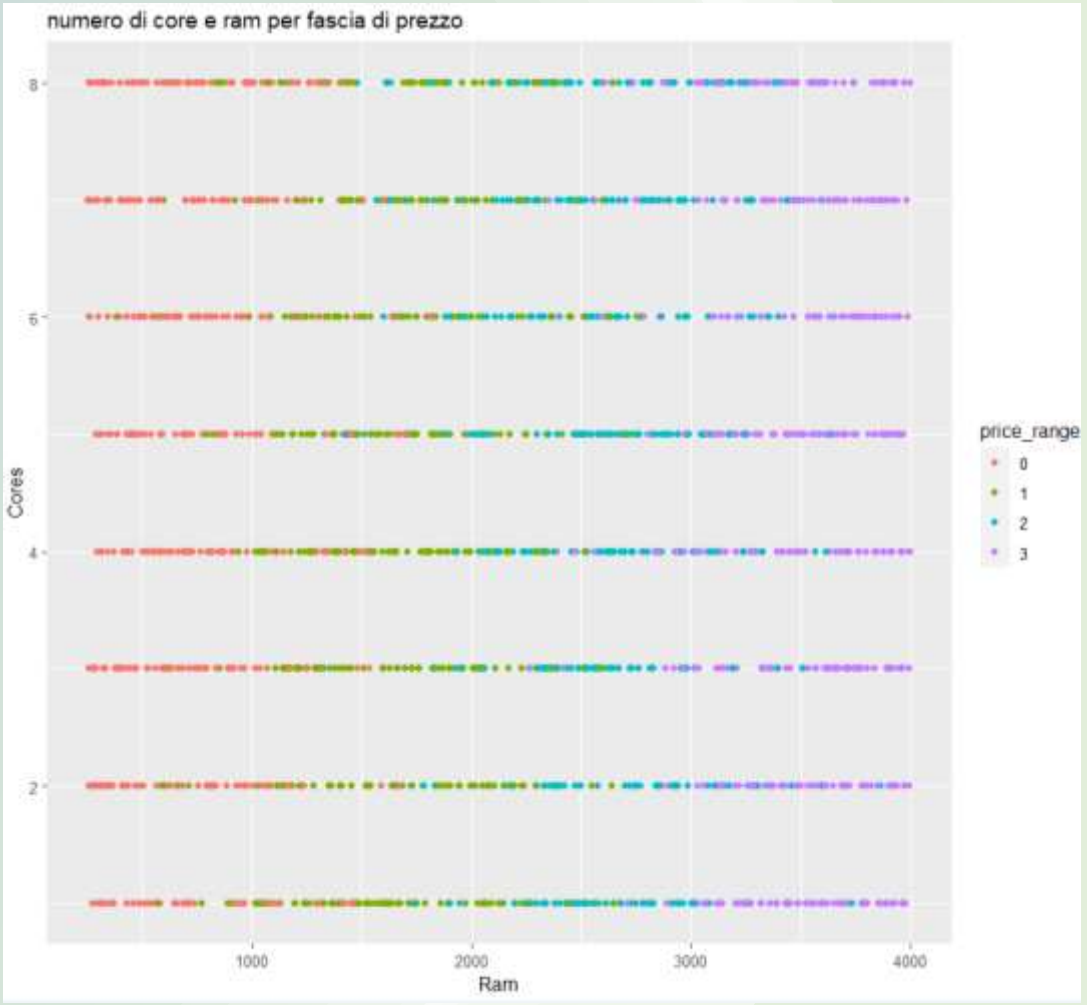
```
> cl = as.factor(train_data$price_range)
> pr_knn2 = knn(train_data, test_data, cl , k = 20)
> tab4.2=xtabs(~pr_knn2+test_data$price_range)
> valori_beccati4.2 = diag(tab4.2)
> accuracy_knn2 = sum(valori_beccati4.2)/length(test_data$price_range)
> tab4.2
       test_data$price_range
pr_knn2   0   1   2   3
      0 130   4   0   0
      1  16 122   5   0
      2   0  11 124  13
      3   0   0   9 137
> accuracy_knn2
[1] 0.8984238
```

Tuttavia, con un K più alto l'accuracy sale poiché le nostre classi sono equamente popolate quindi non abbiamo il rischio che un osservazione finisca nella classe più numerosa.

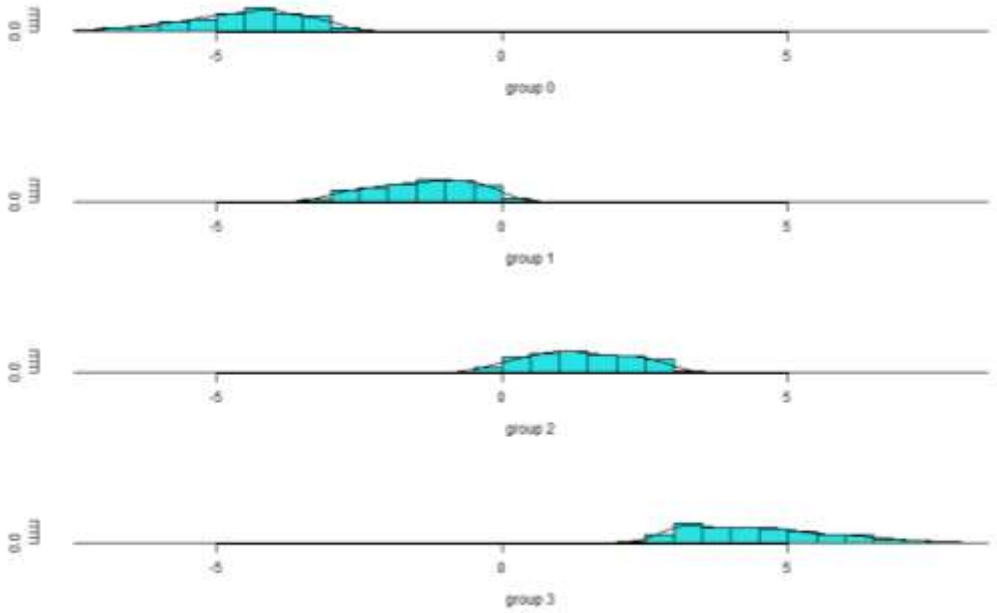
```
> cl = as.factor(train_data$price_range)
> pr_knn2 = knn(train_data, test_data, cl , k = 250)
> tab4.2=xtabs(~pr_knn2+test_data$price_range)
> valori_beccati4.2 = diag(tab4.2)
> accuracy_knn2 = sum(valori_beccati4.2)/length(test_data$price_range)
> tab4.2
       test_data$price_range
pr_knn2   0   1   2   3
      0 144   5   0   0
      1   2 125   6   0
      2   0   7 132  18
      3   0   0   0 132
> accuracy_knn2
[1] 0.9334501
```

RAM & PRICE RANGE





Analisi Discriminante

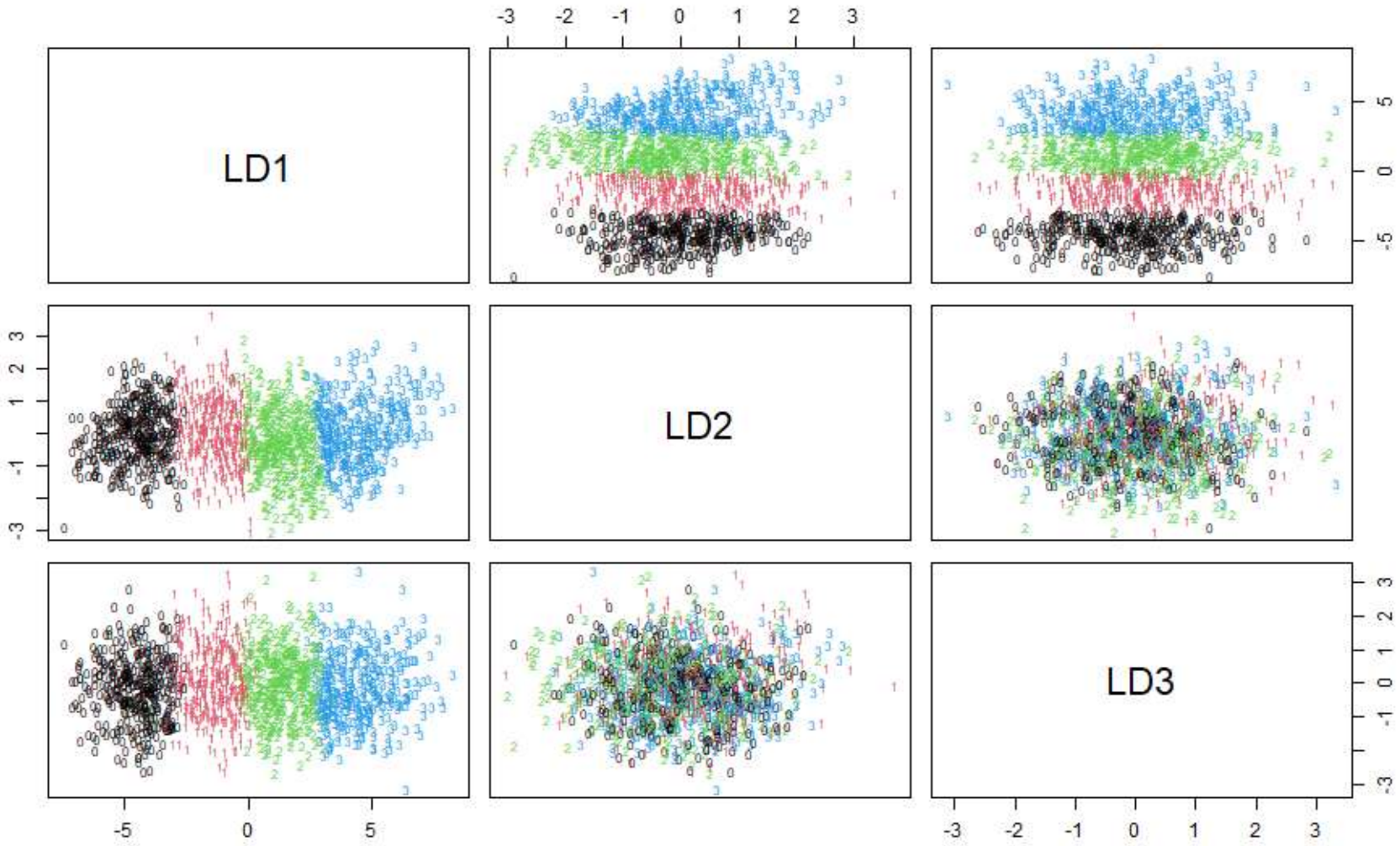


L’immagine parla da sola, l’analisi discriminante sembra essere un ottimo metodo per fare previsioni su questo dataset

Andando a vedere la Confusion Matrix infatti:

```
> confusionMatrix(y_pred,y_oss)
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2   3
         0 142   3   0   0
         1   4 131   9   0
         2   0   3 129   7
         3   0   0   0 143

Overall Statistics

               Accuracy : 0.9545
```

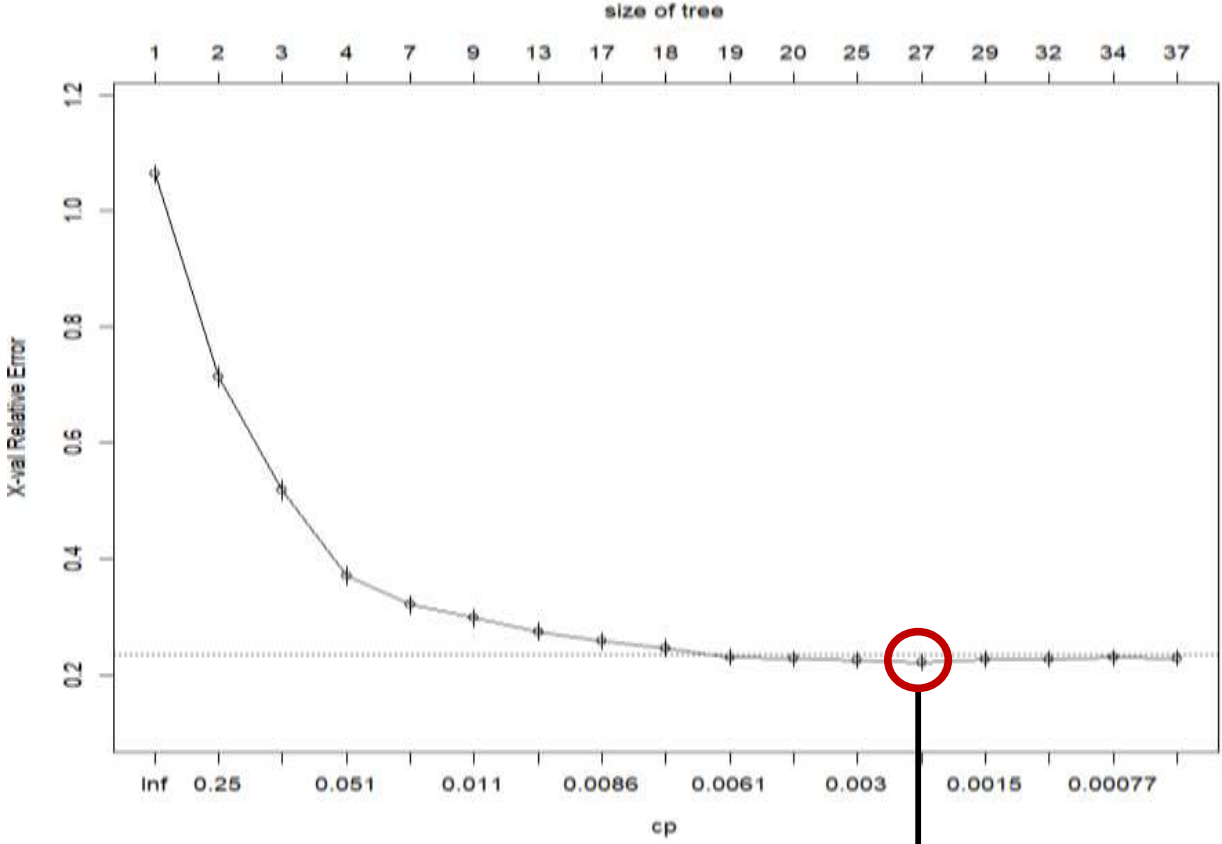
```
Coefficients of linear discriminants:
                      LD1          LD2          LD3
battery_power  0.7205542821  0.46218951 -0.18818901
clock_speed   -0.0368912945 -0.03580524 -0.32205585
fc            -0.0032945151 -0.22388178  0.23992215
int_memory     0.0514573772  0.26101101 -0.09848876
m_dep          0.0304549657  0.13011177  0.34995390
mobile_wt     -0.1283122215 -0.35259265  0.43474642
n_cores        0.0208063641 -0.33260756 -0.25998332
pc             0.0174841556  0.08853178 -0.06782399
px_height      0.4121151241  0.21399362  0.46501081
px_width       0.3898941426  0.30988595 -0.01747648
ram            3.3380777365 -0.21294224 -0.03145825
sc_h           0.0276062302  0.32980332 -0.01825471
sc_w          -0.0129676008 -0.07835732 -0.10072119
talk_time      0.0008143822  0.06350197  0.26197731
blue1         -0.0333176993 -0.03348412 -0.27764035
dual_sim1     -0.1209187770  0.17823477 -0.01399073
four_g1       -0.0291159392  0.35498242 -0.67288796
three_g1       0.0369983789 -0.32628663  0.54743640
touch_screen1 -0.0308031808  0.29930015  0.15998847
wifi1         -0.0845529704 -0.07648633  0.16870807

Proportion of trace:
   LD1    LD2    LD3
0.9962 0.0028 0.0011
```



Coefficienti per la creazione delle funzioni discriminanti.

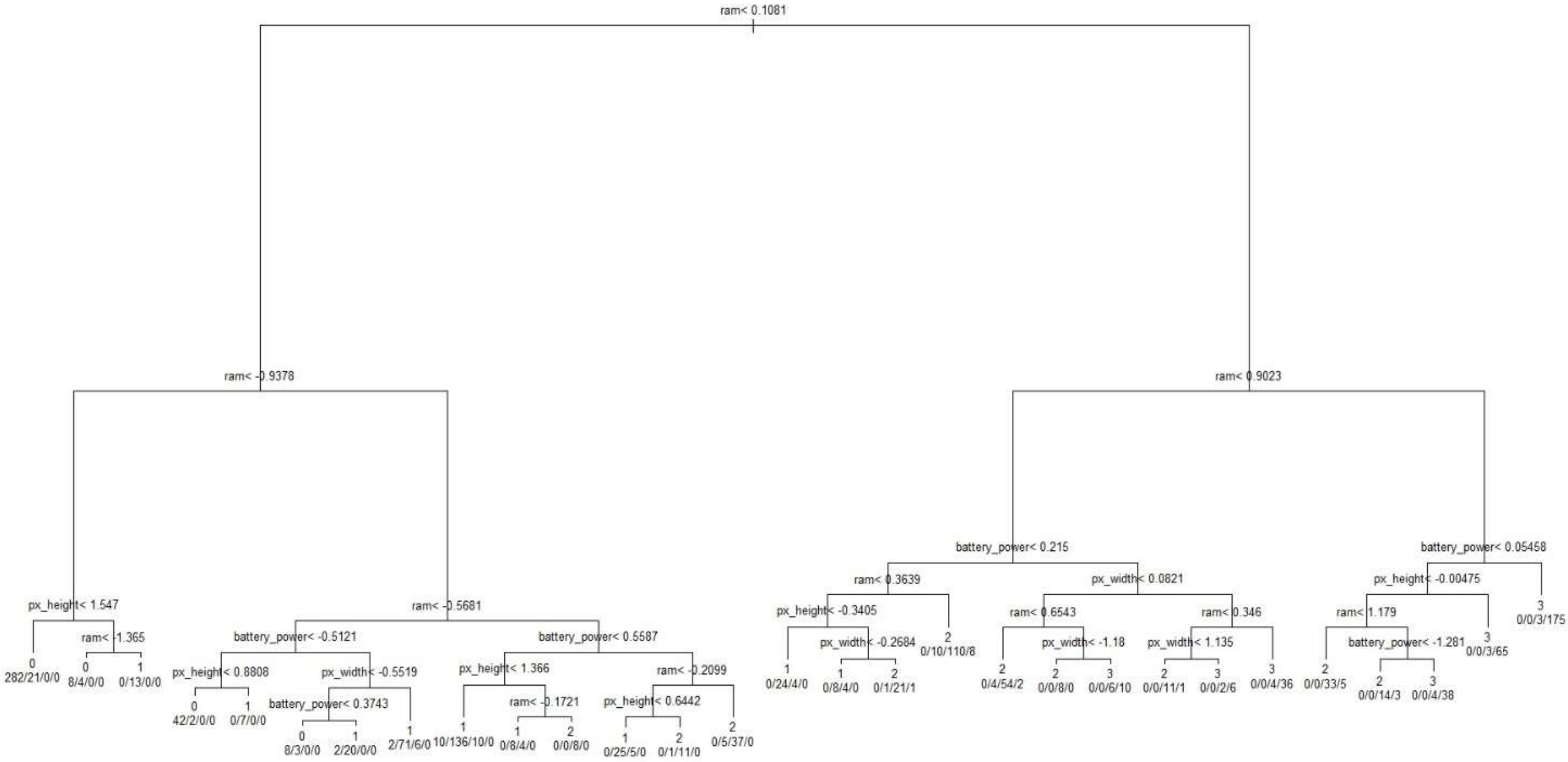
Decision Tree

```
tree<-rpart(price_range~., data = train_data,
            control = rpart.control(cp=0.0001),
            parms = list(split="gini"))
```




```
> bestcp
[1] 0.001876173
```

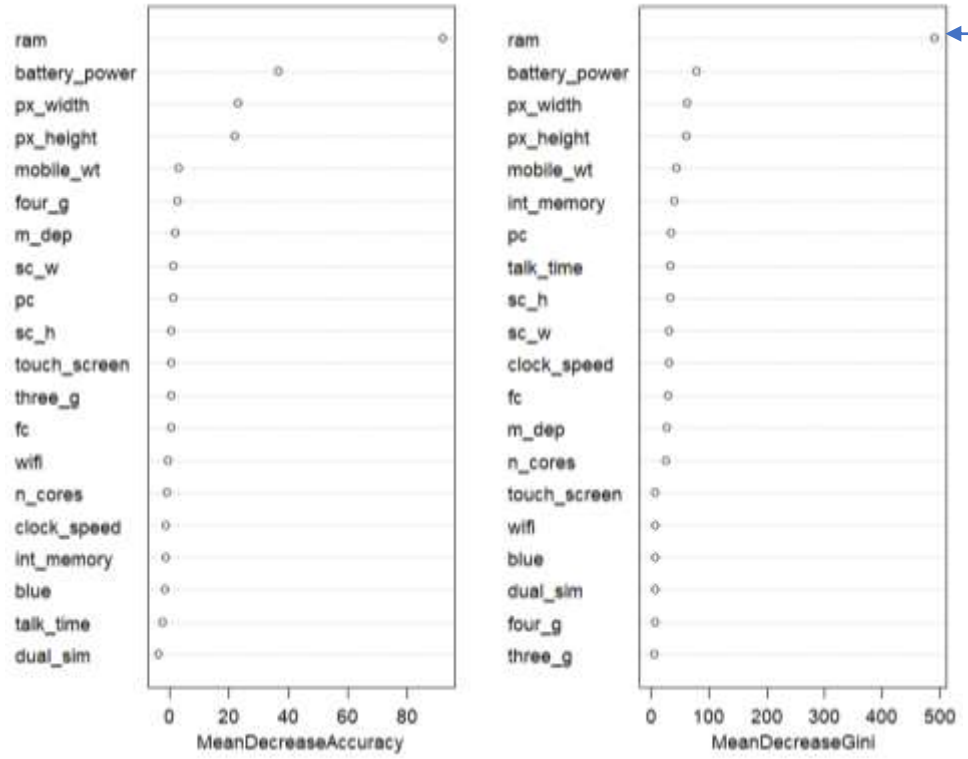
```
Variables actually used in tree construction:
[1] battery_power px_height    px_width      ram
```

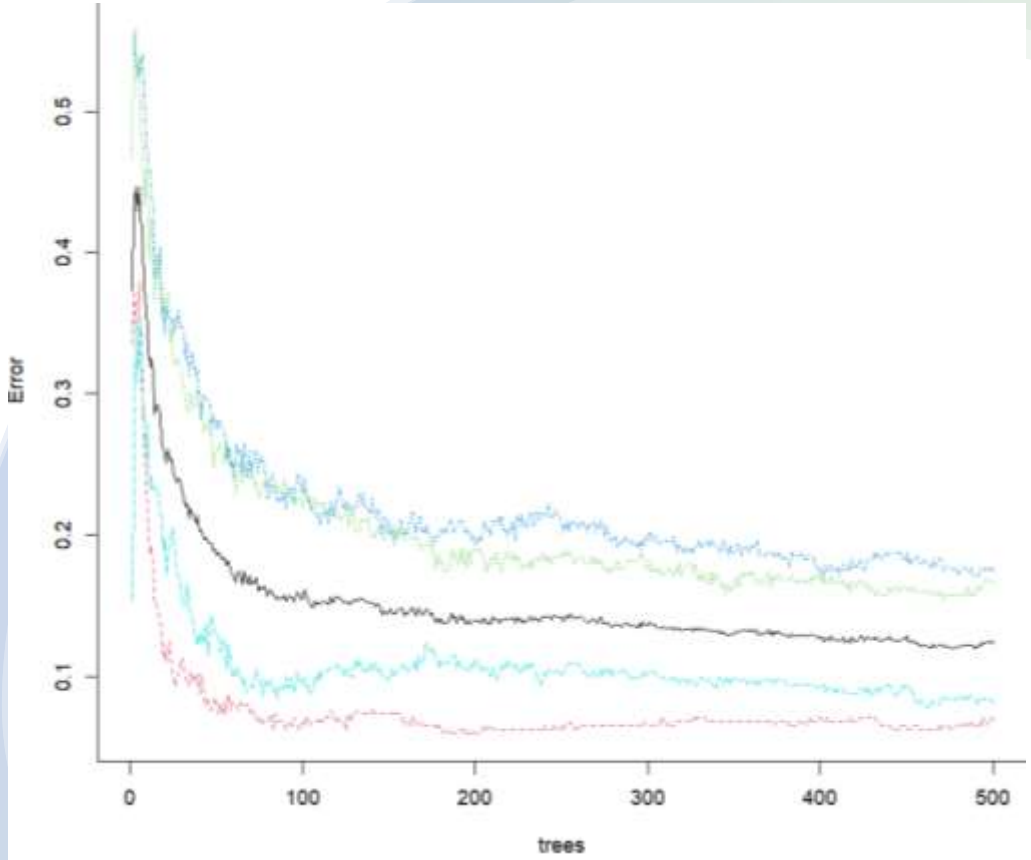



```
> accuracy_tree = sum(diag(tab2))/length(test_data$price_range)
> accuracy_tree
[1] 0.8809107
```

Random Forest

Come possiamo vedere dal grafico che plotta l'importanza delle variabili utilizzate nella random forest, la ram surclassa tutte le altre rendendole pressochè inutili.






```
> tab3
             test_data$price_range
randomf_pred   0   1   2   3
           0 139   9   0   0
           1   7 118  17   0
           2   0  10 114   8
           3   0   0   7 142
> accuracy_rf
[1] 0.8984238
```

Accuracy elevata anche con questo algoritmo.

Conclusioni

- L’algoritmo che meglio classifica le nostre osservazioni è l’analisi discriminante 95,45%.
- Le variabili che più influiscono sul price range sono: ram, capienza batteria, larghezza e altezza schermo (elevata accuratezza anche considerando solamente la ram).
- Nonostante non fossero presenti le quattro classi anche con i metodi non supervisionati siamo riusciti ad ottenere un accuracy del 75%.

GRAZIE PER L’ATTENZIONE