- Si CAP>0: la distribución tiene una asimetría positiva, ya que la media es mayor que la moda.

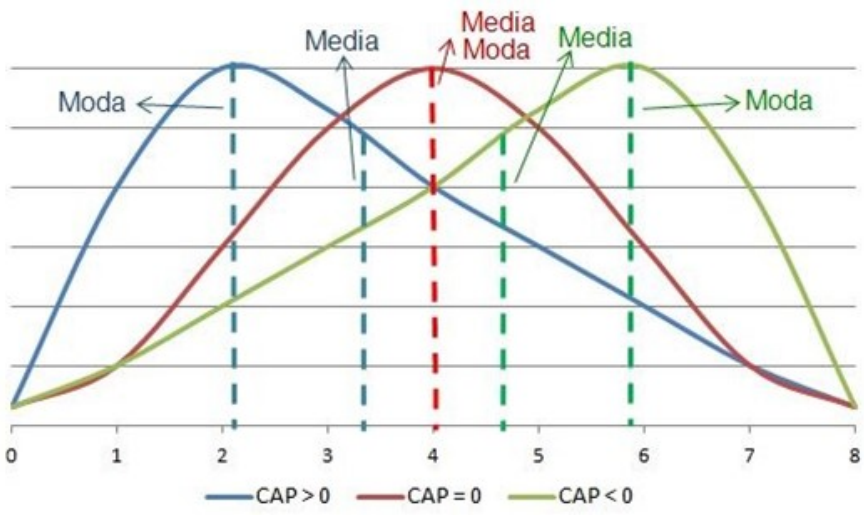

Figura 10. Interpretación de los resultados del Coeficiente de asimetría de Pearson.

El coeficiente de asimetría de Bowley CAB toma como referencia los cuartiles para determinar si la distribución es simétrica o no. Para aplicar este coeficiente, se supone que el comportamiento de la distribución en los extremos es similar. Sea el conjunto $X=(x_1, x_2,\ldots, x_N)$, la asimetría de Bowley es:

$$CA_B = \frac{Q_3 + Q_1 - 2Me(X)}{Q_3 - Q_1}$$

siendo Q_3 el tercer cuartil, Q_1 el primer cuartil y $Me(X)$ la mediana

Por tanto el resultado de este coeficiente como en los dos anteriores es el que sigue:

- Si CAB<0: la distribución tiene una asimetría negativa, puesto que la distancia de la mediana al primer cuartil es mayor que al tercero.

- Si CAB=0: la distribución es simétrica, ya que el primer y tercer cuartil están a la misma distancia de la mediana.

- Si CAB>0: la distribución tiene una asimetría positiva, ya que la distancia de la mediana al tercer cuartil es mayor que al primero.

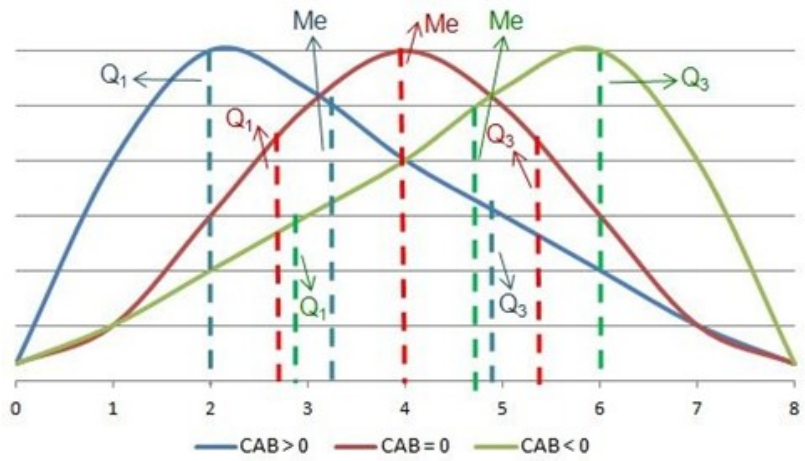

Figura 11. Interpretación de los resultados del Coeficiente de asimetría de Bowley.

Es importante revisar la literatura para ampliar, si se quiere, sobre las distintas formas de obtener esta medida.

Veamos ahora como calculamos la asimetría en Python. Hay varias formas de hacerlo y esta medida está disponible en varios módulos. Aquí, veremos cómo se calcula usando un método de la clase DataFrame de Pandas y la función disponible desde el módulo stats en scipy.

```
import pandas as pd
import numpy as np
import scipy.stats as st
#************************************************
diabeteDF = pd.read_csv("diabetes.csv")
print("Asimetria método de la Clase DataFrame: "+str(diabeteDF["Glucose"].skew()))
print("Asimetria función en Stats de scipy: "+str(st.skew(diabeteDF["Glucose"])))
diabeteDF["Glucose"].hist()
```

```
Asimetria método de la Clase DataFrame: 0.17375350179188992
Asimetria función en Stats de scipy: 0.17341395519987735
<matplotlib.axes._subplots.AxesSubplot at 0x7ff43416bb90>
```

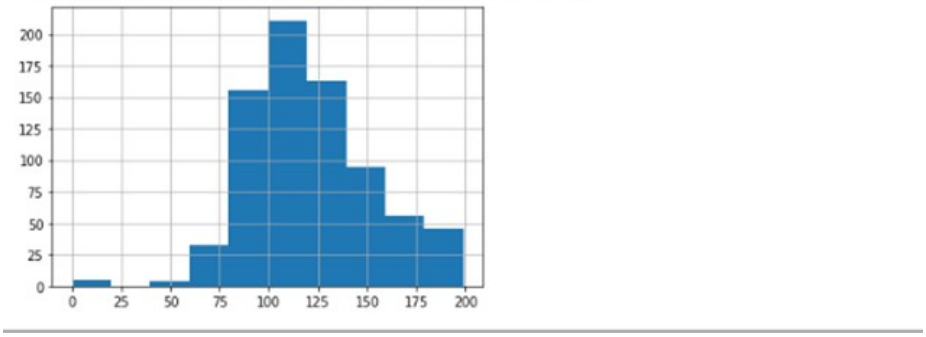

Figura 12. Cálculo del coeficiente de asimetría con scipy.stats y Pandas.

En scipy.stats.skew, la asimetría se calcula usando el coeficiente de Fisher-Pearson. Ver referencia de la API de scipy para más información.

La curtosis, al igual que la asimetría mide la forma de una distribución y permite, de forma analítica, sumarizar algún aspecto de esta. En este caso, el resultado de esta medida está asociado a las colas de la distribución y no a los picos.

- La Kurtosis de una distribución normal univariada es 3. Las distribuciones con kurtosis menores que 3 se les llama distribuciones platykurticas, mientras que a las que tienen un kurtosis mayor que 3 se les llaman leptokurticas.
- La Kurtosis se calcula de la misma forma que en el ejemplo anterior en Python.
- La kurtosis es la última de las medidas necesarias para caracteriza un rasgo o conjunto de datos univariado.

Por tanto, a modo de resumen, tenemos que cuando queremos caracterizar un

rasgo, tenemos que calcular, la media, la mediana, la moda, la asimetría y la kurtosis y tratar de visualizar la distribución de sus valores por medio de un histograma.

Con este conocimiento a mano podemos pasar a evaluar algunas de las formas en podemos saber, de manera analítica la distribución de los rasgos de nuestro conjunto de datos es normal o no.

Pruebas de normalidad

A este tipo de prueba, se les llaman test de normalidad. Entro los más usados se encuentran:

- Test de Shapiro-Wilk.
- El Test de D'Agostino o K-cuadrado.

El test de Shapiro-Wilk tiene como hipótesis nula que los datos provienen de una población Normalmente distribuida. La decisión de apoyar o rechazar la hipótesis nula depende del valor de p-value arrojado de esta.

Si p-value>0.05 Se apoya H0. (Los datos están normalmente distribuidos).

Si p-value<0.05 Se rechasa H0. (Los datos no están normalmente distribuidos).

A este valor 0.05 se le llama nivel de significación y el valor normalmente escogido es el de 5%.

En este pedazo de código se muestra cómo se aplica en Python: un valor mayor que 0.05 dice apoya la hipótesis de que los datos están normalmente distribuidos

```
import pandas as pd
import numpy as np
import scipy.stats as st
#################################################
diabeteDF = pd.read_csv("../input/pima-indians-diabetes-database/diabetes.csv")
print("Asimetria método de la Clase DataFrame: "+str(diabeteDF["Glucose"].skew()))
print("Asimetria función en Stats de scipy: "+str(st.skew(diabeteDF["Glucose"])))
print("Kurtosis en Stats de scipy: "+str(st.kurtosis(diabeteDF["Glucose"])))

print("p-value:"+str(st.shapiro(diabeteDF["Glucose"]).pvalue))
if st.shapiro(diabeteDF["Glucose"]).pvalue>0.05 :
  print("Los datos están normalmente distribuidos")
diabeteDF["Glucose"].hist()
```

```
Asimetria método de la Clase DataFrame: 0.17375350179188992
Asimetria función en Stats de scipy: 0.17341395519987735
Kurtosis en Stats de scipy: 0.6288133337300685
p-value:1.9867612763291298e-11
<AxesSubplot:>
```

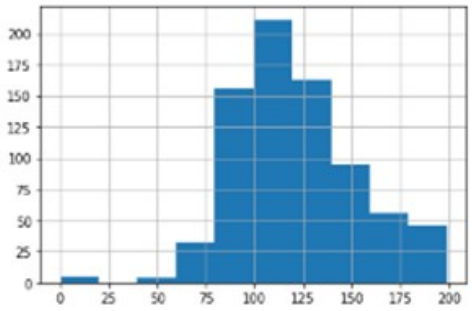

Figura 13. Ejemplo de uso del test Shapiro-Wilk para determinar si los dstos están normalmente distribuidos.

Con el test de Dágostino K-Cuadrado o K2 sucede exactamente lo mismo que con el anterior. Este presupone una hipótesis nula que soporta el hecho de que los datos son tomados de una población normalmente distribuida. Para probar o refutar la hipótesis se calcula el valor pvalue y si este es mayor que el nivel de significación se apoya la H0 y de lo contrario se rechaza.

Un valor mayor que 0.05 dice que se apoya la hipótesis de que los datos están normalmente distribuidos

```
import pandas as pd
import numpy as np
import scipy.stats as st
#************************************************
diabeteDF = pd.read_csv("../input/pima-indians-diabetes-database/diabetes.csv")
print("Asimetria método de la Clase DataFrame: "+str(diabeteDF["Glucose"].skew()))
print("Asimetria función en Stats de scipy: "+str(st.skew(diabeteDF["Glucose"])))
print("Kurtosis en Stats de scipy: "+str(st.kurtosis(diabeteDF["Glucose"])))

print("p-value:"+str(st.normaltest(diabeteDF["Glucose"]).pvalue))
if st.normaltest(diabeteDF["Glucose"]).pvalue>0.05 :
  print("Los datos están normalmente distribuidos")
diabeteDF["Glucose"].hist()
```

```
Asimetria método de la Clase DataFrame: 0.17375350179188992
Asimetria función en Stats de scipy: 0.17341395519987735
Kurtosis en Stats de scipy: 0.6288133337300685
p-value:0.0020446506991363502
<AxesSubplot:>
```

Figura 14. Ejemplo de uso del test Dágostino K-Cuadrado para determinar si los datos están normalmente distribuidos.

Existen varias pruebas de normalidad, entre ellas se encuentran:

- Prueba de normalidad Kolmogorov-Smirnov.
- Prueba de Anderson-Darling.
- Prueba de normalidad de Ryan-Joiner.

Que aunque siguen vías diferentes los resultados son más o menos similares.

En análisis exploratorio de datos, no solo se analizan los rasgos individuales, es importante, para ganar alguna comprensión de los datos, también analizar la correlación entre los rasgos, en busca de altos valores de correlación o ausencia de ellas. Como ya explicamos en la sección anterior para eso se usan algunas medidas de correlación.

Análisis exploratorio de datos, etapa #1: recolección e integración de los datos

En la sección anterior en la parte correspondiente a los problemas que pueden presentarse en esta etapa, vimos cómo nos enfrentamos a problemas como rasgos redundantes o casos redundantes. También se describieron las fuentes de las cuales

podríamos tomar los datos. Sin embargo no se han detallados los formatos más usuales en los que soportan los conjuntos de datos en las tareas de aprendizaje automático.

Por tanto en sección revisaremos formatos de archivos como los csv, o valores separados por coma, los archivos con formato jsom, entre otros.

Archivos con formato csv están especificado en el RFC 4180. Este, es un formato que permite representar arreglos rectangulares de valores textuales o numéricos. Estos arreglos rectangulares están conformados como filas y columnas. Estas columnas están separadas por el carácter coma (%x2C), mientras que las filas o records, se separan por medio de carácter de fin de linea. El RFC4180 especifica que sea CRLF. Cada línea debe contener el mismo número de campos.

Estos archivos pueden contener o no una primera fila que corresponde al nombre de los campos y debe contener el mismo número que los campos del resto del archivo. Aunque pueden haber variaciones, de manera general este es el formato que por defecto hemos estado usando en estas lecciones. En Python hay varias formas de leer este tipo de archivos. Una de las vías es hacerlo desde la librería estándar.

```
import os
os.chdir("C:\\Datos")
os.getcwd()

import csv
f= open("diabetes.csv")
reader = csv.reader(f)
for row in reader:
    print (row)
```

```
['6', '148', '72', '35', '0', '33.6', '0.627', '50', '1']

['1', '85', '66', '29', '0', '26.6', '0.351', '31', '0']

['8', '183', '64', '0', '0', '23.3', '0.672', '32', '1']
```

```
['1', '89', '66', '23', '94', '28.1', '0.167', '21', '0']
```

Este es el aspecto que tiene el contenido de un fichero con formato csv. También puede cargarse directamente a una estructura de datos llamada DataFrame de Pandas usando uno de los métodos de esta librería.

```
import pandas
filename = 'diabetes.csv'
data = pandas.read_csv(filename, header=0)
```

Aunque también, librerías como Numpy tienen sus propios mecanismos para leer este tipo de formato.

Otro de los formatos que son muy comunes a la hora de cargar datos para el aprendizaje automático es el llamado JSON.

JSON significa, por sus siglas en inglés, JavaScript Object Notation. Es un formato de fichero estándar abierto y de intercambios de datos que usa texto completamente entendible por los humanos.

Aunque en sus inicios se derivó de JavaScript hoy se ha convertido en un formato de uso general. Desde el 2006 se ha publicado una especificación, la RFC 4627 y luego estandarizado en 2013 por la ECMA-404 y el RFC8259 en 2017.

El formato de un fichero con formato json:

```
{
 "firstName": "John",
 "lastName": "Smith",
 "isAlive": true,
 "age": 27,
 "address": {
  "streetAddress": "21 2nd Street",
  "city": "New York",
  "state": "NY",
  "postalCode": "10021-3100"
 },
 "phoneNumbers": [
  {
   "type": "home",
   "number": "212 555-1234"
  },
  {
   "type": "office",
   "number": "646 555-4567"
  }
 ],
 "children": [],
 "spouse": null
}
```


Por cuestiones de espacio y tiempo no nos detendremos en la descripción de formato de este tipo de archivo. Por favor, pude remitirse a los correspondientes documentos RFC.

Para importar datos almacenados en formato json en Python lo hacemos

de la siguiente forma:

```
import pandas as pd

data = pd.read_json("data.json")
```

Este es uno de los formatos más comunes, pues Excel es también una excelente herramienta de análisis de datos. Eso hace que en las empresas sea muy común

encontrar grandes volúmenes de datos en forma de documentos .xlsx.

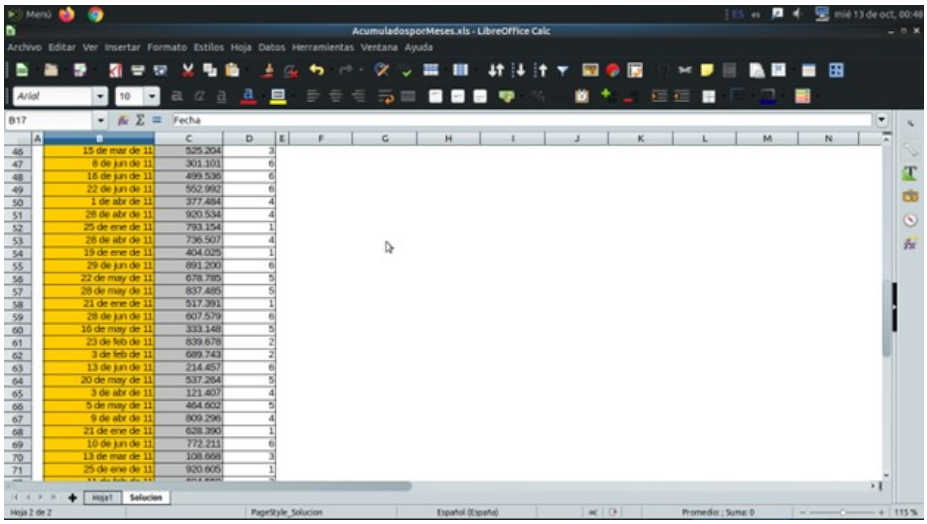

Figura 15. Ejemplo de un fichero .xls abierto en LibreOffice.

Este es un formato basado en XML y abierto de Microsoft. Este tipo de archivos, ya sea .xls o .xlsx, pueden leerse en Python de la siguiente forma:

```
import pandas as pd
df = pd.read_excel("../input/acumuladoxmeses-xls/AcumuladosporMeses.xls", sheet_name="Solucion")
df.describe
```

```
<bound method NDFrame.describe of    Unnamed: 0                Unnamed: 1     Unnamed: 2 Unnamed: 3  \
0         NaN                       NaN            NaN        NaN
1         NaN                       NaN            NaN        NaN
2         NaN                       NaN            NaN        NaN
3         NaN                       NaN            NaN        NaN
4         NaN                       NaN            NaN        NaN
..        ...                       ...            ...        ...
77        NaN  2011-05-08 23:45:55.273000  427486.933567          5
78        NaN  2011-02-28 20:05:05.015000  842768.382835          2
79        NaN  2011-03-05 11:39:25.071000  228402.734439          3
80        NaN  2011-02-25 06:26:57.411000  467385.039845          2
81        NaN  2011-05-03 10:17:08.541000  300136.624519          5

    Unnamed: 4 Unnamed: 5 Unnamed: 6 Unnamed: 7 Unnamed: 8 Unnamed: 9  \
0          NaN        NaN        NaN        NaN        NaN        NaN
1          NaN        NaN        NaN        NaN        NaN        NaN
2          NaN        NaN        NaN        NaN        NaN        NaN
3          NaN        NaN        NaN        NaN        NaN        NaN
4          NaN        NaN        NaN        NaN        NaN        NaN
..         ...        ...        ...        ...        ...        ...
77         NaN        NaN        NaN        NaN        NaN        NaN
78         NaN        NaN        NaN        NaN        NaN        NaN
79         NaN        NaN        NaN        NaN        NaN        NaN
80         NaN        NaN        NaN        NaN        NaN        NaN
81         NaN        NaN        NaN        NaN        NaN        NaN
```

En el ejemplo anterior hemos cargado un documento con formato .xls a una estructura DataFrame de Pandas, en donde podemos tratarlo de forma adecuada para obtener los datos que necesitamos.

De esta misma forma, podemos cargar datos desde un archivo HDF5, ya que estos ficheros, como los de XML son autodescriptivos pueden contener grandes cantidades de datos con relaciones complejas. Como casi siempre, pandas viene a resolvernos los problemas:

```
import pandas as pd

df = pd.read_hdf('train.h5')
```

De igual menara podemos cargar archivos comprimidos en .zip o imagenes y hasta ficheros en formato pdf.

Bueno, hasta aquí hemos visto lo referente a los tipos de archivos y formatos de datos más usados en aprendizaje automático y que de seguro usaran en su día a día, sin embargo no es solo de ficheros o archivos desde donde provienen los datos que necesitamos en nuestro trabajo, también en muchas ocasiones los datos en tablas de bases de datos relacionales o documentos no sql, almacenados en servidores o en la nube.

Ahora, como lo hicimos con la manera que se cargan los datos dese archivos, haremos un recorrido de cómo podemos cargar los datos desde los sistemas de gestión de bases de datos más conocidos.

MySQL Server es uno de los sistemas de gestión de bases de datos relacionales más usados y populares. Una de las tantas formas es usando mysql.connector

```
import mysql.connector
conexion=mysql.connector.connect(host="localhost",
                                 user="root",
                                 passwd="",
                                 database="Test_sch")
c_cursor=conexion.cursor()
c_cursor.execute("insert into testTable (id,nombre) values(200,'Evqueni');")
c_cursor.execute("select * from testTable")
for fila in c_cursor:
    print(fila)
conexion.close()
```

```
(200, 'Evqueni')
```

Figura 16. Conectando con MySQL desde Python, usando mysql.connector.

Otro de los Sistemas de Gestión de Bases de Datos relacionales muy usado es el SQL Server de Microsoft. En esta parte mostramos como se puede ejecutar una consulta en un servidor de este tipo. En este caso haciendo uso de ODBC, por medio de pyodbc.

```
import pandas as pd
import pyodbc
server = 'tcp:localhost'
database = 'testDATA'
username = 'sa'
password = 'sasa'
cnxn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server};\
                      SERVER='+server+';DATABASE='+database+';UID='+username+';PWD='+ password)
cursor = cnxn.cursor()
#cursor.execute("INSERT INTO testTable (id,nombre) VALUES(101,'E');")
cursor.execute("SELECT id, nombre from testTable;")
row = cursor.fetchone()
while row:
    print(row)
    row = cursor.fetchone()
```

```
(100, 'E')
(101, 'E')
```

Figura 17. Conectando con MS SQL Server desde Python, usando pyodbc.

Otra de las fuentes comunes de datos y que cada vez gana más admiradores es MongoDB, de la familia de los sistemas noSQL. En la siguiente figura se muestra un ejemplo de cómo se efectúa dicha conexión.

```
import pymongo
from pymongo import MongoClient
# conexión
con = MongoClient('localhost',27017)
db = con['testDATA']
for i in range(20):
    db['testCollection'].insert_one({'identificador':i, 'nombre':'step_'+str(i)})

records = db['testCollection'].find()
print(list(records))
```

Figura 18. Conectando con MongoDB desde Python, usando pymongo.

Hasta aquí hemos visto, como conectarse y recuperar datos de tres tipos diferentes de Sistemas de Gestión de Bases de Datos. Una vez que los datos son importados,

es posible procesarlo con pandas.

En esta etapa pueden eliminarse rasgos o añadirse o se pueden eliminar casos que se encuentren duplicados o añadir si se considera adecuado.

Siempre es importante destacar que este proceso de Análisis Exploratorio de Datos y sus etapas no son rígidas y tiene que verse más como un proceso iterativo. En este proceso se avanza y se retrocede de una fase a otro en la medida que la adaptación de los datos a las tareas de aprendizaje automático lo requieran.

Análisis exploratorio de datos, etapa #2: realización de una inspección general del conjunto de datos.

Aunque esta tarea de inspeccionar el conjunto de datos puede hacerse de manera manual, inspeccionando las variables una a una, nosotros mostraremos dos métodos que generan un perfil del conjunto de datos que aportan los gráficos y la información que de otra manera obtendríamos de forma más elemental.

El primer método que usaremos es el de generar un informe o reporte completo sobre el dataset que estemos trabajando por medio del uso del módulo sweetviz. Veamos el ejemplo y luego comentemos los detalles de la información que genera.

```
import pandas as pd
import sweetviz as sv
#**************************************************
diabetesDF = pd.read_csv("diabetes.csv")
#**************************************************
Report = sv.analyze(diabetesDF)
Report.show_html('diabetes_report.html')
```

Figura 19. Código para generar un informe en Html de un dataset usando SweetViz.

Como puede verse es sumamente sencillo y brinda muchísima información de todas las variables de dataset sin necesidad de ir mirándolos uno a uno. En la siguiente figura veremos parte del informe.

Figura 20. Reporte en Html generado por el módulo SweetViz, del dataset diabetes.

En la parte central del reporte, se muestran los detalles generales del dataset, como la cantidad de casos, que para este en particular, son 768. Muestra la cantidad de rasgos y si tiene duplicados o no. En la parte izquierda, muestra cada uno de los rasgos del dataset junto a si histograma. Cuando se pone el puntero del mouse sobre uno de los rasgos se abre de forma inmediata a la derecha los detalles de este.

Cuando se pincha sobre el botón que se encuentra en la parte superior central que dice Associations, se muestra la matriz de correlación de estas variables, calculados usando el coeficiente de correlación de Pearson.

En la siguiente imagen se muestra la matriz.

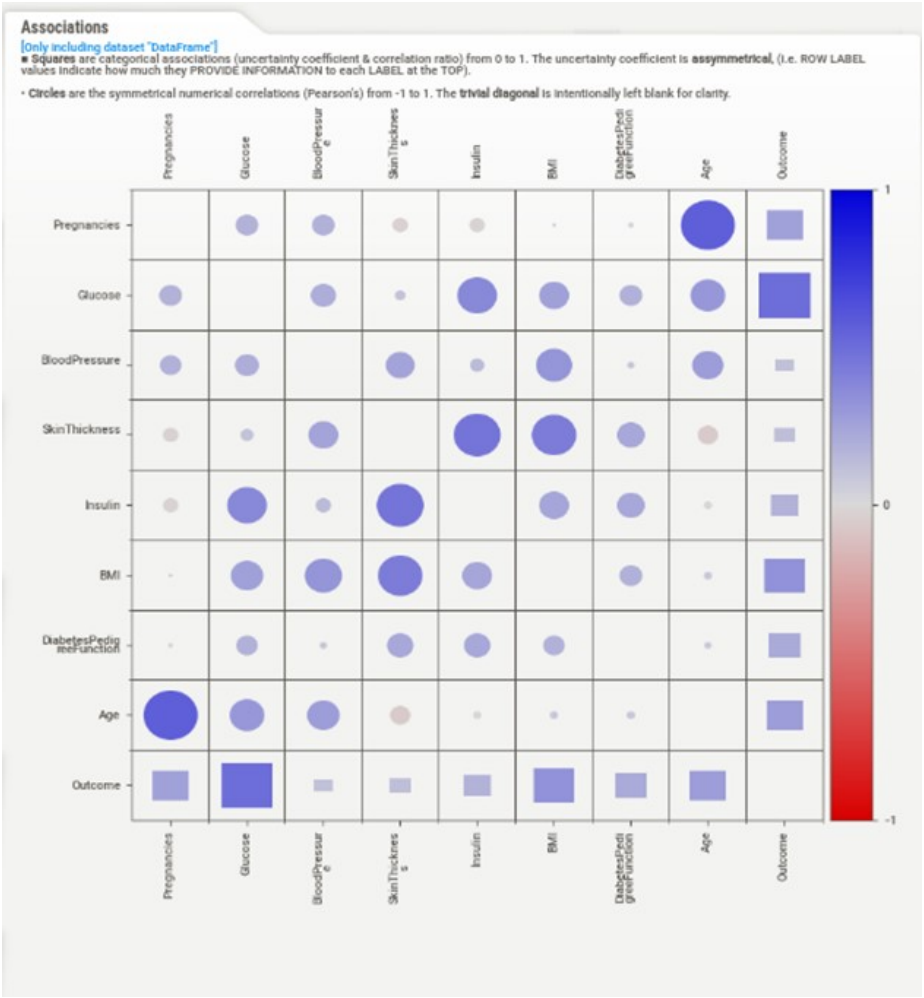

Figura 21. Matriz de correlaciones generada por el módulo SweetViz, usando el coeficiente de correlación de Pearson.

El segundo métodos que se puede usar, puede que sean muchos, es usando el módulo pandas_profiling. Veamos cómo se ejecuta el reporte en código y luego analicemos el resultado.

Este reporte es mucho más completo que el anterior y está compuesto por seisu8 secciones que sumarizan diferentes aspectos del dataset bajo análisis.

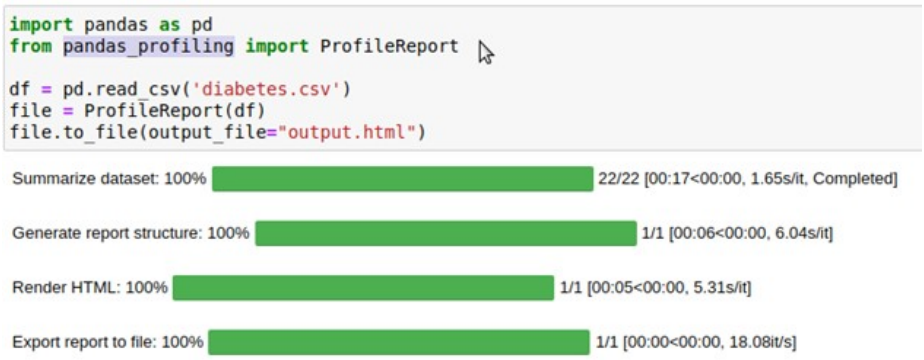

Figura 22. Código para generar un reporte del perfil del conjunto de datos que se analiza, usando el módulo pandas_profiling.

En las siguientes imágenes veremos cada una de las secciones en que está divido este reporte y que información nos aporta. Las secciones en las que se divide este reporte son las siguientes:

- Overview.
- Variables.
- Interacciones.
- Correlaciones.
- Missing Values.
- Samples (Divida a su vez en First Rows y Last Rows).

Veamos la imagen y luego comentemos.

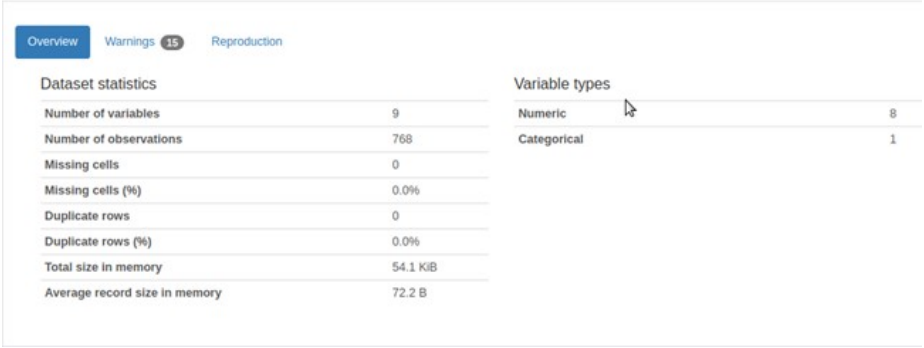

Figura 23. Sección de Overview de un reporte generado por pandas_profiling.

Esta sección de reporte nos brinda una panorámica general del dataset que investigamos. Tiene tres subsecciones, aunque la más informativa es la que está destacada en azul.

Lo interesante de esta parte es que nos dice, la cantidad de casos, la cantidad de rasgos y el tipo de variable. Por ejemplo, este dataset tiene 9 variables, 8 numéricas y 1 categórica. Además nos dice que no tiene casos duplicados, etc.

Sección de variables:

En esta sección se incluye un perfil completo de cada una de las variables o rasgos del dataset.

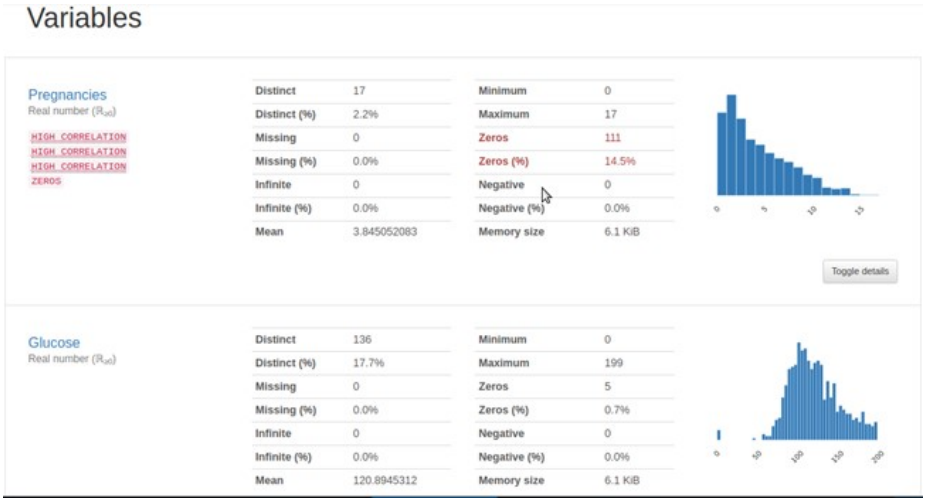

Figura 24. Sección de Variables de un reporte generado por pandas_profiling.

Incluye a la derecha de cada una de las secciones de las variables, un botón que de oprimirse se abre, debajo de esta, los detalles de cada una de las variables.

Esta sección nos permite observar, por ejemplo, que la variable Pregnancies, tiene 111 ceros. Si tenemos en cuenta la naturaleza de esta variable, que cuanta la cantidad de veces que una paciente ha estado embarazada, no es un problema y eso solo significa que no lo ha estado.

Sin embargo, en la siguiente variable, podemos ver que tiene cinco ceros. Esta variable mide la cantidad de glucosa en sangre y es imposible que tome valores cero y de hecho ya sabemos que aquí hay problema a solucionar. Estas son los tipos de problemas que debemos solucionar en esta etapa, aplicando las técnicas que ya hemos visto.

En la sección de interacciones, se muestran los scaterplot entre cada una de las variables del dataset.

En la siguiente imagen se muestra el grafico scatter plot entre las variables BloodPressure y Insulin del conjunto.

Figura 25. Sección de Interactions de un reporte generado por pandas_profiling.

Información adicional de cómo usar Scatter Plot en Python puede encontrarse en Python DataScience Handbook.

La sección correlaciones muestra las correlaciones entre las variables del dataset usando varios coeficientes: Coeficiente de correlación de Pearson, el de Spearman, Kendall y Phik.

En estas matrices de correlaciones, se usan colores para denotar la fortaleza de estas. Un azul fuerte es indicativo de una correlación fuerte positiva, mientras que los colores rojos, denotan una correlación fuerte negativa. Cambiando de pestaña accedemos a la matriz de correlación de cada uno de los coeficientes. Este tipo de matriz es muy interesante y fácil de explorar cuando la cantidad de rasgos son relativamente pocos; pero cuando son muchos se hace difícil.

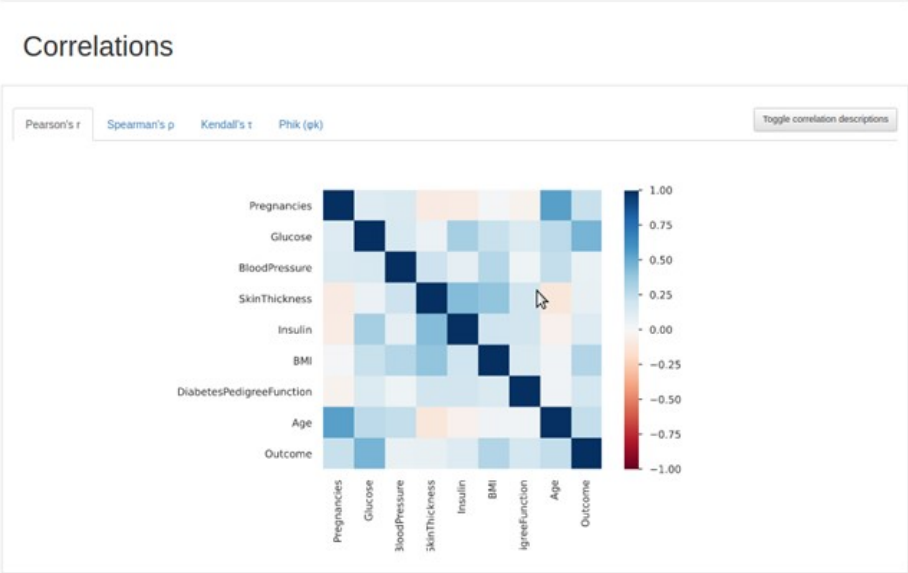

Figura 26. Sección de Correlaciones de un reporte generado por pandas_profiling.

Aunque estos tipos de reportes sin visualmente atractivos y aportan mucha información, la manera más fácil y rápida de obtener un resumen informativo de un dataset es por medio de DataFrame.describe().

Estamos asumiendo que el dataset está almacenado en una estructura DataFrame de Pandas, que una de las formas más comunes cuando analizamos datos con Python. Veamos una imagen que muestra el resultado de llamar el método describe() de un dataframe que almacena el dataset Pima Indian Diabete.

```
import pandas as pd
pima_diabetesDF = pd.read_csv("diabetes.csv")
pima_diabetesDF.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Figura 27. Resultado de aplicar describe a un DataFrame de Pandas.

Este método retorna la cantidad de casos en cada rasgo, la media, la desviación estándar, los valores mínimo y máximo del raso y el 1er, 2do y 3er cuartil.

Con esta información a mano, podemos identificar posibles anomalías en los datos que integramos y tratarlo de forma adecuada aplicando las técnicas que hemos visto a lo largo de esta lección. Si analizamos, por ejemplo, la variable Outcome a partir del reporte obtenido por medio de profiling, nos damos cuenta a simple vista que las clases están desbalanceadas. Veamos la imagen de la variable en el reporte.

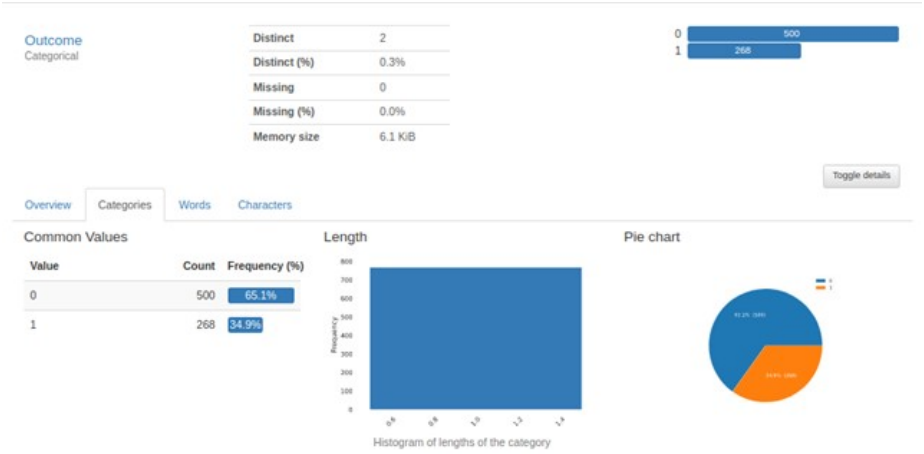

Figura 28. Variable de decisión Outcome del Pima Indian Diabetes dataset.

Queda claro que el 65.1% de los casos, pertenecen a pacientes confirmados no diabéticos y el 34.9% a los que sí se sabe que lo son. La idea que pretendemos transmitir es que, es en la etapa de Análisis Exploratorio, donde se comienza a manosear, por decirlo de alguna forma, los datos y por medio de un proceso iterativo, resolver los problemas que se presentan en el dataset. Es importante subrayar que la idea, básica, de todo este proceso es adaptar los datos a los requerimientos de los métodos de aprendizaje que utilizaremos.

Es en la próxima lección donde comenzaremos a ver los detalles de cada uno de los métodos.

Conclusiones

Tema 2. Análisis exploratorio de datos y preprocesamiento

En esta lección hemos tratado de explicar, aunque en modo alguno muy resumido, las técnicas más importantes y conceptos que forman la base del análisis exploratorio de datos y sus etapas. También hemos visto la manera de realizar algunas tareas que son vitales para para la adaptación de los datasets a los métodos de aprendizaje.

Es sumamente importante que se tenga en cuenta que el Análisis Exploratorio de Datos es un tema basto y lleno de conceptos que es imposible que cubramos en un curso de estas características y por lo vital de su entendimiento es necesario que consulten toda la literatura extra que les sea posible.

Tema 3. Algoritmos de aprendizaje automático supervisado

3.1. Introducción y objetivos

Esta unidad formativa está compuesta por dos lecciones que comprenden lo referente a los algoritmos y modelos de aprendizaje supervisado.

En la primera lección se inicia con una introducción a la regresión para luego pasar a la discusión de los conceptos de bias/varianza. Después de discutir este medular asunto se para a otro concepto no menos importante, el de validación cruzada para entonces analizar las métricas de evaluación de los modelos, tanto de regresión como de clasificación y por ultimo de analizan los principales algoritmos de regresión.

En la segunda lección se tratan los principales algoritmos de clasificación. Se tratan los algoritmos KNN, SVM, naives bayes, Arboles de decisión, random forest y por ultimo ensamblados de clasificadores.

En las lecciones anteriores revisamos todo lo correspondiente a la etapa de preprocesamiento de los conjuntos de datos con el fin de adaptarlos a los métodos de aprendizaje automático que necesitamos para resolver una tarea determinada. En esta lección y la siguiente estaremos revisando lo correspondiente a los algoritmos de aprendizaje automático y las métricas para su evaluación. Creemos necesario definir dichas métricas antes de comenzar el estudio mismo de los algoritmos pues entender como los evaluamos hará mucho más fácil entender cómo funcionan.

Por tanto los objetivos de esta lección serán: Definir aprendizaje supervisado por regresión y clasificación, definir los conceptos de Bias y Varianza y describir el compromiso bias/varianza, definir qué se entiende por validación cruzada, las métricas usadas para evaluar los modelos de regresión y clasificación y por ultimo describir los algoritmos más comunes.

Tema 3. Algoritmos de aprendizaje automático supervisado

Cuando termine estas dos lecciones usted conocerá los algoritmos de aprendizaje supervisado más usados, como se evalúan y cómo se aplican a problemas en la práctica.

Los objetivos que se perseguirán en este tema son:

- Conocer en que consiste la regresión y su diferencia respecto a la clasificación.
- Conocer los conceptos de bias y varianza de un modelo de aprendizaje automático.
- Conocer y aplicar el concepto de validación cruzada.
- Conocer y aplicar las métricas de evaluación de los modelos de regresión, así como los de clasificación.
- Conocer y aplicar algunos de los modelos de regresión más importantes.
- Conocer y aplicar algunos de los modelos de aprendizaje supervisado de clasificación más importantes, como KNN, SVM, entre otros.

Tema 3. Algoritmos de aprendizaje automático supervisado

3.2. Algoritmos de aprendizaje automático supervisado I

Desde la primera lección hemos estado hablando de manera muy informal de que el aprendizaje automático se divide en tres grandes grupos: Aprendizaje supervisado, el no supervisado y el aprendizaje con reforzamiento. En esta lección nos concentraremos en lo correspondiente al aprendizaje supervisado y sus dos tareas, regresión y clasificación. Para iniciar, estableceremos el problema del aprendizaje supervisado, revisaremos los conceptos de bias y varianza, luego revisaremos los métodos de evaluar los modelos de regresión y clasificación y por último veremos algunos de los algoritmos.

Desde las primeras lecciones cuando comenzamos a estudiar el problema del aprendizaje automático dejamos claro que los algoritmos de aprendizaje automático aprendían a partir de una experiencia que se daba en forma de un conjunto de datos. Este conjunto de datos le llamábamos dataset y estaba compuesto por casos y que estos casos estaban compuestos por rasgos o variables y que estos podrían ser clasificados según su función en predictores y variable predicha. Sabemos además que estos rasgos podían ser cuantitativos o cualitativos.

De manera general, el problema del aprendizaje, se resume en aprender un mapeo o función entre las variables predictoras, que en lo adelante llamaremos *x* y una variable predicha la cual llamaremos *y*.

Con estos elementos podemos establecer nuestra primera definición.

Definición #1: Aprendizaje supervisado

Se le llama aprendizaje automático supervisado al problema de aprender un mapeo entre el conjunto de variables predictoras x y la variable predicha y, del conjunto de datos D, tal que:

$h_D: X \rightarrow Y$

Definición #2: Regresión

Se le llama Regresión al problema de aprender un mapeo $h_D: X \rightarrow Y$, tal que la variable Y es una variable cuantitativa. (Numérica)

Definición #3: Clasificación

Se le llama Clasificación al problema de aprender un mapeo $h_D: X \rightarrow Y$, tal que la variable Y es una variable cualitativa.

Estas definiciones no tratan de ser matemáticamente precisas, solo pretenden de alguna manera desarrollar la intuición manteniendo al mínimo la complejidad matemática.

Podemos esclarecer un tanto estos conceptos si examinamos algunos de los conjuntos de datos que se usan a la hora de evaluar los modelos generados por los algoritmos de aprendizaje.

Revisemos el dataset Pima Indian Diebetes, que hemos estado usando en las lecciones anteriores. Este dataset tiene, 9 variables y 768 casos. Las variables predictoras, todas son variables numéricas, mientras que la variable predicha, la cual se llama Outcome, solo toma valores 0 y 1. Si nos atenemos a la definición #3 queda claro que este dataset pertenece a una tarea de clasificación. Específicamente con este dataset lo que se pretende es encontrar un mapeo hD que clasifique los pacientes o casos en diabéticos o no a partir de los 8 rasgos predictores: Insulina en sangre, niveles de glucosa medidas, presión sanguínea, edad, etc.

Por el contrario, si revisamos el dataset de ventas de video juegos, vgsales.csv, este contiene 11 rasgos, 5 de ellos son cadenas de caracteres, 6 son numéricos. Si quisiéramos predecir las ventas globales de juegos para una de las plataformas determinadas, entonces estaríamos desarrollando una tarea de regresión, pues la

variable Global_sales, es una variable numérica.

Introducción a la regresión

La forma que toma el mapeo hD en el caso de una regresión linear simple es el siguiente:

$$y = \beta_1 x_i + \beta_0 + \varepsilon$$

Donde y es la variable dependiente o predicha, x los rasgos del dataset o variables independientes o predictores, β_0 se le conoce como la intersección y β_1 el coeficiente de la regresión. Mientras que $\varepsilon \sim N(\mu,\sigma)$ es el error normalmente distribuido.

De esta forma, para predecir el valor de y a partir del valor de x tenemos que estimar los valores de β_0 y β_1 . La forma más simple de hacerlo la propuso Gauss en 1809 y se llama Mínimos Cuadrados.

Veamos en que consiste este método y como se estiman los parámetros a partir de los datos. Con este método β_1 se calcula de la siguiente forma:

$$\beta_1 = \frac{Cov(x,y)}{{S_x}^2} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})}$$

Mientras que β_0 se estima de la siguiente forma: $\beta_0 = \bar{y} - \widehat{\beta_1}\bar{x}$

Hagamos un ejemplo muy sencillo para demostrar cómo funciona. Lo primero es revisar los datos y sobre los cuales entrenaremos nuestro algoritmo y luego haremos predicciones y veremos como evaluamos el error. Claro, esto lo haremos con una sola variable para simplificar todos los cálculos.

Es importante notar que no hemos hecho ninguna asunción desde el punto de vista estadístico y que nos hemos centrado en exponer los elementos básicos.

Supongamos que contamos con los datos de la producción de trigo y harina de

España:

Producción de trigo (Tonelada)	30	28	32	25	25	25	22	24	35	40
Precio de la Harina (Euros)	25	30	27	40	42	40	50	45	30	25

Con estos datos vamos a estimar los valores de la ecuación de regresión.

$$\widehat{\beta_1} = \frac{\sum_{i=1}^{10} x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^{10} {x_i}^2 - n\bar{x}^2} = \frac{9734 - 10 * 28{,}6 * 35{,}4}{8468 - 10 * (28{,}6)^2} = -1{,}3537$$

Ya tenemos el estimado del primer parámetro, hagamos la estimación del segundo.

$$\beta_0 = \bar{y} - \widehat{\beta_1}\bar{x} = 35{,}4 + 13537 * 28{,}6 = 74{,}116$$

De esta forma, nuestra ecuación de regresión toma la forma siguiente:

$$\widehat{y} = 74{,}115 - 1{,}3537 * x$$

Con esta ecuación obtenida de los datos, podemos comenzar a predecir los valores de la harina a partir de valores la cantidad de trigo producido. Es a este proceso de calcular los valores de los estimadores a lo que se le llama proceso de entrenamiento y la ecuación que resulta es a lo que se llama modelo.

Hagamos un par de ejemplos y tratemos de predecir algunos valores con los que se entrenó, sustituyendo *x* en ella.

Tomemos a x=30 entonces y luego uno a uno el resto de los valores:

$$\widehat{y} = 74{,}115 - 1{,}3537 * 30 = 33{,}5$$

Ahora bien, como evaluamos si la función que generamos está correcta o se desvía demasiado de lo que queremos.

Tomemos todos los valores de la tabla y añadámosle los valores predichos y el error.

Producción de trigo (Tonelada)	30	28	32	25	25	25	22	24	35	40
Precio de la Harina(Euros)	25	30	27	40	42	40	50	45	30	25
Predicción	33.5	36.21	30.79	40.27	40.27	40.27	44.33	41.62	26.73	19.96
Residuo	-8,50	-6.21	-3.79	-0.27	1.72	-0.27	5.66	3.37	3.26	5.03

Después de calcular los residuales, el valor que nos queda:

$$S^2 = \frac{\sum_{i=1}^{10} (y - \hat{y})^2}{n-2} = \frac{207{,}6}{10-2} = 25{,}95$$

Este 25,95 de alguna forma caracteriza la calidad de nuestra función.

Acabamos de hacer un breve recorrido por el entrenamiento de un modelo de aprendizaje o la construcción de un modelo de predicción. Con este ejemplo entendido es necesario ir introduciendo otros conceptos que son necesarios para ir descubriendo aspectos del aprendizaje automático que serán de gran ayuda en el futuro.

Sin embargo es instructivo que hagamos este mismo ejemplo usando las librerías de aprendizaje automático de Python con el fin de comprobar si nuestro procedimiento ha sido el correcto. Veamos este ejemplo:

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import pearsonr
from sklearn.linear_model import LinearRegression
#*************************************************
ProduccionTrigo =[30,28,32,25,25,25,22,24,35,40]
PrecioHarina =[25,30,27,40,42,40,50,45,30,25]
d={'ProduccionTrigo':ProduccionTrigo,'PrecioHarina':PrecioHarina}
#*************************************************
regresionDF = pd.DataFrame(d)
X=regresionDF['ProduccionTrigo'].values.reshape(-1,1)
Y=regresionDF['PrecioHarina'].values
regress=LinearRegression()
regress.fit(X,Y)
print("Intercept:",regress.intercept_)
print("Coeficiente:",regress.coef_)
print("R-Cuadrado:",regress.score(X,Y))
plt.plot(X,(74.115-1.3537*X),color='red')
plt.scatter(X,Y)
plt.xlabel("Producción de Trigo(Toneladas)")
plt.ylabel("Precio de la harina (euro)")
plt.title("Producción de trigo vs precio de la harina")
plt.show()
```

```
Intercept: 74.11511789181691
Coeficiente: [-1.35367545]
R-Cuadrado: 0.7176465181664973
```

Figura 1. Ejemplo de regresión lineal, la cual ya habíamos calculado a mano.

Pues miremos las últimas líneas de la imagen anterior y upsss, los mismos valores de intersección y la pendiente que obtuvimos en nuestros cálculos manuales.

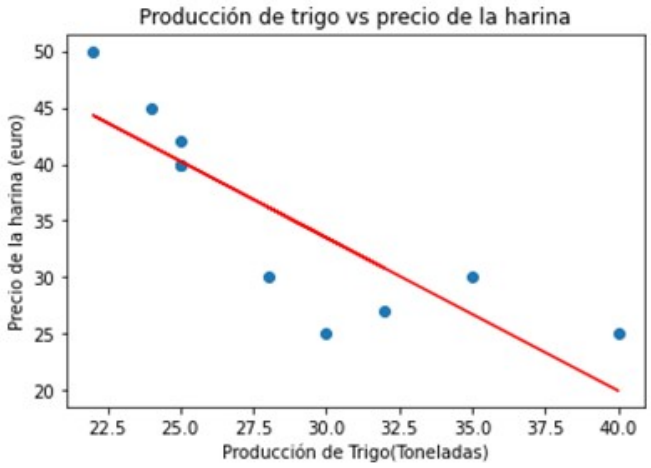

Figura 2. Gráfico resultante de la regresión que calculamos a mano. La línea roja es la y=74,115-1,3537*x, llamada línea de regresión.

Bien, conociendo este tipo de modelo simple y sabiendo de primera mano que es lo que se hace durante el proceso de entrenamiento o de ajuste del modelo, podemos introducir conceptos que pensamos que son imprescindibles para continuar con el aprendizaje del uso de los métodos de aprendizaje automático.

Introducción de los conceptos de Bias, Varianza y compromiso Bias/Varianza de los modelos de aprendizaje automático

Para entender que es el bias o sesgo de un modelo de aprendizaje automático es necesario que introduzcamos varias ideas que pueden parecer complejas; pero trataremos de introducirlas haciéndolas de alguna manera entendible.

Establezcamos el problema del aprendizaje desde una perspectiva más matemática. Supongamos que necesitamos predecir el tiempo de gestación de las pacientes de un hospital. Ponemos este ejemplo, por el simple hecho de que el tiempo de gestación es un proceso natural del cual no tenemos control y es determinado por factores que no conocemos. Por tanto los tiempo de gestación dependen de factores