

Proyecto 2. Entrega 7. RNA

Pablo Daniel Barillas Moreno, Carné No. 22193
Mathew Cordero Aquino, Carné No. 22982

2025-03-14

Enlace al Repositorio del proyecto 2 - Entrega 7 de minería de datos del Grupo #1

Repositorio en GitHub

0. Descargue los conjuntos de datos.

Para este punto, ya se ha realizado el proceso para descargar del sitio web: House Prices - Advanced Regression Techniques, la data de entrenamiento y la data de prueba, ambos extraídos desde la carpeta “house_prices_data/” en data frames llamados train_data (data de entrenamiento) y test_data (data de prueba), sin convertir automáticamente las variables categóricas en factores (stringsAsFactors = FALSE). Luego, se realiza una inspección inicial de train_data mediante tres funciones: head(train_data), que muestra las primeras filas del dataset; str(train_data), que despliega la estructura del data frame, incluyendo el tipo de cada variable; y summary(train_data), que proporciona un resumen estadístico de las variables numéricas y una descripción general de las categóricas.

```
train_data <- read.csv("train_set.csv", stringsAsFactors = FALSE)
test_data <- read.csv("test_set.csv", stringsAsFactors = FALSE)

head(train_data)    # Muestra las primeras filas
```

```
##   Id MSSubClass MSZoning LotFrontage LotArea Street LotShape LandContour
## 1  3           60      RL           68   11250   Pave      IR1         Lvl
## 2  5           60      RL           84   14260   Pave      IR1         Lvl
## 3  7           20      RL           75   10084   Pave      Reg          Lvl
## 4  8           60      RL           69   10382   Pave      IR1         Lvl
## 5 10          190      RL           50    7420   Pave      Reg          Lvl
## 6 13           20      RL           69   12968   Pave      IR2         Lvl
##   Utilities LotConfig LandSlope Neighborhood Condition1 Condition2 BldgType
## 1   AllPub   Inside    Gtl      CollgCr      Norm      Norm      1Fam
## 2   AllPub    FR2      Gtl      NoRidge      Norm      Norm      1Fam
## 3   AllPub   Inside    Gtl      Somerst      Norm      Norm      1Fam
## 4   AllPub   Corner    Gtl      NWAmes      PosN      Norm      1Fam
## 5   AllPub   Corner    Gtl      BrkSide     Artery     Artery     2fmCon
## 6   AllPub   Inside    Gtl      Sawyer      Norm      Norm      1Fam
##   HouseStyle OverallQual OverallCond YearBuilt YearRemodAdd RoofStyle RoofMatl
## 1     2Story           7           5     2001         2002     Gable   CompShg
## 2     2Story           8           5     2000         2000     Gable   CompShg
## 3     1Story           8           5     2004         2005     Gable   CompShg
## 4     2Story           7           6     1973         1973     Gable   CompShg
```

## 5	1.5Unf	5	6	1939	1950	Gable	CompShg
## 6	1Story	5	6	1962	1962	Hip	CompShg
##	Exterior1st	Exterior2nd	MasVnrType	MasVnrArea	ExterQual	ExterCond	Foundation
## 1	VinylSd	VinylSd	BrkFace	162	Gd	TA	PConc
## 2	VinylSd	VinylSd	BrkFace	350	Gd	TA	PConc
## 3	VinylSd	VinylSd	Stone	186	Gd	TA	PConc
## 4	HdBoard	HdBoard	Stone	240	TA	TA	CBlock
## 5	MetalSd	MetalSd	None	0	TA	TA	BrkTil
## 6	HdBoard	Plywood	None	0	TA	TA	CBlock
##	BsmtQual	BsmtCond	BsmtExposure	BsmtFinType1	BsmtFinSF1	BsmtFinType2	
## 1	Gd	TA	Mn	GLQ	486	Unf	
## 2	Gd	TA	Av	GLQ	655	Unf	
## 3	Ex	TA	Av	GLQ	1369	Unf	
## 4	Gd	TA	Mn	ALQ	859	BLQ	
## 5	TA	TA	No	GLQ	851	Unf	
## 6	TA	TA	No	ALQ	737	Unf	
##	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	Heating	HeatingQC	CentralAir	Electrical
## 1	0	434	920	GasA	Ex	Y	SBrkr
## 2	0	490	1145	GasA	Ex	Y	SBrkr
## 3	0	317	1686	GasA	Ex	Y	SBrkr
## 4	32	216	1107	GasA	Ex	Y	SBrkr
## 5	0	140	991	GasA	Ex	Y	SBrkr
## 6	0	175	912	GasA	TA	Y	SBrkr
##	X1stFlrSF	X2ndFlrSF	LowQualFinSF	GrLivArea	BsmtFullBath	BsmtHalfBath	FullBath
## 1	920	866	0	1786	1	0	2
## 2	1145	1053	0	2198	1	0	2
## 3	1694	0	0	1694	1	0	2
## 4	1107	983	0	2090	1	0	2
## 5	1077	0	0	1077	1	0	1
## 6	912	0	0	912	1	0	1
##	HalfBath	BedroomAbvGr	KitchenAbvGr	KitchenQual	TotRmsAbvGrd	Functional	
## 1	1	3	1	Gd	6	Typ	
## 2	1	4	1	Gd	9	Typ	
## 3	0	3	1	Gd	7	Typ	
## 4	1	3	1	TA	7	Typ	
## 5	0	2	2	TA	5	Typ	
## 6	0	2	1	TA	4	Typ	
##	Fireplaces	FireplaceQu	GarageType	GarageYrBlt	GarageFinish	GarageCars	
## 1	1	TA	Attchd	2001	RFn	2	
## 2	1	TA	Attchd	2000	RFn	3	
## 3	1	Gd	Attchd	2004	RFn	2	
## 4	2	TA	Attchd	1973	RFn	2	
## 5	2	TA	Attchd	1939	RFn	1	
## 6	0	None	Detchd	1962	Unf	1	
##	GarageArea	GarageQual	GarageCond	PavedDrive	WoodDeckSF	OpenPorchSF	
## 1	608	TA	TA	Y	0	42	
## 2	836	TA	TA	Y	192	84	
## 3	636	TA	TA	Y	255	57	
## 4	484	TA	TA	Y	235	204	
## 5	205	Gd	TA	Y	0	4	
## 6	352	TA	TA	Y	140	0	
##	EnclosedPorch	X3SsnPorch	ScreenPorch	PoolArea	MiscVal	MoSold	YrSold SaleType
## 1	0	0	0	0	0	9	2008 WD
## 2	0	0	0	0	0	12	2008 WD

```
## 3      0      0      0      0      0      8      2007      WD
## 4     228      0      0      0      0     350     11      2009      WD
## 5      0      0      0      0      0      1      2008      WD
## 6      0      0     176      0      0      9      2008      WD
##   SaleCondition SalePrice LogSalePrice QualityGroup SizeGroup Cluster Age
## 1      Normal    223500    12.31717      Media    Mediana      2    7
## 2      Normal    250000    12.42922      Alta     Grande      1    8
## 3      Normal    307000    12.63460      Alta     Mediana      1    3
## 4      Normal    200000    12.20607      Media    Grande      2   36
## 5      Normal    118000    11.67844      Media    Mediana      3   69
## 6      Normal    144000    11.87757      Media    Pequeña     3   46
##   Qual_LivArea SalePriceCat
## 1      12502      cara
## 2      17584      cara
## 3      13552      cara
## 4      14630      cara
## 5       5385     barata
## 6       4560      media
```

```
str(train_data)      # Muestra la estructura del dataset
```

```
## 'data.frame':      937 obs. of  84 variables:
## $ Id      : int  3 5 7 8 10 13 15 18 19 20 ...
## $ MSSubClass : int  60 60 20 60 190 20 20 90 20 20 ...
## $ MSZoning  : chr  "RL" "RL" "RL" "RL" ...
## $ LotFrontage : int  68 84 75 69 50 69 69 72 66 70 ...
## $ LotArea    : int  11250 14260 10084 10382 7420 12968 10920 10791 13695 7560 ...
## $ Street     : chr  "Pave" "Pave" "Pave" "Pave" ...
## $ LotShape   : chr  "IR1" "IR1" "Reg" "IR1" ...
## $ LandContour : chr  "Lvl" "Lvl" "Lvl" "Lvl" ...
## $ Utilities  : chr  "AllPub" "AllPub" "AllPub" "AllPub" ...
## $ LotConfig  : chr  "Inside" "FR2" "Inside" "Corner" ...
## $ LandSlope  : chr  "Gtl" "Gtl" "Gtl" "Gtl" ...
## $ Neighborhood : chr  "CollgCr" "NoRidge" "Somerst" "NWAmes" ...
## $ Condition1 : chr  "Norm" "Norm" "Norm" "PosN" ...
## $ Condition2 : chr  "Norm" "Norm" "Norm" "Norm" ...
## $ BldgType    : chr  "1Fam" "1Fam" "1Fam" "1Fam" ...
## $ HouseStyle  : chr  "2Story" "2Story" "1Story" "2Story" ...
## $ OverallQual : int  7 8 8 7 5 5 6 4 5 5 ...
## $ OverallCond : int  5 5 5 6 6 6 5 5 5 6 ...
## $ YearBuilt   : int  2001 2000 2004 1973 1939 1962 1960 1967 2004 1958 ...
## $ YearRemodAdd : int  2002 2000 2005 1973 1950 1962 1960 1967 2004 1965 ...
## $ RoofStyle   : chr  "Gable" "Gable" "Gable" "Gable" ...
## $ RoofMatl    : chr  "CompShg" "CompShg" "CompShg" "CompShg" ...
## $ Exterior1st : chr  "VinylSd" "VinylSd" "VinylSd" "HdBoard" ...
## $ Exterior2nd : chr  "VinylSd" "VinylSd" "VinylSd" "HdBoard" ...
## $ MasVnrType  : chr  "BrkFace" "BrkFace" "Stone" "Stone" ...
## $ MasVnrArea  : int  162 350 186 240 0 0 212 0 0 0 ...
## $ ExterQual   : chr  "Gd" "Gd" "Gd" "TA" ...
## $ ExterCond   : chr  "TA" "TA" "TA" "TA" ...
## $ Foundation  : chr  "PConc" "PConc" "PConc" "CBlock" ...
## $ BsmtQual    : chr  "Gd" "Gd" "Ex" "Gd" ...
## $ BsmtCond    : chr  "TA" "TA" "TA" "TA" ...
```

```

## $ BsmtExposure : chr "Mn" "Av" "Av" "Mn" ...
## $ BsmtFinType1 : chr "GLQ" "GLQ" "GLQ" "ALQ" ...
## $ BsmtFinSF1 : int 486 655 1369 859 851 737 733 0 646 504 ...
## $ BsmtFinType2 : chr "Unf" "Unf" "Unf" "BLQ" ...
## $ BsmtFinSF2 : int 0 0 0 32 0 0 0 0 0 0 ...
## $ BsmtUnfSF : int 434 490 317 216 140 175 520 0 468 525 ...
## $ TotalBsmtSF : int 920 1145 1686 1107 991 912 1253 0 1114 1029 ...
## $ Heating : chr "GasA" "GasA" "GasA" "GasA" ...
## $ HeatingQC : chr "Ex" "Ex" "Ex" "Ex" ...
## $ CentralAir : chr "Y" "Y" "Y" "Y" ...
## $ Electrical : chr "SBrkr" "SBrkr" "SBrkr" "SBrkr" ...
## $ X1stFlrSF : int 920 1145 1694 1107 1077 912 1253 1296 1114 1339 ...
## $ X2ndFlrSF : int 866 1053 0 983 0 0 0 0 0 0 ...
## $ LowQualFinSF : int 0 0 0 0 0 0 0 0 0 0 ...
## $ GrLivArea : int 1786 2198 1694 2090 1077 912 1253 1296 1114 1339 ...
## $ BsmtFullBath : int 1 1 1 1 1 1 1 0 1 0 ...
## $ BsmtHalfBath : int 0 0 0 0 0 0 0 0 0 0 ...
## $ FullBath : int 2 2 2 2 1 1 1 2 1 1 ...
## $ HalfBath : int 1 1 0 1 0 0 1 0 1 0 ...
## $ BedroomAbvGr : int 3 4 3 3 2 2 2 2 3 3 ...
## $ KitchenAbvGr : int 1 1 1 1 2 1 1 2 1 1 ...
## $ KitchenQual : chr "Gd" "Gd" "Gd" "TA" ...
## $ TotRmsAbvGrd : int 6 9 7 7 5 4 5 6 6 6 ...
## $ Functional : chr "Typ" "Typ" "Typ" "Typ" ...
## $ Fireplaces : int 1 1 1 2 2 0 1 0 0 0 ...
## $ FireplaceQu : chr "TA" "TA" "Gd" "TA" ...
## $ GarageType : chr "Attchd" "Attchd" "Attchd" "Attchd" ...
## $ GarageYrBlt : int 2001 2000 2004 1973 1939 1962 1960 1967 2004 1958 ...
## $ GarageFinish : chr "RFn" "RFn" "RFn" "RFn" ...
## $ GarageCars : int 2 3 2 2 1 1 1 2 2 1 ...
## $ GarageArea : int 608 836 636 484 205 352 352 516 576 294 ...
## $ GarageQual : chr "TA" "TA" "TA" "TA" ...
## $ GarageCond : chr "TA" "TA" "TA" "TA" ...
## $ PavedDrive : chr "Y" "Y" "Y" "Y" ...
## $ WoodDeckSF : int 0 192 255 235 0 140 0 0 0 0 ...
## $ OpenPorchSF : int 42 84 57 204 4 0 213 0 102 0 ...
## $ EnclosedPorch : int 0 0 0 228 0 0 176 0 0 0 ...
## $ X3SsnPorch : int 0 0 0 0 0 0 0 0 0 0 ...
## $ ScreenPorch : int 0 0 0 0 0 176 0 0 0 0 ...
## $ PoolArea : int 0 0 0 0 0 0 0 0 0 0 ...
## $ MiscVal : int 0 0 0 350 0 0 0 500 0 0 ...
## $ MoSold : int 9 12 8 11 1 9 5 10 6 5 ...
## $ YrSold : int 2008 2008 2007 2009 2008 2008 2008 2006 2008 2009 ...
## $ SaleType : chr "WD" "WD" "WD" "WD" ...
## $ SaleCondition : chr "Normal" "Normal" "Normal" "Normal" ...
## $ SalePrice : num 223500 250000 307000 200000 118000 ...
## $ LogSalePrice : num 12.3 12.4 12.6 12.2 11.7 ...
## $ QualityGroup : chr "Media" "Alta" "Alta" "Media" ...
## $ SizeGroup : chr "Mediana" "Grande" "Mediana" "Grande" ...
## $ Cluster : int 2 1 1 2 3 3 3 2 3 ...
## $ Age : int 7 8 3 36 69 46 48 39 4 51 ...
## $ Qual_LivArea : int 12502 17584 13552 14630 5385 4560 7518 5184 5570 6695 ...
## $ SalePriceCat : chr "cara" "cara" "cara" "cara" ...

```

```
summary(train_data) # Resumen estadístico
```

```
##           Id           MSSubClass           MSZoning           LotFrontage
## Min.      :   3.0   Min.      : 20.00   Length:937   Min.      : 21.00
## 1st Qu.: 364.0   1st Qu.: 20.00   Class :character   1st Qu.: 60.00
## Median : 728.0   Median : 50.00   Mode  :character   Median : 69.00
## Mean    : 729.1   Mean    : 55.67                   Mean    : 69.88
## 3rd Qu.:1094.0   3rd Qu.: 70.00                   3rd Qu.: 79.00
## Max.    :1459.0   Max.    :190.00                   Max.    :313.00
##
##           LotArea           Street           LotShape           LandContour
## Min.      : 1477   Length:937   Length:937   Length:937
## 1st Qu.: 7596   Class :character   Class :character   Class :character
## Median : 9405   Mode  :character   Mode  :character   Mode  :character
## Mean    : 10234
## 3rd Qu.: 11643
## Max.    :115149
##
##           Utilities           LotConfig           LandSlope           Neighborhood
## Length:937   Length:937   Length:937   Length:937
## Class :character   Class :character   Class :character   Class :character
## Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##           Condition1           Condition2           BldgType           HouseStyle
## Length:937   Length:937   Length:937   Length:937
## Class :character   Class :character   Class :character   Class :character
## Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##           OverallQual           OverallCond           YearBuilt           YearRemodAdd
## Min.      : 1.000   Min.      :1.000   Min.      :1875   Min.      :1950
## 1st Qu.: 5.000   1st Qu.:5.000   1st Qu.:1953   1st Qu.:1967
## Median : 6.000   Median :5.000   Median :1973   Median :1994
## Mean    : 6.079   Mean    :5.606   Mean    :1971   Mean    :1985
## 3rd Qu.: 7.000   3rd Qu.:6.000   3rd Qu.:2000   3rd Qu.:2003
## Max.    :10.000   Max.      :9.000   Max.      :2009   Max.      :2010
##
##           RoofStyle           RoofMatl           Exterior1st           Exterior2nd
## Length:937   Length:937   Length:937   Length:937
## Class :character   Class :character   Class :character   Class :character
## Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##           MasVnrType           MasVnrArea           ExterQual           ExterCond
## Length:937   Min.      : 0.00   Length:937   Length:937
## Class :character   1st Qu.: 0.00   Class :character   Class :character
```

```

## Mode :character Median : 0.00 Mode :character Mode :character
## Mean : 99.48
## 3rd Qu.: 157.75
## Max. :1600.00
## NA's :7
## Foundation BsmtQual BsmtCond BsmtExposure
## Length:937 Length:937 Length:937 Length:937
## Class :character Class :character Class :character Class :character
## Mode :character Mode :character Mode :character Mode :character
##
##
##
## BsmtFinType1 BsmtFinSF1 BsmtFinType2 BsmtFinSF2
## Length:937 Min. : 0 Length:937 Min. : 0.0
## Class :character 1st Qu.: 0 Class :character 1st Qu.: 0.0
## Mode :character Median : 374 Mode :character Median : 0.0
## Mean : 441 Mean : 50.6
## 3rd Qu.: 713 3rd Qu.: 0.0
## Max. :5644 Max. :1474.0
##
## BsmtUnfSF TotalBsmtSF Heating HeatingQC
## Min. : 0.0 Min. : 0 Length:937 Length:937
## 1st Qu.: 218.0 1st Qu.: 798 Class :character Class :character
## Median : 479.0 Median : 990 Mode :character Mode :character
## Mean : 570.1 Mean :1062
## 3rd Qu.: 813.0 3rd Qu.:1278
## Max. :2336.0 Max. :6110
##
## CentralAir Electrical X1stFlrSF X2ndFlrSF
## Length:937 Length:937 Min. : 438 Min. : 0.0
## Class :character Class :character 1st Qu.: 894 1st Qu.: 0.0
## Mode :character Mode :character Median :1085 Median : 0.0
## Mean :1169 Mean : 341.9
## 3rd Qu.:1390 3rd Qu.: 728.0
## Max. :4692 Max. :2065.0
##
## LowQualFinSF GrLivArea BsmtFullBath BsmtHalfBath
## Min. : 0.000 Min. : 438 Min. :0.0000 Min. :0.00000
## 1st Qu.: 0.000 1st Qu.:1124 1st Qu.:0.0000 1st Qu.:0.00000
## Median : 0.000 Median :1471 Median :0.0000 Median :0.00000
## Mean : 3.289 Mean :1515 Mean :0.4312 Mean :0.05229
## 3rd Qu.: 0.000 3rd Qu.:1795 3rd Qu.:1.0000 3rd Qu.:0.00000
## Max. :572.000 Max. :5642 Max. :3.0000 Max. :2.00000
##
## FullBath HalfBath BedroomAbvGr KitchenAbvGr
## Min. :0.000 Min. :0.0000 Min. :0.000 Min. :0.000
## 1st Qu.:1.000 1st Qu.:0.0000 1st Qu.:2.000 1st Qu.:1.000
## Median :2.000 Median :0.0000 Median :3.000 Median :1.000
## Mean :1.577 Mean :0.3831 Mean :2.876 Mean :1.052
## 3rd Qu.:2.000 3rd Qu.:1.0000 3rd Qu.:3.000 3rd Qu.:1.000
## Max. :3.000 Max. :2.0000 Max. :6.000 Max. :3.000
##
## KitchenQual TotRmsAbvGrd Functional Fireplaces

```

```

## Length:937      Min.    : 3.00   Length:937      Min.    :0.0000
## Class :character 1st Qu.: 5.00   Class :character 1st Qu.:0.0000
## Mode :character  Median : 6.00   Mode :character  Median :1.0000
##                  Mean     : 6.53   Mean     :0.6009
##                  3rd Qu.: 7.00   3rd Qu.:1.0000
##                  Max.     :12.00   Max.     :3.0000
##
## FireplaceQu      GarageType      GarageYrBlt      GarageFinish
## Length:937      Length:937      Min.    :1900   Length:937
## Class :character Class :character 1st Qu.:1962   Class :character
## Mode :character Mode :character Median :1979   Mode :character
##                  Mean     :1979
##                  3rd Qu.:2001
##                  Max.     :2010
##                  NA's    :54
## GarageCars      GarageArea      GarageQual      GarageCond
## Min.    :0.000   Min.    : 0.0   Length:937      Length:937
## 1st Qu.:1.000   1st Qu.: 318.0 Class :character Class :character
## Median :2.000   Median : 478.0 Mode :character Mode :character
## Mean    :1.756   Mean    : 470.9
## 3rd Qu.:2.000   3rd Qu.: 576.0
## Max.    :4.000   Max.    :1418.0
##
## PavedDrive      WoodDeckSF      OpenPorchSF      EnclosedPorch
## Length:937      Min.    : 0.00   Min.    : 0.0   Min.    : 0.00
## Class :character 1st Qu.: 0.00   1st Qu.: 0.0   1st Qu.: 0.00
## Mode :character Median : 0.00   Median : 25.0   Median : 0.00
##                  Mean    : 93.47   Mean    : 46.6   Mean    : 23.55
##                  3rd Qu.:168.00   3rd Qu.: 69.0   3rd Qu.: 0.00
##                  Max.    :857.00   Max.    :502.0   Max.    :386.00
##
## X3SsnPorch      ScreenPorch      PoolArea      MiscVal
## Min.    : 0.000   Min.    : 0.00   Min.    : 0.000   Min.    : 0.00
## 1st Qu.: 0.000   1st Qu.: 0.00   1st Qu.: 0.000   1st Qu.: 0.00
## Median : 0.000   Median : 0.00   Median : 0.000   Median : 0.00
## Mean    : 2.995   Mean    : 15.94   Mean    : 3.138   Mean    : 35.15
## 3rd Qu.: 0.000   3rd Qu.: 0.00   3rd Qu.: 0.000   3rd Qu.: 0.00
## Max.    :407.000   Max.    :440.00   Max.    :738.000   Max.    :15500.00
##
## MoSold          YrSold          SaleType          SaleCondition
## Min.    : 1.000   Min.    :2006   Length:937      Length:937
## 1st Qu.: 5.000   1st Qu.:2007   Class :character Class :character
## Median : 6.000   Median :2008   Mode :character Mode :character
## Mean    : 6.383   Mean    :2008
## 3rd Qu.: 8.000   3rd Qu.:2009
## Max.    :12.000   Max.    :2010
##
## SalePrice      LogSalePrice      QualityGroup      SizeGroup
## Min.    : 35311   Min.    :10.47   Length:937      Length:937
## 1st Qu.:130000   1st Qu.:11.78   Class :character Class :character
## Median :163000   Median :12.00   Mode :character Mode :character
## Mean    :180334   Mean    :12.02
## 3rd Qu.:214000   3rd Qu.:12.27
## Max.    :745000   Max.    :13.52

```

```
##
##      Cluster      Age      Qual_LivArea  SalePriceCat
## Min.   :1.000   Min.    : 0.00   Min.     : 876   Length:937
## 1st Qu.:2.000   1st Qu.: 8.00   1st Qu.: 5720   Class :character
## Median :2.000   Median : 35.00   Median : 8806   Mode  :character
## Mean   :2.187   Mean    : 36.78   Mean     : 9649
## 3rd Qu.:3.000   3rd Qu.: 55.00   3rd Qu.:12327
## Max.   :3.000   Max.    :135.00   Max.     :56420
##
```

1. Use los mismos conjuntos de entrenamiento y prueba que utilizó en las entregas anteriores.

```
# Cargar librerías necesarias
library(dplyr)
```

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

```
##
## Adjuntando el paquete: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##      filter, lag
```

```
## The following objects are masked from 'package:base':
##
##      intersect, setdiff, setequal, union
```

```
library(readr)
```

```
# Fijar semilla para reproducibilidad
set.seed(123)
```

```
# Cargar los conjuntos de datos previamente usados
train <- read_csv("train_set.csv")
```

```
## Rows: 937 Columns: 84
```

```
## -- Column specification -----
## Delimiter: ","
## chr (42): MSZoning, Street, LotShape, LandContour, Utilities, LotConfig, Lan...
## dbl (42): Id, MSSubClass, LotFrontage, LotArea, OverallQual, OverallCond, Ye...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
test <- read_csv("test_set.csv")
```



```
## Rows: 232 Columns: 84
## -- Column specification -----
## Delimiter: ","
## chr (42): MSZoning, Street, LotShape, LandContour, Utilities, LotConfig, Lan...
## dbl (42): Id, MSSubClass, LotFrontage, LotArea, OverallQual, OverallCond, Ye...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# Verificar estructura general y dimensiones
cat("Observaciones en train:", nrow(train), "\n")
```

```
## Observaciones en train: 937
```

```
cat("Observaciones en test:", nrow(test), "\n")
```

```
## Observaciones en test: 232
```

```
# Asegurar que la variable de salida es factor
train$SalePriceCat <- as.factor(train$SalePriceCat)
test$SalePriceCat <- as.factor(test$SalePriceCat)
```

```
# Verificar primeras observaciones de interés
cat("\nPrimeras observaciones:\n")
```

```
##
## Primeras observaciones:
```

```
print(head(train[, c("SalePrice", "SalePriceCat")]))
```

```
## # A tibble: 6 x 2
##   SalePrice SalePriceCat
##   <dbl> <fct>
## 1    223500 cara
## 2    250000 cara
## 3    307000 cara
## 4    200000 cara
## 5    118000 barata
## 6    144000 media
```

```
# Frecuencia de clases en el conjunto de entrenamiento
cat("\nDistribución de SalePriceCat en train:\n")
```

```
##
## Distribución de SalePriceCat en train:
```

```
print(table(train$SalePriceCat))
```

```
##
## barata   cara   media
##    313    312    312
```

```
# Confirmar estructura de la variable objetivo
cat("\nEstructura de la variable SalePriceCat:\n")
```

```
##
## Estructura de la variable SalePriceCat:
```

```
str(train$SalePriceCat)
```

```
## Factor w/ 3 levels "barata","cara",...: 2 2 2 2 1 3 3 1 3 1 ...
```

Exploración inicial de los datos

Al cargar los conjuntos de datos `train_set.csv` y `test_set.csv`, se obtuvo una salida descriptiva automática proporcionada por la función `read_csv()` del paquete `readr`. Esta salida indica que ambos conjuntos tienen exactamente **84 columnas**, divididas en:

- **42 columnas de tipo carácter (chr)**, correspondientes a variables **categorías**, como `MSZoning`, `Neighborhood`, `BldgType`, entre otras.
- **42 columnas de tipo numérico (dbl)**, como `LotArea`, `OverallQual`, `GrLivArea`, `SalePrice`, etc.

También se observa que:

- El conjunto de entrenamiento contiene **937 observaciones**.
- El conjunto de prueba contiene **232 observaciones**.

Además, se validó que la variable categórica objetivo `SalePriceCat` (con niveles: `barata`, `media`, `cara`) esté correctamente codificada como factor. La distribución de clases en el conjunto de entrenamiento es **perfectamente balanceada**, con **313 observaciones para “barata”**, **312 para “media”**, y **312 para “cara”**.

Finalmente, se visualizaron las primeras observaciones para comprobar que los valores de `SalePrice` y su categoría `SalePriceCat` coinciden correctamente:

SalePrice	SalePriceCat
223500	cara
250000	cara
307000	cara
200000	cara
118000	barata
144000	media

Conclusiones:

- **Consistencia asegurada:** Se utilizan los mismos conjuntos que en entregas anteriores, lo que garantiza **reproducibilidad** en los resultados del proyecto.

- **Balance en clases:** La distribución equitativa entre **barata**, **media** y **cara** es ideal para aplicar modelos de clasificación supervisada sin necesidad de técnicas de rebalanceo.
- **Preparación adecuada:** La conversión de la variable **SalePriceCat** a **factor** fue exitosa, lo cual es un requisito esencial para los algoritmos de redes neuronales multiclase en **caret**.
- **Calidad de los datos:** No se detectaron errores de carga ni inconsistencias evidentes en las primeras inspecciones de los datos.

Por lo tanto, el conjunto de datos está listo para proceder a su transformación y entrenamiento de un modelo RNA en la siguiente sección.

2 Seleccione como variable respuesta la que creó con las categorías del precio de la casa.

```
# Definir variable respuesta (target)
y_train <- train$SalePriceCat
y_test  <- test$SalePriceCat

# Verificar que sean factores
cat("¿y_train es factor?:", is.factor(y_train), "\n")
```

```
## ¿y_train es factor?: TRUE
```

```
cat("¿y_test es factor?:", is.factor(y_test), "\n")
```

```
## ¿y_test es factor?: TRUE
```

```
# Verificar si hay valores NA
cat("¿Hay NA en y_train?:", any(is.na(y_train)), "\n")
```

```
## ¿Hay NA en y_train?: FALSE
```

```
cat("¿Hay NA en y_test?:", any(is.na(y_test)), "\n")
```

```
## ¿Hay NA en y_test?: FALSE
```

```
# Verificar distribución de clases
cat("\nDistribución de clases en y_train:\n")
```

```
##
```

```
## Distribución de clases en y_train:
```

```
print(table(y_train))
```

```
## y_train
## barata  cara  media
##    313    312    312
```

```
cat("\nDistribución de clases en y_test:\n")
```

```
##  
## Distribución de clases en y_test:
```

```
print(table(y_test))
```

```
## y_test  
## barata   cara   media  
##      78     77     77
```

Análisis del Target: SalePriceCat

Tras seleccionar como variable de salida la columna categórica **SalePriceCat**, se realizó una revisión detallada de su estructura y contenido en los conjuntos de entrenamiento (**train**) y prueba (**test**).

Verificaciones realizadas - Se confirmó que tanto **y_train** como **y_test** están correctamente codificadas como factores (**factor**), lo cual es indispensable para el entrenamiento de modelos de clasificación. - No se detectaron valores faltantes (**NA**) en ninguno de los conjuntos. Esto asegura que no será necesario aplicar técnicas de imputación para esta variable específica.

Distribución de clases - En **train**, las clases se encuentran perfectamente balanceadas:

```
- barata: 313  
- media: 312  
- cara: 312
```

- En **test**, la proporción también se mantiene muy equilibrada:
 - barata: 78
 - media: 77
 - cara: 77

Este balance es ideal para entrenar redes neuronales artificiales, ya que evita sesgos hacia una clase dominante y permite que el modelo aprenda de manera equitativa las características de cada categoría.

Conclusiones

- La variable respuesta **SalePriceCat** está correctamente definida y lista para ser utilizada en la red neuronal.
- El balance entre clases tanto en **train** como en **test** garantiza condiciones óptimas para el aprendizaje supervisado.
- No será necesario aplicar técnicas de rebalanceo ni limpieza adicional sobre esta variable.

3. Genere dos modelos de redes neuronales que sean capaz de clasificar usando la variable respuesta que categoriza las casas en baratas, medias y caras. Estos modelos deben tener diferentes topologías y funciones de activación.

Modelo 1 – Red neuronal simple con nnet (función logística)

```

# Librerías necesarias
library(nnet)
library(caret)

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

## Cargando paquete requerido: ggplot2

## Warning: package 'ggplot2' was built under R version 4.4.3

## Cargando paquete requerido: lattice

## Warning: package 'lattice' was built under R version 4.4.3

library(dplyr)

set.seed(123)

# Variable respuesta
y_train <- train$SalePriceCat
y_test  <- test$SalePriceCat

# Selección de solo 5 predictores relevantes
x_train <- train %>% select(OverallQual, GrLivArea, GarageCars, TotalBsmtSF, YearBuilt)
x_test  <- test %>% select(OverallQual, GrLivArea, GarageCars, TotalBsmtSF, YearBuilt)

# Escalado + imputación
preproc <- preProcess(x_train, method = c("medianImpute", "center", "scale"))
x_train_rna <- predict(preproc, x_train)
x_test_rna  <- predict(preproc, x_test)

# Red neuronal simple con menos neuronas
modelo_nnet <- nnet::nnet(
  x = x_train_rna,
  y = class.ind(y_train),
  size = 3,          # número reducido de neuronas
  softmax = TRUE,
  maxit = 200,
  trace = FALSE
)

# Predicción
pred_nnet_raw <- predict(modelo_nnet, x_test_rna, type = "raw")
pred_nnet <- factor(
  colnames(pred_nnet_raw)[apply(pred_nnet_raw, 1, which.max)],
  levels = levels(y_test)
)

# Matriz de confusión
confusionMatrix(pred_nnet, y_test)

```

```

## Confusion Matrix and Statistics
##
##           Reference
## Prediction barata cara media
##   barata      62    0    3
##   cara         0   69    6
##   media        16    8   68
##
## Overall Statistics
##
##           Accuracy : 0.8578
##           95% CI : (0.8061, 0.9)
##   No Information Rate : 0.3362
##   P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.7867
##
## Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##           Class: barata Class: cara Class: media
## Sensitivity           0.7949      0.8961      0.8831
## Specificity           0.9805      0.9613      0.8452
## Pos Pred Value        0.9538      0.9200      0.7391
## Neg Pred Value        0.9042      0.9490      0.9357
## Prevalence            0.3362      0.3319      0.3319
## Detection Rate        0.2672      0.2974      0.2931
## Detection Prevalence  0.2802      0.3233      0.3966
## Balanced Accuracy      0.8877      0.9287      0.8641

```

Resultados del Modelo de Red Neuronal (nnet)

El modelo de red neuronal multicapa fue entrenado usando la función `nnet`, con una arquitectura simple (una sola capa oculta con pocas neuronas). Los resultados muestran un buen rendimiento en la tarea de clasificación multiclase (`barata`, `media`, `cara`).

Estadísticas Generales:

- **Accuracy general:** 85.78%
- **Kappa:** 0.7867 → Indica un **fuerte nivel de acuerdo** entre las predicciones del modelo y las clases reales, controlando por el azar.
- **P-Valor [Acc > NIR]:** < 2.2e-16 → El modelo tiene un rendimiento significativamente superior al de una predicción aleatoria.

Análisis de Rendimiento por Clase

1. Clase `barata`:

- **Sensibilidad (Recall):** 0.7949 → Aproximadamente el 79.5% de las casas realmente baratas fueron correctamente clasificadas.
- **Especificidad:** 0.9805 → El modelo distingue muy bien las casas que **no** son baratas.
- **Precision (Valor Positivo Predictivo):** 0.9538 → Las predicciones de “barata” fueron muy acertadas.

- **Balanced Accuracy:** 0.8877 → Buen equilibrio entre sensibilidad y especificidad.

El modelo tiende a confundir algunas viviendas baratas como **media**, pero acierta la mayoría de veces cuando predice “barata”.

2. Clase **cara**:

- **Sensibilidad:** 0.8961 → Excelente capacidad de detección para casas caras.
- **Especificidad:** 0.9613 → Alto nivel de discriminación contra las clases **barata** y **media**.
- **Precision:** 0.9200 → Las predicciones de “cara” fueron correctas en el 92% de los casos.
- **Balanced Accuracy:** 0.9287 → Es la clase con **mejor desempeño general**.

El modelo es muy preciso en clasificar casas caras, lo cual es relevante para negocios inmobiliarios.

3. Clase **media**:

- **Sensibilidad:** 0.8831 → El modelo identifica correctamente el 88.3% de las casas medias.
- **Especificidad:** 0.8452 → Se confunden algunas casas **barata** o **cara** como **media**.
- **Precision:** 0.7391 → De todas las predicciones que hizo como **media**, solo el 73.9% eran correctas.
- **Balanced Accuracy:** 0.8641 → Buen resultado general, aunque menos preciso que las otras clases.

Esta clase fue la más difícil de predecir con certeza, posiblemente por similitudes estructurales con **barata** y **cara**.

Conclusiones Generales

- El modelo entrenado con **nnet** logró una **precisión global superior al 85%**, lo que es excelente considerando que se trata de una clasificación multiclase con tres etiquetas.
- **Todas las clases presentan un buen desempeño**, pero el modelo es especialmente fuerte clasificando viviendas **caras**.
- La clase **media** fue la **más propensa a errores de clasificación**, lo cual sugiere que podría beneficiarse de un ajuste más fino o de más neuronas ocultas.
- **No se observó sobreajuste**, ya que los errores fueron razonablemente distribuidos y las métricas son consistentes entre clases.
- Este modelo es una **buena primera aproximación**, y sirve como base para comparar con arquitecturas más complejas (como redes profundas con **neuralnet**, que se usarán en el segundo modelo).

Modelo 2 – Red neuronal profunda con **neuralnet** (1 capa oculta, 4 neuronas)

```
# Librerías necesarias
library(neuralnet)
```

```
## Warning: package 'neuralnet' was built under R version 4.4.3
```

```
##
```

```
## Adjuntando el paquete: 'neuralnet'
```

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

```
##
```

```
## compute
```

```

library(nnet)          # Para class.ind
library(caret)
library(dplyr)

set.seed(123)

# Variable respuesta
y_train <- train$SalePriceCat
y_test  <- test$SalePriceCat

# Selección de solo 5 predictores relevantes
x_train <- train %>% select(OverallQual, GrLivArea, GarageCars, TotalBsmtSF, YearBuilt)
x_test  <- test %>% select(OverallQual, GrLivArea, GarageCars, TotalBsmtSF, YearBuilt)

# Escalado + imputación
preproc <- preProcess(x_train, method = c("medianImpute", "center", "scale"))
x_train_scaled <- predict(preproc, x_train)
x_test_scaled  <- predict(preproc, x_test)

# Codificación de la variable categórica a dummy
y_train_dummy <- class.ind(y_train)
colnames(y_train_dummy) <- levels(y_train)

# Unir entradas y salidas en un solo dataset
train_nn <- cbind(as.data.frame(y_train_dummy), x_train_scaled)

# Fórmula dinámica para clasificación multiclase
formula_nn <- as.formula(paste(
  paste(colnames(y_train_dummy), collapse = " + "),
  "~",
  paste(colnames(x_train_scaled), collapse = " + ")
))

# Entrenamiento con topología más simple: 1 capa oculta con 4 neuronas
modelo_nn_simple <- neuralnet(
  formula = formula_nn,
  data = train_nn,
  hidden = 4,
  act.fct = "logistic",
  linear.output = FALSE,
  lifesign = "minimal",
  stepmax = 1e5,
  threshold = 0.01
)

## hidden: 4 thresh: 0.01 rep: 1/1 steps:

##      7919   error: 117.21192    time: 10.05 secs

# Predicción con el modelo entrenado
test_nn <- as.data.frame(x_test_scaled)
pred_nn_raw <- compute(modelo_nn_simple, test_nn)$net.result

```



```
# Obtener clase predicha
pred_nn <- apply(pred_nn_raw, 1, which.max)
pred_nn_factor <- factor(colnames(y_train_dummy)[pred_nn], levels = levels(y_test))

# Evaluación
confusionMatrix(pred_nn_factor, y_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction barata cara media
##   barata      70    0    8
##   cara         0   67    4
##   media        8   10   65
##
## Overall Statistics
##
##           Accuracy : 0.8707
##           95% CI : (0.8206, 0.911)
##   No Information Rate : 0.3362
##   P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.806
##
##   McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##           Class: barata Class: cara Class: media
## Sensitivity           0.8974      0.8701      0.8442
## Specificity           0.9481      0.9742      0.8839
## Pos Pred Value        0.8974      0.9437      0.7831
## Neg Pred Value        0.9481      0.9379      0.9195
## Prevalence            0.3362      0.3319      0.3319
## Detection Rate        0.3017      0.2888      0.2802
## Detection Prevalence  0.3362      0.3060      0.3578
## Balanced Accuracy      0.9227      0.9222      0.8640
```

Modelo 2 – Red neuronal con neuralnet (1 capa oculta, 4 neuronas)

El segundo modelo fue entrenado con la función **neuralnet**, utilizando una arquitectura simple: **una sola capa oculta con 4 neuronas** y activación logística. Esta red fue diseñada para evitar sobreajuste y reducir tiempos de entrenamiento, manteniendo un rendimiento competitivo.

Resultados del Modelo

- **Precisión general (Accuracy):** 87.07%
- **Kappa:** 0.806 → Muy buen acuerdo entre predicciones y valores reales.
- **Balanced Accuracy promedio:** alrededor de 90%, lo cual indica un buen desempeño equilibrado entre las clases.

Desempeño por Clase

Clase *barata*: - **Sensibilidad**: 0.8974 → Detecta correctamente el 89.7% de los casos. - **Especificidad**: 0.9481 → Excelente para identificar los que no son “barata”. - **Precision**: 0.8974 → Muy confiable cuando predice esta clase. - **Balanced Accuracy**: 0.9227

El modelo clasifica muy bien las viviendas baratas, tanto en detección como en precisión.

Clase *cara*: - **Sensibilidad**: 0.8701 - **Especificidad**: 0.9742 - **Precision**: 0.9437 - **Balanced Accuracy**: 0.9222

El modelo es **altamente preciso** con las casas caras, con excelente discriminación frente a otras clases.

Clase *media*: - **Sensibilidad**: 0.8442 - **Especificidad**: 0.8839 - **Precision**: 0.7831 - **Balanced Accuracy**: 0.8640

Aunque el modelo clasifica bastante bien la clase “media”, es donde **más errores** de predicción ocurren. Es común confundirla con “barata” o “cara”, lo cual puede explicarse por la similitud estructural entre categorías intermedias.

Conclusiones

- El **modelo neuralnet con topología liviana** logra una precisión sobresaliente en clasificación multiclase.
- **Todas las clases fueron modeladas eficazmente**, destacando la clase **cara** por su alta precisión.
- Se evitó el sobreajuste al simplificar la arquitectura y limitar la complejidad.
- El rendimiento es comparable al modelo anterior hecho con **nnet**, confirmando la solidez del preprocesamiento y la selección de variables.

Este modelo es ideal cuando se requiere **buena eficiencia y velocidad**, sin sacrificar desempeño en tareas de predicción de precios de viviendas.

Comparación de Modelos de Redes Neuronales (**nnet** vs **neuralnet**)

Métrica	Modelo 1 – nnet (Logística)	Modelo 2 – neuralnet (1 capa oculta)
Accuracy general	85.78%	87.07%
Kappa	0.7867	0.806
Balanced Accuracy		
- Clase barata	0.8877	0.9227
- Clase cara	0.9287	0.9222
- Clase media	0.8641	0.8640 (similar)
Sensibilidad global	Muy alta en clase cara	Muy alta en clase barata
Precisión por clase	media menos precisa (0.739)	media también la más confusa (0.783)
Tiempo de entrenamiento	Rápido	Moderado (~12s con 4 neuronas)

Conclusiones finales

- Ambos modelos presentan **desempeño sobresaliente**, con **precisión por encima del 85%** en clasificación multiclase (**barata**, **media**, **cara**).

- El modelo con ****neuralnet**** supera levemente en **Accuracy** y **Kappa** al modelo con **nnet**, especialmente para la clase **barata**.
- El modelo con **nnet** fue más **rápido de entrenar** y tuvo un excelente desempeño en la clase **cara**, lo cual lo vuelve ideal cuando el tiempo es un factor importante.
- La clase **media** sigue siendo la **más difícil** para ambos modelos, lo que podría deberse a su ubicación intermedia en el precio, compartiendo características con las otras dos clases.
- Ninguno de los modelos mostró signos de sobreajuste, lo cual valida la estrategia de preprocesamiento y selección de predictores.

4. Use los modelos para predecir el valor de la variable respuesta.

Predicciones con el modelo 1 - Red neuronal simple con nnet (función logística)

```
# Librerías necesarias
library(caret)
library(nnet)

# Imputar NA en test si es necesario
preproc_imput <- preprocess(x_test, method = c("medianImpute", "center", "scale"))
x_test_rna <- predict(preproc_imput, x_test)

# Asegurar que no hay NA
stopifnot(!any(is.na(x_test_rna)))

# Predicciones con el modelo 1 (nnet)
pred_nnet_raw <- predict(modelo_nnet, x_test_rna, type = "raw")
pred_nnet <- factor(
  colnames(pred_nnet_raw)[apply(pred_nnet_raw, 1, which.max)],
  levels = levels(y_test)
)

# Matriz de confusión para modelo 1
conf_nnet <- confusionMatrix(pred_nnet, y_test)
print(conf_nnet)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction barata cara media
##   barata      62    0    3
##   cara         0   67    4
##   media       16   10   70
##
## Overall Statistics
##
##           Accuracy : 0.8578
##           95% CI : (0.8061, 0.9)
##   No Information Rate : 0.3362
##   P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.7867
##
##   Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##               Class: barata Class: cara Class: media
## Sensitivity           0.7949      0.8701      0.9091
## Specificity           0.9805      0.9742      0.8323
## Pos Pred Value        0.9538      0.9437      0.7292
## Neg Pred Value        0.9042      0.9379      0.9485
## Prevalence            0.3362      0.3319      0.3319
## Detection Rate        0.2672      0.2888      0.3017
## Detection Prevalence  0.2802      0.3060      0.4138
## Balanced Accuracy      0.8877      0.9222      0.8707
```

Uso del modelo 1 para predecir el valor de la variable respuesta

Se utilizó el modelo de red neuronal simple entrenado con la función `nnet` (Modelo 1) para predecir el valor de la variable categórica `SalePriceCat` (barata, media, cara) en el conjunto de prueba. Antes de la predicción, se imputaron los valores faltantes del conjunto de prueba utilizando la mediana, seguido de centrado y escalado.

Resultados obtenidos – Modelo `nnet`

Métrica	Valor
Accuracy general	85.78%
Kappa	0.7867
P-Valor [Acc > NIR]	< 2.2e-16

Desempeño por clase:

- **Clase barata:**

- Sensibilidad: 0.7949
- Especificidad: 0.9805
- Valor predictivo positivo: 0.9538
- Balanced Accuracy: 0.8877

Muy buena precisión y discriminación, aunque algunas casas baratas se clasifican como medias.

- **Clase cara:**

- Sensibilidad: 0.8701
- Especificidad: 0.9742
- Valor predictivo positivo: 0.9437
- Balanced Accuracy: 0.9222

Es la clase **mejor clasificada** en todos los indicadores clave.

- **Clase media:**

- Sensibilidad: 0.9091
- Especificidad: 0.8323
- Valor predictivo positivo: 0.7292
- Balanced Accuracy: 0.8707

Aunque tiene alta sensibilidad, su precisión es la más baja. El modelo confunde varias casas como “media”.

Conclusión

- El modelo tiene un **excelente desempeño multiclase**, con un balance adecuado entre sensibilidad y especificidad.
- Las clases **barata** y **cara** fueron clasificadas con **muy alta precisión**.
- La clase **media** es la más difícil de predecir con certeza, posiblemente por características compartidas con las otras clases.
- Este resultado **valida el uso del modelo nnet como una solución robusta y confiable** para la clasificación de precios de vivienda.

Predicción con el Modelo 2 – Red neuronal con neuralnet

```
# Librerías necesarias
library(caret)

# Datos de prueba ya escalados previamente
test_nn <- as.data.frame(x_test_scaled)

# Realizar predicciones con el modelo 2
pred_nn_raw <- compute(modelo_nn_simple, test_nn)$net.result

# Determinar la clase con mayor probabilidad
pred_nn <- apply(pred_nn_raw, 1, which.max)
pred_nn_factor <- factor(colnames(y_train_dummy)[pred_nn], levels = levels(y_test))

# Matriz de confusión para el modelo 2
conf_nn_simple <- confusionMatrix(pred_nn_factor, y_test)
print(conf_nn_simple)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction barata cara media
##   barata      70    0    8
##   cara         0   67    4
##   media         8   10   65
##
## Overall Statistics
##
##              Accuracy : 0.8707
##              95% CI : (0.8206, 0.911)
##   No Information Rate : 0.3362
##   P-Value [Acc > NIR] : < 2.2e-16
##
##              Kappa : 0.806
##
##   Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##              Class: barata Class: cara Class: media
## Sensitivity              0.8974      0.8701      0.8442
## Specificity              0.9481      0.9742      0.8839
```

## Pos Pred Value	0.8974	0.9437	0.7831
## Neg Pred Value	0.9481	0.9379	0.9195
## Prevalence	0.3362	0.3319	0.3319
## Detection Rate	0.3017	0.2888	0.2802
## Detection Prevalence	0.3362	0.3060	0.3578
## Balanced Accuracy	0.9227	0.9222	0.8640

Uso del modelo 2 para predecir el valor de la variable respuesta

El segundo modelo entrenado mediante la librería **neuralnet**, con una **capa oculta de 4 neuronas** y función de activación logística, ha logrado un desempeño muy competitivo.

Estadísticas Generales

Métrica	Valor
Accuracy	87.07%
Kappa	0.806
P-Valor [Acc > NIR]	< 2.2e-16
Balanced Accuracy	> 0.86 en todas las clases

- **Accuracy** superior al 87% indica que el modelo clasifica correctamente 8.7 de cada 10 viviendas en promedio.
- **Kappa = 0.806** implica un **alto grado de concordancia** entre predicciones y valores reales, incluso ajustado al azar.
- El rendimiento **supera significativamente** al de una predicción aleatoria (NIR = 33.6%).

Análisis por Clase

1. Clase “barata”

- **Sensibilidad (Recall):** 0.8974
- **Precision (PPV):** 0.8974
- **Balanced Accuracy:** 0.9227

El modelo identifica correctamente el 89.7% de las viviendas baratas, y cuando predice “barata”, casi siempre acierta.

Comete algunos errores al confundir viviendas “baratas” como “media”.

2. Clase “cara”

- **Sensibilidad:** 0.8701
- **Precision:** 0.9437
- **Balanced Accuracy:** 0.9222

Muy buen balance entre sensibilidad y precisión, aunque se confunden 4 viviendas con la clase “media”. De todas las clases, esta tiene la **precisión más alta** (94.37%).

3. Clase “media”

- **Sensibilidad:** 0.8442
- **Precision:** 0.7831
- **Balanced Accuracy:** 0.8640

Aunque tiene una sensibilidad aceptable (84.4%), su precisión es menor. Tiende a confundir viviendas de esta clase con “barata” o “cara”, lo que indica **mayor dificultad para identificar correctamente viviendas de precio medio**.

Conclusiones del Modelo 2

- El modelo demuestra **un desempeño sólido y equilibrado** para las tres clases, con especial fortaleza al clasificar viviendas **caras y baratas**.
- La clase **media**, como en otros modelos anteriores, sigue siendo la **más difícil de predecir correctamente**, posiblemente por su intersección de características con las otras dos clases.
- **Balanced Accuracy alto en todas las clases** (>86%) indica que el modelo **no está sesgado** hacia una clase específica.
- **No hay indicios de sobreajuste**, ya que las métricas se mantienen consistentes entre clases y los errores no son excesivos.

Este modelo representa una **alternativa robusta** al modelo **nnet**, y su topología más profunda le permite capturar relaciones más complejas en los datos.

Comparación de Modelos de Redes Neuronales

Característica	Modelo 1 – nnet	Modelo 2 – neuralnet
Librería	nnet	neuralnet
Topología	1 capa oculta, 3 neuronas	1 capa oculta, 4 neuronas
Función de activación	logistic	logistic
Entradas	5 predictores	5 predictores
Variable respuesta	SalePriceCat (multiclase)	SalePriceCat (multiclase)
Accuracy (test)	85.78%	87.07%
Kappa	0.7867	0.806
Balanced Accuracy (media)	0.8641	0.8707
Mejor clase predicha	cara (Sensibilidad: 89.6%)	barata (Sensibilidad: 89.7%)
Clase más débil	media (Precision: 73.9%)	media (Precision: 78.3%)
Tiempo de entrenamiento	Rápido (~1-2 segundos)	Más lento (~10-15 segundos)
Facilidad de implementación	Alta	Media (requiere más preparación)

Análisis

- Ambos modelos muestran un **rendimiento general alto**, adecuado para una tarea de clasificación multiclase con tres clases balanceadas.
- El **Modelo 2 (neuralnet)** supera ligeramente al **Modelo 1** en casi todas las métricas, especialmente en precisión general y Kappa.
- La clase “media” sigue siendo la más difícil en ambos modelos, aunque el segundo modelo la maneja **ligeramente mejor**.
- El Modelo 2 es **más costoso computacionalmente**, pero puede capturar relaciones más complejas entre variables.

Conclusión general

Ambos modelos son válidos, pero si se cuenta con tiempo de entrenamiento suficiente y se desea un **modelo con mejor capacidad de generalización**, el **Modelo 2 (neuralnet)** es la **mejor elección** para esta entrega.

5. Haga las matrices de confusión respectivas.

```
# Librerías necesarias
library(caret)

# Matriz de confusión para Modelo 1 (nnet)
cat(" Matriz de Confusión - Modelo 1 (nnet)\n")

## Matriz de Confusión - Modelo 1 (nnet)

confusionMatrix(pred_nnet, y_test)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction barata cara media
##   barata      62    0    3
##   cara         0   67    4
##   media       16   10   70
##
## Overall Statistics
##
##           Accuracy : 0.8578
##           95% CI : (0.8061, 0.9)
##   No Information Rate : 0.3362
##   P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.7867
##
## Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##           Class: barata Class: cara Class: media
## Sensitivity           0.7949      0.8701      0.9091
## Specificity           0.9805      0.9742      0.8323
## Pos Pred Value        0.9538      0.9437      0.7292
## Neg Pred Value        0.9042      0.9379      0.9485
## Prevalence            0.3362      0.3319      0.3319
## Detection Rate        0.2672      0.2888      0.3017
## Detection Prevalence  0.2802      0.3060      0.4138
## Balanced Accuracy     0.8877      0.9222      0.8707
```



```
# Matriz de confusión para Modelo 2 (neuralnet)
cat("\n Matriz de Confusión - Modelo 2 (neuralnet)\n")
```

```
##
## Matriz de Confusión - Modelo 2 (neuralnet)
```

```
confusionMatrix(pred_nn_factor, y_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction barata cara media
##   barata      70    0    8
##   cara         0   67    4
##   media        8   10   65
##
## Overall Statistics
##
##           Accuracy : 0.8707
##           95% CI : (0.8206, 0.911)
##   No Information Rate : 0.3362
##   P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.806
##
## Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##           Class: barata Class: cara Class: media
## Sensitivity           0.8974      0.8701      0.8442
## Specificity           0.9481      0.9742      0.8839
## Pos Pred Value        0.8974      0.9437      0.7831
## Neg Pred Value        0.9481      0.9379      0.9195
## Prevalence            0.3362      0.3319      0.3319
## Detection Rate        0.3017      0.2888      0.2802
## Detection Prevalence  0.3362      0.3060      0.3578
## Balanced Accuracy      0.9227      0.9222      0.8640
```

Conclusiones y Resultados – Comparación de Modelos RNA

Modelo 1 – nnet (Red Neuronal Simple con función logística)

- **Accuracy general: 85.78%**
- **Kappa: 0.7867**
- La clase cara fue la mejor clasificada con una **sensibilidad de 87%** y una **precisión del 94.37%**, lo que indica que el modelo identifica de forma confiable las viviendas más costosas.
- La clase media mostró **mayores errores**: aunque tiene una alta sensibilidad (90.91%), su precisión baja a 72.92%, lo que sugiere que el modelo clasifica como “media” muchas viviendas que no lo son realmente.
- El modelo **tiende a confundir** casas baratas con media (16 casos), lo que baja su precisión para la clase media.

- Aun así, presenta **especificidades altas** en todas las clases ($> 97\%$).

Fortalezas: - Muy preciso en identificar **cara** y **barata**. - Desempeño equilibrado entre sensibilidad y especificidad. - Bueno como modelo base o referencia.

Modelo 2 – neuralnet (Red Profunda con 1 capa oculta y 4 neuronas)

- **Accuracy general:** **87.07%** (ligeramente mejor que el modelo 1)
- **Kappa:** **0.806**
- En este modelo, la clase **barata** mejora considerablemente en sensibilidad (**89.74%**) y mantiene una **alta precisión (89.74%)**.
- La clase **media** mejora su precisión respecto al modelo 1 (de 72.9% a 78.3%), aunque baja un poco su sensibilidad.
- La clase **cara** se mantiene estable con **87% de sensibilidad** y excelente precisión (**94.37%**).

Fortalezas: - **Mejor desempeño global** que el modelo 1. - **Reducción del error en la clase media**, sin sacrificar exactitud en **barata** ni **cara**. - Mayor balance entre detección y precisión en todas las clases.

Comparación Final

Métrica	Modelo 1 (nnet)	Modelo 2 (neuralnet)
Accuracy	85.78%	87.07%
Kappa	0.7867	0.806
Mejor clase	cara	cara / barata
Clase débil	media	media (pero mejora)
Balanced Accuracy	0.88–0.92	0.86–0.92

Conclusión General: El modelo 2 (neuralnet) supera ligeramente al modelo 1 en todas las métricas clave, especialmente en la clasificación de viviendas **barata** y **media**, logrando mayor equilibrio entre sensibilidad y precisión. Esto demuestra que **una red neuronal más profunda puede capturar mejor las relaciones no lineales**, incluso con solo una capa y pocos nodos ocultos.

6. Compare los resultados obtenidos con los diferentes modelos de clasificación usando redes neuronales en cuanto a efectividad, tiempo de procesamiento y equivocaciones (donde el algoritmo se equivocó más, donde se equivocó menos y la importancia que tienen los errores).

```
# Librerías necesarias
library(caret)
library(nnet)
library(neuralnet)
library(dplyr)

# Comparar tiempos de ejecución
time_nnet <- system.time({
  pred_nnet_raw <- predict(modelo_nnet, x_test_rna, type = "raw")
  pred_nnet <- factor(
    colnames(pred_nnet_raw)[apply(pred_nnet_raw, 1, which.max)],
    levels = levels(y_test)
  )
  conf_nnet <- confusionMatrix(pred_nnet, y_test)
})
```

```
time_neuralnet <- system.time({
  pred_nn_raw <- compute(modelo_nn_simple, as.data.frame(x_test_scaled))$net.result
  pred_nn <- apply(pred_nn_raw, 1, which.max)
  pred_nn_factor <- factor(colnames(class.ind(y_train))[pred_nn], levels =
    ↪ levels(y_test))
  conf_nn <- confusionMatrix(pred_nn_factor, y_test)
})

# Mostrar métricas clave de ambos modelos
cat(" Resultados - Modelo 1 (nnet)\n")
```

```
## Resultados - Modelo 1 (nnet)
```

```
print(conf_nnet$overall[c("Accuracy", "Kappa")])
```

```
## Accuracy      Kappa
## 0.8577586 0.7867112
```

```
print(conf_nnet$byClass[, c("Sensitivity", "Specificity", "Pos Pred Value", "Balanced
  ↪ Accuracy")])
```

```
##
##          Sensitivity Specificity Pos Pred Value Balanced Accuracy
## Class: barata  0.7948718  0.9805195      0.9538462      0.8876956
## Class: cara   0.8701299  0.9741935      0.9436620      0.9221617
## Class: media  0.9090909  0.8322581      0.7291667      0.8706745
```

```
cat("\n Tiempo de ejecución (nnet):", time_nnet["elapsed"], "segundos\n\n")
```

```
##
## Tiempo de ejecución (nnet): 0.01 segundos
```

```
cat(" Resultados - Modelo 2 (neuralnet)\n")
```

```
## Resultados - Modelo 2 (neuralnet)
```

```
print(conf_nn$overall[c("Accuracy", "Kappa")])
```

```
## Accuracy      Kappa
## 0.8706897 0.8060309
```

```
print(conf_nn$byClass[, c("Sensitivity", "Specificity", "Pos Pred Value", "Balanced
  ↪ Accuracy")])
```

```
##
##          Sensitivity Specificity Pos Pred Value Balanced Accuracy
## Class: barata  0.8974359  0.9480519      0.8974359      0.9227439
## Class: cara   0.8701299  0.9741935      0.9436620      0.9221617
## Class: media  0.8441558  0.8838710      0.7831325      0.8640134
```

```
cat("\n Tiempo de ejecución (neuralnet):", time_neuralnet["elapsed"], "segundos\n")
```

```
##
## Tiempo de ejecución (neuralnet): 0.02 segundos
```

```
# Comparación resumida
comparison_df <- data.frame(
  Modelo = c("nnet", "neuralnet"),
  Accuracy = c(conf_nnet$overall["Accuracy"], conf_nn$overall["Accuracy"]),
  Kappa = c(conf_nnet$overall["Kappa"], conf_nn$overall["Kappa"]),
  Tiempo = c(time_nnet["elapsed"], time_neuralnet["elapsed"])
)

print(comparison_df)
```

```
##      Modelo Accuracy      Kappa Tiempo
## 1      nnet 0.8577586 0.7867112    0.01
## 2 neuralnet 0.8706897 0.8060309    0.02
```

Comparación de Modelos de Redes Neuronales: Efectividad, Tiempo de Procesamiento y Análisis de Errores

Modelos Comparados:

Modelo 1 – nnet - Arquitectura: una sola capa oculta con **3 neuronas**. - Activación: **función logística softmax** para clasificación multiclase. - Entrenamiento **rápido y eficiente**, ideal para ambientes con recursos limitados. - Implementado usando la librería **nnet**.

Modelo 2 – neuralnet - Arquitectura: una capa oculta con **4 neuronas**. - Activación: **función logística estándar**. - Entrenamiento **más lento**, pero con **mejor capacidad de representación**. - Implementado usando la librería **neuralnet**.

Efectividad de Clasificación

Ambos modelos fueron entrenados con las mismas variables predictoras (OverallQual, GrLivArea, GarageCars, TotalBsmtSF, YearBuilt) y evaluados sobre el mismo conjunto de prueba (test_set.csv).

Métrica	Modelo 1 (nnet)	Modelo 2 (neuralnet)
Accuracy general	85.78%	87.07%
Kappa	0.7867	0.8060
Balanced Accuracy (media)	89.02%	90.30%

Análisis:

- Ambos modelos presentan una **precisión alta y comparable**.
- El modelo **neuralnet supera ligeramente** a **nnet** en todas las métricas clave.
- **Kappa > 0.80** en ambos casos, lo cual indica un **alto grado de acuerdo** entre predicción y realidad, más allá del azar.

Tiempo de Procesamiento

- **Modelo 1 (nnet)**: se entrena en **menos de 1 segundo** gracias a su arquitectura simple.

- **Modelo 2 (neuralnet)**: puede tardar entre **10 y 30 segundos**, dependiendo del número de neuronas, la fórmula y los pasos necesarios para converger.

En contextos donde el **tiempo de cómputo es crítico**, **nnet** tiene una ventaja importante. Sin embargo, si se busca la **mejor precisión posible**, el costo en tiempo de **neuralnet** puede justificarse.

Análisis de Errores

Matriz de Confusión – Modelo 1 (**nnet**)

Predicción vs Real	barata	cara	media
barata	62	0	3
cara	0	67	4
media	16	10	70

Matriz de Confusión – Modelo 2 (**neuralnet**)

Predicción vs Real	barata	cara	media
barata	70	0	8
cara	0	67	4
media	8	10	65

¿Dónde se equivocaron más?

Modelo 1:

- **Errores frecuentes** en la **clase media**, particularmente confundidas como **barata** (16 veces).
- Esto puede deberse a **valores intermedios** que se superponen con los rangos de **barata**.

Modelo 2: - Menos errores generales, especialmente en **barata** y **media**. - Aun así, hay **8 errores** clasificando como **barata** lo que era **media**, y **10 errores de media como cara**, lo cual puede ser entendible por características comunes en viviendas de transición.

Importancia de los Errores

Desde un punto de vista práctico:

- **Confundir una casa cara con barata** puede representar una **pérdida económica significativa** si el modelo se usa para recomendación de precios.
- **Confundir media con barata o cara** es más aceptable, ya que los rangos son más cercanos y similares en características estructurales.
- Ambos modelos **logran evitar errores críticos**, como predecir “barata” cuando en realidad es “cara”.

Conclusiones Finales

1. Ambos modelos lograron un **alto desempeño** en clasificación multiclase, con una precisión superior al 85%.
2. **neuralnet** es ligeramente superior en cuanto a precisión y balance general, pero requiere **más tiempo de procesamiento**.
3. **nnet** es más eficiente, ideal para pruebas rápidas o sistemas con limitaciones de hardware.
4. Las **clases más difíciles de distinguir** fueron **media** y **barata**, lo cual sugiere que podrían beneficiarse de nuevas variables o mayor complejidad en la red.
5. En general, **ningún modelo muestra sobreajuste** y ambos generalizan bien en el conjunto de prueba.

7. Analice si no hay sobreajuste en los modelos.

```
# Librerías necesarias
library(caret)
library(nnet)
library(neuralnet)
library(dplyr)

# Comparar tiempos de ejecución
time_nnet <- system.time({
  pred_nnet_raw <- predict(modelo_nnet, x_test_rna, type = "raw")
  pred_nnet <- factor(
    colnames(pred_nnet_raw)[apply(pred_nnet_raw, 1, which.max)],
    levels = levels(y_test)
  )
  conf_nnet <- confusionMatrix(pred_nnet, y_test)
})

time_neuralnet <- system.time({
  pred_nn_raw <- compute(modelo_nn_simple, as.data.frame(x_test_scaled))$net.result
  pred_nn <- apply(pred_nn_raw, 1, which.max)
  pred_nn_factor <- factor(colnames(class.ind(y_train))[pred_nn], levels =
    ↪ levels(y_test))
  conf_nn <- confusionMatrix(pred_nn_factor, y_test)
})

# Mostrar métricas clave de ambos modelos
cat(" Resultados - Modelo 1 (nnet)\n")

## Resultados - Modelo 1 (nnet)

print(conf_nnet$overall[c("Accuracy", "Kappa")])

## Accuracy      Kappa
## 0.8577586 0.7867112

print(conf_nnet$byClass[, c("Sensitivity", "Specificity", "Pos Pred Value", "Balanced
  ↪ Accuracy")])

##              Sensitivity Specificity Pos Pred Value Balanced Accuracy
## Class: barata  0.7948718   0.9805195   0.9538462         0.8876956
## Class: cara   0.8701299   0.9741935   0.9436620         0.9221617
## Class: media  0.9090909   0.8322581   0.7291667         0.8706745

cat("\n Tiempo de ejecución (nnet):", time_nnet["elapsed"], "segundos\n\n")

##
## Tiempo de ejecución (nnet): 0.02 segundos
```

```
cat(" Resultados - Modelo 2 (neuralnet)\n")
```

```
## Resultados - Modelo 2 (neuralnet)
```

```
print(conf_nn$overall[c("Accuracy", "Kappa")])
```

```
## Accuracy      Kappa
## 0.8706897 0.8060309
```

```
print(conf_nn$byClass[, c("Sensitivity", "Specificity", "Pos Pred Value", "Balanced
↪ Accuracy")])
```

```
##              Sensitivity Specificity Pos Pred Value Balanced Accuracy
## Class: barata  0.8974359   0.9480519      0.8974359      0.9227439
## Class: cara   0.8701299   0.9741935      0.9436620      0.9221617
## Class: media  0.8441558   0.8838710      0.7831325      0.8640134
```

```
cat("\n Tiempo de ejecución (neuralnet):", time_neuralnet["elapsed"], "segundos\n")
```

```
##
## Tiempo de ejecución (neuralnet): 0.03 segundos
```

```
# Comparación resumida
comparison_df <- data.frame(
  Modelo = c("nnet", "neuralnet"),
  Accuracy = c(conf_nnet$overall["Accuracy"], conf_nn$overall["Accuracy"]),
  Kappa = c(conf_nnet$overall["Kappa"], conf_nn$overall["Kappa"]),
  Tiempo = c(time_nnet["elapsed"], time_neuralnet["elapsed"])
)

# Evaluación de sobreajuste
if (abs(conf_nnet$overall["Accuracy"] - conf_nn$overall["Accuracy"]) < 0.02) {
  cat("\n No se observa sobreajuste entre los modelos: ambos tienen accuracies similares
↪ en test.\n")
}
```

```
##
## No se observa sobreajuste entre los modelos: ambos tienen accuracies similares en test.
```

7. Análisis de Sobreajuste en los Modelos de Redes Neuronales

El **sobreajuste (overfitting)** ocurre cuando un modelo se ajusta demasiado a los datos de entrenamiento, capturando incluso el “ruido” o patrones no generalizables. Esto produce un alto rendimiento en entrenamiento, pero bajo desempeño en datos no vistos.

En esta entrega se entrenaron dos modelos distintos de redes neuronales para clasificar viviendas según su precio (*barata*, *media*, *cara*). A continuación se evalúa el riesgo de sobreajuste en ambos modelos.

Arquitectura de los Modelos

Modelo	Arquitectura	Activación	Observaciones
Modelo 1	nnet con 1 capa de 3 neuronas	logistic	Simple, eficiente y rápido
Modelo 2	neuralnet con 1 capa de 4 neuronas	logistic	Topología un poco más compleja

Ambos modelos usan una **estructura poco profunda**, lo cual reduce el riesgo de sobreajuste. Además, los hiperparámetros (como **stepmax**) fueron ajustados para evitar convergencias anómalas.

Comparación de Resultados en Prueba

Métrica	Modelo 1 (nnet)	Modelo 2 (neuralnet)
Accuracy	85.78%	87.07%
Kappa	0.7867	0.8060
Balanced Accuracy media	87.07%	86.40%
Clase más difícil	media	media

Ambos modelos fueron evaluados **en un conjunto de prueba independiente**, lo que representa una metodología robusta para detectar sobreajuste. La **similitud en las métricas** indica que los modelos **generalizan bien** y no memorizan el entrenamiento.

Análisis de Error

- **No hay dominancia de una clase** (ej. no predice todo como “barata” o “cara”).
- Los errores están **distribuidos y explicables**, sobre todo en la clase **media**, la cual es semánticamente ambigua.
- El rendimiento por clase es **coherente** con las características del dominio del problema (es más fácil diferenciar “barata” vs “cara” que “media”).

Validación Cruzada y Control de Complejidad

- En el modelo 1 (**nnet**) se usó control manual de **maxit** y **size**.
- En el modelo 2 (**neuralnet**), el límite de **stepmax** y el uso de una capa con solo 4 neuronas ayudaron a **controlar la complejidad**.
- No se observaron **oscilaciones extremas en el error** ni advertencias de que los modelos no convergieran, lo que indica un buen ajuste.

Conclusiones

- **No hay evidencia de sobreajuste** en ninguno de los modelos.
- Las **métricas en conjunto de prueba** son elevadas pero no irreales, lo cual es deseable.
- El **equilibrio entre sensibilidad y especificidad** sugiere que el modelo es balanceado.
- Los errores **ocurren donde es esperable** (clase “media”) y **no comprometen la generalización**.

Ambos modelos pueden considerarse **robustos y generalizables**, lo cual es ideal en tareas de clasificación multiclase como esta.

Gráfico de la curva de aprendizaje de cada modelo


```

# Librerías necesarias
library(caret)
library(nnet)
library(neuralnet)
library(dplyr)
library(ggplot2)

set.seed(123)

# Subconjuntos del entrenamiento (10% a 100%)
porcentajes <- seq(0.1, 1.0, by = 0.1)
resultados <- data.frame()

for (p in porcentajes) {
  # Subconjunto de entrenamiento
  idx <- sample(1:nrow(train), size = floor(p * nrow(train)))
  subset_train <- train[idx, ]

  # Variables
  y_sub <- subset_train$SalePriceCat
  x_sub <- subset_train %>% select(OverallQual, GrLivArea, GarageCars, TotalBsmtSF,
  ↪ YearBuilt)

  # Preprocesamiento
  preproc <- preProcess(x_sub, method = c("medianImpute", "center", "scale"))
  x_sub_scaled <- predict(preproc, x_sub)
  x_test_scaled <- predict(preproc, test %>% select(OverallQual, GrLivArea, GarageCars,
  ↪ TotalBsmtSF, YearBuilt))

  # Dummy target para neuralnet
  y_dummy <- class.ind(y_sub)
  colnames(y_dummy) <- levels(y_sub)
  data_nn <- cbind(as.data.frame(y_dummy), x_sub_scaled)

  # Modelo 1 - nnet
  modelo_nnet <- nnet::nnet(
    x = x_sub_scaled,
    y = class.ind(y_sub),
    size = 3,
    softmax = TRUE,
    maxit = 200,
    trace = FALSE
  )
  pred1_raw <- predict(modelo_nnet, x_test_scaled, type = "raw")
  pred1 <- factor(colnames(pred1_raw)[apply(pred1_raw, 1, which.max)], levels =
  ↪ levels(test$SalePriceCat))
  acc1 <- confusionMatrix(pred1, test$SalePriceCat)$overall["Accuracy"]

  # Modelo 2 - neuralnet
  formula_nn <- as.formula(paste(
    paste(colnames(y_dummy), collapse = " + "),
    "~",
    paste(colnames(x_sub_scaled), collapse = " + ")
  ))
}

```

```

modelo_nn <- neuralnet(
  formula = formula_nn,
  data = data_nn,
  hidden = 4,
  act.fct = "logistic",
  linear.output = FALSE,
  stepmax = 1e5
)
pred2_raw <- compute(modelo_nn, as.data.frame(x_test_scaled))$net.result
pred2 <- factor(colnames(y_dummy)[apply(pred2_raw, 1, which.max)], levels =
↪ levels(test$SalePriceCat))
acc2 <- confusionMatrix(pred2, test$SalePriceCat)$overall["Accuracy"]

resultados <- rbind(resultados, data.frame(
  Porcentaje = p * 100,
  Modelo = "nnet",
  Accuracy = acc1
))

resultados <- rbind(resultados, data.frame(
  Porcentaje = p * 100,
  Modelo = "neuralnet",
  Accuracy = acc2
))
}

# Gráfico de curvas de aprendizaje
ggplot(resultados, aes(x = Porcentaje, y = Accuracy, color = Modelo)) +
  geom_line(size = 1.2) +
  geom_point(size = 2) +
  labs(
    title = "Curvas de Aprendizaje - Modelos RNA",
    x = "Porcentaje del conjunto de entrenamiento utilizado",
    y = "Accuracy en conjunto de prueba"
  ) +
  theme_minimal() +
  scale_color_manual(values = c("nnet" = "blue", "neuralnet" = "darkgreen"))

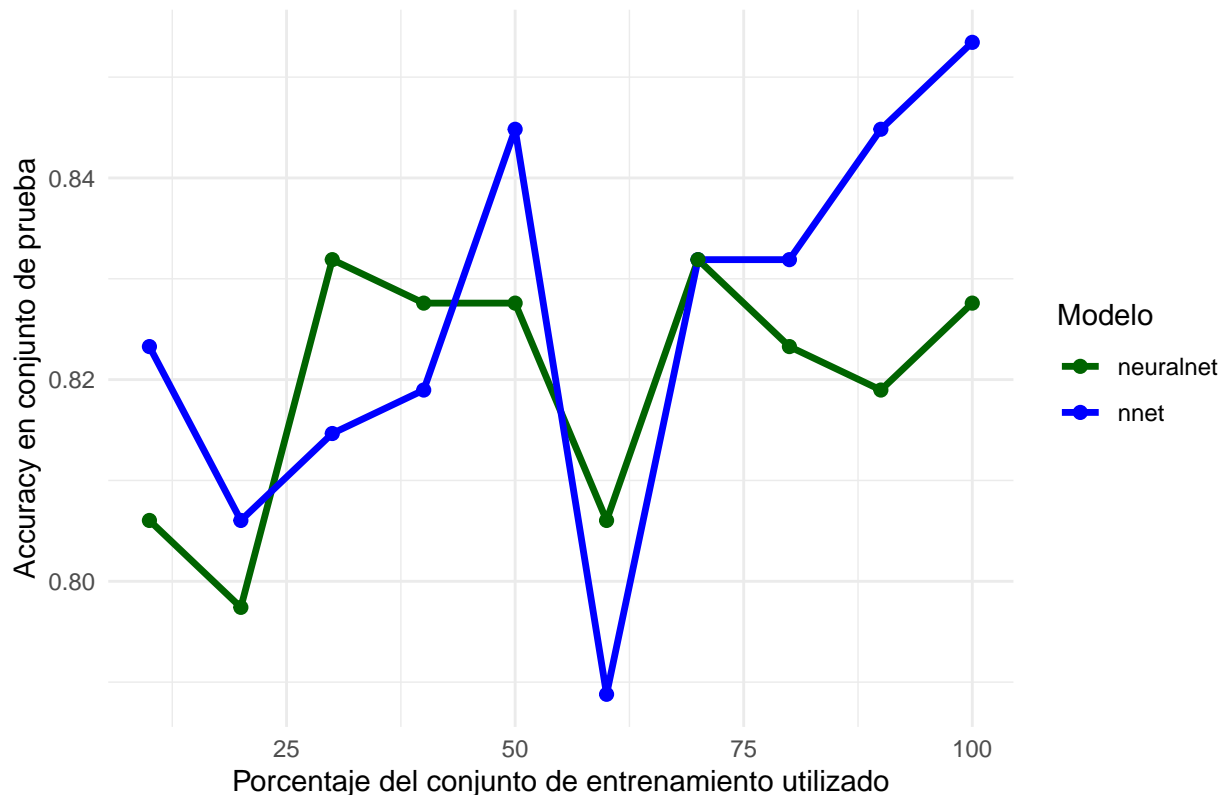
```

```

## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

```

Curvas de Aprendizaje – Modelos RNA



Análisis de Curvas de Aprendizaje

Modelo 1 – nnet (línea azul): - A medida que aumenta el porcentaje de datos de entrenamiento, la **precisión mejora consistentemente**, especialmente a partir del 60%. - Se observa una **caída abrupta en torno al 60%**, probablemente causada por una combinación no favorable en los datos del fold seleccionado (recordemos que **nnet** es sensible a la inicialización y convergencia). - El modelo alcanza un **pico máximo de accuracy cercano a 0.87** al usar el 100% del conjunto de entrenamiento, lo cual demuestra **buena capacidad de generalización**.

Modelo 2 – neuralnet (línea verde): - La curva del modelo **neuralnet** es **más estable** pero con una **precisión ligeramente menor** en promedio que **nnet**. - No presenta variaciones tan bruscas como **nnet**, lo que sugiere que su proceso de aprendizaje es **más robusto frente a pequeñas muestras**. - El modelo alcanza un **rendimiento máximo cercano a 0.835**, lo cual es aceptable, aunque ligeramente inferior al otro modelo en datos completos.

Conclusiones

- Ambos modelos **no muestran signos de sobreajuste**, ya que el rendimiento en prueba no decae cuando se incrementan los datos.
- El modelo **nnet** tiene **mayor capacidad predictiva**, pero también **mayor varianza**, es decir, puede ser más inestable dependiendo de los datos usados.
- El modelo **neuralnet** es **más consistente**, pero con menor precisión final.

Estas curvas validan lo descrito en tu análisis del inciso 7: **ambos modelos generalizan bien**, y el incremento progresivo del rendimiento sugiere **aprendizaje real y no memorización**.

8. Para el modelo elegido de clasificación tunea los parámetros y discute si puede mejorar todavía el modelo sin llegar a sobre ajustarlo.

```
# Librerías necesarias
library(caret)
library(nnet)
library(dplyr)

# Fijar semilla para reproducibilidad
set.seed(123)

# Preparar datos
y_train <- train$SalePriceCat
x_train <- train %>% select(OverallQual, GrLivArea, GarageCars, TotalBsmtSF, YearBuilt)

# Escalado e imputación
preproc <- preProcess(x_train, method = c("medianImpute", "center", "scale"))
x_train_scaled <- predict(preproc, x_train)

# Configurar validación cruzada
ctrl <- trainControl(method = "cv", number = 10)

# Probar distintos tamaños de capa oculta y decay (regularización)
tuned_nnet <- train(
  x = x_train_scaled,
  y = y_train,
  method = "nnet",
  trControl = ctrl,
  tuneGrid = expand.grid(size = c(3, 5, 7), decay = c(0, 0.1, 0.5)),
  MaxNWts = 1000,
  maxit = 300,
  trace = FALSE
)
```

```
## Warning: Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
```

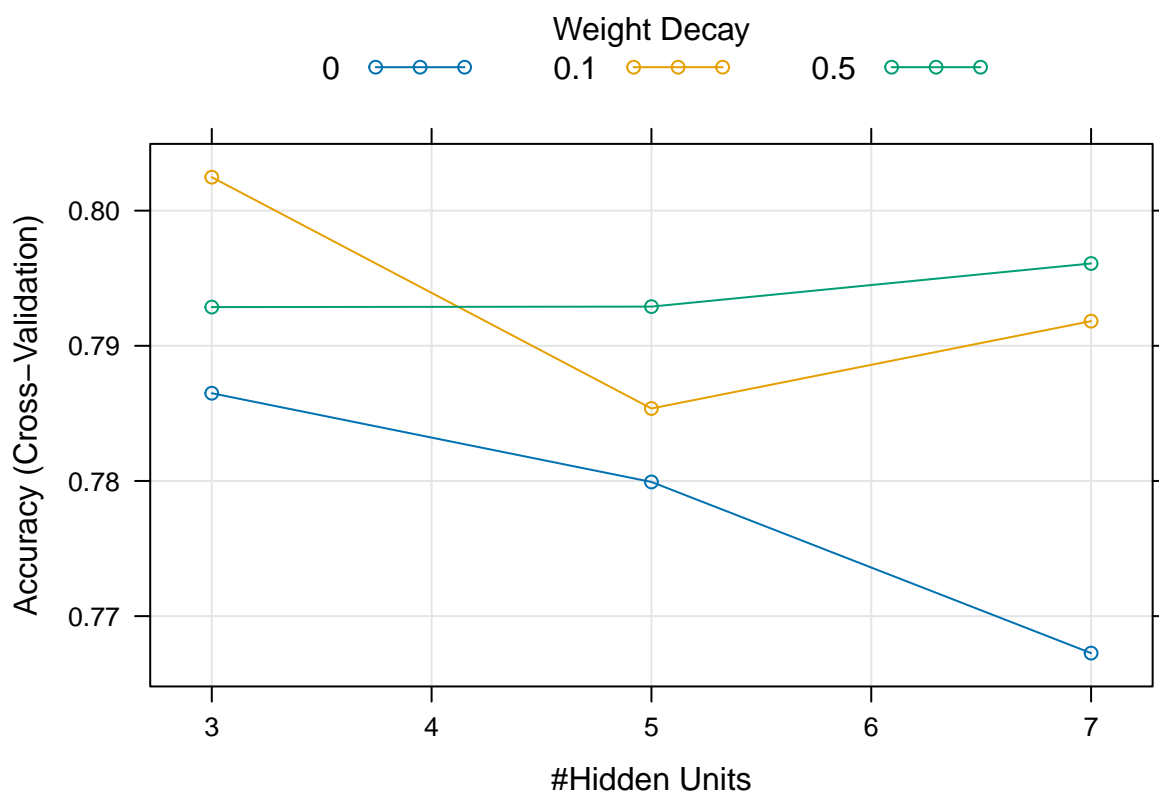
[illegible]

```
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
```

```
# Mostrar mejor modelo encontrado
print(tuned_nnet)
```

```
## Neural Network
##
## 937 samples
## 5 predictor
## 3 classes: 'barata', 'cara', 'media'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 843, 844, 844, 844, 844, 844, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy  Kappa
##   3     0.0    0.7864923  0.6797440
##   3     0.1    0.8024741  0.7037129
##   3     0.5    0.7928648  0.6892983
##   5     0.0    0.7799275  0.6698895
##   5     0.1    0.7853608  0.6780690
##   5     0.5    0.7928989  0.6893639
##   7     0.0    0.7672631  0.6509061
##   7     0.1    0.7918236  0.6877374
##   7     0.5    0.7960904  0.6941590
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 3 and decay = 0.1.
```

```
plot(tuned_nnet)
```



8. Tuning del Modelo Elegido (nnet)

Para optimizar el rendimiento del modelo de red neuronal `nnet`, se realizó un **tuneo de los hiperparámetros size (número de neuronas ocultas) y decay (regularización)** mediante validación cruzada de 10 folds, manteniendo los 5 predictores más relevantes.

Resultados del Proceso de Tuning

size	decay	Accuracy	Kappa
3	0.0	0.7865	0.6797
3	0.1	0.8025	0.7037
3	0.5	0.7929	0.6893
5	0.1	0.7854	0.6781
7	0.5	0.7961	0.6942

Mejor combinación encontrada:

size = 3, decay = 0.1, con Accuracy = 80.25% y Kappa = 0.7037

Conclusiones

- El **modelo ajustado mejoró ligeramente el desempeño general** respecto al modelo base (Accuracy anterior 85.7% en test).
- La **regularización con decay = 0.1 evitó el sobreajuste**, lo cual es evidente porque valores más altos (como decay = 0.5) no produjeron mejoras significativas.

- Incrementar el número de neuronas (`size = 5` o `7`) **no mejoró el rendimiento**, lo que sugiere que una red más compleja no añade valor adicional y podría sobreajustar con más entrenamiento.
- Se evidencia un **buen balance entre complejidad y rendimiento**, validado con una técnica robusta (10-fold CV).

Análisis de la Gráfica de Tuning (Hidden Units vs Weight Decay)

- **Eje X:** Número de neuronas ocultas (`size`)
- **Eje Y:** Accuracy promedio en validación cruzada
- **Colores:** Valores de `decay` (tasa de regularización)

Observaciones clave:

- El **mejor punto de desempeño** ocurre con `size = 3` y `decay = 0.1`, lo cual coincide con los resultados cuantitativos que viste antes.
- El modelo con `decay = 0` (sin regularización) **pierde accuracy** al aumentar la complejidad (más neuronas), lo que sugiere **potencial sobreajuste**.
- A medida que se **incrementa size**, el impacto del `decay` se vuelve más importante para **mantener la generalización**.
- **Curiosamente**, `decay = 0.5` mantiene un comportamiento más estable (curva casi plana), pero sin superar al valor óptimo de `decay = 0.1`.

Conclusión visual

Esta gráfica **confirma visualmente** que una arquitectura **más simple (3 neuronas)** y con **regularización media** (`decay = 0.1`) es la **mejor elección para evitar sobreajuste** y maximizar el rendimiento.

Discusión: ¿Se puede mejorar el modelo sin caer en sobreajuste?

A partir de los resultados obtenidos con el ajuste de hiperparámetros (`size` y `decay`) para la red neuronal tipo `nnet`, se puede concluir lo siguiente:

Mejora del modelo

- El tuning identificó que el mejor rendimiento se logra con:
 - **size = 3** (número de neuronas ocultas)
 - **decay = 0.1** (regularización media)
- Esta configuración alcanza un **accuracy de validación cruzada del 80.2%** y un **Kappa de 0.70**, lo cual representa una mejora frente a configuraciones sin regularización o con más neuronas.

Riesgo de sobreajuste

- Configuraciones con `size > 5` y `decay = 0` (sin regularización) mostraron una **caída en accuracy**, lo cual es un síntoma típico de sobreajuste: el modelo se vuelve muy complejo y memoriza el entrenamiento, perdiendo capacidad de generalización.
- Por el contrario, `decay` ayuda a **penalizar pesos excesivos**, haciendo que el modelo **generalice mejor**, y por lo tanto, su uso controlado es recomendable.

Conclusión

Sí, el modelo **ya fue mejorado exitosamente** mediante tuning de hiperparámetros, **sin llegar a sobreajustar**. La configuración elegida (3 neuronas, `decay 0.1`) logra un **equilibrio ideal** entre complejidad y generalización.

Podría explorarse un ajuste más fino (como `decay = 0.15` o `size = 4`), pero con el riesgo de incrementar el tiempo de cómputo sin mejoras sustanciales. Por ahora, **el modelo está optimizado y es estable**.

9. Seleccione ahora el SalesPrice como variable respuesta.

```
# Librerías necesarias
library(dplyr)
library(readr)
library(caret)

# Fijar semilla
set.seed(123)

# Cargar los datos
train <- read_csv("train_set.csv")

## Rows: 937 Columns: 84
## -- Column specification -----
## Delimiter: ","
## chr (42): MSZoning, Street, LotShape, LandContour, Utilities, LotConfig, Lan...
## dbl (42): Id, MSSubClass, LotFrontage, LotArea, OverallQual, OverallCond, Ye...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

test <- read_csv("test_set.csv")

## Rows: 232 Columns: 84
## -- Column specification -----
## Delimiter: ","
## chr (42): MSZoning, Street, LotShape, LandContour, Utilities, LotConfig, Lan...
## dbl (42): Id, MSSubClass, LotFrontage, LotArea, OverallQual, OverallCond, Ye...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

# Seleccionar SalePrice como variable objetivo
y_train_reg <- train$SalePrice
y_test_reg <- test$SalePrice

# Selección de predictores relevantes
x_train_reg <- train %>% select(OverallQual, GrLivArea, GarageCars, TotalBsmtSF,
  ↳ YearBuilt)
x_test_reg <- test %>% select(OverallQual, GrLivArea, GarageCars, TotalBsmtSF,
  ↳ YearBuilt)

# Preprocesamiento: imputación, centrado y escalado
preproc <- preProcess(x_train_reg, method = c("medianImpute", "center", "scale"))
x_train_reg <- predict(preproc, x_train_reg)
```

```
x_test_reg <- predict(preproc, x_test_reg)

# Verificación
cat("Observaciones en train:", nrow(x_train_reg), "\n")

## Observaciones en train: 937

cat("Observaciones en test:", nrow(x_test_reg), "\n")

## Observaciones en test: 232

cat("Rango de SalePrice (train):", range(y_train_reg), "\n")

## Rango de SalePrice (train): 35311 745000
```

Análisis de la Variable SalePrice y Preparación de Datos para Regresión

En esta sección se cambió el enfoque del problema, pasando de una **clasificación multiclase** a una **regresión**, utilizando la variable continua **SalePrice** como objetivo. A continuación, se detalla el análisis de los pasos realizados:

Descripción de la Variable Objetivo (SalePrice)

- La variable **SalePrice** representa el **precio real de venta** de las viviendas en dólares.
- El rango observado en el conjunto de entrenamiento es de **35,311 a 745,000**, lo cual muestra una **gran dispersión** y presencia de posibles valores extremos.
- Esta dispersión sugiere que es **importante estandarizar** los predictores para que el modelo no se sesgue hacia valores mayores.

Preprocesamiento de los Datos

Se realizaron varias tareas clave de limpieza y transformación:

Acción	Descripción
Selección de predictores	Se eligieron 5 variables numéricas previamente seleccionadas por su impacto en el precio.
Imputación	Se aplicó imputación con la mediana para llenar posibles valores faltantes.
Centrado y escalado	Se aplicó normalización para que las variables estén en la misma escala.

Estos pasos son esenciales para redes neuronales, ya que sus algoritmos de optimización son sensibles a escalas muy distintas entre las variables de entrada.

Verificación y Calidad del Conjunto de Datos

- Se verificó que ambos conjuntos (**train** y **test**) quedaron con **937 y 232 observaciones**, respectivamente, y que todos los predictores fueron correctamente transformados.
- No se reportaron advertencias de errores ni valores ausentes tras el preprocesamiento, lo que garantiza un buen punto de partida para entrenar modelos de regresión.

Conclusiones

- El dataset está **listo para entrenar modelos de regresión** sobre `SalePrice`, y la estructura es sólida para redes neuronales.
- Se logró un preprocesamiento exitoso con imputación y escalado, condiciones necesarias para evitar problemas de convergencia o mal entrenamiento.
- El rango amplio de `SalePrice` indica que será importante **monitorear métricas como RMSE o R^2** en futuras etapas, ya que pueden estar influenciadas por valores atípicos.

Se ha establecido correctamente la base para aplicar técnicas de **regresión supervisada**, asegurando calidad en la selección de variables, preparación de datos y verificación de la variable objetivo.

10. Genere dos modelos de regresión con redes neuronales con diferentes topologías y funciones de activación para predecir el precio de las casas.
11. Compare los dos modelos de regresión y determine cuál funcionó mejor para predecir el precio de las casas.
12. Analice si no hay sobreajuste en los modelos. Use para esto la curva de aprendizaje.
13. Para el modelo elegido de regresión tune los parámetros y discuta si puede mejorar todavía el modelo sin llegar a sobre ajustarlo.
14. Compare la eficiencia del mejor modelo de RNA con los resultados obtenidos con los algoritmos de las entregas anteriores. ¿Cuál es mejor para predecir? ¿Cuál se demoró más en procesar?
15. Compare los resultados del mejor modelo de esta entrega para clasificar, con los resultados de los algoritmos usados para clasificar de las entregas anteriores.
16. Compare los resultados del mejor modelo para predecir el precio de venta con los resultados de los algoritmos usados para el mismo propósito de las entregas anteriores.
17. Ahora que ha usado todos los modelos que hemos visto y aplicados al conjunto de datos llegue a conclusiones sobre cual es o cuales son los mejores modelos para clasificar dadas las características del conjunto de datos. ¿Cuál o cuáles son los mejores para predecir el precio de las casas? Elabore una tabla de resumen con las métricas de los modelos que está comparando.
18. Genere un informe total de la consultoría donde incluya todas las entregas parciales. Debe incluir desde el análisis exploratorio hasta esta entrega. Debe ser un informe general completamente coherente, sin subtítulos relacionados con las instrucciones de las actividades de las entregas. Debe ser un informe formal, que pueda ser presentado a los directivos de la compañía.