

# Máster Universitario en Ciencia de Datos e Ingeniería de Computadores



**Manuel Montero Santana:** [manuelms1993@gmail.com](mailto:manuelms1993@gmail.com)

**Introducción a la Ciencia de Datos**

Trabajo final sobre regresión y clasificación

## ANÁLISIS DE DATOS

En primer lugar realizaremos un estudio previo sobre los dos conjuntos y describiremos de manera básica algunas variables de los datasets.

### **Conjunto 1 → Treasury**

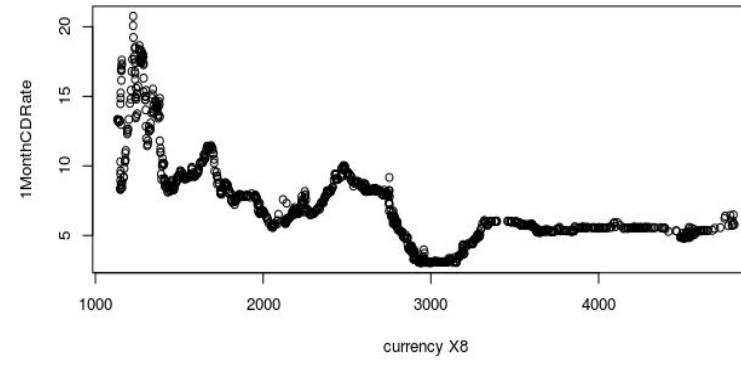
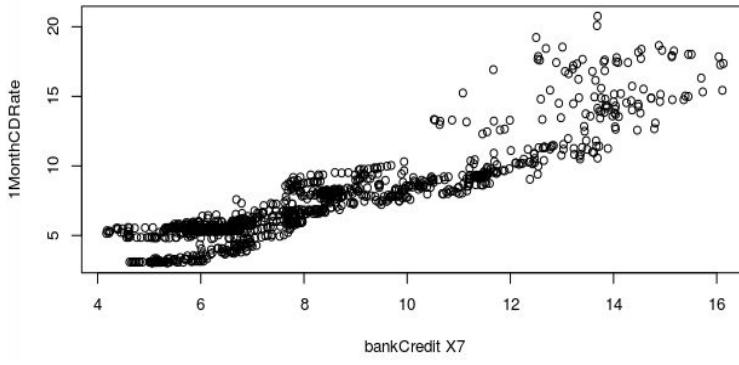
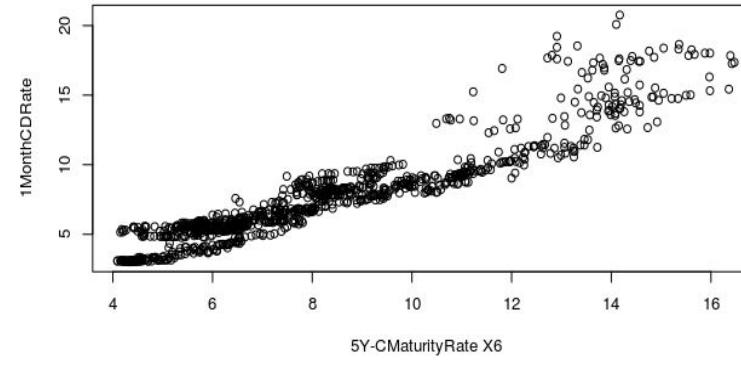
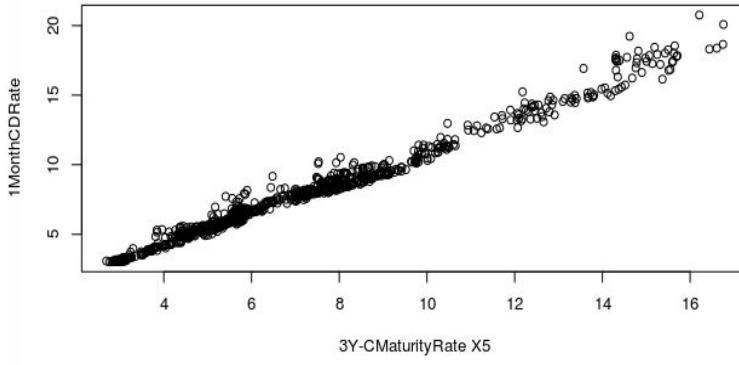
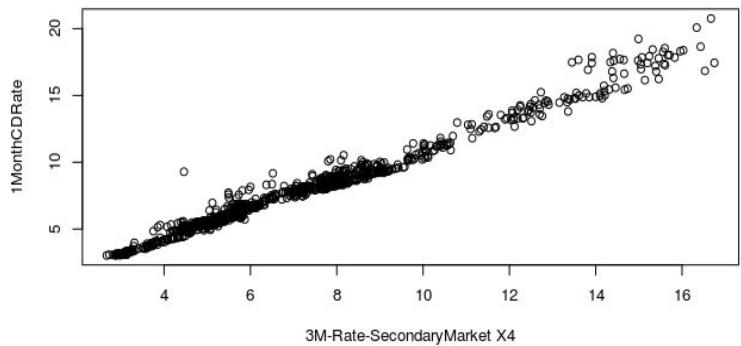
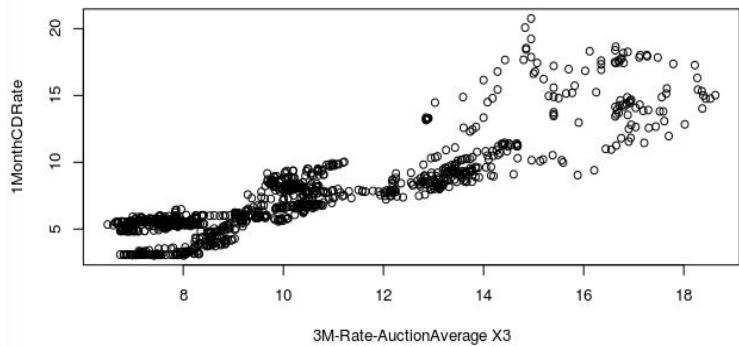
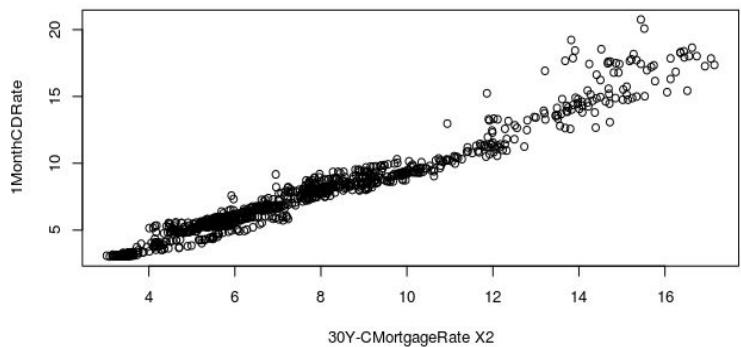
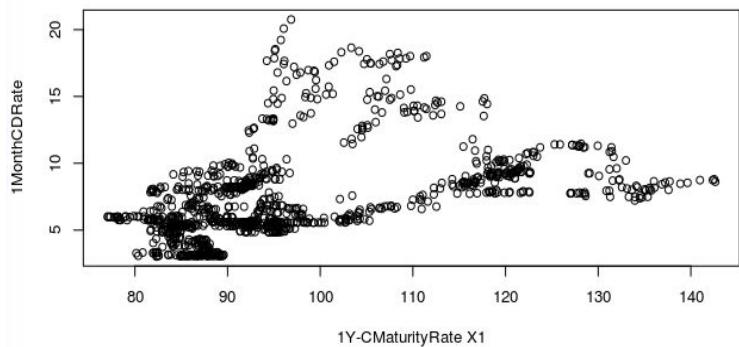
Este archivo contiene la información de datos económicos de EE.UU. del 01/04/1980 al 02/04/2000 semanalmente. De las características dadas, la meta es predecir 1 Month Rate.

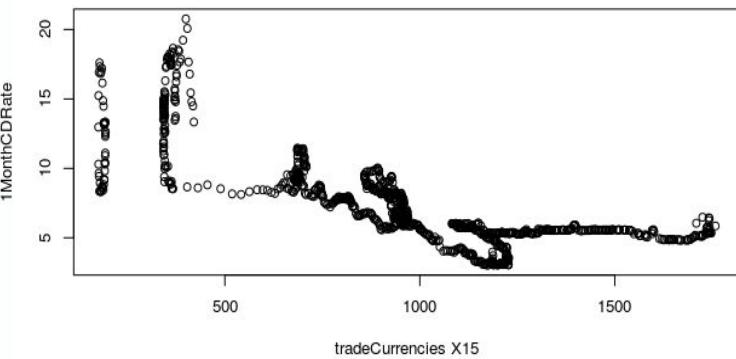
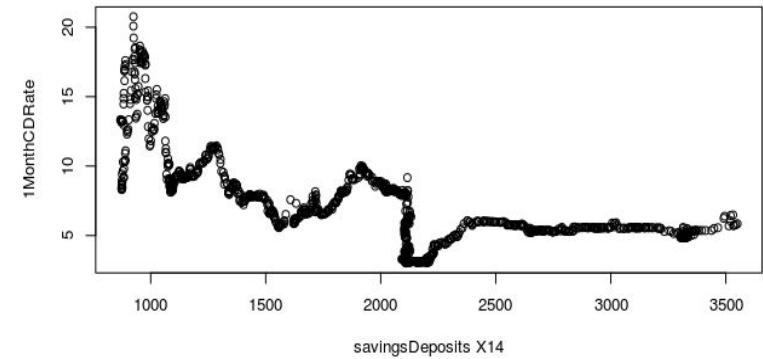
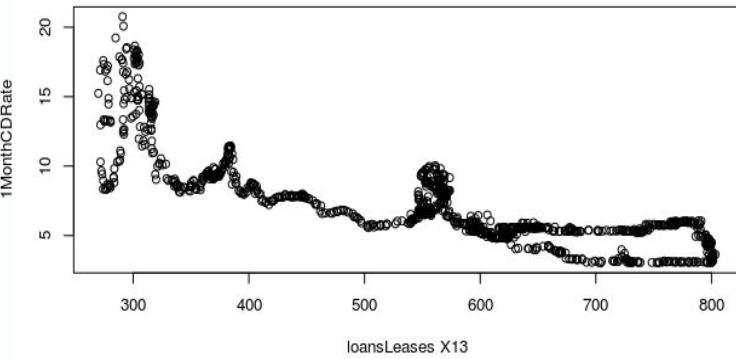
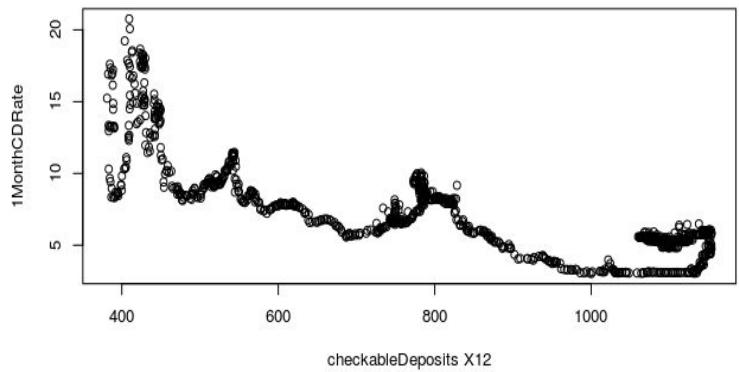
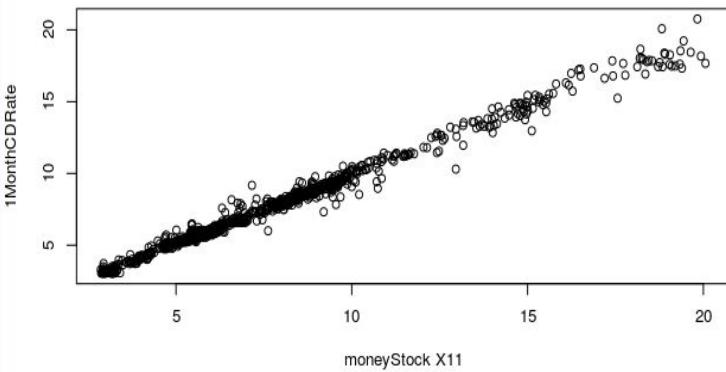
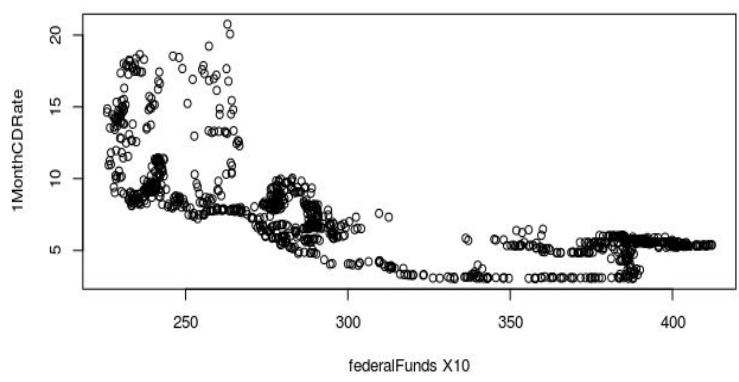
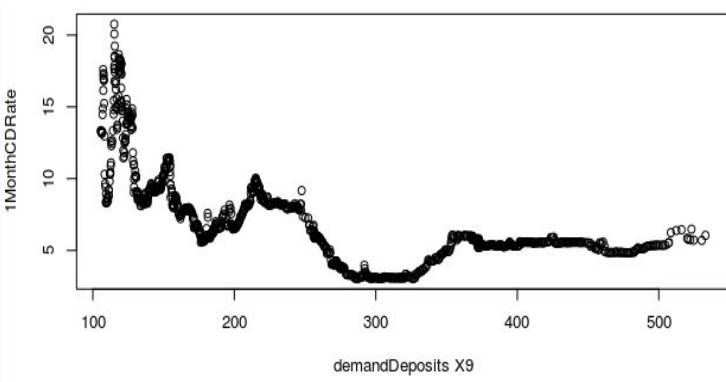
#### **1. Cálculo de media, desviación estándar, etc.**

Treasury	SD	MEAN	MAX	MIN
<b>Tesorería a 1 año Constante Tasa de vencimiento (1Y-CMaturityRate )</b>	14.477436	97.358654	142.645	77.055
<b>Tasa Hipotecaria Convencional de 30 Años (30Y-CMortgageRate)</b>	3.107155	7.543111	17.150	3.020
<b>Promedio de la subasta de Tesorería de 3 meses (3M-Rate-AuctionAverage)</b>	2.960285	10.400840	18.630	6.490
<b>Tasa Mercado Secundario a 3 Meses (3M-Rate-SecondaryMarket)</b>	2.955578	6.850401	16.750	2.670
<b>Tasa de vencimiento constante Tesorería a 3 años (3Y-CMaturityRate)</b>	2.943548	6.828454	16.760	2.690
<b>Tasa de Vencimiento Constante Tesorería a 5 Años (5Y-CMaturityRate)</b>	2.885180	8.116727	16.470	4.090
<b>Crédito bancario de todos los bancos comerciales (bankCredit)</b>	2.767537	8.358693	16.130	4.170
<b>Divisas en Stock (currency)</b>	1011.002725	2639.652767	4809.200	1130.900
<b>Depósitos en bancos comerciales (demandDeposits)</b>	114.627114	256.873092	533.000	105.600
<b>Tasa de Fondos Federales Efectiva (federalFunds)</b>	59.825813	308.145324	412.100	225.800
<b>Dinero en Stock M1 (moneyStock)</b>	3.540282	7.548807	20.060	2.860
<b>Total Depósitos Comprobables (checkableDeposits)</b>	258.811979	813.336450	1154.100	381.100
<b>Préstamos y Arrendamientos (loansLeases)</b>	157.146220	549.667748	803.400	269.900
<b>Depósitos totales de ahorros (savingsDeposits)</b>	720.868819	1959.029198	3550.300	868.100
<b>Índice de cambio ponderado en el comercio de monedas (tradeCurrencies)</b>	372.467967	954.709828	1758.100	175.600
<b>Ratio a 1 mes (1MonthCDRate)</b>	3.378743	7.521202	20.760	3.020

## 2. Gráficos que permitan visualizar los datos adecuadamente.

Variables en función de la salida





### **3. Descripción del conjunto de datos a partir de los puntos anteriores.**

Casi todas las variables a priori parecen susceptibles de poder tomarse en cuenta ya que tienen una estructura adecuada para una función polinómica, quizás las que parecen menos dadas a estas características sean *1Y-CMaturityRate* y *3M-Rate-AuctionAverage*, las cuales tienen un poco más dispersos sus valores.

En el código del programa haré una transformación sobre los nombres de las variables originales para eliminar los guiones y los números al principio del nombre de las variables.

## Conjunto 2 → Vehicle

### 1. Cálculo de media, desviación estándar, etc.

Vehicle	SD	MEAN	MAX	MIN
Compactness	8.239225	93.676923	119	73
Circularity	6.172573	44.857988	59	33
Distance_circularity	15.780842	82.087574	112	40
Radius_ratio	33.490553	168.930178	333	104
Praxis_aspect_ratio	7.884937	61.681657	138	47
Max_length_aspect_ratio	4.603677	8.565680	55	2
Scatter_ratio	33.263833	168.847337	265	112
Elongatedness	7.816100	40.932544	61	26
Praxis_rectangular	2.593596	20.583432	29	17
Length_rectangular	14.519305	147.985799	188	118
Major_variance	31.410420	188.640237	320	130
Minor_variance	176.784811	439.983432	1018	184
Gyration_radius	32.564191	174.692308	268	109
Major_skewness	7.490928	72.465089	135	59
Minor_skewness	4.921249	6.377515	22	0
Minor_kurtosis	8.935762	12.595266	41	0
Major_kurtosis	6.167241	188.934911	206	176
Hollows_ratio	7.443054	195.630769	211	181

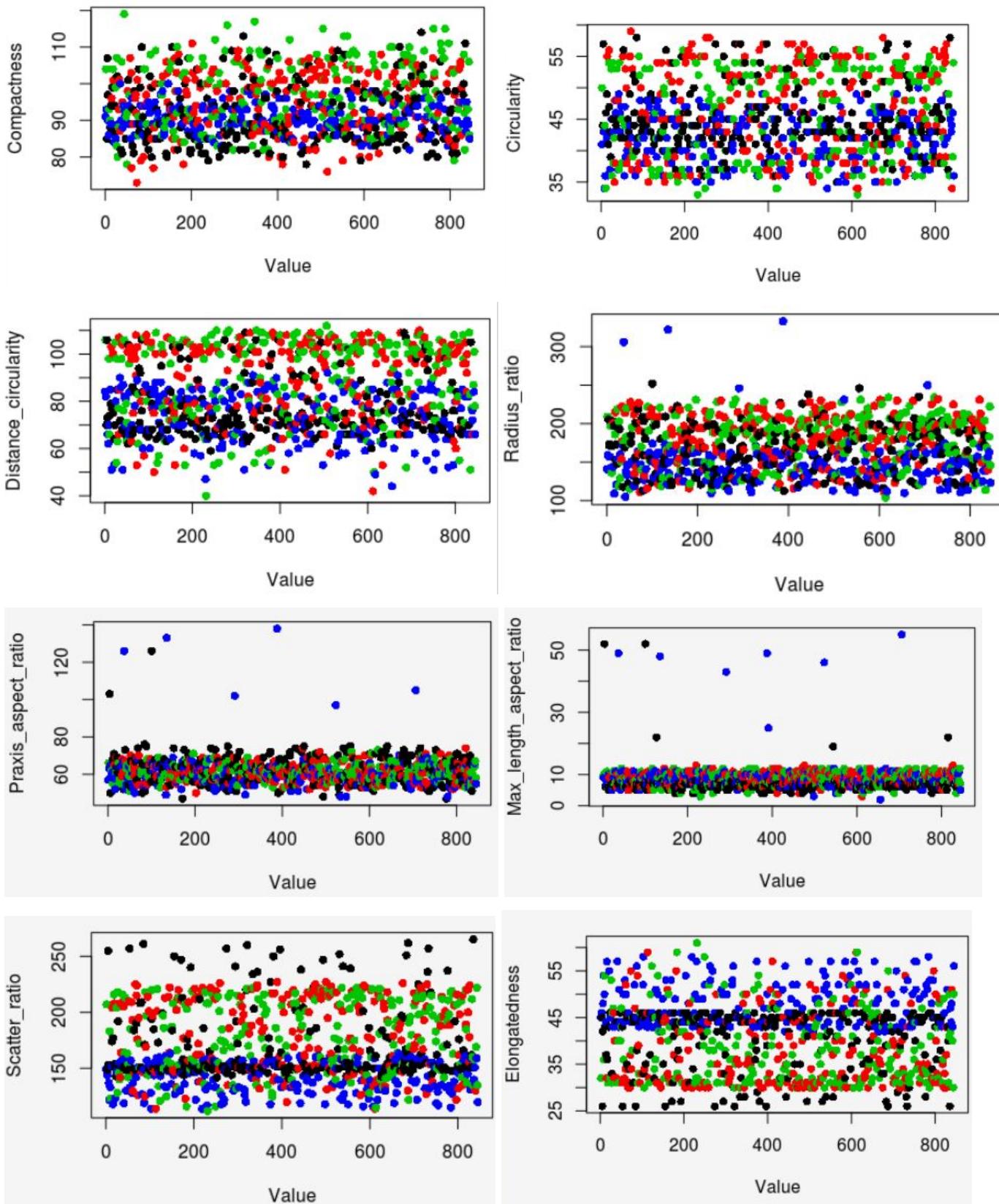
La variable que debemos predecir se llama “Class” y tiene 4 posibles valores, de los cuales tenemos una cantidad bastante equitativa de individuos.

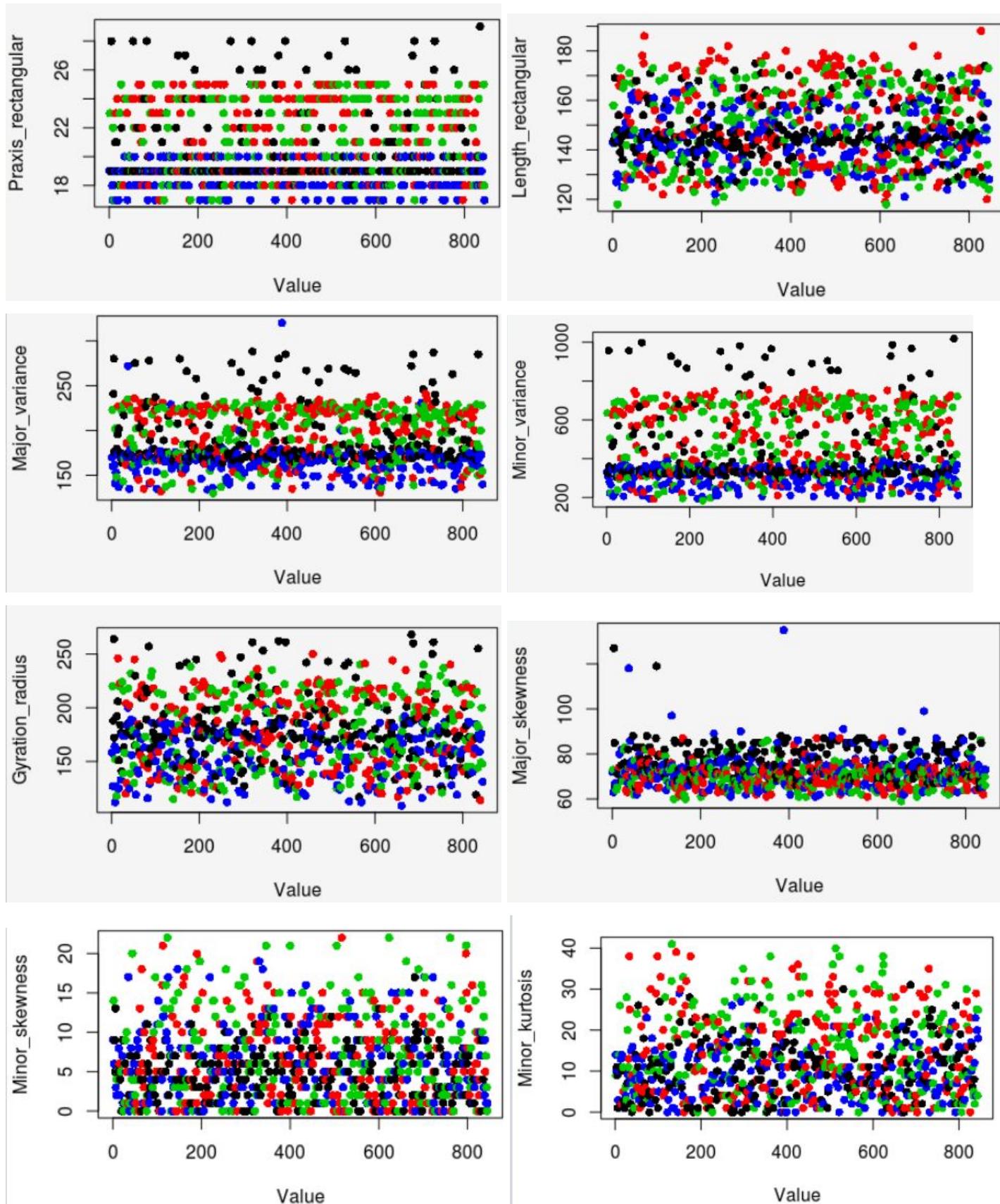
`round(prop.table(table(vehicle$Class)) * 100, digits = 1)`

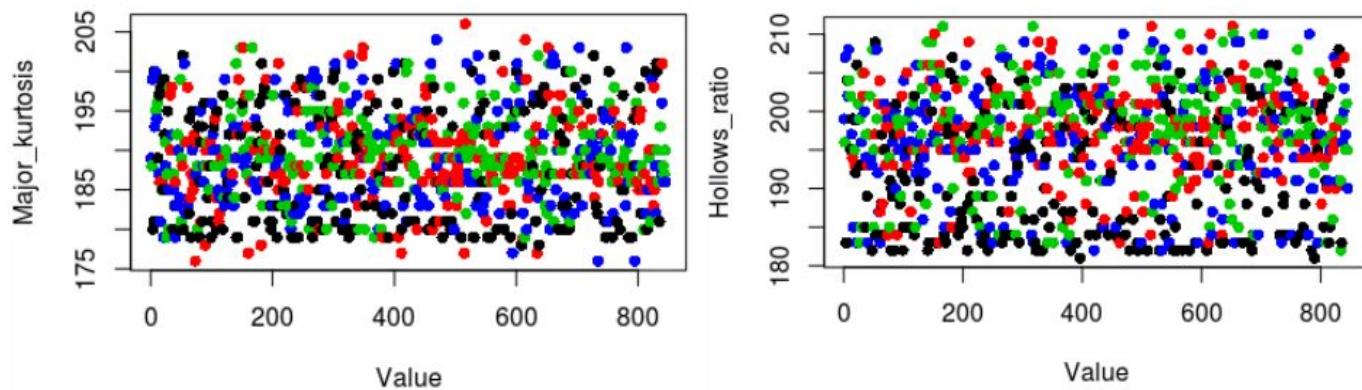
bus	opel	saab	van
25.8	25.1	25.7	23.4

## 2. Gráficos que permitan visualizar los datos adecuadamente.

Representación de variables individualmente por clases.

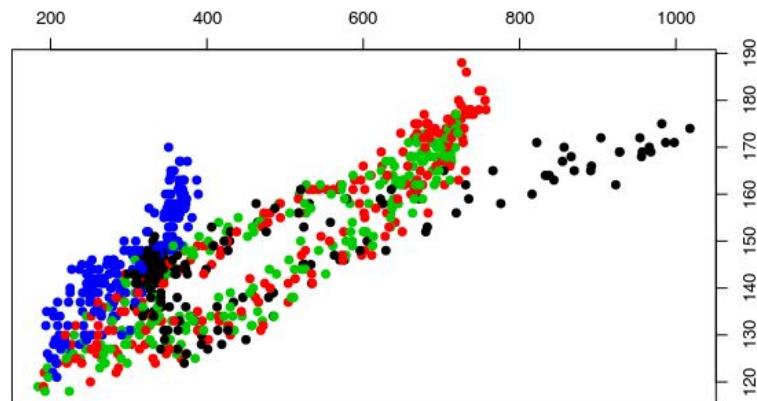




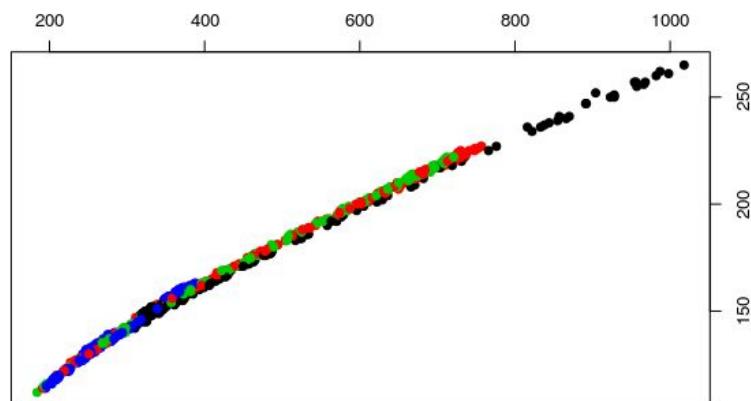


Representación algunas variables que tienen correlación (Solo mostraré algunas porque en mis datasets hay 19 variables y existen numerosas combinaciones).

```
pairs(vehicle$Scatter_ratio~vehicle$Minor_variance,col=vehicle[,19],pch=19)
```



```
pairs(vehicle$Length_rectangular~vehicle$Minor_variance,col=vehicle[,19],pch=19)
```



### 3. Descripción del conjunto de datos a partir de los puntos anteriores.

Podemos observar que muchas de las variables no nos aportan mucha información sobre los conjuntos ya que las clases aparecen repartidas de manera que no es posible apreciar clusters en ellas. Existen algunas excepciones, como puede ser el caso de Elongatedness o Praxis\_rectangular, donde podemos ver una frontera o umbral donde es más probable encontrar ciertas clases. En otras como Major\_kurtosis y Minor\_kurtosis es imposible apreciar a ojo una frontera que divida clases.

Si ejecutamos las siguientes instrucciones obtendremos la proporción en la que es posible encontrar una correlación entre algunas variables.

```
a = table(cor(vehicle[,1:18])>0.7)
#Se restan los 18 positivos de cada variable consigo misma y se divide a la mitad de la matriz
a["TRUE"] = (a["TRUE"] - 18)/2
a["FALSE"] = a["FALSE"]/2
print(prop.table(a))
```

	FALSE	TRUE
FALSE	0.745098	0.254902
TRUE		

## Regresión

1. Utilizar el algoritmo de regresión lineal simple sobre cada regresor(variable de entrada) para obtener los modelos correspondientes. Si el dataset asignado incluye más de 5 regresores, seleccione a su criterio los 5 que considere más relevantes. Una vez obtenidos los modelos, elegir el que considere más adecuado para su conjunto de datos según las medidas de calidad conocidas.

Para este primer apartado me guiaré por las representaciones sobre el conjunto de datos para seleccionar aquellas variables que tengan una distribución de sus valores más fácil de representar con una línea.

En concreto seleccionaré estas para elaborar una tabla con las principales variables estadísticas:

1. Y30CMortgageRate → Model1
2. M3RateSecondaryMarket → Model2
3. Y3CMaturityRate → Model3
4. Y5CMaturityRate → Model4
5. moneyStock → Model5

	<b>Model1</b>	<b>Model2</b>	<b>Model3</b>	<b>Model4</b>	<b>Model5</b>
<b>Estimate (Variable)</b>	1.062	1.132	1.1383	1.0982	0.9492
<b>Std Error</b>	0.007	0.049	0.0045	0.012	0.0030
<b>T Value</b>	149.155	-6.408	250.062	87.39	311.41
<b>Pr (&gt; T )</b>	< 2e-16	< 2.23e-10	< 2e-16	< 2e-16	< 2e-16
<b>Residual standard error</b>	0.7163	0.4688	0.4336	1.173	0.3492
<b>Adjusted R squared</b>	0.9551	0.9807	0.9835	0.8794	0.9893
<b>P-value</b>	< 2.2e-16				

Tabla 1: Variables más relevantes de los modelos entrenados

Observando la tabla 1 podemos concluir que el modelo 2,3 y 5 son mejores que el resto ya que su error residual es más bajo y su valor R ajustado es mayor que el resto. Entre ellas tres no quedaremos con el modelo de 5 ya que sus valores son los mejores con respecto a las otras. Puede ocurrir que estemos haciendo un sobreajuste con un R ajustado tan grande; se debería comprobar con un test MSE si realmente ese modelo está sobreajustado.

## 2. Utilizar el algoritmo para regresión lineal múltiple. Justificar adecuadamente si el modelo obtenido aporta mejoras respecto al modelo elegido en el paso anterior.

Para empezar podemos probar con todas las variables:

- `modelMult = lm(treasury$OneMonthCDRate~, data=treasury)`
- `print(summary(modelMult))`

Coefficients:	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	0.3908242	0.2922845	1.337	0.181473
Y1CMaturityRate	-0.0077989	0.0010165	-7.672	3.90e-14 ***
Y30CMortgageRate	0.1859336	0.0625621	2.972	0.003027 **
M3RateAuctionAverage	-0.0179197	0.0212799	-0.842	0.399930
M3RateSecondaryMarket	0.0469383	0.0384245	1.222	0.222148
V3CMaturityRate	0.2471282	0.0479041	5.159	2.98e-07 ***
Y5CMaturityRate	0.1598128	0.1132227	1.411	0.158401
bankCredit	-0.1449198	0.0771521	-1.878	0.060613 .
currency	0.0017825	0.0004689	3.801	0.000152 ***
demandDeposits	0.0194735	0.0103643	1.879	0.060538 .
federalFunds	0.0001163	0.0005899	0.197	0.843779
moneyStock	0.5809080	0.0158930	36.551	< 2e-16 ***
checkableDeposits	-0.0264670	0.0103129	-2.566	0.010416 *
loansLeases	0.0258287	0.0103871	2.487	0.013054 *
savingsDeposits	-0.0013973	0.0004365	-3.201	0.001411 **
tradeCurrencies	0.0005416	0.0001041	5.202	2.37e-07 ***
---				
Signif. codes:	0 ***	0.001 **	0.01 *	0.05 .
Residual standard error:	0.2377	on 1032 degrees of freedom		
Multiple R-squared:	0.9951	Adjusted R-squared:	0.995	
F-statistic:	1.403e+04	on 15 and 1032 DF,	p-value:	< 2.2e-16

Imagen 1: Ejecución “summary” de un modelo con todas las variables

En la imagen 1 podemos observar como este modelo tan simple ya supera el mejor del ejercicio anterior. Aun así, podemos ver como hay variables que son poco significativas para la predicción observando la columna Pr(>|T|) y los asteriscos de al lado de los números, indicando la escasa relevancia de una variable si hay pocos. El siguiente paso podría ser eliminar dichas variables.

Si quitamos las variables menos relevantes disminuimos muy poco R ajustada(0.001) pero es más legible el modelo obtenido y más interpretable.

`lm(treasury$OneMonthCDRate~.-federalFunds-M3RateAuctionAverage-M3RateSecondaryMarket-Y5CMaturityRate-loansLeases-demandDeposits, data=treasury)`

Coefficients:	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	4.770e-01	1.672e-01	2.853	0.00441 **
Y1CMaturityRate	-8.202e-03	9.628e-04	-8.519	< 2e-16 ***
Y30CMortgageRate	2.471e-01	3.271e-02	7.552	9.38e-14 ***
Y3CMaturityRate	2.938e-01	2.687e-02	10.936	< 2e-16 ***
bankCredit	-6.493e-02	2.060e-02	-3.152	0.00167 **
currency	-3.868e-04	1.537e-04	-2.517	0.01198 *
moneyStock	5.776e-01	1.378e-02	41.924	< 2e-16 ***
checkableDeposits	-4.104e-04	1.489e-04	-2.756	0.00595 **
savingsDeposits	3.971e-04	1.943e-04	2.043	0.04129 *
tradeCurrencies	7.673e-04	9.122e-05	8.411	< 2e-16 ***
---				
Signif. codes:	0 ***	0.001 **	0.01 *	0.05 .
Residual standard error:	0.2407	on 1038 degrees of freedom		
Multiple R-squared:	0.995	Adjusted R-squared:	0.9949	
F-statistic:	2.28e+04	on 9 and 1038 DF,	p-value:	< 2.2e-16

Imagen 2: Ejecución “summary” de un modelo quitando variables no relevantes

El siguiente paso sería observar aquellas variables que pueden tener una curva polinómica y intentar la no linealidad con ellas. Para ello probaré con 3 variables, aquellas con p-value más bajo en las etapas anteriores.

`modelMult = lm(`

```
treasury$OneMonthCDRate~treasury$tradeCurrencies+
treasury$moneyStock+I(treasury$moneyStock^2)+I(treasury$moneyStock^3)+
treasury$Y3CMaturityRate+I(treasury$Y3CMaturityRate^2),
data=treasury)
```

```
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) -4.357e-01 9.810e-02 -4.442 9.88e-06 ***
treasury$tradeCurrencies 2.536e-04 4.035e-05 6.286 4.79e-10 ***
treasury$moneyStock 9.082e-01 5.794e-02 15.675 < 2e-16 ***
I(treasury$moneyStock^2) -2.619e-02 4.728e-03 -5.540 3.84e-08 ***
I(treasury$moneyStock^3) 5.946e-04 1.270e-04 4.683 3.20e-06 ***
treasury$Y3CMaturityRate 2.017e-01 5.160e-02 3.910 9.83e-05 ***
I(treasury$Y3CMaturityRate^2) 1.524e-02 2.491e-03 6.118 1.34e-09 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.253 on 1041 degrees of freedom
Multiple R-squared: 0.9944, Adjusted R-squared: 0.9944
F-statistic: 3.094e+04 on 6 and 1041 DF, p-value: < 2.2e-16
```

Imagen 3: Ejecución “summary” de un modelo utilizando no linealidad

Observando la ejecución podemos concluir que el modelo obtenido no logra alcanzar al modelo previamente definido, sin embargo, si calculamos un modelo que combine todas las variables de 3 en 3 alcanzamos los mejores resultados en R ajustado y error residual.

`modelMult = lm(treasury$OneMonthCDRate~.^3, data=treasury)`

```
Residual standard error: 0.1095 on 474 degrees of freedom
Multiple R-squared: 0.9995, Adjusted R-squared: 0.999
F-statistic: 1740 on 573 and 474 DF, p-value: < 2.2e-16
```

Imagen 4: Ejecución “summary” de un modelo combinando todas las variables

Además, podemos sacar conclusiones de las combinaciones de las variables. Una de ellas es que normalmente están involucradas las mismas variables en combinaciones diferentes que tienen un alto nivel de significación, lo que sugiere la hipótesis de que dichas variables están altamente relacionadas con la salida. Aun así, es previsible que este último modelo tenga un sobreajuste muy alto por la cantidad de variables que tiene.

### 3. Aplicar el algoritmo k-NN para regresión no paramétrica.

Lo que haré en este ejercicio es mostrar una gráfica donde podremos observar como evoluciona el MSE en train y test según aumentamos los vecinos cercanos (parámetro k) de la función “knn”.

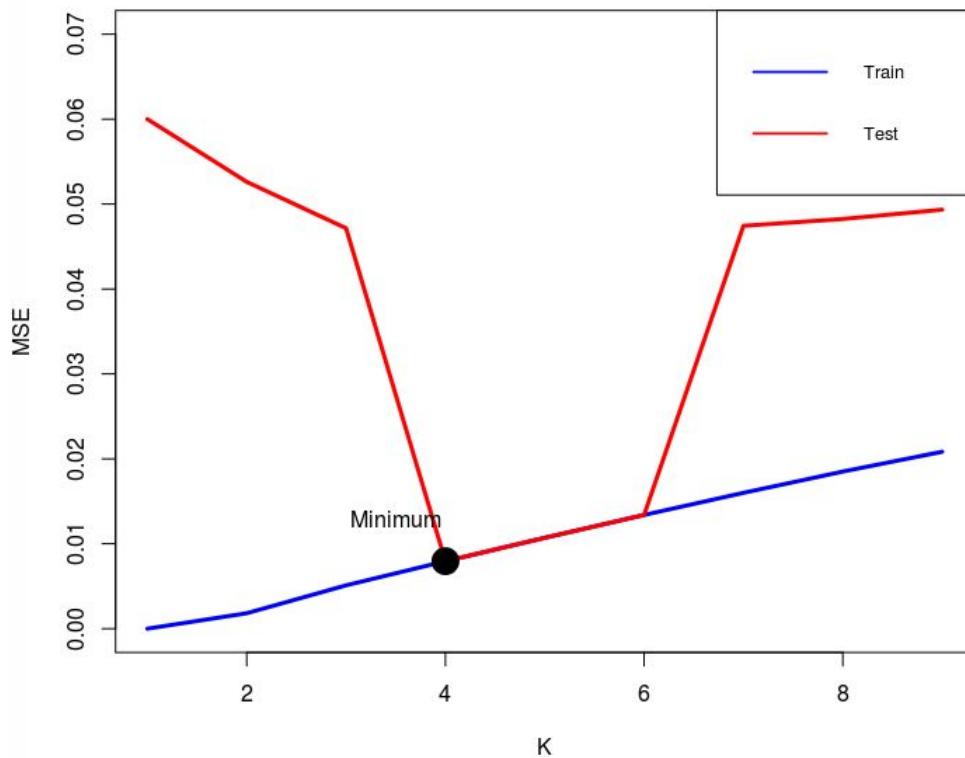


Figura 1: Representación de la evolución del algoritmo knn en train y test aumentando los vecinos.

En la figura 1 podemos observar cómo al ir aumentando los vecinos a tener en cuenta vamos mejorando el test hasta un cierto punto donde empezamos a perder precisión ( $k>6$ ) , por tanto un  $k=4$  es la mejor de las opciones para este dataset.

#### 4. Comparar los resultados de los dos algoritmos de regresión múltiple.

Para comparar primero realizaré un test sobre el mismo modelo ( $\hat{Y}$ ) en LM y KNN.

Tras ejecutar la validación cruzada en LM y KNN obtenemos los siguientes valores en train y test.

lmMSEtrain:	0.0552025
lmMSEtest:	0.06082046
kknnMSEtrain:	0.01599804
kknnMSEtest:	0.04743915

Imagen 5

Tras realizar el test de Wilcoxon sobre la tabla de tests observamos lo esperado, en treasury el KNN gana al LM.

```
LMvsKNNtst = wilcox.test(wilc_1_2[,1], wilc_1_2[,2], alternative = "two.sided", paired=TRUE)
Rmas = LMvsKNNtst$statistic
pvalue = LMvsKNNtst$p.value
LMvsKNNtst = wilcox.test(wilc_1_2[,2], wilc_1_2[,1], alternative = "two.sided", paired=TRUE)
Rmenos = LMvsKNNtst$statistic
cat("Rmas:",Rmas,"\n")
cat("Rmenos:",Rmenos,"\n")
cat("pvalue:",pvalue,"# No hay diferencias significativas entre ambos\n")
```

treasury	0.1000000	0.3200133
----------	-----------	-----------

Imagen 6: Fila normalizada para el test de Wilcoxon

Aun así, en el conjunto, observamos que no existen diferencias significativas entre aplicar LM y KNN porque el p-value está muy por encima de 0.05 usado normalmente.

Rmas: 78
Rmenos: 93
pvalue: 0.7660294 # No hay diferencias significativas entre ambos

Imagen 7: Test de Wilcoxon

Por último, compararemos estos algoritmos con un tercero, M5, introduciendo nuestros valores de test y train en tablas con más evaluaciones de los algoritmos sobre otros datasets.

Si a la comparación le añadimos el algoritmo M5 tendremos que usar una comparativa múltiple, con el test de Freedman. En la imagen 8 observamos que existen diferencias significativas ya que le p-value < 0.05. Si nos fijamos en la imagen 9 vemos como no existen diferencias significativas entre LM(1) y KNN(2) pero si son un poco más grandes con M5, sobre todo entre M5 y LM donde p-value < 0.1.

```
test_friedman = friedman.test(as.matrix(tablatst))
print(test_friedman)
tam = dim(tablatst)
groups = rep(1:tam[2], each=tam[1])
values = pairwise.wilcox.test(as.matrix(tablatst), groups, p.adjust = "holm", paired = TRUE)
print(values)
```

```
data: as.matrix(tablatst)
Friedman chi-squared = 8.4444, df = 2, p-value = 0.01467
```

Imagen 8: Test de Freedman

	1	2
2	0.580	-
3	0.081	0.108

Imagen 9: Comparativa múltiple entre LM(1), KNN(2) y M5(3)

## Clasificación

1. Utilizar el algoritmo k-NN probando con diferentes valores de k. Elegir el que considere más adecuado para su conjunto de datos.

Para comprobar el mejor K para nuestro problema haré un ejecucion con 100 diferentes.

Previamente, se elaboró un código que realiza un shuffle y una división en train y test en los datos, con el fin de distribuir de manera aleatoria las clases.

```
vehicle = vehicle[sample(nrow(vehicle),)]
lapply(vehicle[,1:18], normalize)
percentage = 80
cut = round(nrow(vehicle)*percentage/100)
vehicle_train = vehicle[1:cut, 1:18]
vehicle_test = vehicle[cut:nrow(vehicle), 1:18]
vehicle_train_labels = vehicle[1:cut, 19]
vehicle_test_labels = vehicle[cut:nrow(vehicle), 19]
```

Observando la imagen 10 podemos apreciar una tendencia negativa a medida que aumentamos el K, de manera que en la zona de valores más pequeña parece estar el mejor valor posible.

En la imagen 11 muestro dos ejecuciones para K<40 (las gráficas cambian debido a un “shuffle” en los datos). En ella parece que el valor de K que buscamos está sobre el 6 más o menos.

```
kNum = 1 # change to compare
tst = 1:kNum
for (i in 1:kNum){
  knnFit = train(vehicle_train, vehicle_train_labels, method="knn",
    preProc = c("center", "scale"),
    metric="Accuracy", tuneGrid = data.frame(.k=i),
    tuneLength = 10)
  knnPred = predict(knnFit, newdata = vehicle_test)
  a = postResample(pred = knnPred, obs = vehicle_test_labels)
  tst[i] = a["Accuracy"]
}
plot(tst, col="blue", type="l", xlab = "K", lwd=c(2.5), ylab = "Accuracy", xlim = c(1,kNum), ylim = c(0.5,0.8))
legend("topright",c("Test"), col = c("blue"), lwd=c(2.5))
```

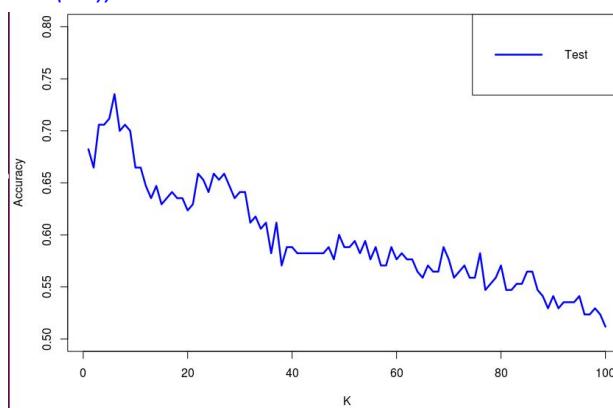


Imagen 10: Comparativa de múltiples valores de K

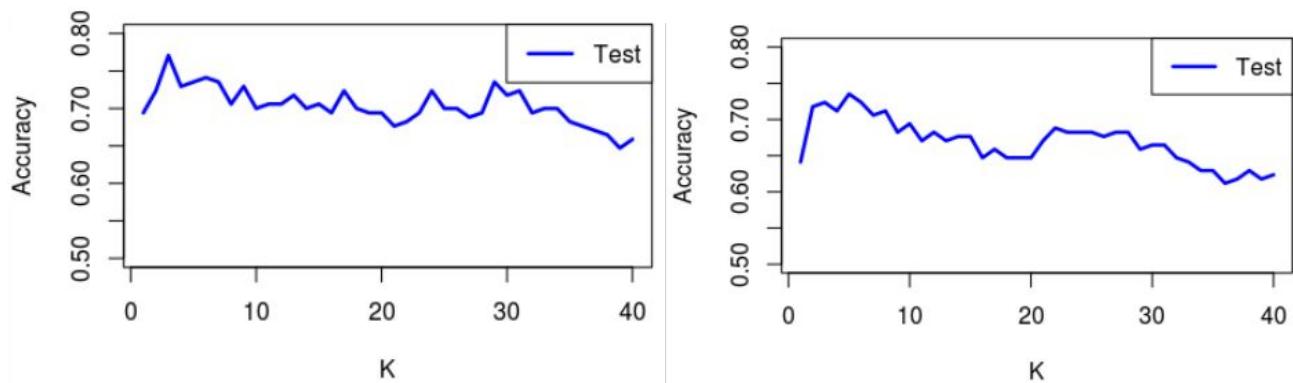


Imagen 11: Comparativa de múltiples valores de  $K < 40$

Por último, evaluamos el modelo con  $K=6$

```
knnFit = train(vehicle[,1:18], vehicle[,19], method="knn",
               preProc = c("center", "scale"),
               tuneLength = 10,
               trControl = trainControl(method = "cv"),
               tuneGrid = data.frame(.k=6))
print(confusionMatrix(knnFit))
```

		Reference			
Prediction	bus	opel	saab	van	
bus	24.7	0.2	0.9	1.2	
opel	0.1	11.8	7.2	0.6	
saab	0.1	11.0	15.7	0.6	
van	0.8	2.0	1.8	21.1	

Accuracy (average) : 0.7337					
-----------------------------	--	--	--	--	--

Imagen 12: Confusion matrix KNN ( $K = 6$ )

Finalmente, con Knn y  $K=6$ , obtenemos la predicciones de la imagen 11, donde destacan los errores cometidos al predecir saab cuando realmente era opel y viceversa. Esto puede significar que ambas clases son muy parecidas.

## 2. Utilizar el algoritmo LDA para clasificar.

Para utilizar el algoritmo LDA utilizaré la librería caret con un preprocesamiento donde el propio algoritmo escalará y centrará los datos para posteriormente aplicar una validación cruzada durante el entrenamiento. Podemos observar que la mayoría de errores aún se concentran en la pareja saab-opel a pesar que el Accuracy ha aumentado con respecto a knn y las predicciones han mejorado.

```
ldaFit = train(vehicle[,1:18], vehicle[,19],  
               method = "lda",  
               preProcess = c("center", "scale"),  
               tuneLength = 10,  
               trControl = trainControl(method = "cv"))  
print(confusionMatrix(ldaFit))
```

		Reference			
Prediction		bus	opel	saab	van
bus	bus	24.9	0.8	1.3	0.2
opel	opel	0.5	15.1	7.1	0.2
saab	saab	0.1	8.3	15.9	0.2
van	van	0.4	0.8	1.4	22.7

Accuracy (average) : 0.7858

Imagen 13: Confusion matrix LDA

## 3. Utilizar el algoritmo QDA para clasificar.

El algoritmo QDA lo podemos aplicar de la misma forma que el LDA usando caret, solamente debemos cambiar el method="qda" en el parámetro de la función train.

Con QDA observamos como aumenta considerablemente el Accuracy global mejorando de gran manera la predicción final del sistema.

```
qdaFit = train(vehicle[,1:18], vehicle[,19],  
               method = "qda",  
               preProcess = c("center", "scale"),  
               tuneLength = 10,  
               trControl = trainControl(method = "cv"))  
print(confusionMatrix(qdaFit))
```

		Reference			
Prediction		bus	opel	saab	van
bus	bus	25.4	0.0	0.2	0.1
opel	opel	0.0	18.9	6.9	0.2
saab	saab	0.0	5.1	17.9	0.1
van	van	0.4	1.1	0.7	23.0

Accuracy (average) : 0.8521

Imagen 14: Confusion matrix QDA

#### 4. Comparar los resultados de los tres algoritmos.

Para realizar una comparación múltiple debemos usar el test de Freedman, usando una tabla previamente cargada con los resultados de la evaluación de los tres algoritmos sobre otros datasets.

```
test_friedman = friedman.test(as.matrix(tablatra))
print(test_friedman)
```

```
Friedman chi-squared = 0.7, df = 2, p-value = 0.7047
```

Imagen 15: Test de Freedman

En la imagen 15 observamos un  $p\text{-value} > 0.05$ , por tanto, no existen diferencias significativas entre usar Knn, Lda y Qda sobre todos los datasets probados. Aun así, lo podemos comprobar finalmente visualizando con el método de Holm.

```
tam = dim(tablatra)
groups = rep(1:tam[2], each=tam[1])
values = pairwise.wilcox.test(as.matrix(tablatra), groups, p.adjust = "holm", paired = TRUE)
print(values)
```

	1	2
2	1.00	-
3	0.53	1.00

Imagen 16: Método de Holm. Knn(1), Lda(2) y Qda(3)

En la imagen 16 vemos como, efectivamente, no existen diferencias significativas entre ninguno de ellos.

## Apéndice

# Código Regresión en forma de script

# Hay líneas comentadas por pruebas que hice y otras para no mostrar gráficas y demás

```
rm(list=ls())
```

```
require(kknn)
```

```
treasury = read.csv("/home/manuelmontero/Escritorio/R Projects/treasury/treasury.dat", comment.char="@")
```

```
names(treasury) = c("Y1CMaturityRate", "Y30CMortgageRate", "M3RateAuctionAverage",
  "M3RateSecondaryMarket", "Y3CMaturityRate", "Y5CMaturityRate",
  "bankCredit", "currency", "demandDeposits", "federalFunds",
  "moneyStock", "checkableDeposits", "loansLeases", "savingsDeposits",
  "tradeCurrencies", "OneMonthCDRate")
```

# # Visualización global de los datos

```
# temp = treasury
```

```
# plotY = function (x,y) {
```

```
# plot(temp[,y]~temp[,x], xlab=paste(names(temp)[x], " X", x, sep=""),
```

```
# ylab=names(temp)[y])
```

```
# }
```

```
#
```

```
# par(mfrow=c(2,2))
```

```
# x = sapply(1:(dim(temp)[2]-1), plotY, dim(temp)[2])
```

```
# par(mfrow=c(1,1))
```

```
# hist(treasury$OneMonthCDRate, xlab = "OneMonthCDRate", main = "Histogram of OneMonthCDRate")
```

# Calculo de Modelos (Pruebas)

```
# model1 = lm(treasury$OneMonthCDRate~treasury$Y30CMortgageRate)
```

```
# print(summary(model1))
```

```
# model2 = lm(treasury$OneMonthCDRate~treasury$M3RateSecondaryMarket)
```

```
# print(summary(model2))
```

```
# model3 = lm(treasury$OneMonthCDRate~treasury$Y3CMaturityRate)
```

```
# print(summary(model3))
```

```
# model4 = lm(treasury$OneMonthCDRate~treasury$Y5CMaturityRate)
```

```
# print(summary(model4))
```

```
# model5 = lm(treasury$OneMonthCDRate~treasury$moneyStock)
```

```
# print(summary(model5))
```

```
# modelMult = lm(treasury$OneMonthCDRate~
```

```
# -federalFunds-M3RateAuctionAverage-M3RateSecondaryMarket-Y5CMaturityRate-loansLeases-demandDeposits,
# data=treasury)
```

```

# print(summary(modelMult))

# modelMult = lm(treasury$OneMonthCDRate~.^3, data=treasury)
# print(summary(modelMult))

name = "/home/manuelmontero/Escritorio/R Projects/treasury/treasury"
run_lm_fold = function(i, x, tt = "test", knn = FALSE) {
  file = paste(x, "-5-", i, "tra.dat", sep="")
  x_tra = read.csv(file, comment.char="@")
  file = paste(x, "-5-", i, "tst.dat", sep="")
  x_tst = read.csv(file, comment.char="@")
  ln = length(names(x_tra)) - 1
  names(x_tra)[1:ln] = paste ("X", 1:ln, sep="")
  names(x_tra)[ln+1] = "Y"
  names(x_tst)[1:ln] = paste ("X", 1:ln, sep="")
  names(x_tst)[ln+1] = "Y"

  if (tt == "train") {test = x_tra}
  else {test = x_tst}
  if(knn) {
    fitMulti = kknn(Y~,x_tra,test)
    yprime = fitMulti$fitted.values
  }
  else {
    fitMulti = lm(Y~,x_tra)
    # fitMulti = lm(Y~.^3,x_tra)
    # fitMulti = lm(Y~.-X3-X4-X6-X8-X12,x_tra)
    # fitMulti = lm(Y~X14+X10+I(X10^2)+I(X10^3)+X5+I(X5^2),x_tra)
    yprime = predict(fitMulti,test)
  }
  sum(abs(test$Y-yprime)^2)/length(yprime)
}

lmMSEtrain = mean(sapply(1:5,run_lm_fold,name,"train"))
lmMSEtest = mean(sapply(1:5,run_lm_fold,name,"test"))
cat("lmMSEtrain: ",lmMSEtrain,"\n")
cat("lmMSEtest: ",lmMSEtest,"\n")

kknnMSEtrain = mean(sapply(1:5,run_lm_fold,name,"train",knn=TRUE))
kknnMSEtest = mean(sapply(1:5,run_lm_fold,name,"test",knn=TRUE))
cat("kknnMSEtrain: ",kknnMSEtrain,"\n")
cat("kknnMSEtest: ",kknnMSEtest,"\n")

#leemos la tabla con los errores medios de test
resultados = read.csv("/home/manuelmontero/Escritorio/R Projects/TablasAlumnos/regr_test_alumnos.csv")

```

Manuel Montero Santana  
 Introducción a la ciencia de datos  
 Universidad de Granada

```

tablatst = cbind(resultados[,2:dim(resultados)[2]])
colnames(tablatst) = names(resultados)[2:dim(resultados)[2]]
rownames(tablatst) = resultados[,1]
tablatst[16,1]=lmMSEtest
tablatst[16,2]=kknnMSEtest

#leemos la tabla con los errores medios de train
resultados = read.csv("/home/manuelmontero/Escritorio/R Projects/TablasAlumnos/regr_train_alumnos.csv")
tablatra = cbind(resultados[,2:dim(resultados)[2]])
colnames(tablatra) = names(resultados)[2:dim(resultados)[2]]
rownames(tablatra) = resultados[,1]
tablatra[16,1]=lmMSEtrain
tablatra[16,2]=kknnMSEtrain

# (Wilcoxon's test)
##lm (other) vs knn (ref)
# + 0.1 porque wilcox R falla para valores == 0 en la tabla
difs = (tablatast[,1] - tablatst[,2]) / tablatst[,1]
wilc_1_2 = cbind(ifelse (difs<0, abs(difs)+0.1, 0.1), ifelse (difs>0, abs(difs)+0.1, 0.1))
colnames(wilc_1_2) = c(colnames(tablatst)[1], colnames(tablatst)[2])
rownames(wilc_1_2) = c(rownames(tablatst))
print(wilc_1_2)

# Se aplica el test y se interpretan los resultados
LMvsKNNtst = wilcox.test(wilc_1_2[,1], wilc_1_2[,2], alternative = "two.sided", paired=TRUE)
Rmas = LMvsKNNtst$statistic
pvalue = LMvsKNNtst$p.value
LMvsKNNtst = wilcox.test(wilc_1_2[,2], wilc_1_2[,1], alternative = "two.sided", paired=TRUE)
Rmenos = LMvsKNNtst$statistic
cat("Rmas:",Rmas,"\\n")
cat("Rmenos:",Rmenos,"\\n")
cat("pvalue:",pvalue,"# No hay diferencias significativas entre ambos\\n")

# Comparativas Múltiples – Usaremos Friedman y como post-
# # hoc Holm (los rankings se calculan por posiciones de los
# # algoritmos y hace falta normalización)
test_friedman = friedman.test(as.matrix(tablatst))
print(test_friedman)
tam = dim(tablatst)
groups = rep(1:tam[2], each=tam[1])
values = pairwise.wilcox.test(as.matrix(tablatst), groups, p.adjust = "holm", paired = TRUE)
print(values)

```

## # Código Clasificación en forma de script

# Hay líneas comentadas por pruebas que hice y otras para no mostrar gráficas y demás

```
rm(list=ls())
require(caret)
library(MASS)
library(ISLR)
library(klaR)
library(class)

normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

vehicle = read.csv("/home/manuelmontero/Escritorio/R Projects/vehicle/vehicle.dat", comment.char="@")

names(vehicle) = c("Compactness", "Circularity", "Distance_circularity",
  "Radius_ratio", "Praxis_aspect_ratio", "Max_length_aspect_ratio",
  "Scatter_ratio", "Elongatedness", "Praxis_rectangular", "Length_rectangular",
  "Major_variance", "Minor_variance", "Gyration_radius", "Major_skewness",
  "Minor_skewness", "Minor_kurtosis", "Major_kurtosis", "Hollows_ratio", "Class")

# Algunos comandos de visualización de datos
# print(round(prop.table(table(vehicle$Class)) * 100, digits = 1))
# plot(vehicle[,1:18], col=vehicle[,19], xlab = "Value", ylab = "Hollows_ratio", pch = 16)
# print(prop.table(table(cor(vehicle[,1:18])>0.7)))
a = table(cor(vehicle[,1:18])>0.7)
a["TRUE"] = (a["TRUE"] - 18)/2
a["FALSE"] = a["FALSE"]/2
prop.table(a)

# Dividir datos en train y test, primero desordenamos y cogemos el 80% train y 20% test
vehicle = vehicle[sample(nrow(vehicle)),]
lapply(vehicle[,1:18], normalize)
percentage = 80
cut = round(nrow(vehicle)*percentage/100)
vehicle_train = vehicle[1:cut, 1:18]
vehicle_test = vehicle[cut:nrow(vehicle), 1:18]
vehicle_train_labels = vehicle[1:cut, 19]
vehicle_test_labels = vehicle[cut:nrow(vehicle), 19]

# Utilizar el algoritmo k-NN probando con diferentes valores de k.
# Elegir el que considere más adecuado para su conjunto de datos.
# el siguiente bucle prueba con el numero k que se ponga en
# Knum y guarda el resultado de un test en vector para representarlo al final
# kNum = 1 # change to compare
# tst = 1:kNum
```

Manuel Montero Santana

Introducción a la ciencia de datos

Universidad de Granada

```

# for (i in 1:kNum){
#   knnFit = train(vehicle_train, vehicle_train_labels, method="knn",
#                 preProc = c("center", "scale"),
#                 metric="Accuracy", tuneGrid = data.frame(.k=i),
#                 tuneLength = 10)
#   knnPred = predict(knnFit, newdata = vehicle_test)
#   a = postResample(pred = knnPred, obs = vehicle_test_labels)
#   tst[i] = a["Accuracy"]
# }
# plot(tst, col="blue", type="l", xlab = "K", lwd=c(2.5), ylab = "Accuracy", xlim = c(1,kNum), ylim = c(0.5,0.8))
# legend("topright",c("Test"), col = c("blue"), lwd=c(2.5))

## Utilizar el algoritmo KNN para clasificar.
knnFit = train(vehicle[,1:18], vehicle[,19], method="knn",
               preProc = c("center", "scale"),
               tuneLength = 10,
               trControl = trainControl(method = "cv"),
               tuneGrid = data.frame(.k=6))
print(confusionMatrix(knnFit))

## Utilizar el algoritmo LDA para clasificar.
ldaFit = train(vehicle[,1:18], vehicle[,19],
               method = "lda",
               preProcess = c("center", "scale"),
               tuneLength = 10,
               trControl = trainControl(method = "cv"))
print(confusionMatrix(ldaFit))

# Utilizar el algoritmo QDA para clasificar.
qdaFit = train(vehicle[,1:18], vehicle[,19], method = "qda",
               preProcess = c("center", "scale"),
               tuneLength = 10,
               trControl = trainControl(method = "cv"))
print(confusionMatrix(qdaFit))

# Comparar los resultados de los tres algoritmos.
resultados = read.csv("/home/manuelmontero/Escritorio/R Projects/TablasAlumnos/clasif_test_alumnos.csv")
tablatra = cbind(resultados[,2:dim(resultados)[2]])
colnames(tablatra) = names(resultados)[2:dim(resultados)[2]]
rownames(tablatra) = resultados[,1]
tablatra[17,1] = knnFit$results$Accuracy
tablatra[17,2] = ldaFit$results$Accuracy
tablatra[17,3] = qdaFit$results$Accuracy

# • Friedman. Using Holm see a winning algorithm
test_friedman = friedman.test(as.matrix(tablatra))

```

```
print(test_friedman)

tam = dim(tablatra)
groups = rep(1:tam[2], each=tam[1])
values = pairwise.wilcox.test(as.matrix(tablatra), groups, p.adjust = "holm", paired = TRUE)
print(values)
```