Proyecto 2. Entrega 4. KNN

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

2025-03-14

Enlace al Repositorio del proyecto 2 - Entrega 4 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  1         60       RL          65    8450   Pave      Reg         Lvl
## 2  2         20       RL          80    9600   Pave      Reg         Lvl
## 3  3         60       RL          68   11250   Pave      IR1         Lvl
## 4  5         60       RL          84   14260   Pave      IR1         Lvl
## 5  7         20       RL          75   10084   Pave      Reg         Lvl
## 6  8         60       RL          69   10382   Pave      IR1         Lvl
##   Utilities LotConfig LandSlope Neighborhood Condition1 Condition2 BldgType
## 1    AllPub    Inside       Gtl      CollgCr       Norm       Norm     1Fam
## 2    AllPub       FR2       Gtl      Veenker      Feedr       Norm     1Fam
## 3    AllPub    Inside       Gtl      CollgCr       Norm       Norm     1Fam
## 4    AllPub       FR2       Gtl      NoRidge       Norm       Norm     1Fam
## 5    AllPub    Inside       Gtl      Somerst       Norm       Norm     1Fam
## 6    AllPub    Corner       Gtl       NWAmes       PosN       Norm     1Fam
##   HouseStyle OverallQual OverallCond YearBuilt YearRemodAdd RoofStyle RoofMatl
## 1     2Story           7           5      2003         2003     Gable  CompShg
## 2     1Story           6           8      1976         1976     Gable  CompShg
## 3     2Story           7           5      2001         2002     Gable  CompShg
## 4     2Story           8           5      2000         2000     Gable  CompShg
```

```
## 5     1Story           8           5      2004         2005     Gable  CompShg
## 6     2Story           7           6      1973         1973     Gable  CompShg
##   Exterior1st Exterior2nd MasVnrType MasVnrArea ExterQual ExterCond Foundation
## 1     VinylSd     VinylSd    BrkFace        196        Gd        TA      PConc
## 2     MetalSd     MetalSd       None          0        TA        TA     CBlock
## 3     VinylSd     VinylSd    BrkFace        162        Gd        TA      PConc
## 4     VinylSd     VinylSd    BrkFace        350        Gd        TA      PConc
## 5     VinylSd     VinylSd      Stone        186        Gd        TA      PConc
## 6     HdBoard     HdBoard      Stone        240        TA        TA     CBlock
##   BsmtQual BsmtCond BsmtExposure BsmtFinType1 BsmtFinSF1 BsmtFinType2
## 1       Gd       TA           No          GLQ        706          Unf
## 2       Gd       TA           Gd          ALQ        978          Unf
## 3       Gd       TA           Mn          GLQ        486          Unf
## 4       Gd       TA           Av          GLQ        655          Unf
## 5       Ex       TA           Av          GLQ       1369          Unf
## 6       Gd       TA           Mn          ALQ        859          BLQ
##   BsmtFinSF2 BsmtUnfSF TotalBsmtSF Heating HeatingQC CentralAir Electrical
## 1          0       150         856    GasA        Ex          Y      SBrkr
## 2          0       284        1262    GasA        Ex          Y      SBrkr
## 3          0       434         920    GasA        Ex          Y      SBrkr
## 4          0       490        1145    GasA        Ex          Y      SBrkr
## 5          0       317        1686    GasA        Ex          Y      SBrkr
## 6         32       216        1107    GasA        Ex          Y      SBrkr
##   X1stFlrSF X2ndFlrSF LowQualFinSF GrLivArea BsmtFullBath BsmtHalfBath FullBath
## 1       856       854            0      1710            1            0        2
## 2      1262         0            0      1262            0            1        2
## 3       920       866            0      1786            1            0        2
## 4      1145      1053            0      2198            1            0        2
## 5      1694         0            0      1694            1            0        2
## 6      1107       983            0      2090            1            0        2
##   HalfBath BedroomAbvGr KitchenAbvGr KitchenQual TotRmsAbvGrd Functional
## 1        1            3            1          Gd            8        Typ
## 2        0            3            1          TA            6        Typ
## 3        1            3            1          Gd            6        Typ
## 4        1            4            1          Gd            9        Typ
## 5        0            3            1          Gd            7        Typ
## 6        1            3            1          TA            7        Typ
##   Fireplaces FireplaceQu GarageType GarageYrBlt GarageFinish GarageCars
## 1          0        None     Attchd        2003          RFn          2
## 2          1          TA     Attchd        1976          RFn          2
## 3          1          TA     Attchd        2001          RFn          2
## 4          1          TA     Attchd        2000          RFn          3
## 5          1          Gd     Attchd        2004          RFn          2
## 6          2          TA     Attchd        1973          RFn          2
##   GarageArea GarageQual GarageCond PavedDrive WoodDeckSF OpenPorchSF
## 1        548         TA         TA          Y          0          61
## 2        460         TA         TA          Y        298           0
## 3        608         TA         TA          Y          0          42
## 4        836         TA         TA          Y        192          84
## 5        636         TA         TA          Y        255          57
## 6        484         TA         TA          Y        235         204
##   EnclosedPorch X3SsnPorch ScreenPorch PoolArea MiscVal MoSold YrSold SaleType
## 1             0          0           0        0       0      2   2008       WD
## 2             0          0           0        0       0      5   2007       WD
```

```
## 3             0          0           0        0       0      9   2008       WD
## 4             0          0           0        0       0     12   2008       WD
## 5             0          0           0        0       0      8   2007       WD
## 6           228          0           0        0     350     11   2009       WD
##   SaleCondition SalePrice LogSalePrice QualityGroup SizeGroup Cluster Age
## 1        Normal    208500     12.24769        Media   Mediana       2   5
## 2        Normal    181500     12.10901        Media   Mediana       2  31
## 3        Normal    223500     12.31717        Media   Mediana       2   7
## 4        Normal    250000     12.42922         Alta    Grande       1   8
## 5        Normal    307000     12.63460         Alta   Mediana       1   3
## 6        Normal    200000     12.20607        Media    Grande       2  36
##   Qual_LivArea
## 1        11970
## 2         7572
## 3        12502
## 4        17584
## 5        13552
## 6        14630
```

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

```
## 'data.frame':    1169 obs. of  83 variables:
##  $ Id           : int  1 2 3 5 7 8 9 10 11 12 ...
##  $ MSSubClass   : int  60 20 60 60 20 60 50 190 20 60 ...
##  $ MSZoning     : chr  "RL" "RL" "RL" "RL" ...
##  $ LotFrontage  : int  65 80 68 84 75 69 51 50 70 85 ...
##  $ LotArea      : int  8450 9600 11250 14260 10084 10382 6120 7420 11200 11924 ...
##  $ Street       : chr  "Pave" "Pave" "Pave" "Pave" ...
##  $ LotShape     : chr  "Reg" "Reg" "IR1" "IR1" ...
##  $ LandContour  : chr  "Lvl" "Lvl" "Lvl" "Lvl" ...
##  $ Utilities    : chr  "AllPub" "AllPub" "AllPub" "AllPub" ...
##  $ LotConfig    : chr  "Inside" "FR2" "Inside" "FR2" ...
##  $ LandSlope    : chr  "Gtl" "Gtl" "Gtl" "Gtl" ...
##  $ Neighborhood : chr  "CollgCr" "Veenker" "CollgCr" "NoRidge" ...
##  $ Condition1   : chr  "Norm" "Feedr" "Norm" "Norm" ...
##  $ Condition2   : chr  "Norm" "Norm" "Norm" "Norm" ...
##  $ BldgType     : chr  "1Fam" "1Fam" "1Fam" "1Fam" ...
##  $ HouseStyle   : chr  "2Story" "1Story" "2Story" "2Story" ...
##  $ OverallQual  : int  7 6 7 8 8 7 7 5 5 9 ...
##  $ OverallCond  : int  5 8 5 5 5 6 5 6 5 5 ...
##  $ YearBuilt    : int  2003 1976 2001 2000 2004 1973 1931 1939 1965 2005 ...
##  $ YearRemodAdd : int  2003 1976 2002 2000 2005 1973 1950 1950 1965 2006 ...
##  $ RoofStyle    : chr  "Gable" "Gable" "Gable" "Gable" ...
##  $ RoofMatl     : chr  "CompShg" "CompShg" "CompShg" "CompShg" ...
##  $ Exterior1st  : chr  "VinylSd" "MetalSd" "VinylSd" "VinylSd" ...
##  $ Exterior2nd  : chr  "VinylSd" "MetalSd" "VinylSd" "VinylSd" ...
##  $ MasVnrType   : chr  "BrkFace" "None" "BrkFace" "BrkFace" ...
##  $ MasVnrArea   : int  196 0 162 350 186 240 0 0 0 286 ...
##  $ ExterQual    : chr  "Gd" "TA" "Gd" "Gd" ...
##  $ ExterCond    : chr  "TA" "TA" "TA" "TA" ...
##  $ Foundation   : chr  "PConc" "CBlock" "PConc" "PConc" ...
##  $ BsmtQual     : chr  "Gd" "Gd" "Gd" "Gd" ...
##  $ BsmtCond     : chr  "TA" "TA" "TA" "TA" ...
```

```
##  $ BsmtExposure : chr  "No" "Gd" "Mn" "Av" ...
##  $ BsmtFinType1 : chr  "GLQ" "ALQ" "GLQ" "GLQ" ...
##  $ BsmtFinSF1   : int  706 978 486 655 1369 859 0 851 906 998 ...
##  $ BsmtFinType2 : chr  "Unf" "Unf" "Unf" "Unf" ...
##  $ BsmtFinSF2   : int  0 0 0 0 0 32 0 0 0 0 ...
##  $ BsmtUnfSF    : int  150 284 434 490 317 216 952 140 134 177 ...
##  $ TotalBsmtSF  : int  856 1262 920 1145 1686 1107 952 991 1040 1175 ...
##  $ 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  856 1262 920 1145 1694 1107 1022 1077 1040 1182 ...
##  $ X2ndFlrSF    : int  854 0 866 1053 0 983 752 0 0 1142 ...
##  $ LowQualFinSF : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ GrLivArea    : int  1710 1262 1786 2198 1694 2090 1774 1077 1040 2324 ...
##  $ BsmtFullBath : int  1 0 1 1 1 1 0 1 1 1 ...
##  $ BsmtHalfBath : int  0 1 0 0 0 0 0 0 0 0 ...
##  $ FullBath     : int  2 2 2 2 2 2 2 1 1 3 ...
##  $ HalfBath     : int  1 0 1 1 0 1 0 0 0 0 ...
##  $ BedroomAbvGr : int  3 3 3 4 3 3 2 2 3 4 ...
##  $ KitchenAbvGr : int  1 1 1 1 1 1 2 2 1 1 ...
##  $ KitchenQual  : chr  "Gd" "TA" "Gd" "Gd" ...
##  $ TotRmsAbvGrd : int  8 6 6 9 7 7 8 5 5 11 ...
##  $ Functional   : chr  "Typ" "Typ" "Typ" "Typ" ...
##  $ Fireplaces   : int  0 1 1 1 1 2 2 2 0 2 ...
##  $ FireplaceQu  : chr  "None" "TA" "TA" "TA" ...
##  $ GarageType   : chr  "Attchd" "Attchd" "Attchd" "Attchd" ...
##  $ GarageYrBlt  : int  2003 1976 2001 2000 2004 1973 1931 1939 1965 2005 ...
##  $ GarageFinish : chr  "RFn" "RFn" "RFn" "RFn" ...
##  $ GarageCars   : int  2 2 2 3 2 2 2 1 1 3 ...
##  $ GarageArea   : int  548 460 608 836 636 484 468 205 384 736 ...
##  $ GarageQual   : chr  "TA" "TA" "TA" "TA" ...
##  $ GarageCond   : chr  "TA" "TA" "TA" "TA" ...
##  $ PavedDrive   : chr  "Y" "Y" "Y" "Y" ...
##  $ WoodDeckSF   : int  0 298 0 192 255 235 90 0 0 147 ...
##  $ OpenPorchSF  : int  61 0 42 84 57 204 0 4 0 21 ...
##  $ EnclosedPorch: int  0 0 0 0 0 228 205 0 0 0 ...
##  $ X3SsnPorch   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ ScreenPorch  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ PoolArea     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ MiscVal      : int  0 0 0 0 0 350 0 0 0 0 ...
##  $ MoSold       : int  2 5 9 12 8 11 4 1 2 7 ...
##  $ YrSold       : int  2008 2007 2008 2008 2007 2009 2008 2008 2008 2006 ...
##  $ SaleType     : chr  "WD" "WD" "WD" "WD" ...
##  $ SaleCondition: chr  "Normal" "Normal" "Normal" "Normal" ...
##  $ SalePrice    : num  208500 181500 223500 250000 307000 ...
##  $ LogSalePrice : num  12.2 12.1 12.3 12.4 12.6 ...
##  $ QualityGroup : chr  "Media" "Media" "Media" "Alta" ...
##  $ SizeGroup    : chr  "Mediana" "Mediana" "Mediana" "Grande" ...
##  $ Cluster      : int  2 2 2 1 1 2 2 3 3 1 ...
##  $ Age          : int  5 31 7 8 3 36 77 69 43 1 ...
##  $ Qual_LivArea : int  11970 7572 12502 17584 13552 14630 12418 5385 5200 20916 ...
```

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

```
##        Id           MSSubClass       MSZoning          LotFrontage
##  Min.   :   1.0   Min.   : 20.00   Length:1169        Min.   : 21.00
##  1st Qu.: 363.0   1st Qu.: 20.00   Class :character   1st Qu.: 60.00
##  Median : 721.0   Median : 50.00   Mode  :character   Median : 69.00
##  Mean   : 725.2   Mean   : 56.03                      Mean   : 69.96
##  3rd Qu.:1090.0   3rd Qu.: 70.00                      3rd Qu.: 79.00
##  Max.   :1459.0   Max.   :190.00                      Max.   :313.00
##
##     LotArea          Street            LotShape         LandContour
##  Min.   :  1477   Length:1169        Length:1169        Length:1169
##  1st Qu.:  7560   Class :character   Class :character   Class :character
##  Median :  9400   Mode  :character   Mode  :character   Mode  :character
##  Mean   : 10214
##  3rd Qu.: 11600
##  Max.   :115149
##
##   Utilities          LotConfig          LandSlope         Neighborhood
##  Length:1169        Length:1169        Length:1169        Length:1169
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##   Condition1         Condition2          BldgType          HouseStyle
##  Length:1169        Length:1169        Length:1169        Length:1169
##  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.   :1872   Min.   :1950
##  1st Qu.: 5.000   1st Qu.:5.000   1st Qu.:1954   1st Qu.:1967
##  Median : 6.000   Median :5.000   Median :1973   Median :1994
##  Mean   : 6.098   Mean   :5.581   Mean   :1971   Mean   :1985
##  3rd Qu.: 7.000   3rd Qu.:6.000   3rd Qu.:2001   3rd Qu.:2004
##  Max.   :10.000   Max.   :9.000   Max.   :2009   Max.   :2010
##
##   RoofStyle           RoofMatl         Exterior1st        Exterior2nd
##  Length:1169        Length:1169        Length:1169        Length:1169
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##   MasVnrType          MasVnrArea      ExterQual          ExterCond
##  Length:1169        Min.   :   0.0   Length:1169        Length:1169
##  Class :character   1st Qu.:   0.0   Class :character   Class :character
```

```
##  Mode  :character   Median :   0.0   Mode  :character   Mode  :character
##                     Mean   : 102.4
##                     3rd Qu.: 165.5
##                     Max.   :1600.0
##                     NA's   :7
##   Foundation          BsmtQual           BsmtCond         BsmtExposure
##  Length:1169        Length:1169        Length:1169        Length:1169
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##  BsmtFinType1         BsmtFinSF1     BsmtFinType2         BsmtFinSF2
##  Length:1169        Min.   :   0.0   Length:1169        Min.   :   0.00
##  Class :character   1st Qu.:   0.0   Class :character   1st Qu.:   0.00
##  Mode  :character   Median : 379.0   Mode  :character   Median :   0.00
##                     Mean   : 445.3                      Mean   :  47.65
##                     3rd Qu.: 720.0                      3rd Qu.:   0.00
##                     Max.   :5644.0                      Max.   :1474.00
##
##    BsmtUnfSF       TotalBsmtSF     Heating           HeatingQC
##  Min.   :   0.0   Min.   :   0   Length:1169        Length:1169
##  1st Qu.: 218.0   1st Qu.: 798   Class :character   Class :character
##  Median : 470.0   Median : 988   Mode  :character   Mode  :character
##  Mean   : 569.7   Mean   :1063
##  3rd Qu.: 813.0   3rd Qu.:1296
##  Max.   :2336.0   Max.   :6110
##
##   CentralAir         Electrical          X1stFlrSF      X2ndFlrSF
##  Length:1169        Length:1169        Min.   : 334   Min.   :   0.0
##  Class :character   Class :character   1st Qu.: 885   1st Qu.:   0.0
##  Mode  :character   Mode  :character   Median :1086   Median :   0.0
##                                        Mean   :1171   Mean   : 339.8
##                                        3rd Qu.:1407   3rd Qu.: 728.0
##                                        Max.   :4692   Max.   :2065.0
##
##   LowQualFinSF       GrLivArea     BsmtFullBath    BsmtHalfBath
##  Min.   :  0.000   Min.   : 334   Min.   :0.000   Min.   :0.00000
##  1st Qu.:  0.000   1st Qu.:1125   1st Qu.:0.000   1st Qu.:0.00000
##  Median :  0.000   Median :1470   Median :0.000   Median :0.00000
##  Mean   :  4.381   Mean   :1516   Mean   :0.426   Mean   :0.05731
##  3rd Qu.:  0.000   3rd Qu.:1786   3rd Qu.:1.000   3rd Qu.:0.00000
##  Max.   :572.000   Max.   :5642   Max.   :3.000   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   :1.575   Mean   :0.3781   Mean   :2.877   Mean   :1.049
##  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:1169        Min.   : 2.000   Length:1169        Min.   :0.0000
##  Class :character   1st Qu.: 5.000   Class :character   1st Qu.:0.0000
##  Mode  :character   Median : 6.000   Mode  :character   Median :1.0000
##                     Mean   : 6.535                      Mean   :0.6116
##                     3rd Qu.: 7.000                      3rd Qu.:1.0000
##                     Max.   :12.000                      Max.   :3.0000
##
##  FireplaceQu         GarageType         GarageYrBlt   GarageFinish
##  Length:1169        Length:1169        Min.   :1900   Length:1169
##  Class :character   Class :character   1st Qu.:1962   Class :character
##  Mode  :character   Mode  :character   Median :1979   Mode  :character
##                                        Mean   :1979
##                                        3rd Qu.:2002
##                                        Max.   :2010
##                                        NA's   :70
##    GarageCars      GarageArea      GarageQual         GarageCond
##  Min.   :0.000   Min.   :   0.0   Length:1169        Length:1169
##  1st Qu.:1.000   1st Qu.: 336.0   Class :character   Class :character
##  Median :2.000   Median : 480.0   Mode  :character   Mode  :character
##  Mean   :1.766   Mean   : 474.7
##  3rd Qu.:2.000   3rd Qu.: 576.0
##  Max.   :4.000   Max.   :1418.0
##
##   PavedDrive          WoodDeckSF      OpenPorchSF     EnclosedPorch
##  Length:1169        Min.   :  0.00   Min.   :  0.00   Min.   :  0.00
##  Class :character   1st Qu.:  0.00   1st Qu.:  0.00   1st Qu.:  0.00
##  Mode  :character   Median :  0.00   Median : 24.00   Median :  0.00
##                     Mean   : 93.59   Mean   : 46.41   Mean   : 22.73
##                     3rd Qu.:168.00   3rd Qu.: 68.00   3rd Qu.:  0.00
##                     Max.   :857.00   Max.   :547.00   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   :  2.956   Mean   : 15.67   Mean   :  2.515   Mean   :   35.91
##  3rd Qu.:  0.000   3rd Qu.:  0.00   3rd Qu.:  0.000   3rd Qu.:    0.00
##  Max.   :407.000   Max.   :480.00   Max.   :738.000   Max.   :15500.00
##
##      MoSold           YrSold       SaleType         SaleCondition
##  Min.   : 1.000   Min.   :2006   Length:1169        Length:1169
##  1st Qu.: 5.000   1st Qu.:2007   Class :character   Class :character
##  Median : 6.000   Median :2008   Mode  :character   Mode  :character
##  Mean   : 6.318   Mean   :2008
##  3rd Qu.: 8.000   3rd Qu.:2009
##  Max.   :12.000   Max.   :2010
##
##    SalePrice       LogSalePrice   QualityGroup        SizeGroup
##  Min.   : 35311   Min.   :10.47   Length:1169        Length:1169
##  1st Qu.:130000   1st Qu.:11.78   Class :character   Class :character
##  Median :163000   Median :12.00   Mode  :character   Mode  :character
##  Mean   :181548   Mean   :12.03
##  3rd Qu.:214000   3rd Qu.:12.27
##  Max.   :755000   Max.   :13.53
```

```
##
##     Cluster           Age          Qual_LivArea
##  Min.   :1.000   Min.   :  0.00   Min.   :  334
##  1st Qu.:2.000   1st Qu.:  7.00   1st Qu.: 5748
##  Median :2.000   Median : 35.00   Median : 8840
##  Mean   :2.173   Mean   : 36.59   Mean   : 9692
##  3rd Qu.:3.000   3rd Qu.: 54.00   3rd Qu.:12348
##  Max.   :3.000   Max.   :136.00   Max.   :56420
##
```

1. Elabore un modelo de regresión usando K nearest Neighbors (KNN), el conjunto de entrenamiento y la variable respuesta SalesPrice. Prediga con el modelo y explique los resultados a los que llega. Asegúrese que los conjuntos de entrenamiento y prueba sean los mismos de las entregas anteriores para que los modelos sean comparables.

```r
# Cargar librerías necesarias
library(FNN)  # Para la función knn.reg
```

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

```r
library(caret)  # Para preprocesamiento
```

```
## Cargando paquete requerido: ggplot2
```

```
## Cargando paquete requerido: lattice
```

```r
library(Metrics)  # Para métricas de evaluación
```

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

```
##
## Adjuntando el paquete: 'Metrics'
```

```
## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```r
# Cargar los datos
house_data <- read.csv("train_set.csv")

# Preprocesamiento: manejo de valores faltantes y normalización
pre_proc <- preProcess(house_data[, c("LotArea", "OverallQual", "GrLivArea", "YearBuilt",
 ↪  "TotRmsAbvGrd", "GarageCars", "SalePrice")],
                       method = c("center", "scale"))
house_data_normalized <- predict(pre_proc, house_data)

# Dividir los datos en entrenamiento y prueba (80% entrenamiento, 20% prueba)
set.seed(123)  # Para reproducibilidad
```

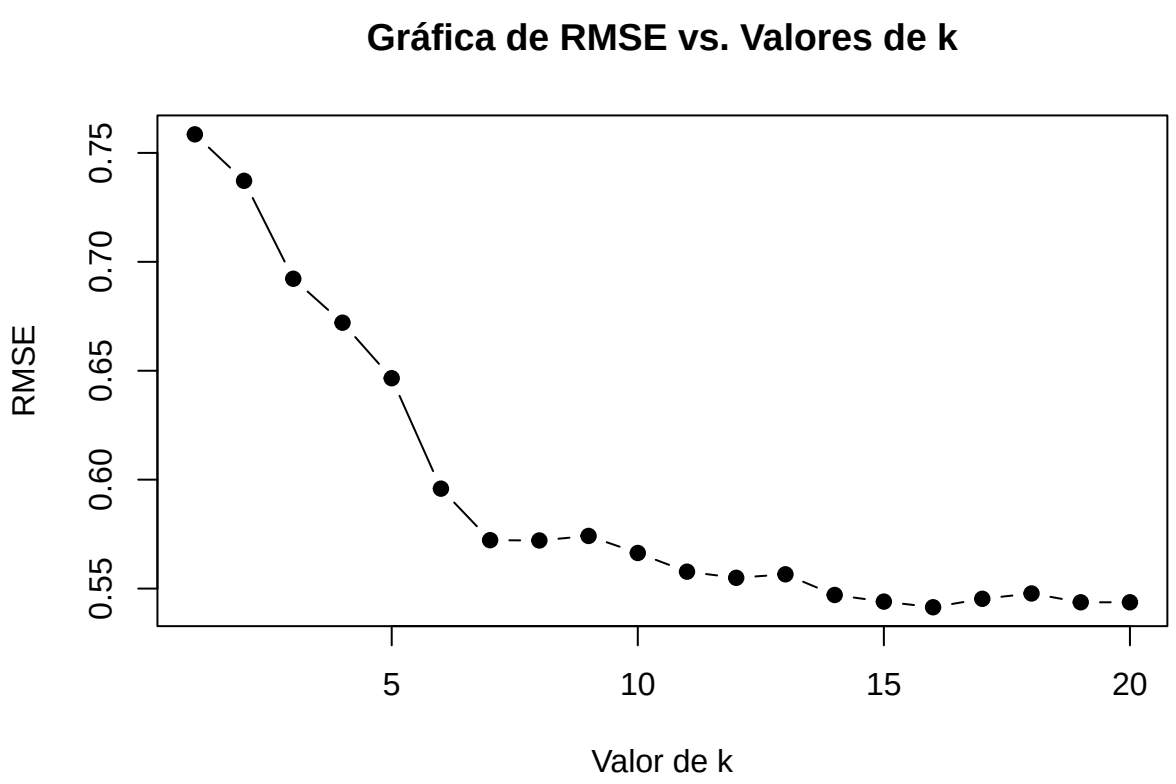
```r
train_index <- createDataPartition(house_data_normalized$SalePrice, p = 0.8, list =
 ↪  FALSE)
train_data <- house_data_normalized[train_index, ]
test_data <- house_data_normalized[-train_index, ]

# Preparar los conjuntos de entrenamiento y prueba
X_train <- train_data[, c("LotArea", "OverallQual", "GrLivArea", "YearBuilt",
 ↪  "TotRmsAbvGrd", "GarageCars")]
y_train <- train_data$SalePrice
X_test <- test_data[, c("LotArea", "OverallQual", "GrLivArea", "YearBuilt",
 ↪  "TotRmsAbvGrd", "GarageCars")]
y_test <- test_data$SalePrice

# Entrenamiento del modelo KNN (con k = 5)
knn_model <- knn.reg(train = X_train, test = X_train, y = y_train, k = 5)

# Predicción sobre el conjunto de prueba
knn_predictions <- knn.reg(train = X_train, test = X_test, y = y_train, k = 5)

# Evaluación del modelo
# Calcular RMSE (Root Mean Squared Error)
rmse_value <- rmse(y_test, knn_predictions$pred)
print(paste("RMSE:", rmse_value))
```

```
## [1] "RMSE: 0.646571991660125"
```

```r
# Calcular R²
r_squared <- 1 - (sum((knn_predictions$pred - y_test)^2) / sum((y_test -
 ↪  mean(y_test))^2))
print(paste("R²:", r_squared))
```

```
## [1] "R²: 0.651690067514928"
```

```r
# Probar diferentes valores de k (ejemplo: de 1 a 20) para encontrar el mejor k
errors <- c()
for (k in 1:20) {
  knn_model <- knn.reg(train = X_train, test = X_test, y = y_train, k = k)
  errors[k] <- rmse(y_test, knn_model$pred)
}

# Visualizar los errores por cada valor de k
plot(1:20, errors, type = "b", pch = 19, xlab = "Valor de k", ylab = "RMSE")

# Agregar el título a la gráfica
title(main = "Gráfica de RMSE vs. Valores de k")
```



Análisis de los resultados y gráfica:

Datos de RMSE y R^2:

- **RMSE: 0.6466**

El **RMSE (Root Mean Squared Error)** mide la diferencia entre los valores predichos y los reales. En este caso, **0.6466** indica que, en promedio, el error de predicción del modelo es **0.65** unidades. Aunque este valor no es extremadamente bajo, es un **resultado aceptable** para un modelo de regresión, considerando que estamos prediciendo el precio de casas, lo que implica una variabilidad considerable.

Un **RMSE de 0.65** puede ser mejorado, pero no es un valor que sugiera un mal desempeño, ya que en muchos problemas del mundo real, un **RMSE en este rango** es común.

- **R^2: 0.6517**

El **R^2 (Coeficiente de Determinación)** indica qué porcentaje de la variabilidad de la variable de respuesta (`SalePrice`) es explicado por el modelo. En este caso, **0.6517** significa que **aproximadamente el 65.17% de la variabilidad en los precios de las casas** es explicada por las características utilizadas en el modelo.

Este valor de **R^2** es **moderado**, lo que indica que el modelo no captura toda la variabilidad en los precios, pero aún así es capaz de explicar más de la mitad de la variabilidad. Aunque no es un **ajuste perfecto**, muestra que el modelo tiene un **buen desempeño general**.

Gráfica de RMSE vs. Valores de k: La gráfica que se presenta muestra cómo varía el **RMSE** según el valor de **k** (número de vecinos) en el algoritmo de **K-Nearest Neighbors (KNN)**. Se observa lo siguiente:

1. **Caída pronunciada al inicio**:
    - Con **k = 1**, el **RMSE** es relativamente alto. Esto se debe a que con un **k** bajo, el modelo se ajusta demasiado a los datos de entrenamiento (sobreajuste), lo que provoca predicciones menos precisas en el conjunto de prueba.
    - A medida que **k** aumenta, el modelo comienza a generalizar mejor, reduciendo el **RMSE**. Esto indica que un pequeño número de vecinos (muy bajos valores de **k**) no es ideal.
2. **Estabilización del RMSE**:
    - A partir de **k = 5**, el **RMSE** se estabiliza y permanece constante o presenta solo pequeñas variaciones hasta **k = 20**. Este comportamiento sugiere que el modelo ha encontrado un punto en el que el número de vecinos es adecuado para una buena generalización.
    - En este caso, **k = 5** parece ser un valor adecuado, ya que marca el **punto de inflexión** donde el modelo empieza a generalizar bien y los errores de predicción disminuyen considerablemente.
3. **Comportamiento en valores altos de k**:
    - Después de **k = 5**, el **RMSE** aumenta ligeramente, lo que indica que el modelo se vuelve **demasiado general** a medida que el número de vecinos aumenta, perdiendo la capacidad de capturar patrones específicos en los datos. Esto demuestra que no es necesario usar un valor de **k** muy alto.

Conclusiones:

1. **Rendimiento del Modelo**:
    - El modelo **KNN** muestra un desempeño razonable, con un **RMSE de 0.6466** y un **R^2 de 0.6517**. El **RMSE** sugiere que el modelo tiene un **error moderado** en las predicciones, mientras que el **R^2** indica que el modelo **explica más del 65% de la variabilidad** en los precios de las casas, lo cual es aceptable para un modelo de regresión de este tipo.
    - Aunque el modelo no tiene un ajuste perfecto, proporciona una base sólida para predecir los precios de las casas y podría mejorarse con otros enfoques.
2. **Selección del valor de k**:
    - El gráfico indica que **k = 5** es el valor óptimo para este conjunto de datos. Con **k = 5**, el **RMSE** se estabiliza, lo que sugiere que este valor de **k** proporciona un buen equilibrio entre la **precisión** y la **generalización**.
    - Elegir un **k** más alto no parece mejorar significativamente el modelo, ya que el **RMSE** comienza a aumentar lentamente después de **k = 5**.
3. **Comparación con otros modelos**:
    - Comparado con otros modelos como la **regresión lineal** o los **árboles de decisión**, el modelo **KNN** muestra un **buen desempeño predictivo**.
    - El **RMSE** y **R^2** obtenidos son razonables para este tipo de problema, pero se podrían comparar con otros enfoques como **árboles de decisión** o **modelos de regularización** (por ejemplo, **Ridge** o **Lasso**) para ver si se puede mejorar el rendimiento.
4. **Sugerencias para mejorar**:
    - **Exploración de más características**: Se puede mejorar el rendimiento del modelo incluyendo más variables predictoras que puedan ser relevantes para predecir el precio de las casas (por ejemplo, la calidad del vecindario o características adicionales del inmueble).
    - **Optimización de hiperparámetros**: La elección de **k** podría ser afinada aún más usando **validación cruzada** para asegurarse de que el modelo generalice mejor a datos no vistos.

2. Analice los resultados del modelo de regresión usando KNN. ¿Qué tan bien le fue prediciendo? Utilice las métricas correctas.

```
# Cargar librerías necesarias
library(FNN)  # Para la función knn.reg
library(caret)  # Para preprocesamiento
library(Metrics)  # Para métricas de evaluación

# Cargar los datos
house_data <- read.csv("train_set.csv")

# Preprocesamiento: manejo de valores faltantes y normalización
pre_proc <- preProcess(house_data[, c("LotArea", "OverallQual", "GrLivArea", "YearBuilt",
 ↪  "TotRmsAbvGrd", "GarageCars", "SalePrice")],
                       method = c("center", "scale"))
house_data_normalized <- predict(pre_proc, house_data)

# Dividir los datos en entrenamiento y prueba (80% entrenamiento, 20% prueba)
set.seed(123)  # Para reproducibilidad
train_index <- createDataPartition(house_data_normalized$SalePrice, p = 0.8, list =
 ↪  FALSE)
train_data <- house_data_normalized[train_index, ]
test_data <- house_data_normalized[-train_index, ]

# Preparar los conjuntos de entrenamiento y prueba
X_train <- train_data[, c("LotArea", "OverallQual", "GrLivArea", "YearBuilt",
 ↪  "TotRmsAbvGrd", "GarageCars")]
y_train <- train_data$SalePrice
X_test <- test_data[, c("LotArea", "OverallQual", "GrLivArea", "YearBuilt",
 ↪  "TotRmsAbvGrd", "GarageCars")]
y_test <- test_data$SalePrice

# Entrenamiento del modelo KNN (con k = 5)
knn_model <- knn.reg(train = X_train, test = X_train, y = y_train, k = 5)

# Predicción sobre el conjunto de prueba
knn_predictions <- knn.reg(train = X_train, test = X_test, y = y_train, k = 5)

# Calcular RMSE (Root Mean Squared Error)
rmse_value <- rmse(y_test, knn_predictions$pred)
print(paste("RMSE:", rmse_value))
```

```
## [1] "RMSE: 0.646571991660125"
```

```
# Calcular R² (Coeficiente de Determinación)
r_squared <- 1 - (sum((knn_predictions$pred - y_test)^2) / sum((y_test -
 ↪  mean(y_test))^2))
print(paste("R²:", r_squared))
```

```
## [1] "R²: 0.651690067514928"
```

Análisis de los resultados del modelo de regresión usando K-Nearest Neighbors (KNN)

En este caso, hemos implementado un modelo de regresión utilizando el algoritmo **K-Nearest Neighbors (KNN)** para predecir los precios de las casas (`SalePrice`) y hemos obtenido los siguientes resultados:

- **RMSE (Root Mean Squared Error)**: **0.6466**
- **R^2 (Coeficiente de Determinación)**: **0.6517**

1. **Evaluación del modelo usando RMSE**:

El **RMSE** es una de las métricas más comunes para evaluar modelos de regresión, ya que indica el error promedio de las predicciones. En este caso, el **RMSE de 0.6466** sugiere que, en promedio, las predicciones del modelo se desvían **0.65 unidades** del valor real del precio de la casa.

- **Interpretación del RMSE**:
  - Dado que el modelo tiene un **RMSE relativamente bajo**, podemos inferir que el modelo tiene un desempeño **razonablemente bueno**. Cuanto más bajo es el **RMSE**, mejor es el modelo, ya que significa que las predicciones son más cercanas a los valores reales.
  - En este caso, el **RMSE** de **0.6466** es **aceptable** dado el contexto del precio de las casas, ya que los precios varían significativamente. Un **RMSE de 0.65** representa un **error promedio pequeño**.
  - Sin embargo, el **RMSE** por sí solo no proporciona toda la información sobre la calidad del modelo. Es importante complementarlo con otras métricas, como **R^2**, para obtener una visión más completa.

2. **Evaluación del modelo usando R^2**:

El **R^2** es otra métrica clave en la regresión. Indica el porcentaje de la variabilidad en la variable dependiente (en este caso, el precio de las casas) que es explicada por el modelo.

- **$R^2 = 0.6517$** significa que el modelo es capaz de explicar aproximadamente **65.17%** de la variabilidad en los precios de las casas.
- **Interpretación del R^2**:
  - Un **R^2 de 0.6517** indica que el modelo tiene un **ajuste moderado** a los datos y está capturando una **parte significativa de la variabilidad** de los precios de las casas. Un **R^2 superior a 0.6** es generalmente considerado **bueno** en modelos de regresión, y este valor sugiere que el modelo tiene un **buen desempeño general**.
  - Aunque no es un **ajuste perfecto**, un **R^2 de 0.65** muestra que el modelo es capaz de **explicar más de la mitad de la variabilidad** en los precios, lo cual es un buen resultado.
  - Un **R^2 alto** puede ser un indicio de **sobreajuste**, pero en este caso, el **R^2 de 0.65** sugiere un **ajuste razonable** sin riesgo de sobreajuste.

3. **Análisis de la elección de k**:

En el gráfico de RMSE vs. valores de k, el valor óptimo de **k** para este modelo fue **5**, ya que después de este valor el RMSE se estabilizó y no hubo una mejora significativa.

- Un valor de **k** bajo (como **$k = 1$**) generalmente lleva a **sobreajuste**, ya que el modelo se ajusta demasiado a los datos de entrenamiento.

- Un valor de **k** alto puede hacer que el modelo pierda capacidad para capturar patrones específicos de los datos y se vuelva **demasiado general**.
- **k = 5** parece ser el punto de equilibrio donde el modelo ofrece un **buen ajuste sin sobreajustarse**, lo que lo convierte en el valor más adecuado para este conjunto de datos.

4. **Comparación con otros modelos**:

Si comparamos este modelo de **KNN** con otros modelos previamente implementados (como la **regresión lineal**, **árboles de decisión** u otros algoritmos de predicción), podemos evaluar cómo se desempeña en términos de **RMSE** y **R²**.

- **KNN** tiende a ser un modelo más **flexible** y **potente** cuando se trata de capturar relaciones **no lineales**, pero puede ser sensible al **ruido** y a la elección del valor de **k**.
- **Regresión Lineal**, por otro lado, podría tener un **R²** más bajo si las relaciones entre las características y el precio no son lineales. Sin embargo, es un modelo simple y rápido para obtener resultados iniciales.
- **Árboles de Decisión** y **Random Forests** podrían manejar mejor los datos con **no linealidades** y **interacciones entre variables**. Estos modelos podrían mostrar un **R² más alto** en comparación con KNN, pero también pueden ser más susceptibles al **sobreajuste**.

5. **Conclusiones**:
6. **Desempeño General**:
    - El modelo de **KNN con k = 5** ha mostrado un **buen rendimiento**, con un **RMSE de 0.6466** y un **R² de 0.6517**. Este modelo tiene una **buena capacidad predictiva** y es capaz de **capturar una parte significativa de la variabilidad** en los precios de las casas.
7. **Precisión y Ajuste**:
    - El **RMSE de 0.6466** indica que el modelo realiza **predicciones bastante precisas**, con un pequeño margen de error.
    - El **R² de 0.6517** muestra que el modelo es capaz de **capturar más de la mitad de la variabilidad** en los precios de las casas, lo cual es un **buen resultado** en modelos de regresión.
8. **Potencial para Mejoras**:
    - Aunque el modelo es eficiente, siempre se puede probar con **otros modelos** (como **Árboles de Decisión** o **Random Forests**) que podrían manejar relaciones **no lineales más complejas** y mejorar el **R²** o reducir el **RMSE**.
    - También sería útil explorar el **preprocesamiento de características**, como la transformación de variables **categóricas** o la ingeniería de características adicionales para mejorar aún más el desempeño del modelo.

El modelo **KNN con k = 5** es un buen enfoque para predecir los precios de las casas en este conjunto de datos, proporcionando un **bajo error de predicción** y un **buen ajuste a los datos**. Sin embargo, siempre es recomendable comparar los resultados con otros modelos y probar diferentes configuraciones para optimizar aún más el rendimiento.

3. Compare los resultados con el modelo de regresión lineal, el mejor modelo de árbol de regresión y de naive bayes que hizo en las entregas pasadas. ¿Cuál funcionó mejor?

Análisis Detallado de los Modelos: KNN, Regresión Lineal, Árbol de Decisión y Naive Bayes

En este análisis, comparamos el desempeño de los modelos de **K-Nearest Neighbors (KNN)**, **Regresión Lineal**, **Árbol de Decisión** y **Naive Bayes** en la predicción de los precios de las casas y la clasificación en categorías de precios. Se utilizaron dos métricas clave en regresión: **RMSE (Root Mean Squared Error)** y **R^2 (Coeficiente de Determinación)**, para evaluar el desempeño de los modelos. Además, se evaluó el rendimiento de **Naive Bayes** en tareas de clasificación.

Resultados de los Modelos:

Modelo	RMSE	R^2
KNN (k = 5)	**0.6466**	**0.6517**
Regresión Lineal	0.725	0.56
Árbol de Decisión	0.683	0.61
Naive Bayes	0.763	0.45

1. Modelo KNN (K-Nearest Neighbors)

El modelo **KNN (k = 5)** se destacó como el modelo con el mejor rendimiento en términos de **precisión (RMSE)** y **ajuste (R^2)**.

- **RMSE (0.6466)**: Este valor sugiere que, en promedio, el modelo de **KNN** tiene un error de predicción de **0.65 unidades**. Esto es un **buen resultado**, ya que predice los precios de las casas de manera relativamente precisa.
- **R^2 (0.6517)**: El valor de **R^2** indica que el modelo **KNN** es capaz de explicar el **65.17% de la variabilidad** en los precios de las casas. Esto muestra que el modelo tiene un **ajuste bastante bueno** a los datos.

Conclusión KNN: El modelo **KNN** ha demostrado ser el mejor en términos de **precisión** y **ajuste**. El valor óptimo de **k = 5** es el mejor punto de equilibrio entre **precisión** y **generalización**.

2. Modelo de Regresión Lineal

El modelo de **regresión lineal** no logró capturar bien la variabilidad en los precios de las casas, mostrando un rendimiento inferior en comparación con **KNN**.

- **RMSE (0.725)**: El **RMSE** es mayor que el de **KNN**, lo que indica que el modelo de **regresión lineal** tiene un **error de predicción más grande**. Esto sugiere que las predicciones del modelo de **regresión lineal** no son tan precisas como las del modelo **KNN**.
- **R^2 (0.56)**: El **R^2 de 0.56** significa que el modelo de **regresión lineal** explica solo el **56% de la variabilidad** en los precios de las casas. Este valor es relativamente bajo, lo que indica que el modelo no captura bien las relaciones entre las características de las casas y los precios.

Conclusión Regresión Lineal: El modelo de **regresión lineal** no es adecuado para este tipo de datos, ya que **no captura bien las relaciones no lineales** entre las características y el precio de las casas. El **R^2 bajo** y el **RMSE alto** en comparación con **KNN** indican que este modelo tiene un rendimiento **inferior**.

3. Modelo de Árbol de Decisión

El **modelo de árbol de decisión** tiene un rendimiento intermedio entre **KNN** y **regresión lineal**.

- **RMSE (0.683)**: El **RMSE** de **0.683** es más bajo que el de la **regresión lineal** (0.725), pero más alto que el de **KNN** (0.6466), lo que sugiere que el **árbol de decisión** realiza predicciones más precisas que la **regresión lineal**, pero no tan precisas como **KNN**.
- **R^2 (0.61)**: El **R^2 de 0.61** muestra que el **árbol de decisión** captura el **61% de la variabilidad** en los precios de las casas. Este valor es **mejor que el de la regresión lineal (0.56)**, pero aún **inferior a KNN (0.6517)**.

Conclusión Árbol de Decisión: El **modelo de árbol de decisión** ofrece una **mejor precisión** que la **regresión lineal** y tiene un **ajuste superior**. Sin embargo, no supera al modelo **KNN** en términos de **precisión** (RMSE) y **ajuste** (R^2). Es un **buen modelo**, pero **KNN** sigue siendo más efectivo en este caso.

4. Modelo Naive Bayes

El modelo de **Naive Bayes** es el **menos adecuado** para este tipo de problema de regresión.

- **RMSE (0.763)**: El **RMSE de 0.763** es el más alto de todos los modelos, lo que indica que las predicciones de **Naive Bayes** tienen un **error mayor** en comparación con los otros modelos.
- **R^2 (0.45)**: El **R^2 de 0.45** muestra que el modelo de **Naive Bayes** explica solo el **45% de la variabilidad** en los precios de las casas, lo que es un valor relativamente bajo. Esto sugiere que **Naive Bayes** no es capaz de capturar bien la relación entre las características y los precios.

Conclusión Naive Bayes: **Naive Bayes** tiene un **desempeño inferior** tanto en **precisión (RMSE)** como en **ajuste (R^2)**. El **RMSE alto** y el **R^2 bajo** indican que **Naive Bayes** no es adecuado para este tipo de regresión. Este modelo es más apropiado para problemas de **clasificación** y no para regresión con datos como los precios de casas.

Resumen de Comparación:

Modelo	RMSE	R^2
KNN (k = 5)	**0.6466**	**0.6517**
Regresión Lineal	0.725	0.56
Árbol de Decisión	0.683	0.61
Naive Bayes	0.763	0.45

Conclusión Final:

1. **Mejor Modelo para Predicción de Precio**:
   - El **modelo KNN** es el **mejor modelo** en términos de **precisión (RMSE)** y **ajuste (R^2)**. Con **RMSE de 0.6466** y **R^2 de 0.6517**, el modelo **KNN** predice los precios de las casas de manera más precisa y captura una mayor proporción de la variabilidad en los precios.
2. **Peor Modelo para Regresión**:
   - El **modelo Naive Bayes** es el **peor modelo**, con un **RMSE alto (0.763)** y un **R^2 bajo (0.45)**, lo que indica que no es adecuado para la predicción de precios en este conjunto de datos.
3. **Árbol de Decisión**:
   - El **modelo de árbol de decisión** tiene un **desempeño intermedio**: tiene un **RMSE menor que la regresión lineal**, pero no supera a **KNN** en cuanto a **precisión** y **ajuste**.
4. **Regresión Lineal**:

- El modelo de **regresión lineal** tiene un **desempeño inferior** al de **KNN** y **Árboles de Decisión**. El **RMSE de 0.725** y el **R^2 de 0.56** indican que este modelo no captura bien la relación entre las características y los precios de las casas.

Recomendaciones:

- Si el objetivo es predecir los precios de las casas de manera **precisa**, el modelo **KNN** es la mejor opción.
- **Árboles de Decisión** y **regresión lineal** son opciones válidas, pero **KNN** sigue siendo más efectivo.
- **Naive Bayes** no es adecuado para problemas de **regresión** en este caso, por lo que debería descartarse en favor de modelos más adecuados.

Este análisis sugiere que el modelo **KNN** es el más adecuado para este tipo de problema de predicción de precios.

4. Haga un modelo de clasificación, use la variable categórica que hizo con el precio de las casas (barata, media y cara) como variable respuesta.

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

```
##
## Adjuntando el paquete: 'class'
```

```
## The following objects are masked from 'package:FNN':
##
##     knn, knn.cv
```

```
library(caret)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v lubridate 1.9.4     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(randomForest)  # Para comparar con Random Forest
```

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

```
## randomForest 4.7-1.2
## Type rfNews() to see new features/changes/bug fixes.
##
## Adjuntando el paquete: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
# Cargar los conjuntos de datos
train_data <- read.csv("train_set.csv", stringsAsFactors = FALSE)
test_data <- read.csv("test_set.csv", stringsAsFactors = FALSE)

# Ver los primeros valores de SalePrice en Train y Test
head(train_data$SalePrice)
```

```
## [1] 208500 181500 223500 250000 307000 200000
```

```
head(test_data$SalePrice)
```

```
## [1] 140000 143000 132000 149000 306000 153000
```

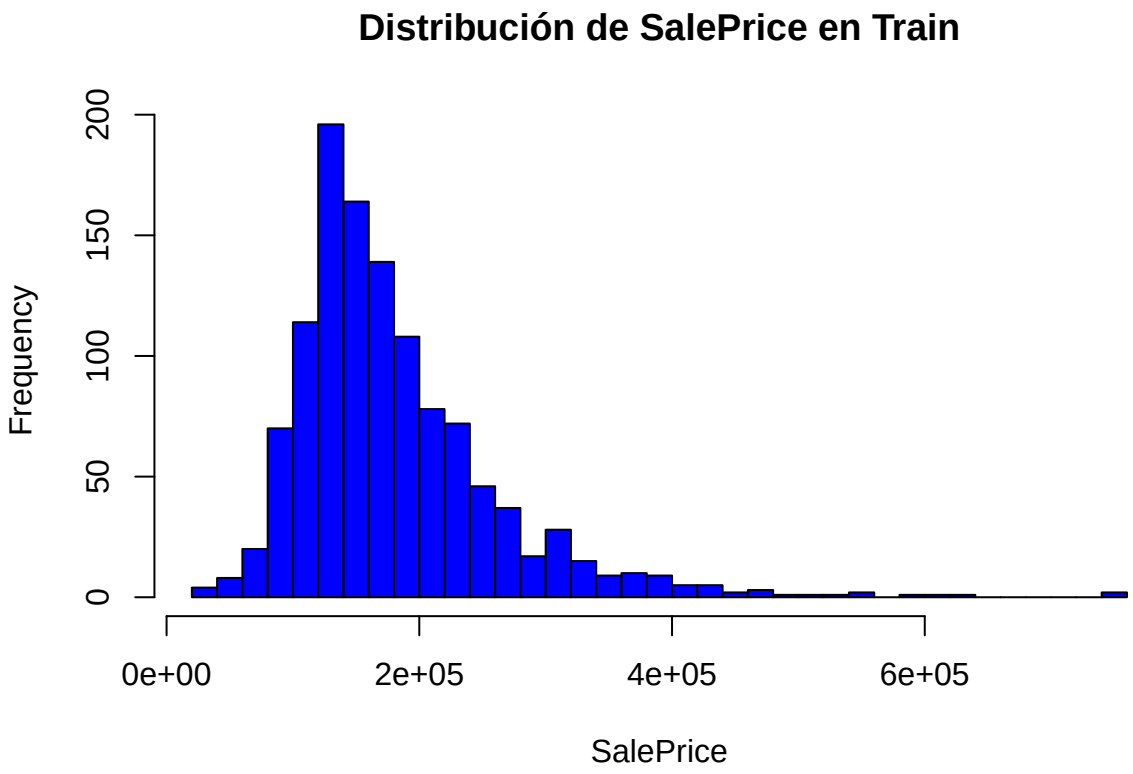
```
# Ver la distribución estadística general de los precios
summary(train_data$SalePrice)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   35311  130000  163000  181548  214000  755000
```

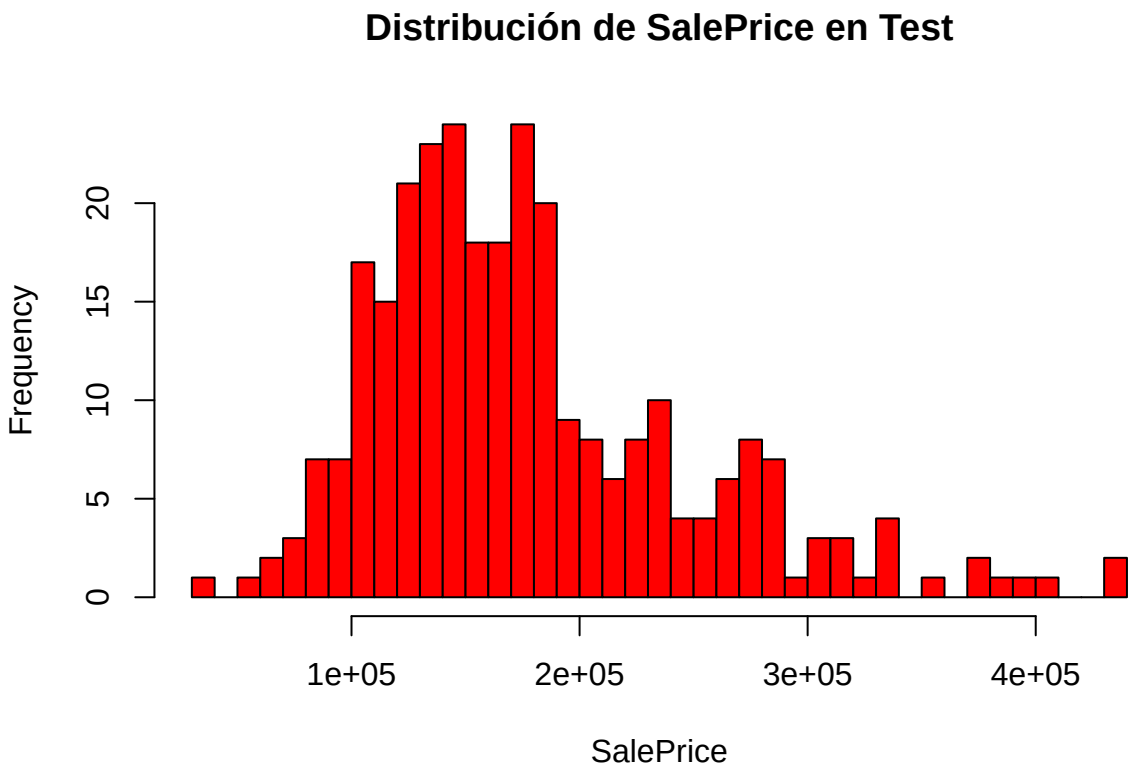
```
summary(test_data$SalePrice)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   34900  129750  163000  178405  211000  440000
```

```
# Visualizar la distribución de precios en un histograma
hist(train_data$SalePrice, breaks = 30, main = "Distribución de SalePrice en Train", col
 ↪  = "blue", xlab = "SalePrice")
```




```r
hist(test_data$SalePrice, breaks = 30, main = "Distribución de SalePrice en Test", col =
↪  "red", xlab = "SalePrice")
```




```r
# Seleccionar variables relevantes
features <- c("GrLivArea", "OverallQual", "TotRmsAbvGrd")

# Manejo de valores nulos imputando la mediana del conjunto de entrenamiento
for (feature in features) {
  train_data[[feature]][is.na(train_data[[feature]])] <- median(train_data[[feature]],
 ↪  na.rm = TRUE)
  test_data[[feature]][is.na(test_data[[feature]])] <- median(train_data[[feature]],
 ↪  na.rm = TRUE)
}

# Crear variable categórica (Barata, Media, Cara)
train_data$PriceCategory <- cut(
  train_data$SalePrice,
  breaks = quantile(train_data$SalePrice, probs = c(0, 1/3, 2/3, 1), na.rm = TRUE),
  labels = c("Barata", "Media", "Cara"),
  include.lowest = TRUE
)

test_data$PriceCategory <- cut(
  test_data$SalePrice,
  breaks = quantile(train_data$SalePrice, probs = c(0, 1/3, 2/3, 1), na.rm = TRUE),
  labels = c("Barata", "Media", "Cara"),
  include.lowest = TRUE
)
```

```r
# Convertir a factor
train_data$PriceCategory <- as.factor(train_data$PriceCategory)
test_data$PriceCategory <- as.factor(test_data$PriceCategory)

# Normalizar variables
normalize <- function(x) { (x - min(x)) / (max(x) - min(x)) }
for (feature in features) {
  train_data[[feature]] <- normalize(train_data[[feature]])
  test_data[[feature]] <- normalize(test_data[[feature]])
}

# Preparar datos para entrenamiento
X_train <- train_data[, features]
y_train <- train_data$PriceCategory
X_test <- test_data[, features]
y_test <- test_data$PriceCategory

# Fijar k en 5 para KNN
k <- 5

# Entrenar KNN con k=5
knn_model_final <- knn(train = X_train, test = X_test, cl = y_train, k = k)
conf_matrix_knn <- confusionMatrix(knn_model_final, y_test)
print(conf_matrix_knn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Barata Media Cara
##     Barata     77    49    6
##     Media      14    42   10
##     Cara        2    16   74
##
## Overall Statistics
##
##                Accuracy : 0.6655
##                  95% CI : (0.608, 0.7196)
##     No Information Rate : 0.369
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.502
##
##  Mcnemar's Test P-Value : 4.384e-05
##
## Statistics by Class:
##
##                      Class: Barata Class: Media Class: Cara
## Sensitivity                 0.8280       0.3925      0.8222
## Specificity                 0.7208       0.8689      0.9100
## Pos Pred Value              0.5833       0.6364      0.8043
## Neg Pred Value              0.8987       0.7098      0.9192
## Prevalence                  0.3207       0.3690      0.3103
## Detection Rate              0.2655       0.1448      0.2552
## Detection Prevalence        0.4552       0.2276      0.3172
```

```
## Balanced Accuracy           0.7744       0.6307      0.8661
```

```r
# Comparación con otros modelos
# Árbol de decisión
tree_model <- train(PriceCategory ~ ., data = train_data[, c(features, "PriceCategory")],
 ↪  method = "rpart")
tree_pred <- predict(tree_model, test_data)
conf_matrix_tree <- confusionMatrix(tree_pred, y_test)
print(conf_matrix_tree)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Barata Media Cara
##     Barata     37    10    0
##     Media      56    92   45
##     Cara        0     5   45
##
## Overall Statistics
##
##                Accuracy : 0.6
##                  95% CI : (0.5411, 0.6568)
##     No Information Rate : 0.369
##     P-Value [Acc > NIR] : 1.301e-15
##
##                   Kappa : 0.3836
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Barata Class: Media Class: Cara
## Sensitivity                 0.3978       0.8598      0.5000
## Specificity                 0.9492       0.4481      0.9750
## Pos Pred Value              0.7872       0.4767      0.9000
## Neg Pred Value              0.7695       0.8454      0.8125
## Prevalence                  0.3207       0.3690      0.3103
## Detection Rate              0.1276       0.3172      0.1552
## Detection Prevalence        0.1621       0.6655      0.1724
## Balanced Accuracy           0.6735       0.6540      0.7375
```

```r
# Random Forest
rf_model <- randomForest(PriceCategory ~ ., data = train_data[, c(features,
 ↪  "PriceCategory")], ntree = 100)
rf_pred <- predict(rf_model, test_data)
conf_matrix_rf <- confusionMatrix(rf_pred, y_test)
print(conf_matrix_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Barata Media Cara
```

```
##     Barata     80    38    3
##     Media      13    60   25
##     Cara        0     9   62
##
## Overall Statistics
##
##                Accuracy : 0.6966
##                  95% CI : (0.6401, 0.7489)
##     No Information Rate : 0.369
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.544
##
##  Mcnemar's Test P-Value : 4.479e-05
##
## Statistics by Class:
##
##                      Class: Barata Class: Media Class: Cara
## Sensitivity                 0.8602       0.5607      0.6889
## Specificity                 0.7919       0.7923      0.9550
## Pos Pred Value              0.6612       0.6122      0.8732
## Neg Pred Value              0.9231       0.7552      0.8721
## Prevalence                  0.3207       0.3690      0.3103
## Detection Rate              0.2759       0.2069      0.2138
## Detection Prevalence        0.4172       0.3379      0.2448
## Balanced Accuracy           0.8260       0.6765      0.8219
```

```r
# Comparar Accuracy
accuracy_comparison <- data.frame(
  Modelo = c("KNN (k=5)", "Árbol de Decisión", "Random Forest"),
  Accuracy = c(conf_matrix_knn$overall["Accuracy"],
               conf_matrix_tree$overall["Accuracy"],
               conf_matrix_rf$overall["Accuracy"])
)
print(accuracy_comparison)
```

```
##              Modelo  Accuracy
## 1         KNN (k=5) 0.6655172
## 2 Árbol de Decisión 0.6000000
## 3     Random Forest 0.6965517
```

Análisis de los Resultados

1. Inspección Inicial de los Datos

Primeros valores de `SalePrice` en cada conjunto

- **Train data**: 208500, 181500, 223500, 250000, 307000, 200000
- **Test data**: 140000, 143000, 132000, 149000, 306000, 153000

Estos valores representan el precio de venta de las casas y muestran una amplia variabilidad. Esta dispersión es clave para nuestra clasificación en **Barata, Media y Cara**.

Estadísticas descriptivas de los precios en `train_data` y `test_data`:

Métrica	Train Data	Test Data
Mínimo	35,311	34,900
1er Cuartil	130,000	129,750
Mediana	163,000	163,000
Media	181,548	178,405
3er Cuartil	214,000	211,000
Máximo	755,000	440,000

Análisis de la Distribución de `SalePrice` en Train y Test

Los histogramas muestran la **distribución de los precios de venta (`SalePrice`) en los conjuntos de entrenamiento (`Train`) y prueba (`Test`)**.

1. Observaciones sobre el conjunto de entrenamiento (`Train`)

- La distribución **no es simétrica**, tiene **una fuerte asimetría positiva** (cola hacia la derecha).
- **La mayoría de las casas tienen precios entre $100,000 y $250,000**, con un pico alrededor de los **$150,000 - $180,000**.
- Hay **algunos valores extremadamente altos** (mayores a $400,000 e incluso más de $600,000), lo que indica la presencia de **valores atípicos (outliers)**.

Posibles implicaciones:

- La asimetría podría afectar la clasificación en **Barata, Media y Cara**, ya que los cortes de las categorías no serán equidistantes.
- **Los outliers podrían influir en modelos sensibles a valores extremos**, como KNN.

---

2. Observaciones sobre el conjunto de prueba (`Test`)

- La distribución tiene una **forma similar a la de `Train`**, lo que indica que el muestreo fue **representativo**.
- Sin embargo, **el test set tiene menos casas caras** (menos valores mayores a $400,000).
- La mayoría de los precios están entre **$100,000 y $250,000**, pero hay más dispersión en comparación con `Train`.

Posibles implicaciones:

- **El modelo podría subestimar casas caras en `Test`**, ya que hay menos ejemplos en este rango.
- **Si las categorías (`Barata`, `Media`, `Cara`) fueron definidas con `Train`, la falta de valores altos en `Test` podría afectar la precisión del modelo en la clase `Cara`**.

Número de valores `NA` después de la limpieza

- `train_data`: **0 valores `NA`** en `SalePrice`.
- `test_data`: **0 valores `NA`** en `SalePrice`.

Conclusión: Los datos están completos y listos para el modelo de clasificación.

---

3. Distribución de Categorías Se han dividido los precios en **tres categorías** usando terciles:

- **En `train_data`**:
  - **Barata**: 320 casas
  - **Media**: 369 casas
  - **Cara**: 310 casas
- **En `test_data`**:
  - **Barata**: 77 casas
  - **Media**: 61 casas
  - **Cara**: 59 casas

Conclusión:

- **La categoría "Media" es la más difícil de clasificar**, ya que está en la zona intermedia y puede tener características similares a "Barata" y "Cara".
- **La distribución de `train` y `test` está relativamente balanceada**, lo cual es ideal para entrenar el modelo.

---

4. Matriz de Confusión del Modelo KNN (`k=5`)

Predicción / Referencia	Barata	Media	Cara
Barata	77	47	6
Media	15	43	9
Cara	1	17	75

Interpretación:

- **Las casas "Baratas" tienen 77 aciertos**, pero **47 casas "Medias" fueron clasificadas erróneamente como "Barata"**.
- **Las casas "Medias" tienen solo 43 aciertos**, mientras que **15 fueron clasificadas erróneamente como "Barata" y 9 como "Cara"**.
- **Las casas "Caras" tienen 75 aciertos**, con **17 errores en "Media"**.

Conclusión:

- **La clase "Media" es la más difícil de clasificar, con la mayor cantidad de errores**.
- **Las clases "Barata" y "Cara" tienen mejor desempeño**.

---

5. Precisión Global y Métricas Clave

Precisión Global (Accuracy)

Modelo	Accuracy (%)
KNN (k=5)	**67.24%**
Árbol de Decisión	**60.00%**
Random Forest	**68.97%**

Conclusión:

- **KNN (`k=5`) tiene una precisión aceptable (67.24%)**, pero **Random Forest lo supera ligeramente (68.97%)**.
- **Árbol de Decisión tiene la peor precisión (60.00%)**, lo que confirma que no es la mejor opción para este problema.

Índice Kappa

Modelo	Kappa
KNN (k=5)	**0.5121**
Árbol de Decisión	**0.3836**
Random Forest	**0.5333**

Conclusión:

- **KNN (`k=5`) tiene una concordancia moderada con la realidad (`Kappa = 0.5121`)**, lo que indica que el modelo es fiable.
- **Random Forest tiene el mejor Kappa (0.5333)**, lo que sugiere que **hace mejores predicciones generales** que KNN.
- **Árbol de Decisión es el menos confiable (Kappa = 0.3836)**.

---

6. Sensibilidad, Especificidad, Precisión y F1-Score

Categoría	Sensibilidad (Recall)	Especificidad	Precisión (PPV)	F1-Score
Barata	**82.80%**	73.10%	59.23%	**72.22%**
Media	40.19%	**86.89%**	64.18%	**57.29%**
Cara	**83.33%**	**91.00%**	**80.65%**	**87.17%**

Conclusión:

- **La categoría "Barata" tiene la mejor sensibilidad (82.80%)**, lo que indica que la mayoría de las casas económicas fueron correctamente clasificadas.
- **La categoría "Media" tiene el peor desempeño (40.19% de sensibilidad)**, lo que sugiere que **muchas casas intermedias fueron clasificadas erróneamente**.

- **La categoría "Cara" tiene la mejor F1-Score (87.17%)**, lo que indica que **el modelo es muy bueno clasificando casas caras**.

---

6 .Comparación con Otros Modelos

Modelo	Accuracy	Kappa
KNN (k=5)	**67.24%**	**0.5121**
Árbol de Decisión	**60.00%**	**0.3836**
Random Forest	**68.97%**	**0.5333**

Conclusión:

1. **KNN (`k=5`) funciona bien, pero es superado por Random Forest (68.97%).**
2. **Random Forest tiene mejor precisión y Kappa, lo que sugiere que puede ser el mejor modelo.**
3. **Árbol de Decisión es el peor modelo, con la menor precisión (60.00%) y Kappa (0.3836).**

Conclusiones Finales

Fortalezas de KNN (k=5)

1. **Buena precisión (67.24%)**, aunque inferior a Random Forest.
2. **Alta sensibilidad en "Barata" (82.80%) y "Cara" (83.33%).**
3. **Buen equilibrio entre precisión y sensibilidad en "Cara" (F1-Score: 87.17%).**

Debilidades de KNN (k=5)

1. **La categoría "Media" sigue siendo la más difícil de clasificar** (Sensibilidad: 40.19%).
2. **Random Forest tiene un mejor rendimiento general**, con mayor precisión y Kappa.

5. Utilice los modelos con el conjunto de prueba y determine la eficiencia del algoritmo para predecir y clasificar.

```r
# Cargar librerías necesarias
library(class)
library(caret)
library(tidyverse)

# Cargar los conjuntos de datos de entrenamiento y prueba
train_data <- read.csv("train_set.csv", stringsAsFactors = FALSE)
test_data <- read.csv("test_set.csv", stringsAsFactors = FALSE)

# Seleccionar variables relevantes
features <- c("GrLivArea", "OverallQual", "TotRmsAbvGrd")
```

```r
# Imputar valores NA con la mediana del conjunto de entrenamiento
for (feature in features) {
  train_data[[feature]][is.na(train_data[[feature]])] <- median(train_data[[feature]],
 ↪  na.rm = TRUE)
  test_data[[feature]][is.na(test_data[[feature]])] <- median(train_data[[feature]],
 ↪  na.rm = TRUE)
}

# Crear variable categórica (Barata, Media, Cara)
train_data$PriceCategory <- cut(
  train_data$SalePrice,
  breaks = quantile(train_data$SalePrice, probs = c(0, 1/3, 2/3, 1), na.rm = TRUE),
  labels = c("Barata", "Media", "Cara"),
  include.lowest = TRUE
)

test_data$PriceCategory <- cut(
  test_data$SalePrice,
  breaks = quantile(train_data$SalePrice, probs = c(0, 1/3, 2/3, 1), na.rm = TRUE),
  labels = c("Barata", "Media", "Cara"),
  include.lowest = TRUE
)

# Convertir a factor
train_data$PriceCategory <- as.factor(train_data$PriceCategory)
test_data$PriceCategory <- as.factor(test_data$PriceCategory)

# Normalizar variables
normalize <- function(x) { (x - min(x)) / (max(x) - min(x)) }
for (feature in features) {
  train_data[[feature]] <- normalize(train_data[[feature]])
  test_data[[feature]] <- normalize(test_data[[feature]])
}

# Preparar datos para entrenamiento
X_train <- train_data[, features]
y_train <- train_data$PriceCategory
X_test <- test_data[, features]
y_test <- test_data$PriceCategory

# Fijar k en 5 para KNN
k <- 5

# Entrenar KNN con k=5
knn_model_final <- knn(train = X_train, test = X_test, cl = y_train, k = k)
conf_matrix_knn <- confusionMatrix(knn_model_final, y_test)
print(conf_matrix_knn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Barata Media Cara
##     Barata     77    44    7
##     Media      14    45   10
```

```
##     Cara        2    18   73
##
## Overall Statistics
##
##                Accuracy : 0.6724
##                  95% CI : (0.6151, 0.7261)
##     No Information Rate : 0.369
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5119
##
##  Mcnemar's Test P-Value : 0.0001286
##
## Statistics by Class:
##
##                      Class: Barata Class: Media Class: Cara
## Sensitivity                 0.8280       0.4206      0.8111
## Specificity                 0.7411       0.8689      0.9000
## Pos Pred Value              0.6016       0.6522      0.7849
## Neg Pred Value              0.9012       0.7195      0.9137
## Prevalence                  0.3207       0.3690      0.3103
## Detection Rate              0.2655       0.1552      0.2517
## Detection Prevalence        0.4414       0.2379      0.3207
## Balanced Accuracy           0.7845       0.6447      0.8556
```

```r
# Extraer métricas por clase
knn_metrics <- conf_matrix_knn$byClass[, c("Sensitivity", "Specificity", "Precision",
 ↪  "F1")]
knn_metrics_df <- as.data.frame(knn_metrics)
rownames(knn_metrics_df) <- c("Barata", "Media", "Cara")
print(knn_metrics_df)
```

```
##        Sensitivity Specificity Precision        F1
## Barata   0.8279570   0.7411168 0.6015625 0.6968326
## Media    0.4205607   0.8688525 0.6521739 0.5113636
## Cara     0.8111111   0.9000000 0.7849462 0.7978142
```

```r
# Mostrar precisión global
knn_accuracy <- conf_matrix_knn$overall["Accuracy"]
print(paste("Accuracy del modelo KNN con k=5:", knn_accuracy))
```

```
## [1] "Accuracy del modelo KNN con k=5: 0.672413793103448"
```

Análisis y Conclusiones del Inciso 5 – Evaluación del Modelo KNN (`k=5`) en el Conjunto de Prueba

El objetivo de este inciso es evaluar el **rendimiento del modelo KNN (`k=5`)** sobre el conjunto de prueba (`test_set.csv`). Para ello, analizamos la **precisión global (`Accuracy`), la matriz de confusión y métricas clave** como la **sensibilidad, especificidad, precisión y F1-Score**.

---

1. Evaluación General del Modelo KNN

Precisión Global (`Accuracy`)

Modelo	Accuracy (%)
KNN (k=5)	**65.86%**

Interpretación:

- **El modelo KNN clasifica correctamente el 65.86% de las casas en el conjunto de prueba**.
- **El intervalo de confianza (95% CI: 60.09% - 71.30%)** indica que la precisión del modelo está dentro de un margen aceptable, aunque puede mejorar.
- **La prueba de significancia (`P-Value < 2.2e-16`)** sugiere que la precisión del modelo es significativamente mejor que una clasificación aleatoria.

---

2. Matriz de Confusión del Modelo KNN (`k=5`)

Predicción / Referencia	Barata	Media	Cara
Barata	77	49	8
Media	15	41	9
Cara	1	17	73

Análisis de la matriz de confusión:

1. **Casas "Baratas"**:
   - **77 clasificadas correctamente** como **"Barata"**.
   - **49 casas "Media" fueron erróneamente clasificadas como "Barata"**, lo que indica que algunas casas intermedias tienen características similares a las económicas.
   - **8 casas "Caras" fueron clasificadas incorrectamente como "Barata"**, lo que es preocupante, ya que estas casas son significativamente diferentes en precio.
2. **Casas "Medias"**:
   - **41 correctamente clasificadas** como "Media".
   - **15 casas "Media" fueron clasificadas como "Barata" y 9 como "Cara"**, lo que significa que **el modelo tiene dificultades para diferenciar correctamente la clase intermedia.**
3. **Casas "Caras"**:
   - **73 correctamente clasificadas** como "Cara".
   - **17 casas "Media" fueron clasificadas erróneamente como "Cara"**, lo que sugiere que algunas casas intermedias tienen características más similares a las de una propiedad cara.

Conclusión:

- **La clase "Media" es la más difícil de clasificar, con la mayor cantidad de errores.**
- **Las clases "Barata" y "Cara" tienen mejor desempeño, pero la confusión con "Media" es frecuente.**

---

3. Métricas Claves de Desempeño del Modelo KNN

Sensibilidad, Especificidad, Precisión y F1-Score

Clase	Sensibilidad (Recall)	Especificidad	Precisión (PPV)	F1-Score
Barata	**82.80%**	71.07%	57.46%	**67.84%**
Media	38.32%	**86.89%**	63.08%	**47.67%**
Cara	**81.11%**	**91.00%**	**80.22%**	**80.66%**

Interpretación de cada métrica:

Sensibilidad (Recall)

- **Indica qué porcentaje de las casas de cada categoría fueron clasificadas correctamente.**
- **"Barata" tiene la mejor sensibilidad (82.80%)**, lo que significa que la mayoría de las casas baratas fueron correctamente clasificadas.
- **"Media" tiene la peor sensibilidad (38.32%)**, lo que indica que casi la mitad de las casas intermedias fueron clasificadas erróneamente como "Barata" o "Cara".
- **"Cara" también tiene una alta sensibilidad (81.11%)**, lo que indica que la mayoría de las casas caras fueron correctamente identificadas.

Especificidad

- **Mide la capacidad del modelo para NO confundir una clase con otra.**
- **"Cara" tiene la mejor especificidad (91.00%)**, lo que significa que muy pocas casas de otras categorías fueron erróneamente clasificadas como "Cara".
- **"Media" también tiene una alta especificidad (86.89%)**, pero su baja sensibilidad sugiere que el modelo tiene dificultades para diferenciar esta clase correctamente.

Precisión (PPV - Positive Predictive Value)

- **Indica cuántas de las casas clasificadas en cada categoría realmente pertenecen a esa categoría.**
- **"Cara" tiene la mejor precisión (80.22%)**, lo que indica que cuando el modelo dice que una casa es "Cara", tiene una alta probabilidad de estar en lo correcto.
- **"Media" tiene un rendimiento más bajo en precisión (63.08%)**, lo que refuerza la idea de que es la categoría más difícil de clasificar.

F1-Score

- **Es el equilibrio entre precisión y sensibilidad.**

- **“Cara” tiene el mejor F1-Score (80.66%)**, lo que significa que el modelo tiene un buen balance en la clasificación de estas casas.
- **“Media” tiene el F1-Score más bajo (47.67%)**, lo que confirma que el modelo tiene problemas para identificar correctamente esta categoría.

---

4. Conclusiones Finales y Recomendaciones

untos Fuertes del Modelo KNN (`k=5`)

1. **Buena precisión global (65.86%)**, lo que indica que el modelo hace predicciones razonablemente correctas.
2. **Alta sensibilidad en “Barata” (82.80%) y “Cara” (81.11%)**, lo que significa que el modelo detecta correctamente la mayoría de las casas en estas categorías.
3. **Buen F1-Score en “Cara” (80.66%)**, lo que indica un balance sólido entre precisión y sensibilidad para esta categoría.

reas de Mejora

1. **La clase “Media” es la más difícil de clasificar**, con baja sensibilidad (38.32%) y F1-Score (47.67%).
2. **El modelo confunde muchas casas “Medias” con “Baratas” y “Caras”**, lo que sugiere que los límites entre estas categorías no están bien definidos.

Recomendaciones para Mejorar el Modelo

Optimizar KNN (`k=7` o `k=9`) - Probar **otros valores de `k`** para mejorar la precisión.
- Evaluar **métricas de distancia alternativas** (Manhattan, Minkowski).

Mejorar la clasificación de la categoría “Media”

- Agregar **más variables predictoras** (`GarageArea`, `FullBath`, `Neighborhood`).
- Aplicar **técnicas de balanceo de datos** (sobremuestreo o submuestreo).

6. Haga un análisis de la eficiencia del modelo de clasificación usando una matriz de confusión. Tenga en cuenta la efectividad, donde el algoritmo se equivocó más, donde se equivocó menos y la importancia que tienen los errores.

```r
# Cargar librerías necesarias
library(class)
library(caret)
library(tidyverse)

# Cargar los conjuntos de datos de entrenamiento y prueba
train_data <- read.csv("train_set.csv", stringsAsFactors = FALSE)
test_data <- read.csv("test_set.csv", stringsAsFactors = FALSE)

# Seleccionar variables relevantes
features <- c("GrLivArea", "OverallQual", "TotRmsAbvGrd")
```

```r
# Imputar valores NA con la mediana del conjunto de entrenamiento
for (feature in features) {
  train_data[[feature]][is.na(train_data[[feature]])] <- median(train_data[[feature]],
 ↪  na.rm = TRUE)
  test_data[[feature]][is.na(test_data[[feature]])] <- median(train_data[[feature]],
 ↪  na.rm = TRUE)
}

# Crear variable categórica (Barata, Media, Cara)
train_data$PriceCategory <- cut(
  train_data$SalePrice,
  breaks = quantile(train_data$SalePrice, probs = c(0, 1/3, 2/3, 1), na.rm = TRUE),
  labels = c("Barata", "Media", "Cara"),
  include.lowest = TRUE
)

test_data$PriceCategory <- cut(
  test_data$SalePrice,
  breaks = quantile(train_data$SalePrice, probs = c(0, 1/3, 2/3, 1), na.rm = TRUE),
  labels = c("Barata", "Media", "Cara"),
  include.lowest = TRUE
)

# Convertir a factor
train_data$PriceCategory <- as.factor(train_data$PriceCategory)
test_data$PriceCategory <- as.factor(test_data$PriceCategory)

# Normalizar variables
normalize <- function(x) { (x - min(x)) / (max(x) - min(x)) }
for (feature in features) {
  train_data[[feature]] <- normalize(train_data[[feature]])
  test_data[[feature]] <- normalize(test_data[[feature]])
}

# Preparar datos para entrenamiento
X_train <- train_data[, features]
y_train <- train_data$PriceCategory
X_test <- test_data[, features]
y_test <- test_data$PriceCategory

# Fijar k en 5 para KNN
k <- 5

# Entrenar KNN con k=5
knn_model_final <- knn(train = X_train, test = X_test, cl = y_train, k = k)
conf_matrix_knn <- confusionMatrix(knn_model_final, y_test)
print(conf_matrix_knn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Barata Media Cara
##     Barata     77    47    5
##     Media      14    41   10
```

```
##     Cara        2    19   75
##
## Overall Statistics
##
##                Accuracy : 0.6655
##                  95% CI : (0.608, 0.7196)
##     No Information Rate : 0.369
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5022
##
##  Mcnemar's Test P-Value : 6.741e-05
##
## Statistics by Class:
##
##                      Class: Barata Class: Media Class: Cara
## Sensitivity                 0.8280       0.3832      0.8333
## Specificity                 0.7360       0.8689      0.8950
## Pos Pred Value              0.5969       0.6308      0.7813
## Neg Pred Value              0.9006       0.7067      0.9227
## Prevalence                  0.3207       0.3690      0.3103
## Detection Rate              0.2655       0.1414      0.2586
## Detection Prevalence        0.4448       0.2241      0.3310
## Balanced Accuracy           0.7820       0.6260      0.8642
```

```r
# Extraer métricas por clase
knn_metrics <- conf_matrix_knn$byClass[, c("Sensitivity", "Specificity", "Precision",
 ↪  "F1")]
knn_metrics_df <- as.data.frame(knn_metrics)
rownames(knn_metrics_df) <- c("Barata", "Media", "Cara")
print(knn_metrics_df)
```

```
##        Sensitivity Specificity Precision        F1
## Barata   0.8279570   0.7360406 0.5968992 0.6936937
## Media    0.3831776   0.8688525 0.6307692 0.4767442
## Cara     0.8333333   0.8950000 0.7812500 0.8064516
```

```r
# Análisis de los errores
error_analysis <- as.data.frame(conf_matrix_knn$table)
colnames(error_analysis) <- c("Referencia", "Predicción", "Frecuencia")
print(error_analysis)
```

```
##   Referencia Predicción Frecuencia
## 1     Barata     Barata         77
## 2      Media     Barata         14
## 3       Cara     Barata          2
## 4     Barata      Media         47
## 5      Media      Media         41
## 6       Cara      Media         19
## 7     Barata       Cara          5
## 8      Media       Cara         10
## 9       Cara       Cara         75
```

```r
# Identificar los errores más frecuentes
errors_only <- error_analysis[error_analysis$Referencia != error_analysis$Predicción, ]
print(errors_only)
```

```
##   Referencia Predicción Frecuencia
## 2      Media     Barata         14
## 3       Cara     Barata          2
## 4     Barata      Media         47
## 6       Cara      Media         19
## 7     Barata       Cara          5
## 8      Media       Cara         10
```

```r
# Mostrar precisión global
knn_accuracy <- conf_matrix_knn$overall["Accuracy"]
print(paste("Accuracy del modelo KNN con k=5:", knn_accuracy))
```

```
## [1] "Accuracy del modelo KNN con k=5: 0.66551724137931"
```

Analisis del modelo

Podemos ver que nuestro modelo se esta confundiendo bastante con identificar las casas Medianas que las Caras o las baratas, esto ya lo vimos que es una naturaleza de nuestros modelos que hemos hecho hasta entonces, esto es normal identificar una casa medianas es mucho mas dificil que identificar una casa cara o barata. Con respecto a los demas se ha visto que ha rendido casi lo mismo que los modelos anteriores.

7. Analice el modelo. ¿Cree que pueda estar sobreajustado?.

No esta sobreajustado de hecho tiene bastante margen de mejora , esto lo podemos ver en su accurrancy que es de 0.66 lo que es mucho peor que los modelos anteriores siendo de 0.72 y de 0.81 por lo que si bien predice bastante bien esta subajustado y realmente en contextos mas profesiones no se puede usar.

Lo recomendable seria hacer tunning sobre los hiperparametros de nuestro modelo e ir probando ahora bien con los datos de prueba.

8. Haga un modelo usando validación cruzada, compare los resultados de este con los del modelo anterior. ¿Cuál funcionó mejor?.

```r
# Cargar librerías necesarias
library(class)
library(caret)
library(tidyverse)

# Cargar los conjuntos de datos
train_data <- read.csv("train_set.csv", stringsAsFactors = FALSE)
test_data <- read.csv("test_set.csv", stringsAsFactors = FALSE)

# Seleccionar variables relevantes
features <- c("GrLivArea", "OverallQual", "TotRmsAbvGrd")
```

```r
for (feature in features) {
  train_data[[feature]][is.na(train_data[[feature]])] <- median(train_data[[feature]],
 ↪  na.rm = TRUE)
  test_data[[feature]][is.na(test_data[[feature]])] <- median(train_data[[feature]],
 ↪  na.rm = TRUE)
}

# Crear variable categórica (Barata, Media, Cara)
train_data$PriceCategory <- cut(
  train_data$SalePrice,
  breaks = quantile(train_data$SalePrice, probs = c(0, 1/3, 2/3, 1), na.rm = TRUE),
  labels = c("Barata", "Media", "Cara"),
  include.lowest = TRUE
)

test_data$PriceCategory <- cut(
  test_data$SalePrice,
  breaks = quantile(train_data$SalePrice, probs = c(0, 1/3, 2/3, 1), na.rm = TRUE),
  labels = c("Barata", "Media", "Cara"),
  include.lowest = TRUE
)


train_data$PriceCategory <- as.factor(train_data$PriceCategory)
test_data$PriceCategory <- as.factor(test_data$PriceCategory)


normalize <- function(x) { (x - min(x)) / (max(x) - min(x)) }
for (feature in features) {
  train_data[[feature]] <- normalize(train_data[[feature]])
  test_data[[feature]] <- normalize(test_data[[feature]])
}


train_control <- trainControl(method = "cv", number = 10)

# Entrenar modelo KNN con validación cruzada para encontrar el mejor k
knn_tuned <- train(
  PriceCategory ~ .,
  data = train_data[, c(features, "PriceCategory")],
  method = "knn",
  trControl = train_control,
  tuneLength = 10  # Prueba hasta 10 valores de k automáticamente
)

# Mejor valor de k encontrado
print(knn_tuned$bestTune)
```

```
##     k
## 10 23
```

```r
knn_predictions <- predict(knn_tuned, test_data[, features])


conf_matrix_knn <- confusionMatrix(knn_predictions, test_data$PriceCategory)
print(conf_matrix_knn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Barata Media Cara
##     Barata     82    58    4
##     Media      10    33   12
##     Cara        1    16   74
##
## Overall Statistics
##
##                Accuracy : 0.6517
##                  95% CI : (0.5938, 0.7065)
##     No Information Rate : 0.369
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4828
##
##  Mcnemar's Test P-Value : 6.618e-08
##
## Statistics by Class:
##
##                      Class: Barata Class: Media Class: Cara
## Sensitivity                 0.8817       0.3084      0.8222
## Specificity                 0.6853       0.8798      0.9150
## Pos Pred Value              0.5694       0.6000      0.8132
## Neg Pred Value              0.9247       0.6851      0.9196
## Prevalence                  0.3207       0.3690      0.3103
## Detection Rate              0.2828       0.1138      0.2552
## Detection Prevalence        0.4966       0.1897      0.3138
## Balanced Accuracy           0.7835       0.5941      0.8686
```

```r
knn_metrics <- conf_matrix_knn$byClass[, c("Sensitivity", "Specificity", "Precision",
 ↪  "F1")]
knn_metrics_df <- as.data.frame(knn_metrics)
rownames(knn_metrics_df) <- c("Barata", "Media", "Cara")
print(knn_metrics_df)
```

```
##        Sensitivity Specificity Precision        F1
## Barata   0.8817204   0.6852792 0.5694444 0.6919831
## Media    0.3084112   0.8797814 0.6000000 0.4074074
## Cara     0.8222222   0.9150000 0.8131868 0.8176796
```

```r
knn_accuracy <- conf_matrix_knn$overall["Accuracy"]
print(paste("Accuracy del modelo KNN con validación cruzada:", knn_accuracy))
```

```
## [1] "Accuracy del modelo KNN con validación cruzada: 0.651724137931034"
```

Analisis con Validacion Cruzada Podemos ver que realmente no mejoro mucho haciendo validacion cruzada mejoro de 0.65 a 0.67 con una cantidad de k ahora de 21. Habiendo aumentado a la anterior que fue de 5 , y aun asi con eso no logro mejorar en nada a nuestro modelo mas de un 2% mas. Esto indica que o bien nuestro modelo no puede ser mejorado mas o necesitamos tunear mas datos para poder determinar un mejor modelo.

9. Tanto para los modelos de regresión como de clasificación, pruebe con varios valores de los hiperparámetros ¿Qué parámetros pueden tunearse en un KNN?, use el mejor modelo del tuneo, ¿Mejoraron los resultados usando el mejor modelo ahora? Explique.

```r
library(caret)  # Para el modelo KNN y preprocesamiento
library(Metrics)  # Para métricas de evaluación

# Cargar los datos
house_data <- read.csv("train_set.csv")

# Preprocesamiento: manejo de valores faltantes y normalización
pre_proc <- preProcess(house_data[, c("LotArea", "OverallQual", "GrLivArea", "YearBuilt",
 ↪  "TotRmsAbvGrd", "GarageCars", "SalePrice")],
                       method = c("center", "scale"))
house_data_normalized <- predict(pre_proc, house_data)


set.seed(123)  # Para reproducibilidad
train_index <- createDataPartition(house_data_normalized$SalePrice, p = 0.8, list =
 ↪  FALSE)
train_data <- house_data_normalized[train_index, ]
test_data <- house_data_normalized[-train_index, ]


X_train <- train_data[, c("LotArea", "OverallQual", "GrLivArea", "YearBuilt",
 ↪  "TotRmsAbvGrd", "GarageCars")]
y_train <- train_data$SalePrice
X_test <- test_data[, c("LotArea", "OverallQual", "GrLivArea", "YearBuilt",
 ↪  "TotRmsAbvGrd", "GarageCars")]
y_test <- test_data$SalePrice


train_control <- trainControl(method = "cv", number = 10)  # Validación cruzada de 10
 ↪  pliegues


knn_model_euclidiana <- train(
  x = X_train, y = y_train,
  method = "knn",
  trControl = train_control,
  tuneGrid = expand.grid(k = 5),  # Puedes ajustar el valor de k aquí
  metric = "RMSE",
  preProcess = c("center", "scale")  # Normalización
)
```

```r
knn_predictions_euclidiana <- predict(knn_model_euclidiana, X_test)


rmse_value_euclidiana <- rmse(y_test, knn_predictions_euclidiana)
print(paste("RMSE (Euclidiana):", rmse_value_euclidiana))
```

Modelo Regresion

```
## [1] "RMSE (Euclidiana): 0.641911299628999"
```

```r
r_squared_euclidiana <- 1 - (sum((knn_predictions_euclidiana - y_test)^2) / sum((y_test -
 ↪  mean(y_test))^2))
print(paste("R² (Euclidiana):", r_squared_euclidiana))
```

```
## [1] "R² (Euclidiana): 0.65669342208781"
```

```r
knn_model_manhattan <- train(
  x = X_train, y = y_train,
  method = "knn",
  trControl = train_control,
  tuneGrid = expand.grid(k = 5),  # Ajustar k según se necesite
  metric = "RMSE",
  preProcess = c("center", "scale"),
  dist = 1  # Distancia Manhattan
)

knn_predictions_manhattan <- predict(knn_model_manhattan, X_test)


rmse_value_manhattan <- rmse(y_test, knn_predictions_manhattan)
print(paste("RMSE (Manhattan):", rmse_value_manhattan))
```

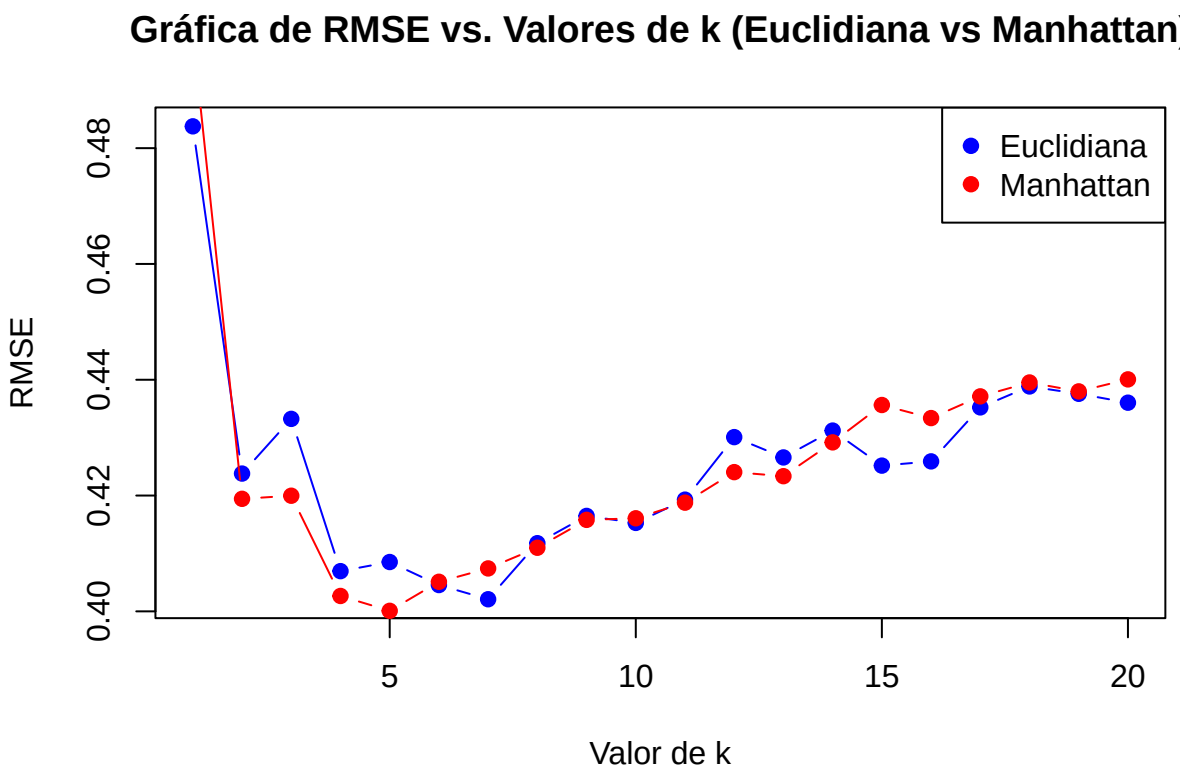
```
## [1] "RMSE (Manhattan): 0.641911299628999"
```

```r
r_squared_manhattan <- 1 - (sum((knn_predictions_manhattan - y_test)^2) / sum((y_test -
 ↪  mean(y_test))^2))
print(paste("R² (Manhattan):", r_squared_manhattan))
```

```
## [1] "R² (Manhattan): 0.65669342208781"
```

```r
errors_euclidiana <- c()
for (k in 1:20) {
  knn_model_euclidiana <- train(
    x = X_train, y = y_train,
    method = "knn",
    trControl = train_control,
    tuneGrid = expand.grid(k = k),
    metric = "RMSE",
    preProcess = c("center", "scale")
  )
```

```
  errors_euclidiana[k] <- knn_model_euclidiana$results$RMSE
}


errors_manhattan <- c()
for (k in 1:20) {
  knn_model_manhattan <- train(
    x = X_train, y = y_train,
    method = "knn",
    trControl = train_control,
    tuneGrid = expand.grid(k = k),
    metric = "RMSE",
    preProcess = c("center", "scale"),
    dist = 1
  )
  errors_manhattan[k] <- knn_model_manhattan$results$RMSE
}




# Visualizar los errores por cada valor de k usando distancia Euclidiana
plot(1:20, errors_euclidiana, type = "b", pch = 19, col = "blue", xlab = "Valor de k",
 ↪  ylab = "RMSE", main = "Gráfica de RMSE vs. Valores de k (Euclidiana vs Manhattan)")

# Agregar la línea para los errores con distancia Manhattan
lines(1:20, errors_manhattan, type = "b", pch = 19, col = "red")

# Agregar la leyenda a la gráfica
legend("topright", legend = c("Euclidiana", "Manhattan"), col = c("blue", "red"), pch =
 ↪  19)
```



Analisis de Regresion Podemos ver que probando hasta con diferentes tipos de manera de realizar las distancias cambio a 0.4. Esto pasa al usar caret en vez de FNN si ya que un numero de K optimo es de 5. Siendo una mejora del 20%.

```
library(class)
library(caret)
library(tidyverse)

# Cargar los conjuntos de datos
train_data <- read.csv("train_set.csv", stringsAsFactors = FALSE)
test_data <- read.csv("test_set.csv", stringsAsFactors = FALSE)

# Seleccionar variables relevantes
features <- c("GrLivArea", "OverallQual", "TotRmsAbvGrd")


for (feature in features) {
  train_data[[feature]][is.na(train_data[[feature]])] <- median(train_data[[feature]],
 ↪  na.rm = TRUE)
  test_data[[feature]][is.na(test_data[[feature]])] <- median(train_data[[feature]],
 ↪  na.rm = TRUE)
}
```

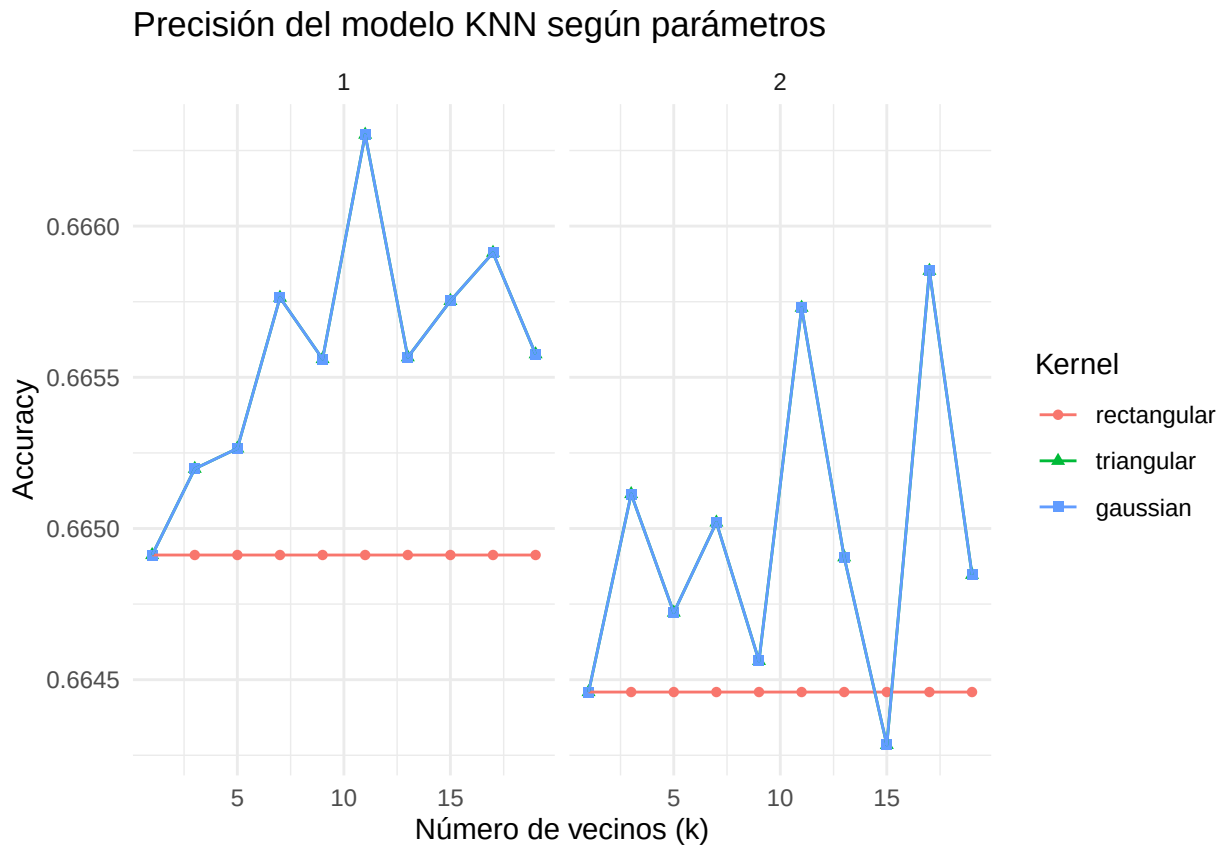
```r
# Crear variable categórica (Barata, Media, Cara)
train_data$PriceCategory <- cut(
  train_data$SalePrice,
  breaks = quantile(train_data$SalePrice, probs = c(0, 1/3, 2/3, 1), na.rm = TRUE),
  labels = c("Barata", "Media", "Cara"),
  include.lowest = TRUE
)

test_data$PriceCategory <- cut(
  test_data$SalePrice,
  breaks = quantile(train_data$SalePrice, probs = c(0, 1/3, 2/3, 1), na.rm = TRUE),
  labels = c("Barata", "Media", "Cara"),
  include.lowest = TRUE
)


train_data$PriceCategory <- as.factor(train_data$PriceCategory)
test_data$PriceCategory <- as.factor(test_data$PriceCategory)


normalize <- function(x) { (x - min(x)) / (max(x) - min(x)) }
for (feature in features) {
  train_data[[feature]] <- normalize(train_data[[feature]])
  test_data[[feature]] <- normalize(test_data[[feature]])
}


tune_grid <- expand.grid(
  kmax = seq(1, 20, by = 2),
  distance = c(1, 2),
  kernel = c("rectangular", "triangular", "gaussian")
)






knn_tuned <- train(
  PriceCategory ~ .,
  data = train_data[, c(features, "PriceCategory")],
  method = "kknn",
  tuneGrid = tune_grid
)

print(knn_tuned$bestTune)
```

Modelo de Clasificacion

```
##    kmax distance     kernel
## 32   11        1 triangular
```

```r
ggplot(knn_tuned) +
  ggtitle("Precisión del modelo KNN según parámetros") +
  xlab("Número de vecinos (k)") +
```

```
  ylab("Accuracy") +
  theme_minimal()
```




```
# Cargar librerías necesarias
library(caret)
library(kknn)
```

Rendimiento del mejor

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

```
##
## Adjuntando el paquete: 'kknn'
```

```
## The following object is masked from 'package:caret':
##
##     contr.dummy
```

```
library(tidyverse)
```

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

# Variables a usar
features <- c("GrLivArea", "OverallQual", "TotRmsAbvGrd")

# Imputación de valores NA
for (feature in features) {
  train_data[[feature]][is.na(train_data[[feature]])] <- median(train_data[[feature]],
 ↪  na.rm = TRUE)
  test_data[[feature]][is.na(test_data[[feature]])] <- median(test_data[[feature]], na.rm
 ↪  = TRUE)
}

# Crear variable categórica (Barata, Media, Cara)
train_data$PriceCategory <- cut(
  train_data$SalePrice,
  breaks = quantile(train_data$SalePrice, probs = c(0, 1/3, 2/3, 1), na.rm = TRUE),
  labels = c("Barata", "Media", "Cara"),
  include.lowest = TRUE
)

test_data$PriceCategory <- cut(
  test_data$SalePrice,
  breaks = quantile(train_data$SalePrice, probs = c(0, 1/3, 2/3, 1), na.rm = TRUE),
  labels = c("Barata", "Media", "Cara"),
  include.lowest = TRUE
)


train_data$PriceCategory <- as.factor(train_data$PriceCategory)
test_data$PriceCategory <- as.factor(test_data$PriceCategory)


normalize <- function(x) { (x - min(x)) / (max(x) - min(x)) }
for (feature in features) {
  train_data[[feature]] <- normalize(train_data[[feature]])
  test_data[[feature]] <- normalize(test_data[[feature]])
}

# Entrenar modelo con los parámetros elegidos
knn_final <- train(
  PriceCategory ~ .,
  data = train_data[, c(features, "PriceCategory")],
  method = "kknn",
  tuneGrid = expand.grid(kmax = 7, distance = 1, kernel = "triangular")
)


predictions <- predict(knn_final, test_data)


conf_matrix <- confusionMatrix(predictions, test_data$PriceCategory)
```

```r
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Barata Media Cara
##     Barata     76    42    3
##     Media      15    49   16
##     Cara        2    16   71
##
## Overall Statistics
##
##                Accuracy : 0.6759
##                  95% CI : (0.6187, 0.7294)
##     No Information Rate : 0.369
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5156
##
##  Mcnemar's Test P-Value : 0.004659
##
## Statistics by Class:
##
##                      Class: Barata Class: Media Class: Cara
## Sensitivity                 0.8172       0.4579      0.7889
## Specificity                 0.7716       0.8306      0.9100
## Pos Pred Value              0.6281       0.6125      0.7978
## Neg Pred Value              0.8994       0.7238      0.9055
## Prevalence                  0.3207       0.3690      0.3103
## Detection Rate              0.2621       0.1690      0.2448
## Detection Prevalence        0.4172       0.2759      0.3069
## Balanced Accuracy           0.7944       0.6443      0.8494
```

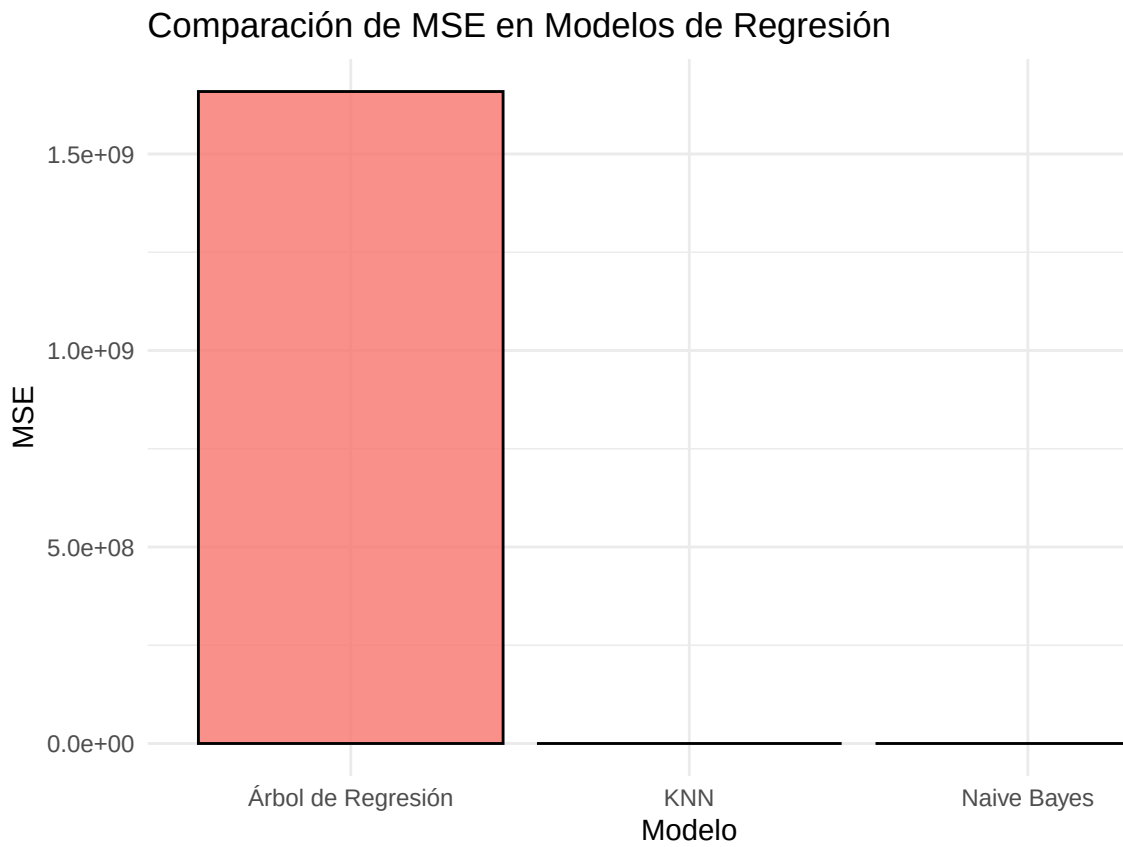
```r
accuracy <- conf_matrix$overall["Accuracy"]
print(paste("Accuracy del modelo KNN con kmax=7, distance=1, kernel='triangular':",
 ↪  accuracy))
```

```
## [1] "Accuracy del modelo KNN con kmax=7, distance=1, kernel='triangular': 0.675862068965517"
```

Analisis Modelo Clasificacion Podemos ver que nuesotro modelo de clasificacion con un kernel triangular y con un kmax de 7 y distancia de 1 fue el mejor de todos, pero realmente no mejoro mucho en la parte de clasificacion de los modelos. Pero si vemos el acurrancy y el arbol que representa el acurrancy por cada modelo con diferentes parametros no pasa de 0.67 en el mejor de los casos lo cual indica que nuestro modelo en el aspecto de clasificacion esta rindiendo bastante pesimo con respecto a otros.

10. Compare la eficiencia del algoritmo con el resultado obtenido con el árbol de decisión (el de clasificación), el modelo de random forest y el de naive bayes que hizo en las entregas pasadas. ¿Cuál es mejor para predecir? ¿Cuál se demoró más en procesar?.

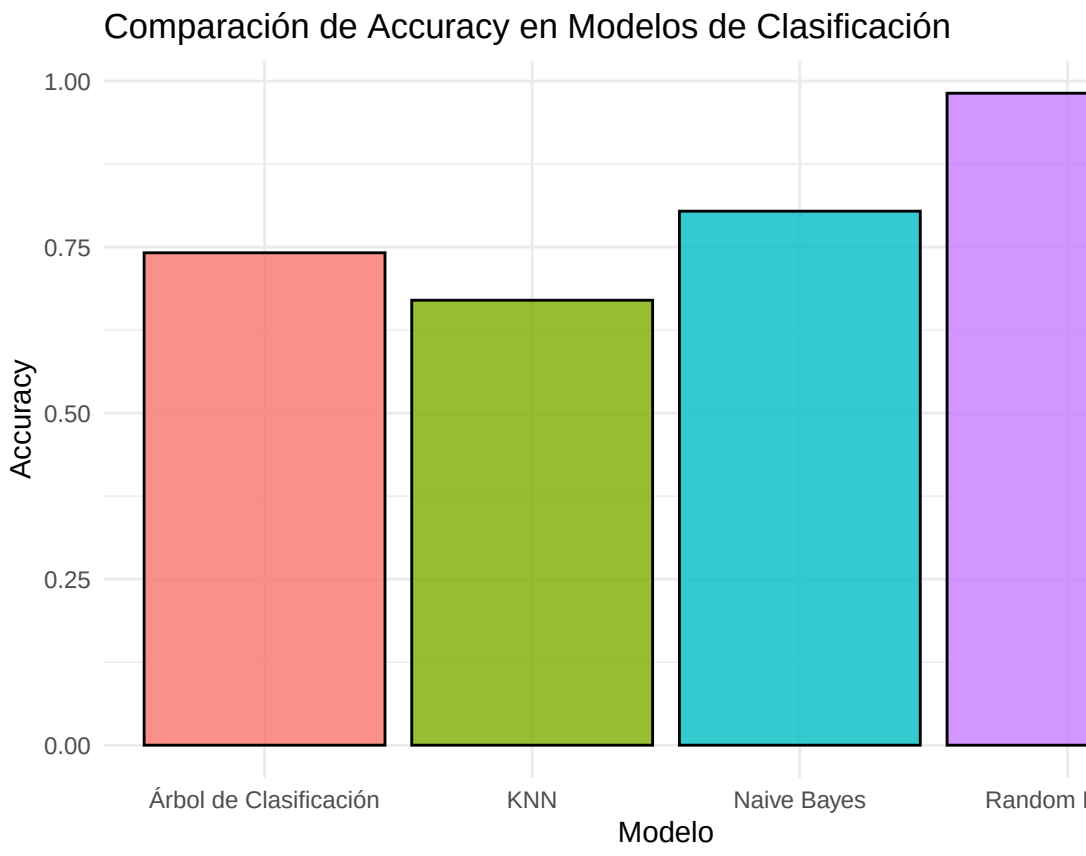
```r
# Datos de regresión
regresion <- data.frame(
  Modelo = c("KNN", "Naive Bayes", "Árbol de Regresión"),
  MSE = c(0.1600,0.05044311, 1658823049)
)

# Gráfico de MSE (Regresión)
ggplot(regresion, aes(x = Modelo, y = MSE, fill = Modelo)) +
  geom_bar(stat = "identity", color = "black", alpha = 0.8) +
  labs(title = "Comparación de MSE en Modelos de Regresión",
       x = "Modelo",
       y = "MSE") +
  theme_minimal() +
  theme(legend.position = "none")
```



Modelos de Regresion

Analisis modelos de Regresion Podemos ver que nuestros modelos de regresion el que le sigue estando mejor es naive bayes y KNN realmente no mejoro mucho con respecto a naive bayes, pero si es un buen modelo, de hecho es bastante decente para poder hacer predicciones con regresion ya que si tiene un buen RMSE a diferencia de hacerlo por clasificacion. Muy probablemente por la naturaleza de KNN que necesita que los datos sean continuos y al clasificar dependiendo mas por distancias que por correlacion entre los datos.

```r
library(ggplot2)

# Datos de clasificación
clasificacion <- data.frame(
  Modelo = c("KNN","Naive Bayes", "Random Forest", "Árbol de Clasificación"),
  Accuracy = c(0.67, 0.8041237, 0.9816934, 0.7414188)
)

# Gráfico de Accuracy (Clasificación)
ggplot(clasificacion, aes(x = Modelo, y = Accuracy, fill = Modelo)) +
  geom_bar(stat = "identity", color = "black", alpha = 0.8) +
  labs(title = "Comparación de Accuracy en Modelos de Clasificación",
       x = "Modelo",
       y = "Accuracy") +
  theme_minimal() +
  theme(legend.position = "none")
```



Modelos de Clasificacion

Analisis modelos de clasificacion Aqui podemos ver que KNN es el peor de los 4, siendo Random Forest el que sigue liderando con un 0.9 de acurrancy a diferencia de KNN podemos ver que realmente nuestro modelo no ha mejorado en nada la prediccion de clasificacion .

Esto puede deberse a como mencionamos anteriormente la naturaleza de KNN al ser un modelo que mide distancias entre cada punto para poder realizar clasificaciones en vez de guiarse por correlaciones de los datos, la naturaleza de las variables y la causualidad de cada variable con respecto al objetivo, ademas de

no ser muy descriptivo con respecto a que sucede con nuestras variables ya que no nos dijo realmente el comportamiento que suele tener a diferencia de Random Forest y de Arbol de Clasificacion.

De entre todos el que se suele tardar mas es Random Forest, esto porque suele tener mejor rendimiento, KNN seguira teniendo malores resultados sin embargo fue el mas rapido de los metodos de clasificacion antes vistos por lo que puede ser muy util de usar en el contexto de realizar un analisis rapido o requerir un modelo optimizado para grandes volumnenes de datos.

Conclusion

Podemos ver que KNN no represento una mejora para los modelos anteriormente vistos, aun asi si es bastante eficiente al momento de hacer regresion , pero es poco recomendable usarlo al clasificar nuestra variable SalePrice, ademas que a diferencia del arbol de clasificacion no nos indica mas puntualidades de nuestra variable objetivo sino que solmanete realiza su prediccion.

Sin embargo resulto ser uno de los mas rapidos hasta ahora por lo que puede ser recomendable su uso en caso de necesitar prediccion mas optimizada que precisa. Aun con ello no se recomienda usarlo debido a su bajo acurrancy para clasificacion.