

Luego de haber filtrado que columnas tienen valores faltantes (**NAs**), eliminaremos columnas en base a los siguientes criterios:

```
In [17]:  
#install.packages("rpart")  
library(rpart)  
library(rpart.plot)  
library(tree)  
set.seed(1)  
#Leer el archivo, donde Los datos vacíos se rellenan de NA's  
housingData = read.csv("housing_train.csv", header = TRUE, sep=",", na.strings="NA") #
```

TRATAMIENTO DE MISSING

Si existen valores faltantes, decidir si eliminar los registros, llenarlos con valores como la media, la mediana o la moda y justifique su respuesta.

Eliminacion de variables con NAs >40%

Hallamos los valores NA faltantes de cada una de las columnas del dataset

```
In [18]:
```

```
#Porcentaje de valores numericos perdidos en La Data  
DataNumeric<-Filter(is.numeric, housingData)  
apply(DataNumeric[,names(DataNumeric[is.na(colMeans(DataNumeric))])], 2, function(col) sum(is.na(col))/length(col)*100) #Porcentaje de valores perdidos por columna de La Data  
  
#Porcentaje de valores categoricos perdidos en La Data  
DataCategorical<-Filter(is.factor, housingData)  
DataCategorical_NAs<-DataCategorical[,colnames(DataCategorical)[colSums(is.na(DataCategorical)) > 0]]  
apply(DataCategorical_NAs, 2, function(col) sum(is.na(col))/length(col)*100) #Porcentaje de valores perdidos por columna de La Data  
  
LotFrontage  
17.7397266273973  
MasVnrArea  
0.547945205479452  
GarageYrBlt  
5.54794520547945
```

Alley
93.7671232876712
MasVnrType
0.547945205479452
BsmtQual
2.53424657534247
BsmtCond
2.53424657534247
BsmtExposure
2.6027397260274
BsmtFinType1
2.53424657534247
BsmtFinType2
2.6027397260274
Electrical
0.0684931506849315
FireplaceQu
47.2602739726027
GarageType
5.54794520547945
GarageFinish
5.54794520547945
GarageQual
5.54794520547945
GarageCond
5.54794520547945
PoolQC
99.5205479452055
Fence
80.7534246575342
MiscFeature
96.3013696630137

Por cuestiones prácticas se eliminan los siguientes variables que superan el 15 % de datos faltantes.

- Alley: .93.76% valores perdidos
- PoolQC :.99.52% valores perdidos
- Fence :.80.75% valores perdidos
- MiscFeature :.96.30% valores perdidos
- FireplaceQu: .47.26% valores perdido
- LotFrontage: .17.73% valores perdidos

```
In [19]:
```

```
#Eliminamos Las columnas que no tiene relevancia en el dataset dado el exceso de datos faltantes.  
borrar <- c("Alley", "PoolQC", "Fence", "MiscFeature", "FireplaceQu", "LotFrontage")  
housingData <- housingData[, !names(housingData) %in% borrar]  
cat("Resultado de Porcentaje valores faltantes:" , sum(is.na(housingData))/prod(dim(housingData))*100, "%")
```

Resultado de Porcentaje valores faltantes: 0.5561644 %

Variables Numericas con NAs

Listamos las variables numéricas que contienen NAs

```
In [20]:
```

```
#Porcentaje de valores numericos perdidos en La Data  
DataNumeric<-Filter(is.numeric, housingData)  
DataNumeric_NAs<-DataNumeric[,names(DataNumeric[is.na(colMeans(DataNumeric))])] > 0  
apply(DataNumeric_NAs, 2, function(col) sum(is.na(col))/length(col)*100) #Porcentaje de valores perdidos por columna de La Data  
  
LotFrontage  
0.547945205479452  
GarageYrBlt  
5.54794520547945
```

Graficamos la distribucion y la correlacion que existen con SalePrice

In [21]:

```
#Graficamos los histogramas de las variables numericas para evaluar su distribucion
#Hallamos la correlacion que existe de las variables independientes con respecto a la variable dependiente "SalePrice"
par(mfrow=c(1,2))

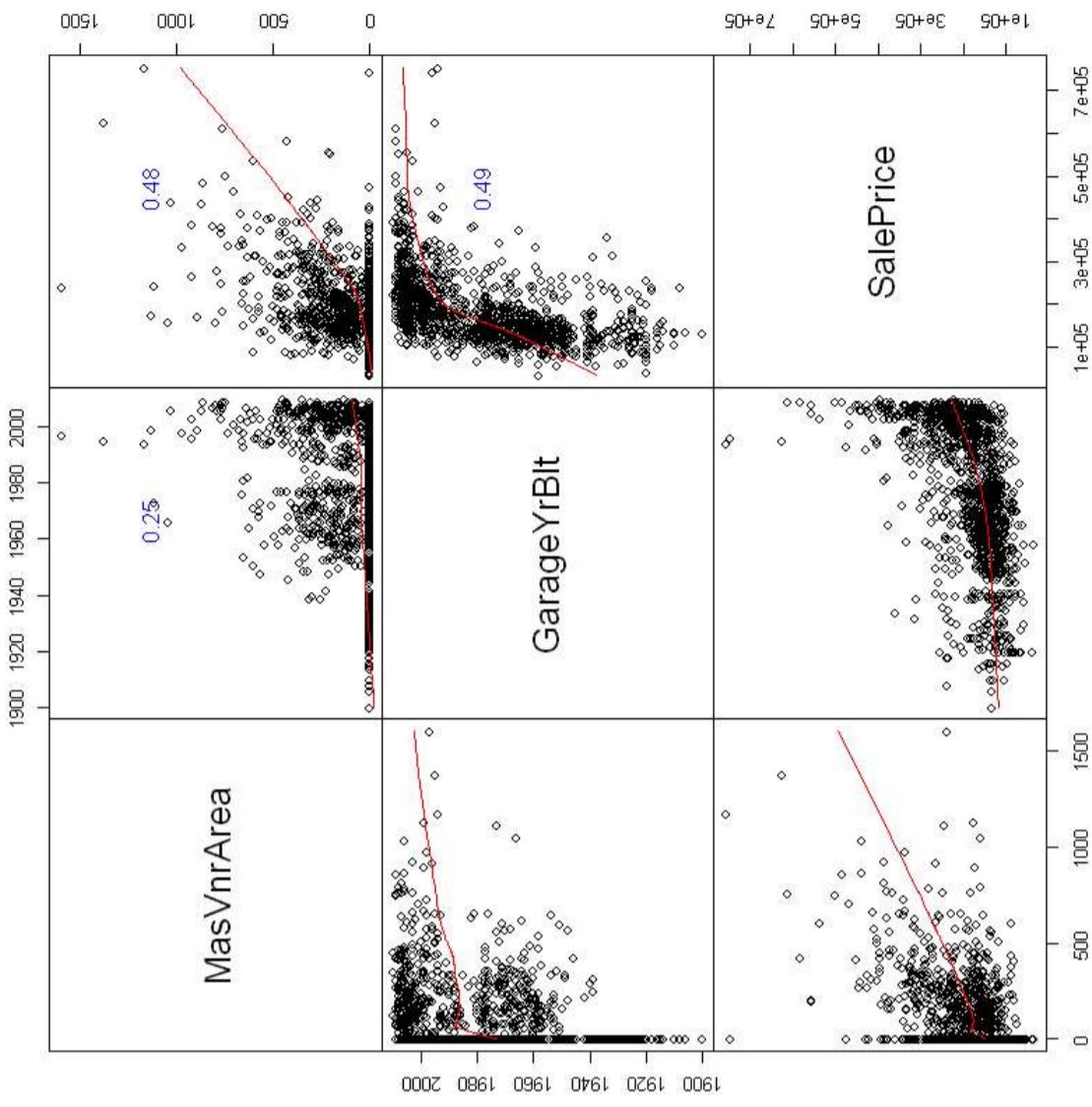
v<-c(which(colnames(DataNumeric)=="MasVnrArea"),which(colnames(DataNumeric)=="GarageYrBlt"),which(colnames(DataNumeric)=="SalePrice"))

a<-DataNumeric[,v]

pairs(a, gap=0, lower.panel = panel.smooth, upper.panel = function(x,y){
  panel.smooth(x,y)
  par usr=c(0,1,0,1)
  correlacion <- cor(x,y, use="complete.obs")
  text(.6,.7, col="blue", cex=1.2, round(correlacion,digits=2))
})

hist(x = DataNumeric$MasVnrArea, main = "Histograma de MasVnrArea", xlab = "MasVnrArea", ylab = "Frecuencia", 3,border="gray",col =
"blue",breaks=c(70))
#plot(DataNumeric$MasVnrArea,DataNumeric$SalePrice , type = 'h', col= 'red', xlab = 'MasVnrArea' , yLab = 'SalePrice')

hist(x = DataNumeric$GarageYrBlt, main = "Histograma de GarageYrBlt", xlab = "GarageYrBlt", ylab = "Frecuencia", 3,border="gray",co
l = "blue",breaks=c(70))
#plot(DataNumeric$GarageYrBlt,DataNumeric$SalePrice , type = 'h' , col= 'red' , xlab = 'GarageYrBlt' , yLab = 'SalePrice')
```



Histograma de MasVnrArea

Reemplazo por la media y eliminacion de variable asimetrica

Segun la distribucion observado en los histogramas de variables numericas , se analiza lo siguiente:

- GarageYrBlt: Cuenta con NAs y cuenta con una distribucion asimetrica por lo tanto, ademas existe una fuerte correlacion con SalePrice, por lo que las NAs sera reemplazados por la Mediana
- MasVnrArea: Cuenta con NAs, cuenta con una distribucion asimetrica, asimismo no esta fuertemente relacionado con SalePrice, por lo que lo descartamos para el modelado.

In [22]:

```
#Reemplazamos La media y mediana para la variable GarageYrBlt
DataNumeric$GarageYrBlt <- ifelse(is.na(DataNumeric$GarageYrBlt),
median(DataNumeric$GarageYrBlt, na.rm = TRUE),
DataNumeric$GarageYrBlt)
```

In [23]:

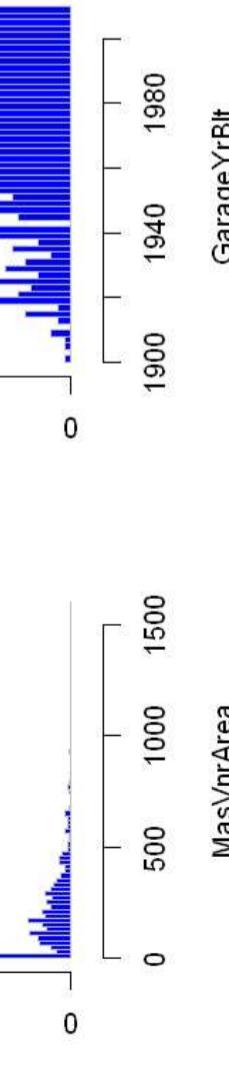
```
#Eliminamos la columna MasVnrArea
DataNumeric <- DataNumeric[, -which(colnames(DataNumeric) == "MasVnrArea")]
```

Variables Categoricas con NAs

Listamos las variables categoricas que contienen NAs

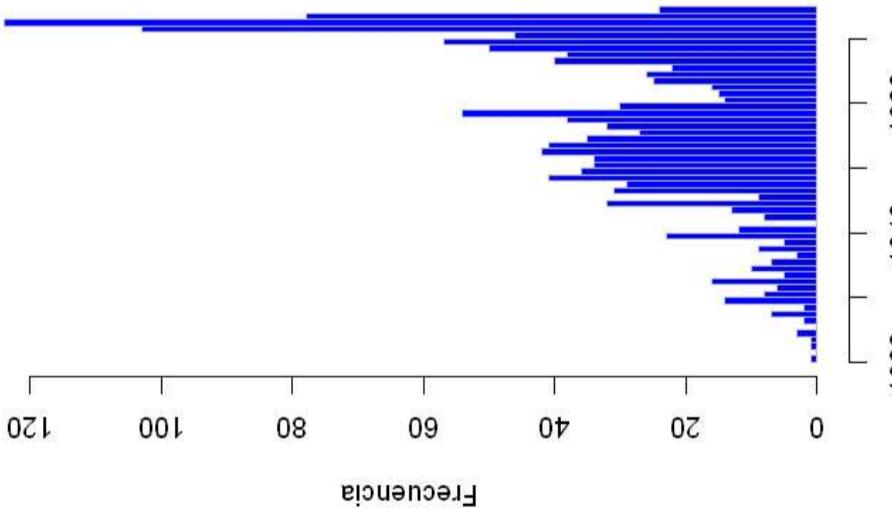
In [24]:

```
#Porcentaje de valores categoricos perdidos en La Data
DataCategorical<-Filter(is.factor, housingData)
DataCategorical_NAs<-DataCategorical[, colnames(DataCategorical)[colSums(is.na(DataCategorical)) > 0]]
apply(DataCategorical_NAs, 2, function(col) sum(is.na(col))/length(col)*100) #Porcentaje de valores perdidos por columna de La Data
```



MasVnrArea

GarageYrBlt



MasVnrArea

GarageYrBlt

Omission de variables categoricas y Reemplazo por la Moda

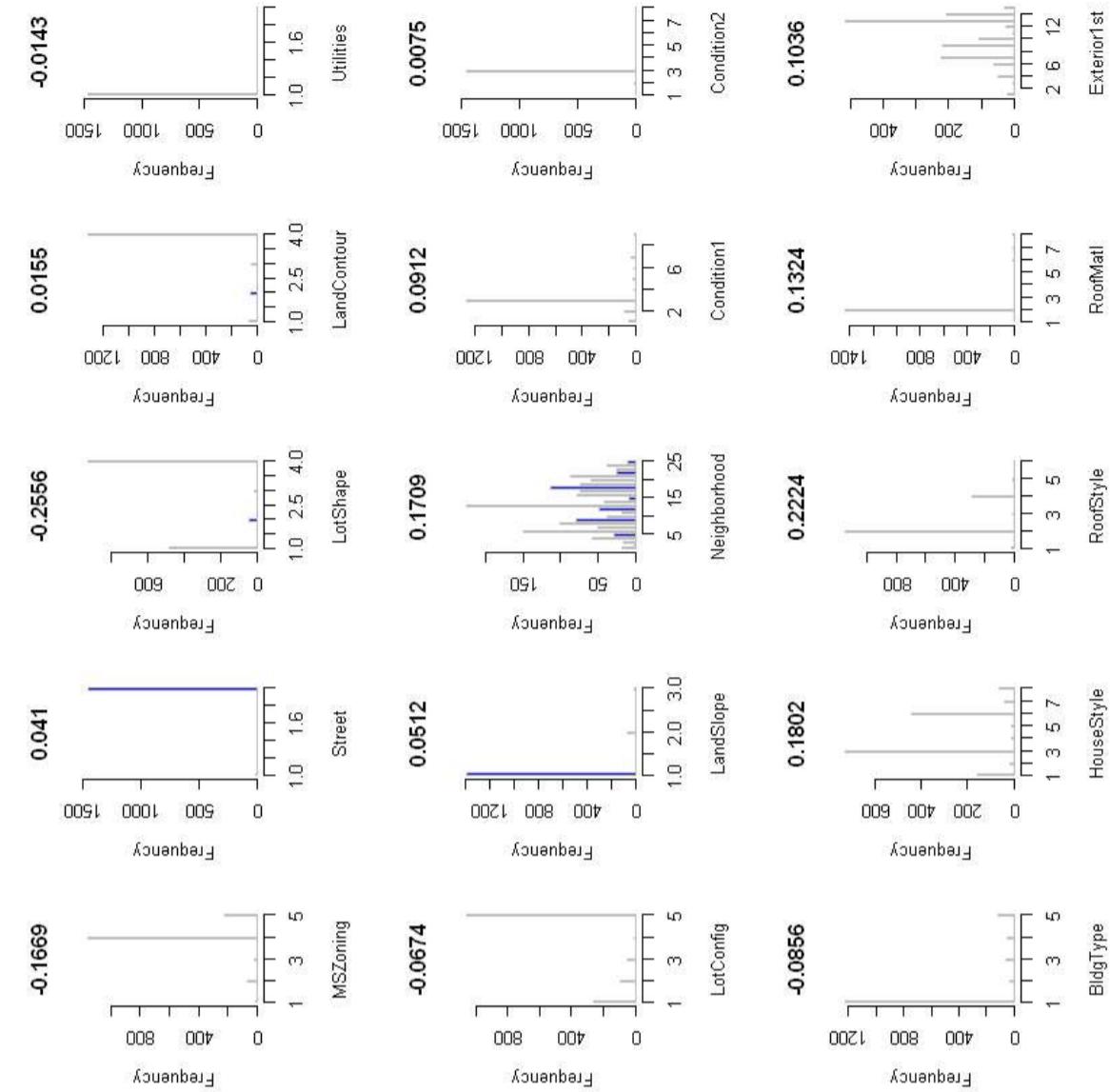
In [25]:

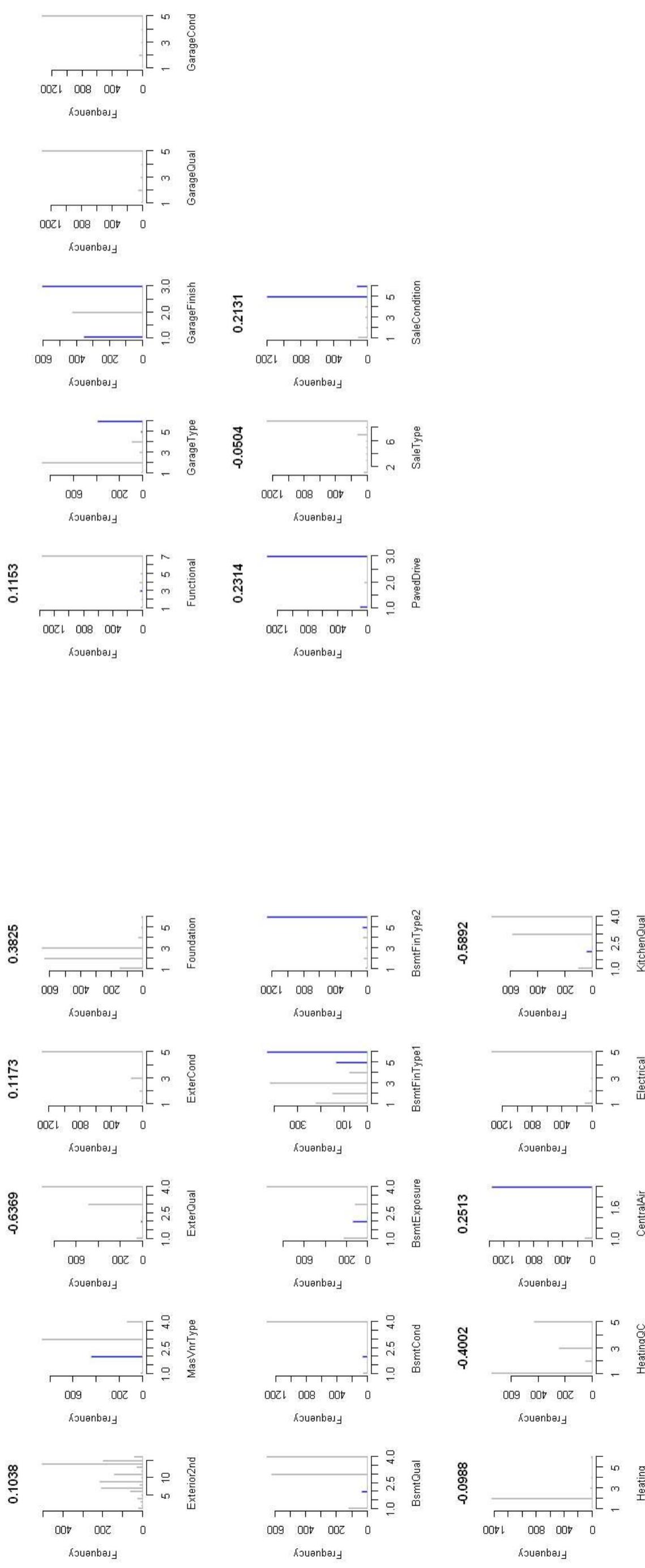
```
#Convertimos las variables categoricas a numericas
DataCategorical[sapply(DataCategorical[is.factor], is.factor)] <- data.matrix(DataCategorical[sapply(DataCategorical, is.factor)])
```

Dado que la cantidad de NAs para las variables categoricas ronda un porcentaje pequeño de 3%, los ignoraremos ya que no son significantes para el gran tamaño del dataset, no obstante conservaremos las variables

In [26]:

```
par(mfrow=c(3,5))
for (i in 1:length(DataCategorical)){
  corx3<-cor(x=DataCategorical[,i],y=DataNumeric$SalePrice)
  a3<-round(corx3, digits=4)
  hist(DataCategorical[,i],xlab=colnames(DataCategorical[i]),main=a3,3,border="gray",col = "blue",breaks=c(70))
}
```





De los graficos que se pueden visualizar, se observa que dichos datos categoricos no mantienen una fuerte relacion con el SalePrice, asimismo poseen bastantes valores atipicos cuya dispersion de las mismas se visualizan en la grafica. Por lo tanto se procede a eliminar del dataset las siguientes variables:

MSZoning Street LotShape LandContour Utilities LotConfig LandSlope Condition1 Condition2 RoofMatl MasVnrType BsmtQual BsmtCond BsmtExposure BsmtFinType1 BsmtFinType2 Electrical GarageType GarageQual GarageCond

In [27]:

```
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="MSZoning")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="Street")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="LotShape")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="LotConfig")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="Utilities")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="LotConfig")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="LandSlope")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="Condition1")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="Condition2")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="RoofMatl")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="MasVnrType")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="BsmtQual")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="BsmtCond")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="BsmtExposure")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="BsmtFinType1")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="BsmtFinType2")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="Electrical")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="GarageType")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="GarageQual")]
DataCategorical <- DataCategorical[,-which(colnames(DataCategorical)=="GarageCond")]
```

In [29]:

```
#Porcentaje de valores numericos perdidos en La Data
DataCategorical<-Filter(is.numeric, DataCategorical)
DataCategorical_NAs<-DataCategorical[,names(DataCategorical[is.na(colMeans(DataCategorical))])]> 0

apply(DataCategorical, 2, function(col) sum(is.na(col))/length(col)*100) #Porcentaje de valores perdidos por columna de La Data
#Filtramos Los datos numericos para halar el minimo,maximo, media y mediana
summary(DataNumeric)
```

	Id	MSSubClass	LotArea	OverallQual
Neighborhood	Min. : 1.0 1st Qu.: 365.8 Median : 730.5 Mean : 730.5 3rd Qu.:1095.2 Max. :1460.0	Min. : 20.0 1st Qu.: 20.0 Median : 50.0 Mean : 56.9 3rd Qu.: 70.0 Max. :190.0	Min. : 1300 1st Qu.: 7554 Median : 9478 Mean : 10517 3rd Qu.: 11602 Max. :215245	Min. : 1.000 1st Qu.: 5.000 Median : 6.000 Mean : 6.099 3rd Qu.: 7.000 Max. :10.000
BldgType	Min. : 0 0	Min. :1872 1st Qu.:1954 Median :1973 Mean :5.575 3rd Qu.:6.000 Max. :9.000	Min. :1950 1st Qu.:1967 Median :1994 Mean :1985 3rd Qu.:2000 Max. :2010	Min. : 0.0 1st Qu.: 0.0 Median : 0.0 Mean : 443.6 3rd Qu.: 712.2 Max. :5644.0
HouseStyle	0	Min. : 0.00 1st Qu.: 0.00 Median : 0.00 Mean : 46.55 3rd Qu.: 0.00 Max. :1474.00	Min. : 0.00 1st Qu.: 223.0 Median : 477.5 Mean : 567.2 3rd Qu.: 888.0 Max. :2336.0	Min. : 0.0 1st Qu.: 795.8 Median : 991.5 Mean :1057.4 3rd Qu.:1298.2 Max. :6110.0
RoofStyle	0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 347 3rd Qu.: 728 Max. :2865	Min. : 0 1st Qu.: 0 Median : 0 Mean : 1.0000 3rd Qu.:1.0000 Max. :572.000	Min. : 0 1st Qu.: 1130 Median : 1464 Mean :1515 3rd Qu.:1777 Max. :5642
Exterior1st	0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0 3rd Qu.: 0 Max. :0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:2.0000 Max. :2.0000	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:3.0000 Max. :2.0000
Exterior2nd	0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0 3rd Qu.: 0 Max. :0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:2.0000 Max. :2.0000	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:1.0000 Max. :3.0000
ExteriorCond	0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0 3rd Qu.: 0 Max. :0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:2.0000 Max. :2.0000	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:1.0000 Max. :3.0000
Foundation	0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0 3rd Qu.: 0 Max. :0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:2.0000 Max. :2.0000	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:1.0000 Max. :3.0000
CentralAir	0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0 3rd Qu.: 0 Max. :0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:2.0000 Max. :2.0000	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:1.0000 Max. :3.0000
Heating	0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0 3rd Qu.: 0 Max. :0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:2.0000 Max. :2.0000	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:1.0000 Max. :3.0000
HeatingQC	0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0 3rd Qu.: 0 Max. :0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:2.0000 Max. :2.0000	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:1.0000 Max. :3.0000
KitchenQual	0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0 3rd Qu.: 0 Max. :0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:2.0000 Max. :2.0000	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:1.0000 Max. :3.0000
Functional	0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0 3rd Qu.: 0 Max. :0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:2.0000 Max. :2.0000	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0.0000 3rd Qu.:1.0000 Max. :3.0000
GarageFinish	5.54794520547945	Min. : 0 1st Qu.: 0 Median : 0 Mean : 1.047 3rd Qu.:1.000 Max. :3.000	Min. : 0 1st Qu.: 0 Median : 0 Mean : 6.518 3rd Qu.:7.000 Max. :14.000	Min. : 0 1st Qu.: 0 Median : 0 Mean : 6.3829 3rd Qu.:1.0000 Max. :2.0000
PavedDrive	0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0 3rd Qu.: 0 Max. :0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 6.518 3rd Qu.:7.000 Max. :14.000	Min. : 0 1st Qu.: 0 Median : 0 Mean : 6.3829 3rd Qu.:1.0000 Max. :2.0000
SaleType	0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0 3rd Qu.: 0 Max. :0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 6.518 3rd Qu.:7.000 Max. :14.000	Min. : 0 1st Qu.: 0 Median : 0 Mean : 6.3829 3rd Qu.:1.0000 Max. :2.0000
SaleCondition	0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 0 3rd Qu.: 0 Max. :0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 6.518 3rd Qu.:7.000 Max. :14.000	Min. : 0 1st Qu.: 0 Median : 0 Mean : 6.3829 3rd Qu.:1.0000 Max. :2.0000
EnclosedPorch	0	Min. : 0 1st Qu.: 0 Median : 0 Mean : 21.95 3rd Qu.: 0.00 Max. :552.00	Min. : 0 1st Qu.: 0 Median : 0 Mean : 3.41 3rd Qu.: 0.00 Max. :508.00	Min. : 0 1st Qu.: 0 Median : 0 Mean : 15.06 3rd Qu.: 0.00 Max. :480.00
MiscVal	Min. : 0 1st Qu.: 0 Median : 0 Mean : 43.49 3rd Qu.: 0.00 Max. :15500.00	Min. : 0 1st Qu.: 0 Median : 0 Mean : 43.49 3rd Qu.: 0.00 Max. :12.0000	Min. : 0 1st Qu.: 5.000 Median : 6.000 Mean : 6.3322 3rd Qu.: 8.000 Max. :2010	Min. : 0 1st Qu.: 5.000 Median : 6.000 Mean : 6.3322 3rd Qu.: 8.000 Max. :2010

Reemplazamos la variable GarageFinish por la moda, ya que mantiene fuertemente una relacion con SalePrice

In [39]:

```
#install.packages("modeest")
library(modeest)
DataCategorical$GarageFinish<-mlv(DataCategorical$GarageFinish, method = "parzen", kernel = "gaussian")
```

ANALISIS DESCRIPTIVO Y TRATAMIENTO DE VARIABLES

De las variables numericas(limpias) hallamos el valor minimo, el maximo, la mediana y la media

Tratamiento Variables Numericas

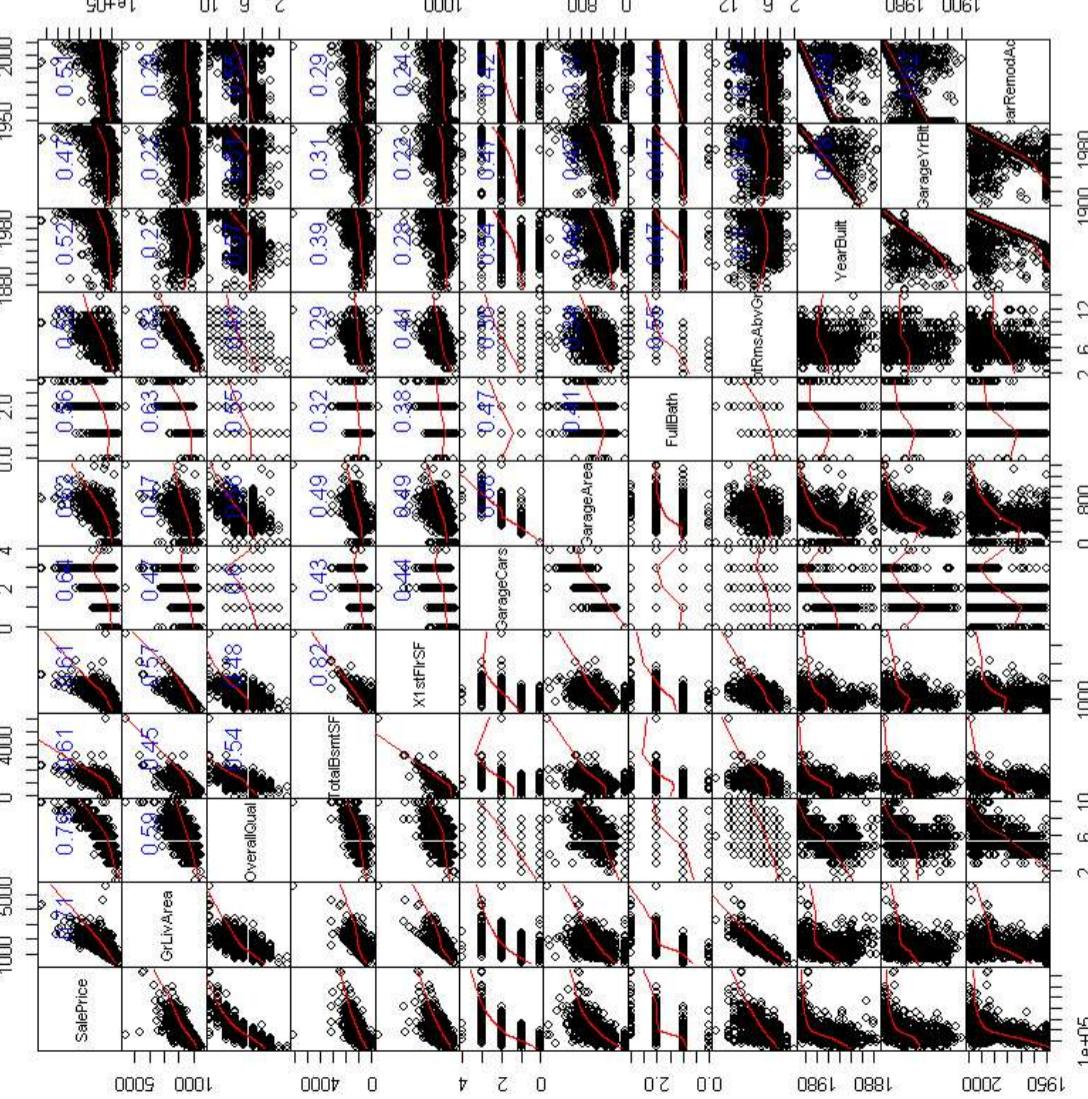
Matriz de Correlacion - Variables Numericas

In [43]:

```
#Cargamos Libreria
#install.packages("caret")
library(caret)
#Hallamos La matriz de correlacion
correlationMatrix <- cor(Data_Numeric[,])
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.10)
Data_Cor<-Filter(is.numeric, Data_Numeric[highlyCorrelated])

```

Pairs(Data_Cor[1:12], gap=0, lower.panel = panel.smooth, upper.panel = function(x,y){
 panel.smooth(x,y)
 par(usrf=c(0,1,0,1))
 correlacion <- cor(x,y, use="complete.obs")
 text(.6,.7, col="blue", cex=1.2, round(correlacion,digits=2))
})



In [44]:

```
#Añadimos La columna de SalePrice para verificar su correlacion con las variables categoricas
pairs(Data_Cor[c(1,13,14,15,16,17,18,19,20,21,22)], gap=0, lower.panel = panel.smooth, upper.panel = function(x,y){  

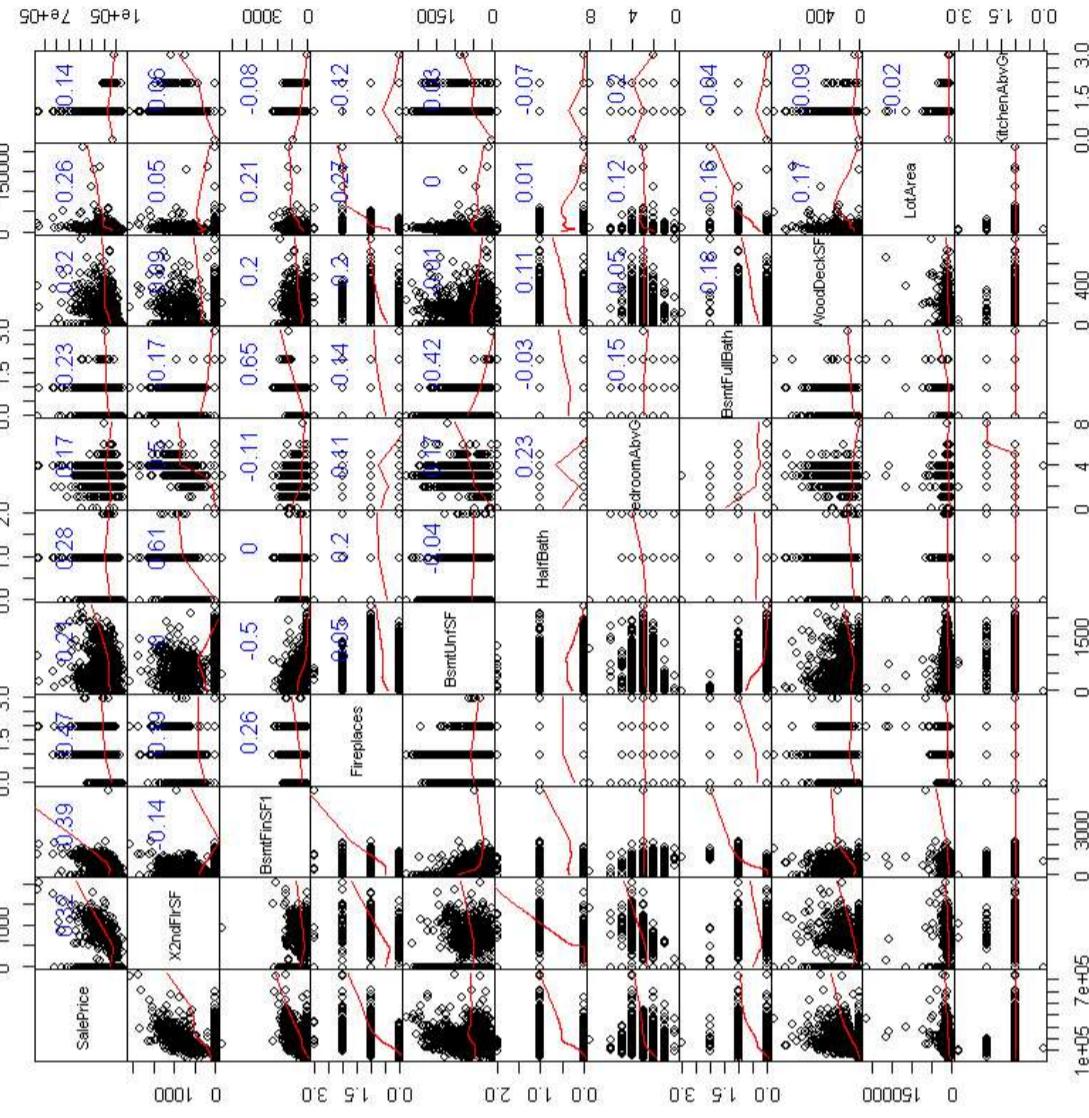
  panel.smooth(x,y)  

  par(usrf=c(0,1,0,1))  

  correlacion <- cor(x,y, use="complete.obs")  

  text(.6,.7, col="blue", cex=1.2, round(correlacion,digits=2))  

})
```



Eliminacion de variables por muy baja correlacion con SalePrice

Según la verificación realizados en los histogramas y la correlación entre las variables independientes con respecto a la variable dependiente(SalePrice), se procede a eliminar las variables numéricas que cuentan con una correlación inferior a 0.1 y superior a -0.1 dado que cuentan con una muy baja correlación y no aportan a la predicción de Modelo:

Para mas detalle de la matriz de correlacion obtenida por el mapa de calor(heatmap), hallamos las variables que posee mayor correlacion con respecto a SalePrice, el cual posiblemente exista alta dependencia dentro del conjunto de variables, por lo que debe ser evaluado.

- Id: Cuenta con una correlacion muy baja de -0.02191672 y no posee una distribucion normal
 - MSSubClass: Cuenta con una correlacion muy baja de -0.08428414
 - OverallCond: Cuenta con una correlacion muy baja de -0.07785589
 - BsmthFinSF2 : Cuenta con una correlacion muy baja de -0.01137812 y no posee una distribucion LowQualFinSF Cuenta con una correlacion muy baja de -0.02560613 y no posee una distribucion
 - BsmthHalfBath Cuenta con una correlacion muy baja de -0.01684415 y no posee una distribucion
 - X3SsnPorch: Cuenta con una correlacion muy baja de 0.046 y no posee una distribucion normal
 - PoolArea: Cuenta con una correlacion muy baja de 0.0924 y no posee una distribucion normal
 - MiscVal: Cuenta con una correlacion muy baja de -0.0212 y no posee una distribucion normal
 - YrSold: Cuenta con una correlacion muy baja de -0.0289 y no posee una distribucion normal

In [49]:

De las correlaciones y covarianzas podemos extraer que las siguientes variables estan altamente correlacionadas por lo que se eliminara uno de ellos:

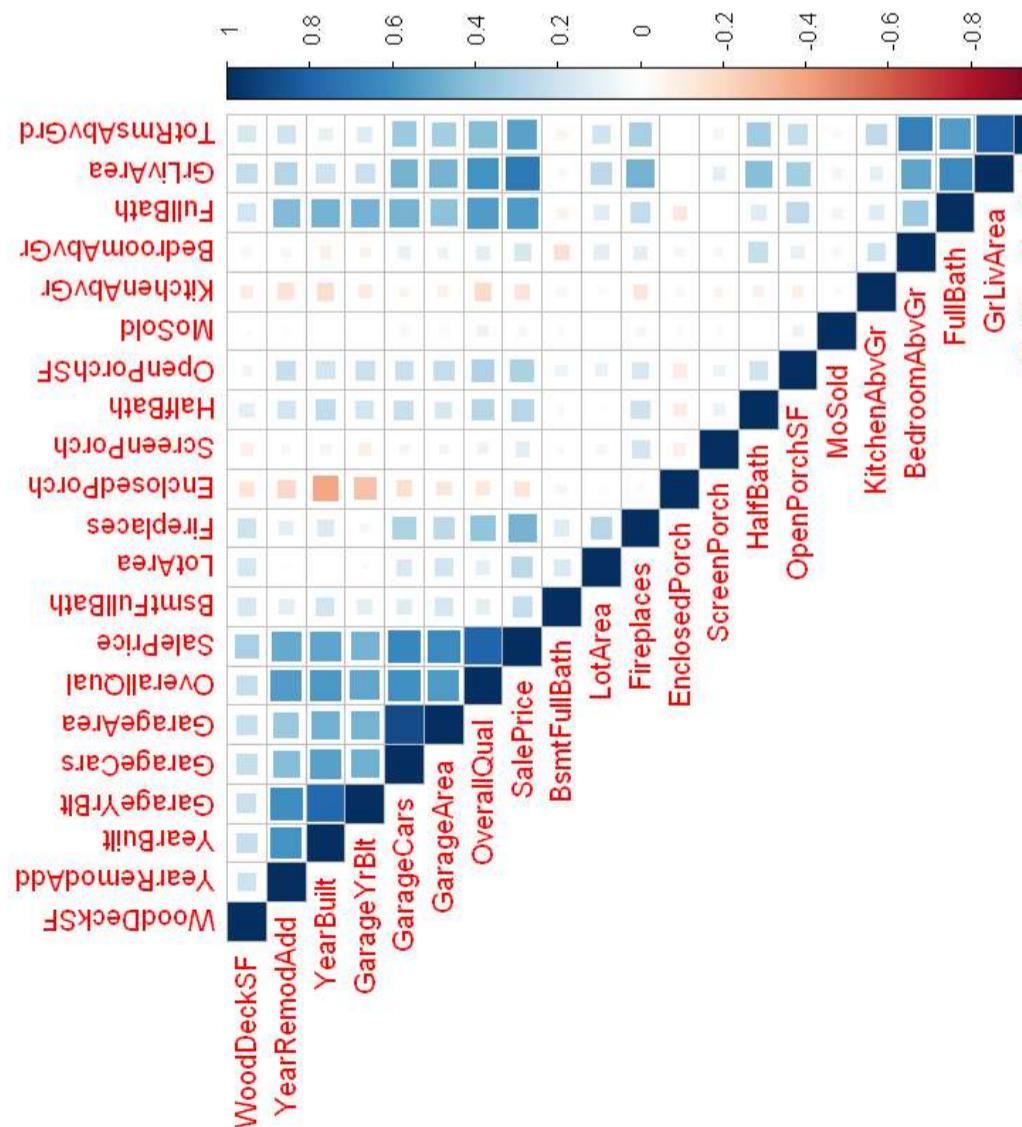
- GarageArea y X1stFlrSF: Se elimina X1stFlrSF dado que el area de garage posee mayor correlacion con SalePrice.
- GarageArea y X2ndFlrSF: Se elimina X2ndFlrSF dado que el area de garage posee mayor correlacion con SalePrice.
- TotalBsmtSF y BsmtFinSF1: Se elimina BsmtFinSF1 dado que la variable TotalBsmtSF posee mayor correlacion con SalePrice.
- BsmtUnfSF y BsmtFinSF1: Se elimina BsmtFinSF1 ya que no aporta mucha correlacion con SalePrice
- TotalBsmtSF y GarageArea: Se elimina TotalBsmtSF ya que se correlaciona menos a SalePrice con respecto a GarageArea
- GarageArea & MasVnrArea: Se elimina MasVnrArea ya que se correlaciona menos a SalePrice con respecto a GarageArea

Respecto a las siguiente variables, se mantendran ya que no cuentan con alta correlacion ni covarianza, asimismo sus datos mantienen una dispersión no lineal

- OverallQual y FullBath

```
In [51]:  
borrar <- c("X1stFlrSF", "X2ndFlrSF", "BsmtFinSF1", "BsmtFinSF2", "TotalBsmtSF", "MasVnrArea")  
DataNumeric <- DataNumeric[, !names(DataNumeric) %in% borrar]
```

```
In [52]:  
library(corrplot)  
corrplot(cor(DataNumeric), type="upper", order="hcclust", method="square")
```



• De las variables categóricas (limpias), listamos las diferentes categorías y hallamos la frecuencia de cada una de ellas.

In [54]:

```
#Filtraremos Los datos categoricos para su evaluacion  
str(DataCategorical)
```

```
'data.frame': 1460 obs. of 18 variables:  
 $ Neighborhood : int 6 25 6 7 14 12 21 17 18 4 ...  
 $ BldgType : int 1 1 1 1 1 1 1 1 1 2 ...  
 $ HouseStyle : int 6 3 6 6 1 3 6 1 2 ...  
 $ RoofStyle : int 2 2 2 2 2 2 2 2 2 ...  
 $ Exterior1st : int 13 9 13 14 13 13 13 13 9 ...  
 $ Exterior2nd : int 14 9 14 16 14 14 14 7 16 9 ...  
 $ ExterQual : int 3 4 3 4 3 4 3 4 4 4 ...  
 $ ExterCond : int 5 5 5 5 5 5 5 5 5 5 ...  
 $ Foundation : int 3 2 3 1 3 6 3 2 1 1 ...  
 $ Heating : int 2 2 2 2 2 2 2 2 2 2 ...  
 $ HeatingQC : int 1 1 1 3 1 1 1 3 1 1 ...  
 $ CentralAir : int 2 2 2 2 2 2 2 2 2 2 ...  
 $ KitchenQual : int 3 4 3 3 4 3 4 4 4 4 ...  
 $ Functional : int 7 7 7 7 7 7 7 7 3 7 ...  
 $ GarageFinish : int 3 3 3 3 3 3 3 3 3 3 ...  
 $ PavedDrive : int 3 3 3 3 3 3 3 3 3 3 ...  
 $ SaleType : int 9 9 9 9 9 9 9 9 9 9 ...  
 $ SaleCondition: int 5 5 1 5 5 5 1 5 1 5 ...
```

In [55]:

```
#Convertimos Las variables categoricas a numericas  
DataCategorical[sapply(DataCategorical, is.factor)] <- data.matrix(DataCategorical[sapply(DataCategorical, is.factor)])
```

Nuevo Dataset a ser modelado (Filtrado)

In [56]:

```
housingdata_set <- cbind(DataNumeric,DataCategorical) #Unimos Los datos numericos y categoricos para La obtencion del dataset que  
sera incluido en el modeloamiento  
str(housingdata_Set)
```

```
'data.frame': 1460 obs. of 39 variables:  
 $ LotArea : int 8450 9600 11250 9550 14260 14115 10084 10382 6120 7420 ...  
 $ OverallQual : int 7 6 7 7 8 5 8 7 7 5 ...  
 $ YearBuilt : int 2003 1976 2001 1915 2000 1993 2084 1973 1931 1939 ...  
 $ YearRemodAdd : int 2003 1976 2002 1978 2000 1995 2085 1973 1950 1950 ...  
 $ GrLivArea : int 1710 1262 1786 1717 2198 1352 1694 2090 1774 1077 ...  
 $ BsmtFullBath : int 1 0 1 1 1 1 1 1 0 1 ...  
 $ FullBath : int 2 2 1 2 1 2 2 2 1 ...  
 $ HalfBath : int 1 0 1 0 1 0 1 0 0 0 ...  
 $ BedromAbvGr : int 3 3 3 3 4 1 3 3 2 2 ...  
 $ KitchenAbvGr : int 1 1 1 1 1 1 1 1 2 2 ...  
 $ TotRmsAbvGrd : int 8 6 6 7 9 5 7 8 5 5 ...  
 $ Fireplaces : int 0 1 1 1 0 1 2 2 2 2 ...  
 $ GarageYrBlt : int 2003 1976 2001 1998 2000 1993 2084 1973 1931 1939 ...  
 $ GarageCars : int 2 2 2 3 3 2 2 2 1 ...  
 $ GarageArea : int 548 460 608 642 836 480 636 484 468 205 ...  
 $ WoodDeckSF : int 0 298 0 192 40 255 235 90 0 ...  
 $ OpenPorchSF : int 61 42 35 84 30 57 284 0 4 ...  
 $ EnclosedPorch: int 0 0 272 0 0 0 228 205 0 ...  
 $ ScreenPorch : int 0 0 0 0 0 0 0 0 0 0 ...  
 $ MoSold : int 2 5 9 2 12 10 8 11 4 1 ...  
 $ Saleprice : int 20500 181500 223500 140000 250000 143000 307000 200000 129900 118000 ...  
 $ Neighborhood : int 6 25 6 7 14 12 21 17 18 4 ...  
 $ BldgType : int 1 1 1 1 1 1 1 1 1 2 ...  
 $ HouseStyle : int 6 3 6 6 1 3 6 1 2 ...  
 $ RoofStyle : int 2 2 2 2 2 2 2 2 2 ...  
 $ Exterior1st : int 13 9 13 14 13 13 13 7 4 9 ...  
 $ Exterior2nd : int 14 9 14 16 14 14 14 7 16 9 ...  
 $ ExterQual : int 3 4 3 4 3 4 4 4 4 ...  
 $ ExterCond : int 5 5 5 5 5 5 5 5 5 ...  
 $ Foundation : int 3 2 3 1 3 6 3 2 1 1 ...  
 $ Heating : int 2 2 2 2 2 2 2 2 2 2 ...  
 $ HeatingQC : int 1 1 1 3 1 1 1 3 1 1 ...  
 $ CentralAir : int 2 2 2 2 2 2 2 2 2 2 ...  
 $ KitchenQual : int 3 4 3 3 4 3 4 4 4 4 ...  
 $ Functional : int 7 7 7 7 7 7 7 7 3 7 ...  
 $ GarageFinish : int 3 3 3 3 3 3 3 3 3 3 ...  
 $ PavedDrive : int 3 3 3 3 3 3 3 3 3 3 ...  
 $ SaleType : int 9 9 9 9 9 9 9 9 9 9 ...  
 $ SaleCondition: int 5 5 1 5 5 5 1 5 ...
```

```
In [53]:  
cat("Porcentaje valores numericos faltantes:", sum(is.na(DataNumeric))/prod(dim(DataNumeric))*100, "%")  
Porcentaje valores numericos faltantes: 0 %
```

Tratamiento de variables categóricas

CONJUNTO DE MODELIZACION Y VALIDACION

REGRESION CON ARBOL DE DECISION

Division de conjuntos de Training y Testing

In [59]:

```
#Creacion del arbol de decision - REGRESION
TreeRegression.Sample = sample.split(housingdata_set$SalePrice, SplitRatio = 0.75)

TreeRegression.Train = subset(housingdata_set, TreeRegression.Sample == TRUE)
TreeRegression.Test = subset(housingdata_set, TreeRegression.Sample == FALSE)
```

Generacion del Modelo

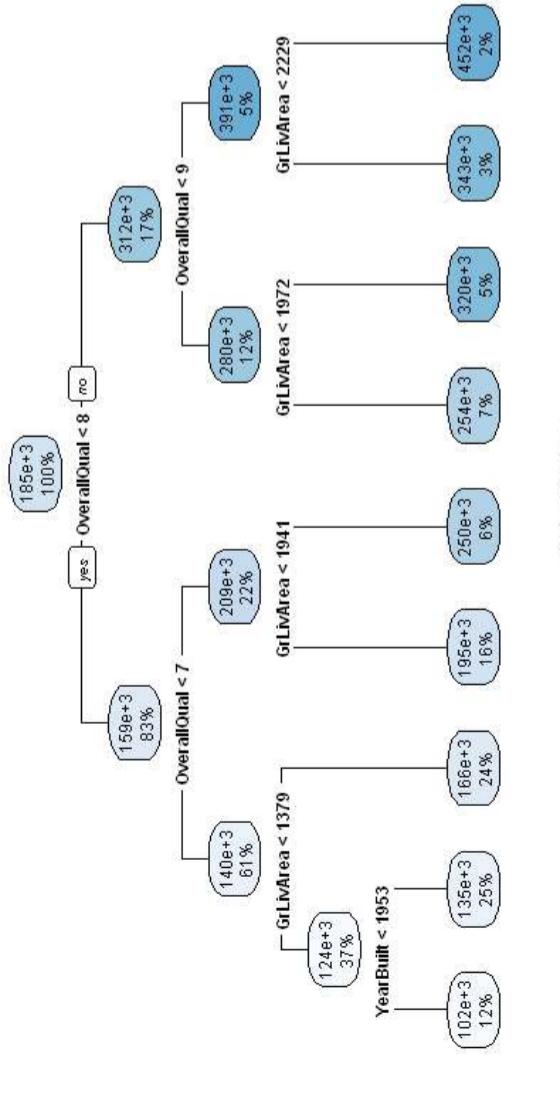
In [60]:

```
#Creacion del arbol de decision - REGRESION
library(rpart)
TreeRegression.Model<- rpart(formula = SalePrice ~ ,
                             data=TreeRegression.Train)
```

In [61]:

```
#Creacion del arbol de decision - REGRESION
TreeRegression.Model<- rpart(SalePrice ~ .,data=TreeRegression.Train)

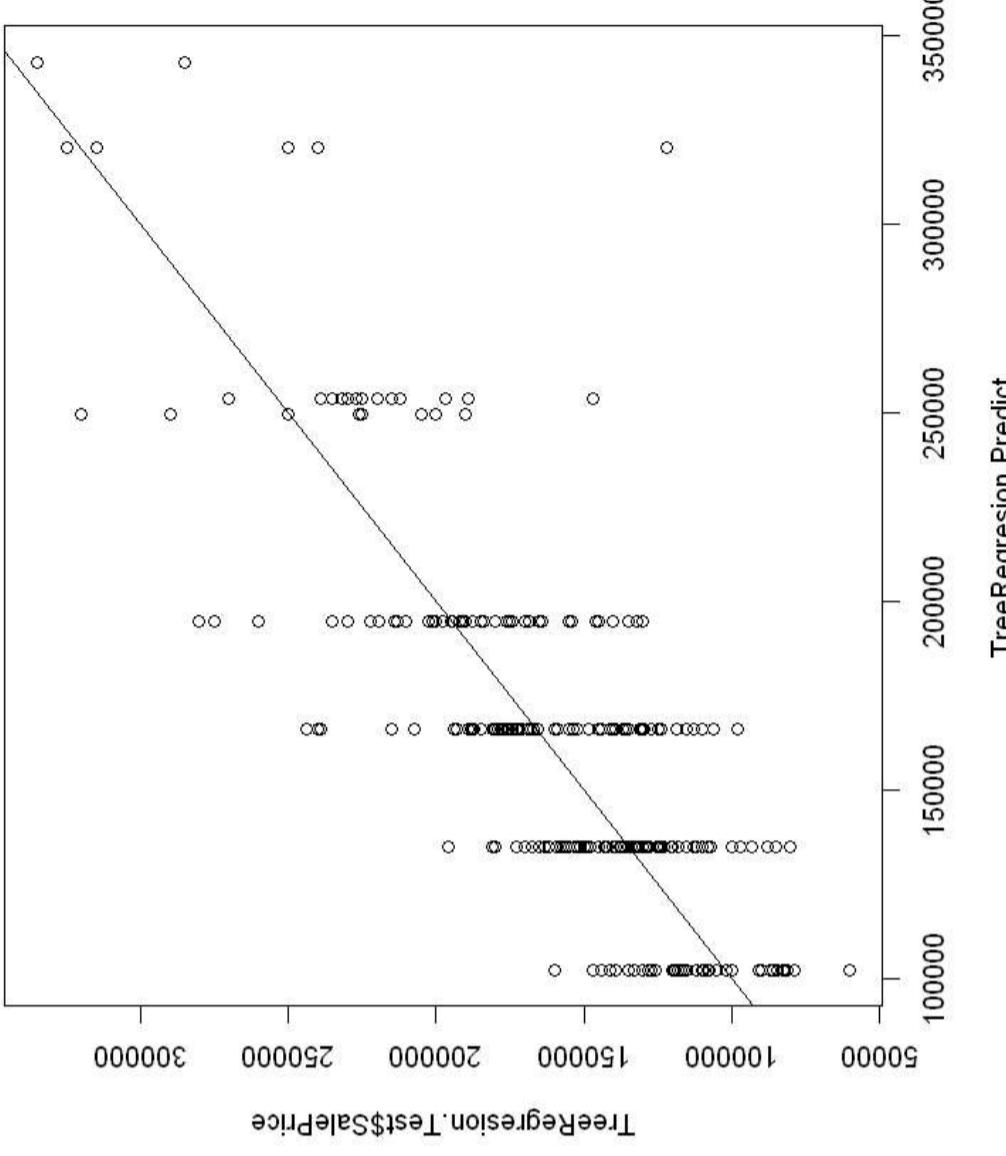
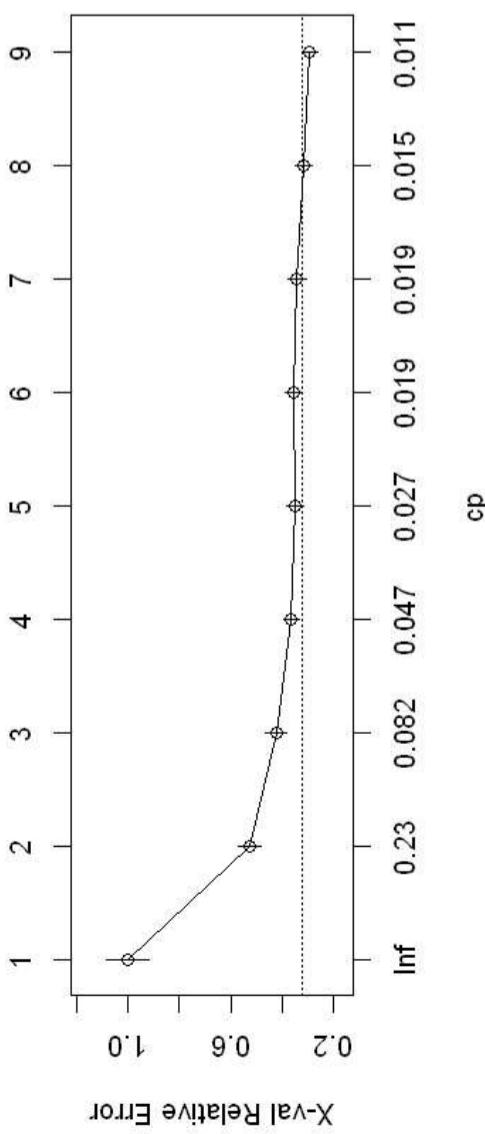
#Gráfico de la Prediccion - REGRESION
par(mfrow=c(2,1))
library(rpart.plot)
rpart.plot(TreeRegression.Model)
plotcp(TreeRegression.Model)
```



Prediccion y Grafico de Relacion de Linealidad

```
In [62]:
#Grafico de Relacion Prediccion vs Test
TreeRegression.Predict <- predict(TreeRegression.Model, newdata = TreeRegression.Test)
plot(TreeRegression.Predict, TreeRegression.Test$SalePrice, main = "arbol sin podar")
abline(0, 1)
```

size of tree



De la grafica extraemos que no es posible podar el arbol dado que con 9 arboles obtenemos el error minimo para el modelamiento

Metrica de Evaluacion (MSE)

```
In [63]:
# Mean Square Error (MSE)
mean((TreeRegression.Predict - TreeRegression.Test$SalePrice)^2)

1153964218.28789
```

In [66]:

```
#Instalacion de librerias
#install.packages("randomForest")
library(randomForest)
library(gmodels)
```

Division de conjuntos de Training y Testing

In [68]:

```
#Particionamos el 70% de las muestras para el conjunto de entrenamiento
RandomForestRegression.Sample = sample.split(housingdata_set$SalePrice, SplitRatio = 0.70)

RandomForestRegression.train = subset(housingdata_set, RandomForestRegression.Sample == TRUE) ##Nuestras del conjunto de entrenamiento
nro
RandomForestRegression.test = subset(housingdata_set, RandomForestRegression.Sample == FALSE) ##Nuestras del conjunto de pruebas
```

Generacion del Modelo

In [69]:

```
#Generamos el modelo de Random Forest
RandomForestRegression.model <- randomForest(x = RandomForestRegression.train[,-1],
y = RandomForestRegression.train$SalePrice,
data = RandomForestRegression.train,
ntree = 300,
do.trace= T)
```

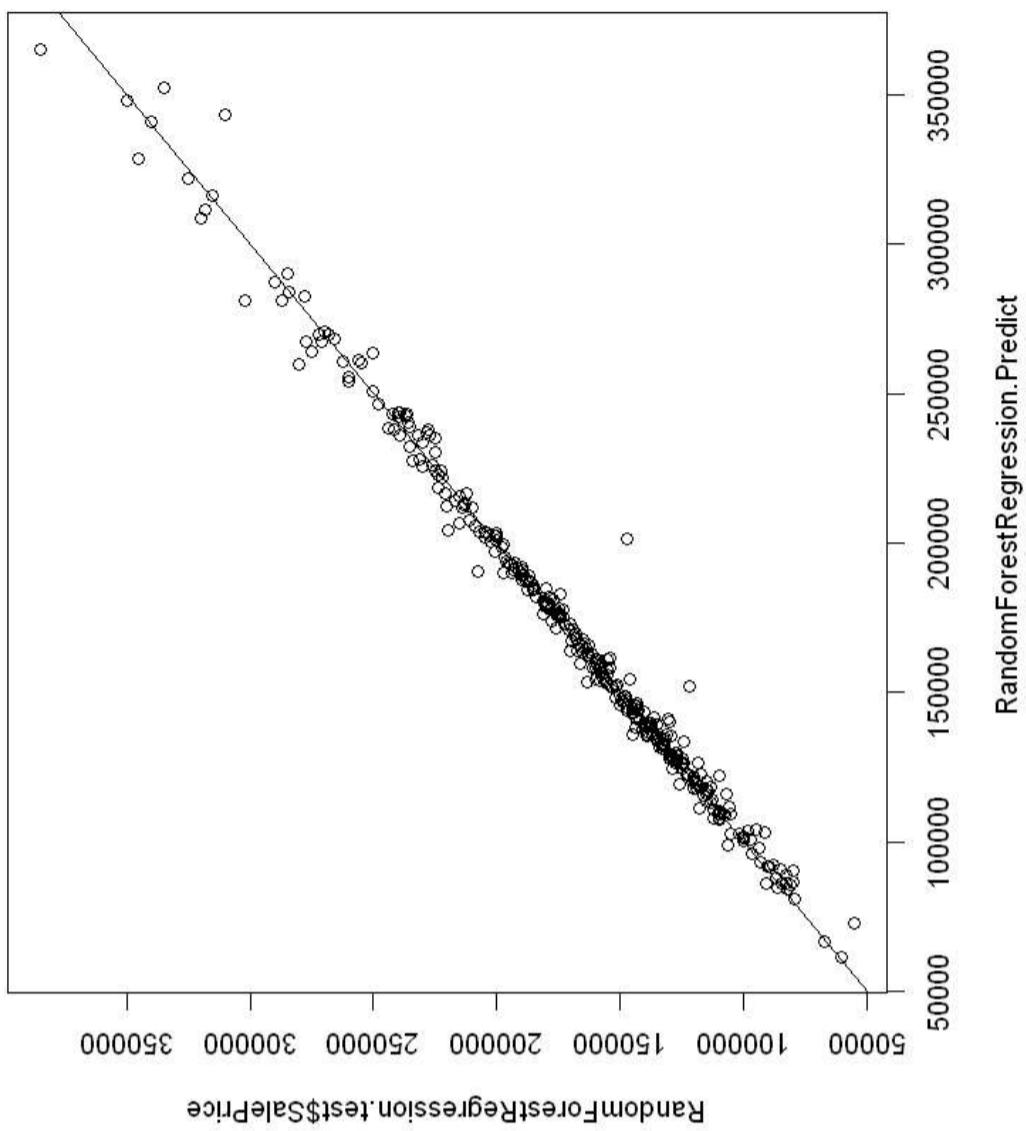
Tree	Out-of-bag			2.24
	MSE	%Var(y)		
1	3.153e+08	4.43		
2	4.934e+08	6.87		
3	4.65e+08	6.47		
4	5.632e+08	7.84		
5	5.38e+08	7.49		
6	4.727e+08	6.58		
7	4.232e+08	5.89		
8	4.226e+08	5.88		
9	3.131e+08	4.36		
10	2.729e+08	3.80		
11	2.719e+08	3.79		
12	2.55e+08	3.48		
13	2.392e+08	3.33		
14	2.229e+08	3.10		
15	2.149e+08	2.99		
16	2.028e+08	2.82		
17	2.156e+08	2.73		
18	2.032e+08	2.90		
19	2.137e+08	2.98		
20	2.01e+08	2.80		
21	1.977e+08	2.75		
22	2.157e+08	2.73		
23	1.983e+08	2.76		
24	1.988e+08	2.76		
25	2.033e+08	2.83		
26	2.055e+08	2.88		
27	2.177e+08	3.03		
28	2.163e+08	3.01		
29	2.07e+08	2.88		
30	1.999e+08	2.78		
31	2.024e+08	2.86		
32	2.019e+08	2.81		
33	2.001e+08	2.79		
34	1.924e+08	2.68		
35	1.995e+08	2.65		
36	1.821e+08	2.54		
37	1.832e+08	2.55		
38	1.863e+08	2.59		
39	1.873e+08	2.61		
40	1.993e+08	2.65		
41	1.824e+08	2.54		
42	1.847e+08	2.57		
43	1.783e+08	2.48		
44	1.771e+08	2.47		
45	1.774e+08	2.47		
46	1.777e+08	2.47		
47	1.769e+08	2.46		
48	1.748e+08	2.43		
49	1.758e+08	2.45		
50	1.718e+08	2.39		
51	1.718e+08	2.39		
52	1.752e+08	2.44		
53	1.711e+08	2.38		
54	1.725e+08	2.40		
55	1.708e+08	2.38		
56	1.752e+08	2.44		
57	1.759e+08	2.42		
58	1.686e+08	2.35		
59	1.68e+08	2.34		
60	1.637e+08	2.28		
61	1.619e+08	2.25		
62	1.636e+08	2.28		
63	1.674e+08	2.33		
64	1.666e+08	2.32		
65	1.664e+08	2.32		
66	1.621e+08	2.26		
67	1.553e+08	2.22		
68	1.555e+08	2.18		
69	1.617e+08	2.25		
70	1.61e+08	2.24		
71	1.605e+08	2.23		
72	1.558e+08	2.21		
73	1.558e+08	2.18		
74	1.561e+08	2.17		
75	1.559e+08	2.21		
80	1.55e+08	2.21		
81	1.556e+08	2.22		
82	1.607e+08	2.24		
83	1.691e+08	2.23		
84	1.584e+08	2.21		
85	1.579e+08	2.20		
86	1.588e+08	2.21		

175	1.52e+08	2.12
176	1.515e+08	2.11
177	1.512e+08	2.11
178	1.553e+08	2.14
179	1.533e+08	2.13
180	1.525e+08	2.12
181	1.521e+08	2.12
182	1.522e+08	2.12
183	1.541e+08	2.14
184	1.542e+08	2.15
185	1.536e+08	2.14
186	1.54e+08	2.14
187	1.535e+08	2.14
188	1.542e+08	2.15
189	1.539e+08	2.17
190	1.551e+08	2.17
191	1.572e+08	2.19
192	1.58e+08	2.17
193	1.554e+08	2.16
194	1.546e+08	2.15
195	1.551e+08	2.16
196	1.543e+08	2.15
197	1.546e+08	2.15
198	1.538e+08	2.14
199	1.549e+08	2.16
200	1.548e+08	2.15
201	1.546e+08	2.15
202	1.551e+08	2.16
203	1.544e+08	2.15
204	1.544e+08	2.15
205	1.529e+08	2.13
206	1.521e+08	2.12
207	1.519e+08	2.11
208	1.522e+08	2.12
209	1.544e+08	2.15
210	1.535e+08	2.14
211	1.521e+08	2.14
212	1.539e+08	2.14
213	1.533e+08	2.13
214	1.539e+08	2.14
215	1.539e+08	2.14
216	1.534e+08	2.14
217	1.539e+08	2.14
218	1.545e+08	2.15
219	1.534e+08	2.14
220	1.527e+08	2.13
221	1.536e+08	2.14
222	1.543e+08	2.15
223	1.544e+08	2.15
224	1.545e+08	2.15
225	1.548e+08	2.16
226	1.548e+08	2.16
227	1.547e+08	2.15
228	1.533e+08	2.13
229	1.528e+08	2.13
230	1.525e+08	2.12
231	1.533e+08	2.13
232	1.531e+08	2.13
233	1.539e+08	2.14
234	1.545e+08	2.15
235	1.534e+08	2.16
236	1.533e+08	2.16
237	1.56e+08	2.17
238	1.534e+08	2.16
239	1.546e+08	2.15
240	1.551e+08	2.16
241	1.543e+08	2.15
242	1.534e+08	2.14
243	1.536e+08	2.14
244	1.536e+08	2.14
245	1.533e+08	2.12
246	1.52e+08	2.12
247	1.519e+08	2.12
248	1.52e+08	2.12
249	1.515e+08	2.11
250	1.51e+08	2.10
251	1.504e+08	2.09
252	1.514e+08	2.11
253	1.511e+08	2.10
254	1.503e+08	2.09
255	1.496e+08	2.08
256	1.492e+08	2.08
257	1.488e+08	2.07
258	1.482e+08	2.06
259	1.479e+08	2.06
260	1.474e+08	2.05
261	1.472e+08	2.05
262	1.47e+08	2.05

Prediccion y Grafico de Relacion de Linealidad

In [70]:

```
#Grafico de Relacion Prediccion vs Test
RandomForestRegression.Predict <- predict(RandomForestRegression.model,
newdata = RandomForestRegression.test)
plot(RandomForestRegression.Predict, RandomForestRegression.test$SalePrice, main = "arbol sin podar")
abline(0, 1)
```



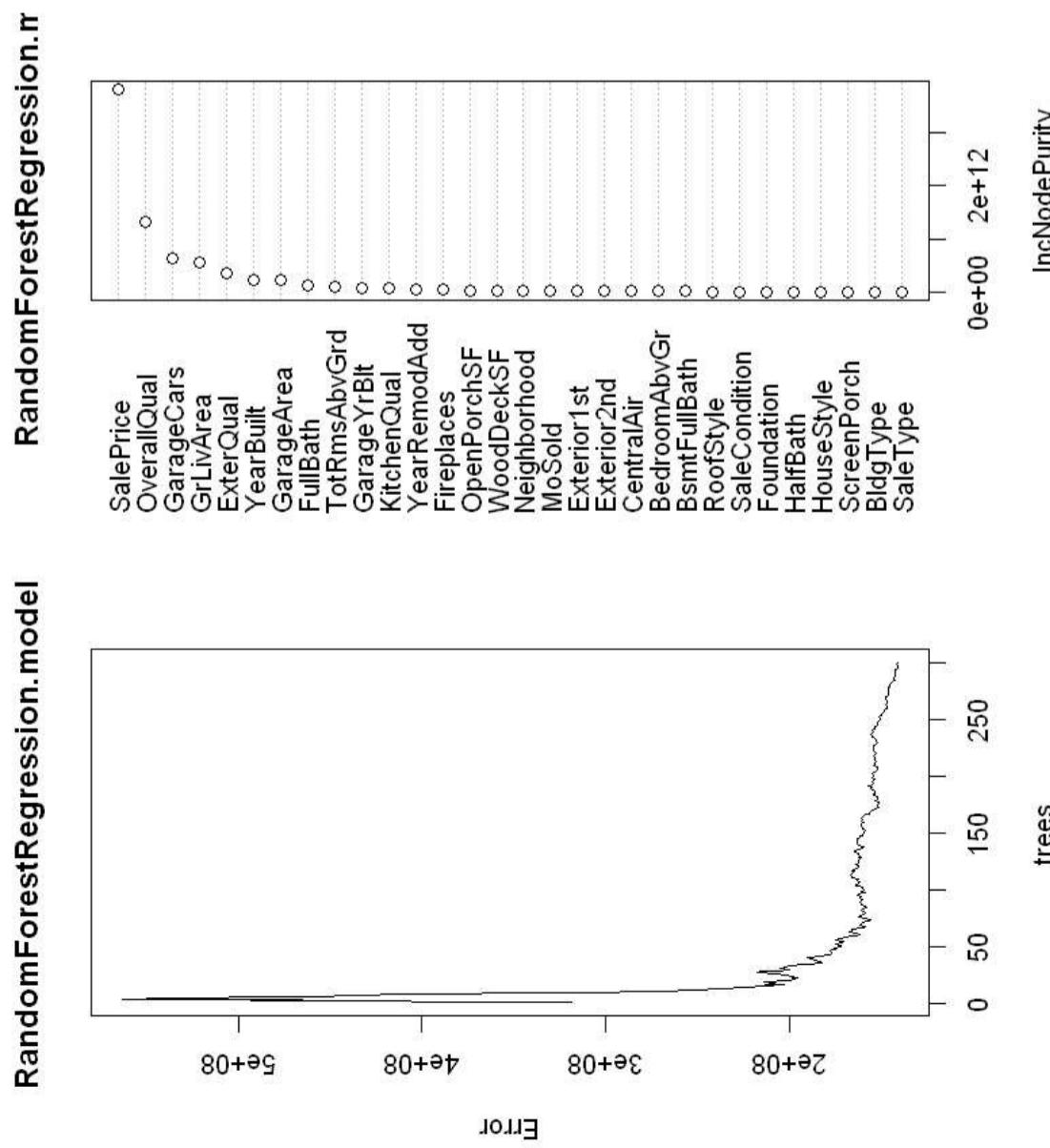
Metrica de evaluacion (MSE)

```
In [71]:  
#Mean Square Error (MSE)  
mean((RandomForestRegression$Predict - RandomForestRegression$test$SalePrice)^2)  
36178452.8378947
```

Grafica del Modelo y Variables Importantes

```
In [73]:  
par(mfcol=c(1,2))  
plot(RandomForestRegression$model)  
varImpPlot(RandomForestRegression$model)
```

RandomForestRegression.model



RandomForestRegression.r

```
In [74]:  
housingdata_set_bin<-housingdata_set #Copiamos temporalmente el dataset a otro dataframe para fines de discretizacion  
b <- c(-Inf, 101000, 500000, Inf) #Creamos un vector con Los limites para la agrupacion de 3 grupos:  
names <- c("<100K", "101K-500K", ">501K")# Establecemos el nombre de Los grupos que reemplazaran Los rangos de valores  
housingdata_set_bin$SalePrice <- cut(housingdata_set_bin$SalePrice, breaks = b, labels = names) #Cortamos el vector columna usando  
Los breaks de b  
housingdata_set_bin$SalePrice <- as.factor(housingdata_set_bin$SalePrice) #Convertimos a factor La columna de variables.  
unique(housingdata_set_bin$SalePrice) #Visualizamos Los grupos establecidos para La columna de SalePrice
```

101K-500K <100K >501K

► Levels:

Los resultados nos indican que se agrupo exitosamente en 3 grupos seguiendo el rango de valores asignados para el precio del inmueble

```
In [76]:  
#Division de datos de Training y Testing - CLASIFICACION  
TreeClassification$Sample = sample.split(housingdata_set_bin$SalePrice, SplitRatio = 0.63) #Extraemos una muestra aleatoria del 65% del total de muestras  
TreeClassification$Train = subset(housingdata_set_bin, TreeClassification$Sample == TRUE) #Repartimos el 65% para el dataset de entrenamiento  
TreeClassification$Test = subset(housingdata_set_bin, TreeClassification$Sample == FALSE) #Repartimos el porcentaje restante al dataset de prueba
```

Division de datos de Training y Testing

Generacion y Grafica del Modelo

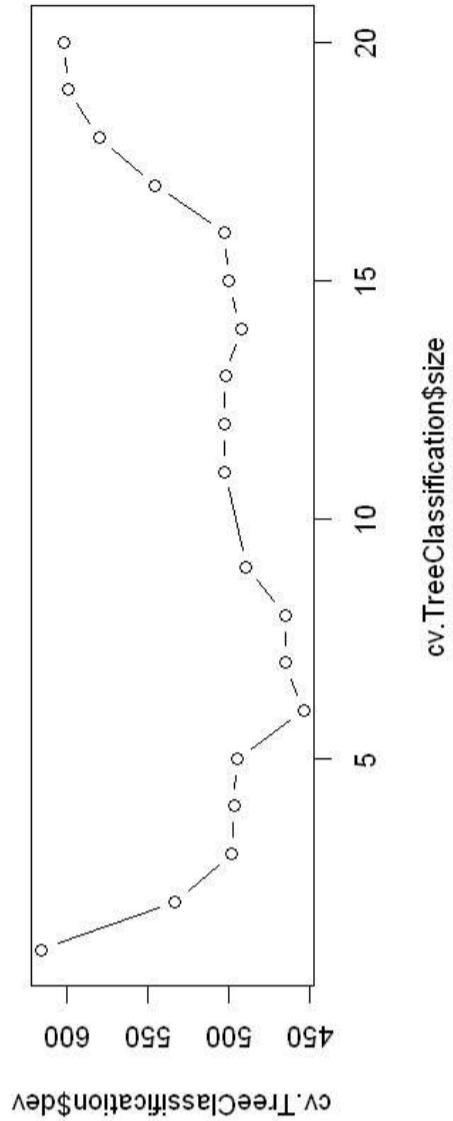
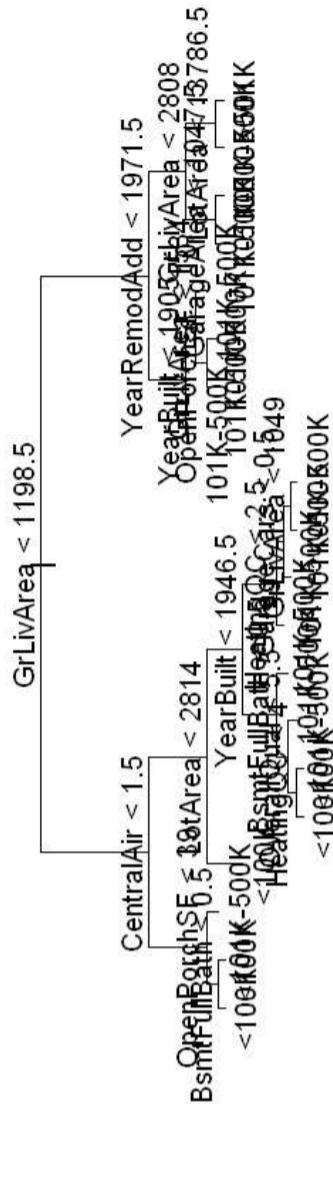
CLASIFICACION CON ARBOL DE DECISION

Agrupacion de variables como factores

In [77]:

```
TreeClassification.Model <- tree(SalePrice ~ . , TreeClassification.Train) #Generamos el arbol definiendo a SalePrice como variable dependiente
e <- dependence
par(mfrow=c(2,1))
plot(TreeClassification.Model)
text(TreeClassification.Model, pretty = 0)

#Cross validation
cv.TreeClassification<- cv.tree(TreeClassification.Model, K = 10) #Realizamos la validacion cruzada con 10 Folds
plot(cv.TreeClassification$size, cv.TreeClassification$dev, type = 'b') #Visualizamos con cuantos arboles es mas efectivo el arbol
```



In [79]:

```
#Segun la grafica Lineas arriba, nos indica que la cantidad mas efectiva de arboles para el modelo es con 7 arboles, por lo que se
teamos dicho parametro para podar el arbol
TreeClassification.Model_Podado <- prune(TreeClassification.Model,best = 7) #Establecemos 7 arboles como el mejor arbol para podar
plot(TreeClassification.Model_Podado)
#Graficamos el arbol podado
text(TreeClassification.Model_Podado, pretty = 0) #Establecemos el texto que describe las variables.
```

Cálculo #1 de las métricas de evaluación

Predicción y Matriz de Confusión

In [78]:

```
#Verificamos que posicion de columna es SalePrice
pos_SalePrice=which(colnames(TreeClassification.Test)== "SalePrice" )
# Prediccion en los resultados con el conjunto de testing
TreeClassification.Predict = predict(TreeClassification.Model,
newdata = TreeClassification.Test[, -pos_SalePrice],
type = "class")

#Matriz de confusión por numero de casos
table = table(TreeClassification.Test$SalePrice, TreeClassification.Predict,
dnn = c("actual", "predicho"))

table
```

Predecido	<100K	101K-500K	>501K
actual	28	18	0
<100K	10	475	6
101K-500K	0	2	1
>501K			

- La matriz de confusión indica que para el modelo construido indica que la mejor predicción se centra en el grupo de 101K-500K, y es evidente dado que es donde se concentra la mayor parte de las variables, y es lo contrario que pasa para el grupo de >501K, el cual tiene muy poco valores por lo que el modelo mantiene un error que confunde al grupo de 101K-500K, finalmente el grupo <100K mantiene un aprox de 50% de efectividad.

Generación del Podado del Árbol

- La grafica nos muestra que efectivamente ha logrado clasificar los tres diferentes grupos tras el podado del arbol.

GrLivArea < 1198.5

Calculo #2 de las metricas de evaluacion

Prediccion y matriz de confusión

In [80]:



In [86]:

```
RandomForestClassification.Sample = sample.split(housingdata_set_bin2$SalePrice, SplitRatio = 0.70) # Muestras aleatorias para el
conjunto de entrenamiento
RandomForestClassification.train = subset(housingdata_set_bin2, RandomForestClassification.Sample == TRUE) #Conjunto de entrenamiento
nto
RandomForestClassification.test = subset(housingdata_set_bin2, RandomForestClassification.Sample == FALSE) #Conjunto de test
```

Generación del Modelo

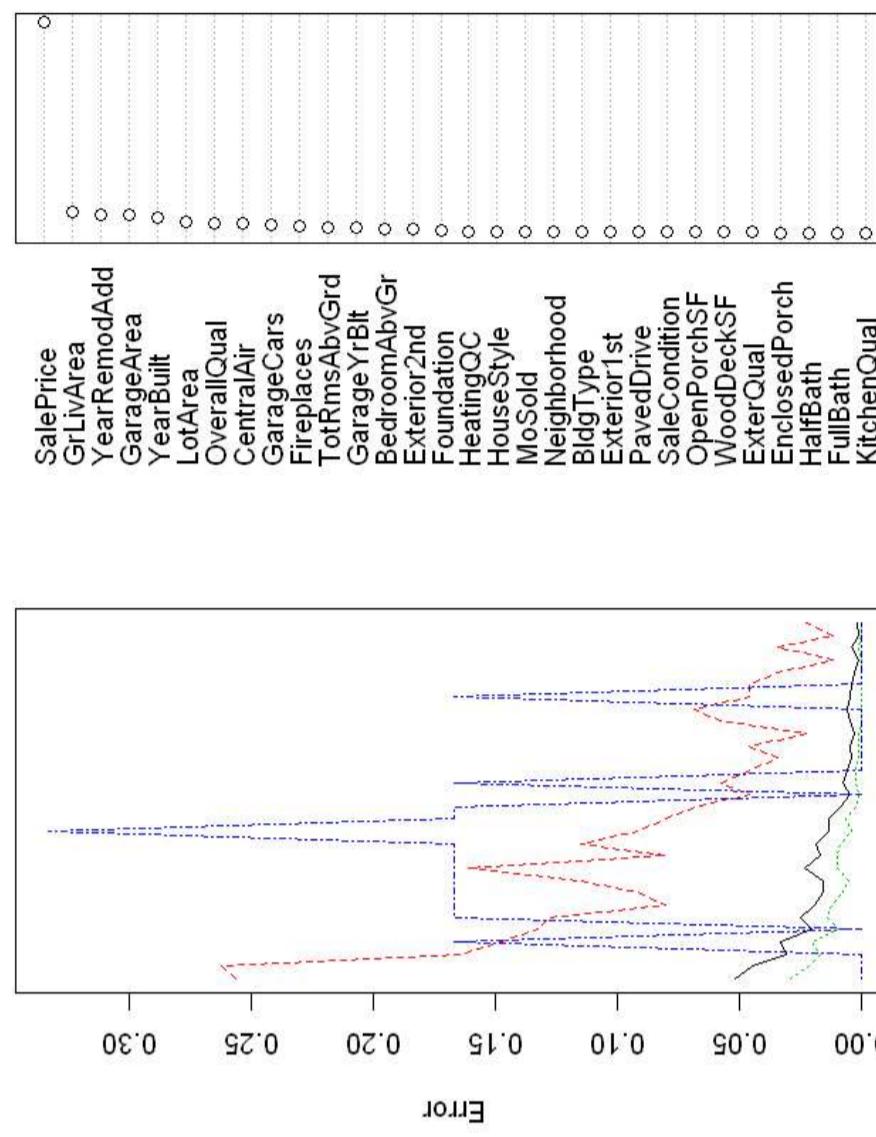
```
In [87]:
columnas <-c("GrLivArea", "OverallQual", "GarageCars", "TotalBsmtSF", "X1stFlrSF", "GarageArea", "FullBath" )
#Entrenamos el Modelo de RandomForest para la clasificaciones de grupos
RandomForestClassification.model1 <- randomForest(
  x = RandomForestClassification.train,
  y = RandomForestClassification.train$SalePrice,
  data = RandomForestClassification.train,
  ntree = 30,
  do.trace= T)
```

ntree	008	1	2	3
1:	5.21%	25.64%	2.92%	0.00%
2:	4.47%	26.23%	2.14%	0.00%
3:	3.08%	16.44%	1.71%	0.00%
4:	3.34%	14.81%	2.05%	16.67%
5:	2.05%	13.25%	0.96%	0.00%
6:	2.50%	12.79%	1.38%	16.67%
7:	1.93%	8.05%	1.24%	16.67%
8:	1.59%	9.20%	0.77%	16.67%
9:	1.58%	11.49%	0.54%	16.67%
10:	2.36%	16.09%	0.98%	16.67%
11:	1.67%	8.05%	0.97%	16.67%
12:	1.86%	11.49%	0.86%	16.67%
13:	1.37%	9.20%	0.43%	33.33%
14:	1.37%	8.05%	0.65%	16.67%
15:	0.88%	6.90%	0.22%	16.67%
16:	0.49%	4.60%	0.11%	0.00%
17:	0.78%	5.75%	0.22%	16.67%
18:	0.59%	4.60%	0.22%	0.00%
19:	0.39%	3.45%	0.11%	0.00%
20:	0.49%	4.60%	0.11%	0.00%
21:	0.29%	2.30%	0.11%	0.00%
22:	0.49%	5.75%	0.00%	0.00%
23:	0.59%	6.90%	0.00%	0.00%
24:	0.49%	4.60%	0.00%	16.67%
25:	0.39%	4.60%	0.00%	0.00%
26:	0.29%	3.45%	0.00%	0.00%
27:	0.10%	1.15%	0.00%	0.00%
28:	0.39%	3.45%	0.11%	0.00%
29:	0.10%	1.15%	0.00%	0.00%
30:	0.20%	2.30%	0.00%	0.00%

-Se observa que a medida que se van generando mas arboles el error OOB va reduciendo, sin embargo se ve claramente que dependiendo de la cantidad de valores que contienen cada grupo, la cantidad de arboles requeridos varia. Es decir para el Grupo 3(>500K) al algoritmo le cuesta mas arboles para mejorar su clasificación dado que su rango de valores son pocos, mientras que en el grupo 2(entre 101K y 500K) que contienen gran cantidad de valores, el error se reduce con poca cantidad de arboles.

Grafica del Modelo y variables importantes

RandomForestClassification.modelo RandomForestClassification.



MeanDecreaseGini

- En la primera grafica se aprecia el error segun la cantidad de arboles, para este algoritmo seleccionamos 30 arboles, ya que al algoritmo le permite generalizar mas la clase a predecir, sin embargo si bajamos la cantidad de arboles se observa que el error se expande mientras que si incrementamos el numero de arboles logramos un 100% de acierto generando un overfitting en el algoritmo.
- la segunda grafica se puede visualizar las variables mas importantes que se han incluido para la creacion del Random Forest.

Predicción y matriz de confusión

In [89]:

```
RandomForestClassification.model  
  
Call:  
  randomForest(x = RandomForestClassification.train, y = RandomForestClassification.train$SalePrice,  
  ntree = 30,  
  do.trace = T, data = RandomForestClassification.train)  
  Type of random forest: classification  
  Number of trees: 30  
  No. of variables tried at each split: 6  
  OOB estimate of error rate: 0.2%  
  
Confusion matrix:  
             <100K 101K-500K >501K class.error  
<100K      85       2     0 0.02299851  
101K-500K    0      929     0 0.00000000  
>501K      0       0     6 0.00000000
```

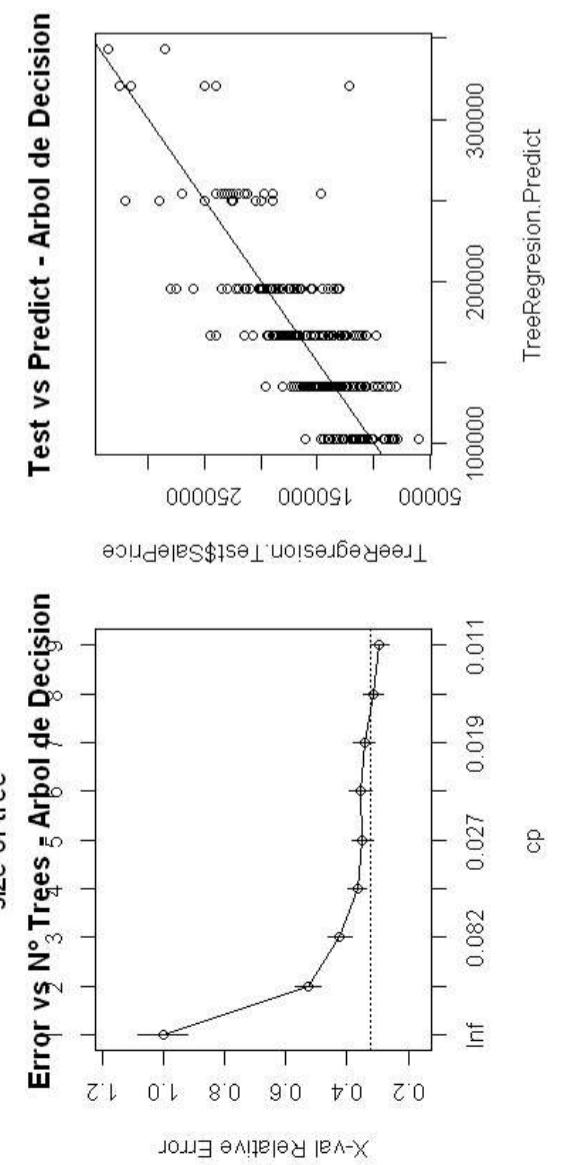
- La matriz de confusión nos indica que el modelo ha clasificado exitosamente con un OOB tasa de error estimado de 0.50%, por lo que además si ajustamos el numero de arboles para reducir el error a 0%, este clasificador ya no logaría generalizar las clases , causando un problema mayor de sobre ajuste, en este punto se puede concluir que RandomForest a superado significativamente al Arbol de decision de clasificación.

COMPARACION DE RESULTADOS FINALES

Comparacion de Error Cuadratico - Regresion

In [90]:

```
par(mfcol=c(2,2))  
  
#Grafica del Error vs Numero de Arboles - Arbol de Decision  
plotcp(TreeRegression.Model, main= "Error vs N° Trees - Arbol de Decision")  
#Grafica del Error vs Numero de Arboles - RandomForest  
plot(RandomForestRegression.model, main= "Error vs N° Trees - RandomForest")  
  
#Grafico de Relacion Prediccion vs Test - Arbol de Decision  
plot(TreeRegression.Predict, TreeRegression.Test$SalePrice, main = "Test vs Predict - Arbol de Decision")  
abline(0, 1)  
  
#Grafico de Relacion Prediccion vs Test - RandomForest  
RandomForestRegression.Predict <- predict(RandomForestRegression.model,  
newdata = RandomForestRegression.test)  
plot(RandomForestRegression.Predict, RandomForestRegression.test$SalePrice, main = "Test vs Predict - RandomForest")  
abline(0, 1)
```



Las graficas muestran que RandomForest ha conseguido una mejor predicción que el arbol de decisión.

```
In [91]:
#Modelo Regression - ARBOL DE DECISION
# Mean Square Error (MSE)
mean((TreeRegression.Predict - TreeRegression.Test$SalePrice)^2)

1153964218.28789
```



```
In [92]:
#Modelo Regression - RANDOM FOREST
#Mean Square Error (MSE)
mean((RandomForestRegression.Predict - RandomForestRegression.test$SalePrice)^2)

36178452.8378947
```

Conclusion:

- Comparando el MSE con ambos algoritmos, se concluye que el mejor es el algoritmo RandomForest, dado que presenta un menor error cuadrático, a pesar que no se ha obtenido un error mínimo, se puede observar la gran diferencia significativa entre ambos modelos de regresión.

Comparacion Matriz de confusion - Clasificación

```
In [97]:
print("*****ARBOL DE DECISION - Matriz de Confusion:")
print("Matriz de confusión por numero de casos")
table_podado = table(TreeClassification.train$SalePrice, TreeClassification.Predict_Podado,
table_podado
dnn = c("actual", "Predicido(Luego de ser Podado)")

print("*****RANDOM FOREST - Matriz de Confusion:")
print("Matriz de confusión por numero de casos")
RandomForestClassification.model
[1] "*****RANDOM FOREST - Matriz de Confusion:"

[1] "*****ARBOL DE DECISION - Matriz de Confusion:"
RandomForestClassification.model
Predictido(Luego de ser Podado)
actual <100K 101K-500K >501K
<100K 20 26 0
101K-500K 5 486 0
>501K 0 3 0

[1] "*****RANDOM FOREST - Matriz de Confusion:"
```

Call:

```
randomForest(x = RandomForestClassification.train, y = RandomForestClassification.train$SalePrice,
do.trace = T, data = RandomForestClassification.train)
```

Type of random forest: classification

Number of trees: 30

No. of variables tried at each split: 6

OOB estimate of error rate: 0.2%

Confusion matrix:

	<100K	101K-500K	>501K
<100K	85	2	0
101K-500K	0	929	0
>501K	0	0	6

Conclusion:

- Comparando la matriz de confusión obtenida por ambos modelos, se concluye que el mejor es el algoritmo de RandomForest, dado que mantiene una mejor exactitud independientemente del overfitting que pueda presentar.

Ventajas y Desventajas segun Resultados obtenidos

Algoritmo de Árbol de decisión

Las graficas muestran que RandomForest ha conseguido una mejor predicción que el arbol de decisión.

Ventajas:

- Segun las pruebas realizadas, tiene un mejor performance en clasificacion respecto a regresion, por lo que se afirma que para una gran cantidad de datos, la eleccion de un modelo de clasificacion seria lo mas adecuado.
- Durante la compilacion de la construccion del arbol, se observa que no presenta un alto coste computacional, lo que logra reducir el tiempo en la generacion del modelo.
- Debido a su facil interpretacion de la construccion del arbol, es posible seguir intuitivamente las decisiones que se van tomando para llegar al objetivo final.

Desventajas:

- Estos algoritmos no tienen la misma exactitud en predicción en comparación con RandomForest, la cual se puede apreciar en el gráfico de relación test vs predict.
- Para obtener un buen resultado a comparación de RandomForest, este algoritmo requiere una mayor preparación de limpieza de datos ya que se ve afectado a los outliers que pudieron no estar bien procesados.
- Durante pruebas se observa que no posee buena exactitud en predecir valores continuos por lo que se concluye que algo inadecuado crear modelo de regresion cuando se trabaja con gran cantidad de datos de entrada, no obstante podría trabajar mejor con menos cantidad de input para modelos de regresion.

Algoritmo de RandomForest

Ventajas:

- Segun la relacion de Test vs Predict, el algoritmo maneja mejor los outliers a comparacion del arbol de decision, el cual se ve afectado por la dispersion de estos valores atipicos, por lo que se concluye que es robusto frente a los outliers.
- Segun la matriz de confusion se puede visualizar que tiene una mejor prediccion frente a los arboles de decision, independientemente del overfitting que pueda tener.

Desventajas:

- Segun la compilacion realizada toma mas coste computacional ya que el algoritmo debe crear mas arboles, para este caso a partir de 50 arboles ya podia alcanzar la estabilidad del error minimo.
- No es posible interpretar las decisiones que va tomando los arboles, en la cual si es posible en los arboles de decision.
- Segun se observa en la matriz de confusion y grafico de relacion test vs predict, estos algoritmos tienden a generar overfitting para modelos de clasificacion o regresion con outliers.

Otros comentarios

- Para los modelos de clasificación en RandomForest y Arbol de decisión, se observa que debido a que la cantidad del grupo (SalePrice>500000) es muy pequeña, se tiene un rango de mayor errores que con respecto a los otros grupos de clasificación de SalePrice, cuyo rango de valores son mayores por ende los aciertos que hallaron.
- De las pruebas realizadas, se ve claramente que el modelo de regresión basado en arbol decision es más vulnerable a los outliers en comparacion a RandomForest regression.
- En la practica, independientemente de la limpieza de datos, para poder mejorar el modelo, se observa que el modelo debe pasar pruebas con diferente tamaño de muestras de entrenamiento, el cual segun las pruebas variando el grupo de entrenamiento en un rango de 50% a 80% se consigue estabilizar y mejorar el modelo predictivo