# Práctica 1
# Regresión lineal

Mario Emilio Jiménez Vizcaíno
A01173359@itesm.mx
Tecnológico de Monterrey
Ingeniería en Tecnologías Computacionales
Monterrey, N.L., México

Jesus Abraham Haros Madrid
A01252642@itesm.mx
Tecnológico de Monterrey
Ingeniería en Tecnologías Computacionales
Monterrey, N.L., México

## ABSTRACT

Esta práctica tiene el propósito de demostrar dos metodologías para la creación modelos de regresión lineal, cuyo objetivo es crear una función a partir de una combinación lineal de sus parámetros para aproximar el resultado real. Estos métodos son la regresión lineal por mínimos cuadrados y el gradiente descendente, ambos implementados dentro de la librería scikit-learn, usada durante la práctica.

## 1 INTRODUCCIÓN

El aprendizaje automático es la rama de los algoritmos computacionales cuyo objetivo es emular la inteligencia humana a través de ”aprender” del entorno. Las técnicas basadas en el aprendizaje automático se han aplicado con éxito en diversos campos como el reconocimiento de patrones, la visión computacional, diferentes ramas de la ingeniería, las finanzas y la biología.[1]

Uno de los modelos más comunes del aprendizaje automático es la regresión, un tipo de análisis predictivo básico y comúnmente utilizado. Se diferencia de los modelos de clasificación porque estima un valor numérico, mientras que los modelos de clasificación identifican a qué categoría pertenece una observación.

Las estimaciones de regresión se utilizan para explicar la relación entre una variable dependiente y una o más variables independientes. La forma más sencilla de de este modelo es la ecuación de regresión con una variable dependiente y otra independiente, definida por la fórmula $y = c + b * x$ donde $y$ representa el valor estimado de la variable dependiente, $c$ es una constante, $b$ es el coeficiente de regresión y $x$ es el valor de la variable independiente.

## 2 CONCEPTOS PREVIOS

- Programación básica en los lenguajes R y Python
- Conocimiento de las librerías scikit-learn, pandas y numpy
- Conocimientos de estadística y de regresión lineal

## 3 METODOLOGÍA

Para la demostración de los modelos de regresión se utilizaron las implementaciones de estos algoritmos de scikit-learn, específicamente las clases *LinearRegression* y *SGDRegressor*, probadas en dos datasets diferentes: genero.txt y mtcars.txt.

### 3.1 Dataset GENERO

Este dataset está compuesto por 10,000 filas de información, que representan la información del género, el peso y la estatura de una persona. El género está representado por una cadena de caracteres ”*Male*” o ”*Female*”, el peso por un número decimal de libras, y la estatura por un número decimal de pulgadas.

#### *3.1.1 Análisis exploratorio.*

Inicialmente se utilizó R para hacer un análisis estadístico de este dataset, específicamente las columnas *Weight* y *Height*, usadas como variable independiente y dependiente respectivamente. La función *summary* de R arrojó el siguiente resultado:

| Gender | Height | Weight |
|---|---|---|
| Length: 10000 | Min: 54.26 | Min: 64.7 |
| Class: character | 1st Qu: 63.51 | 1st Qu: 135.8 |
| Mode: character | Median: 66.32 | Median: 161.2 |
| | Mean: 66.37 | Mean: 161.4 |
| | 3rd Qu: 69.17 | 3rd Qu: 187.2 |
| | Max: 79.00 | Max: 270.0 |

También las funciones *boxplot* y *plot* fueron utilizadas para crear dos gráficas que nos dan una idea general sobre cómo se comportan nuestros datos. La figura 1 representa dos boxplots de las variables de GENERO que usaremos.

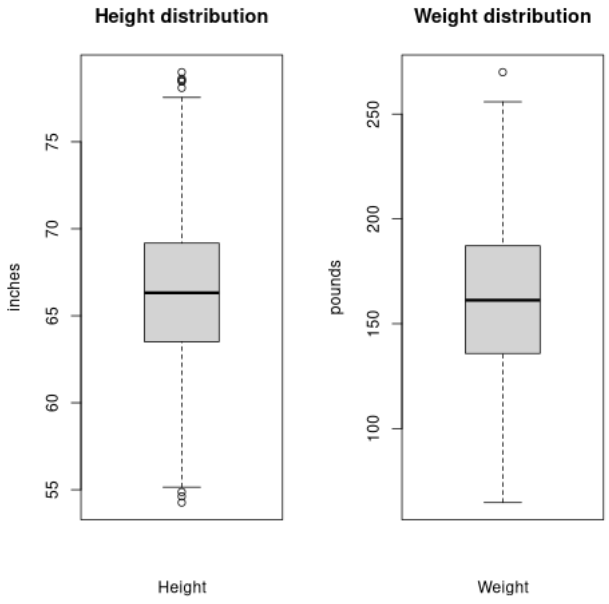



**Figure 1: Boxplot del dataset GENERO**

Por otro lado, la figura 2 representa los diversos puntos de información graficados en un plano con el peso en el eje horizontal y la altura en el eje vertical.

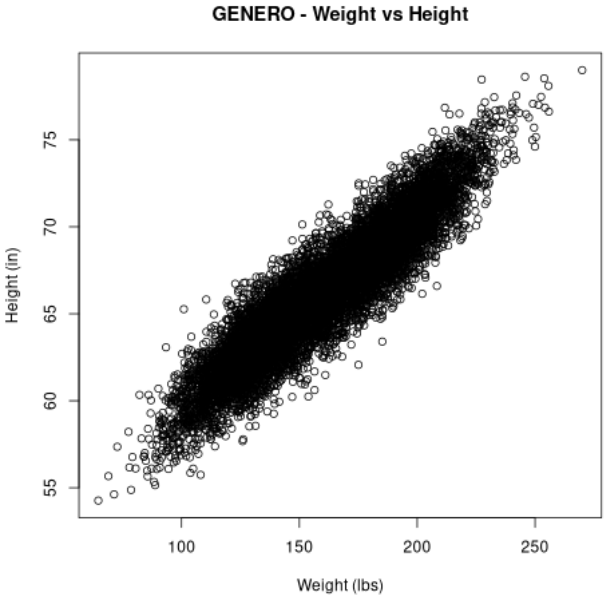



**Figure 2: Gráfico de dispersión del dataset GENERO**

El código fuente de este análisis puede ser encontrado en el apéndice C.

*3.1.2 Regresión lineal.*
Como se mencionó anteriormente, se utilizó la implementación del algorimo de regresión lineal de *scikit-learn*, encontrado en la clase *sklearn.linear_model.LinearRegression*. El código para este ejemplo es simple:

(1) Usando la librería *pandas* se carga el archivo *genero.txt* a un *Dataframe* usando la función *read_csv*
(2) Se extraen las columnas de altura y peso del *Dataframe*
(3) De la librería *sklearn*, se utiliza la función *train_test_split* para dividir tanto la columna de altura y de peso en dos partes cada una: la parte que se usará para entrenar al modelo, con el 80% de los datos, y la parte que se usará para probar el modelo, con el restante 20%
(4) Se instancia la clase *LinearRegression* y se utiliza el método *fit* para asignarle la columna de datos independiente (en este caso la altura) y la columna de datos dependiente (el peso)
(5) Se llama al método *predict* de la instancia del modelo de regresión con los datos de prueba de la variable independiente, lo que nos devuelve una columna de datos predecidos
(6) Finalmente, se comparan estos datos predecidos con los datos reales (los datos de prueba de la variable dependiente) para sacar un error cuadrático medio, que suele rondar entre 140 y 155

En este caso se decidió normalizar la variable independiente (Height) para que fuera fácilmente comparable con la implementación de gradiente descendente. El código usado en para demostrar la regresión lineal puedes ser encontrado en el apéndice D.

*3.1.3 Gradiente descendente.*
Para la demostración de este algorimo se optó por una implementación propia, que exponga las mismas funciones básicas que la clase de *sklearn* pero con un mecanismo interno diferente. La clase de *GradientDescent* se compone por 3 funciones principales:

- El constructor, que inicializa variables como la velocidad de aprendizaje, el número máximo de iteraciones y la precisión final.
- La función *fit*, que recibe dos *Dataframe*s, uno de las variables independientes y otro de la dependiente. Este método ejecuta las siguientes instrucciones:
  (1) Primero normaliza cada variable independiente, esto con el fin de que el algoritmo tenga un mejor funcionamiento. Normalizar las variables se trata de encontrar el promedio (*mu*) y la desviación estándar (*sigma*) de los datos, para así primero restarle el promedio y dividirlos entre la desviación, con el fin de que los datos se comporten como una distribución normal
  (2) Después se añade una columna de 1's para que en la función de salida tenga una constante no multiplicada por una variable independiente
  (3) Se crea la variable de los coeficientes e intercepto *theta*, inicializándola con 0's
  (4) Después, por cada iteración:
    (a) Se genera una nueva predicción haciendo el producto punto entre las variables independientes y la variable de coeficientes e intercepto
    (b) A la predicción se le resta la variable dependiente para obtener los errores de la predicción
    (c) Se multiplica la velocidad de aprendizaje, entre el número de filas en las variables independientes, por el producto punto de las variables independientes por los errores
    (d) Se resta este resultado a *theta*
    (e) Se calcula el "costo" de esta predicción usando el error cuadrático medio
    (f) Se compara el costo con el costo de la iteración previa, y si la diferencia es menor a la precisión del algoritmo, este se detiene
- La función *predict*, que:
  (1) Recibe variables independientes $x$
  (2) Las normaliza con los mismos parámetros (promedio y desviación estándar) con los que fueron normalizados los originales
  (3) Añade la columna de 1's
  (4) Multiplica estos datos por *theta* usando el producto punto, para así obtener la predicción de la variable dependiente

El código fuente de esta clase se encuentra en el apéndice F y está basado en el artículo de Gunjal, S.[2].

Finalmente, gracias a que esta clase expone las mismas funciones que *LinearRegression* de *sklearn*, se pueden sustituir fácilmente, por lo que el código de esta demostración es muy parecido al de la regresión lineal, lo que cambian son los resultados, que no son conclusivos ya que la selección de datos de entrenamiento y pruebas es

aleatoria. El código fuente de este apartado puede ser encontrado en el apéndice G.

## 3.2 Dataset MTCARS

Este segundo dataset se compone de 32 filas, cada uno con una variedad de datos sobre el rendimiento y las características del motor de diferentes modelos de automóviles. En esta demostración se utilizarán solamente las columnas de cilindrada ("*disp*"), peso ("*wt*") y caballos de fuerza ("*hp*"); las dos primeras como variables independientes y la tercera como variable dependiente o de salida.

### *3.2.1 Análisis exploratorio.*

Al igual que con el dataset GENERO, se realizó un análisis estadístico inicial con R. La función *summary* arrojó los siguientes resultados sobre las columnas que nos conciernen en este ejemplo:

| disp | hp | wt |
|---|---|---|
| Min: 71.1 | Min: 52.0 | Min: 1.513 |
| 1st Qu: 120.8 | 1st Qu: 96.5 | 1st Qu: 2.581 |
| Median: 196.3 | Median: 123.0 | Median: 3.325 |
| Mean: 230.7 | Mean: 146.7 | Mean: 3.217 |
| 3rd Qu: 326.0 | 3rd Qu: 180.0 | 3rd Qu: 3.610 |
| Max: 472.0 | Max: 335.0 | Max: 5.424 |

Para este dataset se usaron las funciones *boxplot* y *pairs* para crear dos tipos de gráficos con los datos: La figura 3 incluye tres boxplots de las variables que incluiremos en la demostración: la cilindrada, el peso y los caballos de fuerza del motor.

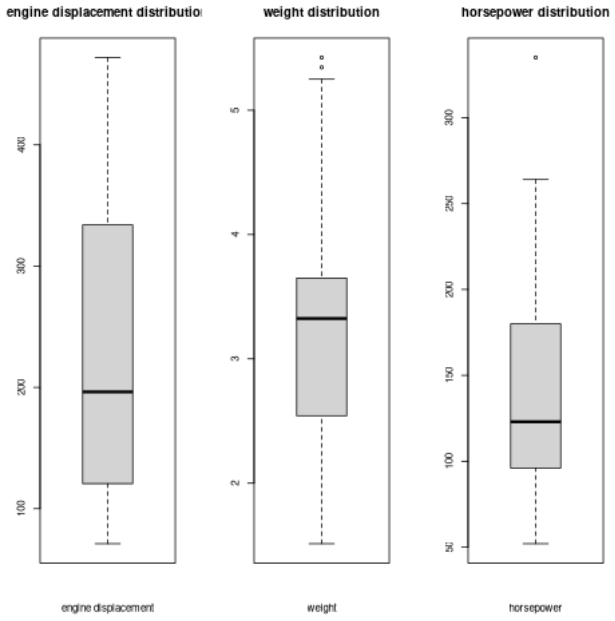



**Figure 3: Boxplot del dataset MTCARS**

En segundo lugar está la figura 4, que incluye gráficas 6 gráficas de dispersión, comparando las 3 variables con cada combinación posible. De izquierda a derecha, de arriba hacia abajo:

(1) Caballos de fuerza vs cilindrada
(2) Peso vs cilindrada
(3) Cilindrada vs caballos de fuerza
(4) Peso vs caballos de fuerza
(5) Cilindrada vs peso
(6) Caballos de fuerza vs peso

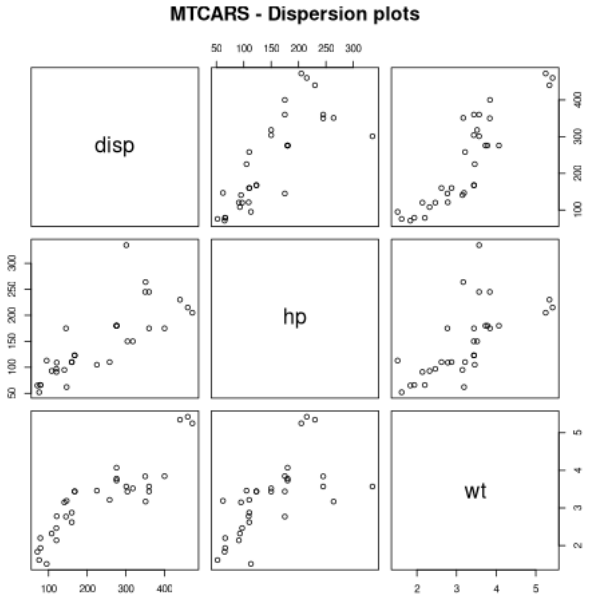



**Figure 4: Gráficos de dispersión del dataset MTCARS**

El código fuente de este análisis puede ser encontrado en el apéndice C.

### *3.2.2 Regresión lineal.*

Como consecuencia de que este dataset incluye muy pocas instancias o filas, se utilizó la metodología de validación *n-fold cross*, en la que se divide el dataset en pequeños conjuntos, y así se escoje uno de estos para realizar las pruebas y los demás dataset para realizar el entrenamiento. Esta selección de conjunto de elementos rota entre todo el dataset para así poder escoger el conjunto que al ser probado tenga el menor error. Para realizar estas iteraciones entre los conjuntos se utilizó la clase *KFold* del paquete *sklearn.model_selection*. Para este ejemplo el código sigue los pasos:

(1) Primero se lee el dataset utilizando *pandas.read_csv* y se obtiene un *Dataframe*
(2) Se crea el modelo de validación con *sklearn.model_selection.KFold* y se iteran sobre los conjuntos de datos de entrenamiento y pruebas. Por cada uno de ellos:
   (a) Se separan las columnas de variables independientes ("*disp*" y "*wt*") y dependientes ("*hp*") de los datos de entrenamiento
   (b) Se crea un modelo de regresión lineal con la clase *sklearn.linear_model.LinearRegression*[3].
   (c) Se entrena el modelo con los datos de entrenamiento (independientes y dependientes)

(d) Se extraen las columnas de los datos de prueba
(e) Se predice una serie de datos usando las variables independientes de los datos de prueba
(f) Se comparan los datos predecidos con la variable dependiente de los datos de prueba, mediante la fórmula del error cuadrático medio
(g) Se añade el valor del error a una lista
(3) Finalmente se promedian los errores para obtener un error promedio

Por defecto, la implementación de *scikit-learn* de *KFold* divide el dataset en 5 partes[3], por lo que se obtienen 5 errores cuadráticos medios y coeficientes de determinación:

(1) 1084.202, 0.583
(2) 489.059, 0.742
(3) 1113.586, 0.818
(4) 1739.342, 0.463
(5) 7259.875, 0.102

El promedio de estos errores es de un error cuadrático medio de 2337.21 y un coeficiente de determinación de 0.542.

El código fuente de este análisis puede ser encontrado en el apéndice E.

#### 3.2.3 *Gradiente descendente.*

Al igual que en el apartado de gradiente descendente del anterior dataset, se utilizó la misma implementación de gradiente descendente, y como consecuencia de que exporta los mismos métodos que *LinearRegression*, el código es muy parecido. Como resultados obtuvimos 5 errores cuadráticos medios y 5 coeficientes de determinación:

(1) 1050.437 ,0.597
(2) 447.73, 0.764
(3) 1124.836, 0.816
(4) 1570.729, 0.515
(5) 7431.13, 0.082

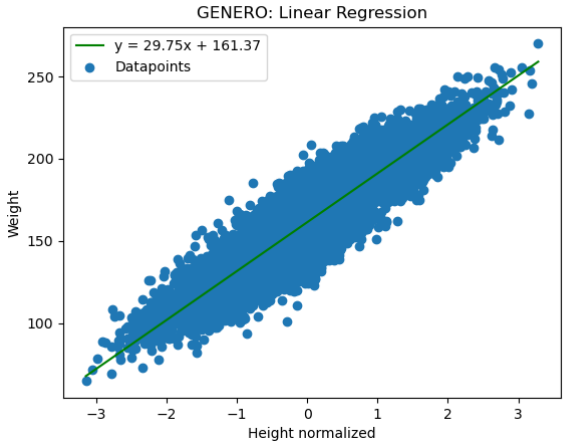
El promedio de estos es 2324.97 para el error cuadrático medio y 0.555 para el coeficiente de determinación.

El código fuente de este análisis puede ser encontrado en el apéndice H.

## 4 RESULTADOS

### 4.1 Dataset GENERO

Se elaboró un análisis exploratorio para las 10,000 filas de información del dataset; al observar el resumen de los datos hechos con R se puede apreciar que el comportamiento de los datos es muy semejante a una distribución normal pero agregando algunos outliers, es por esto que el boxplot tiene dicha figura centrada y balanceada; al graficar podemos apreciar que la distribución de los puntos es muy concentrada en el centro, dispersándose poco a poco a los extremos, parecido a la ¨campana¨ que se forma al graficar una distribución normal. Al elaborar la regresión lineal se utilizó la metodología de validación simple ya que el dataset era grande, y al evaluar el método nos indica que su **coeficiente de determinación es de 0.85** y **su error cuadrático medio (MSE) de 146**. Al graficar la regresión lineal se aprecia que la linea sigue la tendencia de los datos, concordando con el coeficiente de determinación mencionado anteriormente.

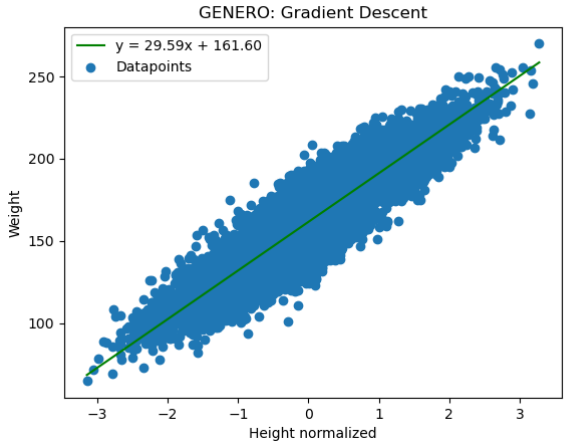



### 4.2 Dataset MTCARS

Se elaboró un análisis exploratorio para las 32 filas de información del dataset; de primera instancia se aprecia que el dataset no es muy grande, por lo que hacer un modelo de regresión lineal o de predicción resultará en algo muy alejado de la realidad. Al comenzar a explorarlo en R también observamos como se comportan los datos, los diagramas de caja muestran distribuciones diferentes para cada variable, al graficar los datos también quedan demasiado dispersos y no es posible apreciar una tendencia. Cuando se hizo la regresión lineal se utilizó la metodología de n-fold cross validation ya que el dataset era pequeño, el evaluar el método nos indica que **coeficiente de determinación promedio es de 0.542** y su **error cuadrático medio (MSE) promedio de 2337.21**, como era de esperarse, muy alejados de un resultado óptimo. Al graficarlo se muestra una tendencia entre los puntos, pero la dispersión es grande que la correlación termina siendo pequeña. Las gráficas de las validaciones pueden ser encontradas en apéndice A.

### 4.3 Gradiente descendente

Los resultados con el método del gradiente descendente fueron muy similares al de regresión lineal en términos del comportamiento y de precisión. El error cuadrático de este método arrojaba resultados parecidos al de la regresión lineal, aunque con un costo computacional mayor, por lo que consideramos más conveniente el método de regresión lineal. Como continuación de esta práctica nos gustaría experimentar con datasets más grandes y con diferentes comportamientos de datos para poder comparar cuál es mejor y en qué tipo de casos.




Las gráficas del dataset MTCARS usando gradiente descendente se encuentran en el apéndice B.

## 5 CONCLUSIONES Y REFLEXIONES

Existen múltiples metodologías para la creación de modelos de regresión lineal, en esta práctica pudimos probar dos: regresión lineal por mínimos cuadrados y gradiente descendente. Sin importar la metodología utilizada, las regresiones lineales tratan de predecir el comportamiento de una función a partir de variables. Durante esta práctica pudimos apreciar que estos métodos se comportan distinto dependiendo de la situación, por lo que no existe un método universal, si no que todo depende del problema a resolver y de las herramientas con las que se cuenta. Nosotros pudimos haber programado el cálculo de la regresión lineal desde cero pero al existir herramientas como scikit learn, podemos desarrollar fácilmente un modelo de regresión lineal para un conjunto de datos; algunas veces no todas la funciones se encuentran descritas en librerías, como la de BGD, por lo que siempre debemos de conocer los conceptos y teoría de las herramientas que empleamos. Si no existiera scikit learn aun así deberíamos poder haber realizado esta practica, herramientas como numpy, pandas y R la hacen mucho más sencilla, pero el conocer y dominar un tema como este es invaluable. El análisis de los datos resulta muy útil para entender su comportamiento antes de empezar a trabajar con ellos y comprobar si nuestro modelo nos está dando los resultados que esperamos. Quizá no pudimos llegar al 100% de precisión con estos métodos, pero entre más grande es nuestro conjunto de datos, mejor nos aproximamos.

### 5.1 Refrexión de Abraham

Esta práctica fue todo un reto desde el principio, en la clase había entendido a un muy alto nivel los conceptos, pero al ponerlos en práctica me doy cuenta que hay que ejercitar el conocimiento en entregables como este, ya que lo que había entendido aquel día de clase ahora es mucho más claro al implementarlo. Pensé que esta práctica sería sencilla, pero mis conocimientos de Python, R y estadística no son muchos, es por ello que era difícil conectar conceptos cuando las instrucciones no eran del todo claras, por ejemplo cuando en el análisis pedía generar los boxplots ¨correspondientes¨ no sabía a que columnas se refería, igual que al implementar métodos de scikit learn, tuve que ir aprendiendo y comprendiendo desde la marcha. Mis conocimientos son mejores ahora, también mi habilidad para investigar mejoró. Ahora hablando especifícamente de la práctica, se me hizo muy interesante cómo la regresión lineal nos puede ayudar a predecir y también como el BGD también nos ayuda a aproximarnos, para la siguiente práctica me gustaría poder implementar las cosas con más facilidad y comprender mejor para no revolverme, el resultado se logró y estoy seguro de que lo aprendido me será de mucha utilidad en mi carrera profesional.

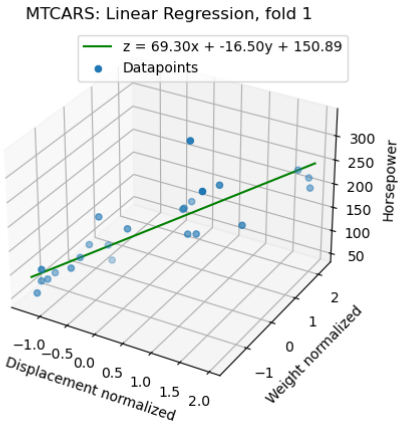
### 5.2 Reflexión de Mario

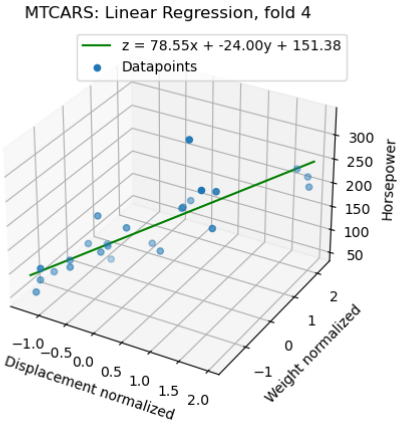
Considero que durante todo el desarrollo de la práctica pude obtener nuevos conocimientos: por un lado el uso de las librerías de Python sklearn, numpy, y pandas, y por otro en el ámbito de la estadística y los métodos numéricos aplicados a problemas reales como éste. La parte de la práctica que más disfruté fue implementar el algoritmo de gradiente descendente, ya que pude comprender la importancia de las librerías y cómo facilitan la creación de programas complejos, que en otros lenguajes sería un gran desafío.

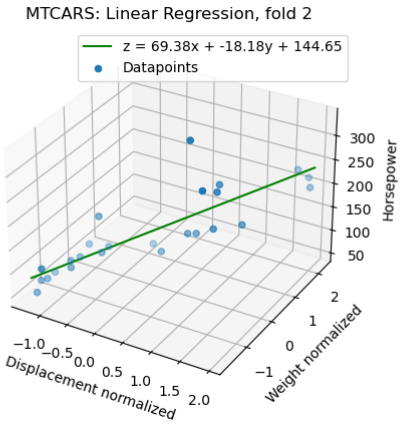
## REFERENCES

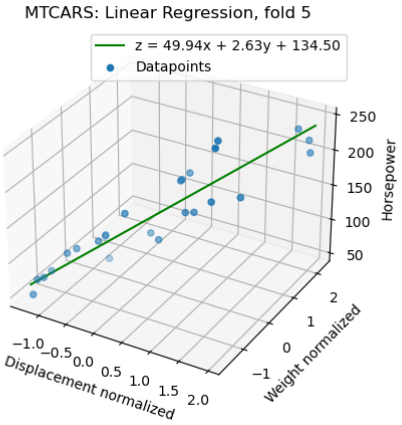
[1] Issam El Naqa and Martin J Murphy. 2015. What is machine learning? In *machine learning in radiation oncology*. Springer, 3–11.
[2] S. Gunjal. 2020. *Multivariate Linear Regression From Scratch With Python.* Retrieved August 22, 2020 from https://satishgunjal.com/multivariate_lr/
[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

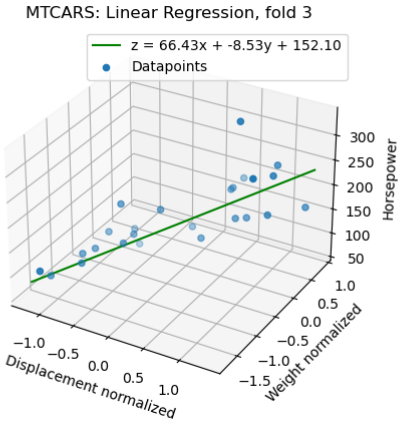
# A GRÁFICAS DE LA METODOLOGÍA N-FOLD CROSS VALIDATION USANDO REGRESIÓN LINEAL EN EL DATASET MTCARS



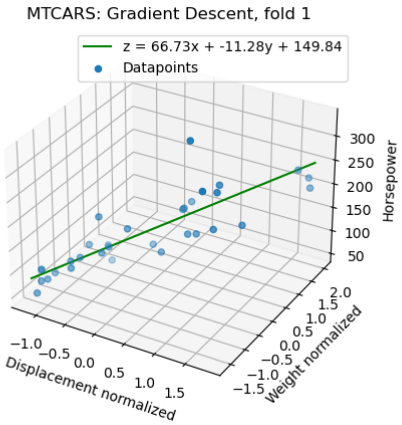



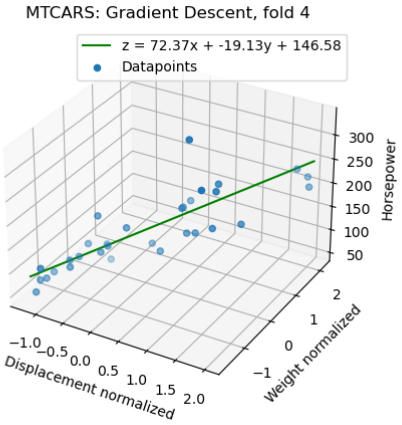



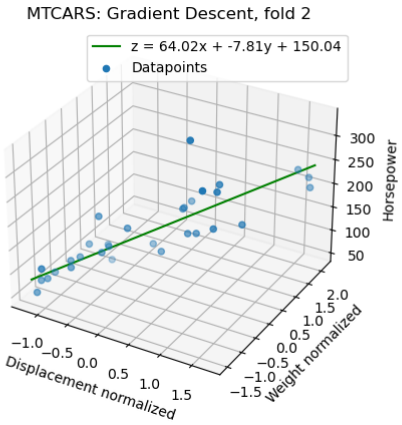



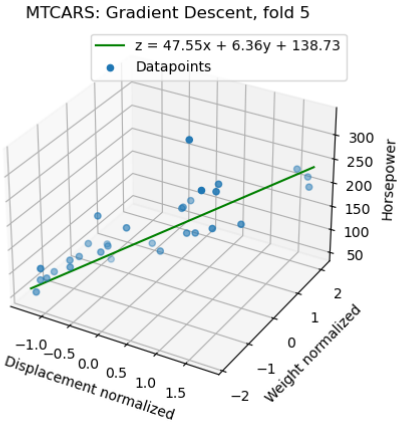



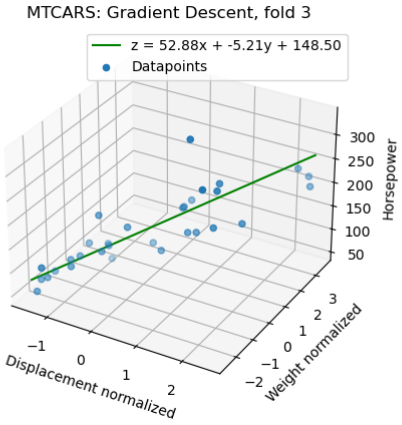
# B GRÁFICAS DE LA METODOLOGÍA N-FOLD CROSS VALIDATION USANDO GRADIENTE DESCENDENTE EN EL DATASET MTCARS



## C CÓDIGO DE ANÁLISIS ESTADÍSTICO

```
# Dataset GENERO
genero = read.table("genero.txt", header=TRUE, sep=",")

paste("Resumen estadístico de GENERO")
summary(genero)

paste("Generando boxplot para GENERO")
png(file="generoBoxplot.png")
par(mfcol=c(1,2))
boxplot(genero$Height, ylab="inches", xlab="Height", main="Height distribution")
boxplot(genero$Weight, ylab="pounds", xlab="Weight", main="Weight distribution")
dev.off()

paste("Generando gráfica de dispersión para GENERO")
png(file="generoDispersion.png")
plot(x=genero$Weight, y=genero$Height, xlab="Weight (lbs)", ylab="Height (in)",
    main="GENERO - Weight vs Height")
dev.off()

cat("\n\n\n")

# Dataset MTCARS
mtcars = read.table("mtcars.txt", header=TRUE)

paste("Resumen estadístico de MTCARS")
summary(mtcars)

paste("Generando boxplot para MTCARS")
png(file="mtcarsBoxplot.png")
par(mfcol=c(1,3))
boxplot(mtcars$disp, xlab="engine displacement",
    main="engine displacement distribution")
boxplot(mtcars$wt, xlab="weight", main="weight distribution")
boxplot(mtcars$hp, xlab="horsepower", main="horsepower distribution")
dev.off()

paste("Generando gráficas de dispersión para MTCARS")
png(file="mtcarsDispersion.png")
pairs(~disp+hp+wt, data=mtcars, main="MTCARS - Dispersion plots")
dev.off()
```

## D CÓDIGO DE REGRESIÓN LINEAL DEL DATASET GENERO

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

from GradientDescent import normalize
from graphs import graphGENERO

print(" ~ Reading genero.txt and generating train and test sets")
generoData = pd.read_csv('genero.txt')
height = generoData.iloc[:, 1].values.reshape(-1, 1)
height, _, _ = normalize(height)
weight = generoData.iloc[:, 2].values.reshape(-1, 1)
heightTrain, heightTest, weightTrain, weightTest = train_test_split(
    height, weight, test_size=0.2)

print(" ~ Creating linear regression model")
regressor = LinearRegression()
regressor.fit(heightTrain, weightTrain)

print(" ~ Testing linear regression model")
predictedWeight = regressor.predict(heightTest)
regressorMSE = mean_squared_error(weightTest, predictedWeight)
print(" →   MSE:", regressorMSE)
regressorR2 = r2_score(weightTest, predictedWeight)
print(" →   R2:", regressorR2)

theta = np.append(regressor.coef_.copy(), regressor.intercept_)
graphGENERO(height, weight, theta, "Linear Regression", "genero_linearReg.png")
```

## E CÓDIGO DE REGRESIÓN LINEAL DEL DATASET MTCARS

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, r2_score

from GradientDescent import normalize
from graphs import graphMTCARS

print(" ~ Reading mtcars.txt")
data = pd.read_csv('mtcars.txt', sep=' ')
data["disp"], _, _ = normalize(data["disp"])
data["wt"], _, _ = normalize(data["wt"])

print(" ~ Creating the k-fold cross validation model")
kfold = KFold()
mses = []
r2s = []

print(" ~ Creating the regression model for every train/test division")
for trainIdxs, testIdxs in kfold.split(data):
    # Fit model to data
    train = data.iloc[trainIdxs]
    trainX = train[["disp", "wt"]]
    trainY = train[["hp"]]
    regressor = LinearRegression()
    regressor.fit(trainX, trainY)

    # Graph model
    idx = len(mses) + 1
    theta = np.append(regressor.coef_.copy(), regressor.intercept_)
    graphMTCARS(trainX["disp"], trainX["wt"], trainY["hp"],
                theta, "Linear Regression, fold {}".format(idx),
                "mtcars_linearReg_{}.png".format(idx))

    # Predict data using model
    test = data.iloc[testIdxs]
    testX = test[["disp", "wt"]]
    testY = test[["hp"]]
    predictedY = regressor.predict(testX)

    # Score predicted data
    regressorMSE = mean_squared_error(testY, predictedY)
    regressorR2 = r2_score(testY, predictedY)
    print(" →   Partial MSE:", regressorMSE, "\tPartial R2: ", regressorR2)
    mses.append(regressorMSE)
    r2s.append(regressorR2)

print(" ~ Calculating final scores")
print(" →   Final MSE:", np.average(mses))
print(" →   Final R2:", np.average(r2s))
```

## F NUESTRA IMPLEMENTACIÓN DE GRADIENTE DESCENDENTE

```python
from typing import Tuple
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error


def normalize(x: pd.DataFrame) -> Tuple[np.ndarray, float, float]:
    mu = np.mean(x, axis=0)
    sigma = np.std(x, axis=0, ddof=1)
    x_norm = (x - mu) / sigma
    return x_norm, mu, sigma


class GradientDescent:
    def __init__(self, learning_rate: float = 0.1, max_iterations: int = 200,
                 precision: float = 0.00001):
        self.learning_rate = learning_rate
        self.max_iterations = max_iterations
        self.precision = precision
        self.theta = None
        self.mu = None
        self.sigma = None

    def fit(self, x: pd.DataFrame, y: pd.DataFrame):
        x, self.mu, self.sigma = normalize(x)
        x = np.hstack((x, np.ones((x.shape[0], 1))))
        self.theta = np.zeros(x.shape[1])
        prev_cost = -1
        for _ in range(self.max_iterations):
            predictions = x.dot(self.theta)
            errors = np.subtract(predictions, y)
            sum_delta = (self.learning_rate / x.shape[0]) * x.T.dot(errors)
            self.theta -= sum_delta
            cost = mean_squared_error(y, predictions)
            if abs(cost - prev_cost) < self.precision:
                break
            prev_cost = cost

    def predict(self, x: pd.DataFrame):
        if self.theta is None or self.mu is None or self.sigma is None:
            raise Exception(
                "GradientDescent::predict() called before model was trained")

        x = (x - self.mu) / self.sigma
        x = np.hstack((x, np.ones((x.shape[0], 1))))
        return x.dot(self.theta)
```

## G CÓDIGO DE GRADIENTE DESCENDENTE DEL DATASET GENERO

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

from GradientDescent import GradientDescent
from graphs import graphGENERO

print(" ~ Reading genero.txt and generating train and test sets")
data = pd.read_csv('genero.txt')
height = data.iloc[:, 1].values.reshape(-1, 1)
weight = data.iloc[:, 2].values.reshape(-1, 1)
heightTrain, heightTest, weightTrain, weightTest = train_test_split(
    height, weight, test_size=0.2)

print(" ~ Creating gradient descent model")
regressor = GradientDescent()
regressor.fit(heightTrain, weightTrain.ravel())

print(" ~ Testing gradient descent model")
predictedWeight = regressor.predict(heightTest)
regressorMSE = mean_squared_error(weightTest, predictedWeight)
print(" →   MSE:", regressorMSE)
regressorR2 = r2_score(weightTest, predictedWeight)
print(" →   R2:", regressorR2)

normHeight = (height - regressor.mu) / regressor.sigma
graphGENERO(normHeight, weight, regressor.theta,
            "Gradient Descent", "genero_gradient.png")
```

## H CÓDIGO DE GRADIENTE DESCENDENTE DEL DATASET MTCARS

```
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, r2_score

from GradientDescent import GradientDescent
from graphs import graphMTCARS

print(" ~ Reading mtcars.txt")
data = pd.read_csv('mtcars.txt', sep=' ')

print(" ~ Creating the k-fold cross validation model")
kfold = KFold()
mses = []
r2s = []

print(" ~ Creating the gradient descent model for every train/test division")
for trainIdxs, testIdxs in kfold.split(data):
    # Fit model to data
    train = data.iloc[trainIdxs]
    trainX = train[["disp", "wt"]]
    trainY = train[["hp"]]
    regressor = GradientDescent()
    regressor.fit(trainX, trainY.values.ravel())

    # Graph model
    idx = len(mses) + 1
    normX = (data[["disp", "wt"]] - regressor.mu) / regressor.sigma
    graphMTCARS(normX["disp"], normX["wt"], data["hp"], regressor.theta,
                "Gradient Descent, fold {}".format(idx),
                "mtcars_gradient_{}.png".format(idx))

    # Predict data using model
    test = data.iloc[testIdxs]
    testX = test[["disp", "wt"]]
    testY = test[["hp"]]
    predictedY = regressor.predict(testX)

    # Score predicted data
    regressorMSE = mean_squared_error(testY, predictedY)
    regressorR2 = r2_score(testY, predictedY)
    print(" →   Partial MSE:", regressorMSE, "\tPartial R2: ", regressorR2)
    mses.append(regressorMSE)
    r2s.append(regressorR2)

print(" ~ Calculating final MSE")
print(" →   Final MSE:", np.average(mses))
print(" →   Final R2:", np.average(r2s))
```

# I CÓDIGO DE GENERACIÓN DE GRÁFICAS

```
import sys
import matplotlib.pyplot as plt
import numpy as np

shouldDisplay = "--display-graphs" in sys.argv
shouldSave = "--save-graphs" in sys.argv


def graphGENERO(height, weight, theta, title, filename):
    fig, ax = plt.subplots()
    ax.set_title("GENERO: " + title)
    ax.set_xlabel("Height normalized")
    ax.set_ylabel("Weight")

    # Add scatter plot
    ax.scatter(height, weight, label="Datapoints")

    # Add line from theta parameters
    points = 60
    lineX = np.linspace(height.min(), height.max(), points)
    constant = np.ones(points)
    lineY = np.dot(np.vstack((lineX, constant)).T, theta)
    equation = "y = {:.2f}x + {:.2f}".format(theta[0], theta[1])
    ax.plot(lineX, lineY, 'g', label=equation)

    ax.legend()
    if shouldSave:
        fig.savefig(filename)
    if shouldDisplay:
        plt.show()


def graphMTCARS(disp, wt, hp, theta, title, filename):
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.set_title("MTCARS: " + title)
    ax.set_xlabel("Displacement normalized")
    ax.set_ylabel("Weight normalized")
    ax.set_zlabel("Horsepower")

    # Add scatter plot
    ax.scatter(disp, wt, hp, label="Datapoints")

    # Add line from theta parameters
    points = 60
    lineX = np.linspace(disp.min(), disp.max(), points)
    lineY = np.linspace(wt.min(), wt.max(), points)
    constant = np.ones(points)
    lineZ = np.dot(np.vstack((lineX, lineY, constant)).T, theta)
    equation = "z = {:.2f}x + {:.2f}y + {:.2f}".format(
        theta[0], theta[1], theta[2])
    ax.plot(lineX, lineY, lineZ, 'g', label=equation)

    ax.legend()
    if shouldSave:
        fig.savefig(filename)
    if shouldDisplay:
        plt.show()
```