Minicourse II
Introduction to Statistical Learning with tidymodels

Natalia da Silva

IESTA-FCEA-UdelaR
VII Latin American Conference on Statistical Computing

natalia.dasilva@fcea.edu.uy - natydasilva.com - @pacocuak

Abril 2023

Course description

Introduce students to the main concepts in statistical learning focusing on supervised techniques for classification and regression problems such as decision trees, random forest among others.

All the methods will be implemented using `tidymodels` R package. This mini-course will be interactive with some hands-on practice. To take most of the course, students should use R at basic or intermediate level and it is recommended some previous contact with `tidyverse` R package.

Topics for today

- Intro to statistical learning
- Assessing Model Accuracy
- Classification and regression decision trees.
- Focused on CART algorithm
- Introduction to `tidymodels`
- Example with `tidymodels`

GitHub repo with slides and data

Type of Statistical learning methods

General context

- We observe a quantitative response Y and a set of p different predictors $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_p)$
- We are interested in the relationship between Y and $\mathbf{X}$ such that:

$$Y = f(\mathbf{X}) + \varepsilon$$

Where f is some fixed but unknown function of $(\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_p)$ and represents the systematic information that $\mathbf{X}$ provides about Y. ε is a random error term. Which is independent of $\mathbf{X}$ and has mean zero.

Type of Statistical learning methods

We are interested to study the relationship between Y y $\mathbf{X}$

$$Y = f(\mathbf{X}) + \varepsilon$$

Type of problems:

- Supervised, the response variable y_i it is available For each observation of the predictor variable(s) x_i. Regression problems (y_i is quantitative) or classification (y_i is qualitative)

- Unsupervised, for every observation we observe a vector of measurements x_i but no associated response y_i

- Semi supervise, y_i available for some x_i

It is important to identify the type of problem we have to explore and select possible methods to apply.

Model evaluation

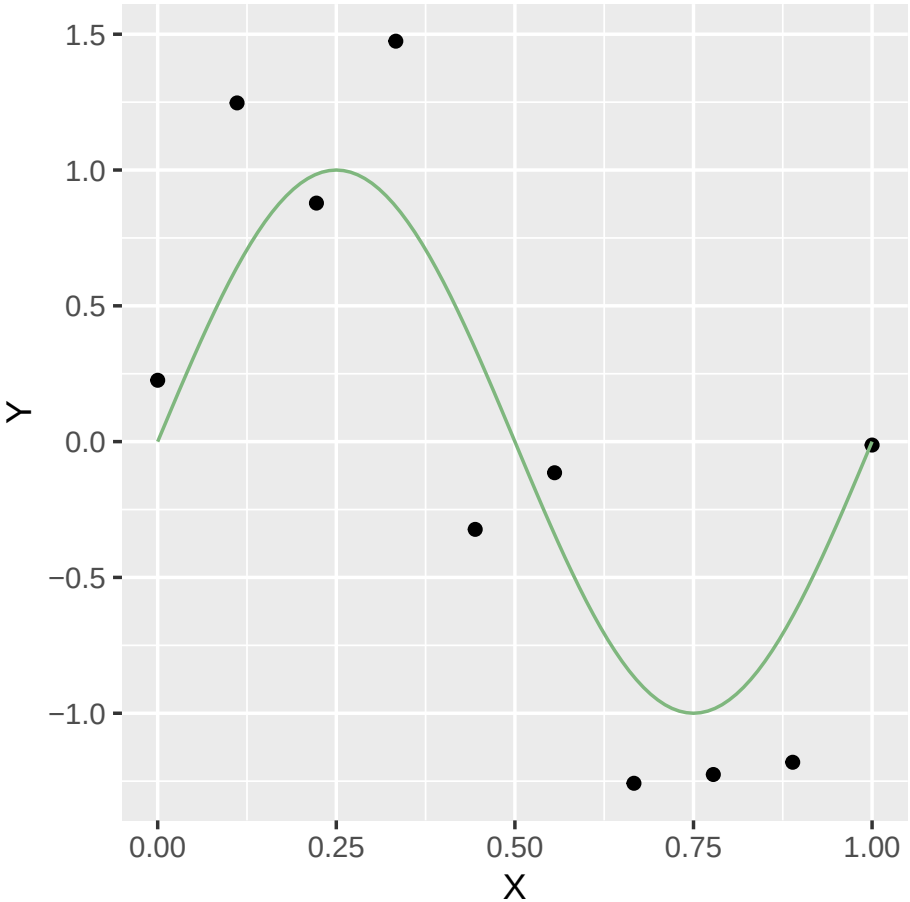
- There is not a statistical method that is the best for all the possible problems.
- Some methods works really well in some problems but not in others.
- We need to decide for each problem which method works better.
- Selecting the best approach can be one of the most challenging parts of performing statistical learning in practice.

Example: Polynomial linear model

Regression problem

- we observe a predictor variable $x \in R$.
- Objective is to predict $y \in R$.
- In general $y = f(x) + \varepsilon$.
- Supose we know the data generating process, where $f(x) = sin(2 * \pi * x)$ plus a random noise ($N(10,0,3)$), (exemple PRML-Bishop).
- Let x equidistant in the interval $[0,1]$.

Scatter plot of simulated data (training).
Green curve is $f(x) = sin(2 * x * \pi)$.




```
d <- tibble(x = seq(0,1,length.out = 10 ),
            f=sin(2*pi*x),
       y = f + rnorm(10, sd = 0.3))

p <- ggplot(d, aes(x = x, y = y)) +
  geom_point() +
  xlab("X") + ylab("Y")

f <- function(x){sin(2*pi*x)}
p1 <- p +  geom_function(fun = f,
                         color = '#7FB77E')

p1
```

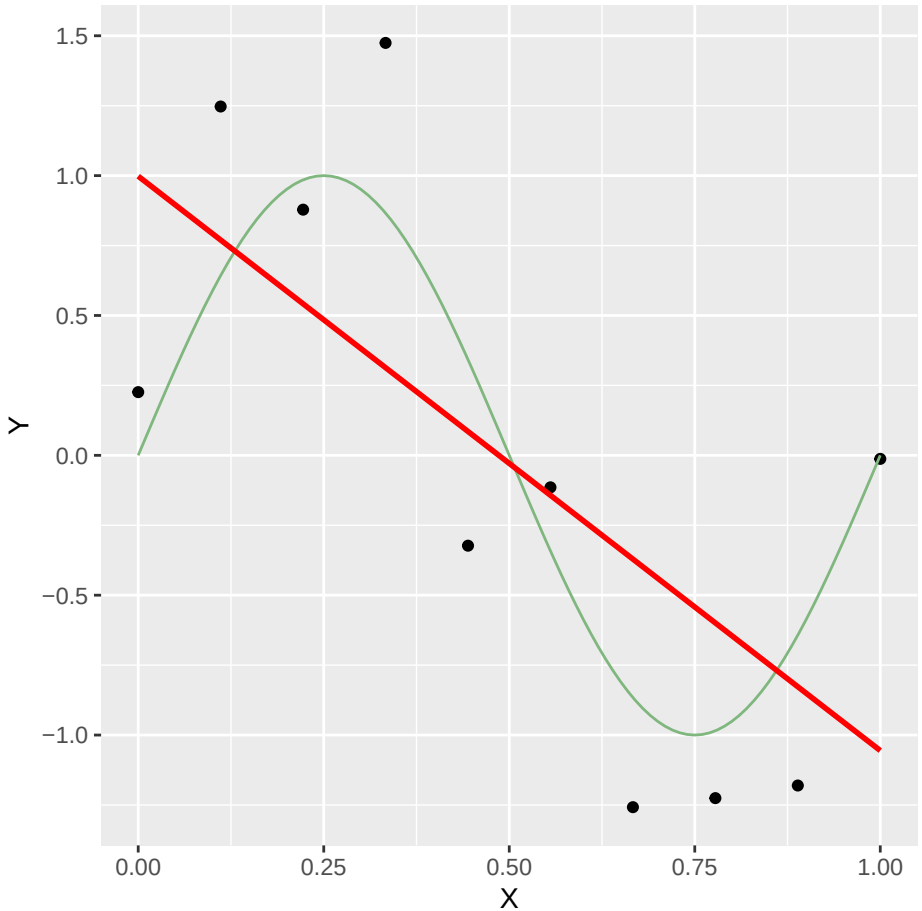
- Simulate the data in this way allows us to capture properties from real data with certain regularity we want to learn.
- Observations are twisted by a random noise we don't control.
- The objective is to explore the training set to predict y values for new x values without knowing the true f.

A simple curve aproximation can be done using polynomial function such that:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots \beta_k x^K = \sum_{k=0}^{K} \beta_k x^k$$

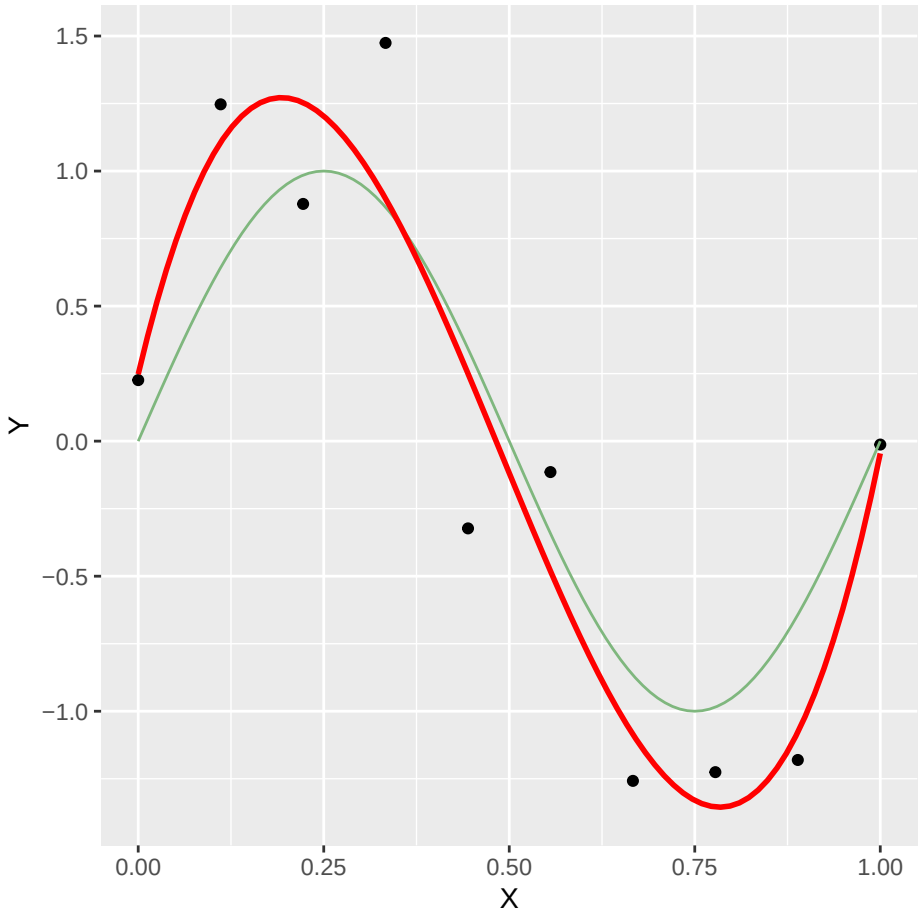
Where K is the polynomial degree, it is a linear model because is linear in its parameters.

Example: with $K = 1$



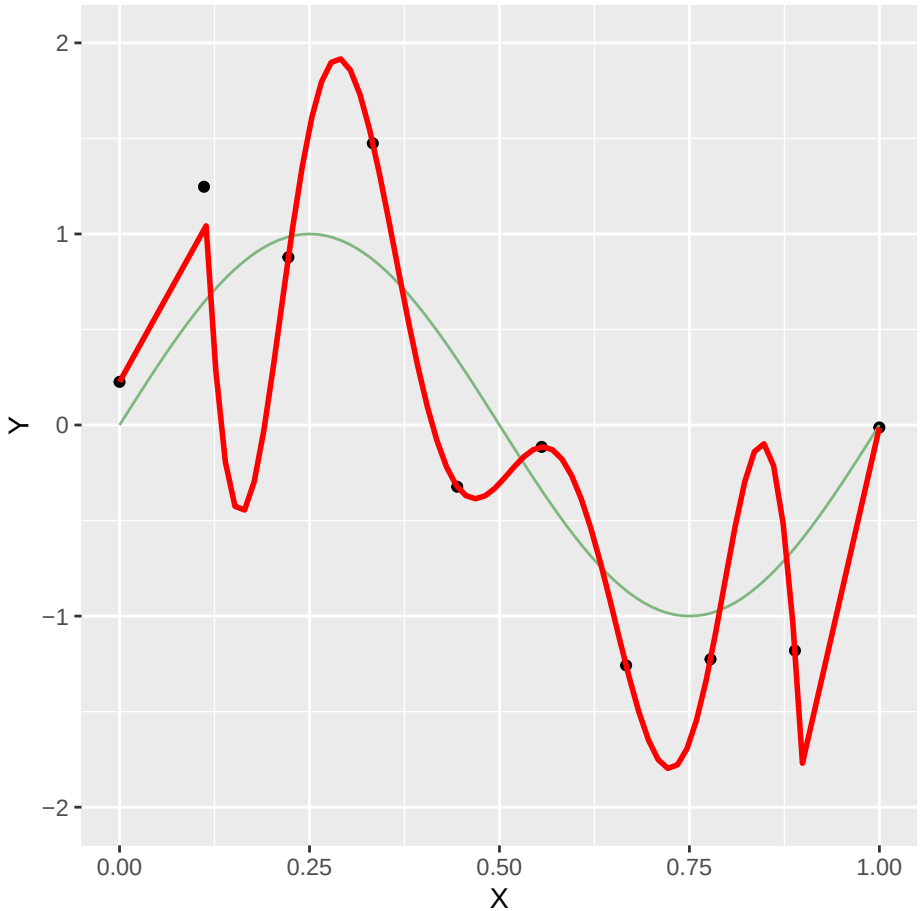

```
d <- tibble(x = seq(0,1,length.out = 20 ),
            f=sin(2*pi*x),
       y = f + rnorm(20, sd = 0.3))

p <- ggplot(d, aes(x = x, y = y)) +
  geom_point() +
  xlab("X") + ylab("Y")

f <- function(x){sin(2*pi*x)}
p1 <- p +
  geom_function(fun = f, color = '#7FB77E'

p1 + stat_smooth(method ='lm',
formula = y ~ poly(x, 1, raw=TRUE),
se = FALSE, colour="red")
```

Example: with $K = 3$




```
d <- tibble(x = seq(0,1,length.out = 10 ),
            f=sin(2*pi*x),
       y = f + rnorm(10, sd = 0.3))

p <- ggplot(d, aes(x = x, y = y)) +
  geom_point() +
  xlab("X") + ylab("Y")

f <- function(x){sin(2*pi*x)}
p1 <- p +  geom_function(fun = f, color =

 p1 + stat_smooth(method ='lm',
formula = y ~ poly(x, 3, raw=TRUE),
se = FALSE, colour="red")
```

Example: with $K = 9$




```
d <- tibble(x = seq(0,1,length.out = 10 ),
            f=sin(2*pi*x),
       y = f + rnorm(10, sd = 0.3))

p <- ggplot(d, aes(x = x, y = y)) +
  geom_point() +
  xlab("X") + ylab("Y")

f <- function(x){sin(2*pi*x)}
p1 <- p +
  geom_function(fun = f, color = '#7FB77E'

 p1 + stat_smooth(method ='lm',
formula = y ~ poly(x, 9, raw=TRUE),
se = FALSE, colour="red")
```

How to elect the model?

A polynomial with $K = 9$ overfits, presents a samall error but is not a good approximation of $f()$

- To evaluate a statistical learning model in a data set, we need a measure of how well the predictions match the observed data.
- Quantify how close is the response predicted value of an observaation respect to the true response.
- In regression problem the most common measure is the mean square error (MSE).

How to select the model?

- $Tr = \{(y_i, \mathbf{x_i})\}_{i=1}^{n}$: training data set use to fit the model and estimate f.
- Most common accuracy measure Training MSEs.

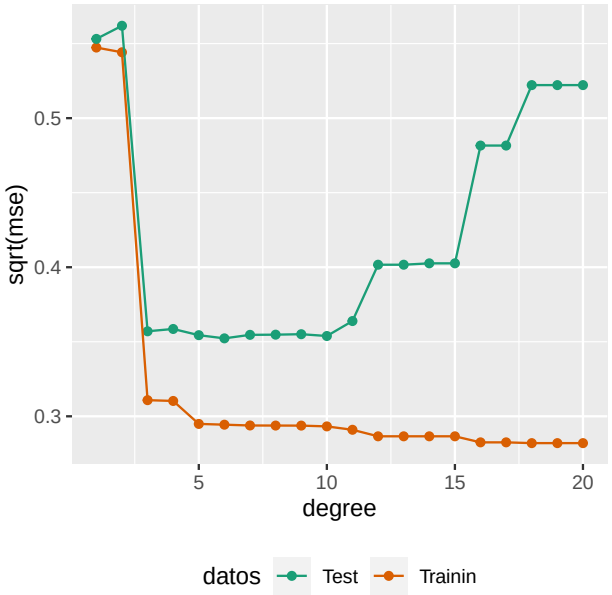
$$MSE_{Tr} = Ave_{i \in Tr}[y_i - \hat{f}(\mathbf{x_i})]^2 = \frac{1}{n}\sum_{i=1}^{n}[y_i - \hat{f}(\mathbf{x_i})]^2$$

How to select the model?

- $Te = \{(y_i, \mathbf{x_i})\}_{i=1}^{m}$: test data set, observations to compute de accuracy, where not used to estimate f.
- A better accuracy measure is used based on the test set. Testing MSEe testeo.

$$MSE_{Te} = Ave_{i \in Te}[y_i - \hat{f}(\mathbf{x_i})]^2 = \frac{1}{m}\sum_{i=1}^{m}[y_i - \hat{f}(\mathbf{x_i})]^2$$

$RMSE_{Tr}$, vs $RMSE_{Te}$




```
d <- tibble(x = seq(0,1,length.out = 200 ),
            f=sin(2*pi*x),
       y = f + rnorm(200, sd = 0.3))
train.id <-sample(1:200,100)
err <- matrix(NA, ncol = 2, nrow = 20 )

for(k in 1:20) {
  m <-  lm(y ~ poly(x, k, raw = TRUE),
           data = d[train.id,c('x','y')])

  pp <- predict(m, newdata = d[-train.id, c('x','y')] )

  err.ts <- mean((d$y[-train.id]-pp)^2)
  err.tr <-mean((d$y[train.id]-fitted(m))^2)
  err[k, ] <- cbind(err.tr, err.ts)
}

dt.err <- data.frame(grado=1:20, err) %>%
  set_names(nm = c('degree', 'Training', 'Test') ) %>%
  pivot_longer(cols = 2:3, values_to='mse', names_to='datos')

ggplot( dt.err, aes(x=degree, y= sqrt(mse), color=datos)) +
  geom_point() + geom_line() +
  scale_color_brewer(palette = 'Dark2')+
    theme(legend.position = 'bottom')
```

- We are interested in the accuracy of the predictions that we obtain when we apply our method to previously unseen test data.
- Generally, training error will be smaller than test error
- Because the training data is used to fit the model, by design the error will be small relative to the error when the model is used on new data.
- The generalized performance of a statistical learning method is related with its predictive prformance computed with the testing data set.

Bias-variance, trade-off

There are two competing forces that govern the choice of learning method: bias and variance.

- Bias is the error that is introduced by modeling a complicated problem by a simpler problem.
- Variance refers to how much your estimate would change if you had different training data. Its measuring how much your model depends on the data you have, to the neglect of future data.

Bias-variance, trade-off

$$MSE_{te} = Bias^2 + variance + ireducible$$

As the flexibility of $\hat{f}$ increases, its variance increases and its bias decreases. The decide on the best model, from a range with different flexibility, you choose based on average test MSE at the bias-variance trade-off, where both are minimized.

Decision trees

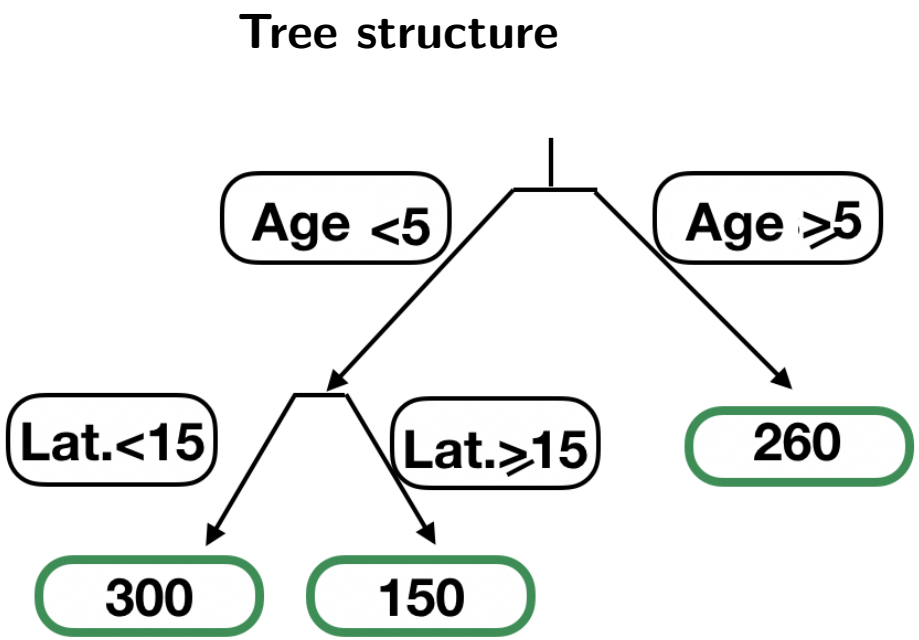
- Fifty Years of Classification and Regression Trees (Loh, 2014). Paper link.
- Tree algorithms can be used for classification or regession problems.
- First regression tree (Automatic Interaction Detection, AID) Morgan , 1963. There are a wide variety of tree methods and software to implemented them which increases its popularity.
- We will focused on CART algorithm (Breiman, 1984).

Decision trees

- Tree based methods consist in a nested partition squence which divide the predictor space, within these partitions, a model is used to predict the outcome.
- Then tree based models consist of one or more of nested if-then statements for the predictors that partition the data
- In theory regions defined by trees can be with different shapes. However we select methods where the predictor space is dividen in hight dimensional rectangles.

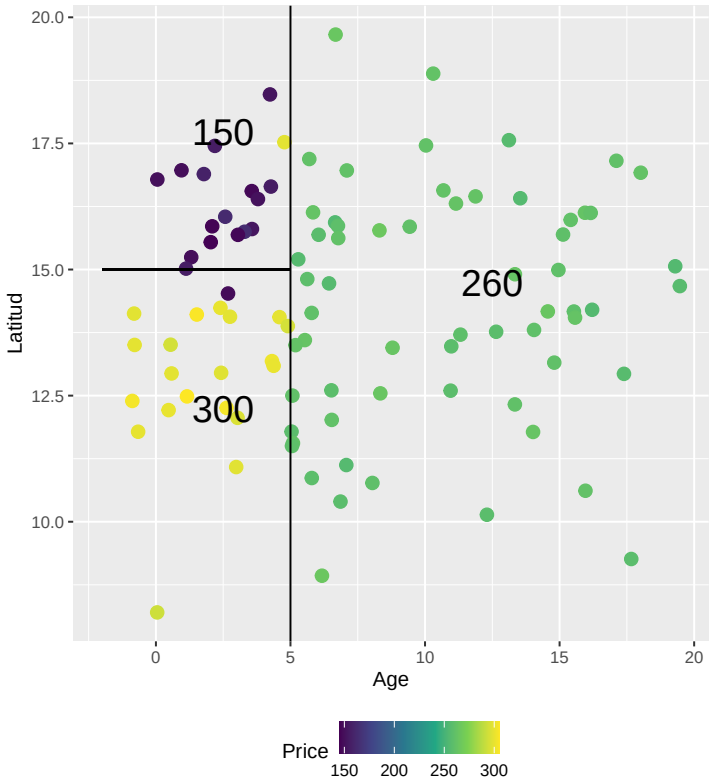
Regression trees (CART), examples

- Response: Appartment prices in thousands of USD in Montevideo
- Predictor variales: Construction age, latitude.
- Trees are composed with nodes, brunches and leaft.
- First node is the root node.
- Terminal nodes are leaves



Regression tree regions

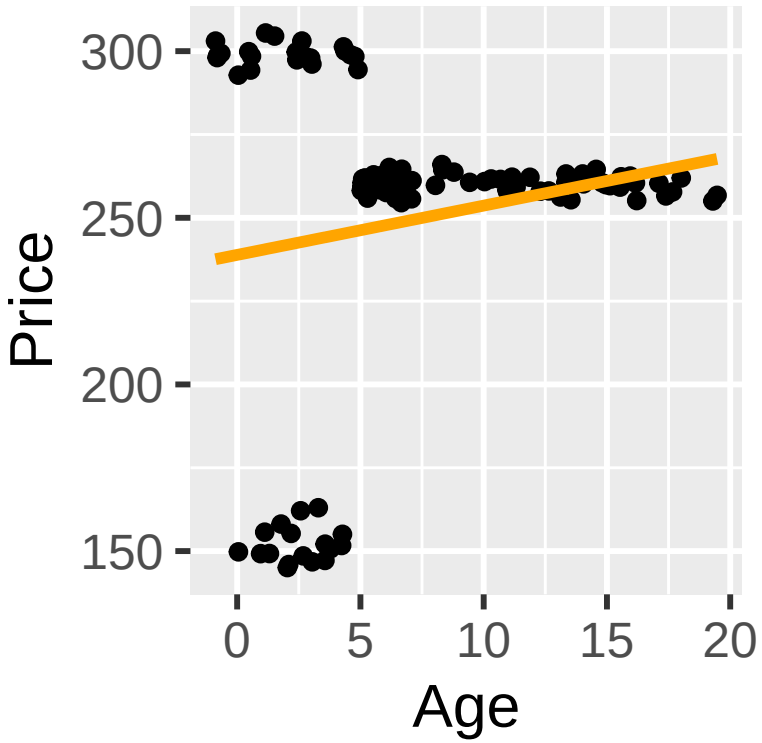
Predictor space partition



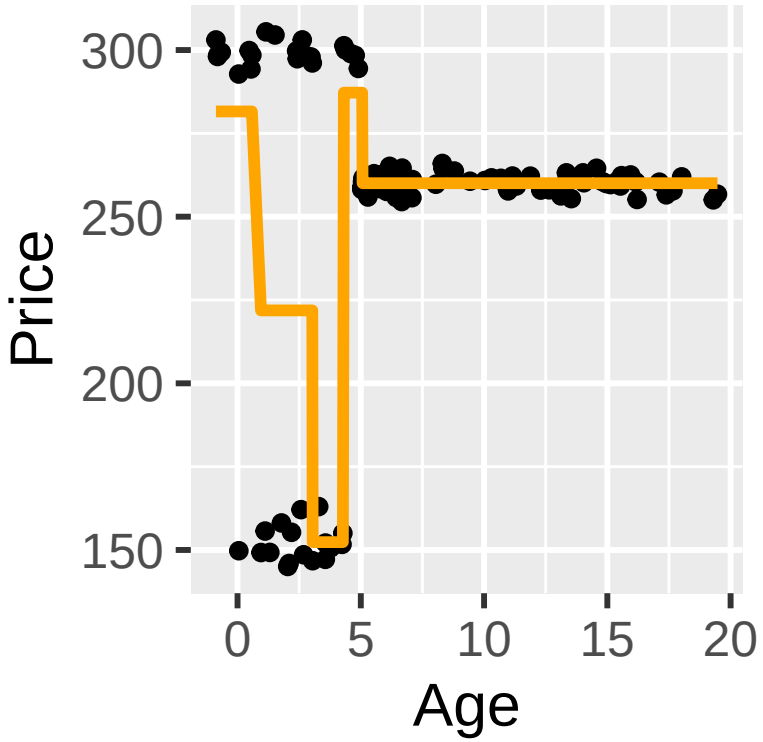
- First split Age$=5$, region Age<5 splits into $Lat = 15$ two regions are defined $Lat > 15$ and $Lat \leq 15$.
- This process defines 3 non overlap regions (R_1, R_2, R_3).

Lineal regression vs Regression Tree, housing example

$f(X) = \sum_{i=1}^{n} \beta X_i$



$f(X) = \sum_{m=1}^{M} c_m I\{(X) \in R_m\}$



Algorithm: growing a tree

1. All observations in a single set
2. Sort values on first variable
3. Compute split criteria for all possible splits into two sets
4. Choose the best split on this variable
5. Repeat 2-4 for all other variables
6. Choose the best split among all variables. Your data is now in two sets.
7. Repeat 1-6 on each subset.
8. Stop when stopping rule is achieved.

Main idea

Main idea is segmenting the predictor space into a number of simple regions, in each region a simple models is fitted.

The set of rules to partition the predictor space can be sumarized in a *decision tree*

$f(X)$ estimate:

Regression	Mean response in each leaft
Classification	most frequent class on the leaf

The objective will be to define regions $R_1, R_2 \dots R_M$ such that minimize RSS

$$RSS = \sum_{m=1}^{M} \sum_{i \in R_m} (y_i - \hat{y}_{R_m})^2$$

$\hat{y}_{R_m}$ is the mean response for the training observations within R_m region.

Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into M boxes.
Then a top-down gready approach is used, called recursive binary splitting.

Algorithm construction

Let(x_i, y_i) for $i = 1, \ldots N$ con $x_i = (x_{i1}, x_{i2}, \ldots x_{ip})$, regression tree constructions consists in tree basic steps :

- Begin with all the training data.
- Select X_j and s which defines the partition $\{X_j < s\}$. This is based on gready algorithms, select the best partition in each step.
- The data are divided in two 'sons': $R_1(j, s) = \{X | X_j \leq s\}$ y $R_2(j, s) = \{X | X_j > s\}$
- Repeat the procedure until reach the `stoping rule` criteria. Example: less than 5 observation per leaft.

How to find the best partition

In each step, find the best partition:

- $\{X_j < s\}$ divide the data in $R_1(j,s) = \{X|X_j \leq s\}$ y $R_2(j,s) = \{X|X_j > s\}$
- Select the variable j the cut point s such tha minimizes:

$$min_{j,s}\left[\sum_{x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2\right]$$

- $\hat{y}_{R_1}$ y $\hat{y}_{R_2}$ mean response in each region.
- Repeat the same procedure for all the regions until the stoping rule is met.

If Y is categorical, the evaluation have to changes.

How long do I let the tree grow?

The tree size is a tunning parameter and governs the tree complexity.

- Very big, overfit (more complex, less bias)
- Very small, can not capture important patterns in the data (simpler, smaller variance)

Different stoping rules

- Not reduction in test error
- Small number of data in the leaf

Prefered method is to grow a big tree (T_0) with some stoping rule and *pruning* T_0 based on some cost-complexity penalty.

Regression tree, Example

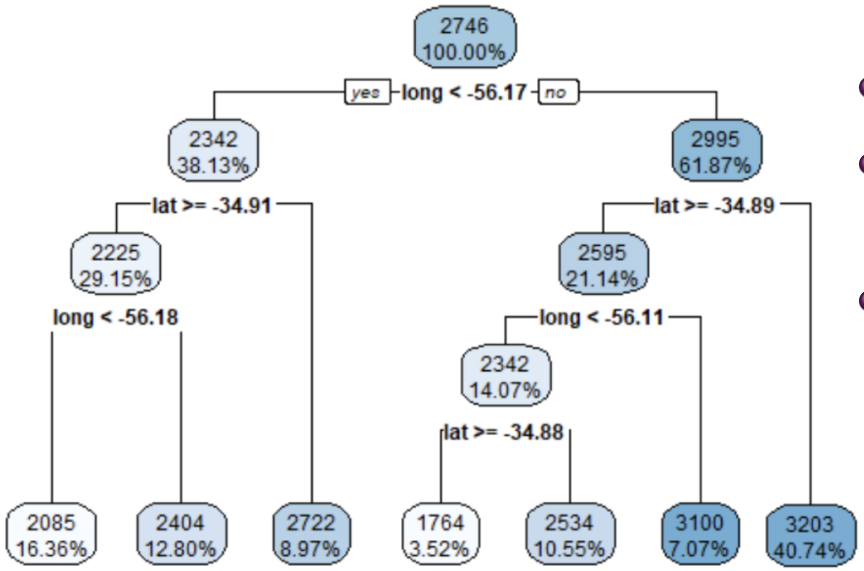
- **Objective:** Predict price for Montevideo Apartments
- **Data:** Apartments sales February 2018- January 2019. Data from Mercado libre

rpart R package

data are in the object dat_apt , fit a tree without pruning.

```
# Divide data in training and test
intrain <- sample(x = 1:nrow(dat_apt), size = nrow(dat_apt)*.8)
training <- dat_apt[intrain,]
testing <- dat_apt[-intrain,]

# rpart to fit the tree
# Two arguments: formula and data
# Just to describe the problem use lat and long as predictors
tree_apto <- rpart(precio_apt ~ lat + long, data = training)

# resulting tree structure
rpart.plot(tree_apt)
```

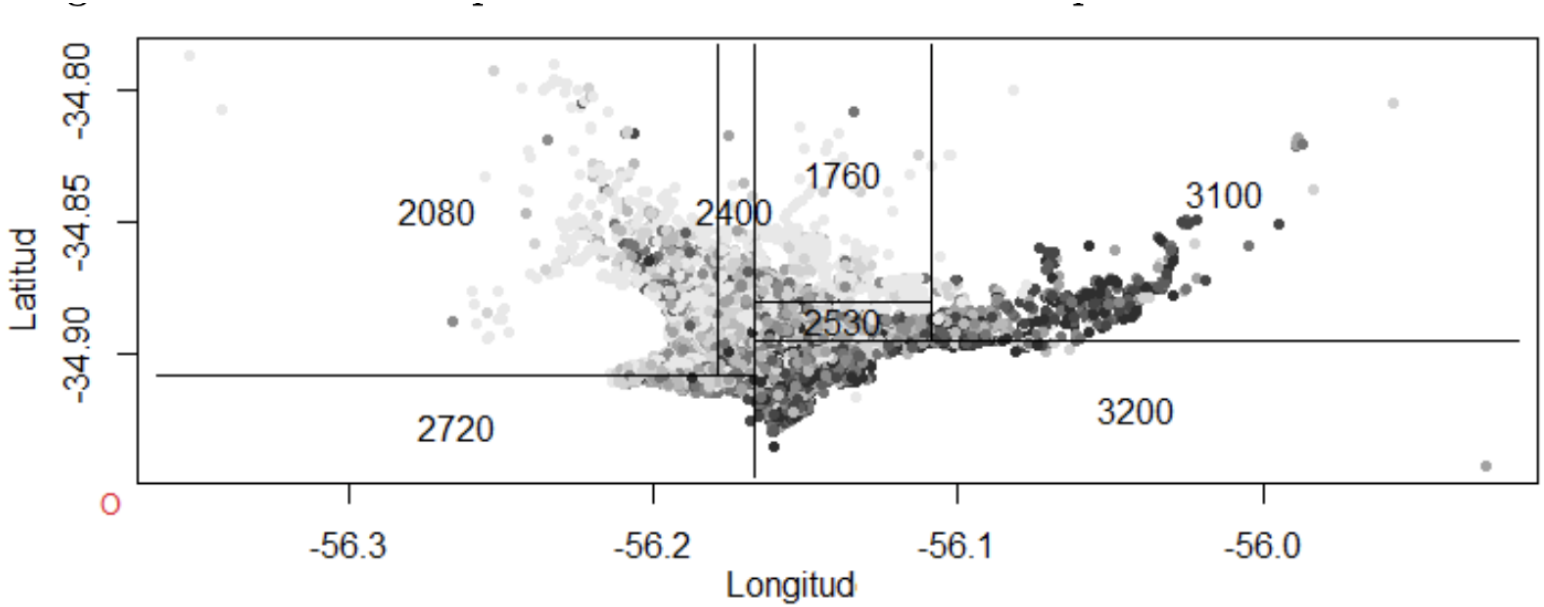
Regression tree, apartments price in MVD



- Mean price sqm in USD, percentage obs.
- Average apartment price in all the sample USD 2.746 per sqm.
- First partition on long (-56,17), at west the average price is USD 2.342 while east USD 2.995.

FCEA FACULTAD DE CIENCIAS ECONÓMICAS Y DE ADMINISTRACIÓN IESTA INSTITUTO DE ESTADÍSTICA UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Regression tree



Some characteristics

- Simple to use and interpret.
- Trees can handle a mix of predictor types, categorical and quantitative.
- Incorporates interactions and monotonic transformations atomatically.
- Trees efficiently operate when there are missing values in the predictors.

Some problems

- Can be unestable in particular if are not pruned
- In general the predictive error is bigger than other ML methods.
- Predicts a finite number of values, even when Y is continuous.

Classification

- The response variable have values in $\{1, \ldots, K\}$.
- Tree construction similar to regression.
- We need to modify:
  - $f(X)$ estimation
  - Criterio to node spliting and prune the tree

$f(X)$ estimate

Let $\hat{p}_{mk}$ the proportion of observations in the classs k in the nodo m,

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$

The observations in node m are classified based on **majority vote**,

$$k(m) = \text{argmax}_k \hat{p}_{mk}$$

majority class in node m.

Partition criterio for nodo m

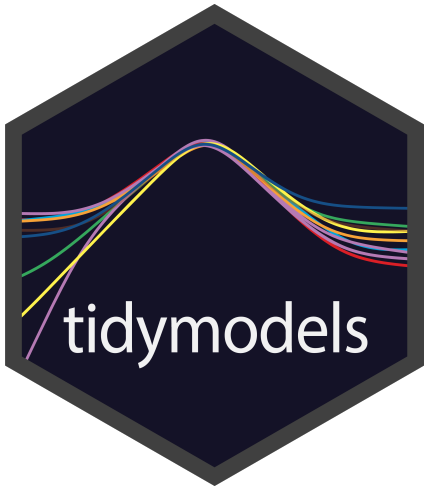
Several measures to evaluate the partitions.
Impurity measures for node m:

- **Classification error:** $\frac{1}{N_m}\sum_{i\in R_m} I(y_i \neq k(m)) = 1 - max_k(\hat{p}_{mk}(m))$ proportions of observations in the region R_m which are not from the majority class.
- **Gini index:** $\sum_{k\neq k'} \hat{p}_{mk}\hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk})$ is small if all the $\hat{p}_{mk}$ are close to 0 or 1, this means if the node has predominant observations from only one class (pure node).
- **Cross-entropy or deviance:** $-\sum_{k=1}^{K} \hat{p}_{mk} log \hat{p}_{mk}$ small value is the node m is pure.

Any of these measures can be used to guide the pruning of the tree having into account the cost-complexity, in general the classification error is used.

Other tree algorithms

- CHAID, C4.5, FACT, QUEST, CRUISE, GUIDE, CTREE, PPtree, etc.
- Main differences between them are the node partition method.
- Some use kernel methods, nearest neighbor, lineal partitions in a subset of selected variables.
- Can be bivariate trees or multiple partitions.
- Trees which use only one variable in each node partition generate partitions orthogonal to the axes. When more variables are used, partitions are obliques to the axes.



- Collection of packages for modeling and machine learning using tidyverse principles.
- `install.packages("tidymodels")`
- `library(tidymodels)` load the core packages and make them available in your current R session.

- `rsample` provides infrastructure for efficient data splitting and resampling
- `parsnip` is a tidy, unified interface for models which provides a fluent and standardized interface for a variety of different models.
- `recipes` recipes is a tidy interface to data pre-processing tools for feature engineering.
- `tune` helps you optimize the hyperparameters of your model and pre-processing steps.
- `yardstick` measures the effectiveness of models using performance metrics
- ...

Apartment values in Montevideo

`apt_redu` data set contains information from one neighborhood in Montevideo (Pocitos) with apartments prices and some apartment characteristics.
Regression problem, response variable log price in sqm (`lpreciom2`) the rest are used as predictor variables.
GitHub repo with slides and data

```
library(tidyverse)
library(here)
library(tidymodels)
library(rpart.plot)
data <- read_csv(here("Minicourse/apt_redu.csv"))
```

Data splitting

- The primary approach for empirical model validation is to split the existing pool of data into two distinct sets, the training set and the test set.
- This training set is usually the majority of the data
- The other portion of the data is placed into the test set
- This is held in reserve until one or two models are chosen as the methods most likely to succeed

First we set the random number stream with `set.seed()` to be able to reproduce the results
Save the split information for an 80/20 split of the data (80% training and 20% test)

```
set.seed(2023)
data_split <- initial_split(data, prop = 0.80)
data_split
```

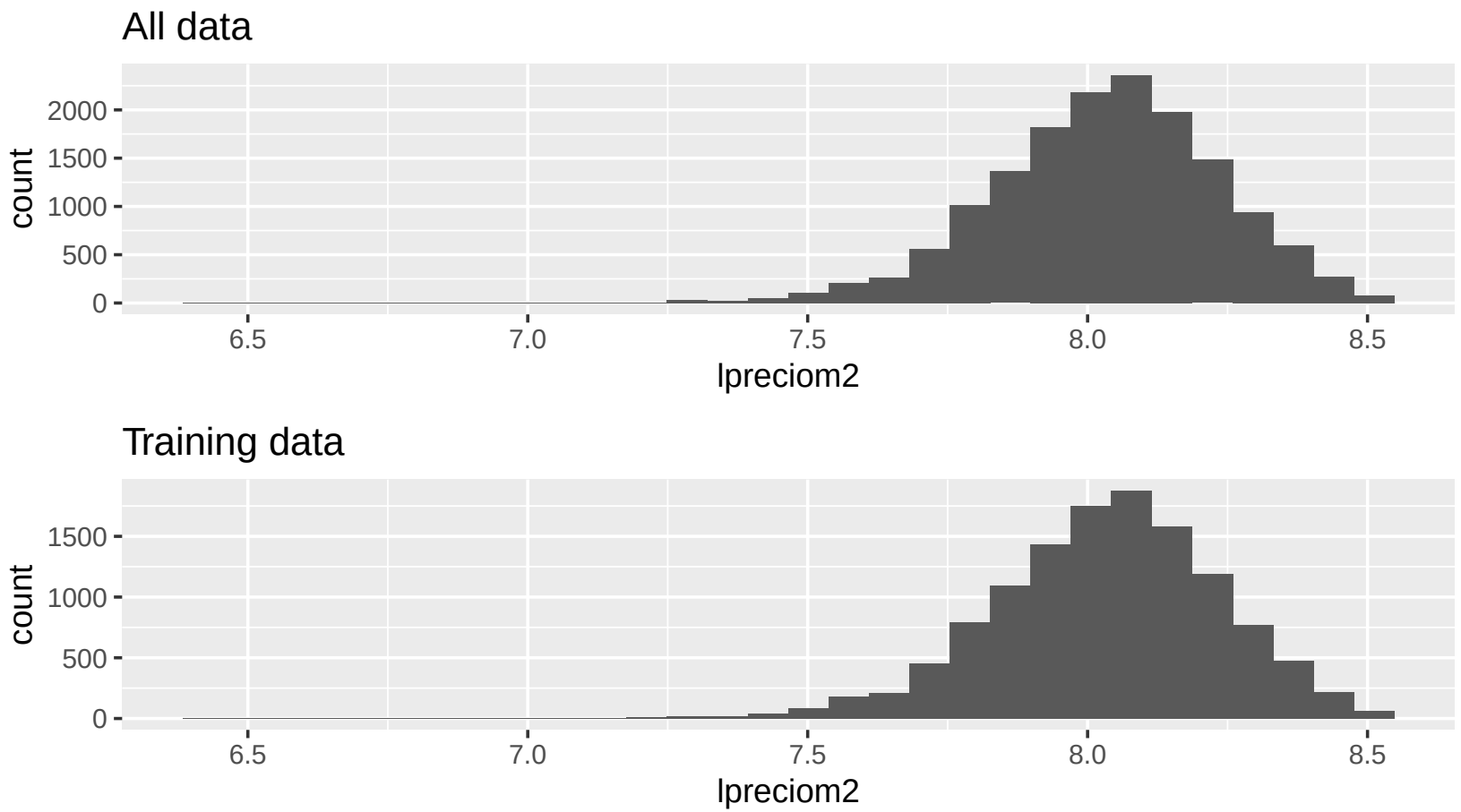
```
## <Training/Testing/Total>
## <12280/3070/15350>
```

The printed information denotes the amount of data in the training set

- The object `data_split` is an `rsplit` object and contains only the partitioning information
- To get the resulting data sets, we apply two more functions, `training` and `test`

```
data_train <- training(data_split)
data_test  <-  testing(data_split)

dim(data_train)

## [1] 12280    11
```

Distribution response in all data and training data



Fitting Models with parsnip

1. **Pick a model:** Specify the type of model based on its mathematical structure

2. **Set the engine:**, Most often this reflects the software package that should be used

3. **Set the mode (if needed):**, The mode reflects the type of prediction outcome. For numeric outcomes, the mode is regression; for qualitative outcomes, it is classification

Linear regression

```
lm_mod <- linear_reg() %>% # Step 1:  Pick a model

          set_engine("lm") # Step 2: Set the engine

lm_mod
```

```
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

`set_mode()` is note needed because we specify the model in the first step.

There are different ways to specify a linear regression selecting different engines
We can see the possible engines availables

```
show_engines("linear_reg")
```

```
## # A tibble: 7 x 2
##   engine mode
##   <chr>  <chr>
## 1 lm     regression
## 2 glm    regression
## 3 glmnet regression
## 4 stan   regression
## 5 spark  regression
## 6 keras  regression
## 7 brulee regression
```

Once the details of the model have been specified, the model estimation can be done with either the `fit()`function (to use a formula) or the `fit_xy()` function (when your data are already pre-processed).

```
lm_form_fit <-
  lm_mod %>%
  fit(lpreciom2 ~ lsup_constru + long, data = data)

lm_xy_fit <-
  lm_mod %>%
  fit_xy(
    x =  data %>% select(lsup_constru, long),
    y =  data %>% select(lpreciom2)
  )
```

```
lm_form_fit

## parsnip model object
##
##
## Call:
## stats::lm(formula = lpreciom2 ~ lsup_constru + long, data = data)
##
## Coefficients:
##  (Intercept)  lsup_constru          long
##     135.1762       -0.1666        2.2520
```

Decision tree

Link parsnip decision tree.

These tree specification can be used to create a classification tree. We can see the `parsnip` flexibility to create different models

```
tree_mod <- decision_tree() %>%
  set_engine("rpart")
```

With the model especification and the data we can fit the model

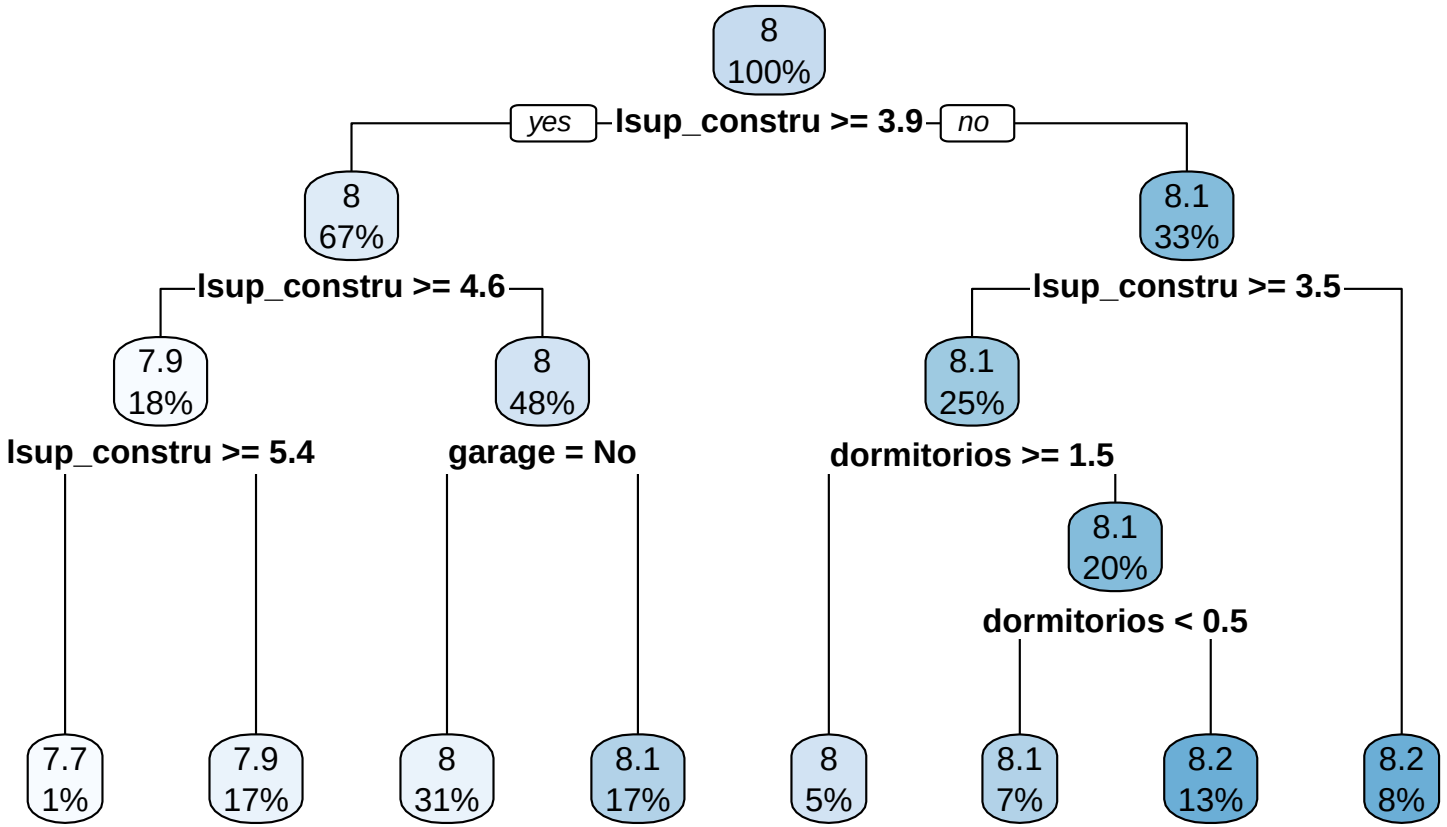
```
tree_fit <- tree_mod %>%
    set_mode("regression") %>%
  fit(lpreciom2 ~ ., data = data_train)
```

```
tree_fit

## parsnip model object
##
## n= 12280
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 12280 512.62880 8.031064
##    2) lsup_constru>=3.906998 8192 330.21000 7.980633
##      4) lsup_constru>=4.620047 2261  92.79654 7.897630
##        8) lsup_constru>=5.389061 154  10.65301 7.716920 *
##        9) lsup_constru< 5.389061 2107  76.74693 7.910838 *
##      5) lsup_constru< 4.620047 5931 215.89820 8.012275
##       10) garage=No 3816 143.90470 7.978957 *
##       11) garage=Si 2115  60.11451 8.072389 *
##    3) lsup_constru< 3.906998 4088 119.83370 8.132123
##      6) lsup_constru>=3.481122 3078  92.45661 8.103078
```

```
tree_fit %>%
  extract_fit_engine() %>%
  summary()

## Call:
## rpart::rpart(formula = lpreciom2 ~ ., data = data)
##   n= 12280
##
##           CP nsplit rel error    xerror       xstd
## 1 0.12208652      0 1.0000000 1.0000572 0.01668184
## 2 0.04197041      1 0.8779135 0.8781177 0.01556716
## 3 0.02317277      2 0.8359431 0.8392745 0.01498730
## 4 0.02050192      3 0.8127703 0.8156918 0.01493354
## 5 0.01129189      4 0.7922684 0.7943303 0.01474401
## 6 0.01052729      6 0.7696846 0.7617377 0.01449786
## 7 0.01000000      7 0.7591573 0.7606741 0.01450776
##
## Variable importance
## lsup_constru  dormitorios        banos    lexpensas       garage          a
```

When the tree has to many nodes is more complex to read the original output. The `rpart.plot` package has some usefult functions to visualize the tree structure in a simple way. Only works for tree objects fitted with `rpart`

```
tree_fit %>%
  extract_fit_engine() %>%
  rpart.plot(roundint=FALSE)
```



FCEA FACULTAD DE CIENCIAS ECONÓMICAS Y DE ADMINISTRACIÓN IESTA INSTITUTO DE ESTADÍSTICA UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Predictions

augment will add column(s) for predictions to the given data

```
augment(tree_fit , new_data = data_test) %>%
  select(1,12,13)
```

```
## # A tibble: 3,070 x 3
##    lpreciom2 .pred   .resid
##        <dbl> <dbl>    <dbl>
##  1      7.98  8.07 -0.0885
##  2      8.24  8.16  0.0774
##  3      8.33  8.16  0.175
##  4      7.85  7.98 -0.125
##  5      8.07  8.07 -0.00322
##  6      8.09  8.16 -0.0720
##  7      8.04  8.02  0.0205
##  8      8.26  8.02  0.237
##  9      8.23  7.91  0.319
## 10      7.72  7.98 -0.262
```

```
predict(tree_fit, new_data = data_test)
```

```
## # A tibble: 3,070 x 1
##    .pred
##    <dbl>
##  1  8.07
##  2  8.16
##  3  8.16
##  4  7.98
##  5  8.07
##  6  8.16
##  7  8.02
##  8  8.02
##  9  7.91
## 10  7.98
## # ... with 3,060 more rows
```

```
augment(tree_fit , new_data = data_test) %>%
  rmse(truth = lpreciom2 , estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.168
```

How to check all possible engines for decision trees?

```
show_engines("decision_tree")
```

```
## # A tibble: 5 x 2
##   engine mode
##   <chr>  <chr>
## 1 rpart  classification
## 2 rpart  regression
## 3 C5.0   classification
## 4 spark  classification
## 5 spark  regression
```

Your turn!

1. To the complete data set, create a new response variable `lpreciom2_c`, categorical variable with two levels, high if `lpreciom2` is bigger or equal than the mean of `lpreciom2` and low if `lpreciom2` is smaller than its mean value. Save the data in an object called `new_data`
2. In the `new_data` set select all the variables but `lpreciom2` because it is a perfect predictor for `lpreciom2_c`
3. Divide in training and test the `new_data`
4. Using the object `tree_mod` fit a classification tree with the new variable

Solution: Part 1 and 2

```
new_data <- data %>%
  mutate(lpreciom2_c = as.factor(case_when(lpreciom2< mean(lpreciom2)~'
                                lpreciom2>=mean(lpreciom2)~'high'))) %>%
                                select(-lpreciom2)

#Altenative
new_data <-data %>%
            mutate(lpreciom2_c =
                     as.factor(ifelse(lpreciom2< mean(lpreciom2),
                                      'low','high'))) %>%
  select(-lpreciom2)
```

Part 3

```
set.seed(2023)
data_split_c <- initial_split(new_data)

data_train_c <- training(data_split_c)
data_test_c <- testing(data_split_c)
```

Part 4

```
tree_fit_cl <- tree_mod %>%
  set_mode("classification") %>%
  fit(lpreciom2_c ~ ., data = data_train_c)
```

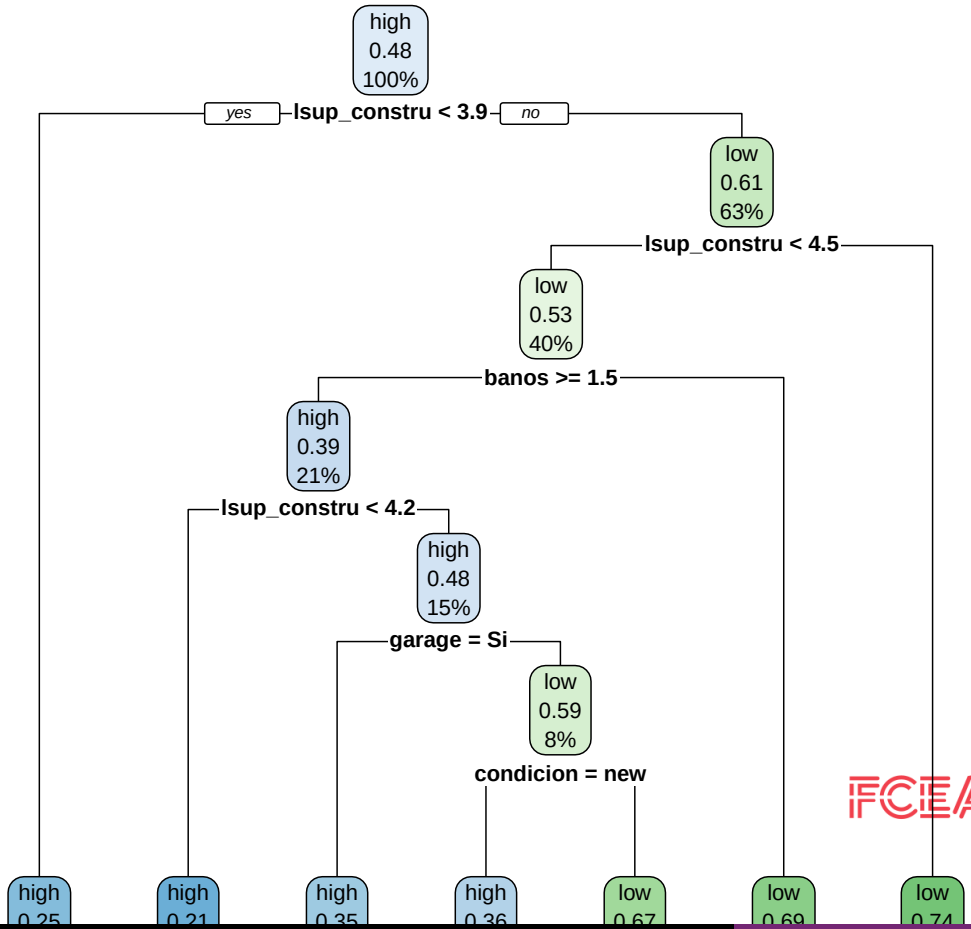
Your turn!

1. Plot the tree structure
2. Compute the accuracy in the test set

Solution: Part 1

```
tree_fit_cl %>%
 extract_fit_engine() %>%
  rpart.plot(roundint=FALSE)
```

Solution: Part 1



Solution: Part 2

```
augment(tree_fit_cl, new_data = data_test_c ) %>%
accuracy(truth = lpreciom2_c , estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.735
```

You can get also the confussion matrix `conf_mat(truth =lpreciom2_c , estimate = .pred_class)`

MORE! Serch for all parsnip models