

## Lista 3 - Mineração

Victor Alves Dogo Martins, RA: 744878      Ana Beatriz Alves Monteiro, RA: 727838  
Larissa Torres, RA: 631914

14-08-2022

### Itens 1 e 2

Para a transformação das variáveis categóricas (**US**, **Urban** e **ShelveLoc**) em dummies, utilizamos a função `model.matrix()`, que as transforma automaticamente e são apresentadas da seguinte maneira:

- **US:** transformada na variável  $X_{USYes}$ :

$$X_{USYes} = \begin{cases} 1, \text{ caso a loja estiver localizada nos EUA;} \\ 0, \text{ caso contrário.} \end{cases}$$

- **Urban:** transformada na variável  $X_{UrbanYes}$ :

$$X_{UrbanYes} = \begin{cases} 1, \text{ caso a loja estiver localizada na zona urbana;} \\ 0, \text{ caso contrário.} \end{cases}$$

- **ShelveLoc:** transformada nas variáveis  $X_{ShelveLocGood}$   $X_{ShelveLocMedium}$ :

$$X_{ShelveLocGood} = \begin{cases} 1, \text{ caso o produto tiver localização boa na prateleira;} \\ 0, \text{ caso contrário.} \end{cases}$$

$$X_{ShelveLocMedium} = \begin{cases} 1, \text{ caso o produto tiver localização média (mas não boa) na prateleira;} \\ 0, \text{ caso contrário.} \end{cases}$$

Por outro lado, como feito na Lista 2, a divisão do banco de dados entre treino e teste foi feita com o auxílio da função `initial_split()` do pacote `{rsample}`:

```
## Lendo pacotes

set.seed(1)

library(progress)
library(tidyverse)
library(rsample)
library(knitr)
```

```

library(kableExtra)
library(caret)
library(FNN)
library(rpart)
library(rpart.plot)
library(randomForest)
library(locfit)
library(pdp)

## Definindo funcao de risco (utilizada na lista 2)

funcao_risco <- function(y_pred, y_obs){

  w <- (y_pred-y_obs)^2
  sigma <- var(w)
  risco <- mean(w)
  liminf <- risco - (2*sqrt((1/length(w))*sigma))
  limsup <- risco + (2*sqrt((1/length(w))*sigma))

  return(data.frame(risco, liminf, limsup))
}

## Lendo Dados

df <- ISLR::Carseats |>
  mutate(US=as.factor(US),
         Urban=as.factor(Urban),
         ShelveLoc=as.factor(ShelveLoc))

## Divisão entre treino e teste

split <- initial_split(df, prop=0.6)

tre <- training(split)
tes <- testing(split)

x_tre <- model.matrix(Sales~., tre)
y_tre <- tre[,1]

x_tes <- model.matrix(Sales~., tes)
y_tes <- tes[,1]

```

## Item 3

### KNN

O KNN ou vizinho mais proximo tem como objetivo auxiliar no problema de predição usando classificação. Para isso é calculada a distancia entre a observação a ser predita e seus k vizinhos mais próximos. Isso possibilita que o número de venda de cadeirinhas de determinada observação seja predito usando a proximidade de outras k observações. Sendo assim, um ponto importante é calcular quantos vizinhos são necessários de serem considerados a fim de minimizar o risco estimado. Por isso, realizamos a validação cruzada com o nosso grupo de treino e teste, com código disponibilizado a seguir:

```

### ITEM 3

## KNN

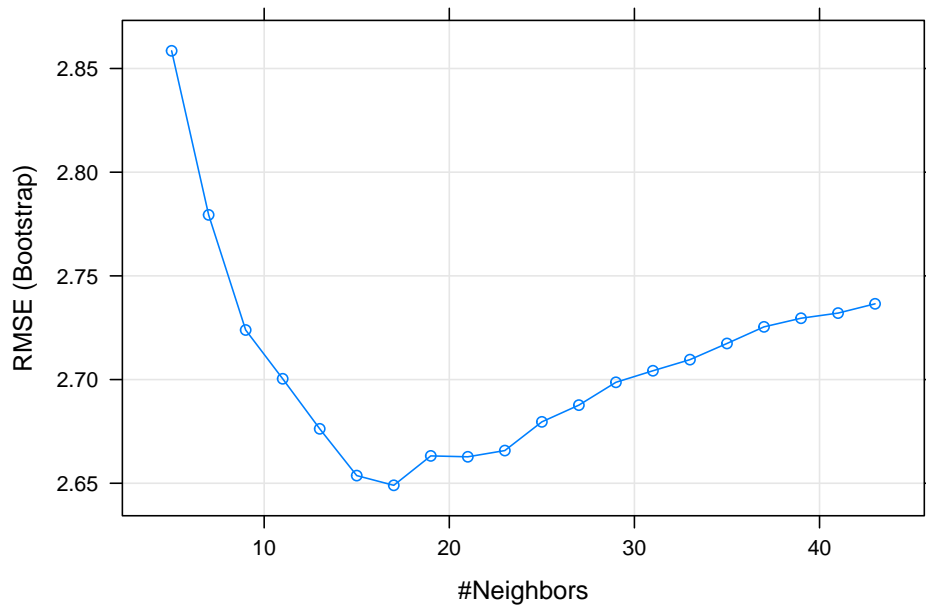
# Realizando calculo do melhor K

ajuste_knn <- train(
  x=x_tre,
  y=y_tre,
  method = 'knn',
  tuneLength = 20
)

# Plotando grafico de K vs Risco

plot(ajuste_knn)

```



```

paste0('O melhor K é: ', ajuste_knn$bestTune)

```

```

## [1] "O melhor K é: 17"

```

Com o gráfico acima e o resultado apresentado via função `paste0`, nosso `k` ótimo é igual a 17. Seguindo a lógica, para encontrar a quantidade de cadeirinhas vendidas de uma nova observação seriam utilizadas 17 vizinhos (observações) para realizar a predição com o menor risco estimado.

## Floresta Aleatória

Random Forest é um método utilizado para melhorar a predição de uma árvore cujo o método é classificatório e mais simples e sujeito a maiores erros. Então é realizado o ajuste de diversas árvores sendo que para cada

ajuste de árvore ocorre uma aleatorização de escolhas de variáveis disponíveis no banco de dados, o número de variáveis escolhidas é sempre bem menor que o número de variáveis do banco de dados, consequentemente por serem variáveis escolhidas de forma aleatória temos árvores com baixa correlação entre si e gerando predições cada vez melhores.

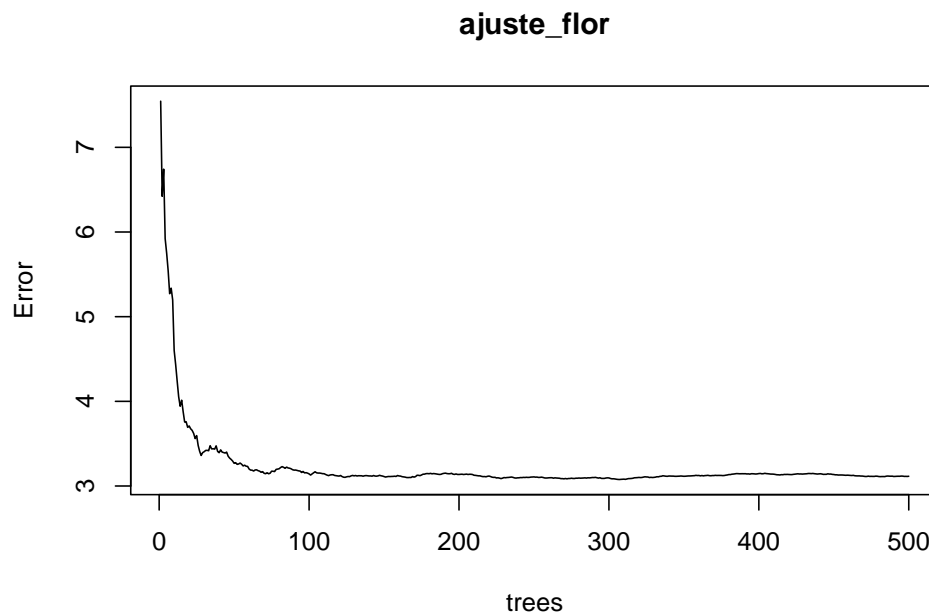
```
## Floresta Aleatória

# Realizando Ajuste

ajuste_flor <- randomForest(x=x_tre, y=y_tre, mtry = round(ncol(df)/3),
                           importance=TRUE)

# Ajuste x Numero de Arvores

plot(ajuste_flor)
```



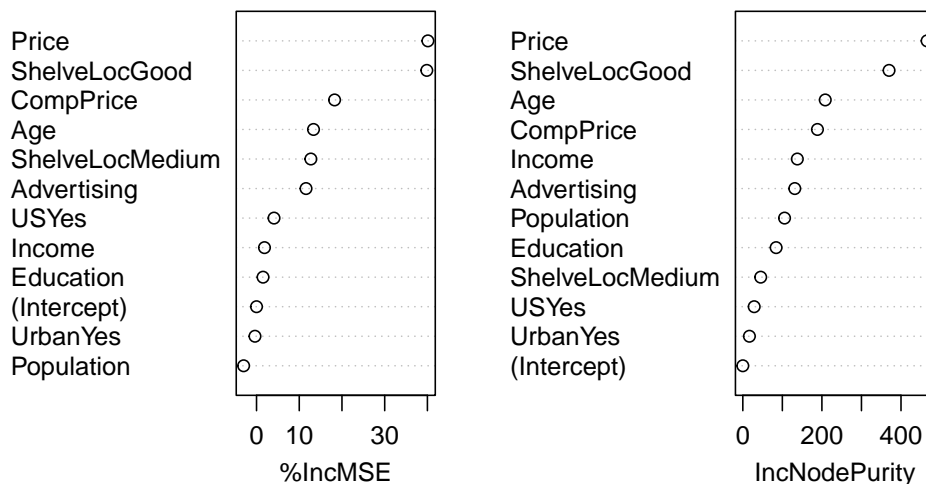
Após realizar o ajuste temos a função de risco, note que o erro cai rapidamente a medida que vamos aumentando o número de árvores geradas, e se estabiliza a partir de 200 árvores geradas, isso acontece porque a medida que aumentamos o número de árvores aumentamos a correlação dos dados, praticamente alguns sorteios podem acabar se repetindo, isso acontece mais rapidamente principalmente quando temos poucas variáveis disponíveis.

Para mensurar o nível de importância, podemos observar o quando o risco estimado diminuí ao adicionar uma variável específica.

```
# Importancia

varImpPlot(ajuste_flor)
```

## ajuste\_flor



Acima, temos que as variáveis que mais discriminam a variável resposta, ou seja, de maiores níveis de significância é, o preço e a boa localização: ao adicionar uma dessas duas variáveis no “crescimento” da árvore ou na sua composição, temos riscos menores.

## Árvore de Regressão

A árvore de regressão é um método não paramétrico classificatório que consiste em realizar uma divisão no espaço das covariáveis de acordo com a variável resposta. As condições são chamadas de nós, gerando as folhas, na qual contém a média da variável resposta respectiva a variável preditora contida naquele grupo.

### ## Árvore de Regressão

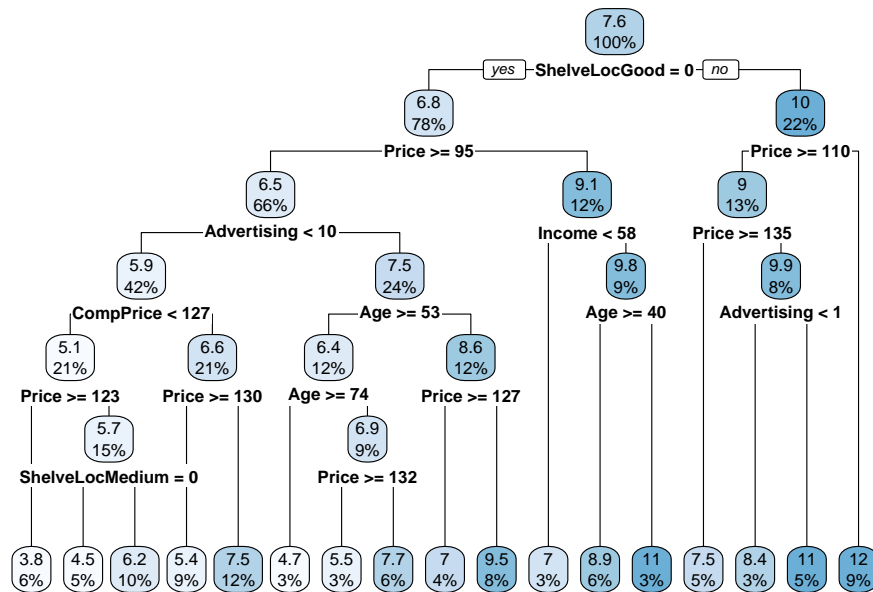
```
tre_arv <- data.frame(y_tre, x_tre[, -1])
```

Com o auxílio da função “rpart” foi gerado a árvore, e para a sua poda foi considerado o melhor cp obtido através do componente “cptable”. Com a poda da árvore controlamos a variância, o número de observações em cada folha e possíveis Overfitting.

### # Ajustando para melhor cp automatico

```
ajuste_arv <- rpart(y_tre~, data=tre_arv, method='anova')
melhor_cp <- ajuste_arv$cptable[which.min(ajuste_arv$cptable[, 'xerror']), 'CP']
ajuste_arv <- prune(ajuste_arv, cp=melhor_cp)

rpart.plot(ajuste_arv)
```



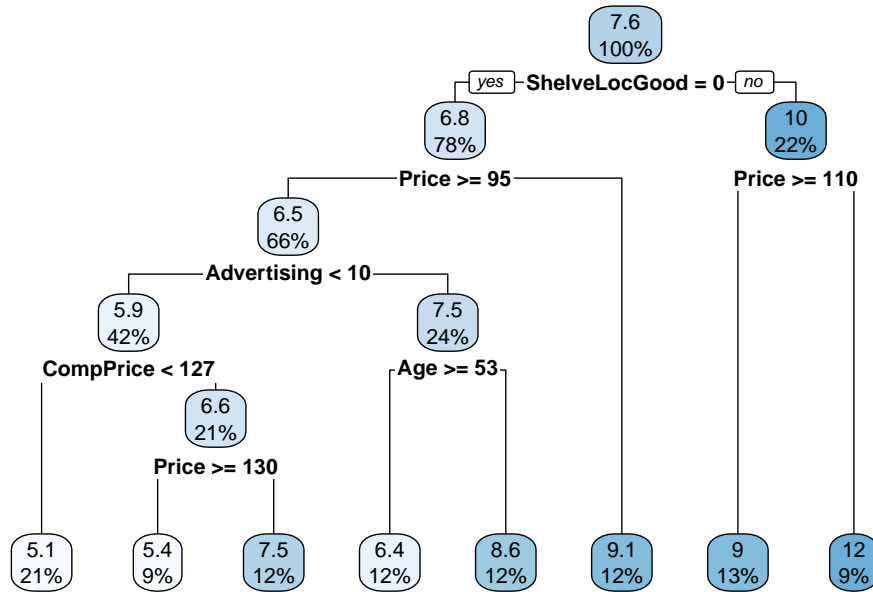
Assim, temos a primeira árvore obtida, onde a mais pura quebra é dada pela a variável “Boa localização na prateleira “, sendo que o produto estar em uma boa localização tem uma média de venda de 10 e não estar resulta em uma venda média igual a 6.8. Em seguida a árvore realiza a quebra em mais 6 variáveis distintas, sendo que a primeira folha da esquerda apresenta o pior grupo com uma média de 3.8, e a primeira folha da direita apresenta a melhor média de vendas de cadeirinhas, 12. Note que, existem algumas inversões entre o pior e a melhor folha, e junto a isso temos algumas folhas com poucas observações.

A poda da árvore pode resultar em quebras melhores, e é o que faremos a seguir:

```
# Ajustando para melhor cp na mão

ajuste_arv <- rpart(y_tre~., data=tre_arv, method='anova')
ajuste_arv <- prune(ajuste_arv, cp=0.027)

rpart.plot(ajuste_arv)
```



Após realizar a poda, temos uma árvore com melhores quebras, uma interpretação mais clara, menos variáveis e melhor ordenação da média de vendas do produto.

Foram geradas 8 classificações ordenadas sendo que a pior classificação tem a média igual a 5.1 e a melhor classificação tem a média equivalente a 12. Interpretando em termos do problema, o produto não estar em uma boa localização tem uma média de vendas de 6.8, o que piora essa venda para 6.5 caso o preço for maior ou igual a 95, o nível de propaganda ser maior do que 10 pode impulsionar a média de vendas para 7.5, em contrapartida piorar para 5.9, por fim, o produto ser vendido por mais de 127 despenca a média de vendas para 5.1. A interpretação é análoga para os outros ramos da árvore.

## Nadaraya-Watson

Pelo que foi visto em sala de aula, sabemos que a função de regressão ajustada via Nadaraya-Watson é dada por:

$$g(x) = \sum_{i=1}^n w_i(x) \cdot y_i$$

Onde  $x$  é uma observação do conjunto de teste,  $y_i$  é a  $i$ -ésima variável resposta do conjunto de treino e  $w_i$  é o  $i$ -ésimo peso referente à  $i$ -ésima observação do conjunto de treino e a observação do teste utilizada.  $w_i$ , especificamente, mede o quão parecido  $x$  é com a  $i$ -ésima observação do conjunto de treino, e é dado por:

$$w_i(x) = \frac{K(x, x_i)}{\sum_{j=1}^n K(x, x_j)}$$

Onde  $K$  é a função kernel (em nosso caso, utilizaremos o Kernel Gaussiano),  $x_i$  é a  $i$ -ésima observação do conjunto de treino e  $x$  é uma observação do conjunto de teste. A soma no numerador da fração é a soma dos resultados da função  $K$  para todas as observações do treino com a observação nova do teste, funcionando como uma constante de normalização.

A função Kernel Gaussiana é dada por:

$$K(x, x_i) = (\sqrt{2\pi h^2})^{-1} \exp \left\{ -\frac{d^2(x, x_i)}{2h^2} \right\}$$

Onde  $d(x, x_i)$  é a distância euclidiana da observação  $x$  do conjunto de teste e a  $i$ -ésima observação do treino e  $h$  é um *tuning-parameter* que escolheremos por validação cruzada.

Como orientado, tivemos que ajustar esta metodologia com uma função própria, e abaixo seguem os códigos utilizados devidamente comentados (tanto o do ajuste quanto o da escolha do *tuning-parameter*).

```
## Nadaraya-Watson

# Função de Ajuste do NW

nw_func <- function(x_tes, x_tre, y, h){

  # Normalizando covariáveis de treino e teste

  x_tes <- scale(x_tes)
  x_tre <- scale(x_tre)

  # Definindo Kernel Gaussiano

  k <- function(h,d){(1/(h*sqrt(2*pi)))*exp(-0.5* (d/h)^2)}

  # Calculando matriz de distancia euclidiana (linha=i-esima obs. de teste,
  #coluna=j-esima obs. de treino)

  dis <- fields::rdist(x_tes[,1], x_tre[,1])

  # Definindo vetor vazio para y predito

  y_pred <- numeric(ncol(x_tes))

  # Para cada i-esima linha do conjunto de teste...

  for (i in 1:nrow(dis)) {

    # Criando vetor vazio para resultado dos kernels

    kx <- numeric(ncol(dis))

    # Para cada j-esima linha do conjunto de treino...

    for (j in 1:ncol(dis)) {

      # Calculando j-esimo kernel com base em h e distancia entre i-esima obs do teste e
      # j-esima obs do treino

      kx[j] <- k(h, dis[i,j])

    }

    # Calculando vetor de pesos
```



```

    wx <- kx/sum(kx)

    # Calculando i-esimo y predito
    y_pred[i] <- sum(wx*y)
  }

  # Retornando y preditos
  return(y_pred)
}

# Funcao de melhor h via validacao cruzada no treino
melhor_h_nw <- function(x_tre, y_tre, seed=1){

  set.seed(seed)

  # Embaralhando dados

  df <- data.frame(y_tre,x_tre)
  df <- df[sample(1:nrow(df)),]

  # Definindo kfolds com k=5

  size <- round(nrow(df)/5)

  # Lista com cada fold

  kfoldlist <- list(
    df[1:size,],
    df[(size+1):(2*size),],
    df[(2*size+1):(3*size),],
    df[(3*size+1):(4*size),],
    df[(4*size+1):(5*size),]
  )

  # Definindo vetor de h para ser testado

  h <- seq(0.4,10,0.1)

  # Dataframe com resultados

  result <- data.frame(
    h=h,
    risco=numeric(length(h))
  )

  # Barra de progresso

  pb <- progress_bar$new(

```

```

total=length(h),
format = "[:bar] :percent eta: :eta elapsed: :elapsed")

for (jj in 1:length(h)) {

  hresult <- numeric(5)

  for (ii in 1:5) {

    # Definindo conjunto de treino e teste para i-esima iteracao

    df_tre <- do.call(rbind.data.frame, kfoldlist[-ii])

    df_tes <- kfoldlist[[ii]]

    # Ajustando

    ypred <- nw_func(x_tes=df_tes[,-1],
                     x_tre=df_tre[,-1],
                     y=df_tre[,1],
                     h=h[jj])

    # Calculando risco

    hresult[ii] <- funcao_risco(y_pred=ypred, y_obs=df_tes[,1])$risco[1]

  }

  # Risco medio do j-esimo h

  result$risco[jj] <- mean(hresult)

  pb$tick()
  Sys.sleep(1 / length(h))

}

# Retornando data.frame com resultados

return(result)

}

```

Com a função definida, primeiramente visualizaremos de que forma o risco estimado via validação cruzada no conjunto de treino se comporta com relação ao parâmetro  $h$ , escolhendo o valor que retorna o melhor risco.

```

# Calculando risco para cada H

resultados_h <- melhor_h_nw(x_tre=x_tre,y_tre=y_tre)

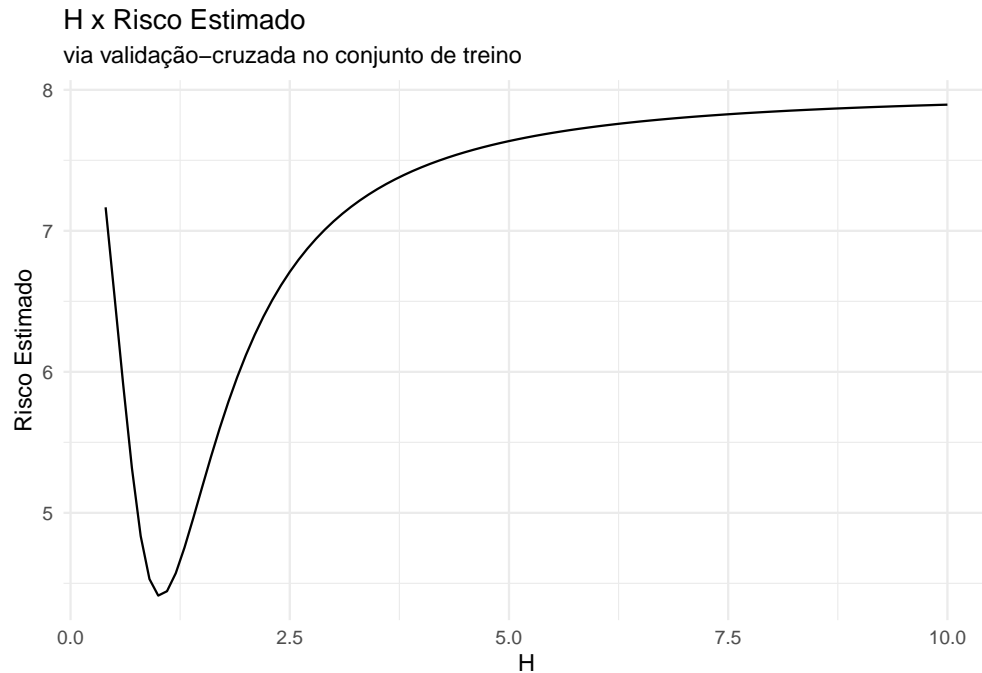
resultados_h |>
ggplot()+

```

```

aes(x=h,y=risco)+
geom_line()+
labs(x='H', y='Risco Estimado',
      title='H x Risco Estimado',
      subtitle = 'via validação-cruzada no conjunto de treino')+
theme_minimal()

```



```

# Encontrando melhor H

paste0('O melhor H é: ', resultados_h[resultados_h$risco==min(resultados_h$risco),][1])

```

```
## [1] "O melhor H é: 1"
```

Ainda que não fique tão claro pelo gráfico, dada a quantidade de valores, temos que a função criada nos retornou um valor ótimo de  $h = 1$ . Assim, podemos prosseguir para o ajuste.

```

# Realizando ajuste Nadaraya-Watson com melhor H

ajuste_nw <- nw_func(x_tes, x_tre, y_tre,
                     h=resultados_h[resultados_h$risco==min(resultados_h$risco),]$h)

```

## Item 4

```

### ITEM 4

funcao_risco <- function(y_pred, y_obs){

```

```

w <- (y_pred-y_obs)^2
sigma <- var(w)
risco <- mean(w)
liminf <- risco - (2*sqrt((1/length(w))*sigma))
limsup <- risco + (2*sqrt((1/length(w))*sigma))

return(data.frame(risco, liminf, limsup))
}

# Apresentando riscos e ICs com 95% de confiança para cada ajuste

tibble(
  `Variável`=c("Risco Estimado", "Limite Inferior", "Limite Superior"),
  `Nadaraya-Watson`=as.numeric(funcao_risco(ajuste_nw, y_tes)),
  `KNN`=as.numeric(funcao_risco(predict(ajuste_knn, x_tes), y_tes)),
  `Floresta Aleatória`=as.numeric(funcao_risco(predict(ajuste_flor, x_tes), y_tes)),
  `Árvore de Regressão`=as.numeric(funcao_risco(predict(ajuste_arv,
                                                    data.frame(x_tes)), y_tes))
) |>
kable('latex', align='cccc',
      caption = 'Risco e Intervalos de Confiança para Ajustes') |>
kable_styling(position="center",
              latex_options="HOLD_position")

```

Table 1: Risco e Intervalos de Confiança para Ajustes

| Variável        | Nadaraya-Watson | KNN      | Floresta Aleatória | Árvore de Regressão |
|-----------------|-----------------|----------|--------------------|---------------------|
| Risco Estimado  | 3.925197        | 7.337696 | 3.050677           | 4.863114            |
| Limite Inferior | 3.033761        | 5.746294 | 2.221421           | 3.764944            |
| Limite Superior | 4.816633        | 8.929098 | 3.879933           | 5.961284            |

Dentre os métodos ajustados nesta lista, o que apresentou melhor desempenho foi o ajuste via Florestas Aleatórias, com risco estimado de 3.05, com 95% de confiança que o risco esteja entre 2.22 e 3.87 (assim, apresentando a menor amplitude intervalar). Após ele, o ajuste via Nadaraya-Watson apresentou desempenho relativamente bom, com risco estimado de 3.92 e 95% de confiança de que ele está entre 3.03 e 4.82.

Se comparados aos métodos da lista anterior, apresentam desempenho inferior (MQ e Lasso demonstraram risco estimado de aproximadamente 1.2). Isso se dá pelo fato de estarmos tratando, nesta lista, de métodos menos específicos para a regressão (comumente utilizados para classificação). Por outro lado, ajustes via Mínimos Quadrados e com penalização via Lasso são mais adequados para estes fins e que, no geral, retornam melhores predições no contexto de regressão.

## Item 5

Para este item, procederemos de forma similar ao que foi feito anteriormente, mas realizando validação cruzada e calculando apenas a estimativa pontual do risco para os quatro métodos trabalhados. Isso foi realizado através de um for loop, e os códigos utilizados estão disponibilizados a seguir:

```
### ITEM 5
```

```

set.seed(1999)

# Embaralhando dados

df_k <- df[sample(1:nrow(df)),]

# Separando 5 lotes

size <- round(nrow(df_k)/5)

kfoldlist <- list(
  df_k[1:size,],
  df_k[(size+1):(2*size),],
  df_k[(2*size+1):(3*size),],
  df_k[(3*size+1):(4*size),],
  df_k[(4*size+1):(5*size),]
)

# Calculando risco para cada metodo

result_kfold <- data.frame(
  `Método`=c('KNN', 'Nadaraya-Watson', 'Árvore de Regressão', 'Floresta Aleatória'),
  `Risco`=numeric(4)
)

knn <- numeric(5)
nw <- numeric(5)
arv <- numeric(5)
flor <- numeric(5)

for (ii in 1:5) {

  # Definindo dados de treino e de teste

  df_tre <- do.call(rbind.data.frame, kfoldlist[-ii])

  df_tes <- kfoldlist[[ii]]

  x_tre <- model.matrix(Sales~., df_tre)
  y_tre <- df_tre[,1]

  x_tes <- model.matrix(Sales~., df_tes)
  y_tes <- df_tes[,1]

  # Realizando ajustes

  ## KNN

  aj_knn <- train(
    x=x_tre,
    y=y_tre,
    method = 'knn',
    tuneLength = 20
  )
}

```

```

)

aj_knn <- knn.reg(train=x_tre, test=x_tes, y=y_tre, k=aj_knn$bestTune$k)

## Floresta Aleatória

aj_flor <- randomForest(x=x_tre, y=y_tre, mtry = round(ncol(df)/3),
                        importance=TRUE)

## Árvore de Regressão

tre_arv <- data.frame(y_tre, x_tre[, -1])

aj_arv <- rpart(y_tre ~ ., data=tre_arv, method='anova')
melhor_cp <- aj_arv$cptable[which.min(aj_arv$cptable[, 'xerror']), 'CP']
aj_arv <- prune(aj_arv, cp=melhor_cp)

## Nadaraya Watson

melhor_h <- melhor_h_nw(x_tre, y_tre, seed=12)

aj_nw <- nw_func(x_tes, x_tre, y_tre,
                h=melhor_h[melhor_h$risco == min(melhor_h$risco), ]$h)

# Calculando risco

knn[ii] <- funcao_risco(aj_knn$pred, y_tes)$risco
nw[ii] <- funcao_risco(aj_nw, y_tes)$risco
arv[ii] <- funcao_risco(predict(aj_arv, data.frame(x_tes)), y_tes)$risco
flor[ii] <- funcao_risco(predict(aj_flor, x_tes), y_tes)$risco

}

result_kfold[1,2] <- mean(knn)
result_kfold[2,2] <- mean(nw)
result_kfold[3,2] <- mean(arv)
result_kfold[4,2] <- mean(flor)

result_kfold |>
  kable('latex', align='cc',
        caption = 'Risco estimado para ajustes com validação cruzada de 5 lotes') |>
  kable_styling(position="center",
                latex_options="HOLD_position")

```

Table 2: Risco estimado para ajustes com validação cruzada de 5 lotes

| Método              | Risco    |
|---------------------|----------|
| KNN                 | 6.939318 |
| Nadaraya-Watson     | 3.798340 |
| Árvore de Regressão | 4.612563 |
| Floresta Aleatória  | 2.789174 |

Assim como anteriormente, o ajuste com melhor desempenho preditivo foi o feito via Florestas Aleatórias, com risco estimado de aproximadamente 2.789. O segundo método com melhor desempenho preditivo também foi o Nadaraya Watson neste caso, com risco estimado de aproximadamente 3.798. Os ajustes via KNN e Árvore de Regressão apresentaram desempenho pior, com valores próximos ao que foi obtido anteriormente.

## Item 6

Os gráficos PDP (Gráficos de Dependência Parcial) representam de que forma os valores de uma covariável influenciam nas previsões, permitindo que nós identifiquemos de que forma modelos do tipo caixa-preta (como é o caso do KNN e das Florestas Aleatórias).

A curva do PDP para uma covariável  $i$  é dada por:

$$h_i(x) = \frac{1}{n} \sum_{j=1}^n g(x_{j'1}, \dots, x_{j'i-1}, x, x_{j'i+1}, \dots, x_{j'd})$$

Abaixo, seguem as curvas de dependência parcial para todas as covariáveis dos dois métodos solicitados no enunciado.

```
### ITEM 6

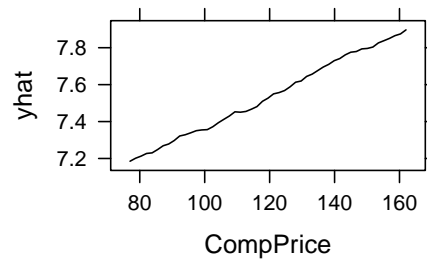
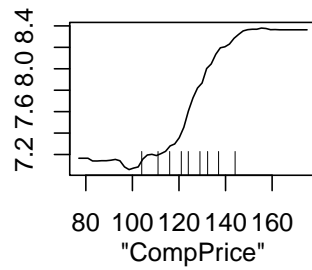
# Graficos de CompPrice e Income

flor1 <- ggplotify::as.ggplot(~partialPlot(ajuste_flor, pred.data=x_tre, x.var='CompPrice'),
                             scale=0.8)
knn1 <- plotPartial(pdp::partial(ajuste_knn, pred.var='CompPrice'))

flor2 <- ggplotify::as.ggplot(~partialPlot(ajuste_flor, pred.data=x_tre, x.var='Income'),
                             scale=0.8)
knn2 <- plotPartial(pdp::partial(ajuste_knn, pred.var='Income'))

gridExtra::grid.arrange(fl1, knn1, fl2, knn2, ncol=2)
```

### Partial Dependence on "CompPrice"



### Partial Dependence on "Income"

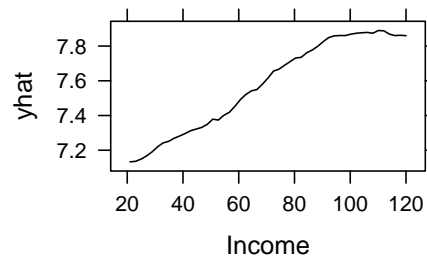
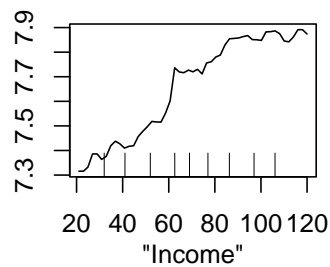


Figure 1: PDP para variáveis CompPrice (Linha 1) e Income (Linha 2) de florestas aleatórias (Coluna 1) e KNN (Coluna 2)

Acima, temos que em todos os casos as covariáveis apresentam relação crescente positiva, ou seja, as previsões crescem à medida que as variáveis 'Income' e 'CompPrice' também crescem. No entanto, para o caso do KNN (na segunda coluna de gráficos), a relação observada apresenta comportamento muito mais próximo do linear do que podemos observar no caso do ajuste via Florestas Aleatórias. No entanto, vale ressaltarmos que isso não significa que o modelo KNN é melhor (algo que pudemos comprovar com os riscos estimados nas questões anteriores).

#### # Gráficos de Advertising e Population

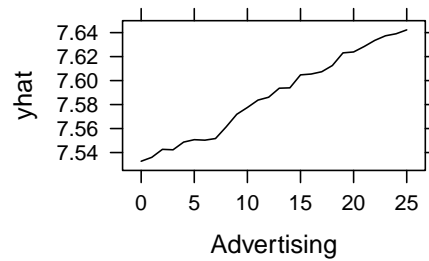
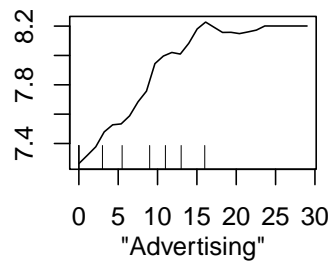
```
flor3 <- ggplotify::as.ggplot(~partialPlot(ajuste_flor, pred.data=x_tre, x.var='Advertising'),
                             scale=0.8)
knn3 <- plotPartial(pdp::partial(ajuste_knn, pred.var='Advertising'))

flor4 <- ggplotify::as.ggplot(~partialPlot(ajuste_flor, pred.data=x_tre, x.var='Population'),
                             scale=0.8)
knn4 <- plotPartial(pdp::partial(ajuste_knn, pred.var='Population'))

gridExtra::grid.arrange(fl3, knn3, fl4, knn4, ncol=2)
```



### Partial Dependence on "Advertising"



### Partial Dependence on "Population"

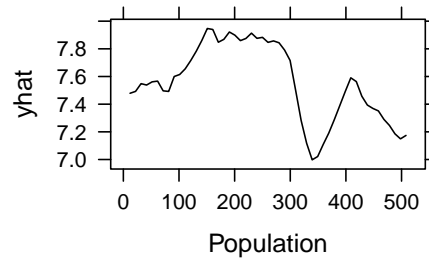
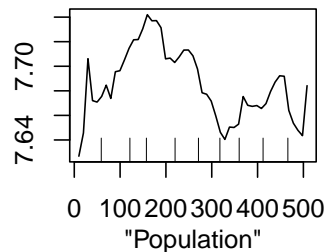


Figure 2: PDP para variáveis Advertising (Linha 1) e Population (Linha 2) de florestas aleatórias (Coluna 1) e KNN (Coluna 2)

Para a variável ‘Advertising’ em florestas aleatórias é possível notar uma estabilidade a partir dos 15 mil em gastos de publicidade. Enquanto para KNN é uma reta quase que crescente. Em ambos os ajustes, temos que há uma relação crescente positiva entre a variável e as predições de vendas de cadeirinhas

Já observando a população para ambos os casos, há uma queda logo após 300 mil e uma subida perto dos 400 mil. Não existe relação crescente ou decrescente óbvia, com os valores variando ao longo do eixo x correspondente aos valores da variável ‘Population’.

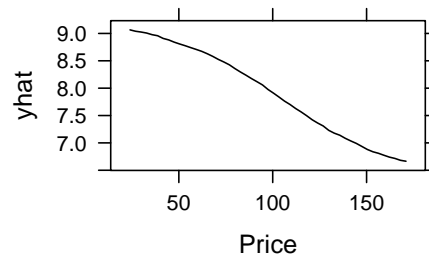
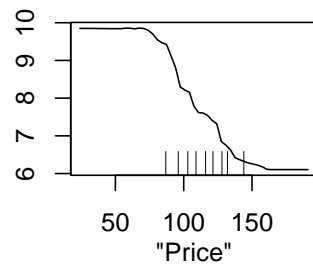
#### # Gráficos de Price e Age

```
flor5 <- ggplotify::as.ggplot(~partialPlot(ajuste_flor, pred.data=x_tre, x.var='Price'),
                             scale=0.8)
knn5 <- plotPartial(pdp::partial(ajuste_knn, pred.var='Price'))

flor6 <- ggplotify::as.ggplot(~partialPlot(ajuste_flor, pred.data=x_tre, x.var='Age'),
                             scale=0.8)
knn6 <- plotPartial(pdp::partial(ajuste_knn, pred.var='Age'))

gridExtra::grid.arrange(flor5, knn5, flor6, knn6, ncol=2)
```

### Partial Dependence on "Price"



### Partial Dependence on "Age"

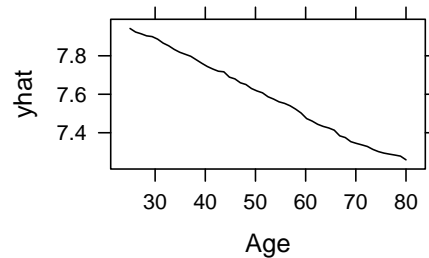
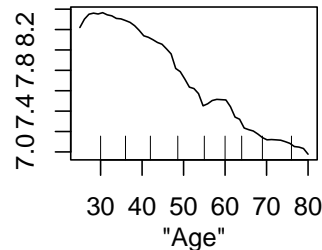


Figure 3: PDP para variáveis Price (Linha 1) e Age (Linha 2) de florestas aleatórias (Coluna 1) e KNN (Coluna 2)

Já as variáveis preço e idade apresentam uma relação inversa, a medida que o preço aumenta as vendas caem, e a medida que a idade cresce as vendas também diminuem. Em relação aos dois métodos quando olhamos para preço o gráfico de floresta aleatória apresenta uma constante nos extremos enquanto que o KNN apresenta uma reta praticamente linear. Agora para a variável idade, apresenta algumas pequenas inversões entre queda e aumento para florestas aleatórias, já o KNN apresenta uma linha linear decrescente.

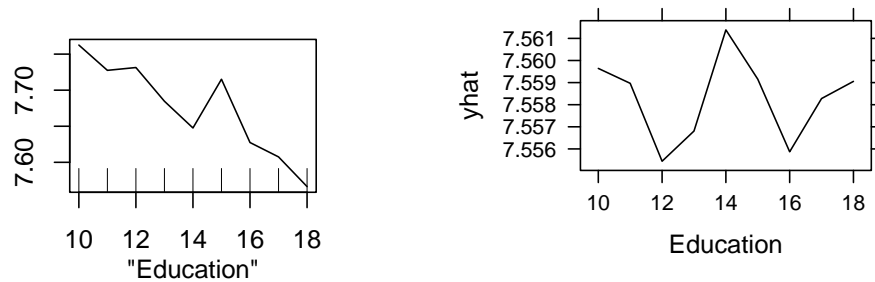
#### # Gráficos de Education e ShelveLocGood

```
flor7 <- ggplotify::as.ggplot(~partialPlot(ajuste_flor, pred.data=x_tre, x.var='Education'),
                             scale=0.8)
knn7 <- plotPartial(pdp::partial(ajuste_knn, pred.var='Education'))

flor8 <- ggplotify::as.ggplot(~partialPlot(ajuste_flor, pred.data=x_tre,
                                           x.var='ShelveLocGood'), scale=0.8)
knn8 <- plotPartial(pdp::partial(ajuste_knn, pred.var='ShelveLocGood'))

gridExtra::grid.arrange(flor7, knn7, flor8, knn8, ncol=2)
```

### Partial Dependence on "Education"



### Partial Dependence on "ShelveLocGood"

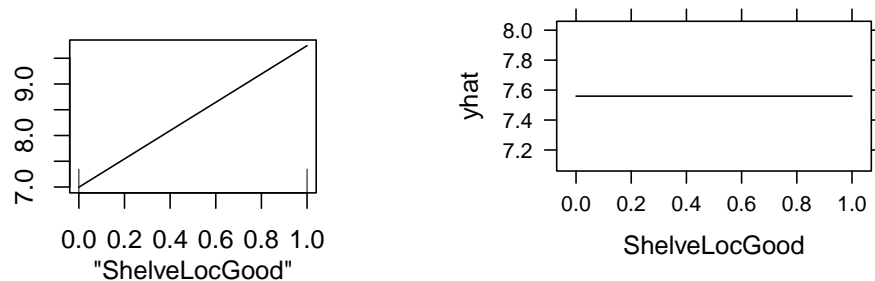


Figure 4: PDP para variáveis Education (Linha 1) e ShelveLocGood (Linha 2) de florestas aleatórias (Coluna 1) e KNN (Coluna 2)

Na figura acima, temos que a variável ‘Education’ apresentou relação decrescente em relação às predições da variável ‘Sales’, ou seja, o ajuste via florestas aleatórias interpreta que as vendas de cadeiras infantis diminuem na medida que os anos de escolaridade aumenta. No caso da variável ‘ShelveLocGood’, temos uma relação inversa: se a localização nas prateleiras for boa, as vendas aumentam.

Já para o ajuste via KNN, temos que a variável ‘Education’ não apresenta relação homogênea, com as vendas de carrinhos aumentando e diminuindo, mas principalmente indicando que escolaridades muito baixas e muito altas apresentam padrão de compras parecidos. No caso da variável ‘ShelveLocGood’, não há nenhuma influência.

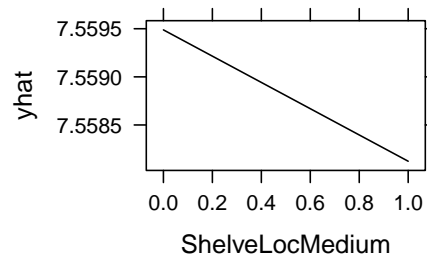
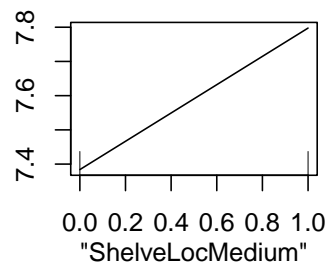
*# Gráficos de ShelveLocMedium e UrbanYes*

```
flor9 <- ggplotify::as.ggplot(~partialPlot(ajuste_flor, pred.data=x_tre,
                                          x.var='ShelveLocMedium'), scale=0.8)
knn9 <- plotPartial(pdp::partial(ajuste_knn, pred.var='ShelveLocMedium'))

flor10 <- ggplotify::as.ggplot(~partialPlot(ajuste_flor, pred.data=x_tre, x.var='UrbanYes'),
                              scale=0.8)
knn10 <- plotPartial(pdp::partial(ajuste_knn, pred.var='UrbanYes'))

gridExtra::grid.arrange(flor9, knn9, flor10, knn10, ncol=2)
```

### Partial Dependence on "ShelveLocMed"



### Partial Dependence on "UrbanYes"

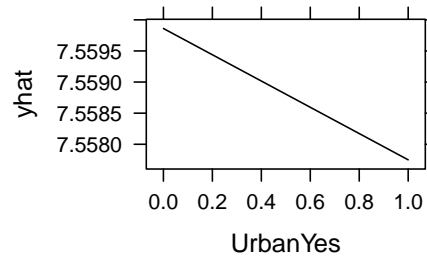


Figure 5: PDP para variáveis ShelveLocMedium (Linha 1) e UrbanYes (Linha 2) de florestas aleatórias (Coluna 1) e KNN (Coluna 2)

Para o ajuste via Florestas Aleatórias, temos que a localização nas prateleiras de qualidade média (pela variável 'ShelveLocMedium') faz crescer a predição do número de vendas, bem como, segundo a variável 'UrbanYes' nos indica que o número de vendas cai se uma cidade for de zona urbana.

Já no caso do ajuste via KNN, temos que a localização nas prateleiras de qualidade média faz cair a predição do número de vendas. Da mesma forma, a predição do número de vendas também cai se uma cidade for de zona urbana, segundo o ajuste via KNN.

#### # Gráficos de USYes

```
flor11 <- ggplotify::as.ggplot(~partialPlot(ajuste_flor, pred.data=x_tre, x.var='USYes'),
                              scale=0.8)
knn11 <- plotPartial(pdp::partial(ajuste_knn, pred.var='USYes'))
gridExtra::grid.arrange(fl11, knn11, ncol=2)
```

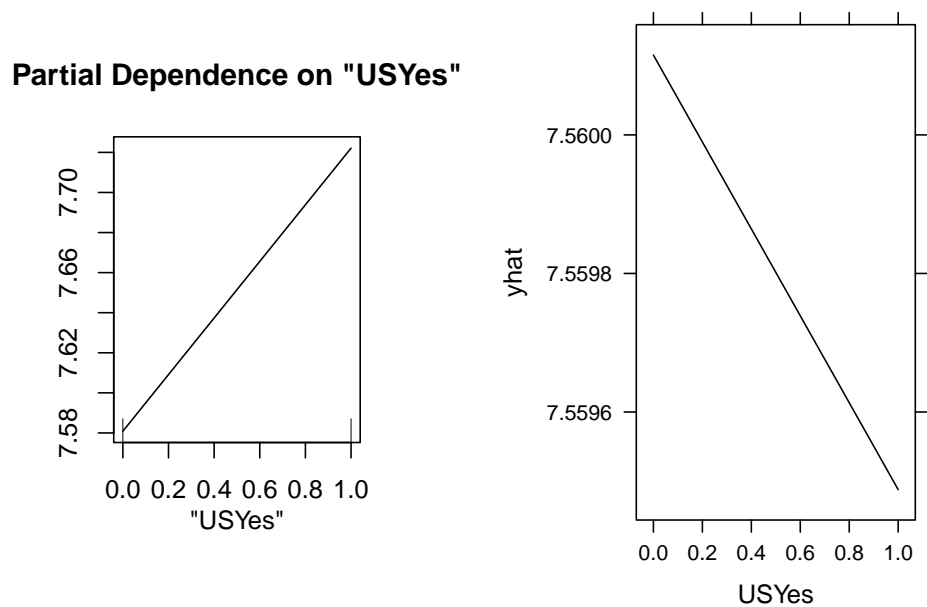


Figure 6: PDP para variável USYes de florestas aleatórias (Coluna 1) e KNN (Coluna 2)

Na variável 'USYes', o comportamento fica oposto para os dois métodos já que na floresta aleatória é crescente e para o KNN é decrescente. Em termos do problema, segundo Florestas Aleatórias, temos que as vendas de cadeirinhas aumentam se uma loja estiver nos EUA, e o contrário no KNN. Isso está relacionado ao fato de estarmos tratando de uma variável categórica e, como o KNN é calculado baseado em distâncias, a tendência é que a predição não seja tão boa. No geral, é interessante confiarmos mais nos comportamentos observados no ajuste via florestas aleatórias dado que este método apresentou menor risco estimado.