

ANÁLISIS DE DATOS

Índice de usuario

PARTE I. CONCEPTOS BÁSICOS.....	4
1.1. EL PROBLEMA.....	4
1.2. ENTENDIENDO LOS DATOS.....	5
1.3. PAQUETES PYTHON PARA DS.....	6
1.4. IMPORTANDO Y EXPORTANDO DATOS.....	9
1.5. COMENZANDO EL ANÁLISIS DE DATOS.....	13
1.6. ACCEDIENDO A BASES DE DATOS CON PYTHON.....	18
1.7. LABORATORIO.....	21
PARTE II. DATA WRANGLING.....	22
2.1. PRE-PROCESANDO DATOS EN PYTHON.....	22
2.2. TRATANDO CON VALORES FALTANTES EN PYTHON.....	25
2.3. FORMATEO DE DATOS EN PYTHON.....	29
2.4. NORMALIZACIÓN DE DATOS EN PYTHON.....	32
2.5. BINNING EN PYTHON.....	36
2.6. CONVIRTIENDO VARIABLES CATEGÓRICAS EN CUANTITATIVAS.....	43
2.7. LABORATORIO.....	46
2.8. ANEXO.....	47
8.1. Glosario.....	47
8.2. Conceptos.....	47
PARTE III. ANÁLISIS EXPLORATORIO.....	50
3.1. INTRODUCCIÓN.....	50
3.2. ESTADÍSTICA DESCRIPTIVA.....	51
3.3 GROUP BY EN PYTHON.....	59
3.4. CORRELACIÓN.....	63
3.5. CORRELACIÓN - ESTADÍSTICA.....	66
3.6. ANÀLISIS DE LA VARIANZA – ANOVA.....	70
3.7. LABORATORIO.....	76
3.8. ANEXO.....	77
3.8.1. Glosario.....	77
3.8.2. Conceptos.....	77
PARTE IV: MODELO.....	79
4.1. DESARROLLO DEL MODELO.....	79
4.2. REGRESIÓN LINEAL Y REGRESIÓN LINEAL MÚLTIPLE.....	85

4.3. EVALUACIÓN DEL MODELO UTILIZANDO VISUALIZACIÓN.....	96
4.4. REGRESIÓN POLINÓMICA Y PIPELINES.....	102
4.5. MEDIDAS PARA EVALUACIÓN IN-SAMPLE.....	109
4.6. PREDICCIÓN Y TOMA DE DECISIONES.....	114
4.7. LABORATORIO.....	127
PARTE 5. EVALUACIÓN DEL MODELO Y REFINAMIENTO.....	128
5.1. EVALUACIÓN DEL MODELO Y REFINAMIENTO.....	128
5.2. OVERFITTING, UNDERFITTING Y SELECCIÓN DEL MODELO.....	135
5.3. RIDGE REGRESSION.....	149
5.4. GRID SEARCH.....	157

PARTE I. CONCEPTOS BÁSICOS

1.1. EL PROBLEMA

Los datos son recolectados en todas partes, ya sea manualmente o digitalmente por científicos de datos cada vez que se hace clic en un sitio web o en un teléfono móvil.

Pero datos NO significa información.

El análisis de datos nos ayuda a desbloquear la información y los conocimientos de los datos crudos para responder nuestras preguntas.

Entonces, el análisis de datos juega un rol importante ayudándonos a:

- Descubrir información útil.
- Responder preguntas.
- Predecir el futuro o lo desconocido.

Comencemos con nuestro caso de estudio.

Digamos que tenemos un amigo llamado Tom que quiere vender su auto, pero no sabe a cuánto. Quiere venderlo por todo lo que se pueda pero también establecer un precio razonable, así alguien se lo puede comprar. Entonces el precio que establezca debe representar el valor del auto.

Cómo podemos ayudar a Tom a determinar el mejor precio para este auto?

Pensemos como un científico de datos y definamos claramente algunos de los problemas, por ejemplo:

- Hay datos de los precios de otros autos y sus características?
- Qué características de los datos afectan su precio?
 - Color? Marca? Caballos de fuerza? Otro?

Para responder a estas preguntas vamos a necesitar datos.

1.2. ENTENDIENDO LOS DATOS

Estudiaremos el dataset de precios de autos usados suministrado por Jeffrey Schlemmer.

El dataset se encuentra en formato CSV, donde cada línea representa una fila en el dataset.

Observa algo diferente en la primera línea?

A veces la primera línea es un encabezado, que contiene un nombre de columna, pero en este ejemplo es solamente otra fila de datos.

Veamos algunas columnas:

- symboling
 - Corresponde al nivel de riesgo de seguro del auto.
 - A los autos se les asigna inicialmente un símbolo de factor de riesgo asociado con su precio. Luego, si el auto se vuelve más riesgoso, se ajusta moviéndose en la escala.
 - Un valor de + indica que el auto es riesgoso y – que es probablemente bastante seguro.
- Normalized-losses
 - Pago de pérdida promedio relativo por año de vehículo asegurado.
 - El valor está normalizado para todos los autos dentro una clasificación de tamaño particular, 2 puertas pequeñas, camionetas, deportivas, etc. y representa la pérdida promedio por auto por año.
 - Los valores pertenecen al rango entre 65 y 256.

Luego de entender el significado de cada característica, observamos que el número 26 es el precio. Este es nuestro valor objetivo (target) o etiqueta (label). Esto significa que el precio es el valor que queremos predecir a partir del dataset y los predictores son las otras variables (symboling, normalized-losses. Etc.).

Así, el objetivo es predecir el precio en términos de otras características del auto.

1.3. PAQUETES PYTHON PARA DS

Una biblioteca de Python es una colección de funciones y métodos que le permiten realizar muchas acciones sin escribir ningún código.

Hemos dividido las bibliotecas de análisis de datos de Python en tres grupos.

El primer grupo se llama **bibliotecas de computación científica**.

- **Pandas**
 - Ofrece estructura de datos y herramientas para una manipulación y análisis efectivos de datos.
 - Proporciona acceso a datos estructurados.
 - Ofrece estructura de datos y herramientas para una manipulación y análisis efectivos de datos.
 - Está diseñado para proporcionar una funcionalidad de indexación fácil.
- **NumPy**
 - Utiliza matrices para sus entradas y salidas.
 - Se puede extender a objetos para matrices y con cambios menores de codificación, los desarrolladores pueden realizar un procesamiento rápido de matrices.
- **SciPy**
 - Incluye funciones para algunos problemas matemáticos avanzados, así como para visualización de datos.

Usar métodos de visualización de datos es la mejor manera de comunicarse con otros. **Las bibliotecas de visualización** permiten crear gráficos, gráficos y mapas.

- **Matplotlib**
 - Es la biblioteca más conocida para la visualización de datos.
 - Es ideal para hacer gráficos, que también son altamente personalizables.
- **Seaborn**
 - Se basa en Matplotlib.
 - Es fácil generar mapas de calor, series temporales, etc.

Con algoritmos de aprendizaje automático, somos capaces de desarrollar un modelo usando nuestro conjunto de datos y obtener predicciones. Las **bibliotecas algorítmicas** abordan las tareas de aprendizaje automático desde el básico hasta el complejo.

- **Scikit-Learn**

- Contiene herramientas de modelado estadístico, incluyendo regresión, clasificación, clustering, etc.
 - Está construida con NumPy, SciPy y Matplotlib.
- **StatsModel**
 - Permite a los usuarios explorar datos, estimar modelos estadísticos y realizar pruebas estadísticas.

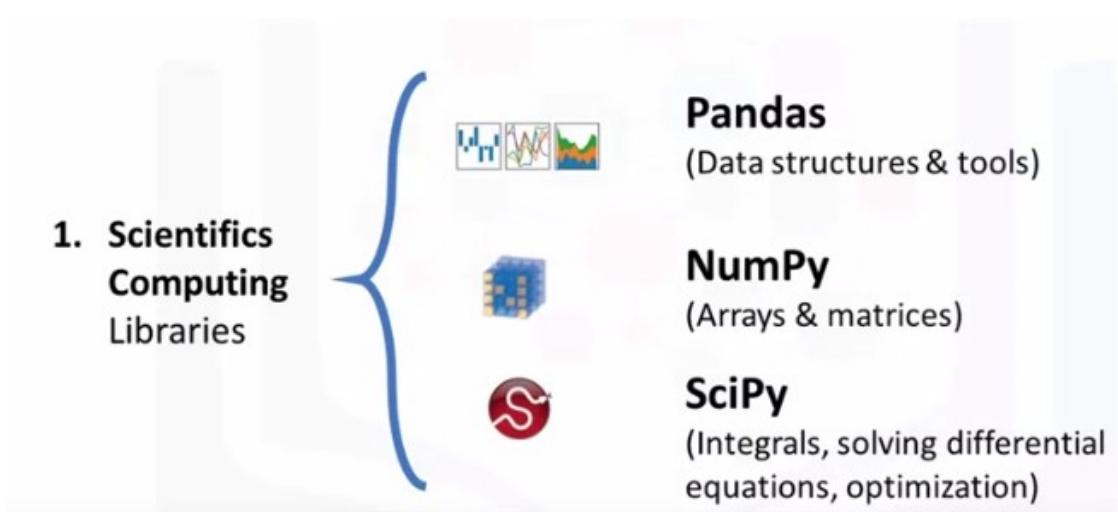


Figura. Bibliotecas de computación científica.

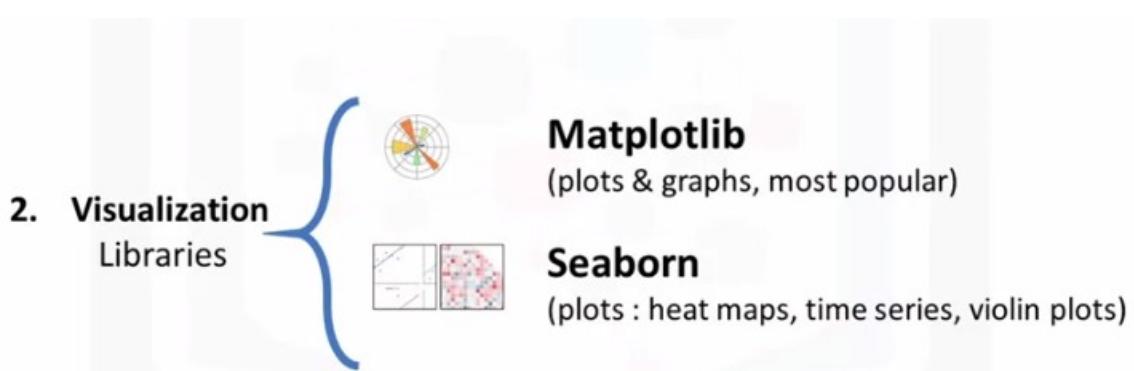


Figura. Bibliotecas de visualización.



Figura. Bibliotecas algorítmicas.

1.4. IMPORTANDO Y EXPORTANDO DATOS

Veremos cómo leer datos usando Pandas.

Una vez que tenemos nuestros datos en Python, podemos proceder a realizar el subsecuente análisis.

La adquisición de datos es el proceso de cargar y leer datos desde varias fuentes.

Para leer datos usando Pandas hay 2 factores importantes a considerar:

- Formato
- File path (camino al archivo)

El formato es la forma en que los datos son codificados. Algunas codificaciones comunes son CSV, JSON, XLSX, etc.

El camino nos dice dónde está almacenado el archivo. Usualmente es en la propia PC que estamos utilizando u online en Internet. En nuestro ejemplo, el dataset es obtenido desde la web.

Cuando se ingresa a la web, puede verse algo así:

```
3,?,alfa-romero,gas,std,two,convertible,rwd,front,86.60,168.80,64.10,48.80,2548,dohc,four,130,mpfi,3.47,2.68,9.00,111,5000,21,27,13495  
3,?,alfa-romero,gas,std,two,convertible,rwd,front,88.60,168.80,64.10,48.80,2548,dohc,four,130,mpfi,3.47,2.68,9.00,111,5000,21,27,16500  
1,?,alfa-romero,gas,std,two,hatchback,rwd,front,94.50,171.20,65.50,52.40,2823,ohcv,six,152,mpfi,2.68,3.47,9.00,154,5000,19,26,13950  
2,164,audi,gas,std,four,sedan,fwd,front,99.80,176.60,66.20,54.30,2337,ohc,four,109,mpfi,3.19,3.40,10.00,102,5500,24,30,13950  
2,164,audi,gas,std,four,sedan,4wd,front,99.40,176.60,66.40,54.30,2824,ohc,five,136,mpfi,3.19,3.40,8.00,115,5500,18,22,17450  
2,?,audi,gas,std,two,sedan,fwd,front,99.80,177.30,66.30,53.10,2507,ohc,five,136,mpfi,3.19,3.40,8.50,110,5500,19,25,15250  
1,158,audi,gas,std,four,sedan,fwd,front,105.80,192.70,71.40,55.70,2844,ohc,five,136,mpfi,3.19,3.40,8.50,110,5500,19,25,17710  
1,?,audi,gas,std,four,wagon,fwd,front,105.80,192.70,71.40,55.70,2954,ohc,five,136,mpfi,3.19,3.40,8.50,110,5500,19,25,18920  
1,158,audi,gas,turbo,four,sedan,fwd,front,105.80,192.70,71.40,55.90,3086,ohc,five,131,mpfi,3.13,3.40,8.30,140,5500,17,20,23875  
0,?,audi,gas,turbo,two,hatchback,4wd,front,99.50,178.20,67.90,52.00,3053,ohc,five,131,mpfi,3.13,3.40,7.00,160,5500,16,22,?  
2,192,bmw,gas,std,two,sedan,rwd,front,101.20,176.80,64.80,54.30,2395,ohc,four,108,mpfi,3.50,2.80,B.80,101,5800,23,29,16430  
0,192,bmw,gas,std,four,sedan,rwd,front,101.20,176.80,64.80,54.30,2395,ohc,four,108,mpfi,3.50,2.80,B.80,101,5800,23,29,16925  
0,188,bmw,gas,std,two,sedan,rwd,front,101.20,176.80,64.80,54.30,2710,ohc,six,164,mpfi,3.31,3.19,9.00,121,4250,21,28,20970  
0,188,bmw,gas,std,four,sedan,rwd,front,101.20,176.80,64.80,54.30,2765,ohc,six,164,mpfi,3.31,3.19,9.00,121,4250,21,28,21105  
1,?,bmw,gas,std,four,sedan,rwd,front,103.50,189.00,66.90,55.70,3055,ohc,six,164,mpfi,3.31,3.19,9.00,121,4250,20,25,24565  
0,?,bmw,gas,std,four,sedan,rwd,front,103.50,189.00,66.90,55.70,3230,ohc,six,209,mpfi,3.62,3.39,8.00,182,5400,16,22,30760  
0,?,bmw,gas,std,two,sedan,rwd,front,103.50,193.80,67.90,53.70,3380,ohc,six,209,mpfi,3.62,3.39,8.00,182,5400,16,22,41315  
0,?,bmw,gas,std,four,sedan,rwd,front,110.00,197.00,70.90,56.30,3505,ohc,six,209,mpfi,3.62,3.39,8.00,182,5400,15,20,36880  
2,121,chevrolet,gas,std,two,hatchback,fwd,front,88.40,141.10,60.30,53.20,1488,1,three,61,2bb1,2.91,3.03,9.50,48,5100,47,53,5151  
1,98,chevrolet,gas,std,two,hatchback,fwd,front,94.50,158.80,63.60,52.00,1874,ohc,four,90,2bb1,3.03,3.11,9.60,70,5400,38,43,6295  
0,81,chevrolet,gas,std,four,sedan,fwd,front,94.50,158.80,63.60,52.00,1909,ohc,four,90,2bb1,3.03,3.11,9.60,70,5400,38,43,6575  
1,118,dodge,gas,std,two,hatchback,fwd,front,93.70,157.30,63.80,50.80,1876,ohc,four,90,2bb1,2.97,3.23,9.41,68,5500,37,41,5572  
1,118,dodge,gas,std,two,hatchback,fwd,front,93.70,157.30,63.80,50.80,1876,ohc,four,90,2bb1,2.97,3.23,9.40,68,5500,31,38,6377  
1,118,dodge,gas,turbo,two,hatchback,fwd,front,93.70,157.30,63.80,50.80,2128,ohc,four,98,mpfi,3.03,3.39,7.60,102,5500,24,30,7957  
1,148,dodge,gas,std,four,hatchback,fwd,front,93.70,157.30,63.80,50.60,1967,ohc,four,90,2bb1,2.97,3.23,9.40,68,5500,31,38,6229  
1,148,dodge,gas,std,four,sedan,fwd,front,93.70,157.30,63.80,50.60,1989,ohc,four,90,2bb1,2.97,3.23,9.40,68,5500,31,38,6692  
1,148,dodge,gas,std,four,sedan,fwd,front,93.70,157.30,63.80,50.60,1989,ohc,four,90,2bb1,2.97,3.23,9.40,68,5500,31,38,7609  
1,148,dodge,gas,turbo,?,sedan,fwd,front,93.70,157.30,63.80,50.60,2191,ohc,four,98,mpfi,3.03,3.39,7.60,102,5500,24,30,8558
```

Figura. Ejemplo de dataset en formato CSV.

Cada fila es un datapoint. Un gran número de propiedades están asociadas con cada datapoint. Como las propiedades están separadas por comas, podemos adivinar que el formato es CSV.

En este punto, lo que se ve son solamente números y no significan mucho para los humanos, pero una vez que leamos estos datos podemos encontrarles un sentido.

En Pandas el método `read_CSV` se utiliza para leer archivos CSV. Tenga presente que `read_csv` asume que los datos contienen un encabezado; como en nuestro caso no lo tienen debe establecerse `header=None`.

```
import pandas as pd  
url = "https://....."  
df = pd.read_csv(url, header=None)
```

Luego de leer el dataset es una buena idea mirar el dataframe para lograr una mejor intuición y asegurarse de que todo ocurra de la forma esperada. Como imprimir todo el dataset puede tomar mucho tiempo y recursos, usamos `dataframe.head` para mostrar las primeras n filas. De modo similar, `dataframe.tail` muestra las filas finales.

En la figura siguiente imprimimos las primeras 5 filas. Parece ser que el dataset fue leído correctamente.

0	1	2	3	4	5	6	7	8	9	...	16	17	18	19	20	21	22	23	24	25	
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450

Figura. Obteniendo las primeras filas del dataset.

Vemos también que Pandas automáticamente estableció la columna de encabezado como una lista de enteros ya que pusimos `header=None` cuando leímos los datos. Es difícil trabajar con dataframes que no tengan nombres de columna que signifiquen algo. Sin embargo, podemos asignarles nombres:

```
headers = ["symboling","normalized-losses","make","fuel-type","aspiration","num-of-doors","body-style",
"drive-wheels","engine-location","wheel-base","length","width","height","curb-weight","engine-type",
"num-of-cylinders","engine-size","fuel-system","bore","stroke","compression-ratio","horsepower","peak-
rpm","city-mpg","highway-mpg","price"]
```

```
df.columns=headers
```

```
df.head(5)
```

symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	hp
0 3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	1
1 3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	1
2 1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	1
3 2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	1
4 2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	1

Figura. Reemplazando los nombres de encabezados.

En algún punto en el tiempo, luego de realizar operaciones sobre su dataset, quizás quiera exportar el dataframe de Pandas a un nuevo CSV. Esto puede lograrse con el método `to_CSV`.

```
path = "C:/...../nombreArchivo.csv"
df.to_csv(path)
```

En este curso trabajaremos con CSV, pero Pandas permite trabajar con muchos formatos de archivo diferentes.

1.5. COMENZANDO EL ANÁLISIS DE DATOS

En este punto asumimos que todos los datos ya han sido cargados, por lo que es hora de explorar el dataset.

Pandas tiene muchos métodos incorporados que pueden ser utilizados para entender las características del dataset. Usar estos métodos nos da un vistazo del dataset y nos ayuda a encontrar posibles problemas, como tipos de datos equivocados que deberán resolverse tarde o temprano.

Los datos tienen una variedad de tipos. Los tipos principales almacenados en objetos Pandas son object, float, int y datetime. Los nombres de los tipos de datos son diferentes de Python nativo, la siguiente tabla muestra diferencias y similaridades entre ellos.

Pandas Type	Native Python Type	Description
object	string	numbers and strings
int64	int	Numeric characters
float64	float	Numeric characters with decimals
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	time data.

Figura. Diferencias entre tipos en Pandas y Python nativo.

Algunos son muy similares, como los tipos numéricos, mientras que el tipo datetime en Pandas es muy útil para trabajar con series temporales.

Hay 2 razones para chequear los tipos de datos en Python:

- Pandas asigna automáticamente tipos basados en la codificación que detecta del datatable original. Por varias razones, esta asignación puede ser incorrecta.
 - Por ejemplo, puede ser incómodo si a la columna precio que se espera contenga números reales se les asigna el tipo de datos object. Sería más natural que el tipo sea float.
- Para ver qué funciones de Python pueden ser aplicadas a una columna específica.
 - Por ejemplo, algunas funciones matemáticas sólo pueden ser aplicadas a datos numéricos, de lo contrario darán error.

Cuando el método dtype es aplicado al dataset, el tipo de datos de cada columna es returnedo en una serie.

```
df.dtypes
```

symboling	int64
normalized-losses	object
make	object
fuel-type	object
aspiration	object
num-of-doors	object
body-style	object
drive-wheels	object
engine-location	object
wheel-base	float64
length	float64
width	float64
height	float64
curb-weight	int64
engine-type	object
num-of-cylinders	object
engine-size	int64
fuel-system	object
bore	object
stroke	object
compression-ratio	float64
horsepower	object
peak-rpm	object
city-mpg	int64
highway-mpg	int64
price	object
dtype:	object

Figura. Obteniendo el tipo de datos de cada columna.

La intuición nos dice que la mayoría de los tipos de datos son correctos, por ejemplo las marcas de autos son nombres, por lo que deberían ser del tipo object. La última fila en la lista podría ser un problema: como que el calibre (bore) es una dimensión de un motor, esperamos un tipo numérico, pero en su lugar es utilizado el tipo object. Habrá que corregir luego estas discrepancias.

Ahora queremos ver un resumen estadístico de cada columna para aprender acerca de la distribución de los datos en ellas. Podemos encontrar por ejemplo outliers o grandes desviaciones, temas que deberán tratarse tarde o temprano.

Para obtener un resumen rápido de estadísticas, usamos el método describe, que retorna:

- El número de términos en la columna como count.
- El valor promedio de la columna como mean.
- La desviación estándar de la columna como std.
- Los valores máximo y mínimo.
- Las fronteras de cada uno de los cuartiles.

```
df.describe()
```



	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-mpg	highway-mpg
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	10.142537	25.219512	30.751220
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	3.972040	6.542142	6.886443
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	7.000000	13.000000	16.000000
25%	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	8.600000	19.000000	25.000000
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	9.000000	24.000000	30.000000
75%	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	9.400000	30.000000	34.000000
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	23.000000	49.000000	54.000000

Figura. Resumen rápido de estadísticas.

Por defecto, dataframe.describe saltea las filas y columnas que no contengan números, pero es posible utilizarlo para tipos de columna object agregando un argumento.

```
df.describe(include="all")
```



	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke
count	205.000000	205	205	205	205	205	205	205	205	205.000000	...	205.000000	205	205	205
unique	NaN	52	22	2	2	3	5	3	2	NaN	...	NaN	8	39	37
top	NaN	?	toyota	gas	std	four	sedan	fwd	front	NaN	...	NaN	mpg	3.62	3.40
freq	NaN	41	32	185	168	114	96	120	202	NaN	...	NaN	94	23	20
mean	0.834146	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	98.756585	...	126.907317	NaN	NaN	NaN
std	1.245307	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.021776	...	41.642693	NaN	NaN	NaN
min	-2.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	36.600000	...	61.000000	NaN	NaN	NaN
25%	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	34.500000	...	97.000000	NaN	NaN	NaN
50%	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	37.000000	...	120.000000	NaN	NaN	NaN
75%	2.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	402.400000	...	141.000000	NaN	NaN	NaN
max	3.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	20.900000	...	326.000000	NaN	NaN	NaN

Figura. Estadística completa.

Vemos que para los tipos de columna object, un conjunto diferente de estadísticas es evaluado, a saber:

- unique
 - Es el número de objetos distintos en la columna.
- Top
 - Es el objeto que ocurre más frecuentemente.
- freq
 - Es el número de veces que aparece el objeto top.

Algunos valores en la tabla se muestran como NaN (Not a Number). Esto se debe a que esa métrica estadística particular no puede ser calculada para ese tipo de columna.

Otro método que puede usar para chequear el dataset es dataframe.info, que muestra las 30 filas iniciales y finales.

1.6. ACCEDIENDO A BASES DE DATOS CON PYTHON

Así es como un usuario típico accede a bases de datos usando código Python escrito en un cuaderno Jupyter, un editor basado en Web. Hay un mecanismo por el cual el programa Python se comunica con el DBMS. El código Python se conecta a la base de datos mediante llamadas a la API.

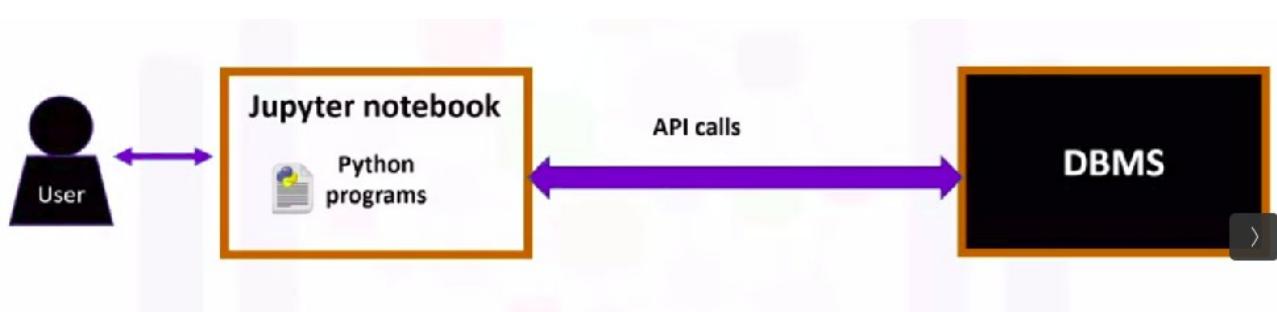


Figura. Conexión a base de datos desde Python.

Una interfaz de programación de aplicaciones (API) es un conjunto de funciones que puede llamar para obtener acceso a algún tipo de servicios.

La API SQL consiste en llamadas a funciones de biblioteca como una interfaz de programación de aplicaciones, API, para el DBMS.

Para pasar sentencias SQL al DBMS, un programa de aplicación llama a funciones en la API, y llama a otras funciones para recuperar los resultados de la consulta y la información de estado del DBMS.

La operación básica de una API SQL típica se ilustra en la figura.

- El programa de aplicación comienza su acceso a la base de datos con una o más llamadas API que conectan el programa con el DBMS.
- Para enviar la instrucción SQL al DBMS, el programa genera la sentencia como una cadena de texto en un búfer y, a continuación, realiza una llamada a la API para pasar el contenido del búfer al DBMS.
- El programa de aplicación realiza llamadas API para comprobar el estado de su solicitud DBMS y para manejar errores.
- El programa de aplicación finaliza su acceso a la base de datos con una llamada API que lo desconecta de la base de datos.

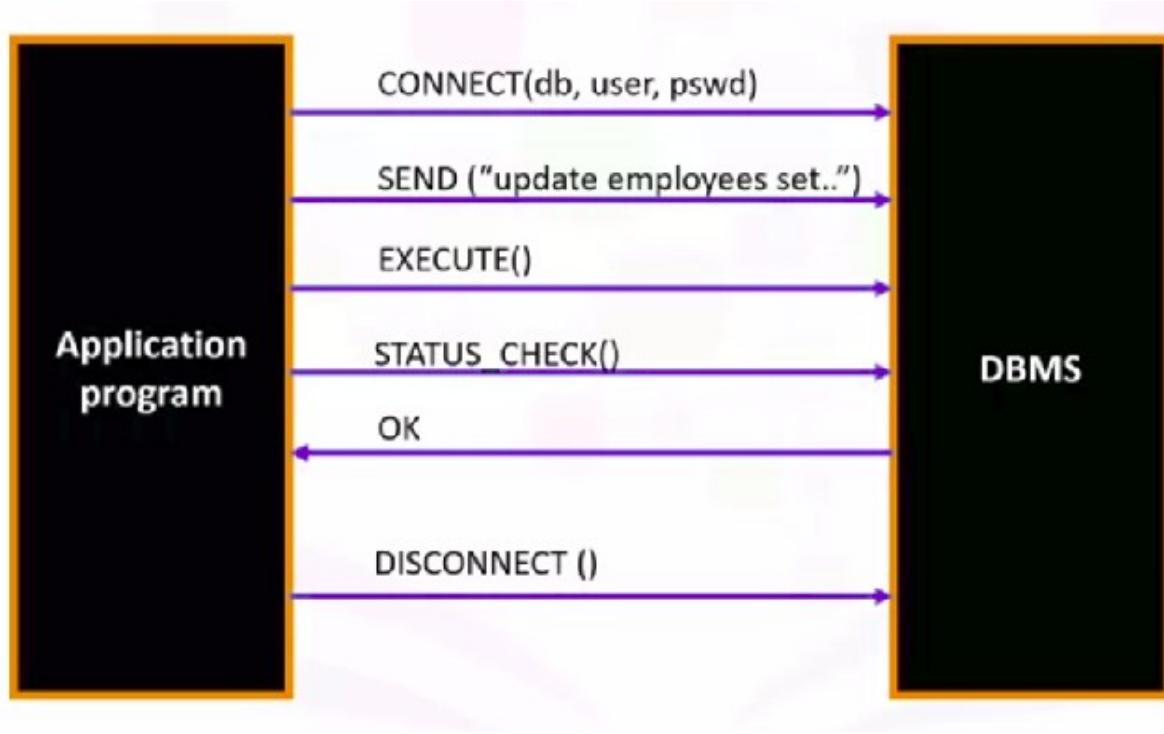


Figura. Operación básica de API SQL.

DB-API es la API estándar de Python para acceder a bases de datos relacionales. Es un estándar que le permite escribir un único programa que funcione con múltiples tipos de bases de datos relacionales en lugar de escribir un programa separado para cada uno. Entonces, si aprende las funciones de DB-API, puede aplicar ese conocimiento para usar cualquier base de datos con Python.

Los dos conceptos principales en Python DB-API son:

- Objetos de conexión
- Objetos de consulta (cursor)

Utilice objetos de conexión para conectarse a una base de datos y administrar sus transacciones.

Los objetos de cursor se utilizan para ejecutar consultas. Se abre un objeto cursor y, a continuación, ejecutar consultas. El cursor funciona de forma similar a un cursor en un sistema de procesamiento de texto donde se desplaza hacia abajo en su conjunto de resultados y obtener sus datos en la aplicación.

Los cursores se utilizan para analizar los resultados de una base de datos. Estos son los métodos utilizados con objetos de conexión.

- cursor () devuelve un nuevo objeto cursor utilizando la conexión.

- `commit()` método se utiliza para confirmar cualquier transacción pendiente a la base de datos.
- `rollback()` hace que la base de datos para volver al inicio de cualquier transacción pendiente.
- `close()` se utiliza para cerrar una conexión a base de datos.

Vamos a recorrer una aplicación de Python que utiliza la DB-API para consultar una base de datos.

- En primer lugar, se importa el módulo de base de datos mediante la API de conexión de ese módulo.
- Para abrir una conexión a la base de datos, utilice la función de conexión y pase los parámetros que son el nombre de la base de datos, el nombre de usuario y la contraseña.
- La función de conexión devuelve el objeto de conexión.
- Después de esto, creará un objeto de cursor en el objeto de conexión.
- El cursor se utiliza para ejecutar consultas y obtener resultados.
- Después de ejecutar las consultas usando el cursor, también usamos el cursor para obtener los resultados de la consulta.
- Finalmente, cuando el sistema termina de ejecutar las consultas, libera todos los recursos cerrando la conexión.
 - Recuerde que siempre es importante cerrar las conexiones para evitar que las conexiones no utilizadas tomen recursos.

```
from dbmodule import connect
# creo un objeto conexión
connection = connect ('databasename','user','password')
# creo un objeto cursor
cursor = connection.cursor()
# ejecuto consultas
cursor.execute('select * from mytable')
results = cursor.fetchall()
# libero recursos
cursor.close()
connection.close()
```

1.7. LABORATORIO

Notebook: 1. review-introduction

PARTE II. DATA WRANGLING

2.1. PRE-PROCESANDO DATOS EN PYTHON

El **preprocesamiento de datos** es un paso necesario en el análisis de datos. Es el proceso de convertir o mapear datos de una forma cruda (raw) a otro formato para que esté listo para su análisis posterior.

El preprocesamiento de datos a menudo se llama **limpieza de datos o disputa de datos (data wrangling)**.

En esta parte:

- Veremos cómo identificar y manejar los valores que faltan.
 - Se produce una condición de valor faltante cada vez que una entrada de datos se deja vacía.
- Estudiaremos los formatos de datos.
 - Datos de diferentes fuentes tal vez en varios formatos, en diferentes unidades, o en varias convenciones.
 - Vamos a introducir algunos métodos en Python Pandas que pueden estandarizar los valores en el mismo formato, o unidad, o convención.
- Cubriremos la normalización de datos.
 - Las diferentes columnas de datos numéricos pueden tener rangos muy diferentes y comparación directa a menudo no es significativa. La normalización es una forma de traer todos los datos a un rango similar para una comparación más útil.
 - Específicamente, nos centraremos en las técnicas de centrado y escalado.
- Introduciremos rangos de datos.
 - El binning crea categorías más grandes a partir de un conjunto de valores numéricos.
 - Es particularmente útil para la comparación entre grupos de datos.
- Hablaremos de variables categóricas y le mostraremos cómo convertir valores categóricos en variables numéricas para facilitar el modelado estadístico.

En Python, generalmente realizamos operaciones a lo largo de columnas. Cada fila de la columna representa una muestra, en nuestro ejemplo, un coche usado diferente en la base de datos. Para acceder a una columna, especifique el nombre de la columna. Por ejemplo, puede acceder a “symboling” y “body-style”.

Simple Dataframe Operations

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0
5	2	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5
6	1	158	audi	gas	std	four	sedan	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5
7	1	?	audi	gas	std	four	wagon	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5
8	1	158	audi	gas	turbo	four	sedan	fwd	front	105.8	...	131	mpfi	3.13	3.40	8.3
9	0	?	audi	gas	turbo	two	hatchback	4wd	front	99.5	...	131	mpfi	3.13	3.40	7.0

Each of these columns is a Panda series.

IBM Developer



Figura. Accediendo a columnas en Python.

Cada una de estas columnas es una **serie**.

Hay muchas maneras de manipular Dataframes en Python. Por ejemplo, puede agregar un valor a cada entrada de una columna. Para agregar uno a cada entrada de “symboling”, utilice el comando de la figura siguiente. Esto cambia cada valor de la columna “symboling” agregando uno al valor actual.

Simple Dataframe Operations

```
df["symboling"] = df["symboling"] + 1
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	I
0	4	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	-
1	4	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	-
2	2	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	-
3	3	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	-
4	3	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	-
5	3	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5	-
6	2	158	audi	gas	std	four	sedan	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5	-
7	2	?	audi	gas	std	four	wagon	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5	-
8	2	158	audi	gas	turbo	four	sedan	fwd	front	105.8	...	131	mpfi	3.13	3.40	8.3	-
9	1	?	audi	gas	turbo	two	hatchback	4wd	front	99.5	...	131	mpfi	3.13	3.40	7.0	-

This changes each value of

IBM Developer

SKILLS NETWORK

Figura. Manipulando datos en Python.

2.2. TRATANDO CON VALORES FALTANTES EN PYTHON

Vamos a presentar el problema generalizado de valores faltantes, así como estrategias sobre qué hacer cuando encuentre valores faltantes en sus datos.

Cuando no se almacena ningún valor de datos para la entidad para una observación en particular, decimos que esta característica tiene un valor que falta.

Normalmente, el valor que falta en el conjunto de datos aparece como signo de interrogación y un cero o simplemente una celda en blanco. En el ejemplo aquí, la característica de pérdidas normalizadas tiene un valor faltante que se representa con NaN.

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location
0	3	NaN ←	alfa-romero	gas	std	two	convertible	rwd	front

Figura. Valor faltante.

¿Cómo puedes lidiar con los datos perdidos?

Hay muchas maneras de lidiar con los valores que faltan y esto es independientemente de Python, R o cualquier herramienta que use.

Por supuesto, cada situación es diferente y debe ser juzgada de manera diferente. Sin embargo, estas son las **opciones típicas** que puede considerar.

- Verificar si la persona o grupo que recogió los datos pueden volver atrás y encontrar cuál debería ser el valor real.
- Eliminar los datos donde se encuentra ese valor perdido.
 - Cuando elimina datos, puede eliminar toda la variable o simplemente la única entrada de datos con el valor que falta. Si no tiene muchas observaciones con datos faltantes, generalmente eliminar la entrada en particular es la mejor.
- Si está eliminando datos, desea hacer algo que tenga la menor cantidad de impacto. Reemplazar datos es mejor ya que no se desperdicia ningún dato. Sin embargo, es menos preciso ya que necesitamos reemplazar los datos faltantes con una conjetura de cuáles deberían ser los datos.
 - Un estándar para la técnica de colocación es reemplazar valores faltantes por el valor promedio de toda la variable.

- Como ejemplo, supongamos que tenemos algunas entradas que tienen valores faltantes para la columna de pérdidas normalizadas y el promedio de columna para entradas con datos es 4500. Si bien no hay manera de obtener una conjectura exacta de lo que el valor que falta en la columna de pérdidas normalizadas debería haber sido, puede aproximar sus valores usando el valor promedio de la columna 4500.
- Pero, ¿y si los valores no se pueden promediar como con variables categóricas?
 - Para una variable como el tipo de combustible, no hay un tipo de combustible promedio, ya que los valores de las variables no son números. En este caso, una posibilidad es intentar usar el modo, el más común como la gasolina.
- A veces podemos encontrar otra manera de adivinar los datos que faltan. Esto suele deberse a que los datos recopilados saben algo adicional sobre los datos que faltan. Por ejemplo, él puede saber que los valores faltantes tienden a ser coches viejos y las pérdidas normalizadas de coches viejos son significativamente más altos que el vehículo promedio.
- En algunos casos, es posible que simplemente desee dejar los datos faltantes como datos faltantes.
 - Por una razón u otra, puede ser útil mantener esa observación incluso si faltan algunas características.

Ahora, vamos a ver cómo eliminar los valores que faltan o reemplazar los valores que faltan en Python.

Para eliminar datos que contienen valores faltantes la biblioteca Pandas tiene un método incorporado llamado dropna. Esencialmente, con el método dropna, puede elegir eliminar filas o columnas que contienen valores faltantes como NaN. Por lo tanto, deberá especificar acceso igual a cero para eliminar las filas o acceso igual a uno para eliminar las columnas que contienen los valores faltantes.

En este ejemplo, falta un valor en la columna de precio. Dado que el precio de los coches usados es lo que estamos tratando de predecir en nuestro próximo análisis, tenemos que eliminar los coches, las filas, que no tienen un precio listado.

Simplemente se puede hacer en una línea de código usando dataframe.dropna. Al establecer el argumento “inplace” en true, la modificación se realiza directamente en el conjunto de datos.

- Use `dataframes.dropna()` :

highway-mpg	price
...	...
20	23875
22	NaN
29	16430
...	...

→

highway-mpg	price
...	...
20	23875
29	16430
...	...

axis=0 drops the entire row
axis=1 drops the entire column

```
df.dropna(subset=["price"], axis=0, inplace = True)  
df = df.dropna(subset=["price"], axis=0)
```

Figura. Eliminando datos faltantes.

Para reemplazar valores faltantes como NaNs con valores reales, Pandas tiene un método incorporado llamado `replace` que se puede usar para llenar los valores faltantes con los valores recién calculados.

Como ejemplo, supongamos que queremos reemplazar los valores faltantes de la variable normalizada pérdidas por el valor medio de la variable. Por lo tanto, el valor que falta debe ser reemplazado por el promedio de las entradas dentro de esa columna.

En Python, primero calculamos la media de la columna. Luego usamos el método `replace` to especificar el valor que nos gustaría ser reemplazado como el primer parámetro, en este caso NaN. El segundo parámetro es el valor que nos gustaría reemplazarlo con, es decir, la media en este ejemplo.

Use `dataframe.replace(missing_value, new_value)`:

The diagram illustrates the process of replacing missing values in a DataFrame. On the left, a table shows a row with a missing value ('NaN') highlighted in yellow. An arrow points from this row to the right, where the same table is shown with the missing value replaced by the mean value (162), also highlighted in yellow.

normalized-losses	make
...	...
164	audi
164	audi
NaN	audi
158	audi
...	...

→

normalized-losses	make
...	...
164	audi
164	audi
162	audi
158	audi
...	...

```
mean = df["normalized-losses"].mean()  
df["normalized-losses"].replace(np.nan, mean)
```

Figura. Remplazando valores faltantes.

2.3. FORMATEO DE DATOS EN PYTHON

Vamos a ver el problema de los datos con diferentes formatos, unidades y convenciones y los métodos pandas que nos ayudan a lidiar con estos problemas.

Los datos suelen ser recogidos de diferentes lugares por personas diferentes que pueden ser almacenados en diferentes formatos. El formato de datos significa llevar los datos a un estándar común de expresión que permite a los usuarios realizar comparaciones significativas.

Como parte de la limpieza de conjuntos de datos, el formato de datos garantiza que los datos sean consistentes y fácilmente comprensibles. Por ejemplo, las personas pueden usar diferentes expresiones para representar la ciudad de Nueva York, como N mayúscula Y, mayúscula N minúscula y, N mayúscula Y y Nueva York. A veces, estos datos impuros son algo bueno para ver. Por ejemplo, si está mirando las diferentes formas en que la gente tiende a escribir Nueva York, , entonces estos son exactamente los datos que desea. O si usted está buscando maneras de detectar el fraude, tal vez escribir N.Y. es más probable que prediga una anomalía que si alguien escribió Nueva York en su totalidad. Pero quizás más a menudo que no, simplemente queremos tratarlos a todos como la misma entidad o formato para facilitar los análisis estadísticos en el futuro.

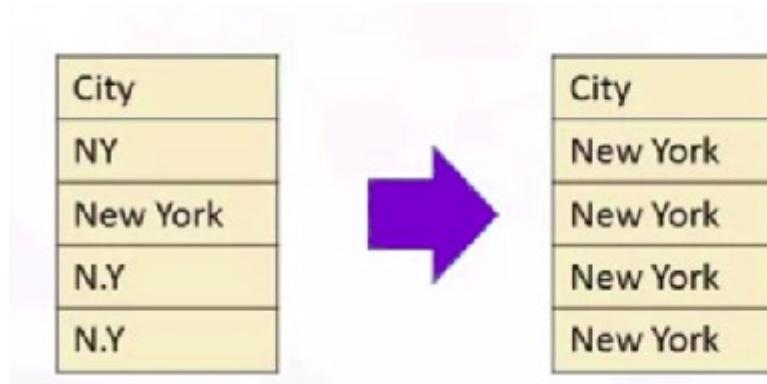


Figura. Unificando la forma de referirse a New York.

Refiriéndose a nuestro conjunto de datos de vehículos usados, hay una característica llamada ciudad-millas por galón en el conjunto de datos, que se refiere a un consumo de combustible de automóviles en millas por galón. Sin embargo, puede ser alguien que vive en un país que utiliza unidades métricas. Por lo tanto, le gustaría convertir esos valores a litros por 100 kilómetros, la versión métrica.

Para transformar millas por galón en litros por 100 kilómetros, tenemos que dividir 235 por cada valor en la columna ciudad-millas por galón. En Python, esto se puede hacer

fácilmente. Usted toma la columna y la establece en igual a 235, divídela por toda la columna. En la segunda línea de código, cambie el nombre de la columna de millas por galón a ciudades-litros por 100 kilómetros utilizando el método de cambio de nombre del marco de datos.

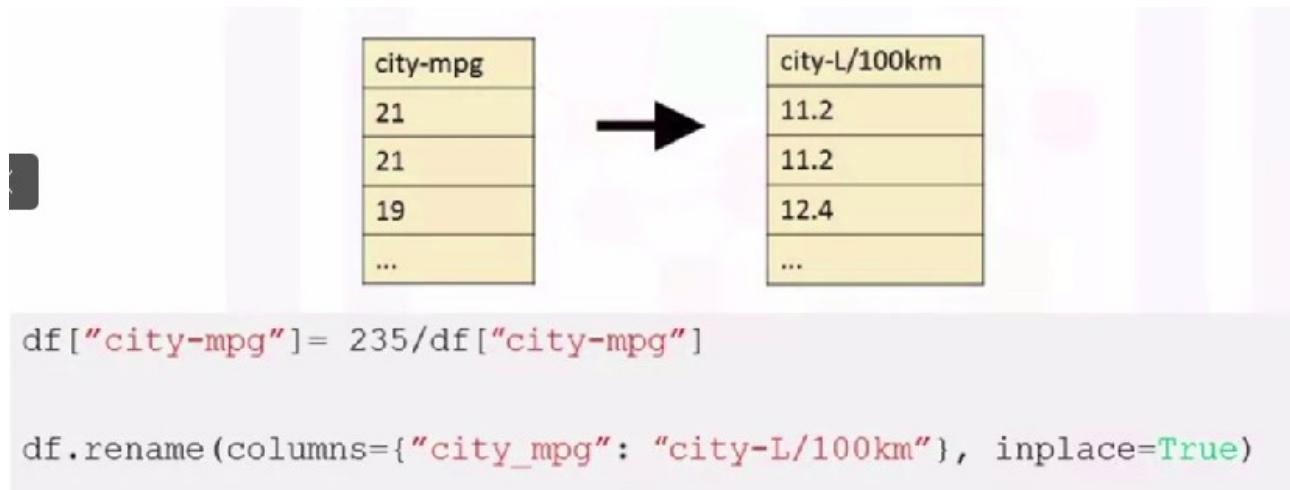


Figura. Convirtiendo datos.

Por varias razones, incluyendo cuando se importa un conjunto de datos en Python, el tipo de datos puede estar incorrectamente establecido. Por ejemplo, en la figura siguiente notamos que el tipo de datos asignado a la característica de precio es objeto aunque el tipo de datos esperado realmente debería ser un entero o tipo flotante.

```
df["price"].tail(5)

200      16845
201      19045
202      21485
203      22470
204      22625
Name: price, dtype: object
```

Figura. Tipo de datos incorrectamente establecido.

Es importante para un análisis posterior explorar el tipo de datos de entidades y convertirlas a los tipos de datos correctos. De lo contrario, los modelos desarrollados más adelante pueden comportarse extrañamente, y los datos totalmente válidos pueden terminar siendo tratados como datos faltantes.

Hay muchos tipos de datos en pandas. Para identificar entidades tipo de datos, en Python podemos utilizar el método `dataframe.dtypes` y verificar el tipo de datos de cada variable en un marco de datos. En el caso de tipos de datos incorrectos, el método `dataframe.astype` puede ser utilizado para convertir un tipo de datos de un formato a otro. Por ejemplo, usando `astype int` para la columna `price`, puede convertir la columna `object` en una variable de tipo entero.

```
df['price'] = df['price'].astype("int")
```

2.4. NORMALIZACIÓN DE DATOS EN PYTHON

Hablaremos de la normalización de datos, una técnica importante a entender en el preprocesamiento de datos.

Cuando echamos un vistazo al conjunto de datos del coche usado, notamos en los datos que la longitud de la entidad oscila entre 150-250, mientras que la anchura y la altura de la entidad oscila entre 50-100. Es posible que queramos normalizar estas variables para que el rango de los valores sea consistente. Esta normalización puede facilitar algunos análisis estadísticos. Al hacer que los rangos sean consistentes entre variables, la normalización permite una comparación justa entre las diferentes características, asegurándose de que tengan el mismo impacto. También es importante por razones computacionales.

length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
171.2	65.5	52.4
176.6	66.2	54.3
176.6	66.4	54.3
177.3	66.3	53.1
192.7	71.4	55.7
192.7	71.4	55.7
192.7	71.4	55.9

scale	[150,250]	[50,100]	[50,100]
impact	large	small	small

Figura. Normalización de variables.

Aquí hay otro ejemplo que le ayudará a entender por qué la normalización es importante. Considere un conjunto de datos que contenga dos características, edad e ingresos. Donde la edad oscila entre 0-100, mientras que los ingresos oscilan entre 0 y 20.000 y más. El ingreso es aproximadamente 1.000 veces mayor que la edad y oscila entre 20.000 y 500.000. Por lo tanto, estas dos características están en rangos muy diferentes. Cuando hacemos más análisis, como regresión lineal por ejemplo, el ingreso del atributo intrínsecamente influirá más en el resultado debido a su mayor valor. Pero esto no significa necesariamente que sea más importante como predictor. Entonces, la naturaleza de los datos sesga el modelo de regresión lineal para pesar los ingresos más fuertemente que la edad. Para evitar esto, podemos normalizar estas dos variables en valores que van de cero a uno. Compare las dos tablas a la derecha. Después de la normalización, ambas variables ahora tienen una influencia similar en los modelos que construiremos más adelante.

age	income
20	100000
30	20000
40	500000

age	income
0.2	0.2
0.3	0.04
0.4	1

Figura. Importancia de la normalización.

Hay varias maneras de normalizar los datos, veamos 3 técnicas:

- El primer método llamado escala de entidad simple solo divide cada valor por el valor máximo para esa entidad.
 - Esto hace que los nuevos valores oscilen entre cero y uno.
- El segundo método llamado min-max toma cada valor x_{old} lo resta del valor mínimo de esa entidad, luego se divide por el rango de esa entidad.
 - De nuevo, los nuevos valores resultantes oscilan entre cero y uno.
- El tercer método se llama puntuación z o puntuación estándar. En esta fórmula para cada valor se resta el μ que es el promedio de la entidad, y luego se divide por la desviación estándar σ .
 - Los valores resultantes se desplazan alrededor de cero, y normalmente oscilan entre -3 y +3, pero pueden ser más altos o más bajos.

①

$$x_{new} = \frac{x_{old}}{x_{max}}$$

Simple Feature scaling

②

$$x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$$

Min-Max

③

$$x_{new} = \frac{x_{old} - \mu}{\sigma}$$

Z-score

Figura. Métodos de normalización de datos.

Siguiendo nuestro ejemplo anterior, podemos aplicar el método de normalización en la entidad de longitud.

En primer lugar, usamos el método simple de escala de entidades, donde lo dividimos por el valor máximo de la entidad. Usando el método pandas max, esto se puede hacer en una sola línea de código.

The diagram illustrates the scaling process. On the left, there is a table with columns: length, width, and height. The first two rows have values 168.8, 64.1, 48.8 respectively. The third row has values 180.0, 65.5, 52.4. The fourth row is indicated by three dots. An arrow points to the right, leading to another table with the same structure. In this second table, the first two rows now have values 0.81, 64.1, 48.8 respectively. The third row has values 0.87, 65.5, 52.4. The fourth row is indicated by three dots.

length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
180.0	65.5	52.4
...

→

length	width	height
0.81	64.1	48.8
0.81	64.1	48.8
0.87	65.5	52.4
...

```
df["length"] = df["length"]/df["length"].max()
```

Aquí está el método min-max en la entidad de longitud.

The diagram illustrates the min-max scaling process. On the left, there is a table with columns: length, width, and height. The first two rows have values 168.8, 64.1, 48.8 respectively. The third row has values 180.0, 65.5, 52.4. The fourth row is indicated by three dots. An arrow points to the right, leading to another table with the same structure. In this second table, the first two rows now have values 0.41, 64.1, 48.8 respectively. The third row has values 0.58, 65.5, 52.4. The fourth row is indicated by three dots.

length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
180.0	65.5	52.4
...

→

length	width	height
0.41	64.1	48.8
0.41	64.1	48.8
0.58	65.5	52.4
...

```
df["length"] = (df["length"]-df["length"].min())/(df["length"].max()-df["length"].min())
```

Finalmente, aplicamos el método z-score en la característica de longitud para normalizar los valores. Aquí aplicamos los métodos mean y std en la entidad de longitud.

El método mean devolverá el valor promedio de la entidad en el conjunto de datos, y el método std devolverá la desviación estándar de las entidades en el conjunto de datos.

length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
180.0	65.5	52.4
...



length	width	height
-0.034	64.1	48.8
-0.034	64.1	48.8
0.039	65.5	52.4
...

```
df["length"] = (df["length"] - df["length"].mean()) / df["length"].std()
```

2.5. BINNING EN PYTHON

Hablaremos sobre el binning como método de preprocesamiento de datos

Binning es cuando se agrupan valores en bins (contenedores). Por ejemplo, puede colocar «edad» en [0 a 5], [6 a 10], [11 a 15], etc.

A veces, el binning puede mejorar la precisión de los modelos predictivos.

Además, a veces usamos rangos de datos para agrupar un conjunto de valores numéricos en un número menor de bins para tener una mejor comprensión de la distribución de datos.

Como ejemplo, «price» aquí es un rango de atributos de 5.000 a 45.500. Usando rangos, clasificamos el precio en tres rangos: precio bajo, precio medio y precios altos.

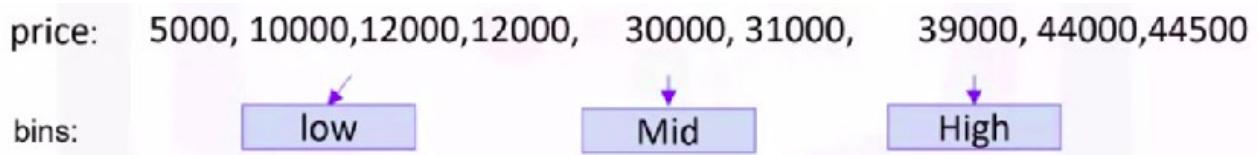


Figura. Clasificación del precio en rangos.

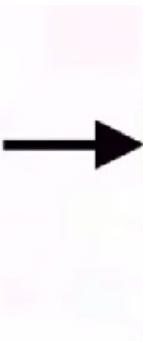
En el conjunto de datos del automóvil real, «precio» es una variable numérica que va desde 5188 a 45400, tiene 201 valores únicos. Podemos clasificarlos en 3 compartimentos: coches de bajo, medio y alto precio.

The diagram shows the transformation of raw car prices into binned categories. On the left, a table titled 'price' lists several car prices: 13495, 16500, 18920, 41315, 5151, 6295, and an ellipsis. An arrow points from this table to a second table on the right. The second table is titled 'price-binned' and shows the same list of prices with their corresponding binned categories: Low, Low, Medium, High, Low, Low, and an ellipsis. The 'Low' category covers the range from 5188 to approximately 18,000. The 'Medium' category covers the range from approximately 18,000 to 35,000. The 'High' category covers the range from 35,000 to 45,400.

price	price-binned
13495	Low
16500	Low
18920	Medium
41315	High
5151	Low
6295	Low
...	...

Figura. Clasificación del precio de autos.

En Python podemos implementar fácilmente el binning: nos gustaría 3 bins de igual ancho de binado, por lo que necesitamos 4 números como divisores que estén a igual distancia. Primero usamos la función numpy «linspace» para devolver la matriz «bins» que contiene 4 números igualmente espaciados en el intervalo especificado del precio. Creamos una lista «group_names» que contiene los diferentes nombres de bin. Utilizamos la función pandas «cortar» para segmentar y ordenar los valores de datos en bins.



price	price-binned
13495	Low
16500	Low
18920	Medium
41315	High
5151	Low
6295	Low
...	...

```

bins = np.linspace(min(df["price"]), max(df["price"]), 4)
group_names = ["Low", "Medium", "High"]
df[“price-binned”] = pd.cut(df[“price”], bins, labels=group_names, include_lowest=True )

```

Figura. Binning en Pandas.

A continuación, puede utilizar histogramas para visualizar la distribución de los datos después de que hayan sido divididos en bins. El siguiente es el histograma que trazamos basado en el rango que aplicamos en la característica price . De la trama, está claro que la mayoría de los coches tienen un precio bajo, y sólo muy pocos coches tienen precio alto.

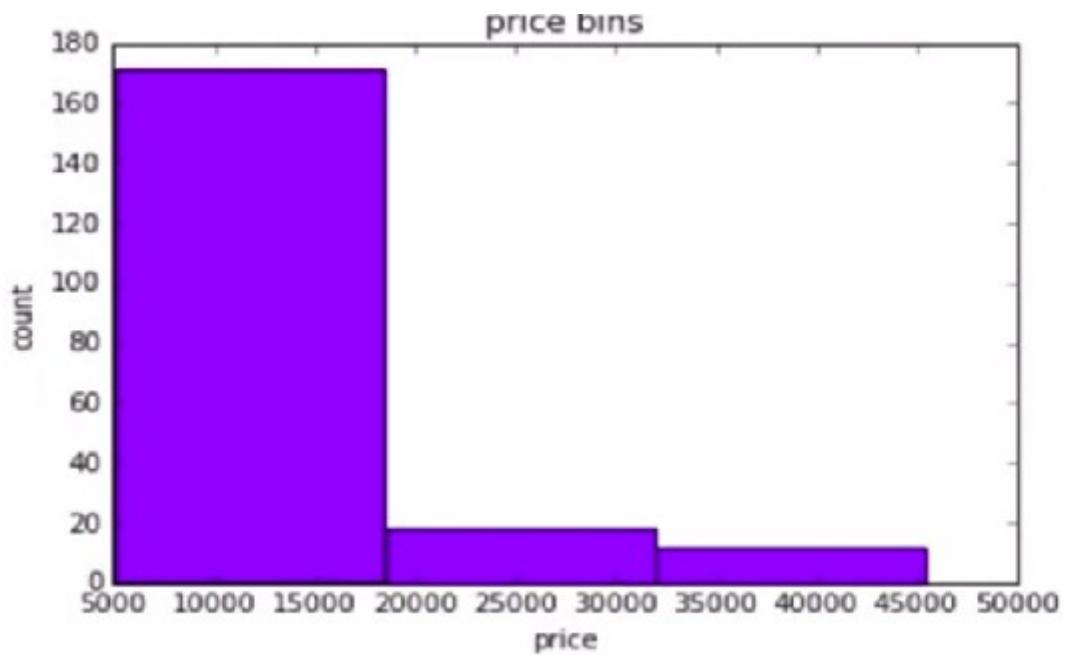


Figura. Histograma para el precio de los autos.

2.6. CONVIRTIENDO VARIABLES CATEGORICAS EN CUANTITATIVAS

Vamos a discutir cómo convertir variables categóricas en variables cuantitativas en Python. La mayoría de los modelos estadísticos no pueden tomar objetos o cadenas como entrada y para el entrenamiento del modelo solo toman los números como entradas.

En el conjunto de datos del coche, la característica de tipo de combustible como variable categórica tiene dos valores, gas o diesel, que están en formato de cadena. Para mayor análisis, se tienen que convertir estas variables en alguna forma de formato numérico. Codificamos los valores añadiendo nuevas características correspondientes a cada elemento único de la característica original que nos gustaría codificar.

En el caso en que la característica combustible tiene dos valores únicos, gas y diesel, creamos dos nuevas características, gas y diesel. Cuando se produce un valor en la entidad original, establecemos el valor correspondiente en uno en la nueva entidad. El resto de las características se establece en cero.

Figura.

Car	Fuel	...	gas	diesel
A	gas	...	1	0
B	diesel	...	0	1
C	gas	...	1	0
D	gas	...	1	0

“One-hot encoding”

Codificación en la característica “combustible”.

En el ejemplo de combustible, para el coche B, el valor del combustible es diesel. Por lo tanto, establecemos la característica diesel igual a uno y la entidad de gas a cero.

Car	Fuel	...	gas	diesel
A	gas	...	1	0
B	diesel	...	0	1
C	gas	...	1	0
D	gas	...	1	0

Figura. Ejemplo de codificación.

Esta técnica se llama a menudo **one-hot encoding**.

En Pandas, podemos utilizar get_dummies método para convertir variables categóricas en cuantitativas.

El método get_dummies genera automáticamente una lista de números, cada uno correspondiente a una categoría particular de la variable.

The diagram illustrates the one-hot encoding process. On the left, a vertical table shows the 'fuel' column with five rows: 'gas', 'diesel', 'gas', 'gas', and 'gas'. An arrow points from this table to the right, where another table shows two columns: 'gas' and 'diesel'. The 'gas' column has values 1, 0, 1, 1, and 1 respectively. The 'diesel' column has values 0, 1, 0, 0, and 0 respectively. This represents a binary encoding where each category ('gas' or 'diesel') is represented by a single column with a value of 1 if the row belongs to that category and 0 otherwise.

fuel
gas
diesel
gas
gas
gas

gas	diesel
1	0
0	1
1	0
1	0

```
pd.get_dummies(df['fuel'])
```

Figura. Convirtiendo variables categóricas en cuantitativas en Pandas.

2.7. LABORATORIO

Notebook: 2. data-wrangling

2.8. ANEXO.

8.1. Glosario

1.Preprocesamiento de datos.

A veces llamado data wrangling es el proceso de convertir o mapear datos de una forma cruda (raw) a otro formato para que esté listo para su análisis posterior.

2. Binning.

Es cuando se agrupan valores en bins (contenedores). Por ejemplo, puede colocar «edad» en [0 a 5], [6 a 10], [11 a 15], etc.

8.2. Conceptos

1. Qué hacer ante valores faltantes?

- Verificar si la persona o grupo que recogió los datos pueden volver atrás y encontrar cuál debería ser el valor real.
- Eliminar los datos donde se encuentra ese valor perdido.
 - Cuando elimina datos, puede eliminar toda la variable o simplemente la única entrada de datos con el valor que falta. Si no tiene muchas observaciones con datos faltantes, generalmente eliminar la entrada en particular es la mejor.
- Si está eliminando datos, desea hacer algo que tenga la menor cantidad de impacto. Reemplazar datos es mejor ya que no se desperdicia ningún dato. Sin embargo, es menos preciso ya que necesitamos reemplazar los datos faltantes con una conjetura de cuáles deberían ser los datos.
 - Un estándar para la técnica de colocación es reemplazar valores faltantes por el valor promedio de toda la variable.
 - Como ejemplo, supongamos que tenemos algunas entradas que tienen valores faltantes para la columna de pérdidas normalizadas y el promedio de columna para entradas con datos es 4500. Si bien no hay manera de obtener una conjetura exacta de lo que el valor que falta en la columna de pérdidas normalizadas debería haber sido, puede aproximar sus valores usando el valor promedio de la columna 4500.
 - Pero, ¿y si los valores no se pueden promediar como con variables categóricas?
 - Para una variable como el tipo de combustible, no hay un tipo de combustible promedio, ya que los valores de las variables no son números. En este caso, una posibilidad es intentar usar el modo, el más común como la gasolina.

- A veces podemos encontrar otra manera de adivinar los datos que faltan. Esto suele deberse a que los datos recopilados saben algo adicional sobre los datos que faltan. Por ejemplo, él puede saber que los valores faltantes tienden a ser coches viejos y las pérdidas normalizadas de coches viejos son significativamente más altos que el vehículo promedio.
- En algunos casos, es posible que simplemente desee dejar los datos faltantes como datos faltantes.
 - Por una razón u otra, puede ser útil mantener esa observación incluso si faltan algunas características.

2. Técnicas de normalización

- SFS
- Min-Max
- Z-Score

8.3. Códigos

DESCRIPCIÓN	CÓDIGO
Acceder a columna de dataframe (symboling df[“symboling”] en el ejemplo)	
Manipular columna del dataframe (sumo 1 df[“symboling”] = df[“symboling”] + 1 en el ejemplo)	
Borrar datos faltantes.	df.dropna(subset=[“price”], axis=0, inplace=True)
	axis = 0: borra toda la fila axis = 1: borra toda la columna. inplace = True: modifica directamente el conjunto de datos.
Reemplazar datos faltantes (por la media en el ejemplo).	mean = df[“normalized-losses”].mean() df[“normalized-losses”].replaces(np.nan, mean)

Renombrar una columna.

```
df.rename(columns = {"city-mpg": "city-  
mpg/100kmh"}, inplace=True)
```

Tipos de datos.

```
dataframe.dtypes
```

Convertir tipo de datos (a entero en el
ejemplo).

```
df['price'] = df['price'].astype("int")
```

Binning (divido en 3 bins en el ejemplo).

```
bins = np.linspace(min(df["price"]),  
max(df["price"]), 4)
```

```
group_names = ["Low", "Medium", "High"]
```

```
df["price-binned"] = pd.cut(df["price"], bins  
labels=group_names,  
include_lowest=True)
```

Convertir variables categóricas en
cuantitativas.

```
pd.get_dummies(df['fuel'])
```

PARTE III. ANÁLISIS EXPLORATORIO.

3.1. INTRODUCCIÓN

En este módulo, vamos a cubrir los fundamentos del **Análisis Exploratorio de Datos EDA** usando Python.

EDA es un enfoque para:

- Analizar datos en con el fin de resumir las principales características de los mismos.
- Obtener una mejor comprensión del conjunto de datos.
- Descubrir relaciones entre diferentes variables.
- Extraer variables importantes para el problema que estamos tratando de resolver.

La pregunta principal que estamos tratando de responder en este módulo es, **¿cuáles son las características que más impactaron en el precio del automóvil?**

Aprenderemos:

- Estadísticas descriptivas, que describen las características básicas de un conjunto de datos.
- A agrupar datos utilizando GroupBy, y cómo esto puede ayudar a transformar nuestro conjunto de datos.
- ANOVA, el análisis de varianza, un método estadístico en el que la variación en un conjunto de observaciones se divide en componentes distintos.
- La correlación entre diferentes variables.
- Correlación avanzada, en particular:
 - Correlación de Pearson.
 - Mapas de calor de correlación.

3.2. ESTADÍSTICA DESCRIPTIVA

Estudiaremos la estadística descriptiva.

Cuando se comienza a analizar los datos, es importante explorarlos primero antes de pasar tiempo construyendo modelos complicados. Una forma de hacer esto es la estadística descriptiva.

La estadística descriptiva ayuda a describir características básicas del conjunto de datos y obtener un breve resumen acerca de la muestra y de medidas en los datos.

Veamos un par de métodos útiles.

Podemos utilizar la función `describe` de Pandas, que computa automáticamente estadísticas básica para todas las variables numéricas. En otras cosas muestra:

- La media
- El número total de puntos de datos.
- La desviación estándar.
- Los cuartiles.
- Los valores extremos.

Los valores NaN son automáticamente ignorados.

Esta función da una idea de la distribución de las diferentes variables.

```
df.describe()
```

	Unnamed: 0	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke
count	201.000000	201.000000	164.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000
mean	100.000000	0.840796	122.000000	96.797015	174.200995	65.889055	53.766667	2555.666667	126.875622	3.319154	3.256766
std	58.167861	1.254802	35.442168	6.066366	12.322175	2.101471	2.447822	517.296727	41.546834	0.280130	0.316049
min	0.000000	-2.000000	65.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000
25%	50.000000	0.000000	NaN	94.500000	166.800000	64.100000	52.000000	2169.000000	98.000000	3.150000	3.110000
50%	100.000000	1.000000	NaN	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000
75%	150.000000	2.000000	NaN	102.400000	183.500000	66.600000	55.500000	2926.000000	141.000000	3.580000	3.410000
max	200.000000	3.000000	256.000000	120.900000	208.100000	72.000000	59.800000	4066.000000	326.000000	3.940000	4.170000

Figura. Función `describe` en Pandas.

También puede haber variables categóricas en su dataset. Estas variables pueden ser divididas en diferentes categorías o grupos y tienen valores discretos. Por ejemplo, en nuestro dataset tenemos el sistema de manejo (drive system) como variable categórica; las categorías son front wheel drive, rear wheel drive y four wheel drive.

Una forma de resumir variables categóricas es usar la función value_counts. Podemos cambiar el nombre de la columna para facilitar la legibilidad. Vemos que hay 118 autos en la categoría front wheel, 75 en rear wheel y 8 en four wheel.


```
drive_wheels_counts=df[ "drive-wheels" ].value_counts().to_frame()  
drive_wheels_counts.rename(columns={ 'drive-wheels' : 'value_counts' }, inplace=True)  
drive_wheels_counts
```

	value_counts
drive-wheels	
fwd	118
rwd	75
4wd	8

IBM Developer

SKILLS NETWORK

Figura. Análisis de variables categóricas.

Los **diagramas de caja** (box plots) son una gran forma de visualizar datos numéricos.

Las principales características que muestra un diagrama de caja son:

- La mediana de los datos.
 - Que representa dónde está el punto medio.
- El cuartil superior.
 - Donde se encuentra el cuartil del 75%.
- El cuartil inferior.
 - Donde se encuentra el cuartil del 25%.
- Los extremos inferior y superior.
 - Se calculan como 1.5 veces el rango intercuartílico.
- Outliers
 - Se representan como puntos individuales que ocurren fuera de los extremos.

Los datos entre los cuartiles superior e inferior representan el rango intercuartílico.

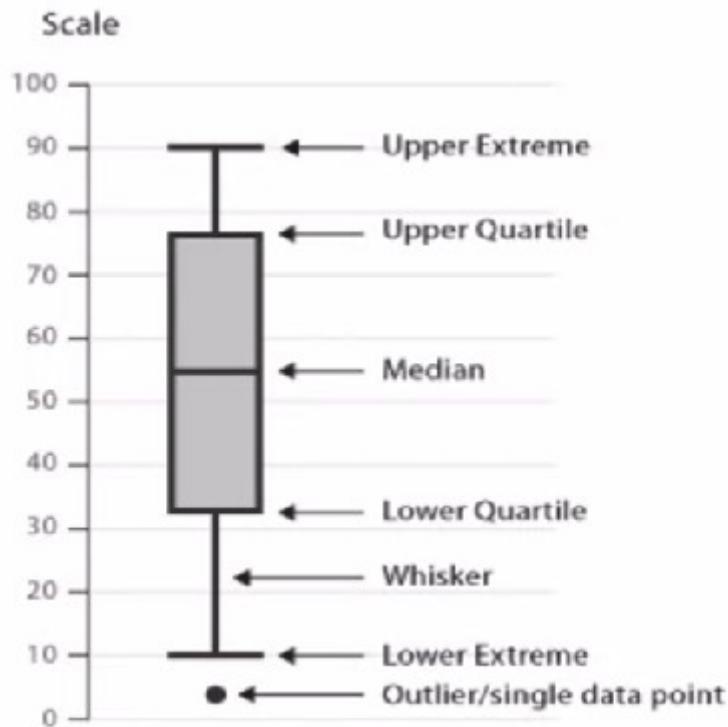


Figura. Diagrama de caja.

Con los diagramas de caja:

- Pueden detectarse fácilmente los outliers.
- Puede verse la distribución y las asimetrías de los datos.
- Se hace fácil comparar entre grupos.

En este ejemplo, usando diagramas de caja vemos la distribución de las diferentes categorías de la característica “drive wheel” sobre “price”.

Vemos que la distribución del precio entre rear wheel drive y otras categorías es distinta, pero entre front wheel drive y four wheel drive es casi indistinguible.


```
sns.boxplot(x= "drive-wheels", y= "price", data=df)
```

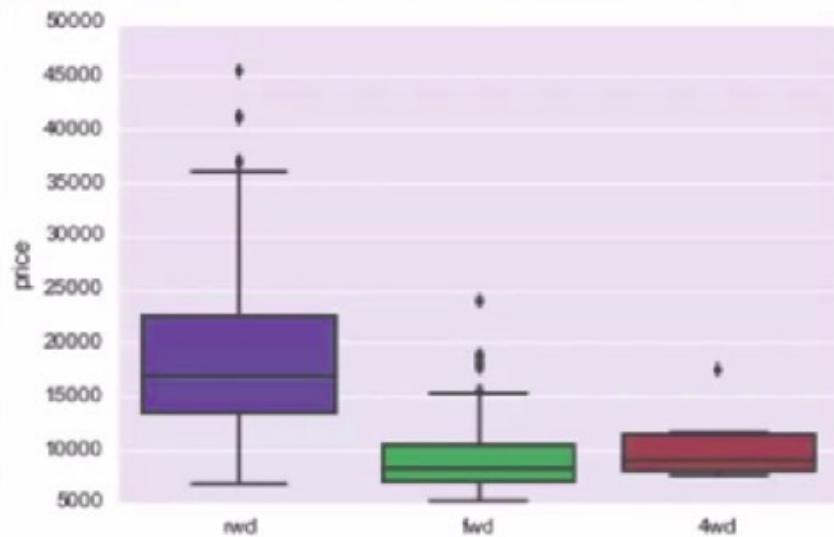


Figura. Diagrama de caja para “drive wheel” sobre “price”.

A menudo tendemos a ver variables continuas en nuestros datos. Estos puntos de datos son números contenidos en un rango.

Por ejemplo, en nuestro dataset, price y “engine size” (tamaño del motor) son variables continuas y queremos entender la relación entre ellas. Podría el tamaño del motor predecir el precio del auto?

Una forma de visualizar esto es utilizando un **gráfico de dispersión** (scatter plot).

Cada observación en un gráfico de dispersión es representado como un punto. Este punto muestra la relación entre 2 variables. La variable **predictora (predictor)** es la que se usa para predecir un resultado, en este caso “engine size”. La variable **objetivo (target)** es la variable que se está intentando predecir, en este caso “price”.

Típicamente establecemos la variable predictora en el eje x y la objetivo en el eje y.

Para implementarlo podemos utilizar funciones de matplotlib.

```

y=df[ "price" ]
x=df[ "engine-size" ]
plt.scatter(x,y)

plt.title("Scatterplot of Engine Size vs Price")
plt.xlabel("Engine Size")
plt.ylabel("Price")

```

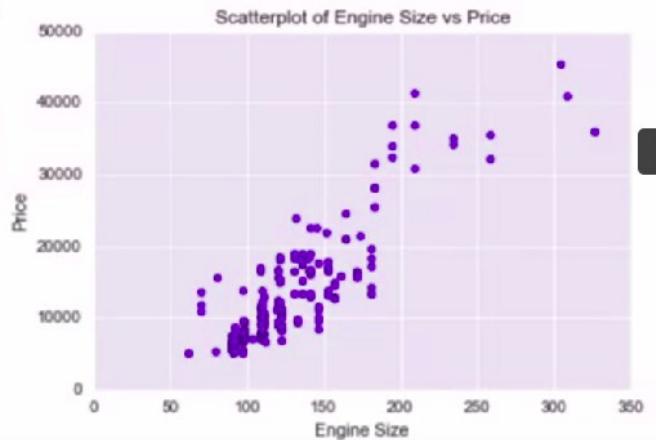


Figura. Gráfico de dispersión.

Ahora, cómo se relacionan “engine size” y “price”?

Del gráfico de dispersión vemos que cuando el “engine size” crece, el precio también lo hace. Esto nos da una indicación inicial de que hay una relación positiva entre estas 2 variables.

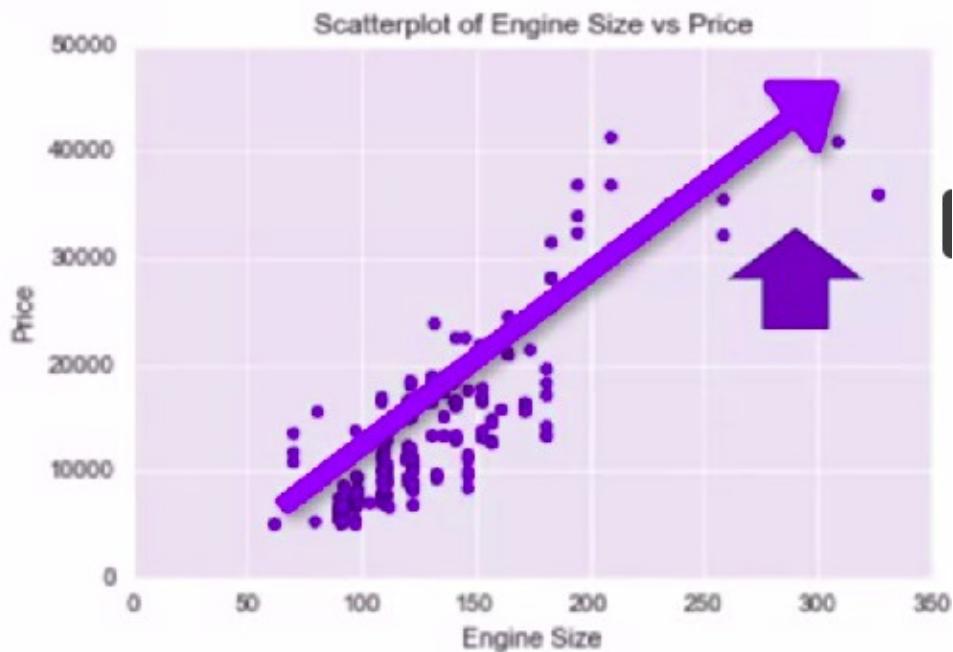


Figura. Relación lineal positiva entre “engine size” y “price”.

3.3 GROUP BY EN PYTHON

Veremos los conceptos básicos de agrupamiento y cómo esto puede ayudarnos a transformar nuestro dataset.

Asuma que quiere saber, si hay alguna, cuál es la relación entre los diferentes sistemas de manejo, forward, rear y four-wheel y el precio de los vehículos, y de ser así, qué tipo agrega más valor al vehículo.

Sería bueno si pudiéramos agrupar los datos por los diferentes tipos de manejo y comparar los resultados uno contra otro.

Podemos lograr esto en Pandas mediante el método “group by”.

El método group by se usa sobre variables categóricas, y agrupa los datos en subconjuntos acorde a las diferentes categorías de la variable.

Puede agrupar mediante una única variable o mediante un grupo.

Como ejemplo digamos que estamos interesados en encontrar el precio promedio de vehículos y observar cómo difieren entre los diferentes estilos de cuerpo (body styles) y ruedas de manejo (drive wheels).

Para hacer esto primero elegimos las 3 columnas de datos en las que estamos interesados.

Luego agrupamos los datos reducidos acorde a “drive wheels” y “body styles”.

Dado que estamos interesados en saber cómo difiere el precio promedio en todos los ámbitos, podemos tomar la media de cada grupo y agregarlo también al final de la línea.

Los datos están ahora agrupados en subcategorías y solamente el precio promedio de cada subcategoría se muestra.

Podemos ver que acorde a nuestros datos, “rear wheel drive convertibles” y “rear wheel drive hard hardtops” tienen los valores más altos, y “four wheel drive hatchbacks” el más bajo.


```

df_test = df[['drive-wheels', 'body-style', 'price']]
df_grp = df_test.groupby(['drive-wheels', 'body-style'], as_index=False).mean()
df_grp

```

	drive-wheels	body-style	price
0	4wd	convertible	20239.229524
1	4wd	sedan	12847.333333
2	4wd	wagon	9095.750000
3	fwd	convertible	11595.000000
4	fwd	hardtop	8249.000000
5	fwd	hatchback	8396.387755
6	fwd	sedan	9811.800000
7	fwd	wagon	9997.333333
8	rwd	convertible	23949.600000
9	rwd	hardtop	24202.714286
10	rwd	hatchback	14337.777778
11	rwd	sedan	21711.833333
12	rwd	wagon	16994.222222

Figura. Categorizando en Pandas.

Una tabla de esta forma no es la más fácil de leer y visualizar. Para hacerla más fácil de entender podemos transformarla en una **tabla pivote** usando el método pivot.

En la tabla previa, drive wheels y body style eran columnas oyentes. Una tabla pivote tiene una variable desplegada a lo largo de las columnas y la otra a lo largo de las filas. Con el método pivot de Pandas podemos pivotear la variable body style para que se despliegue a lo largo de las columnas y que drive wheels lo haga a lo largo de las filas. Los datos de precios se vuelven ahora una grilla rectangular, que es más fácil de visualizar. Esto es similar a lo que se hace habitualmente en Excel.

```
df_pivot = df_grp.pivot(index= 'drive-wheels', columns='body-style')
```

	price					
body-style	convertible	hardtop	hatchback	sedan	wagon	
drive-wheels						
4wd	20239.229524	20239.229524	7603.000000	12647.333333	9095.750000	
fwd	11595.000000	8249.000000	8396.387755	9811.800000	9997.333333	
rwd	23949.600000	24202.714286	14337.777778	21711.833333	16994.222222	

Figura. Convirtiendo a una tabla pivot para facilitar la visualización.

Otra forma de representar una tabla pivot es usar un gráfico de mapa de calor. Un **mapa de calor** toma una grilla rectangular de datos y asigna una intensidad de color basada en el valor de los datos en los puntos de la grilla. Es una gran forma de graficar la variable objetivo sobre múltiples variables y obtener pistas visibles de la relación entre ellas y el objetivo.

En el ejemplo usamos el método `color` de `pyplot` para graficar el mapa de calor y convertir la tabla pivot previa a una forma gráfica. Especificamos el esquema de color rojo-azul. En la salida, cada tipo de body style es numerado a lo largo del eje x y cada tipo de drive wheels en el eje y.

Los precios promedio son graficados con colores que varían de acuerdo a sus valores. Conforme la barra de colores, vemos que la sección superior del mapa de calor parece tener precios más altos que la inferior.

```
plt.pcolor(df_pivot, cmap='RdBu')
plt.colorbar()
plt.show()
```

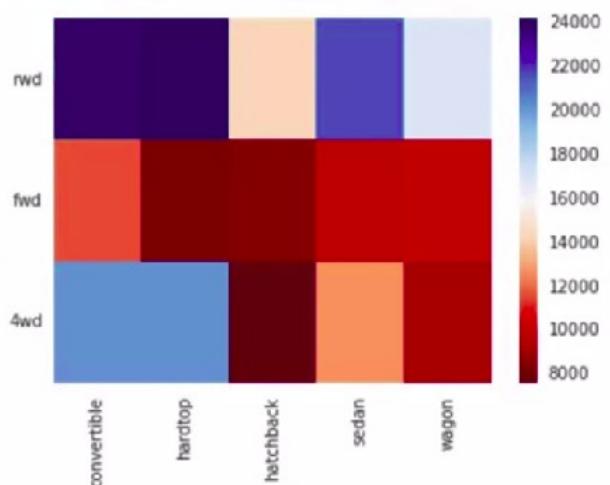


Figura. Mapa de calor.

3.4. CORRELACIÓN

Vamos a hablar de la correlación entre diferentes variables.

La **correlación** es una métrica estadística para que mide hasta qué punto diferentes variables son interdependientes.

En otras palabras, cuando miramos dos variables a lo largo del tiempo, si una variable cambia ¿cómo afecta esto al cambio en la otra variable?

Por ejemplo, se sabe que fumar está correlacionado con el cáncer de pulmón ya que usted tiene una mayor probabilidad de contraer cáncer de pulmón si fuma.

En otro ejemplo, hay una correlación entre paraguas y variables de lluvia donde más precipitación significa que más personas usan paraguas. Además, si no llueve la gente no llevaría paraguas. Por lo tanto, podemos decir que los paraguas y la lluvia son interdependientes y por definición están correlacionados.

Es importante saber que **la correlación no implica causalidad**.

De hecho, podemos decir que el paraguas y la lluvia están correlacionados, pero no tendríamos suficiente información para decir si el paraguas causó la lluvia o la lluvia causó el paraguas.

Veamos la correlación entre el tamaño del motor y el precio. Esta vez vamos a visualizar estas dos variables usando un diagrama de dispersión y una línea lineal añadida llamada línea de regresión, que indica la relación entre las dos. El objetivo principal de esta parcela es ver si el tamaño del motor tiene algún impacto en el precio. En este ejemplo, puede ver que la línea recta a través de los puntos de datos es muy empinada, lo que muestra que hay una relación lineal positiva entre las dos variables. Con el aumento de los valores del tamaño del motor, los valores de precio aumentan también y la pendiente de la línea es positiva. Así que hay una correlación positiva entre el tamaño del motor y el precio. Podemos usar seaborn.regplot para crear el diagrama de dispersión.

```
sns.regplot(x="engine-size", y="price", data=df)  
plt.ylim(0,)
```

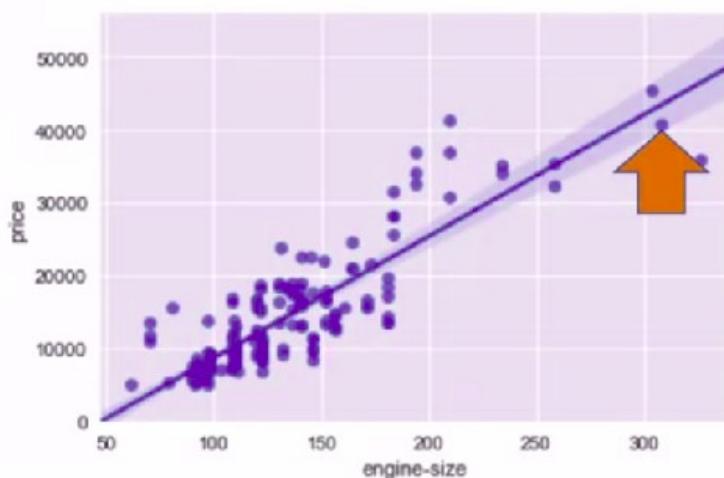


Figura. Correlación positiva entre “engine size” y “price”.

Como otro ejemplo, ahora veamos la relación entre millas por galón para ver su impacto en el precio del automóvil. Como podemos ver en esta parcela, cuando el valor de millas por galón sube el precio del valor baja. Por lo tanto, hay una relación lineal negativa entre millas por galón y precio. Aunque esta relación es negativa, la pendiente de la línea es empinada lo que significa que las millas de carretera por galón siguen siendo un buen predictor de precio. Se dice que estas dos variables tienen una correlación negativa.

```
sns.regplot(x="highway-mpg", y="price", data=df)  
plt.ylim(0,)
```

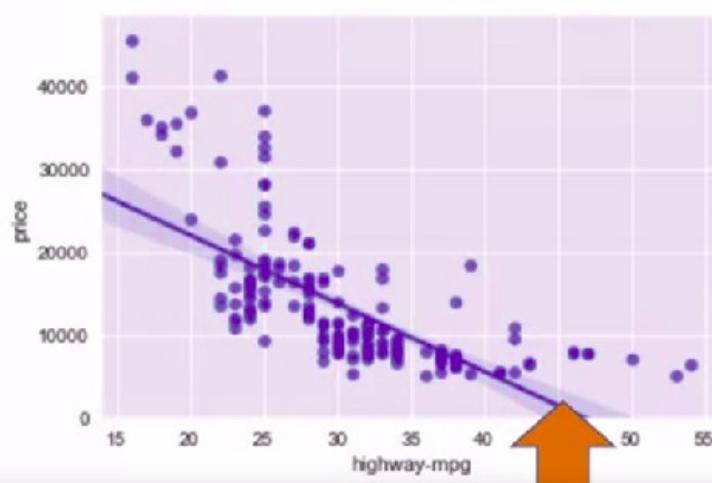


Figura. Correlación negativa entre “highway-mpg” y “price”.

Finalmente, tenemos un ejemplo de una correlación débil. Por ejemplo, tanto las RPM de pico bajo como los valores altos de RPM pico tienen precios bajos y altos. Por lo tanto, no podemos usar RPM para predecir los valores.

```
sns.regplot(x="peak-rpm", y="price", data=df)  
plt.ylim(0,)
```

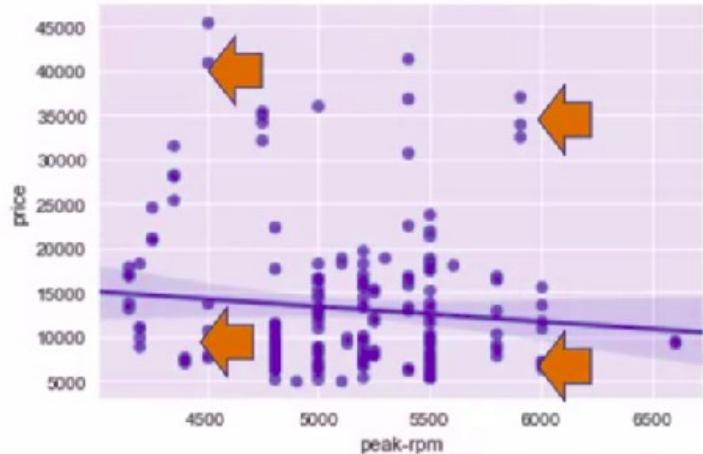


Figura. “peak-rpm” y “price” son variables poco correlacionadas.

3.5. CORRELACIÓN - ESTADÍSTICA

Presentamos varios métodos estadísticos de correlaciones.

Una forma de medir la fuerza de la correlación entre la variable numérica continua es mediante el uso de un método llamado **correlación Pearson**.

El método de correlación Pearson le dará dos valores:

- El coeficiente de correlación
 - Un valor cercano a 1 implica una gran correlación positiva.
 - Un valor cercano a -1 implica una gran correlación negativa.
 - Un valor cercano a cero implica ninguna correlación entre las variables.
- El valor P (p-valor)
 - Nos dice qué tan seguros estamos de la correlación que calculamos.
 - Un valor menor que .001 nos da una fuerte certeza.
 - Un valor entre .001 y .05 nos da una certeza moderada.
 - Un valor entre .05 y .1 nos dará una certeza débil.
 - Y un valor P mayor que .1 no nos dará certeza de correlación en absoluto.

Podemos decir que hay una fuerte correlación cuando el coeficiente de correlación es cercano a 1 o negativo 1, y el valor P es menor que .001.

El siguiente diagrama muestra datos con diferentes valores de correlación.

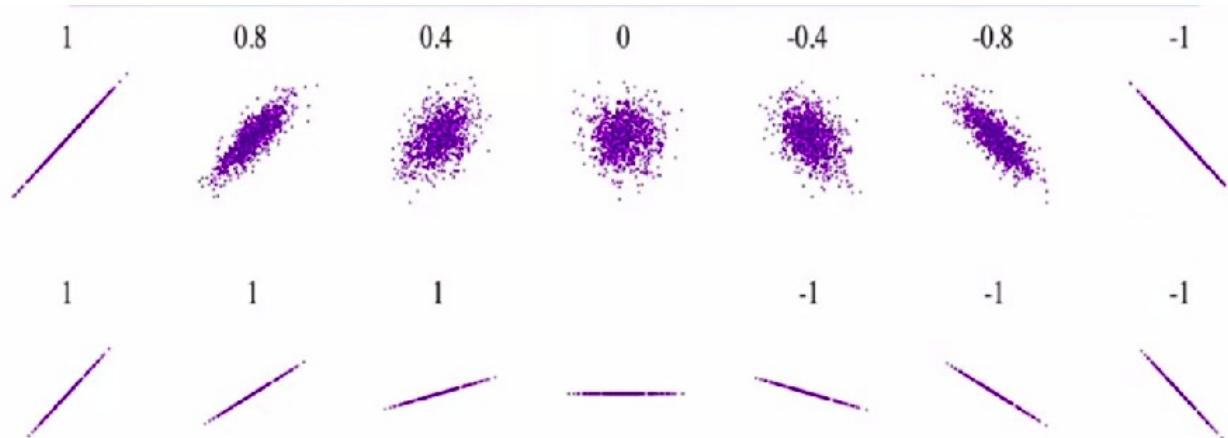


Figura. Datos con diferentes correlaciones.

En este ejemplo, queremos ver la correlación entre la potencia de la variable y el precio del automóvil.

¿Ves lo fácil que puedes calcular la correlación de Pearson usando el paquete de estadísticas sciPy?

```
pearson_coef, p_value = stats.pearsonr(df['horsepower'], df['price'])
```

- Pearson correlation: 0.81
- P-value : 9.35 e-48

Figura. Cálculo de la correlación.

Podemos ver que el coeficiente de correlación es aproximadamente 0.8, y esto es cercano a 1. Así que hay una fuerte correlación positiva. También podemos ver que el valor P es muy pequeño, mucho más pequeño que .001. Y así podemos concluir que estamos seguros de la fuerte correlación positiva.

Teniendo en cuenta todas las variables, ahora podemos crear un mapa de calor que indique la correlación entre cada una de las variables entre sí. La combinación de colores indica el coeficiente de correlación de Pearson, indicando la fuerza de la correlación entre dos variables. Podemos ver una línea diagonal con un color rojo oscuro, indicando que todos los valores en esta diagonal están altamente correlacionados. Esto tiene sentido porque cuando miras más de cerca, los valores en la diagonal son la correlación de todas las variables consigo mismas, que será siempre 1. Este mapa de calor de correlación nos da un buen panorama de cómo las diferentes variables están relacionadas entre sí y, , lo más importante, cómo estas variables están relacionadas con el precio.

Correlation-Heatmap

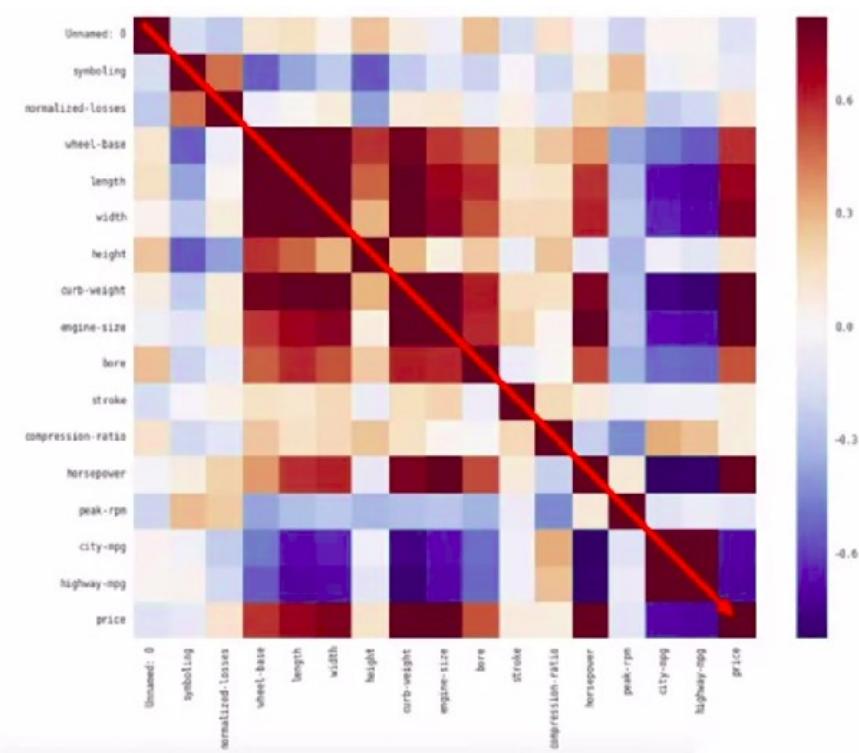


Figura. Mapa de calor de correlaciones.

3.6. ANÀLISIS DE LA VARIANZA – ANOVA

Estudiaremos la varianza.

Asumamos que queremos analizar una variable categórica y ver la correlación entre las diferentes categorías.

Por ejemplo, considere el dataset de autos. La pregunta sería:

Cómo las diferentes categorías que hacen que la marca sea una variable categórica tienen impacto en el precio?

El diagrama muestra el precio promedio de diferentes marcas de vehículos.

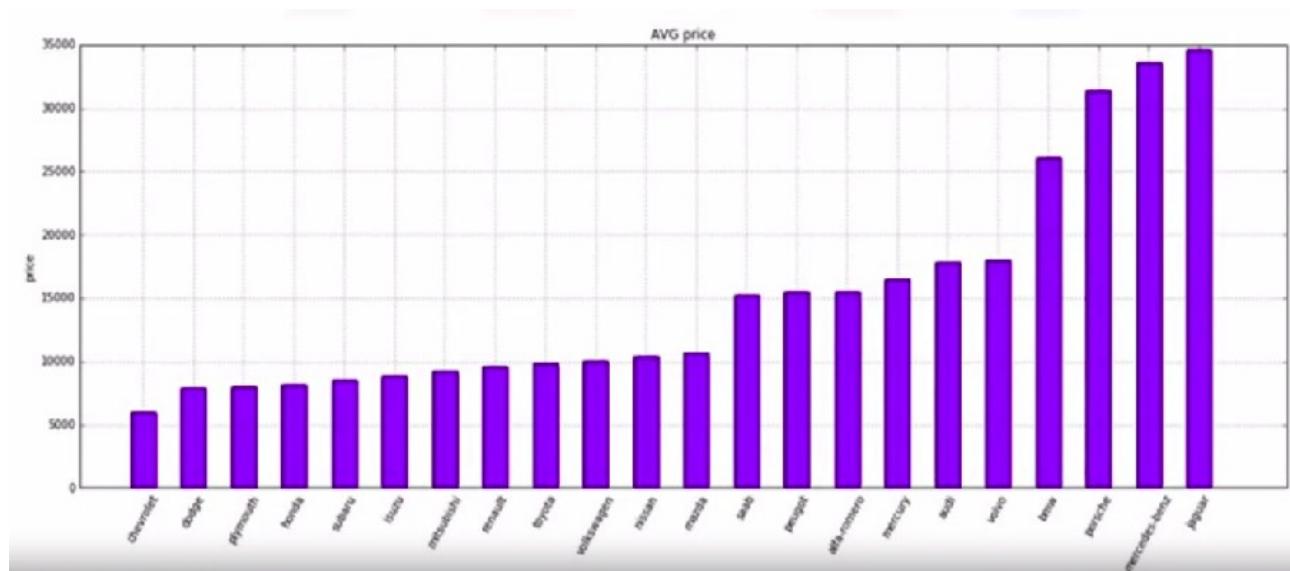


Figura. Precio promedio de diferentes marcas de vehículos.

Vemos una tendencia en el incremento del precio conforme nos movemos hacia la derecha.

Pero qué categoría en la característica marca tiene el mayor y cuál tiene el menor impacto en la predicción del precio?

Para analizar variables categóricas como la marca, podemos usar el método **ANOVA**.

ANOVA es un test estadístico que significa Análisis de la Varianza (Analysis of Variance).

ANOVA puede ser utilizado para encontrar la correlación entre los diferentes grupos de una variable categórica.

De acuerdo al dataset de autos, podemos usar ANOVA para ver si hay una diferencia significativa en el precio medio para diferentes marcas tales como Subaru y Honda.

El test ANOVA retorna 2 valores:

- F-test score

- Calcula el cociente de variación entre la media de los grupos, sobre la variación dentro de cada uno de los grupos de muestra.
- P-value
 - Muestra si el resultado obtenido es estadísticamente significativo.

Sin entrar en mayores detalles el F-test calcula el cociente de variación entre las medias de los grupos sobre la variación dentro de cada una de las medias de los grupos de muestra.

El diagrama ilustra un caso donde el F-test score es pequeño, porque, como podemos ver, la variación de los precios en cada grupo de datos es mucho mayor que las diferencias entre los valores promedio de cada grupo.

Mirando el diagrama, asumimos que el grupo 1 es Honda y el 2 es Subaru. Ambos son las categorías de la característica “marca”. Ya que el F-score es pequeño, la correlación entre precio como variable objetivo y los agrupamientos es pequeña,

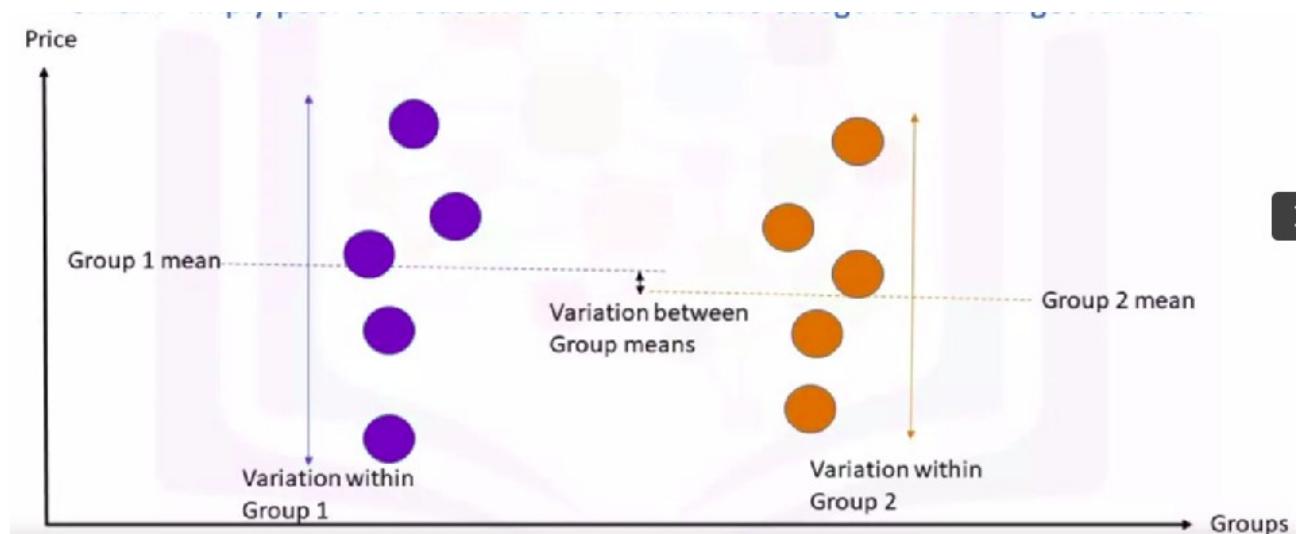


Figura. F-test score pequeño.

En el segundo diagrama vemos un caso donde el F-test score es grande. La variación entre los promedios de los 2 grupos es comparable a las variaciones dentro de los 2 grupos.

Asumimos que el grupo1 es Jaguar y el 2 es Honda. Ambos son las categorías de la característica “marca”. Como el F-score es grande, la correlación es fuerte.

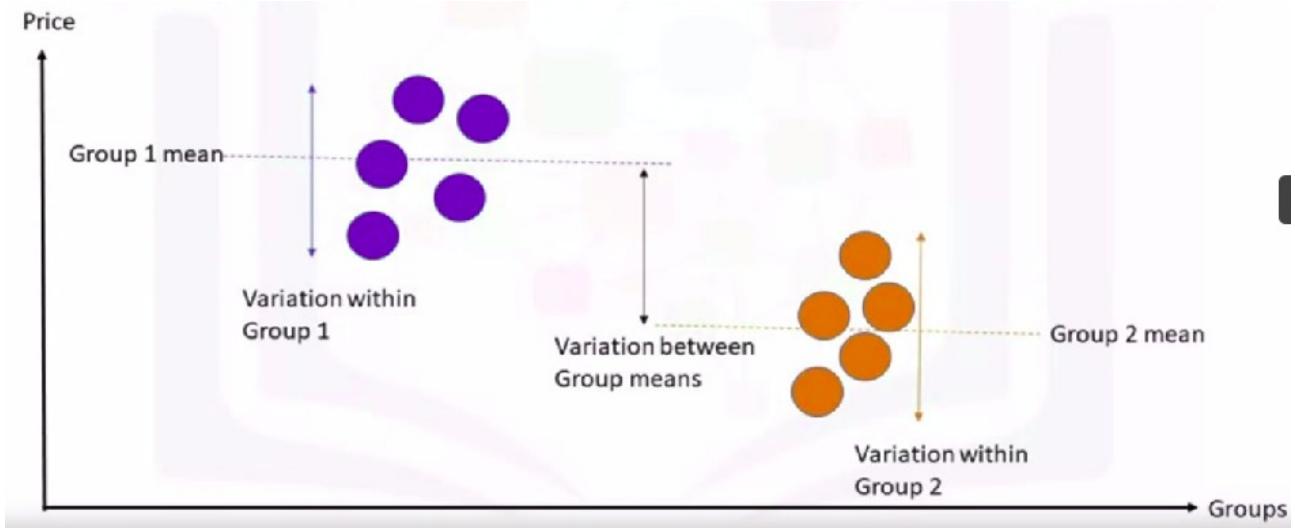


Figura. F-test score grande.

Volviendo al ejemplo, la gráfica de barras muestra el precio promedio para diferentes categorías de la característica precio.

Como se ve del gráfico de barras, esperamos un F-score pequeño entre Honda y Subaru porque hay una pequeña diferencia entre los precios promedio.

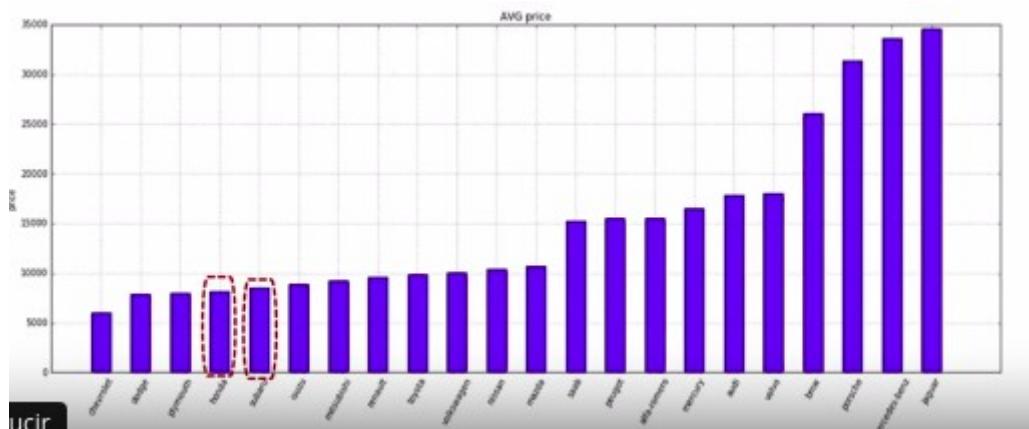


Figura. Esperamos un F-test score pequeño entre Honda y Subaru.

Por otro lado, esperamos un F-value grande entre Hondas y Jaguars, porque las diferencias entre los precios son muy significativas.

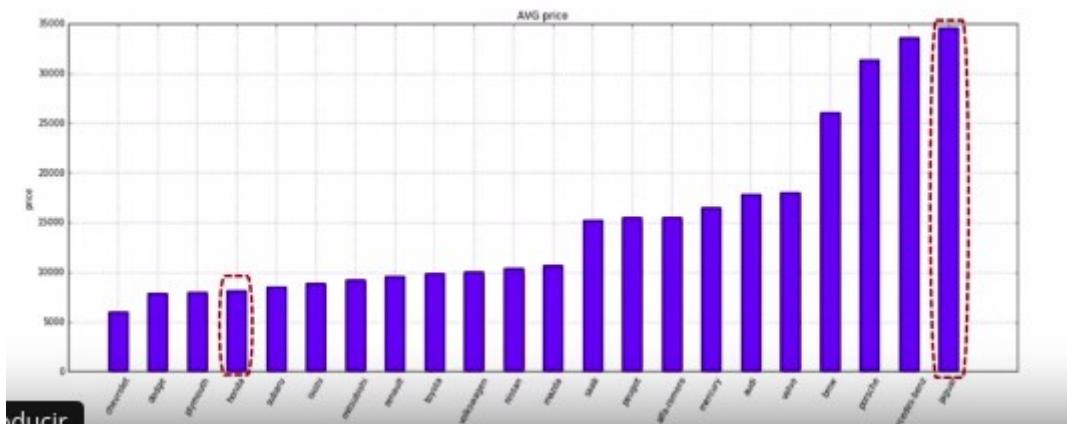


Figura. Esperamos un F-test score grande entre Hondas y Jaguars.

Sin embargo, del gráfico, no conocemos las varianzas exactas.

Hagamos un test ANOVA para ver si nuestra intuición es correcta.

En la primera línea extraemos los datos de marca y precio y luego los agrupamos por diferentes marcas. En Python, el test ANOVA se implementa usando el método `f_oneway` de SciPy. Le pasamos los datos de precio de los 2 grupos de marcas que queremos comparar y calculamos los resultados.

El resultado confirma nuestra suposición. Los precios entre Hondas y Subarus son no significativamente diferentes ya que el F-test score es menor que 1 y *el p-value es mayor que 0.05*.

Podemos hacer lo mismo para Honda u Jaguar. Los precios entre ellos significativamente diferentes ya que el F-score es muy grande.

- ANOVA between “Honda” and “Subaru”

```
df_anova=df[["make", "price"]]  
grouped_anova=df_anova.groupby(["make"])
```

```
eva_results_1=stats.f_oneway(grouped_anova.get_group("honda")["price"], grouped_anova.get_group("toyota")["price"])
VA results: F=0.19744031275, p-Value onewayResult(statistic=0.19744031275), pvalue=0.6609478241
```

- ANOVA between “Honda” and “Jaguar”

```
anova_results_1=stats.f_oneway(grouped_anova.get_group("honda")["price"], grouped_anova.get_group("jaguar")["price"])

ANOVA results: F=400.92587, p=F onewayResult(statistic=400.92587), pvalue=1.05861e-11
```

Figura. Implementación del test ANOVA.

Resumiendo, podemos decir que hay una correlación fuerte entre una variable categórica y otras variables si el test ANOVA da:

- Un F-test value grande.
- Un p-value pequeño.

3.7. LABORATORIO.

Notebook: 3. exploratory-data-analysis

3.8. ANEXO.

3.8.1. Glosario.

1. Gráfico de dispersión

Cada observación en un gráfico de dispersión es representado como un punto. Este punto muestra la relación entre 2 variables.

La variable **predictora (predictor)** es la que se usa para predecir un resultado.

La variable **objetivo (target)** es la variable que se está intentando predecir.

Típicamente establecemos la variable predictora en el eje x y la objetivo en el eje y.

2. Mapa de calor

Un **mapa de calor** toma una grilla rectangular de datos y asigna una intensidad de color basada en el valor de los datos en los puntos de la grilla. Es una gran forma de graficar la variable objetivo sobre múltiples variables y obtener pistas visibles de la relación entre ellas y el objetivo.

3.8.2. Conceptos.

1. Análisis Exploratorio de Datos (Exploratory Data Analysis, EDA)

EDA es un enfoque para:

- Analizar datos en con el fin de resumir las principales características de los mismos.
- Obtener una mejor comprensión del conjunto de datos.
- Descubrir relaciones entre diferentes variables.
- Extraer variables importantes para el problema que estamos tratando de resolver.

2. Estadística Descriptiva

Una estadística descriptiva básica suele incluir:

- La media
- El número total de puntos de datos.
- La desviación estándar.
- Los cuartiles.
- Los valores extremos.

3. Utilización de diagramas de caja

Con los diagramas de caja:

- Pueden detectarse fácilmente los outliers.
- Puede verse la distribución y las asimetrías de los datos.

- Es fácil comparar entre grupos.

4. Correlación

La **correlación** es una métrica estadística para que mide hasta qué punto diferentes variables son interdependientes.

En otras palabras, cuando miramos dos variables a lo largo del tiempo, si una variable cambia ¿cómo afecta esto al cambio en la otra variable?

Es importante saber que la correlación no implica causalidad.

Una forma de medir la fuerza de la correlación entre la variable numérica continua es mediante el uso de un método llamado correlación Pearson.

El método de correlación Pearson le dará dos valores:

- El coeficiente de correlación
 - Un valor cercano a 1 implica una gran correlación positiva.
 - Un valor cercano a -1 implica una gran correlación negativa.
 - Un valor cercano a cero implica ninguna correlación entre las variables.
- El valor P (p-valor)
 - Nos dice qué tan seguros estamos de la correlación que calculamos.
 - Un valor menor que .001 nos da una fuerte certeza.
 - Un valor entre .001 y .05 nos da una certeza moderada.
 - Un valor entre .05 y .1 nos dará una certeza débil.
 - Y un valor P mayor que .1 no nos dará certeza de correlación en absoluto.

5. Análisis de la varianza (ANOVA)

El test ANOVA retorna 2 valores:

- F-test score
 - Calcula el cociente de variación entre la media de los grupos, sobre la variación dentro de cada uno de los grupos de muestra.
- P-value
 - Muestra si el resultado obtenido es estadísticamente significativo.

Resumiendo, podemos decir que hay una correlación fuerte entre una variable categórica y otras variables si el test ANOVA da:

- Un F-test value grande.
- Un p-value pequeño.

PARTE IV: MODELO

4.1. DESARROLLO DEL MODELO

Vamos a examinar el desarrollo del modelo tratando de predecir el precio de un automóvil usando nuestro conjunto de datos.

En este módulo, aprenderá acerca de:

- Regresión lineal simple y múltiple.
- La evaluación del modelo usando la visualización.
- Regresión polinómica y tuberías (pipelines).
- R cuadrado y MSE para la evaluación in-sample.
- Predicción y toma de decisiones.
- Cómo se puede determinar un valor razonable para un usado coche.

Un modelo o estimador puede ser considerado como una ecuación matemática utilizada para predecir el valor a partir de uno o más valores.

Se relacionan una o más variables o entidades independientes con variables dependientes. Por ejemplo, “millas de carretera por galón” puede ser la variable o entidad independiente y la salida del modelo o variable dependiente el precio.

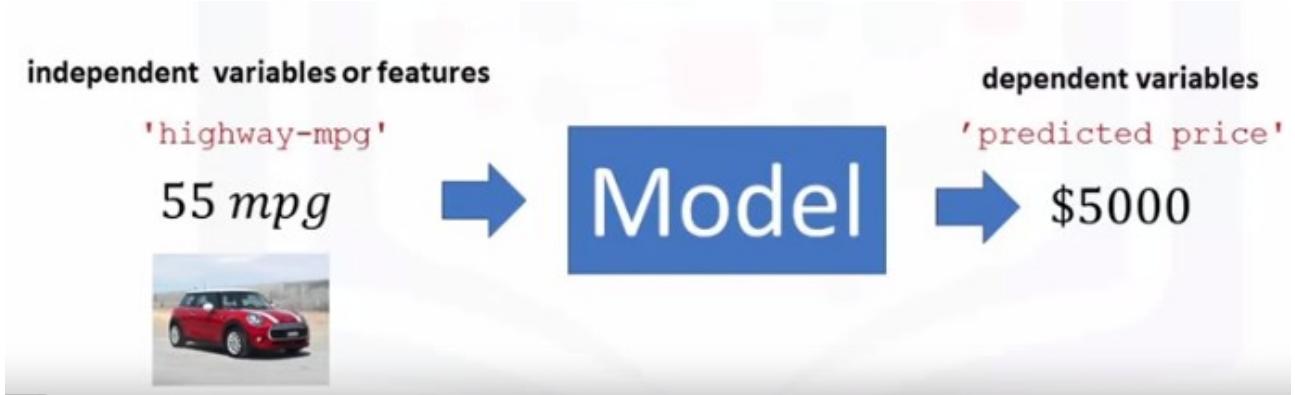


Figura. Modelo.

Por lo general, mientras más datos relevantes tenga, más preciso será su modelo.

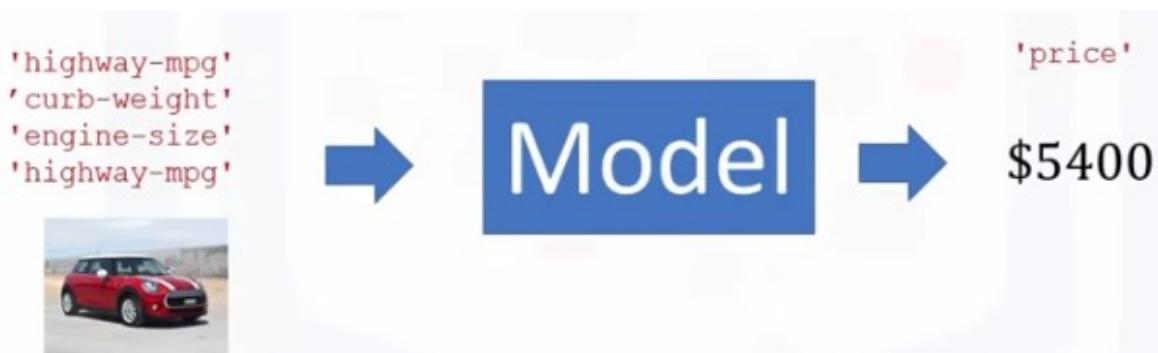


Figura. En general, mientras más datos se tenga, más preciso es el modelo.

Para entender por qué más datos son importantes, considere la siguiente situación. Usted tiene dos coches casi idénticos. Los coches de color rosa se venden por significativamente menos. Usted desea utilizar su modelo para determinar el precio de dos coches, uno rosa, uno rojo. Si sus variables independientes o características no incluyen color, su modelo predice el mismo precio para ambos.

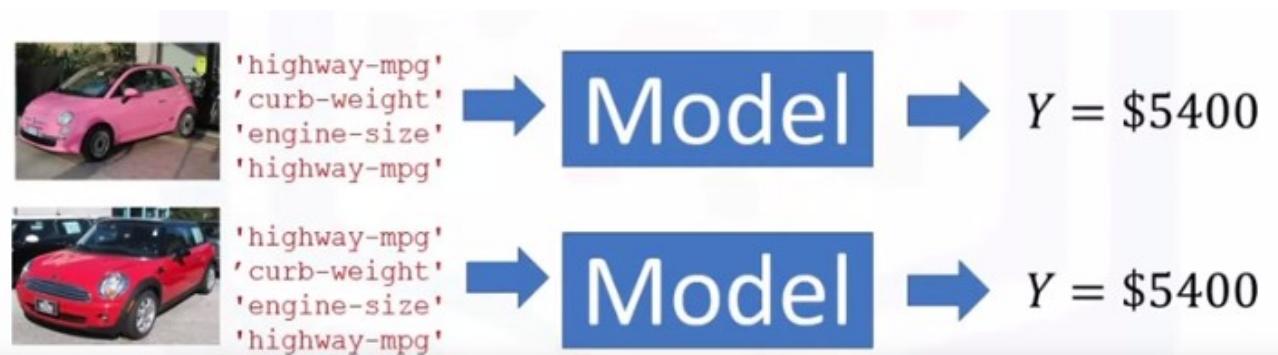


Figura. Si no incluimos la característica color predeciríamos el mismo precio para ambos autos.

4.2. REGRESIÓN LINEAL Y REGRESIÓN LINEAL MÚLTIPLE

Estudiaremos la regresión lineal simple y la regresión lineal múltiple.

La **regresión lineal simple SLR** hace referencia a que una única variable independiente realiza la predicción.

En la **regresión lineal múltiple** varias variables realizan la predicción.

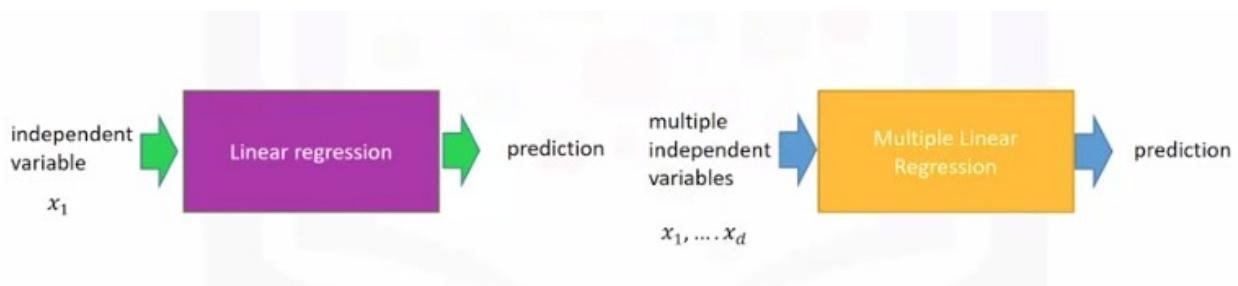


Figura. Regresión Lineal.

SLR es un método que nos ayuda a entender la relación entre 2 variables:

- Una variable independiente x (predictora)
- Una variable dependiente y (objetivo)

Nos gustaría llegar a una relación lineal entre las variables de la forma:

$$y = b_0 + b_1 x$$

donde b_0 es la intercepción y b_1 la pendiente.

Cuando entrenemos o ajustemos el modelo obtendremos estos parámetros.

Aclaremos el paso de predicción.

Es difícil imaginar cuánto cuesta un auto, pero las “millas de carretera por galón” están en el manual del fabricante. Si asumimos hay una relación lineal entre ellas, podemos formular un modelo para determinar el precio del auto.

Si las “millas de carretera por galón” son 20, podemos usar este valor como entrada en el modelo para obtener una predicción de \$22000 en el modelo del ejemplo.

$$\begin{aligned}
 y &= 38423 - 821x \\
 &= 38423 - 821(20) \\
 &= 22\,003
 \end{aligned}$$

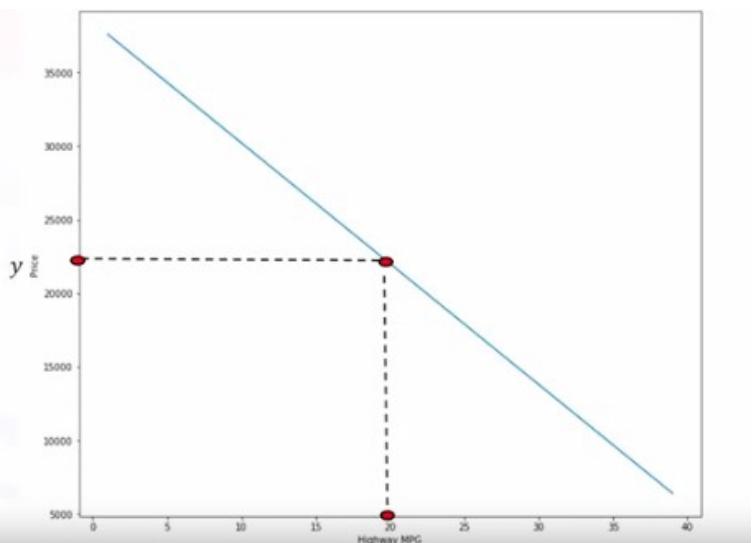


Figura. Obteniendo el valor de una predicción.

Para determinar la recta, tomamos puntos de datos de nuestro dataset y marcarlos. Luego usamos estos puntos para entrenar nuestro modelo. Como resultado obtendremos los parámetros.

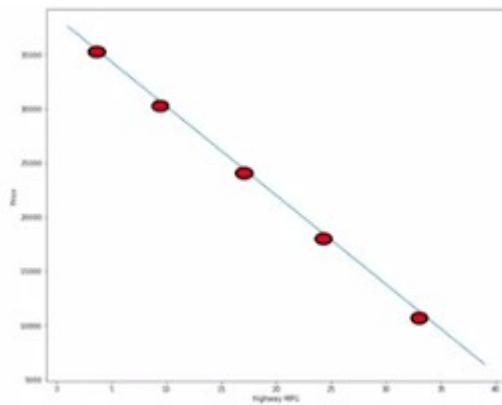
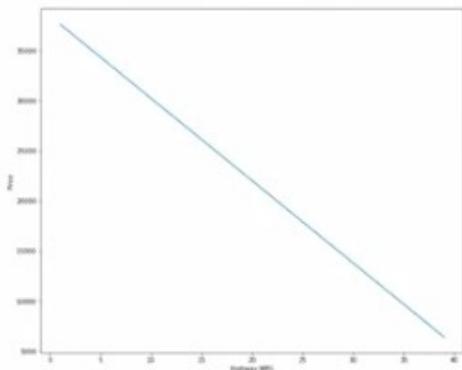


Figura. Datos para entrenar el modelo.



Fit

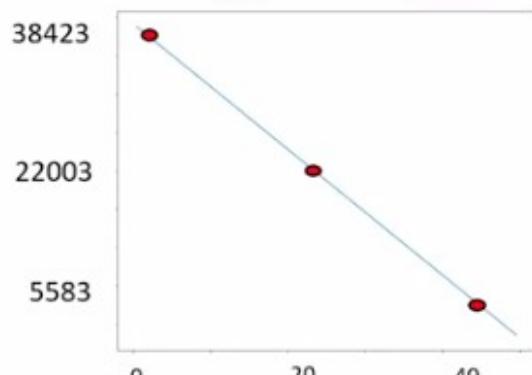
(b_0, b_1)

Figura. Luego de entrenar el modelo tendremos como resultado los parámetros.

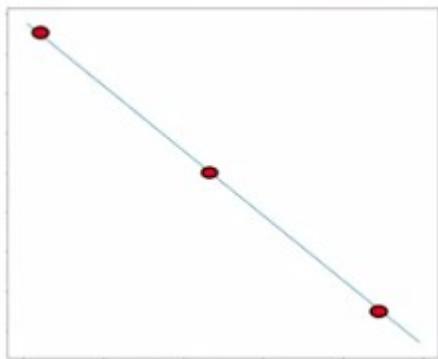
Usualmente almacenamos los puntos de datos en un dataframe o en arrays de numpy.

- El valor que deseamos predecir se dice es el objetivo y lo almacenamos en un array y.
- La variable independiente la almacenamos en un dataframe o array x.

Cada muestra corresponde a una fila diferente en cada dataframe o array.



$$X = \begin{bmatrix} & \\ & \\ & \end{bmatrix} \quad Y = \begin{bmatrix} \\ \\ \end{bmatrix}$$



$$X = \begin{bmatrix} 0 \\ 20 \\ 40 \end{bmatrix} \quad Y = \begin{bmatrix} 38423 \\ 22003 \\ 5583 \end{bmatrix}$$

Figura. Almacenando los puntos de datos en dataframes o arrays.

Cada muestra corresponde a una fila diferente en cada dataframe o array.

En muchos casos, varios factores influencia qué tanto la gente paga por un auto. Por ejemplo, qué tan viejo es. En este modelo, la incertidumbre es tomada en cuenta asumiendo que un valor aleatorio pequeño es agregado a cada punto en la recta. Esto se llama ruido.

En la siguiente figura, a la izquierda, se muestra la distribución del ruido. El eje vertical muestra el valor agregado y el horizontal la probabilidad de que ese valor haya sido agregado. En general es pequeño, pero a veces se agregan valores grandes.

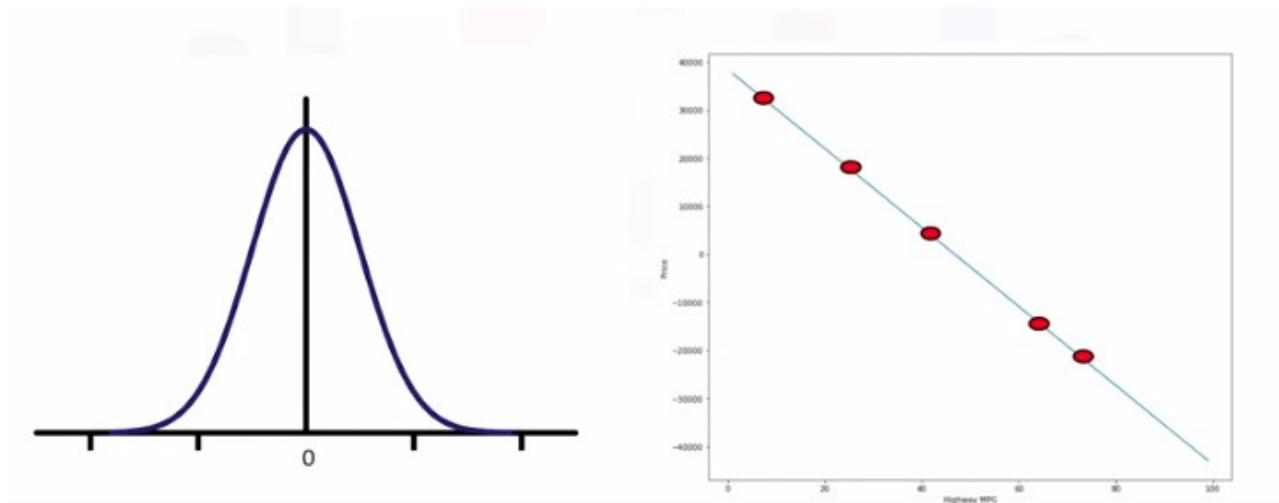


Figura. Distribución del ruido.

Podemos resumir el proceso así:

- Tenemos un conjunto de puntos de entrenamiento.
- Usamos estos puntos para entrenar o ajustar el modelo y obtener los parámetros.
- Usamos estos parámetros en el modelo.
- Tenemos ahora un modelo.

Usamos \hat{y} para denotar que es un estimado.

Podemos usar este modelo para predecir valores que no hemos visto. Por ejemplo, no tenemos un auto con 20 millas de carretera por galón. Para un auto de este tipo podemos usar el modelo para predecir su precio.

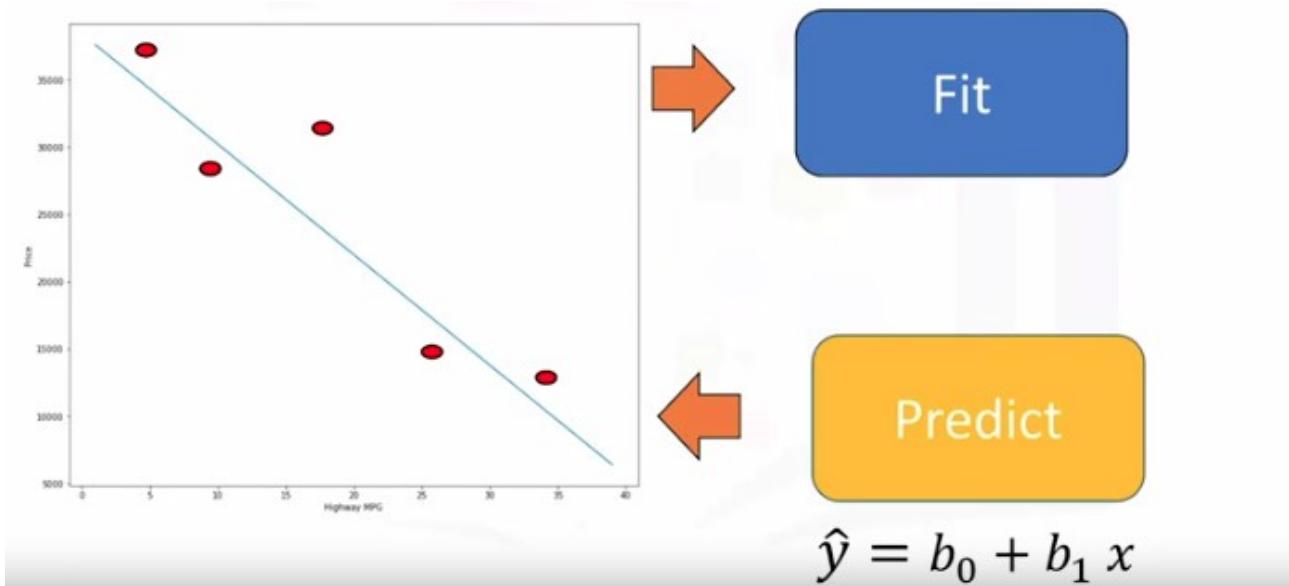


Figura. Resumen del proceso y uso posterior para predicción.

No olvide que el modelo no siempre es correcto. Podemos ver esto comparando el valor predicho con el valor verdadero, por ejemplo para el auto de 10 millas de carretera por galón.

Si la asunción de que el modelo es lineal es correcta, este error se debe posiblemente al ruido, aunque puede haber otras razones.

Para ajustar un modelo en Python, podemos usar scikitLearn. Primero importamos el modelo y luego creamos el objeto usando el constructor.

```
import linear_model from scikit-learn
# creo un objeto regresión lineal usando el constructor
lm = LinearRegression()
```

Definimos las variables predictora y objetivo y luego usamos el método fit para ajustar el modelo y sus parámetros b_0 y b_1 . La entrada son las características y los objetivos. Podemos obtener una predicción usando el método predict. La salida es un array, que tiene el mismo número de muestras que x . b_0 es un atributo del objeto lm al igual que b_1 .

- We define the predictor variable and target variable

```
X = df[['highway-mpg']]
Y = df['price']
```

- Then use lm.fit (X, Y) to fit the model , i.e fine the parameters b_0 and b_1

```
lm.fit(X, Y)
```

- We can obtain a prediction

```
Yhat=lm.predict(X)
```

Yhat	X
2	5
:	
3	4

Figura. Ajustando el modelo y prediciendo.

La relación entre precio y millas de carretera por galón es:

$$\text{Price} = 38423.31 - 821.73 * \text{highway-mpg}$$

- We can view the intercept (b_0): lm.intercept_

38423.305858
- We can also view the slope (b_1): lm.coef_

-821.73337832
- The Relationship between Price and Highway MPG is given by:
- Price = 38423.31 - 821.73 * highway-mpg

$$\hat{Y} = b_0 + b_1 x$$

Figura. Datos del modelo.

La regresión lineal múltiple es usada para explicar la relación entre un objetivo y 2 o más variables x predictoras.

Si tenemos por ejemplo 4 variables predictoras:

- b_0 es la intersección con el eje x
- b_1 es el coeficiente en x
- b_2 es el coeficiente en x^2
- b_3 es el coeficiente en x^3

$$\hat{Y} = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4$$

Si hay solamente 2 variables podemos visualizar los valores.

Consideré la función:

$$\hat{Y} = 1 + 2x_1 + 3x_2$$

x_1 y x_2 pueden ser visualizadas en un plano 2D.

Veamos un ejemplo. La tabla de la figura siguiente contiene diferentes valores de las variables predictoras x_1 y x_2 . La posición de cada punto se ubica en el plano 2D conforme debe.

n	x_1	x_2
1	0	0
2	0	2
3	1	0
4	3	2

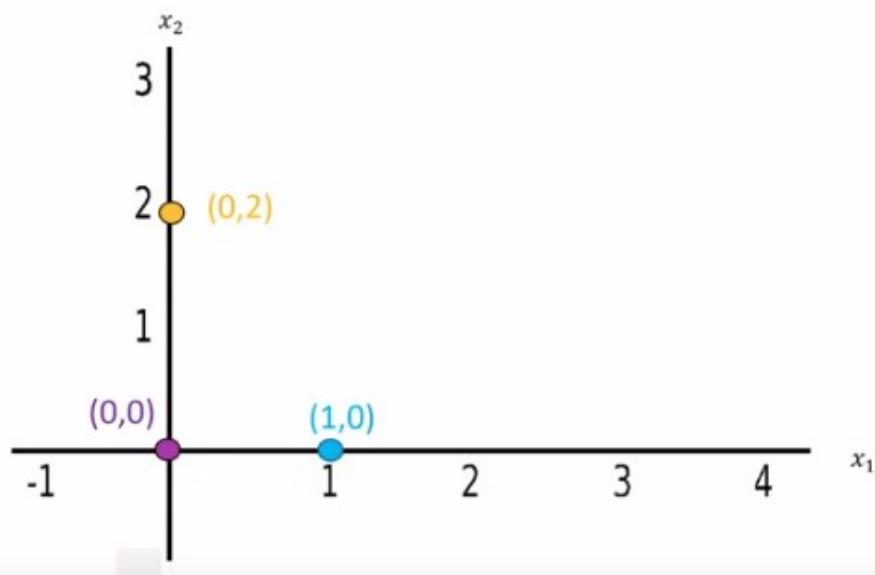


Figura. Plano x_1, x_2 .

Cada valor de los predictores x_1 y x_2 será mapeado a un valor de \hat{y} .

- THIS IS SHOWN BELOW WITH

$$\hat{Y} = 1 + 2x_1 + 3x_2$$

n	x_1	x_2
1	0	0
2	0	2
3	1	0
4	3	2

	\hat{Y}
1	1
2	6
3	2
4	13

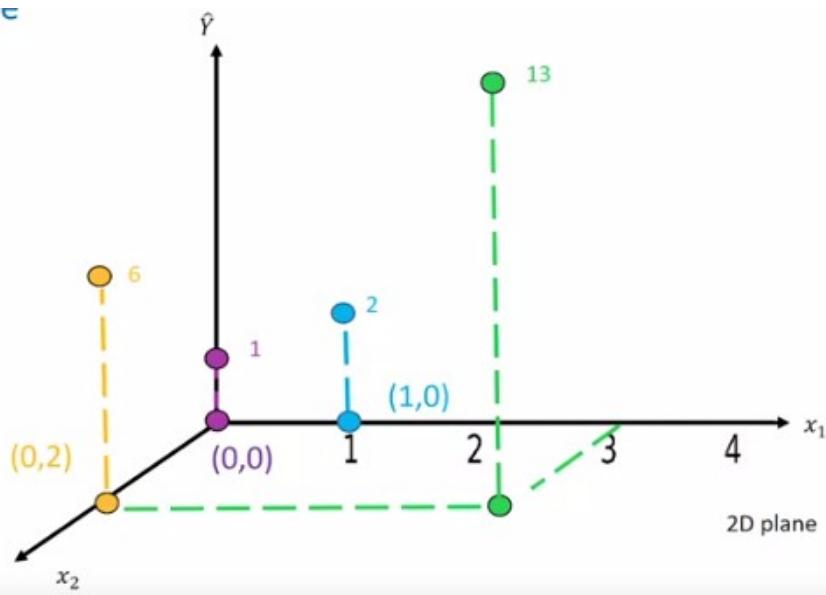


Figura. Graficando la regresión lineal múltiple (2 variables).

Podemos ajustar el modelo como sigue:

- Extraemos las variables predictoras y las almacenamos en la variable z.
- Entrenamos el modelo con el método train.

Podemos obtener una predicción mediante el método predict. En este caso la entrada es un array o dataframe de 4 columnas, en el que el número de filas es el número de muestras. La salida es un array con el mismo número de elementos que de muestras.

1. We can extract the for 4 predictor variables and store them in the variable Z

```
Z = df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']]
```

2. Then train the model as before:

```
lm.fit(Z, df['price'])
```

3. We can also obtain a prediction

```
Yhat=lm.predict(X)
```

x_1	x_2	x_3	x_4
3	5	-4	3
:	:	:	:
2	4	2	-4

Yhat
2
:
3

Figura. Entrenamiento y predicción.

La intercepción y los coeficientes son atributos del objeto.

1. Find the intercept (b_0)

```
lm.intercept_
-15678.742628061467
```

2. Find the coefficients (b_1, b_2, b_3, b_4)

```
lm.coef_
array([52.65851272 ,4.69878948,81.95906216 , 33.58258185])
```

The Estimated Linear Model:

- Price = -15678.74 + (52.66) * horsepower + (4.70) * curb-weight + (81.96) * engine-size + (33.58) * highway-mpg

$$\hat{Y} = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4$$

Figura. Modelo estimado.

4.3. EVALUACIÓN DEL MODELO UTILIZANDO VISUALIZACIÓN

Los gráficos de regresión son un buen estimado de la relación entre 2 variables, la fuerza de la correlación y la dirección de la relación (positiva o negativa). El eje horizontal es la variable independiente y el vertical la dependiente. Cada punto representa un punto objetivo diferente. La recta ajustada representa los valores predecidos.

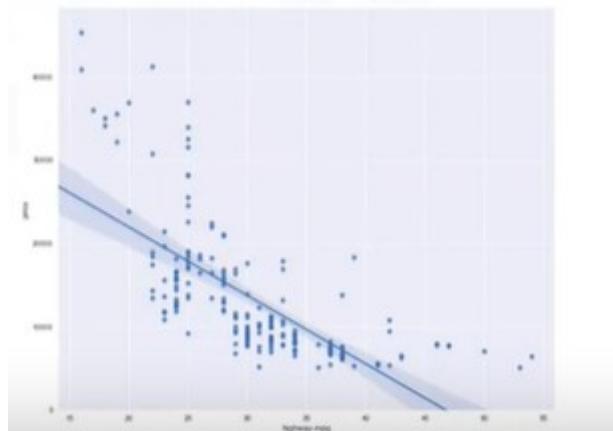


Figura. Gráfico de regresión.

Hay varias formas de implementar el gráfico de regresión. Una forma es utilizar regplot de la librería seaborn.

```
import seaborn as sns  
sns.regplot(x="highway-mpg", y="price", data=df)  
plt.ylim(0,)
```



Figura. Gráfico de regresión implementado con regplot.

En el ejemplo, x es el nombre de la columna que contiene la variable dependiente, y la independiente y data es el nombre del dataframe.

El gráfico residual representa el error respecto al valor verdadero. Al examinar el valor predicho y el verdadero vemos una diferencia, que obtenemos restando los valores. Luego graficamos esos valores en el eje vertical con la variable independiente en el eje horizontal. Repetimos el proceso para el resto de las muestras.

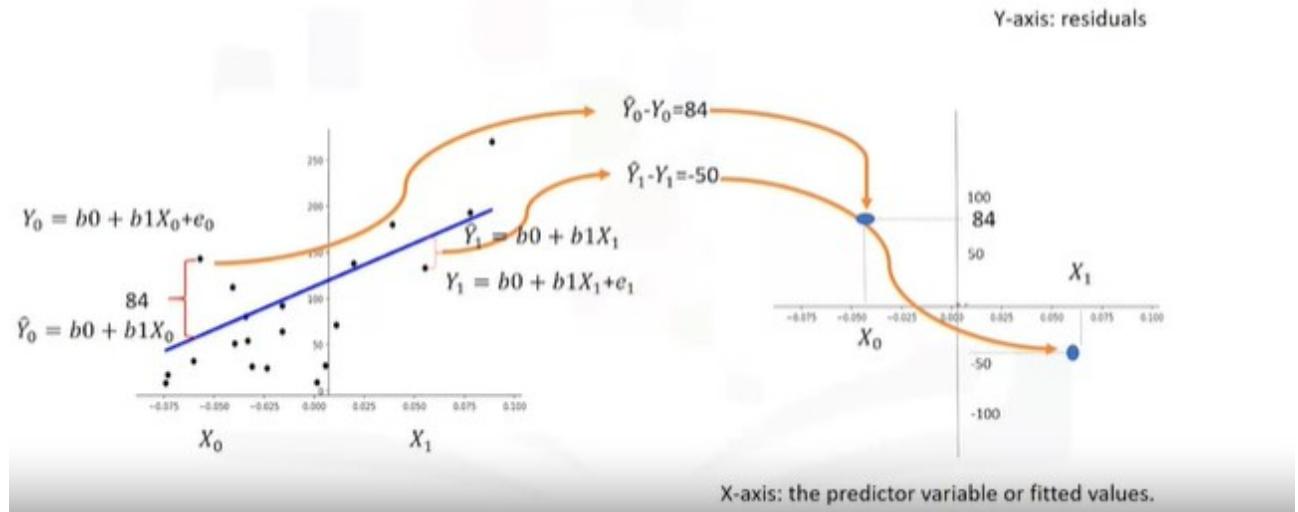


Figura. Construcción de un gráfico residual.

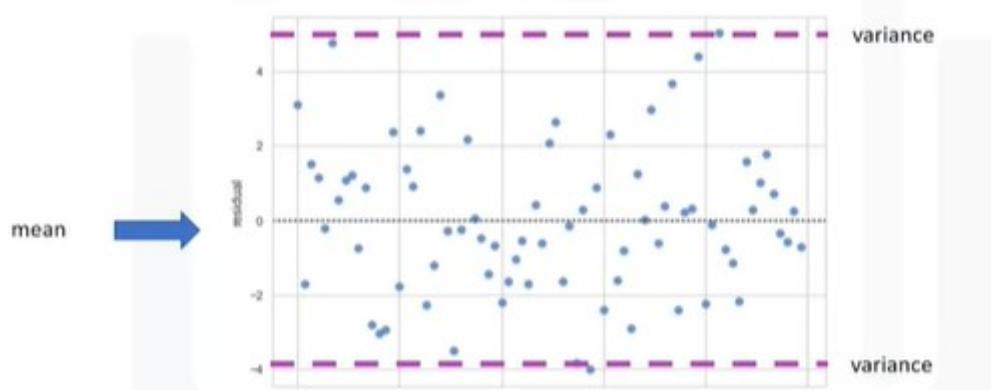


Figura. Gráfico residual.

Mirar el gráfico nos da cierto conocimiento de nuestros datos.

Esperamos que los resultados tengan media 0 y estén distribuidos de forma pareja alrededor del eje x con similar varianza. No hay curvatura. Este tipo de gráfico residual sugiere que el gráfico lineal es apropiado.

En el gráfico residual siguiente se tiene una curvatura. Aquí asumir linealidad sería un error; el gráfico sugiere una función no lineal.

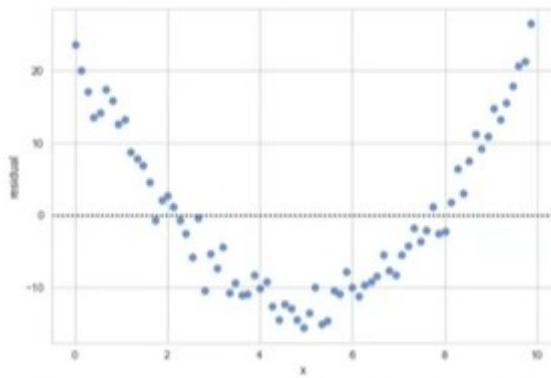


Figura. Gráfico residual con curvatura.

En el gráfico siguiente, la varianza de los residuos crece con x y por tanto el modelo lineal es incorrecto.

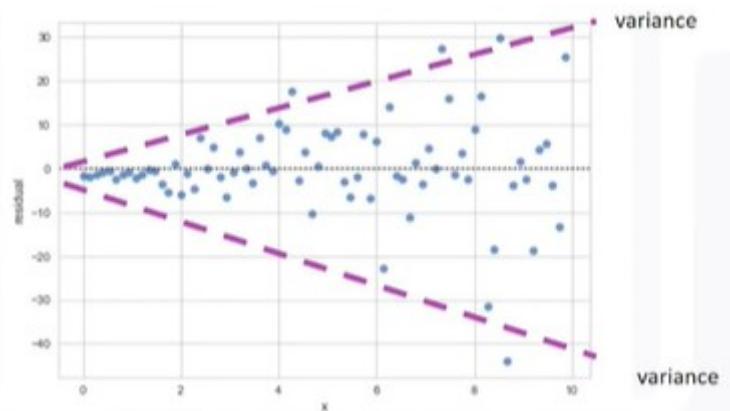


Figura. Varianza que crece con x.

Podemos usar seaborn para crear un gráfico residual.

```
import seaborn as sns  
sns.residplot(df['highway-mpg'], df['price'])
```

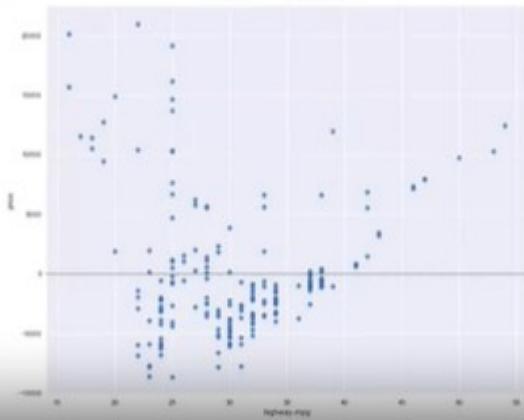


Figura. Gráfico residual con seaborn.

El primer parámetro es la variable dependiente y el segundo la dependiente u objetivo. Vemos que en este caso los residuos tienen una curvatura.

Un **gráfico de distribución** cuenta los valores predichos versus el valor verdadero. Estos gráficos son extremadamente útiles para visualizar modelos con más de una variable independiente.

Veamos un ejemplo simple. Examinamos el eje vertical y luego contamos y graficamos el número de puntos predichos que son aproximadamente iguales a 1. Luego hacemos lo mismo con los que son aproximadamente igual a 2. Para los puntos predichos, son aproximadamente igual a 3. Luego repetimos el proceso para los valores objetivo (target), que en este caso son aproximadamente igual a 2.

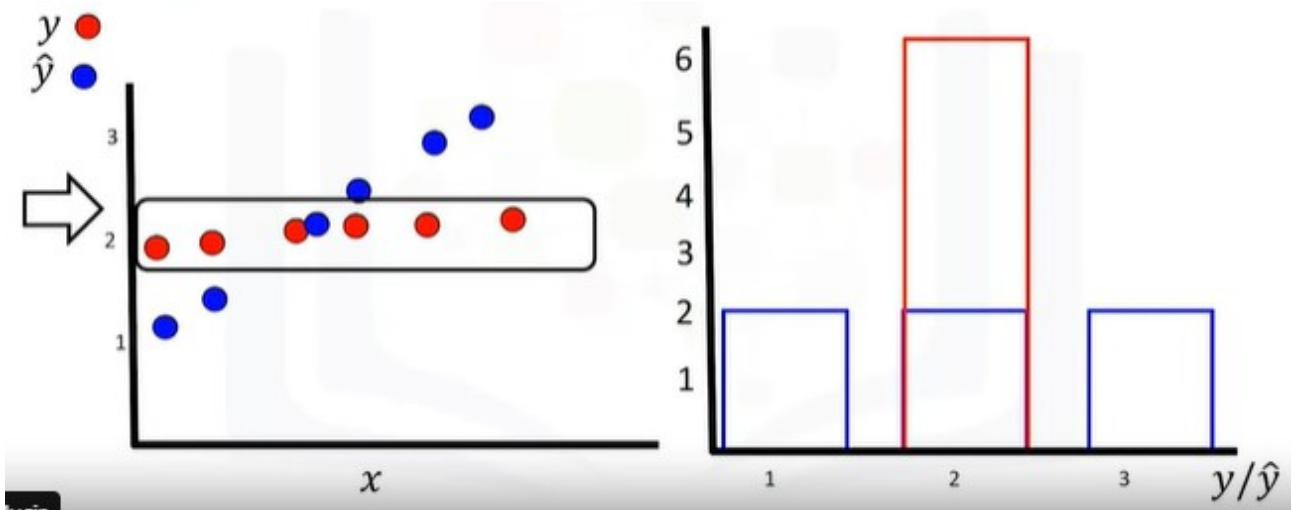


Figura. Gráfico de distribución.

Los valores de los objetivos y los valores predecidos son continuos. Un histograma se realiza para valores discretos, por tanto Pandas los convertirá a una distribución. El eje vertical es escalado de modo que el área bajo la distribución sea 1.

Este es un ejemplo de la utilización de un gráfico de distribución. La variable dependiente es el precio. Los valores ajustados que resultan del modelo están en azul. Los valores verdaderos están en rojo.

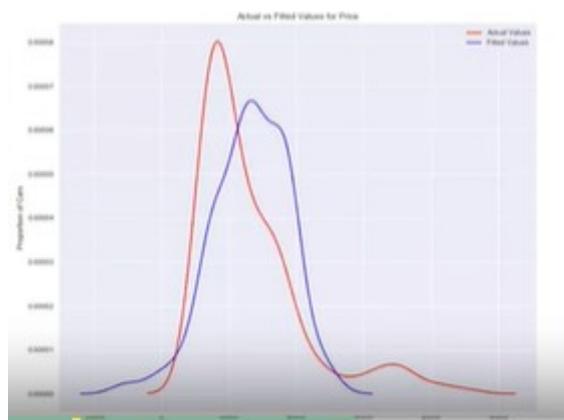


Figura. Gráfico de distribución.

Vemos que los valores predecidos para precios en el rango 40.000 a 50.000 son inexactos; los precios en la región de 10.000 a 20.000 están más cerca del objetivo. En este ejemplo usamos muchas características o variables independientes.

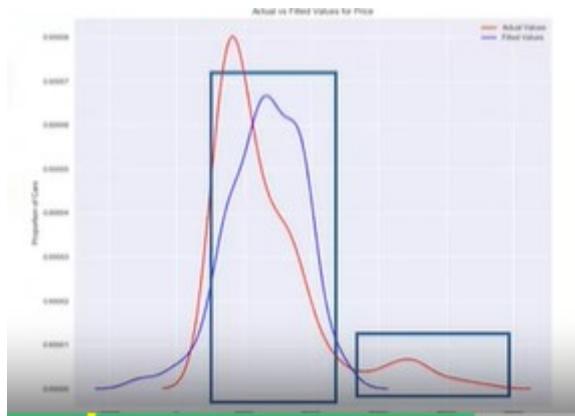


Figura. Análisis de un gráfico de distribución.

Aquí el código para crear un gráfico de distribución.

```
import seaborn as sns

ax1 = sns.distplot(df['price'], hist=False, color="r", label="Actual Value")

sns.distplot(Yhat, hist=False, color="b", label="Fitted Values" , ax=ax1)
```

Figura. Código para crear un gráfico de distribución.

- Los valores verdaderos se usan como parámetros.
- Queremos una distribución en lugar de un histograma por lo que el parámetro hist se establece en False.
- El color es rojo.
- Se incluye la etiqueta.
- Los valores predecidos se incluyen en el segundo gráfico.
- El resto de los parámetros se establecen acordemente.

4.4. REGRESIÓN POLINÓMICA Y PIPELINES

¿Qué hacemos cuando un modelo lineal no es el mejor ajuste para nuestros datos? Veamos otro tipo de modelo de regresión, **la regresión polinómica**.

Los **pipelines** son una forma de simplificar su código.

La regresión polinómica es un caso especial de la regresión lineal general. Este método es beneficioso para describir relaciones curvilíneas.

¿Qué es una relación curvilínea?

Es lo que obtienes al establecer términos de orden superior de las variables predictoras en el modelo que transforma los datos.

Esta es una regresión polinómica de segundo orden, con una figura que representa la función.

- **Quadratic – 2nd order**

$$\hat{Y} = b_0 + b_1 x_1 + b_2 (x_1)^2$$

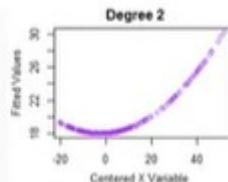


Figura. Ejemplo de modelo cuadrático.

El modelo también puede ser cúbico, lo que significa que la variable predictora es cúbica. Esta es la regresión polinómica de tercer orden. Vemos examinando la figura que la función tiene más variación respecto a la cuadrática.

- **Cubic – 3rd order**

$$\hat{Y} = b_0 + b_1 x_1 + b_2 (x_1)^2 + b_3 (x_1)^3$$

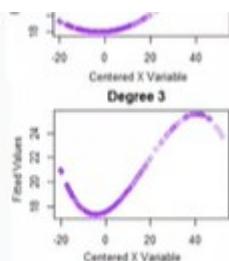


Figura. Ejemplo de modelo cúbico.

También existe regresiones polinomiales de orden superior que se utilizan cuando un buen ajuste no se ha logrado con modelos de segundo o tercer orden.

- Higher order

$$\hat{Y} = b_0 + b_1 x_1 + b_2 (x_1)^2 + b_3 (x_1)^3 + \dots$$

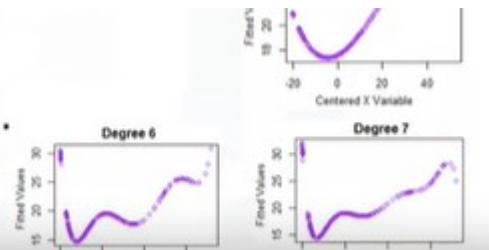


Figura. Modelos de orden superior.

Podemos ver en cifras cuánto cambian los gráficos, cuando cambiamos el orden de la regresión polinómica. El grado de regresión hace una gran diferencia y puede resultar en un mejor ajuste si elige el valor correcto. **En todos los casos, la relación entre la variable y el parámetro es siempre lineal.**

Veamos un ejemplo de nuestros datos donde generamos un modelo de regresión polinómica. En Python hacemos esto mediante el uso de la función polyfit. En este ejemplo, desarrollamos una base de modelo de regresión polinómica de tercer orden.

1. Calculate Polynomial of 3rd order

```
f=np.polyfit(x,y,3)
p=np.poly1d(f)
```

2. We can print out the model

```
print (p)
```

$$-1.557(x_1)^3 + 204.8(x_1)^2 + 8965 x_1 + 1.37 \times 10^5$$

Figura. Modelo de regresión cúbico en Python.

También podríamos tener **regresión lineal polinómica multidimensional**. Estos son solo algunos de los términos para un polinomio de segundo orden bidimensional:

$$\hat{Y} = b_0 + b_1 X_1 + b_2 X_2 + b_3 X_1 X_2 + b_4 (X_1)^2 + b_5 (X_2)^2 + \dots$$

La función de poliajuste de Numpy no puede realizar este tipo de regresión. Utilizamos la biblioteca de preprocesamiento en scikit-learn para crear un objeto de entidad polinómica. El constructor toma el grado del polinomio como parámetro. Luego transformamos las entidades en una entidad polinómica con el método de transformación de subrayado de ajuste.

```
from sklearn.preprocessing import PolynomialFeatures  
pr=PolynomialFeatures(degree=2, include_bias=False)
```

Figura. Regresión multidimensional con Scikit-Learn.

Hagamos un ejemplo más intuitivo. Considere las características X1 y X2 que se muestran en la figura siguiente. Aplicando el método transformamos los datos, ahora tenemos un nuevo conjunto de características que son una versión transformada de nuestras características originales.

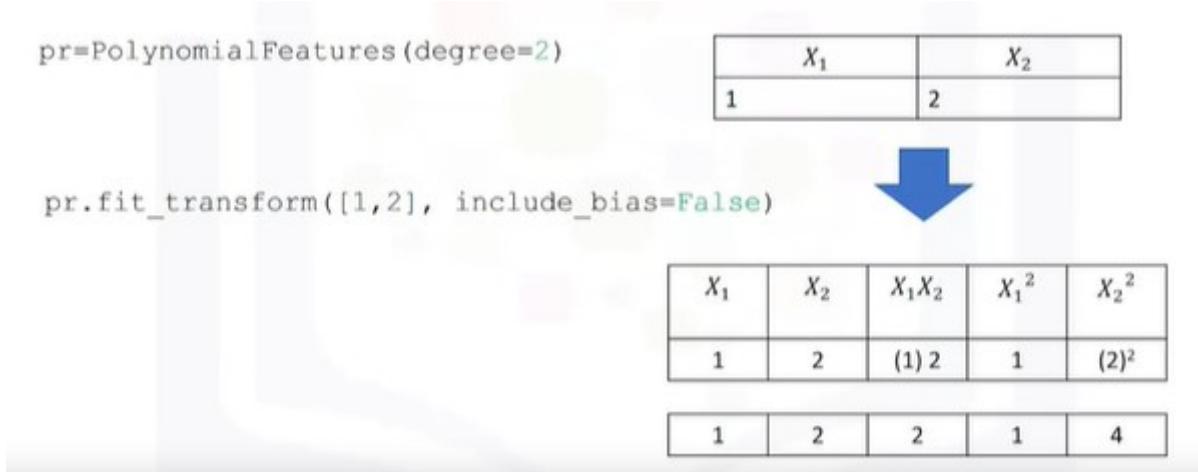


Figura. Transformación de datos.

A medida que la dimensión de los datos se hace más grande, es posible que queramos normalizar múltiples características en scikit-learn. En su lugar, podemos utilizar el módulo de preprocessamiento para simplificar muchas tareas. Por ejemplo, podemos estandarizar cada característica simultáneamente.

```
from sklearn.preprocessing import StandardScaler
SCALE=StandardScaler()
SCALE.fit(x_data[['horsepower', 'highway-mpg']])
x_scale=SCALE.transform(x_data[['horsepower', 'highway-mpg']])
```

Figura. Simplificando tareas con preprocessing.

Hay más métodos de normalización disponibles en la biblioteca de preprocessamiento, así como otras transformaciones.

Podemos simplificar nuestro código mediante el uso de una biblioteca de tuberías (pipelines).

Hay muchos pasos para obtener una predicción. Por ejemplo, normalización, transformación polinómica y regresión lineal.

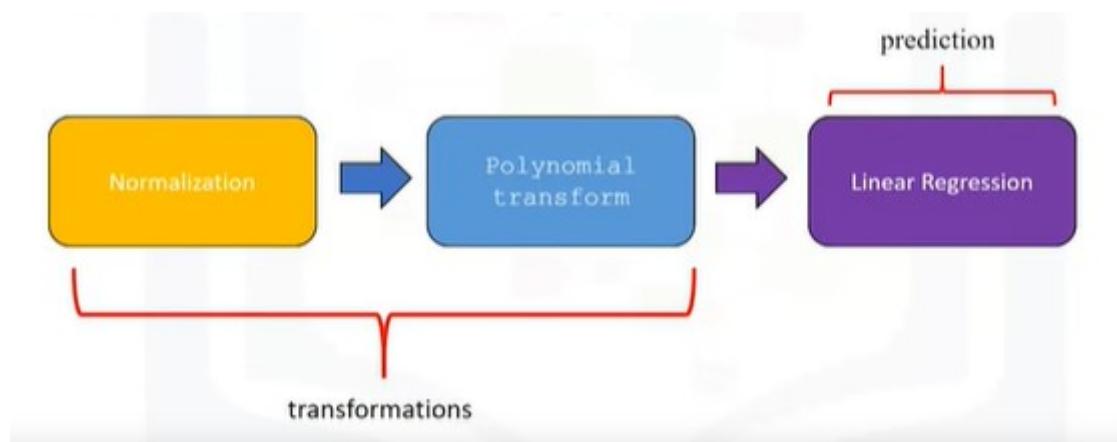


Figura. Pasos para obtener una predicción.

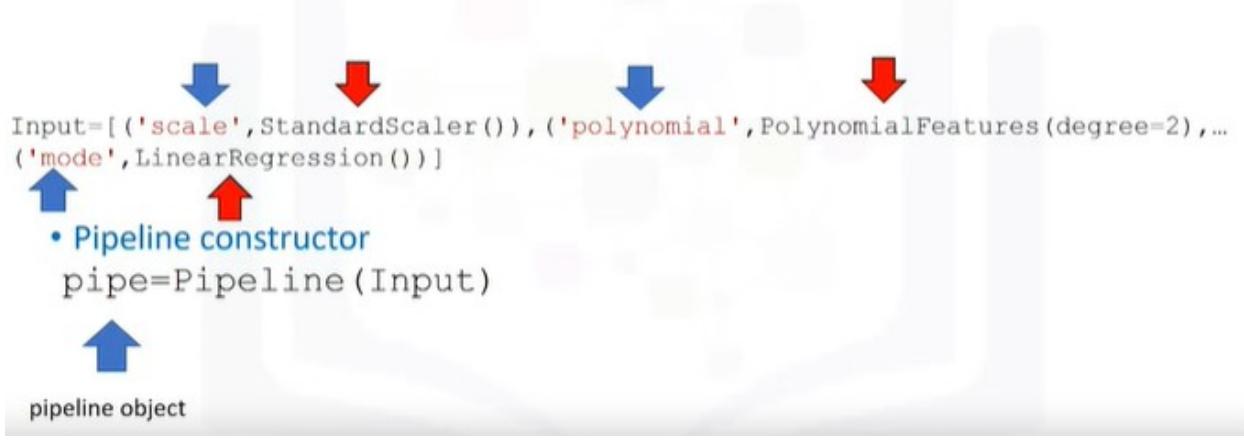
Simplificamos el proceso usando una tubería. Pipeline realiza secuencialmente una serie de transformaciones. El último paso lleva a cabo una predicción.

- Importamos todos los módulos que necesitamos.
- Creamos una lista de tuplas, el primer elemento de la tupla contiene el nombre del modelo estimador.
- El segundo elemento contiene el constructor del modelo.
- Ingresamos la lista en el constructor de la tubería.
- Ahora tenemos un objeto pipeline.
- Podemos entrenar el pipeline aplicando el método de entrenamiento al objeto de pipeline.
- También podemos producir una predicción.
- El método normaliza los datos, realiza una transformación polinómica, luego genera una predicción.

```

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

```



```

Pipe.fit(df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']],y)  

yhat=Pipe.predict(X[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']])

```



Figura. Utilización de un pipeline.

4.5. MEDIDAS PARA EVALUACIÓN IN-SAMPLE

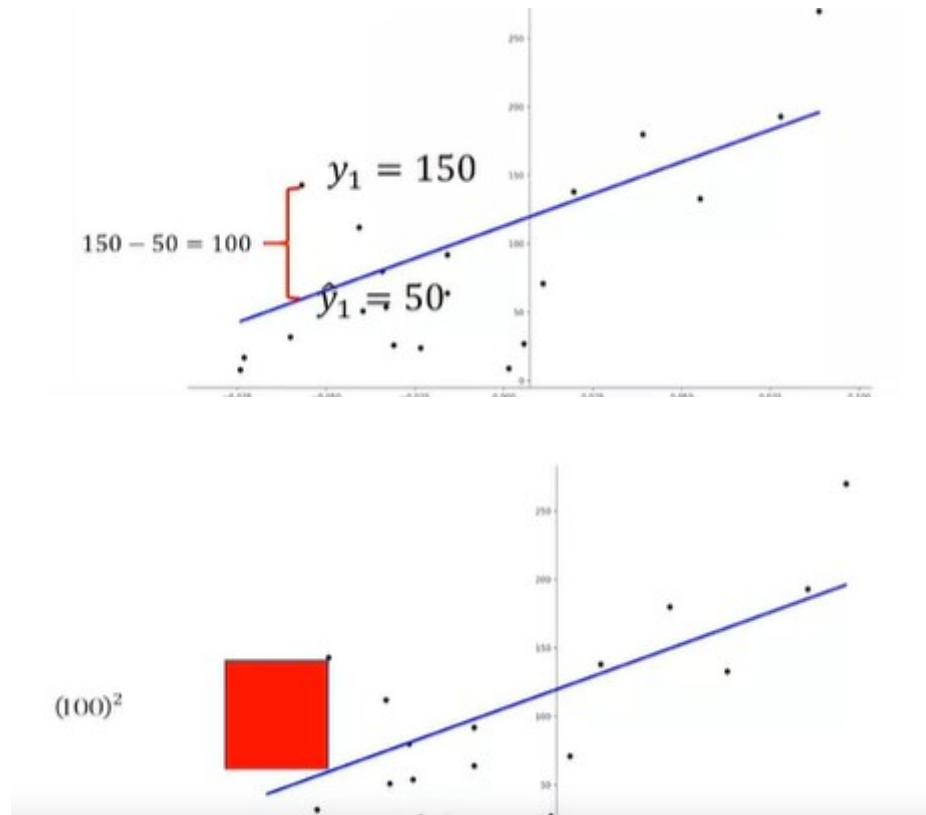
Ahora que hemos visto cómo podemos evaluar un modelo mediante la visualización, queremos evaluarlos numéricamente.

Veamos algunas de las medidas que utilizamos para la evaluación en la muestra (in-sample). Estas medidas son una forma de determinar numéricamente qué tan bueno encaja el modelo en nuestros datos.

Dos medidas importantes que utilizamos a menudo para determinar el ajuste de un modelo son:

- Error cuadrado medio (MSE)
- R-cuadrado.

Para medir el MSE, encontramos la diferencia entre el valor real y el valor predicho y luego elevamos al cuadrado. En el ejemplo de la figura siguiente, el valor real es 150; el valor previsto es 50. Al restar estos puntos obtenemos 100. Luego cuadramos el número. Luego tomamos la media o media de todos los errores agregando luego todos juntos y dividiendo por el número de muestras.



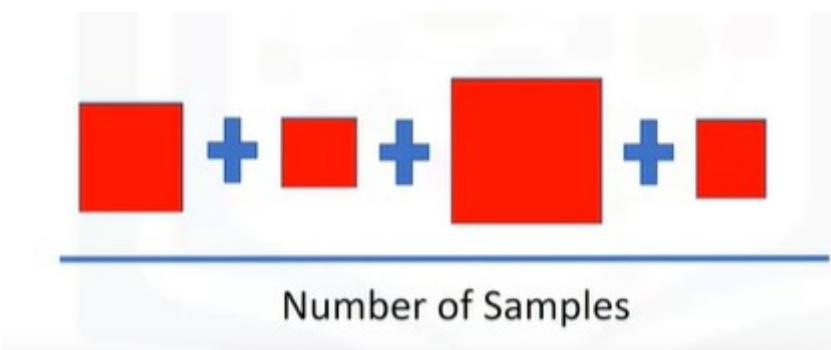


Figura. Obteniendo el MSE.

Para encontrar el MSE en Python, podemos importar el «MEAN_Squared_Error» de «scikit-learn.metrics». La función «MEAN_Squared_Error» tiene dos entradas: el valor real de la variable objetivo y el valor predicho de la variable objetivo.

```
from sklearn.metrics import mean_squared_error  
  
mean_squared_error(df['price'], Y_predict_simple_fit)  
  
3163502.944639888
```

R-cuadrado:

- También se llama coeficiente de determinación.
- Es una medida para determinar qué tan cerca están los datos de la línea de regresión ajustada.
- Entonces, ¿qué tan cerca están nuestros datos reales de nuestro modelo estimado?
 - Piense en ello como comparar un modelo de regresión con un modelo simple, es decir, la media de los puntos de datos. Si la variable x es un buen predictor nuestro modelo debería funcionar mucho mejor que [con] solo la media.

$$R^2 = \left(1 - \frac{\text{MSE of regression line}}{\text{MSE of the average of the data}} \right)$$

Figura. Cálculo de R^2 .

En su mayor parte, toma valores entre 0 y 1.

Veamos un caso donde la línea proporciona un ajuste relativamente bueno.

- La línea azul representa la línea de regresión.
- Los cuadrados azules representan el MSE de la línea de regresión.
- La línea roja representa el valor medio de los puntos de datos.
- Los cuadrados rojos representan el MSE de la línea roja.

Vemos que el área de los cuadrados azules es mucho más pequeña que el área de los cuadrados rojos.

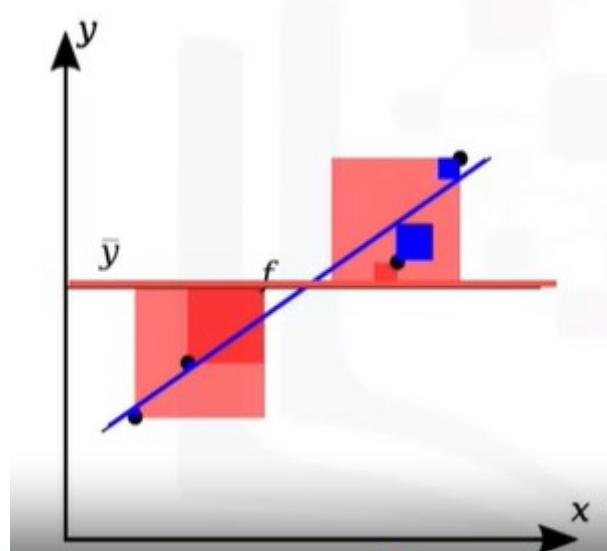


Figura. Ajuste relativamente bueno usando una recta.

En este caso, debido a que la línea es un buen ajuste, el error cuadrado medio es pequeño, por lo tanto, el numerador es pequeño. El error medio cuadrado de la línea es relativamente grande, como tal el numerador es grande. Un número pequeño dividido por un número mayor es un número aún más pequeño. Llevado al extremo este valor tiende a cero. Si

conectamos este valor de la diapositiva anterior para R^2 , obtenemos un valor cercano a uno, esto significa que la línea es un buen ajuste para los datos.

Aquí hay un ejemplo de una línea que no se ajusta bien a los datos.

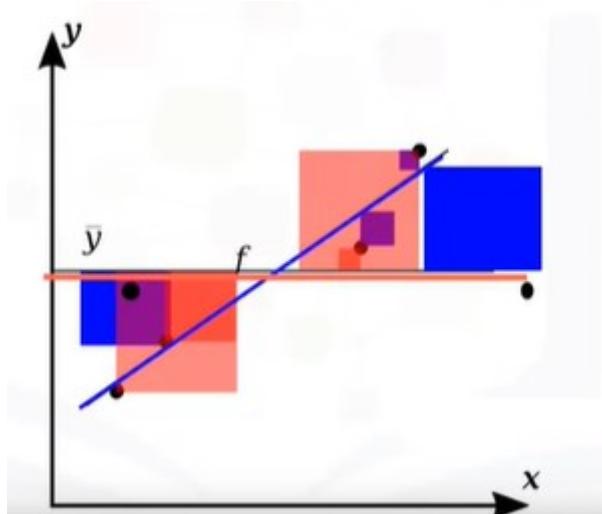


Figura. Recta que no se ajusta bien a los datos.

Si sólo examinamos el área de los cuadrados rojos en comparación con los cuadrados azules, vemos que el área es casi idéntica. La proporción de las áreas es cercana a una. En este caso, el R^2 está cerca de cero. Esta línea realiza aproximadamente lo mismo que simplemente usando el promedio de los puntos de datos, por lo tanto, esta línea no funcionó bien.

Encontramos el valor R cuadrado en Python utilizando el método de `score`, del objeto de regresión lineal.

```
X = df[['highway-mpg']]
Y = df['price']

lm.fit(X, Y)

lm.score(X, y)
0.496591188
```

Figura. R^2 en Python.

A partir del valor que obtenemos de este ejemplo, podemos decir que aproximadamente el 49,695% de la variación de precio se explica por este simple modelo lineal.

Si R^2 es negativo, puede deberse a exceso de ajuste (overfitting) que discutiremos en el siguiente módulo.

4.6. PREDICCIÓN Y TOMA DE DECISIONES

¿Cómo podemos determinar si nuestro modelo es correcto?

Lo primero que debe hacer es asegurarse de que los resultados de su modelo tengan sentido.

Siempre debe utilizar:

- La visualización.
- Medidas numéricas para la evaluación.
- Comparación entre diferentes modelos.

Veamos un ejemplo de predicción.

Si recuerda, entrenamos el modelo usando el método de ajuste (fit).

Ahora queremos averiguar cuál sería el precio para un coche que tiene una autopista millas por galón de 30.

Conectar este valor en el método de predicción nos da un precio resultante de \$13,771,30.

Esto parece tener sentido, el valor no es ni demasiado alto ni demasiado bajo.

Podemos ver los coeficientes examinando el `coef_attribute`.

Si recuerda la expresión para el modelo lineal simple que predice el precio de millas de carretera por galón. Este valor corresponde al múltiplo de la característica de millas de carretera por galón, como tal, un aumento de una unidad en millas de carretera por galón. El valor del coche disminuye aproximadamente \$821. Este valor también parece razonable.

- First we train the model
`lm.fit(df['highway-mpg'], df['prices'])`
 - Let's predict the price of a car with 30 highway-mpg.
`lm.predict(np.array(30.0).reshape(-1,1))`
 - Result: \$ 13771.30
`lm.coef_`
-821.73337832
- Price = 38423.31 - 821.73 * highway-mpg

Figura. Ejemplo de predicción.

A veces su modelo producirá valores que no tienen sentido. Por ejemplo, si trazamos el modelo para millas de carretera por galón en los rangos de 0 a 100 obtenemos valores

negativos para el precio. Esto podría deberse a que los valores en ese rango no son realistas. La suposición lineal es incorrecta o no tenemos datos de coches en ese rango. En este caso, es poco probable que un coche tenga kilometraje de combustible en ese rango, así que nuestro modelo parece válido igualmente.

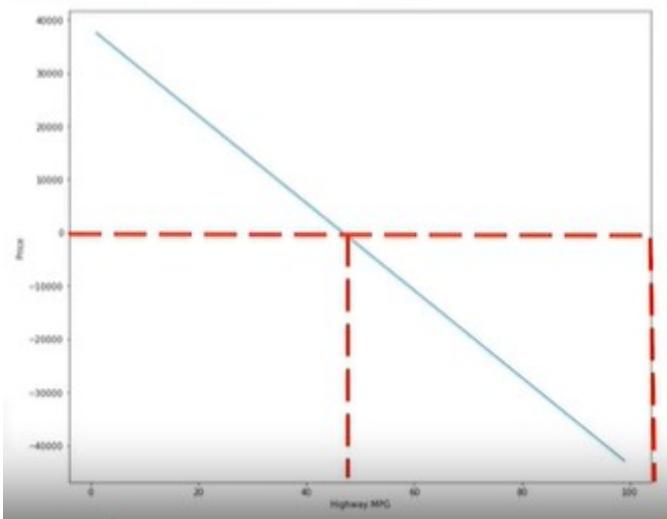


Figura. Nuestro modelo puede producir valores que no tengan sentido.

Para generar una secuencia de valores en un rango especificado, importa numpy y, a continuación, utiliza la función numpy arrange para generar la secuencia.

En el ejemplo siguiente la secuencia comienza en uno y se incrementa en uno hasta llegar a 100.

- El primer parámetro es el punto de partida de la secuencia.
- El segundo parámetro es el punto final más uno de la secuencia.
- El parámetro final es el tamaño del paso entre los elementos de la secuencia. En este caso, es uno.

- First we import numpy
`import numpy as np`
- We use the numpy function `arrange` to generate a sequence from 1 to 100
new_input=np.arange(1,101,1).reshape(-1,1)

1	2	...	99	100
---	---	-----	----	-----

Figura. Generando valores en un rango con numpy.

Podemos usar la salida para predecir nuevos valores. La salida es una matriz numpy, muchos de los valores son negativos.

```
yhat=lm.predict(new_input)
```

```
array([-37601.57247984, -36779.83910151, -35958.10572319, -35136.37234487,  
       34314.63894655,  33492.90558823,  32671.1722099,  31849.43883158,  
       31027.70545326,  30205.97207494,  29384.23849642,  28542.50531829,  
       27740.77193997,  26919.03856165,  26097.30518333,  25275.57180501,  
       24453.83842668,  23632.10594834,  22810.37167094,  21998.63829172,  
       21166.9049134,   20345.17153508,  19523.43815675,  18701.70477843,  
       17879.97140011,  17058.23802179,  16236.50444347,  15414.77126514,  
       14593.03789682,  13771.3045085,  12949.57113018,  12127.83775186,  
       11306.10437353,  10484.37099521,  9662.63761689,  8840.90423857,  
       8019.17086025,  7197.43748192,  6375.7041036,  5553.97072528,  
       4732.23734696,   3910.50396864,  3088.77059031,  2267.03721199,  
       1445.30383367,   623.57045535,  -198.16292297,  -1019.8963013,  
      -1841.62967962,  -2463.36305579,  -3485.09643626,  -4306.82981458,  
      -5128.5631929,  -5950.29657123,  -6772.02994955,  -7593.76332787,  
      -8415.49670619,  -9237.23008451,  -10058.96346284,  -10880.69684116,  
      -11702.43021948,  -12524.1635978,  -13345.89697612,  -14167.63035445,  
      -14989.36373277,  -15811.09711109,  -16632.83048945,  -17454.56386773,  
      -18276.29724604,  -19098.03042438,  -19919.7640027,  -20741.49738102,  
      -21563.23015934,  -22384.96413767,  -23209.69751599,  -24028.43089431,  
      -24850.16427263,  -25671.89765099,  -26493.63102927,  -27311.3644076,  
      -28137.59778592,  -28958.83116424,  -29780.56454256,  -30602.29792088,  
      -31424.03129921,  -32245.76467753,  -33067.49805585,  -33899.23143417,  
      -34710.96481249,  -35532.69819082,  -36354.42156934,  -37176.16494746,  
      -37997.99822579,  -38819.6317041,  -39641.36598243,  -40463.09848979,
```

Figura. Prediciendo valores.

Usar un diagrama de regresión para visualizar sus datos es el primer método que debe probar.

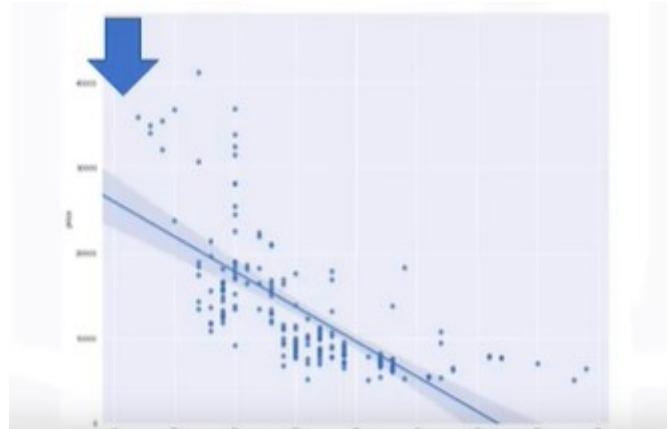


Figura. Diagrama de regresión.

Para este ejemplo, el efecto de la variable independiente es evidente.

Las tendencias de los datos bajan a medida que aumenta la variable dependiente.

La trama también muestra un comportamiento no lineal.

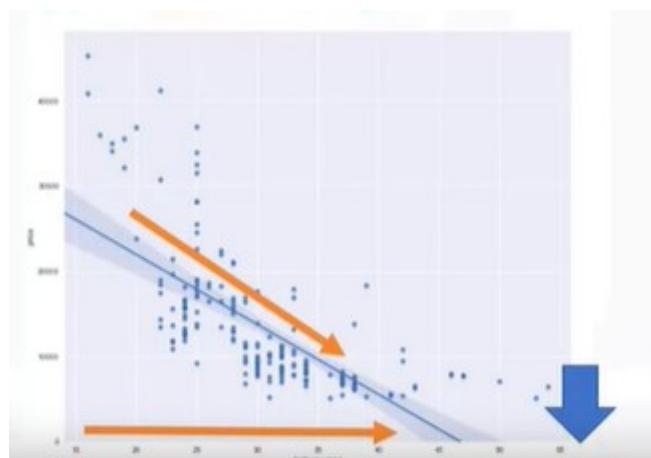


Figura. Análisis del diagrama de regresión.

Examinando el trazado residual, vemos en este caso que los residuales tienen una curvatura que sugiere un comportamiento no lineal.

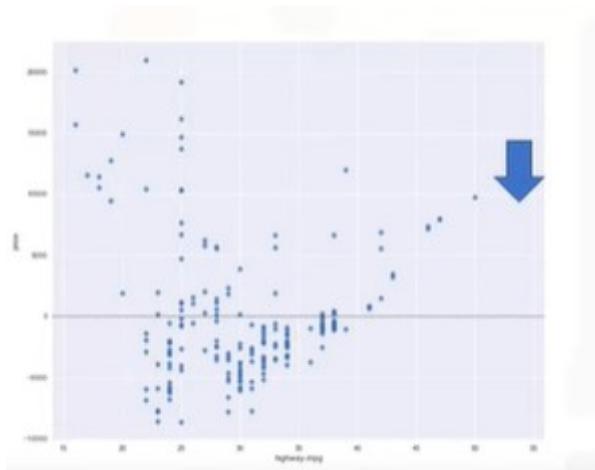


Figura. El gráfico residual sugiere un comportamiento no lineal.

Un diagrama de distribución es un buen método para la regresión lineal múltiple. Por ejemplo, vemos que los valores predictivos para los precios en el rango de 30,000 a 50,000 son inexactos. Esto sugiere que un modelo no lineal puede ser más adecuado o necesitamos más datos en este rango.

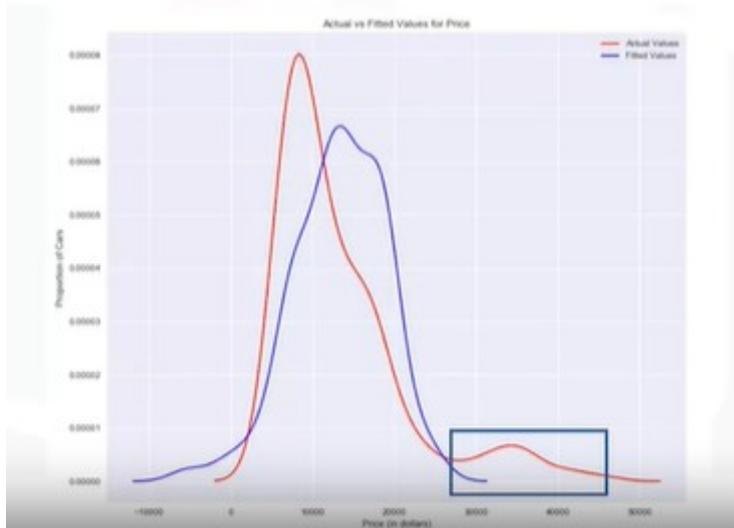


Figura. Vemos que en el rango de 30.000 a 50.000 los precios son inexactos.

El error cuadrado medio es quizás la medida numérica más intuitiva para determinar si un modelo es bueno o no. Veamos cómo las diferentes medidas de error cuadrado medio impactan en el modelo. La figura muestra un ejemplo de un error cuadrado medio de 3.495.

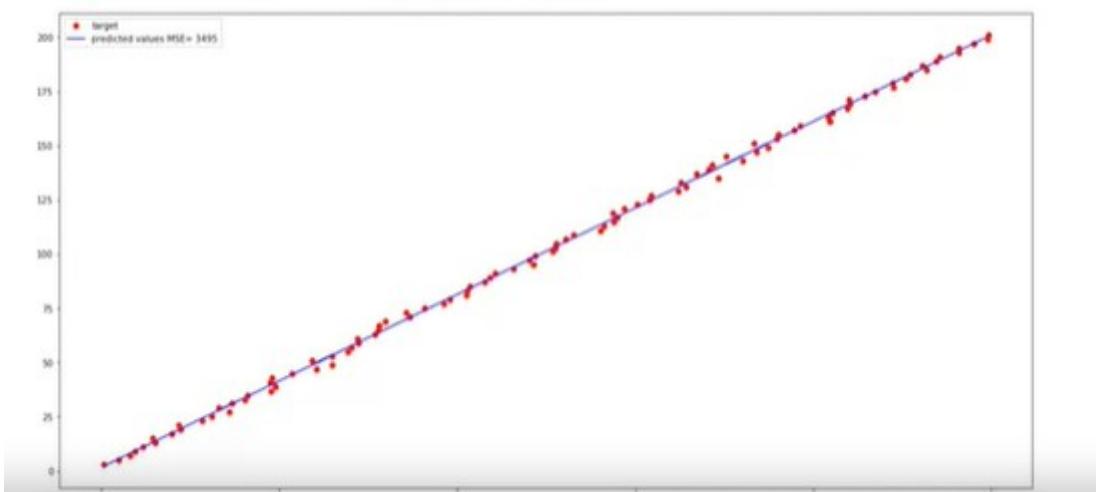


Figura. Error cuadrático medio de 3495.

Este ejemplo tiene un error cuadrado medio de 3.652.

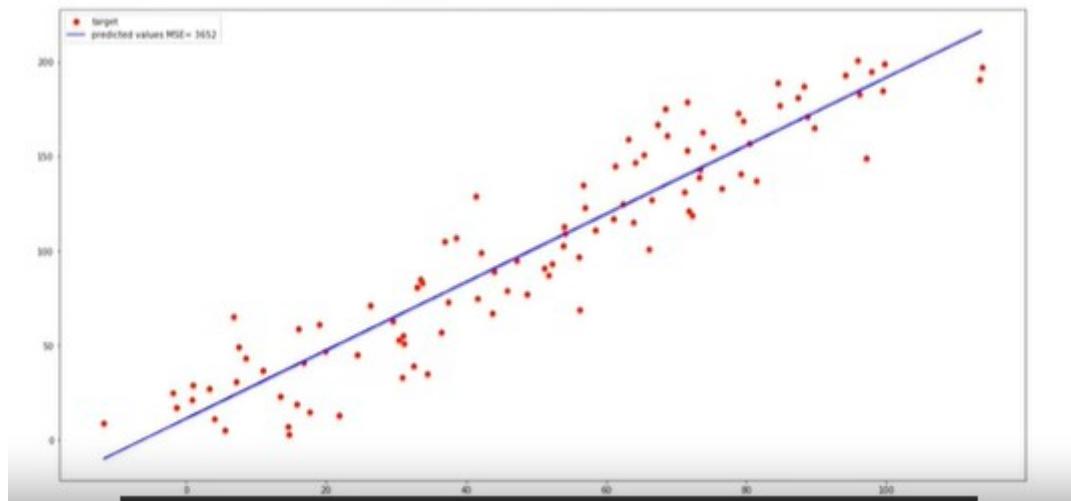


Figura. Error cuadrático medio de 3652.

La trama final tiene un error cuadrado medio de 12.870.

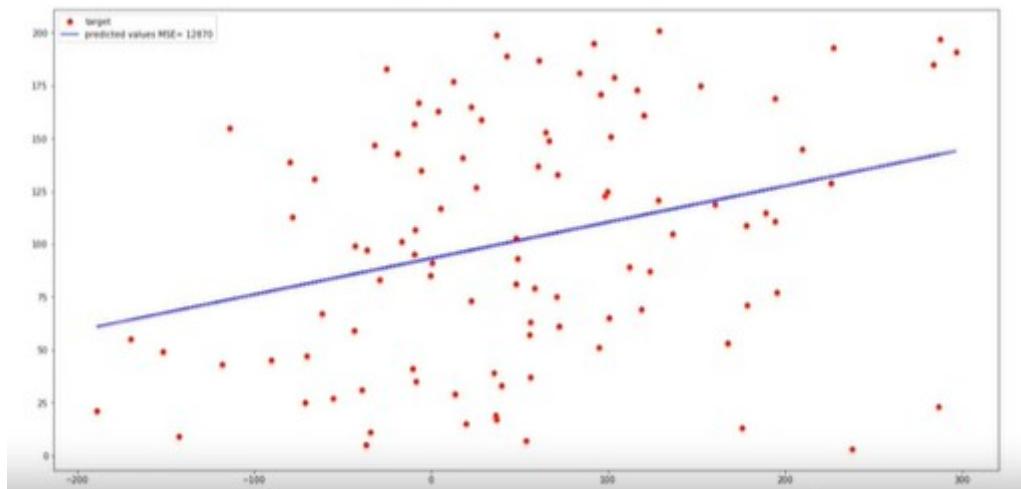


Figura. Error cuadrático medio de 12870.

A medida que aumenta el error cuadrado, los objetivos van más lejos de los puntos previstos.

Como hemos discutido, R-cuadrado es otro método popular para evaluar su modelo.

En esta trama, vemos los puntos objetivo en rojo y la línea prevista en azul siendo R cuadrado igual a 0.9986. El modelo parece ser un buen ajuste.

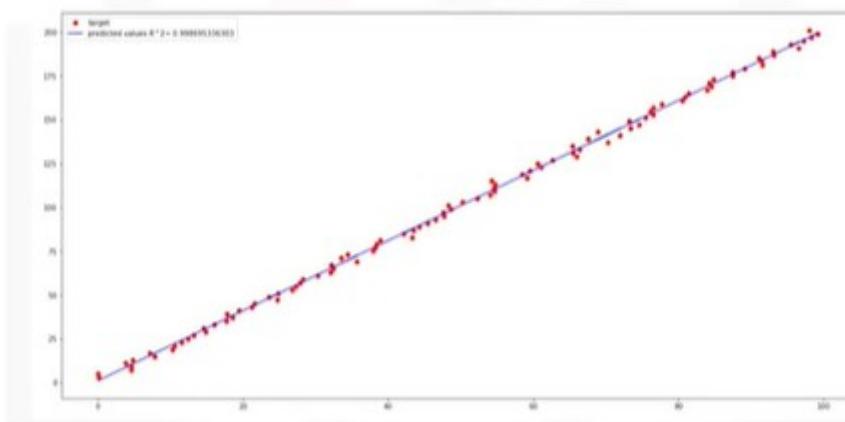


Figura. $R^2 = 0.9986$.

Este modelo tiene un R cuadrado de 0.9226. Todavía hay una fuerte relación lineal.

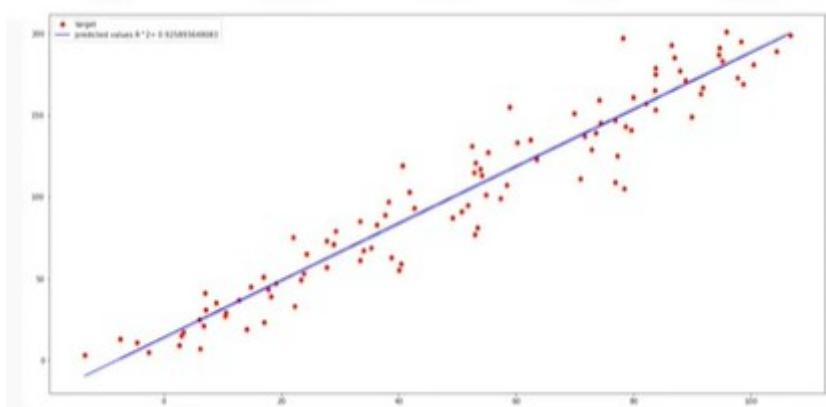


Figura. $R^2 = 0.9226$.

Con R cuadrado de 0.806, los datos son mucho más desordenados pero la relación lineal es evidente.

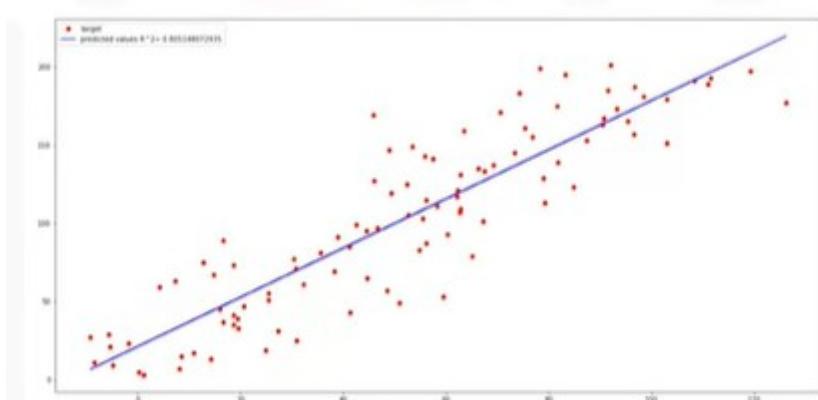


Figura. $R^2 = 0.806$.

Con R-cuadrado 0.61, la función lineal es más difícil de ver, pero en una inspección más cercana vemos los datos están aumentando con la variable independiente.

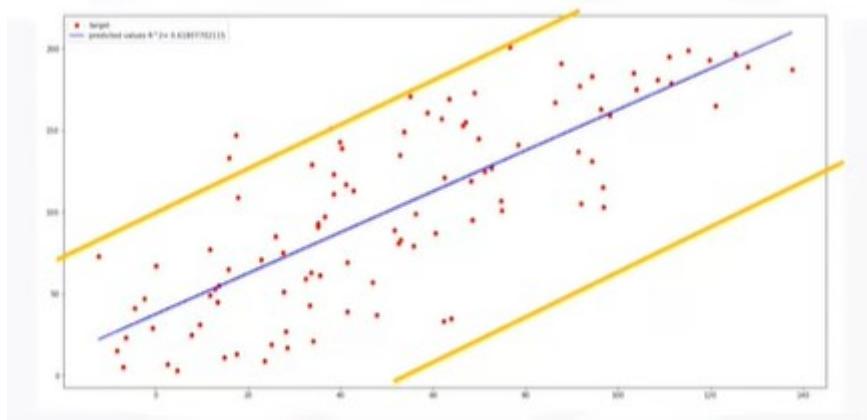


Figura. $R^2 = 0.61$.

El valor aceptable para R-cuadrado depende del campo que estés estudiando. Comparando MLR y SLR:

¿Un MSE más bajo siempre implica un mejor ajuste?

No necesariamente.

MSE para un modelo MLR será menor que el MSE para un modelo SLR, ya que los errores de los datos disminuirán cuando se incluyan más variables en el modelo. La regresión polinómica también tendrá un MSE menor que la regresión regular. Una relación inversa similar se mantiene para R cuadrado.

En la siguiente sección, veremos mejores maneras de evaluar el modelo.

4.7. LABORATORIO

Notebook: 4. model-deployment

PARTE 5. EVALUACIÓN DEL MODELO Y REFINAMIENTO

5.1. EVALUACIÓN DEL MODELO Y REFINAMIENTO

La evaluación del modelo nos dice cómo funciona nuestro modelo en el mundo real.

En el módulo anterior, hablamos de la evaluación en muestra, que nos dice qué tan bien nuestro modelo se ajusta a los datos ya dados para entrenarlo. No nos da una estimación de lo bien que el modelo de entrenamiento puede predecir nuevos datos.

La solución es dividir nuestros datos:

- Se utilizan los **datos de muestra** o los datos de entrenamiento para entrenar el modelo.
- El resto de los datos, llamados **datos de prueba (test)**, se utilizan como datos fuera de la muestra.
 - Estos datos se utilizan entonces para aproximar el rendimiento del modelo en el mundo real.

La separación de datos en conjuntos de entrenamiento y pruebas es una parte importante de la evaluación del modelo. Utilizamos los datos de prueba para tener una idea de cómo funcionará nuestro modelo en el mundo real.

Cuando dividimos un conjunto de datos, generalmente la mayor parte de los datos se utiliza para el entrenamiento y una parte más pequeña se utiliza para las pruebas. Por ejemplo, podemos usar el 70 por ciento de los datos para el entrenamiento. Luego usamos el 30 por ciento para las pruebas.



Figura. Dividiendo los datos en entrenamiento y test.

- Utilizamos el conjunto de entrenamiento para construir un modelo y descubrir relaciones predictivas.
- A continuación, utilizamos un conjunto de pruebas para evaluar el rendimiento del modelo.
- Cuando hayamos terminado de probar nuestro modelo, deberíamos usar todos los datos para entrenarlo.

Para dividir los conjuntos de datos podemos usar la función train_test_split de scikit-learn. Esta función divide aleatoriamente un conjunto de datos en subconjuntos de entrenamiento y pruebas.

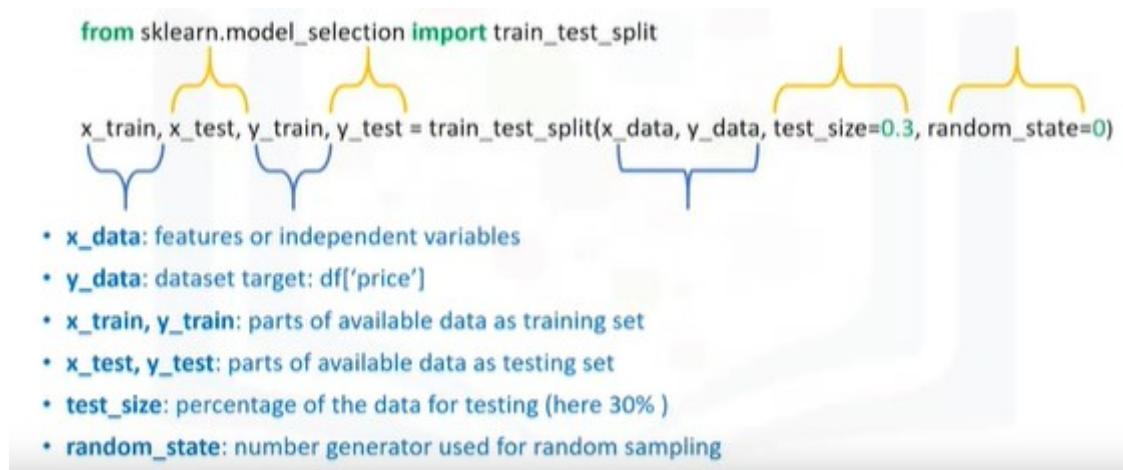


Figura. Dividiendo los conjuntos de datos en entrenamiento y test.

El estado aleatorio es una semilla aleatoria para la división de conjuntos de datos aleatorios. El **error de generalización** es una medida de lo bien que hacen nuestros datos al predecir datos previamente no vistos.

El error que obtenemos usando nuestros datos de prueba es una aproximación a este error. La figura siguiente a la izquierda muestra la distribución de los valores reales en rojo comparado con los valores predichos de una regresión lineal en azul. Vemos que las distribuciones son algo similares. Si generamos el mismo diagrama usando los datos de prueba, vemos que las distribuciones son relativamente diferentes. La diferencia se debe a un error de generalización y representa lo que vemos en el mundo real.

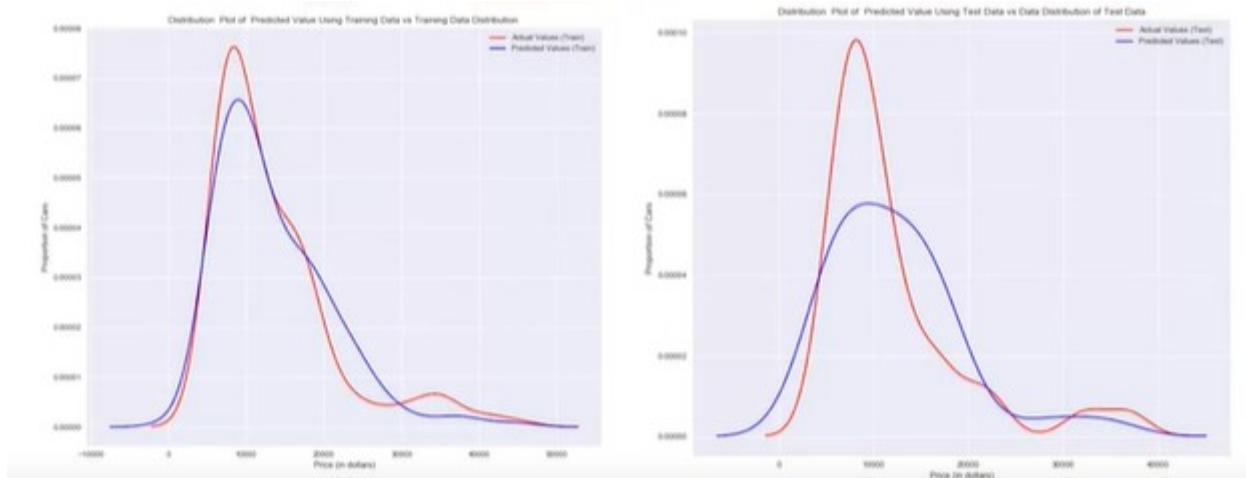


Figura. Gráficos de distribución; en el de la derecha se utilizaron los valores de prueba.

Usar una gran cantidad de datos para el entrenamiento nos da un medio preciso para determinar qué tan bien funcionará nuestro modelo en el mundo real. Pero la precisión del rendimiento será baja.

Vamos a aclarar esto con un ejemplo. El centro de este ojo de toro representa el error de generalización correcto. Supongamos que tomamos una muestra aleatoria de los datos usando 90 por ciento de los datos para entrenamiento y 10 por ciento para pruebas. La primera vez que experimentamos, obtenemos una buena estimación de los datos de entrenamiento. Si experimentamos de nuevo entrenando el modelo con una combinación diferente de muestras, también obtenemos un buen resultado. Pero, los resultados serán diferentes en relación con la primera vez que ejecutamos el experimento. Repitiendo el experimento de nuevo con una combinación diferente de entrenamiento y pruebas de muestras, los resultados son relativamente cercanos al error de generalización, pero distintos unos de otros. Repitiendo el proceso, obtenemos una buena aproximación del error de generalización, pero la precisión es pobre, es decir, todos los resultados son extremadamente diferentes entre sí.

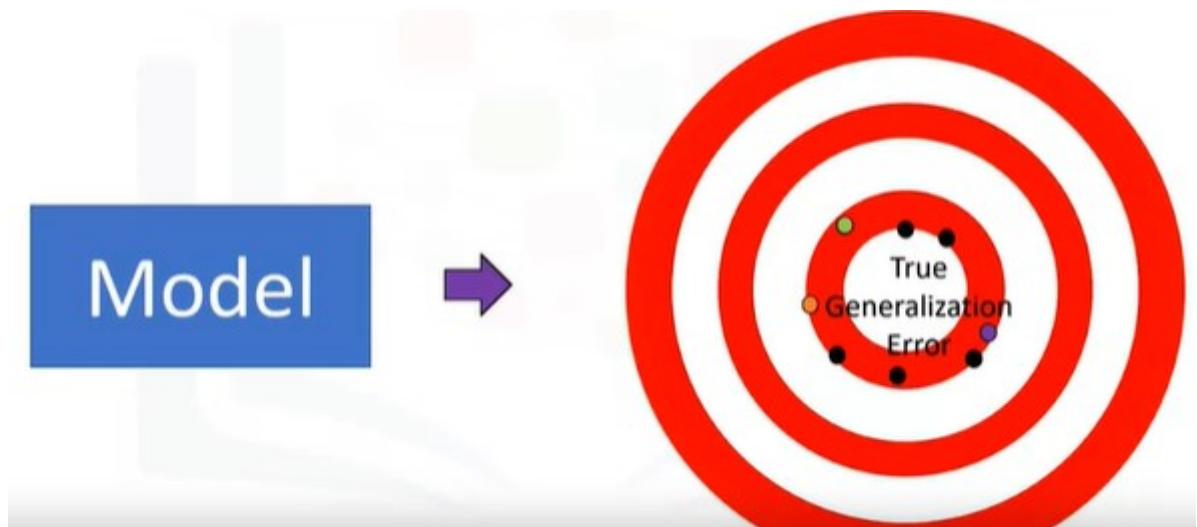


Figura. Buena aproximación del error de generalización pero mala precisión (resultados muy diferentes entre sí).

Si usamos menos puntos de datos para entrenar el modelo y más para probar el modelo, la precisión del rendimiento de generalización será menor, pero el modelo tendrá una buena precisión. La figura anterior lo demuestra. Todas nuestras estimaciones de errores son relativamente cercanas, pero están más lejos del verdadero error de generalización.



Figura. Precisión del rendimiento de generalización menor pero mayor precisión (nuestras estimaciones son cercanas entre sí).

Para superar este problema, utilizamos la **validación cruzada**.

Una de las métricas de evaluación de muestras más comunes es la validación cruzada. En este método:

- El conjunto de datos se divide en K grupos iguales.
- Cada grupo se conoce como un pliegue (fold).
 - Por ejemplo, pueden tenerse cuatro pliegues.
- Algunos de los pliegues se pueden utilizar como un conjunto de entrenamiento que usamos para entrenar el modelo y las partes restantes se utilizan como un conjunto de prueba, que usamos para probar el modelo. P
 - Por ejemplo, podemos usar tres pliegues para entrenar, y luego usar un pliegue para probar.
- Esto se repite hasta que cada partición se utiliza tanto para entrenamiento como para pruebas.
- Al final, usamos los resultados promedio como la estimación del error fuera de muestra.

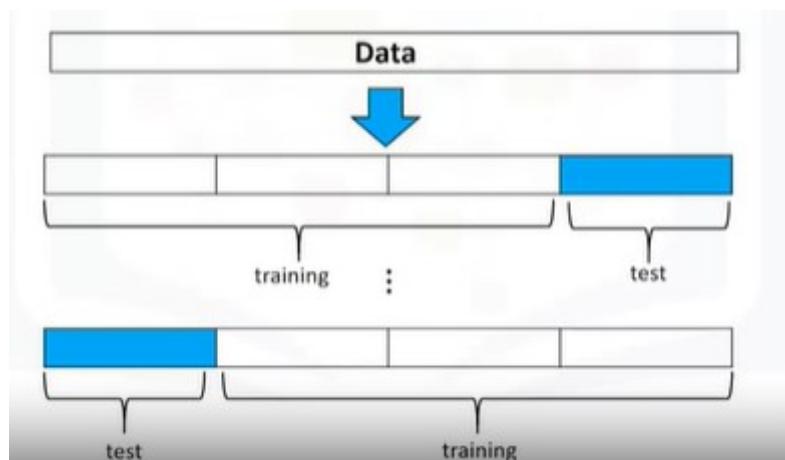


Figura. Validación cruzada.

La métrica de evaluación depende del modelo, por ejemplo, el r cuadrado.

La forma más sencilla de aplicar la validación cruzada es llamar a la función `cross_val_score`, que realiza múltiples evaluaciones fuera de muestra. Este método se importa del paquete de selección de modelo de `sklearn`. Luego usamos la función `cross_val_score`.

Los primeros parámetros de entrada son:

- El tipo de modelo que estamos utilizando para hacer la validación cruzada.

- En este ejemplo, inicializamos un modelo de regresión lineal u objeto lr que pasamos la función cross_val_score.
- x_data: los datos de la variable predictiva.
- y_data: los datos de la variable objetivo.
- Podemos gestionar el número de particiones con el parámetro cv.
 - En el ejemplo cv es igual a tres, lo que significa que el conjunto de datos se divide en tres particiones iguales.

La función devuelve una matriz de puntuaciones, una para cada partición que se eligió como el conjunto de pruebas.

Podemos promediar el resultado juntos para estimar de muestra r al cuadrado usando la función mean de NumPy.

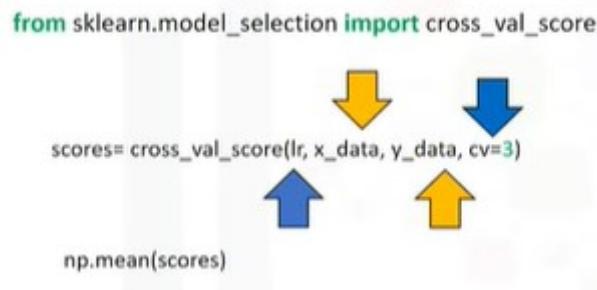


Figura. Validación cruzada en Python.

La función cross_val_score devuelve un valor de puntuación para indicarnos el resultado de la validación cruzada.

¿Y si queremos un poco más de información? ¿Qué pasa si queremos conocer los valores predichos reales suministrados por nuestro modelo antes de que se calculen los valores r cuadrados?

Para ello, utilizamos la función cross_val_predict.

Los parámetros de entrada son exactamente los mismos que la función cross_val_score, pero la salida es una predicción.

```
from sklearn.model_selection import cross_val_predict  
yhat= cross_val_predict (lr2e, x_data, y_data, cv=3)
```

Figura. Uso de cross_val_predict.

Las predicciones se almacenan en una matriz.

5.2. OVERFITTING, UNDERFITTING Y SELECCIÓN DEL MODELO

En esta sección, vamos a discutir cómo elegir el mejor orden polinomial y los problemas que surgen al seleccionar el polinomio de orden incorrecto.

Consideré la siguiente función, asumimos que los puntos de entrenamiento provienen de una función polinómica más algo de ruido. El objetivo de la Selección de Modelo es determinar el orden del polinomio para proporcionar la mejor estimación de la función $y(x)$.

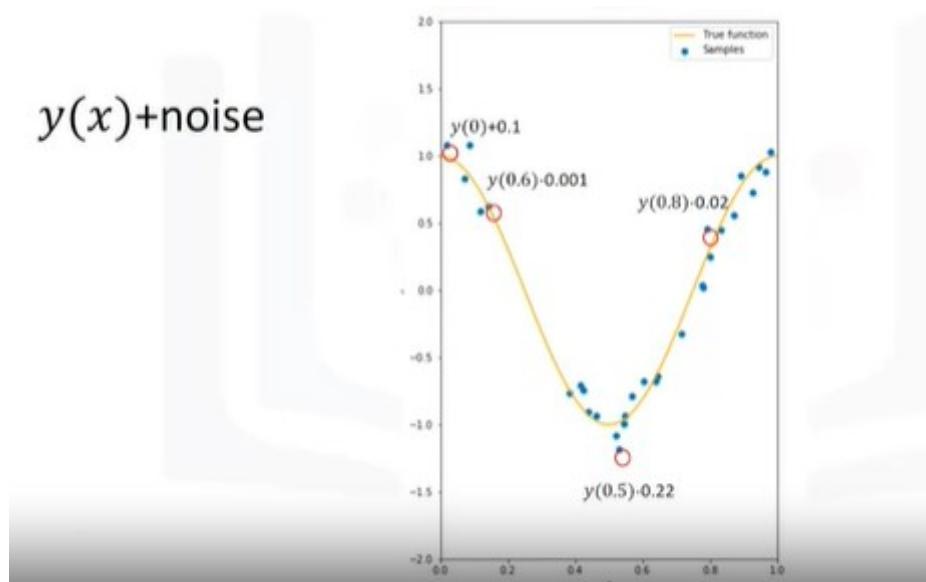


Figura. Función a aproximar: un polinomio más algo de ruido.

Si intentamos ajustar con una función lineal, veremos que en este caso la recta no es lo suficientemente compleja como para ajustarse a los datos. Como resultado, hay muchos errores. Esto se llama **underfitting**, donde el modelo es demasiado simple para ajustarse a los datos.

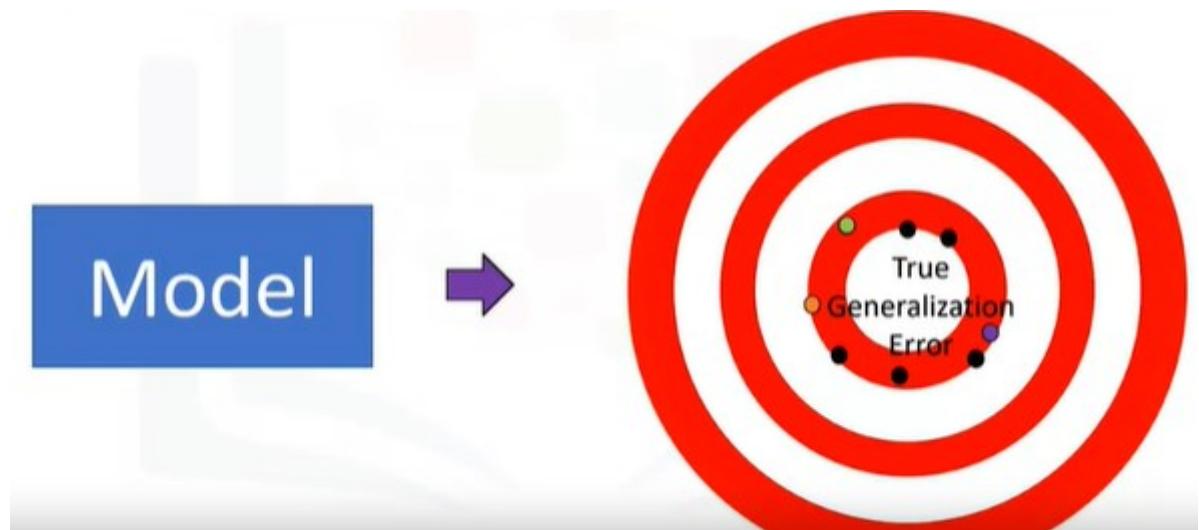


Figura. En este caso si intentamos ajustar con un modelo lineal tenemos underfitting.

Si aumentamos el orden del polinomio, el modelo encaja mejor, pero todavía no es lo suficientemente flexible y exhibe underfitting.

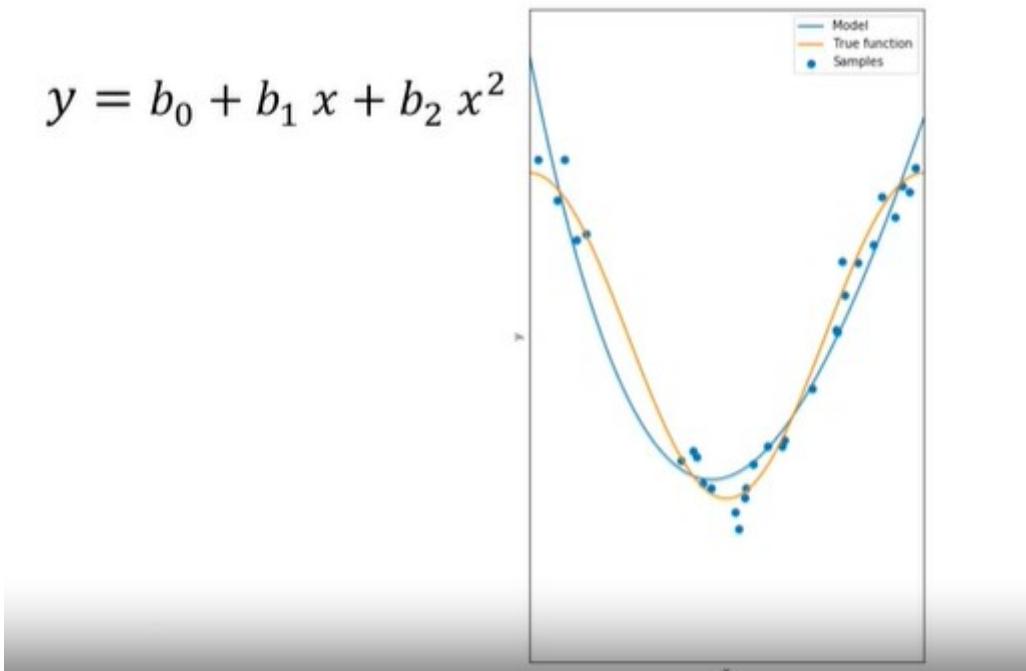


Figura. En este caso un modelo de segundo orden funciona mejor, pero sigue teniendo underfitting.

Este es un ejemplo del polinomio de octavo orden utilizado para ajustarse a los datos. Vemos que el modelo hace bien ajustando los datos y estimando la función incluso en los puntos de inflexión.

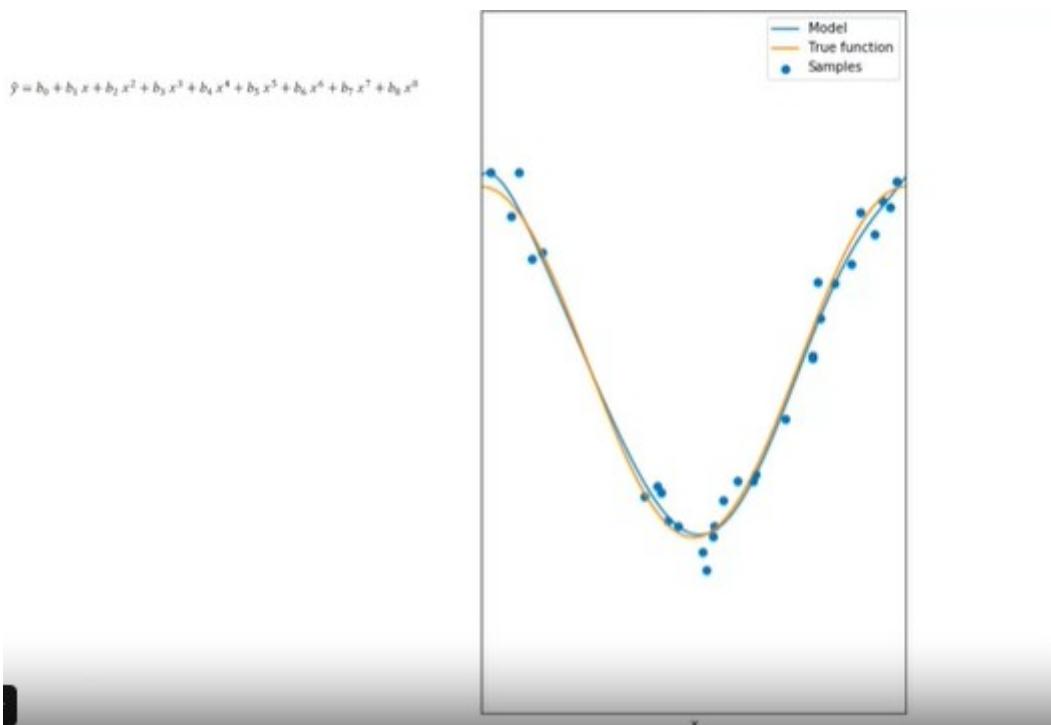


Figura. En este caso un polinomio de octavo grado produce un buen ajuste.

Incrementándolo a un polinomio de 16 orden, el modelo hace muy bien en el seguimiento del punto de entrenamiento, pero tiene un rendimiento deficiente en la estimación de la función. Esto es especialmente evidente cuando hay pocos datos de entrenamiento. La función estimada oscila sin rastrear la función. Esto se llama **overfitting**, donde el modelo es demasiado flexible y se ajusta al ruido en lugar de a la función.

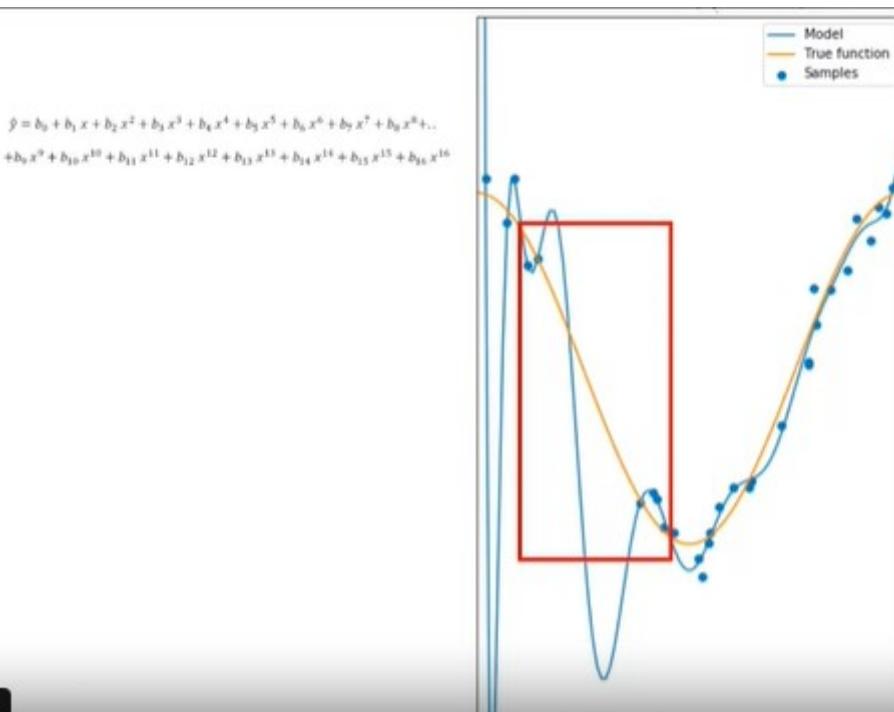


Figura. En este caso el polinomio de orden 16 produce overfitting.

Veamos una trama del error cuadrado medio para el conjunto de entrenamiento y pruebas de polinomios de orden diferente.

El eje horizontal representa el orden del polinomio.

El eje vertical es el error cuadrado medio.

El error de entrenamiento disminuye con el orden del polinomio.

El error de prueba es un mejor medio de estimar el error de un polinomio. El error disminuye hasta que se determina el mejor orden del polinomio . Entonces el error comienza a aumentar. Seleccionamos el orden que minimiza el error de prueba. En este caso, 8. Cualquier cosa de la izquierda se consideraría underfitting. Cualquier cosa de la derecha se considera overfitting.



Figura. Seleccionamos el mejor orden del polinomio, que en este caso es 8. A la izquierda se tiene underfitting y a la derecha overfitting.

Si seleccionamos el mejor orden del polinomio, tendremos algunos errores. Si recuerda la expresión original de los puntos de entrenamiento vemos un término de ruido. Este término es una de las razones del error. Esto se debe a que el ruido es aleatorio, y no podemos predecir. Esto se denomina a veces un error irreductible.

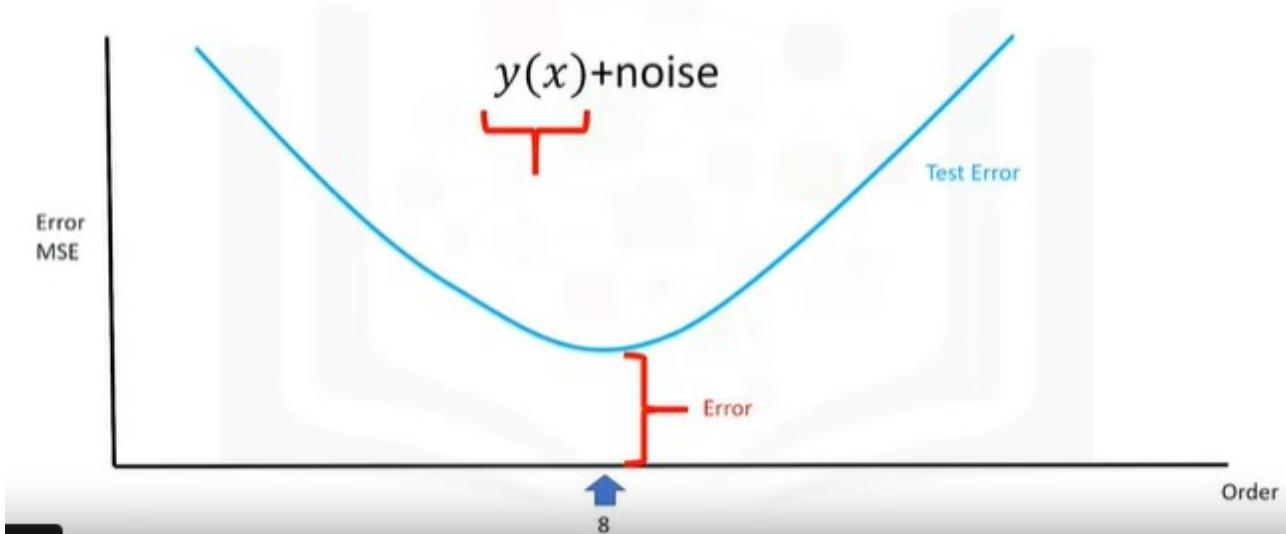


Figura. Tendremos errores aún seleccionando el mejor orden del polinomio. También hay otras fuentes de errores. Por ejemplo, nuestra suposición polinómica puede ser incorrecta. Nuestros puntos de muestra pueden haber venido de una función diferente. Por

ejemplo, en este diagrama, los datos se generan a partir de una onda sinusoidal. La función polinómica no hace un buen trabajo en el ajuste de la onda sinusoidal.

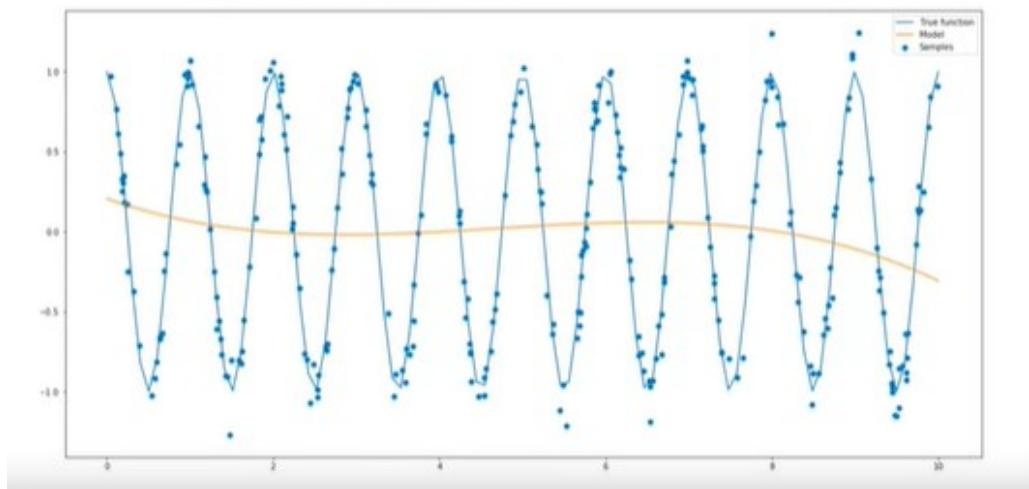


Figura. Caso donde asumir un modelo polinómico no es apropiado.

Para datos reales, el modelo puede ser demasiado difícil de ajustar o puede que no tengamos el tipo correcto de datos para estimar la función. Probemos diferentes polinomios de orden en los datos reales usando caballos de fuerza (horsepower). Los puntos rojos representan los datos de entrenamiento. Los puntos verdes representan los datos de prueba.

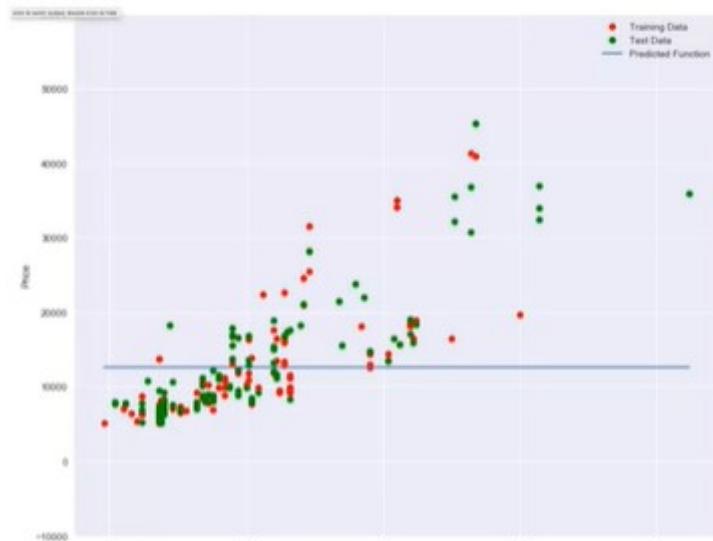


Figura. Datos reales.

Si solo usamos la media de los datos, nuestro modelo no funciona bien. Una función lineal se ajusta mejor a los datos. Un modelo de segundo orden tiene un aspecto similar a la función lineal. Una función de tercer orden también parece aumentar, como los dos pedidos anteriores. Aquí, vemos un polinomio de cuarto orden. Con alrededor de 200 caballos de fuerza, el precio previsto disminuye repentinamente. Esto parece erróneo.

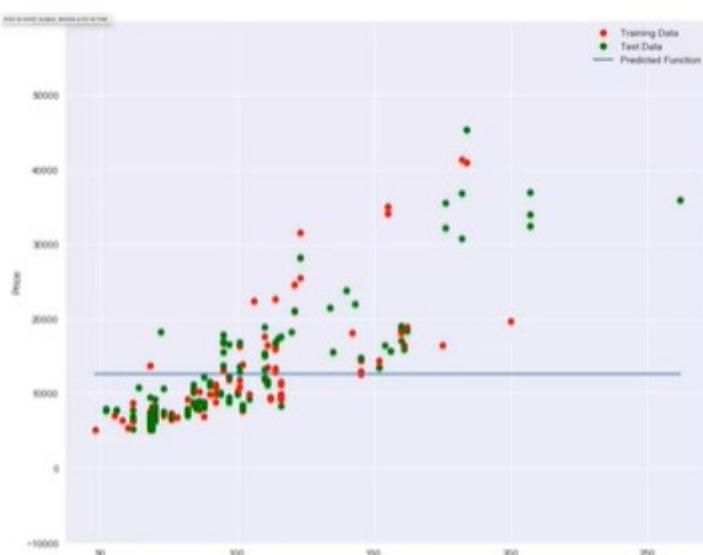


Figura. Utilizando sólo la media el modelo no funciona bien.

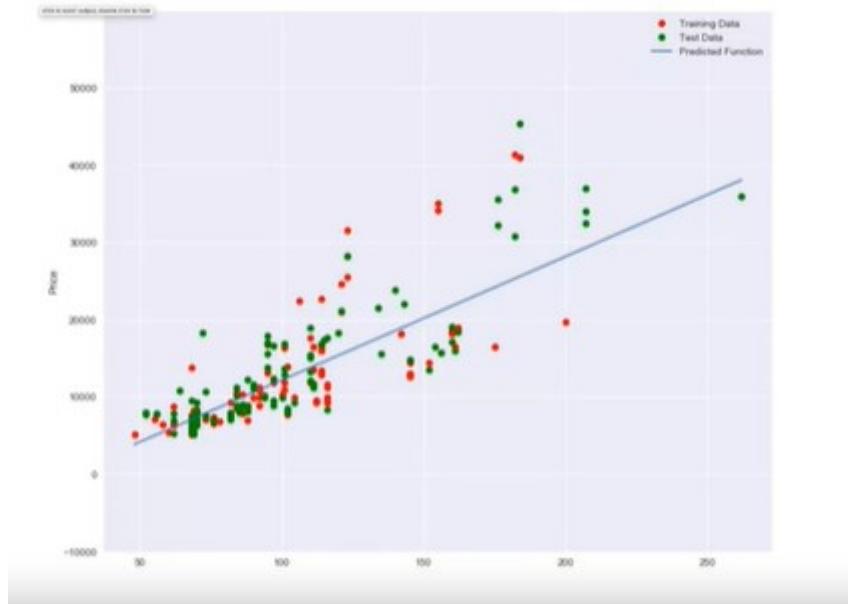


Figura. Una recta funciona mejor.

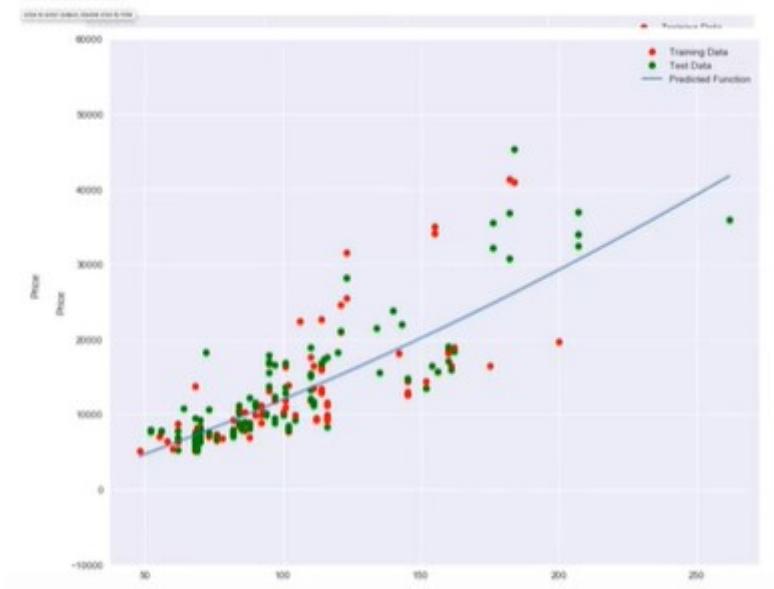


Figura. Un modelo de segundo orden es similar al de primer orden.

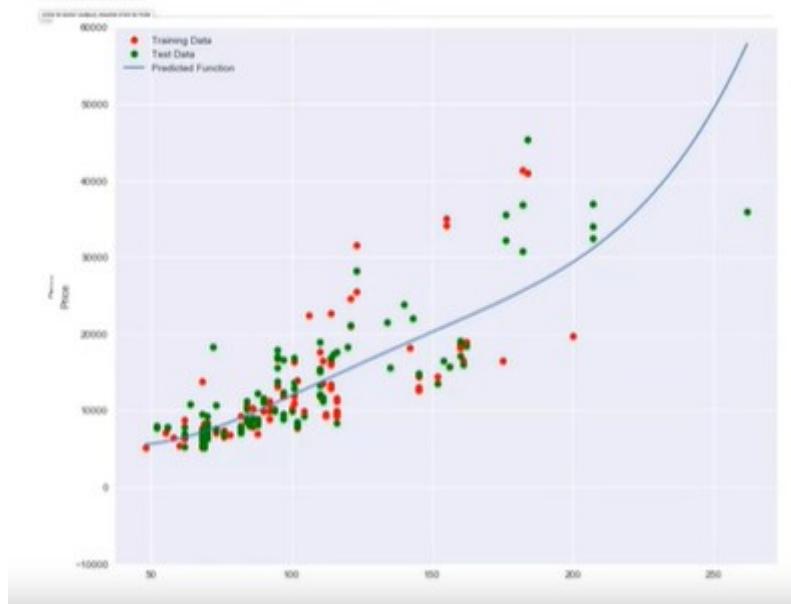


Figura. Modelo de tercer orden.

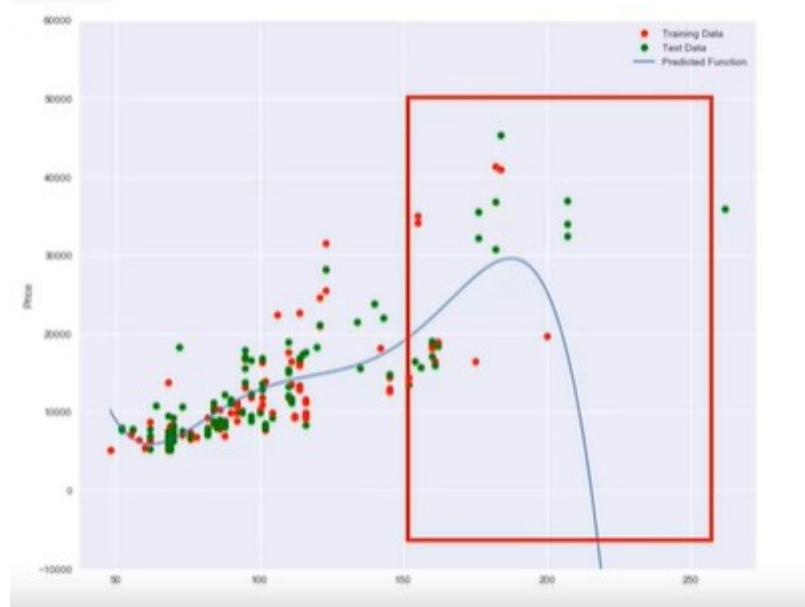


Figura. Un modelo de cuarto orden exhibe un comportamiento aparentemente erróneo.

Usemos R-cuadrado para ver si nuestra suposición es correcta. El siguiente es un diagrama del valor R cuadrado. El eje horizontal representa el orden de los modelos polinomios. Cuanto más cerca está el R cuadrado de uno, más preciso es el modelo. Aquí, vemos que el R cuadrado es óptimo cuando el orden del polinomio es tres. El R cuadrado disminuye drásticamente cuando el pedido se incrementa a cuatro, validando nuestra suposición inicial.



Figura. Diagrama del valor de R^2 . Vemos que el óptimo cuando el orden del polinomio es 3.

Podemos calcular diferentes valores R cuadrados de la siguiente manera.

- Creamos una lista vacía para almacenar los valores.
- Creamos una lista que contiene diferentes órdenes de polinomios.
- Iteramos a través de la lista usando un bucle.
- Creamos un objeto de entidad polinómica con el orden del polinomio como parámetro.
- Transformamos los datos de entrenamiento y pruebas en un polinomio utilizando el método de transformación de ajuste.
- Ajustamos el modelo de regresión usando los datos de transformación.
- Calculamos el R cuadrado usando los datos de prueba y lo almacenamos en la matriz.

```
Rsqu_test=[]  
order=[1,2,3,4]  
for n in order:  
  
    pr=PolynomialFeatures(degree=n)  
  
    x_train_pr=pr.fit_transform(x_train[['horsepower']])  
  
    x_test_pr=pr.fit_transform(x_test[['horsepower']])  
  
    lr.fit(x_train_pr,y_train)  
  
    Rsqu_test.append(lr.score(x_test_pr,y_test))
```

Figura. Calculando diferentes valores de R^2 .

5.3. RIDGE REGRESSION

La regresión Ridge evita el exceso de ajuste. Nos centraremos en la regresión polinómica para la visualización, pero el sobreajuste también es un gran problema cuando usted tiene múltiples variables independientes, o características.

Consideré el siguiente polinomio de cuarto orden en naranja. Los puntos azules se generan a partir de esta función. Podemos usar un polinomio de décimo orden para ajustar los datos. La función estimada en azul hace un buen trabajo aproximando la función verdadera.

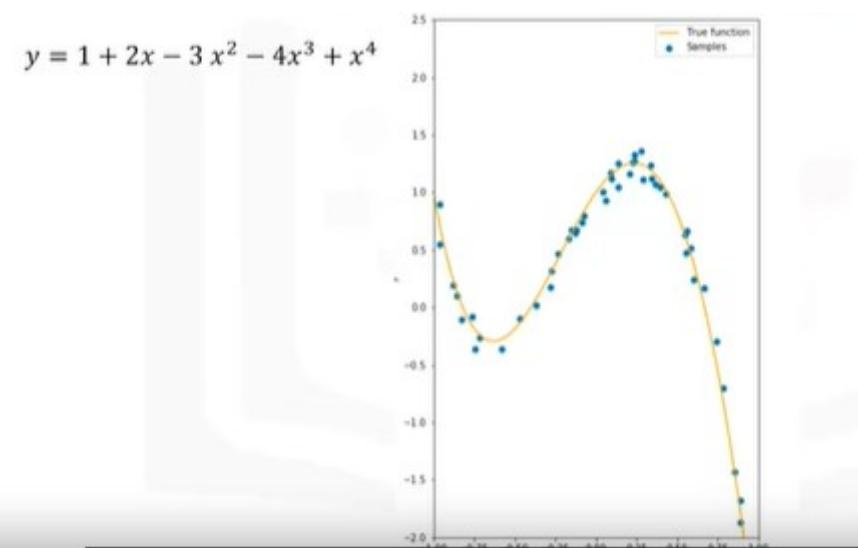


Figura. Ajustes de polinomios.

En muchos casos, los datos reales tienen valores atípicos. Si usamos una función polinómica de décimo orden para ajustar los datos, la función estimada en azul es incorrecta, y no es una buena estimación de la función real en naranja.

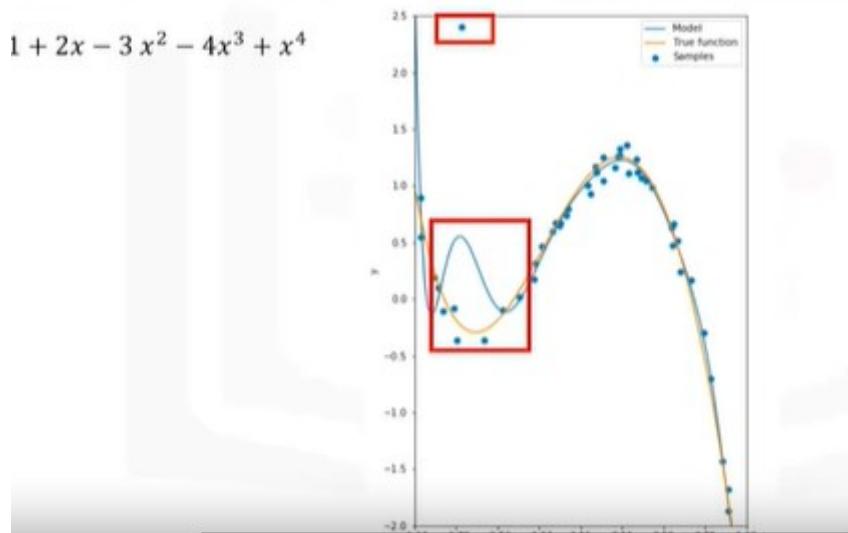


Figura. En muchos casos se tienen valores atípicos (punto solitario de arriba). Aquí la función azul no ajusta bien.

Si examinamos la expresión para la función estimada, vemos que los coeficientes polinomios estimados tienen una magnitud muy grande. Esto es especialmente evidente para los polinomios de orden superior.

$$\hat{y} = 1 + 2x - 3x^2 - 2x^3 - 12x^4 - 40x^5 + 80x^6 + 71x^7 - 141x^8 - 38x^9 + 75x^{10}$$

Figura. Los coeficientes del polinomio son grandes.

La regresión Ridge controla la magnitud de estos coeficientes polinomios mediante la introducción del parámetro alfa.

Alpha es un parámetro que seleccionamos antes de instalar o entrenar el modelo.

Cada fila de la tabla siguiente (primera tabla a la izquierda) representa un valor creciente de alfa.

Veamos cómo los diferentes valores de alfa cambian el modelo.

La segunda tabla representa los coeficientes polinomiales para los diferentes valores de alfa.

La columna corresponde a los diferentes coeficientes de los polinomios, y las filas corresponden a los diferentes valores de alfa.

A medida que el alfa aumenta los parámetros se hacen más pequeños. Esto es más evidente para las características polinómicas de orden superior.

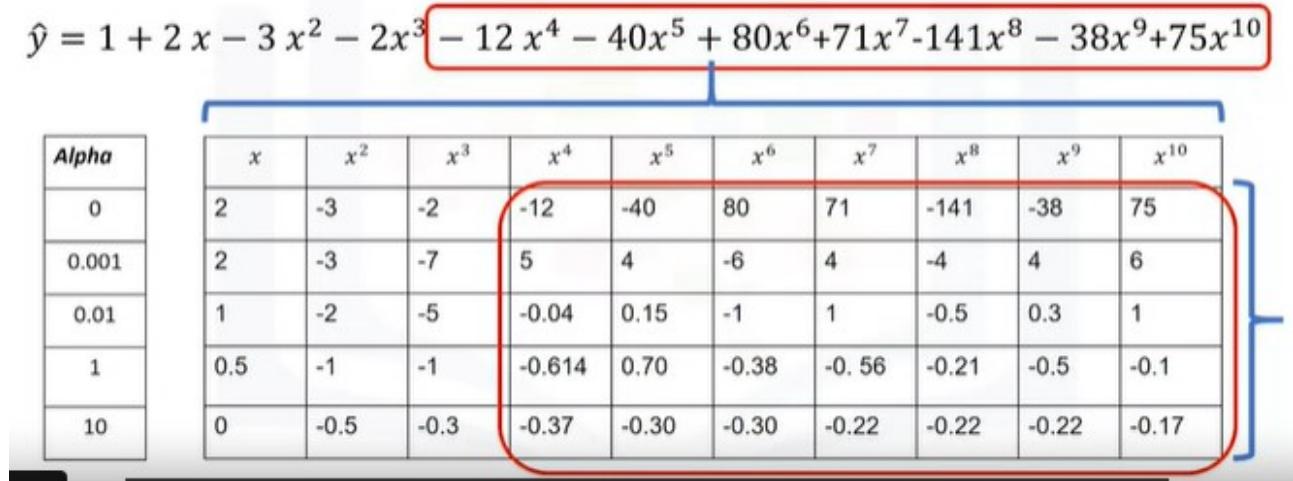


Figura. La regresión Ridge controla los coeficientes del polinomio mediante la introducción del parámetro alfa.

Alfa debe ser seleccionado cuidadosamente.

Si alfa es demasiado grande, los coeficientes se acercarán a cero y se ajustarán a los datos.

En este ejemplo:

- Si alfa es cero, el exceso de ajuste es evidente.
- Para alfa igual a 0.001, el sobreajuste comienza a disminuir.
- Para Alfa igual a 0.01, la función estimada rastrea la función real.
- Cuando alfa es igual a uno, vemos los primeros signos de falta de ajuste. La función estimada no tiene suficiente flexibilidad.
- En alfa es igual a 10, vemos un insuficiente ajuste extremo. Ni siquiera realiza un seguimiento de los dos puntos.

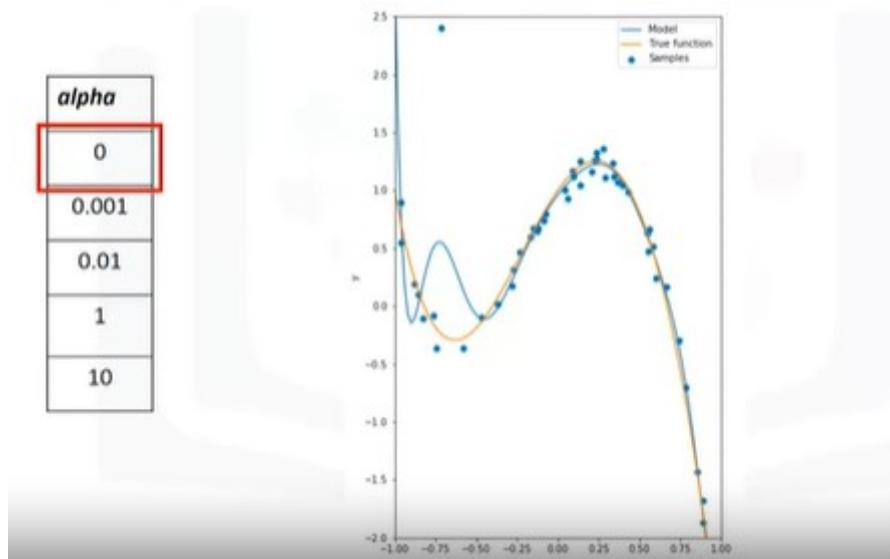


Figura. Alfa = 0. Tenemos overfitting.

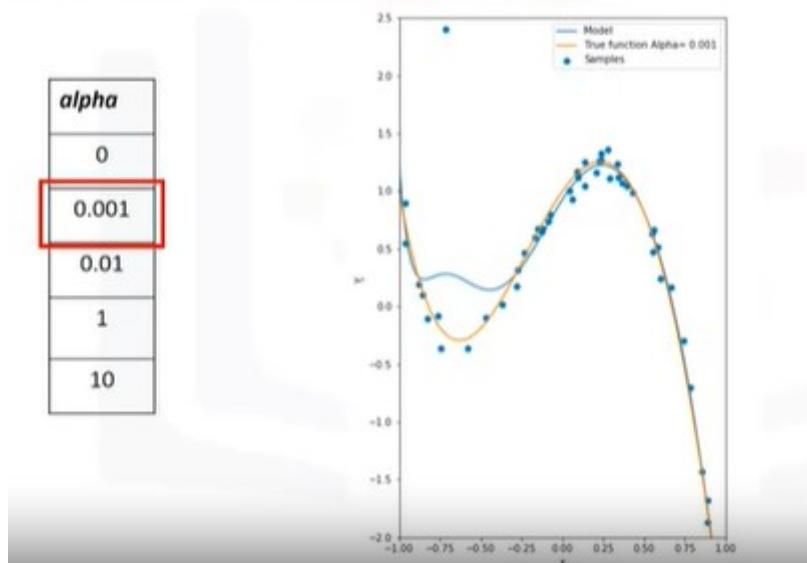


Figura. Alfa = 0.001. El overfitting disminuye.

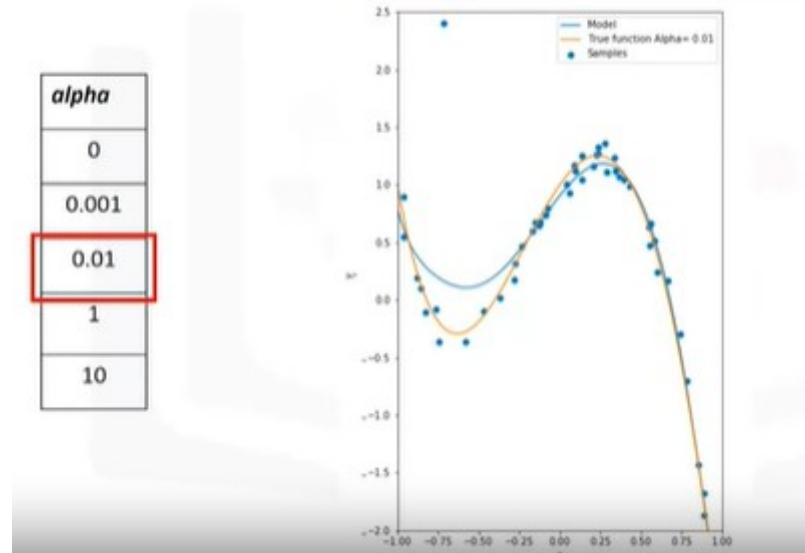


Figura. Alfa = 0.01. Buena aproximación.

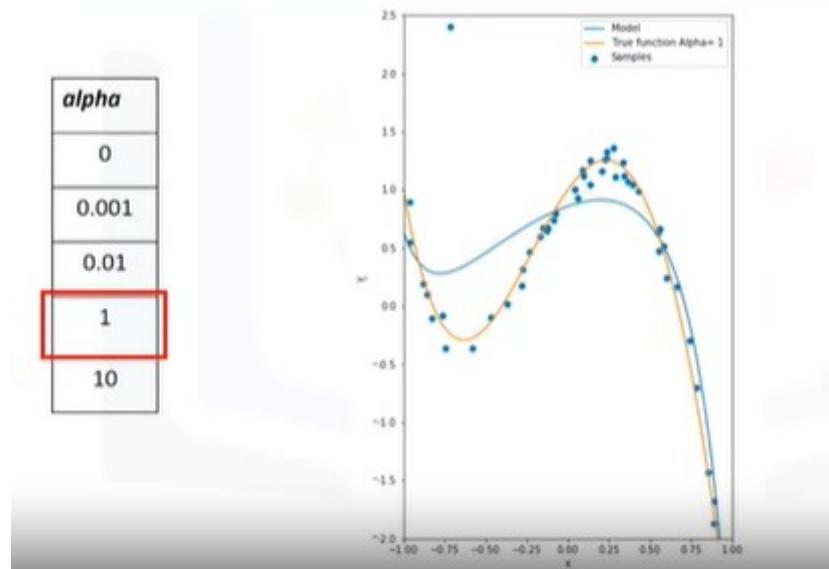


Figura. Alfa = 0.1. Primeras señales de underfitting.

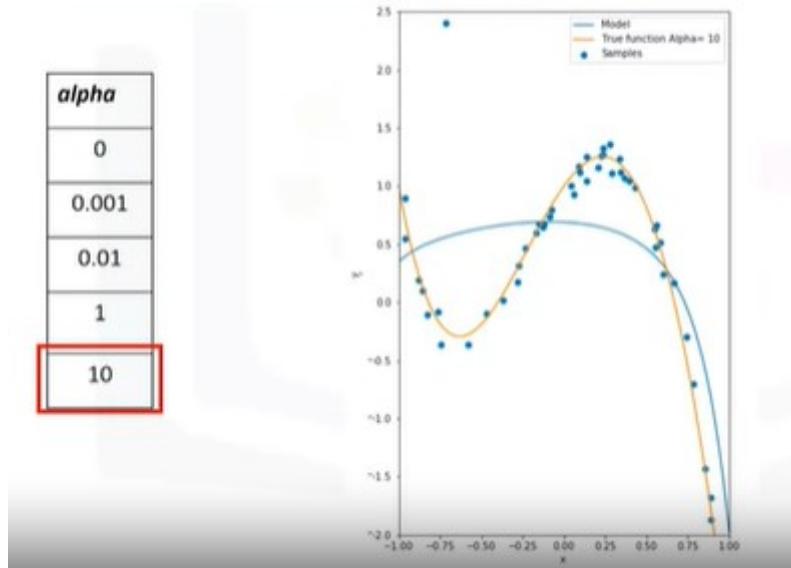


Figura. Alfa = 10. Demasiado underfitting.

Para seleccionar alfa, usamos validación cruzada.

Para hacer una predicción usando regresión de Ridge, usamos `sklearn.linear_models`.

- Creamos un objeto de Ridge usando el constructor.
 - El parámetro alpha es uno de los argumentos del constructor.
- Entrenamos el modelo usando el método de ajuste.
- Para hacer una predicción, utilizamos el método de predicción.

```
from sklearn.linear_model import Ridge
RidgeModel=Ridge(alpha=0.1)
RidgeModel.fit(X,y)
Yhat=RidgeModel.predict(X)
```

Figura. Predicción utilizando regresión de Ridge.

Para determinar el parámetro alfa:

- Utilizamos algunos datos para el entrenamiento.
- Utilizamos un segundo conjunto llamado datos de validación.

- Esto es similar a los datos de prueba, pero se utiliza para seleccionar parámetros como alfa.
- Comenzamos con un pequeño valor de alfa.
- Entrenamos el modelo, hacemos una predicción usando los datos de validación, luego calculamos el R cuadrado y almacenamos los valores.
- Repetimos para otros valores de alfa.
- Seleccionamos el valor de alfa que maximiza el R cuadrado.

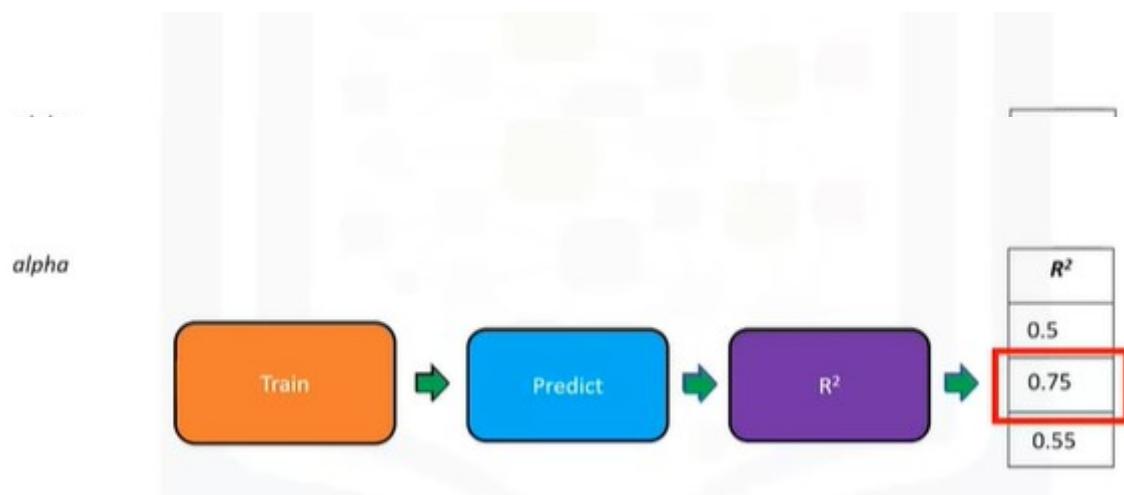


Figura. Selección de alfa.

Tenga en cuenta que podemos usar otras métricas para seleccionar el valor de alfa, como error cuadrado medio.

El problema de sobreajuste es aún peor si tenemos muchas características.

En el siguiente diagrama:

- Muestra los diferentes valores de R cuadrado en el eje vertical.
- El eje horizontal representa valores diferentes para alfa.

Utilizamos varias características de nuestro conjunto de datos del coche usado y una función polinómica de segundo orden.

Los datos de entrenamiento están en rojo y los datos de validación están en azul.

Vemos como el valor de alfa aumenta, el valor de R cuadrado aumenta y converge en aproximadamente 0,75. En este caso, seleccionamos el valor máximo de alfa porque ejecutar el experimento para valores más altos de alfa tiene poco impacto.

Por el contrario, a medida que el alfa aumenta, el R cuadrado en los datos de prueba disminuye. Esto se debe a que el término alfa evita el exceso de ajuste. Esto puede mejorar

los resultados en los datos no vistos, pero el modelo tiene peor rendimiento en los datos de prueba.

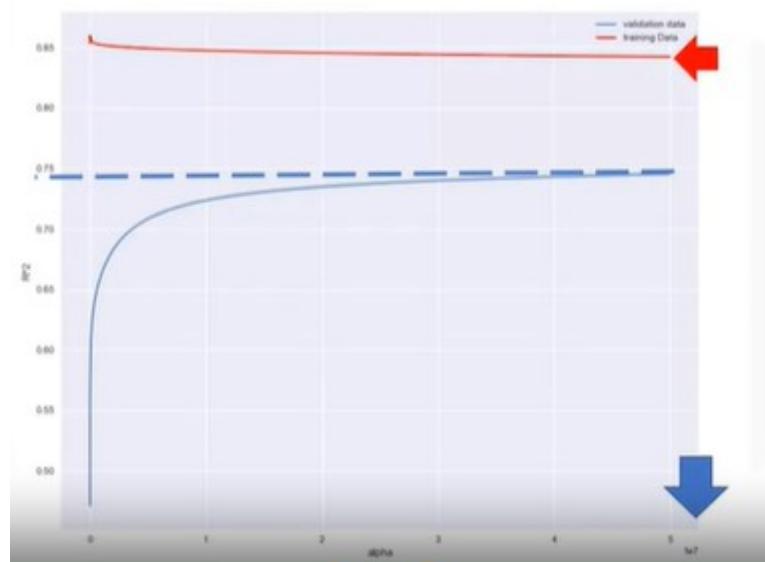


Figura. El problema del sobreajuste es peor si se tienen muchas características.

5.4. GRID SEARCH

Grid Search nos permite escanear a través de múltiples parámetros libres con pocas líneas de código.

Parámetros como el término alfa discutido en la sección anterior no son parte del proceso de ajuste o entrenamiento. Estos valores se denominan **hiperparámetros**.

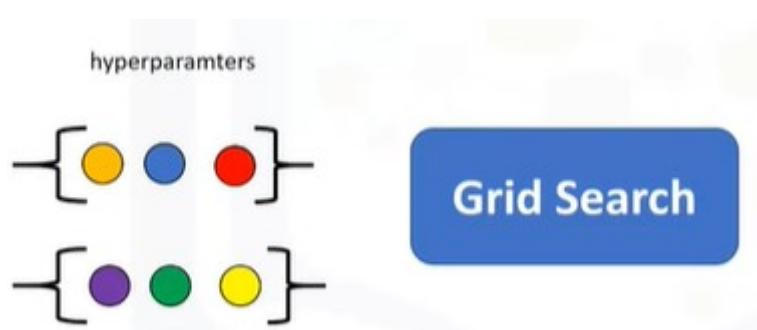


Figura. Hiperparámetros.

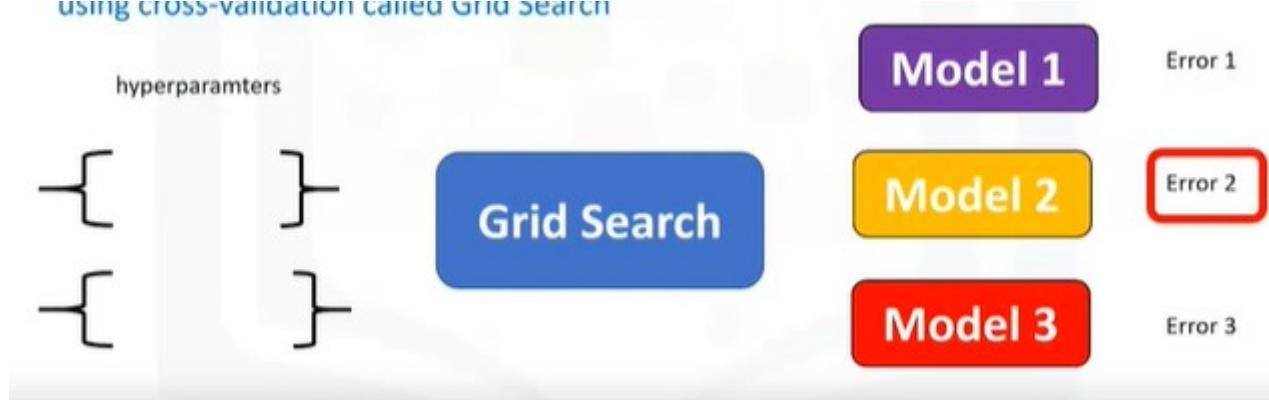
SciKit-Learn tiene un medio de iterar automáticamente sobre estos hiperparámetros usando validación cruzada. Este método se llama Búsqueda de cuadrícula (Grid Search). **La búsqueda de cuadrícula:**

- Toma el modelo u objetos que desea entrenar y diferentes valores de los hiperparámetros.
- Luego calcula el error cuadrado medio o R cuadrado para varios valores de hiperparámetro, lo que le permite elegir los mejores valores.
- En la figura siguiente los círculos pequeños representan diferentes hiperparámetros. Comenzamos con un valor para hiperparámetros y entrenamos el modelo. Utilizamos diferentes hiperparámetros para entrenar el modelo.
- Continuamos el proceso hasta que hayamos agotado los diferentes valores de parámetros libres.
- Cada modelo produce un error.
- Seleccionamos el hiperparámetro que minimiza el error.

hyperparamters

Model 1

Grid Search

**Figura.** Grid search.**Para seleccionar el hiperparámetro:**

- Dividimos nuestro conjunto de datos en tres partes:
 - El conjunto de entrenamiento.
 - El conjunto de validación.
 - El conjunto de pruebas.
- Entrenamos el modelo para diferentes hiperparámetros.
- Utilizamos el error R cuadrado o cuadrado medio para cada modelo.
- **Seleccionamos el hiperparámetro que minimiza el error cuadrado medio o maximiza el R cuadrado en el conjunto de validación.**

Finalmente probamos el rendimiento de nuestro modelo usando los datos de prueba.

En la página web de scikit-learn, se dan los parámetros del constructor del objeto. Cabe señalar que los atributos de un objeto también se llaman parámetros. Noaremos la distinción a pesar de que algunas de las opciones no son hiperparámetros per se.

En este módulo, nos centraremos en el hiperparámetro alfa y el parámetro de normalización.

Figura. Página web de Scikit-Learn

El valor de su búsqueda de cuadrícula es una lista de Python que contiene un diccionario de Python. La clave es el nombre del parámetro libre. El valor del diccionario es los diferentes valores del parámetro libre. Esto se puede ver como una tabla con varios valores de parámetros libres. También tenemos el objeto o modelo.

parameters = [{ 'alpha' : [1, 10, 100, 1000] }]

Alpha	1	10	100	1000
-------	---	----	-----	------

Ridge()

Figura. Parámetros y objetos.

La Búsqueda de Cuadrícula adopta el método de puntuación. En este caso, R cuadrado el número de pliegues, el modelo u objeto, y los valores del parámetro libre. Algunas de las salidas incluyen las diferentes puntuaciones para diferentes valores de parámetros libres.

En este caso, el R cuadrado junto con un parámetro libre valores que tienen la mejor puntuación.

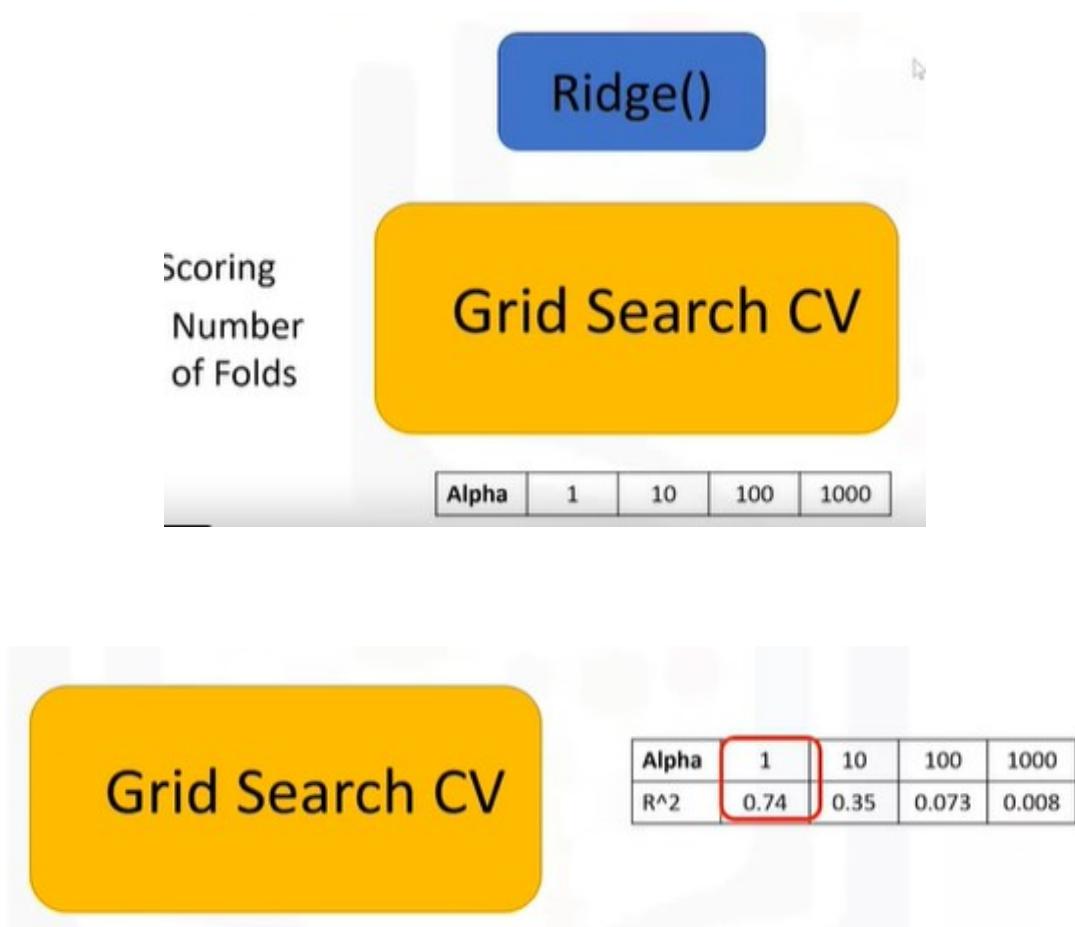


Figura. Entradas y salidas en Grid Search.

Para la implementación:

- Importamos las bibliotecas que necesitamos, incluyendo GridSearchCV, el diccionario de valores de parámetros.
- Creamos un objeto o modelo de regresión de cresta.
- A continuación, creamos un objeto GridSearchCV.
 - Las entradas son el objeto de regresión de cresta, los valores de parámetro y el número de pliegues. Vamos a utilizar R-cuadrado, que es el método de puntuación predeterminado.
 - Encajamos (fit) el objeto.

- Podemos encontrar los mejores valores para los parámetros libres usando el atributo best estimator.
- También podemos obtener información como la puntuación media en los datos de validación utilizando el resultado CV del atributo.

Una de las ventajas de la Grid Search es la rapidez con la que podemos probar múltiples parámetros.

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

parameters1= [{"alpha": [0.001,0.1,1, 10, 100, 1000,10000,100000,1000000]}]

RR=Ridge()

Grid1 = GridSearchCV(RR, parameters1, cv=4)

Grid1.fit(x_data[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']], y_data)

Grid1.best_estimator_

scores = Grid1.cv_results_
scores['mean_test_score']

array([ 0.66549413, 0.66554568, 0.66602936, 0.66896822, 0.67334636, 0.65781884, 0.65781884])
```

Figura. Implementación de Grid Search.

Por ejemplo, la regresión de cresta tiene la opción de normalizar los datos. El término alfa es el primer elemento del diccionario. El segundo elemento es la opción normalizada. La clave es el nombre del parámetro. El valor son las diferentes opciones en este caso porque podemos normalizar los datos o no. Los valores son True o False respectivamente. El diccionario es una tabla o cuadrícula que contiene dos valores diferentes. Como antes, necesitamos el objeto o modelo de regresión de cresta.

```
parameters = [{ 'alpha': [1, 10, 100, 1000], 'normalize': [True, False]}]
```

Alpha	1	10	100	1000
Normalize	True	True	True	True
	False	False	False	False

Ridge()

Figura. La regresión de Ridge tiene la opción de normalizar.

El procedimiento es similar excepto que tenemos una tabla o cuadrícula de diferentes valores de parámetros. La salida es la puntuación de todas las diferentes combinaciones de valores de parámetros.

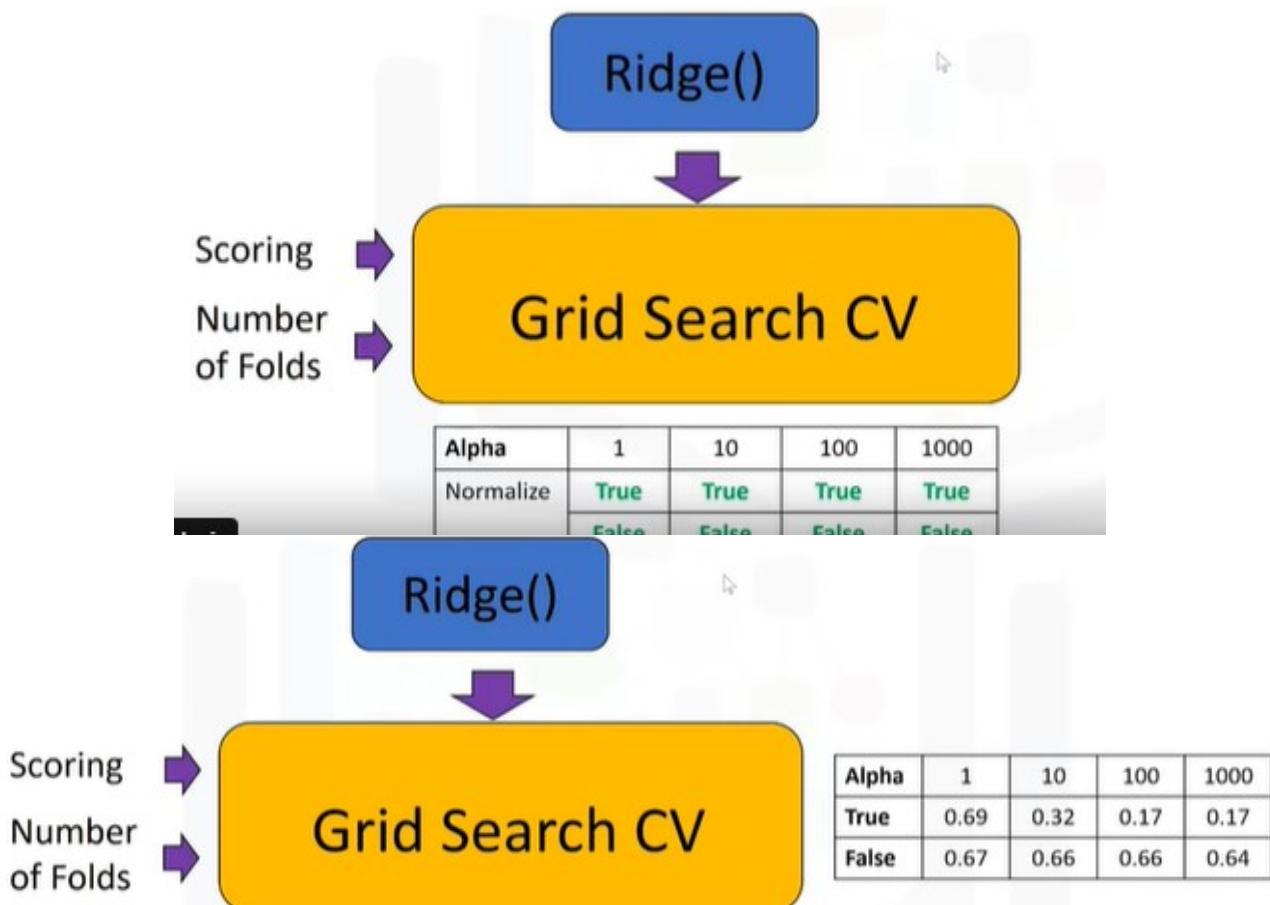


Figura. Procedimiento pudiendo normalizar.

El código también es similar. El diccionario contiene los diferentes valores de parámetros libres. Podemos encontrar el mejor valor para los parámetros gratuitos. Las puntuaciones resultantes de los diferentes parámetros libres se almacenan en este diccionario, Grid1.cv_Results _.

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

parameters2= [{"alpha': [0.001,0.1,1, 10, 100], 'normalize': [True, False] }]

RR=Ridge()

Grid1 = GridSearchCV(RR, parameters2, cv=4)

Grid1.fit(x_data[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']],y_data)

Grid1.best_estimator_

scores = Grid1.cv_results_
```

Figura. Código en el caso donde es posible normalizar.

Podemos imprimir la puntuación para los diferentes valores de parámetros libres. Los valores de los parámetros se almacenan como se muestra aquí.

```
for param, mean_val, mean_test in zip(scores['params'], scores['mean_test_score'], scores['mean_train_score']):
    print(param, "R^2 on test data:", mean_val, "R^2 on train data:", mean_test)
```

```
{"alpha": 0.001, "normalize": True} R^2 on tesst data: 0.66605547293 R^2 on train data: 0.814001968709
{"alpha": 0.001, "normalize": False} R^2 on tesst data: 0.665488366584 R^2 on train data: 0.814002698797
{"alpha": 0.1, "normalize": True} R^2 on tesst data: 0.694175625356 R^2 on train data: 0.810546768311
{"alpha": 0.1, "normalize": False} R^2 on tesst data: 0.66548937796 R^2 on train data: 0.814002698794
{"alpha": 1, "normalize": True} R^2 on tesst data: 0.690486934584 R^2 on train data: 0.749104440368
{"alpha": 1, "normalize": False} R^2 on tesst data: 0.665494127178 R^2 on train data: 0.814002698472
{"alpha": 10, "normalize": True} R^2 on tesst data: 0.321376875232 R^2 on train data: 0.341856042902
{"alpha": 10, "normalize": False} R^2 on tesst data: 0.665545680812 R^2 on train data: 0.8140026666
{"alpha": 100, "normalize": True} R^2 on tesst data: 0.0170551710263 R^2 on train data: 0.0496044796826
{"alpha": 100, "normalize": False} R^2 on tesst data: 0.666029359996 R^2 on train data: 0.813999791851
{"alpha": 1000, "normalize": True} R^2 on tesst data: -0.0301961745066 R^2 on train data: 0.005184451599
{"alpha": 1000, "normalize": False} R^2 on tesst data: 0.668968215369 R^2 on train data: 0.813870488264
{"alpha": 10000, "normalize": True} R^2 on tesst data: -0.0351687400461 R^2 on train data: 0.000520784757979
{"alpha": 10000, "normalize": False} R^2 on tesst data: 0.673346359342 R^2 on train data: 0.812583743226
{"alpha": 100000, "normalize": True} R^2 on tesst data: -0.0356685844558 R^2 on train data: 5.2101975528e-05
{"alpha": 100000, "normalize": False} R^2 on tesst data: 0.657818838432 R^2 on train data: 0.789541446486
{"alpha": 100000, "normalize": True} R^2 on tesst data: -0.0356685844558 R^2 on train data: 5.2101975528e-05
{"alpha": 100000, "normalize": False} R^2 on tesst data: 0.657818838432 R^2 on train data: 0.789541446486
```

Figura. Imprimiendo valores de los parámetros libres.