

Fastbook 09

Programación en Python

Análisis exploratorio de datos



09. Análisis exploratorio de datos

Durante los últimos cuatro fastbooks hemos aprendido a manejarnos con soltura en Pandas, especialmente con los dataframes o tablas. En este fastbook veremos cómo **aplicar todos estos conocimientos en un proyecto de analítica de datos**.

Para ejemplificar este proceso, continuaremos con el contexto de nuestro supermercado, y trataremos de explicar las unidades vendidas por un producto a lo largo de los últimos dos años. Para ello, contaremos con el histórico del precio del producto, y las campañas publicitarias que se realizaron sobre él, así como con las ofertas que puso el supermercado. ¡Vamos a ello!

Autor: Jesús Aguirre Pemán

- Análisis exploratorio de una base de datos
- Análisis gráfico de variables
- Outliers
- Relaciones entre variables
- Regresiones lineales
- Conclusiones

Lesson 1 of 6

Análisis exploratorio de una base de datos

X Edix Educación

En este primer apartado nos situaremos en el comienzo de un proyecto de analítica de datos. Antes de comenzar con los análisis, deberemos **leer los datos y realizar una serie de comprobaciones** sobre ellos.

Lectura de datos

Lo primero es leer los datos y comprobar que su carga ha sido correcta. Para ello, ya vimos en el fastbook 05 que disponemos de las funciones de la familia `pd.read_*`.

Para hacer este proceso más ágil, hemos condensado toda la información en **un único fichero**, donde aparece el histórico con una granularidad semanal.

Recordad que si tenéis varias tablas de datos, podéis juntarlas con el método `merge` de Pandas, y si necesitáis cambiar la granularidad de la serie, basta con repasar los conceptos que vimos en el fastbook anterior.

```
In [2]: 1 dataset_fastbook_9 = pd.read_excel("../data/dataset_fastbook_9.xlsx")
```

Una vez hemos leído los datos, deberemos comprobar que el formato es el correcto. Algunos de los checks básicos son:

- Los nombres de las columnas del dataframe son correctos.
- Las series que forman el dataframe aparecen con el tipo esperado (los números son de tipo `int` o `float`, o las fechas de tipo `datetime`).

Podemos utilizar el método `head` para echar un primer vistazo al dataframe.

```
In [3]: 1 dataset_fastbook_9.head()
```

Out[3]:

	semana	unidades_vendidas	precio	publi1	publi2	publi3	oferta1	oferta2	oferta3
0	2018-12-31	12340.0	1.5	0	0	0	0.0	0.0	0.0
1	2019-01-07	12242.0	1.5	0	0	0	0.0	0.0	0.0
2	2019-01-14	12430.0	1.5	0	0	0	0.0	0.0	0.0
3	2019-01-21	12327.0	1.5	0	0	0	0.0	0.0	0.0
4	2019-01-28	12289.0	1.5	0	0	0	0.0	0.0	0.0

En nuestro ejemplo, el dataframe es una serie temporal, con granularidad semanal marcada por la columna semana. Además, tenemos el dato de unidades vendidas, el precio al que se vendieron, tres campañas publicitarias y tres ofertas realizadas sobre el producto. En este fastbook trabajaremos con las publicidades y ofertas separadas, pero habrá ocasiones donde se considere que las variables deban ser conjuntas, y tengamos que continuar con la fase de transformación de datos.

Con el atributo **columns** veremos los nombres de las columnas, y con **dtypes**, los tipos de cada una de ellas.

```
In [4]: 1 dataset_fastbook_9.columns
Out[4]: Index(['semana', 'unidades_vendidas', 'precio', 'publi1', 'publi2', 'publi3',
               'oferta1', 'oferta2', 'oferta3'],
              dtype='object')

In [5]: 1 dataset_fastbook_9.dtypes
Out[5]: semana          datetime64[ns]
unidades_vendidas    float64
precio                float64
publi1                int64
publi2                int64
publi3                int64
oferta1               float64
oferta2               float64
oferta3               float64
dtype: object
```

Vemos que tanto los nombres como los tipos de las columnas parecen correctos.

Otro aspecto importante al tratarse de una serie temporal es conocer el periodo que abarca. Para ello, aplicaremos el método **describe** sobre la columna que contiene la fecha.

Vemos que los datos abarcan dos años: 2019 y 2020. Además, podemos ver que cada semana contiene un único dato. Una vez que hemos importado y validado los datos, pasemos a analizar su contenido.

```
In [6]: 1 dataset_fastbook_9.semana.describe()
Out[6]: count           105
unique          105
top      2020-08-17 00:00:00
freq                 1
first     2018-12-31 00:00:00
last      2020-12-28 00:00:00
Name: semana, dtype: object
```

Missing values

Tras comprobar que la lectura es correcta, deberemos analizar el número de *missing values* o valores perdidos en nuestros dataframes.

Para ello, podemos utilizar el método *isna*, y realizar una suma por columnas. Finalmente, filtramos los valores mayores a 0 para quedarnos con las columnas que contienen *missing values*.

```
In [7]: 1 dataset_fastbook_9.isna().sum().loc[lambda x: x>0]
```

```
Out[7]: unidades_vendidas    1
         precio            3
         oferta1           2
         oferta2           2
         oferta3           2
         dtype: int64
```

Otra opción sencilla para ver el número de valores válidos es utilizar el método *info* de los dataframes.

```
In [8]: 1 dataset_fastbook_9.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 105 entries, 0 to 104
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   semana          105 non-null    datetime64[ns]
 1   unidades_vendidas 104 non-null   float64
 2   precio           102 non-null   float64
 3   publi1           105 non-null   int64  
 4   publi2           105 non-null   int64  
 5   publi3           105 non-null   int64  
 6   oferta1          103 non-null   float64
 7   oferta2          103 non-null   float64
 8   oferta3          103 non-null   float64
dtypes: datetime64[ns](1), float64(5), int64(3)
memory usage: 7.5 KB
```

Como ya sabemos, podemos optar por llenar los valores perdidos, descartar las variables, o descartar las observaciones. La decisión dependerá de las particularidades del caso concreto que estemos tratando, pero normalmente se opta por una de las dos primeras opciones.

Veamos en profundidad estos casos de *missing values*.

Comencemos por las unidades.

```
In [9]: 1 dataset_fastbook_9.loc[lambda x: x.unidades_vendidas.isna()]
Out[9]:
      semana unidades_vendidas precio publi1 publi2 publi3 oferta1 oferta2 oferta3
11 2019-03-18           NaN     1.5     0     0     0     0.0     0.0     0.0
```

Nuestro único valor perdido ocurre en marzo de 2019. En este caso, podemos optar por interpolar el dato utilizando los de las semanas anterior y posterior, o eliminar el registro. Ya que nuestro objetivo en este fastbook es aplicar todo lo que hemos aprendido en un caso práctico, vamos a optar por la **interpolación**, aunque en un caso real esta decisión dependería del modelo que fuéramos a utilizar y de las particularidades de la variable que estuvieramos manipulando.

Tras interpolar, el valor de la semana del 18 de marzo se ha imputado como la media entre la anterior y la posterior, que además contaban con valores muy cercanos entre sí.

```
In [10]: 1 dataset_fastbook_9 = dataset_fastbook_9\
2 .assign(unidades_vendidas= lambda x:
3             x.unidades_vendidas.interpolate())
In [11]: 1 dataset_fastbook_9.loc[lambda x: (x.semana >= "2019-03-01") & \
2                   (x.semana <= "2019-04-01")]
Out[11]:
      semana unidades_vendidas precio publi1 publi2 publi3 oferta1 oferta2 oferta3
9 2019-03-04          11893.0     1.5     0     0     0     0.0     0.0     0.0
10 2019-03-11          11869.0     1.5     0     0     0     0.0     0.0     0.0
11 2019-03-18          11865.0     1.5     0     0     0     0.0     0.0     0.0
12 2019-03-25          11861.0     1.5     0     0     0     0.0     0.0     0.0
13 2019-04-01          11957.0     1.5     0     0     0     0.0     0.0     0.0
```

Veamos ahora el caso del precio.

```
In [12]: 1 dataset_fastbook_9.loc[lambda x: x.precio.isna()]
```

Out[12]:

	semana	unidades_vendidas	precio	publi1	publi2	publi3	oferta1	oferta2	oferta3
65	2020-03-30	11061.0	NaN	0	0	0	0.0	0.0	0.0
66	2020-04-06	11654.0	NaN	0	0	0	0.0	0.0	0.0
67	2020-04-13	11963.0	NaN	0	0	0	0.0	0.0	0.0

Aparecen tres valores perdidos consecutivos. Veamos los datos que rodean a este periodo.

```
In [13]: 1 dataset_fastbook_9.loc[lambda x: (x.semana >= "2020-03-01") &\n2 (x.semana <= "2020-05-01")]
```

Out[13]:

	semana	unidades_vendidas	precio	publi1	publi2	publi3	oferta1	oferta2	oferta3
61	2020-03-02	11168.0	1.5	0	0	0	0.0	0.0	0.0
62	2020-03-09	11093.0	1.5	0	0	0	0.0	0.0	0.0
63	2020-03-16	10923.0	1.5	0	0	0	0.0	0.0	0.0
64	2020-03-23	10964.0	1.5	0	0	0	0.0	0.0	0.0
65	2020-03-30	11061.0	NaN	0	0	0	0.0	0.0	0.0
66	2020-04-06	11654.0	NaN	0	0	0	0.0	0.0	0.0
67	2020-04-13	11963.0	NaN	0	0	0	0.0	0.0	0.0
68	2020-04-20	11845.0	1.5	0	0	0	0.0	0.0	0.0
69	2020-04-27	11620.0	1.5	0	0	0	0.0	0.0	0.0

No parece que haya cambios en el precio (ni en las variables de publicidad y oferta), ni variaciones significativas en las unidades vendidas. Por tanto, podemos imputar los datos usando el método `fillna`.

```
In [14]: 1 dataset_fastbook_9 = dataset_fastbook_9\  
2     .assign(precio = lambda x: x.precio.fillna(method = "ffill"))
```

Por último, tratemos las variables de oferta.

```
In [15]: 1 dataset_fastbook_9.loc[lambda x: x.oferta1.isna()]  
  
Out[15]:  
    semana unidades_vendidas precio publi1 publi2 publi3 oferta1 oferta2 oferta3  
103 2020-12-21          10360.0   1.5      0      0      0      NaN      NaN      NaN  
104 2020-12-28          10412.0   1.5      0      0      0      NaN      NaN      NaN
```

Vemos que los *missing values* se sitúan al final del histórico, por lo que, muy probablemente, se deben a que no existe dato para ese periodo. En un proyecto real, confirmaríamos el periodo disponible en los datos de ofertas, y tras ello podríamos imputarlo a cero. Para llenar las tres columnas en una sola línea, utilizaremos el condicional ternario que aprendimos en el fastbook 02 (otra opción es realizar tres asignaciones consecutivas).

```
In [16]: 1 dataset_fastbook_9 = dataset_fastbook_9\  
2     .apply(lambda x: x.fillna(0) if "oferta" in x.name else x)
```

Tras haber limpiado los *missing values*, estamos listos para realizar un primer análisis descriptivo de las variables.

Lesson 2 of 6

Análisis gráfico de variables

 Edix Educación

Una parte muy importante en los proyectos de Data Science es la representación gráfica de las variables que componen los datos. En este paso podremos ver cuál es su distribución, las particularidades que pueda tener asociada, y si existen valores extremos que debamos eliminar.

En esta asignatura no vamos a profundizar en la construcción de los gráficos, sino en su utilidad para el proceso de analítica de datos.

Utilizaremos el módulo de visualización integrado en Pandas (*pandas plot*), así como la librería *seaborn* para determinados gráficos.

Histogramas

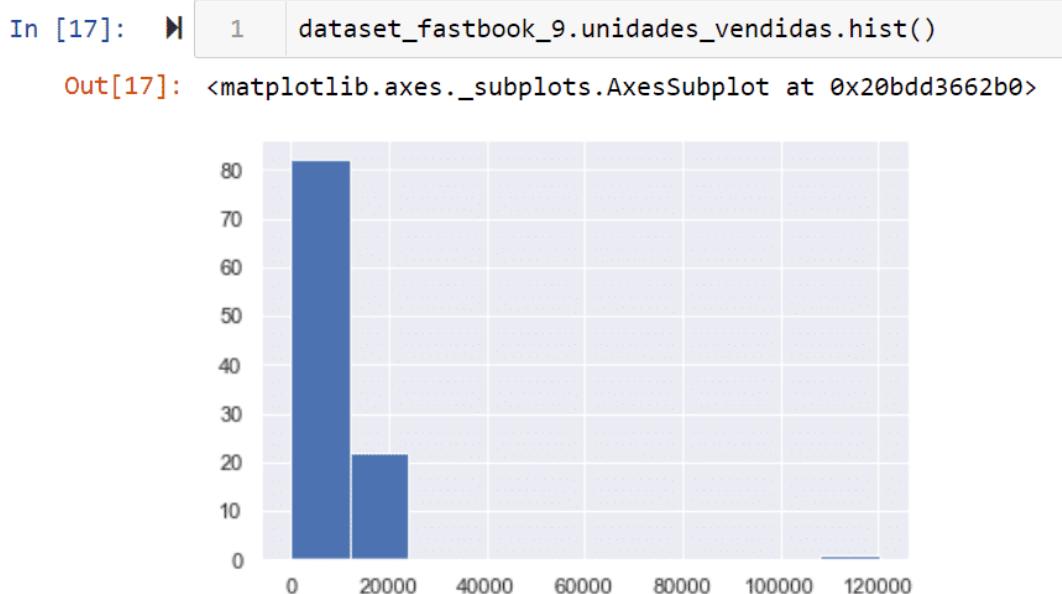
Una primera aproximación rápida para conocer el contenido de los datos son los histogramas.

Representación en forma de barras, donde se agrupan los valores de la variable por tramos, y la altura de cada barra está relacionada con el número de veces que la variable se encuentra en el tramo.

Este tipo de gráficas son especialmente útiles para variables cuantitativas continuas, como es el caso de las unidades en nuestro set de datos.

En el gráfico podemos ver que la mayoría de las observaciones caen en la primera barra, teniendo la segunda en torno a la cuarta parte y, la tercera, una sola.

Además, hay una pequeña barra alejada de las tres primeras; nos centraremos en ella en el siguiente capítulo.



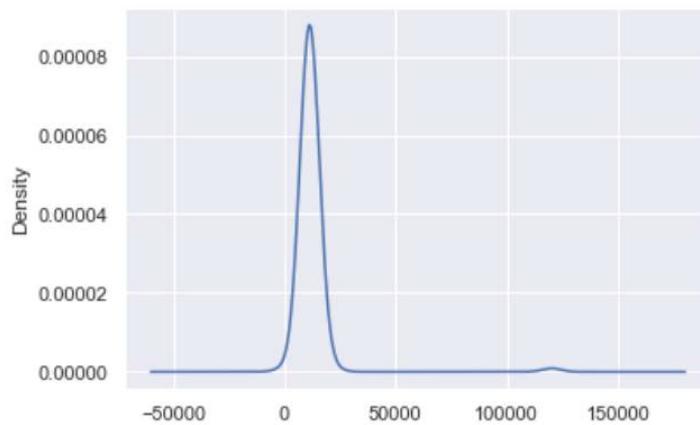
Gráficos de densidad

Otra forma de representar variables continuas es mediante gráficos de densidad.

En lugar de agrupaciones de valores como los histogramas, utilizan un suavizado estadístico que disminuye el ruido de los *outliers*.

```
In [18]: 1 dataset_fastbook_9.unidades_vendidas.plot.density()
```

```
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x20bdd3111d0>
```

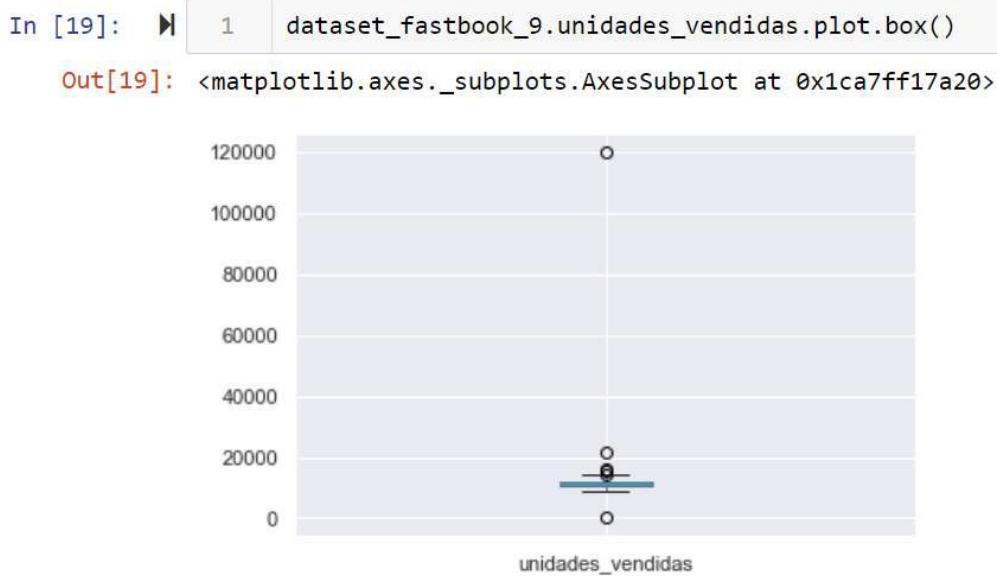


Diagramas de caja

Los diagramas de caja o *box plots* nos permiten representar gráficamente una variable centrándonos en su distribución.

En este tipo de diagramas, los cuartiles formarán una caja, donde se señalará también la mediana. De esta forma, podremos ver de forma sencilla las observaciones que se salgan de este rango, sobre las que nos centraremos en el siguiente apartado.

En nuestro ejemplo, la caja que forma nuestra variable de unidades queda empequeñecida por el valor extremo que ya habíamos visto en el histograma.



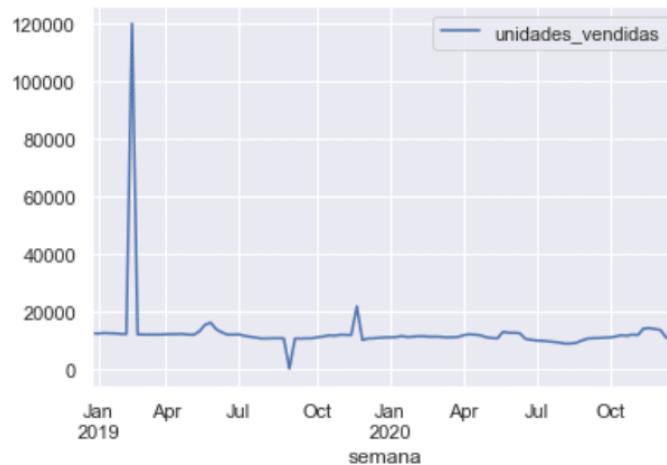
Gráficos de evolución

En el caso de series temporales, como ya veímos en el fastbook anterior, lo natural es pintar la evolución con un gráfico de líneas.

Vemos que la serie parece tener un valor base bastante constante, algunos picos arriba y abajo, y varias subidas y bajadas más graduales.

```
In [20]: 1 dataset_fastbook_9.plot(x = "semana", y = "unidades_vendidas")
```

```
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1ca7ff17630>
```



Gráficos de barras

Si tenemos variables categóricas, una forma sencilla e intuitiva de representarlas es mediante un gráfico de barras, ya sean horizontales o verticales.



En cada barra se mostrará el conteo de cada valor de la variable.

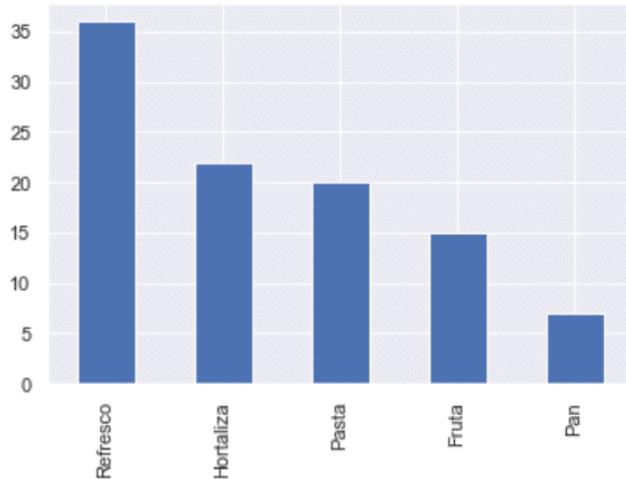


Ya que nuestro dataset no tiene ninguna variable categórica, recuperaremos el dataset de productos que utilizábamos en fastbooks anteriores.

Si calculamos el conteo de productos por tipo mediante *value_counts*, obtendremos una serie, que podemos representar visualmente mediante un gráfico de barras.

```
In [21]: 1 productos = pd.read_csv("../data/dataset_productos.csv")
2 productos.tipo.value_counts().plot.bar()
```

```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x2064ab69e80>
```



Además de estos gráficos, tenemos disponibles muchos más para explorar los datos, y profundizaréis en ellos en las asignaturas de visualización.

Outliers

X Edix Educación



Los outliers son **observaciones extremas dentro de una variable**, cuya inclusión en modelos analíticos puede distorsionarlos y alejarlos de la realidad que conforman la mayoría de las observaciones.

Los outliers pueden deberse a distintas causas, como errores en la imputación o en el procesamiento de los datos, o incluso ser datos genuinamente reales.

Además, para considerar si una observación es un outlier, podemos fijarnos en una única variable, lo que se denomina un enfoque univariante; o en más de una (multivariante).

En este fastbook nos centraremos en los outliers univariantes, ya que son los más sencillos de detectar. Para ello, podemos analizar la distribución de la variable, ya sea mediante funciones como *describe*, o de forma gráfica, como por ejemplo, mediante un histograma o un box plot.

Volviendo a nuestro dataset, ya habíamos visto una observación alejada en la variable de *unidades_vendidas* al pintar tanto el histograma como el box plot.

Si filtramos el set de datos para quedarnos con esta observación, vemos que no tiene valores en ninguna variable de publicidad u oferta.

```
In [22]: 1 dataset_fastbook_9.loc[lambda x: x.unidades_vendidas > 30000]
```

Out[22]:

	semana	unidades_vendidas	precio	publi1	publi2	publi3	oferta1	oferta2	oferta3
7	2019-02-18	120080.0	1.5	0	0	0	0.0	0.0	0.0

Veamos los datos que rodean a esta observación.

```
In [23]: 1 dataset_fastbook_9.loc[lambda x: (x.semana >= "2019-02-01") &\n2 (x.semana <= "2019-03-01")]
```

Out[23]:

	semana	unidades_vendidas	precio	publi1	publi2	publi3	oferta1	oferta2	oferta3
5	2019-02-04	12121.0	1.5	0	0	0	0.0	0.0	0.0
6	2019-02-11	12037.0	1.5	0	0	0	0.0	0.0	0.0
7	2019-02-18	120080.0	1.5	0	0	0	0.0	0.0	0.0
8	2019-02-25	11943.0	1.5	0	0	0	0.0	0.0	0.0

En las semanas próximas al outlier, el resto de las variables de la tabla permanecen constantes. Además, la fecha (semana del 18 de febrero) no parece indicar ningún festivo o estacionalidad especial, ya que, además, un año después el valor no se acerca a esa magnitud.

Para evitar que este dato nos desvirtúe un posible modelo, eliminémoslo e imputemos el valor mediante una interpolación.

```
In [24]: 1 dataset_fastbook_9_corregido = dataset_fastbook_9\
2     .assign(unidades_vendidas = lambda x:
3         np.where(x.unidades_vendidas > 3000,
4                 np.nan, x.unidades_vendidas))\
5     .assign(unidades_vendidas = lambda x:
6         x.unidades_vendidas.interpolate())
```

```
In [25]: 1 dataset_fastbook_9_corregido\
2     .loc[lambda x: (x.semana >= "2019-02-01") &
3          (x.semana <= "2019-03-01")]
```

Out[25]:

	semana	unidades_vendidas	precio	publi1	publi2	publi3	oferta1	oferta2	oferta3
5	2019-02-04	12121.0	1.5	0	0	0	0.0	0.0	0.0
6	2019-02-11	12037.0	1.5	0	0	0	0.0	0.0	0.0
7	2019-02-18	11990.0	1.5	0	0	0	0.0	0.0	0.0
8	2019-02-25	11943.0	1.5	0	0	0	0.0	0.0	0.0

Lo mismo podemos realizar con el dato que estaba próximo a 0.

```
In [26]: 1 dataset_fastbook_9_corregido\
2     .loc[lambda x: x.unidades_vendidas < 1000]
```

Out[26]:

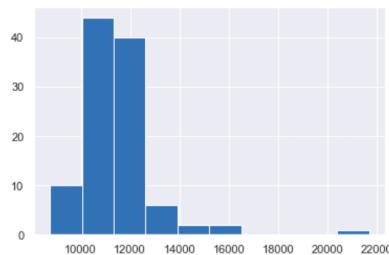
	semana	unidades_vendidas	precio	publi1	publi2	publi3	oferta1	oferta2	oferta3
35	2019-09-02	18.0	1.6	0	0	0	0.0	0.0	0.0

```
In [27]: 1 dataset_fastbook_9_corregido = dataset_fastbook_9_corregido\
2     .assign(unidades_vendidas = lambda x:
3         np.where(x.unidades_vendidas < 50,
4                 np.nan, x.unidades_vendidas))\
5     .assign(unidades_vendidas = lambda x:
6         x.unidades_vendidas.interpolate())
```

Y tras eliminar estos outliers, podemos volver a pintar el histograma o el box plot para comprobar los cambios.

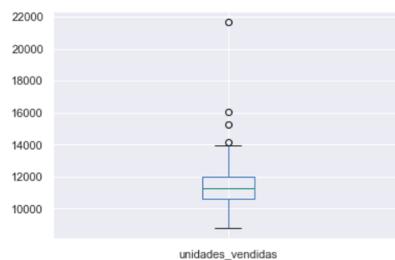
```
In [28]: 1 dataset_fastbook_9_corregido.unidades_vendidas.hist()
```

```
Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x1af901ad400>
```



```
In [29]: 1 dataset_fastbook_9_corregido.unidades_vendidas.plot.box()
```

```
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x1af90212d68>
```



Vemos que, aunque quedan valores alejados en el histograma y en el box plot, ya entran en el rango de lo esperable. Además, si miramos los datos, veremos que se deben a campañas publicitarias u ofertas.

```
In [30]: 1 dataset_fastbook_9_corregido\
2 .loc[lambda x: (x.semana > "2019-11-07") & \
3       (x.semana < "2019-12-07")]
```

```
Out[30]:
```

	semana	unidades_vendidas	precio	publi1	publi2	publi3	oferta1	oferta2	oferta3
45	2019-11-11	11784.0	1.5	0	0	0	0.0	0.0	0.0
46	2019-11-18	11704.0	1.5	0	0	0	0.0	0.0	0.0
47	2019-11-25	21690.0	1.5	0	0	0	1.0	0.0	0.0
48	2019-12-02	10009.0	1.5	0	0	0	0.0	0.0	0.0

Criterios de consideración de outliers

Un problema en la detección de outliers es precisamente **establecer un umbral** que determine cuándo una observación se considera valor extremo.

En nuestro ejemplo de este fastbook, el tamaño del set de datos es pequeño, y podemos investigar y razonar las posibles causas de los outliers. No obstante, en datasets más grandes y con muchas variables, esto no será viable. De ahí la importancia de establecer **criterios para detectar y tratar outliers**.

Existen varios métodos para la detección automática de outliers, dependiendo de si son univariantes o multivariantes. A continuación, explicaremos dos sencillos **criterios univariantes**.

1

Estandarización de la variable

Una forma de detección de outliers es aplicar una estandarización a la variable en consideración, para posteriormente establecer un umbral a partir del que consideremos a las observaciones como outliers.

La fórmula concreta de esta estandarización es:

$$Z = \frac{x - \mu}{\sigma}$$

x son los valores de la variable, μ su media poblacional, y σ su desviación típica poblacional.

En la práctica, dado que probablemente no vayamos a disponer de los datos poblacionales, podemos sustituirlos por la media y desviación típica de la muestra x.

Mediante la anterior estandarización (también conocida como **z-score**), el valor absoluto de z indica la distancia de la observación a la media, expresada en las unidades de la varianza.

Como estas medidas ya son independientes de medias y desviaciones, podemos establecer **umbrales comunes** para cualquier set de datos. Probablemente, el más utilizado para detección de outliers sea **2.5**: si el valor absoluto de z supera este umbral, se considerará la observación como valor extremo. Si contamos con una gran variabilidad en la muestra, podemos subir este límite hasta **3.5**, o reducirlo **por debajo de 2** si la variabilidad es muy pequeña.

Veamos un ejemplo con nuestro dataset anterior.

Si aplicamos la fórmula anterior, vemos que los dos outliers que hemos detectado previamente de forma manual son los que mayor valor absoluto de z presentan. En particular, el caso de 120.000 unidades parece un outlier claro con esta fórmula.

```
In [31]: 1  dataset_fastbook_9\  
2  .assign(z = lambda x: (x.unidades_vendidas -  
3  x.unidades_vendidas.mean())/\\  
4  x.unidades_vendidas.std())\  
5  .assign(z_abs = lambda x: x.z.abs())\  
6  .loc[lambda x: x.z_abs > 1]\\  
7  .sort_values("z_abs", ascending = False)\  
8  [["semana", "unidades_vendidas", "oferta1", "z", "z_abs"]]
```

Out[31]:

	semana	unidades_vendidas	oferta1	z	z_abs
7	2019-02-18	120080.0	0.0	9.986703	9.986703
35	2019-09-02	18.0	0.0	-1.149545	1.149545

2

Desviación mediana absoluta

La desviación mediana absoluta (**MAD**, del inglés *median absolute deviation*) es una medida estadística que se calcula como la **mediana de las distancias de cada punto a la mediana de los datos**.

$$MAD(x) = \text{mediana}(|x_i - \text{mediana}(x)|)$$

La ventaja de esta medida es que es muy robusta a outliers, dado que está basada en la mediana y no en la media (cuyo valor sí se puede ver alterado significativamente por un outlier).

Para aplicarla a la detección de outliers, la utilizaremos junto a la mediana para sustituir a la desviación típica y la media que veíamos en el apartado anterior. Por tanto, la nueva fórmula sería la siguiente.

$$\frac{x - \text{mediana}(x)}{MAD(x)}$$

Y de igual forma que en el apartado anterior, podemos utilizar un umbral de detección de outliers común para cualquier dataset.

Veamos qué ocurre si aplicamos este nuevo criterio sobre nuestro dataset.

De nuevo, los dos outliers aparecen en primer lugar. No obstante, en este caso, aunque el dato de 120.000 unidades aparezca muy alejado, nuestro segundo outlier, con 18 unidades, está muy cerca del caso de 24.000. Como este método es univariante, no ha tenido en cuenta la información de publicidad y ofertas, y no puede detectar que este último valor se debe a la presencia de una oferta.

```
In [32]: 1  dataset_fastbook_9\
2  .assign(z_mad = lambda x:
3      (x.unidades_vendidas - x.unidades_vendidas.median())/\\
4      x.unidades_vendidas.mad())\
5  .assign(z_mad_abs = lambda x: x.z_mad.abs())\
6  .loc[lambda x: x.z_mad_abs > 1]\\
7  .sort_values("z_mad_abs", ascending = False)\\
8  [[{"semana": "unidades_vendidas", "oferta1": "z_mad", "z_mad_abs"}]]
```

Out[32]:

	semana	unidades_vendidas	oferta1	z_mad	z_mad_abs
7	2019-02-18	120080.0	0.0	43.286473	43.286473
35	2019-09-02	18.0	0.0	-4.476676	4.476676
47	2019-11-25	21690.0	1.0	4.144894	4.144894
21	2019-05-27	16050.0	0.0	1.901185	1.901185
20	2019-05-20	15281.0	0.0	1.595261	1.595261
99	2020-11-23	14136.0	0.0	1.139756	1.139756
98	2020-11-16	13968.0	0.0	1.072922	1.072922
100	2020-11-30	13811.0	0.0	1.010465	1.010465
84	2020-08-10	8750.0	0.0	-1.002906	1.002906

Eliminación de outliers

Otra cuestión importante referente a los outliers es cuándo debemos eliminarlos. **Dependerá de la importancia de estas observaciones dentro de nuestros datos, y del tipo de modelo que queramos realizar.**

Tened en cuenta que, si eliminamos estas observaciones extremas para la realización del modelo, y posteriormente aparecen más datos extremos, el modelo no habrá contado con **ninguna observación de referencia durante su entrenamiento. Y por tanto las predicciones de**

En nuestro ejemplo parecía claro que los valores de 120.000 y 18 eran errores de datos, mientras que el de 24.000 se debía a la primera oferta.

Relaciones entre variables

X Edix Educación

Correlaciones

La **correlación de Pearson** se define como una medida de dependencia lineal entre dos variables cuantitativas.

Esta métrica nos ayudará a entender cómo se relacionan las variables entre sí, y en particular, **cuáles son las variables que mayor relación tienen con nuestra variable objetivo de modelización.**

La correlación de Pearson es una métrica normalizada, es decir, no depende de la escala de las variables que mide. Siempre oscilará entre -1 y 1, donde la magnitud (el valor absoluto) indica la fuerza de la relación; y el signo, si se trata de una relación directa (positivo) o inversa (negativo).

En Pandas, tenemos disponible el método `corr` para calcular las correlaciones entre pares de variables de un dataframe.

Centrémonos en la columna de unidades vendidas, ya que es la variable que queremos explicar.

In [33]:	1	dataset_fastbook_9_corregido.corr()
Out[33]:		
		unidades_vendidas precio publi1 publi2 publi3 oferta1
	unidades_vendidas	1.000000 -0.464637 0.380821 0.135934 0.303047 0.641993
	precio	-0.464637 1.000000 -0.109005 -0.097013 -0.097013 -0.047802
	publi1	0.380821 -0.109005 1.000000 -0.044499 -0.044499 -0.021926
	publi2	0.135934 -0.097013 -0.044499 1.000000 -0.039604 -0.019514
	publi3	0.303047 -0.097013 -0.044499 -0.039604 1.000000 -0.019514
	oferta1	0.641993 -0.047802 -0.021926 -0.019514 -0.019514 1.000000
	oferta2	0.085077 -0.047802 -0.021926 0.492736 -0.019514 -0.009615
	oferta3	0.156335 -0.047802 -0.021926 -0.019514 0.492736 -0.009615

Vemos que existe una correlación negativa con el precio, y positiva con la primera oferta. El resto de las variables tienen una correlación muy baja, principalmente debido a que en la mayoría de las semanas mantienen el valor 0. Esto nos puede indicar que hay una relación entre realizar una oferta o reducir el precio (ya que el signo de la correlación es negativo) y aumentar las ventas.

No obstante, la correlación no tiene por qué implicar causalidad, ya que dos series de datos pueden ser muy similares sin tener ninguna relación entre sí. Este tipo de relaciones se conocen como espurias (podéis encontrar casos muy curiosos de correlaciones espurias [aquí](#)). Además, puede existir una tercera variable que influya en la relación de las otras dos. Por ejemplo, si decimos que a final de noviembre existe una estacionalidad que hace que suban las ventas de un producto, pero esto se debe a que se realizan ofertas de Black Friday. A esto se le denomina **factor de confusión, o confounding variable**.

Por último, si utilizamos correlaciones lineales podemos estar dejando escapar relaciones no lineales entre variables. Por ello, lo recomendable siempre es visualizar las relaciones entre variables, como vamos a ver en el siguiente apartado.

Visualización

Los histogramas nos pueden dar mucha información sobre las variables de un dataframe, pero por su propia definición son gráficos univariantes, y no nos mostrarán la relación que tienen las columnas (o variables en el contexto de un modelo) entre sí.

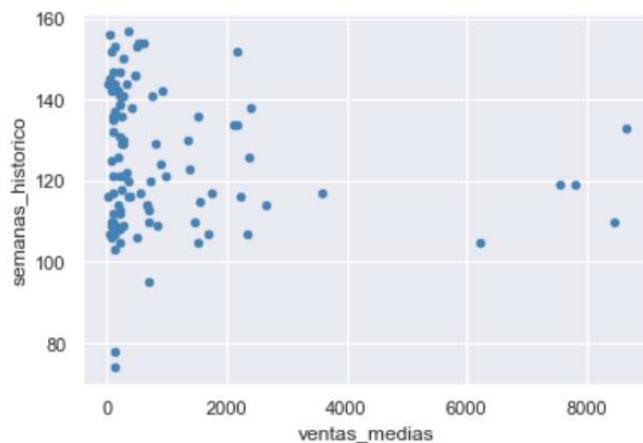
Lo más sencillo para representar gráficamente la relación entre dos variables es realizar un **gráfico de dispersión** (también llamado *scatterplot*), donde una variable se represente en el eje horizontal, y otra en el eje vertical. De esta forma, los puntos nos ayudarán a entender cómo se relacionan estas variables.

Veamos un ejemplo con el dataset de productos de los fastbooks anteriores.

En este caso vemos una dispersión de los puntos que nos impide sacar conclusiones sobre la relación entre los datos, más allá de que los productos con muy pocas semanas de histórico han tenido pocas ventas medias.

```
In [34]: ⏎ 1 ▾   productos\  
2 ▾     .plot(x = "ventas_medias", y = "semanas_historico",  
3           kind = "scatter", c = "steelblue")
```

```
Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x2136b6bcf28>
```

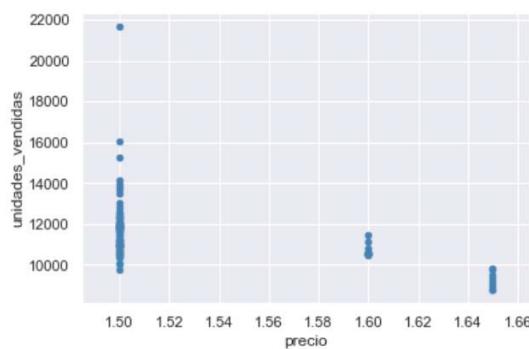


Si una de las variables tiene pocos valores, será complicado interpretar la información con este tipo de gráficos. Veámoslo sobre el dataset de este fastbook.

Como los puntos se solapan, es complicado interpretar la información visual.

```
In [35]: 1 v  dataset_fastbook_9_corregido\  
2 v      .plot(x = "precio", y = "unidades_vendidas",  
3       kind = "scatter", c = "steelblue")
```

```
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x2136daf3ba8>
```

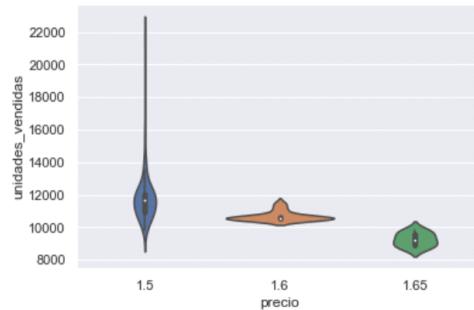


En casos como este, podemos utilizar los **gráficos de violín**, donde se muestra la densidad de los puntos en el eje horizontal. Para ello, utilizaremos la librería de visualización *seaborn*.

Ahora ya vemos cómo los niveles de unidades vendidas decrecen conforme aumenta el precio del producto.

```
In [36]: 1 import seaborn as sns  
2 sns.violinplot('precio','unidades_vendidas',  
3                  data=dataset_fastbook_9_corregido)
```

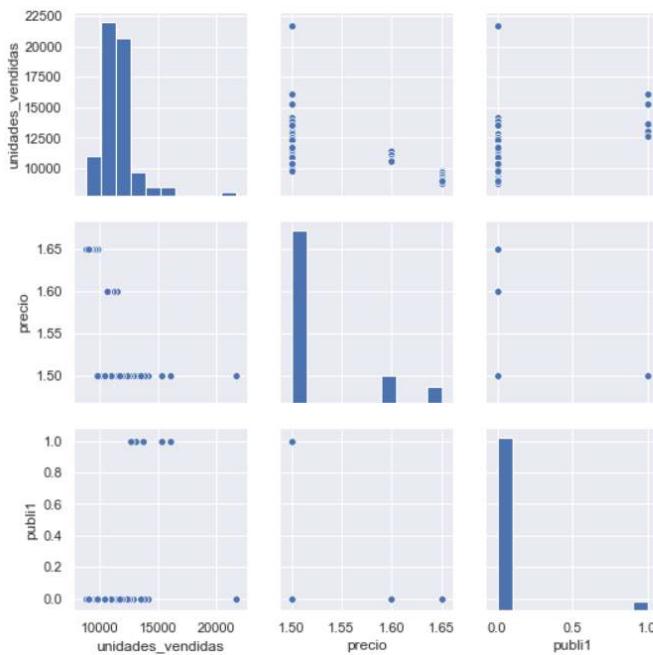
```
Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x2136dbea128>
```



Por último, podemos generar los gráficos de dispersión para cada par de variables mediante la función *pairplot*, también de *seaborn*. Además, para representar la relación de una variable consigo misma, creará un histograma.

```
In [37]: 1  sns.pairplot(data=dataset_fastbook_9_corregido\
2          [[ "unidades_vendidas", "precio", "publi1"]])
```

```
Out[37]: <seaborn.axisgrid.PairGrid at 0x2136dc05f60>
```



Lesson 5 of 6

Regresiones lineales

 Edix Educación

La regresión lineal es uno de los modelos matemáticos más sencillos.

Nos permite aproximar la relación de dependencia entre una variable objetivo y una o más variables independientes.

Con la regresión lineal podremos **explicar los comportamientos pasados e incluso predecir los futuros.**

Por ejemplo, podremos entender qué medios publicitarios realmente han tenido influencia en las ventas de un producto, y llegar a predecir sus ventas futuras si conocemos (o podemos aproximar) el comportamiento futuro de las variables de nuestra regresión.

Dado que no es el objetivo de esta asignatura profundizar en los conceptos matemáticos de la regresión lineal, nos limitaremos a recordar su fórmula concreta:

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_n X_n$$

En la fórmula, la variable objetivo Y se expresa mediante una suma de variables explicativas X_1, \dots, X_n , multiplicada cada una por un coeficiente β_1, \dots, β_n . Además, se incluye el término β_0 (también llamado *intercept*) como nivel base, sin multiplicar a ninguna variable.

Por tanto, podemos explicar la variable objetivo a partir de una suma ponderada de las variables explicativas: cuanto más grande sea su coeficiente en valor absoluto, mayor importancia tendrán en el modelo lineal.

En Python, disponemos de varias librerías que implementan este tipo de modelos. Nosotros utilizaremos *statsmodels*, una librería centrada en modelos y tests estadísticos.

Para utilizarla, deberemos importarla previamente de la siguiente forma:

```
In [38]: 1 import statsmodels.api as sm
```

La función que utilizaremos para obtener la regresión lineal es *OLS* (iniciales de *ordinary least squares*, o mínimos cuadrados ordinarios). Este método minimiza la suma de los cuadrados de las diferencias entre los valores reales de la variable objetivo y las predicciones.

Esta función aceptará dos parámetros principales: el array unidimensional con los valores de la variable objetivo (parámetro *endog*, de variable endógena), y el array bidimensional (o dataframe) con los valores de las variables explicativas (parámetro *exog*, de exógena).

Creemos en primer lugar esos dos objetos.

```
In [39]: 1 endog = dataset_fastbook_9_corregido.unidades_vendidas
          2 exog = dataset_fastbook_9_corregido\
          3         .drop(["unidades_vendidas", "semana"], axis = 1)
          4 exog['intercept'] = 1
          5 regresion_lineal = sm.OLS(endog, exog)
          6
          7 regresion_lineal
Out[39]: <statsmodels.regression.linear_model.OLS at 0x2136fb4b9b0>
```

Como veis, hemos incluido una variable explicativa adicional: el término constante o *intercept*. Ya tenemos listo nuestro modelo lineal, pero antes de poder utilizarlo, deberemos ajustarlo mediante el método de mínimos cuadrados que hemos comentado. Para ello, ejecutaremos el método *fit* de nuestro objeto *regresion_lineal*.

```
In [40]: 1 regresion_lineal = regresion_lineal.fit()
```

Una vez que nuestro modelo está entrenado, podremos obtener sus predicciones para el dataset. Para ello, utilizaremos el método *predict* del modelo que acabamos de generar sobre nuestro dataframe de variables exógenas.

```
In [41]: 1 predicion = regresion_lineal.predict(exog)
          2 predicion
```



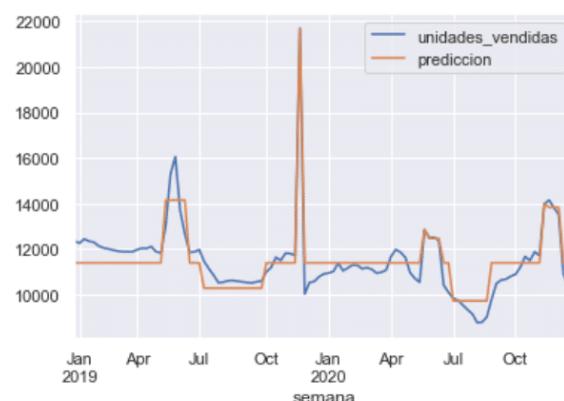
```
Out[41]: 0    11379.495902
         1    11379.495902
         2    11379.495902
         3    11379.495902
         4    11379.495902
         ...
        100   13819.333333
        101   13819.333333
        102   11379.495902
        103   11379.495902
        104   11379.495902
Length: 105, dtype: float64
```

Con esta predicción, ya podemos pintarla y compararla con el dato real.

Vemos que, aunque nuestro modelo es muy simple, es capaz de explicar las subidas y bajadas de los niveles de ventas debidas a los cambios de precio, campañas publicitarias y ofertas.

```
In [42]: 1 v dataset_fastbook_9_corregido\
          2     .assign(predicion = predicion)\\
          3     .plot(x = "semana", y = ["unidades_vendidas", "predicion"])
```

```
Out[42]: <matplotlib.axes._subplots.AxesSubplot at 0x2023456d4e0>
```



Por último, si queremos obtener un resumen del modelo, utilizaremos el método ***summary***, con el que podremos ver los coeficientes de las variables o el ajuste del modelo (veréis más sobre métricas de calidad para saber cómo de buenos son nuestros modelos en otras asignaturas).

Por ejemplo, vemos que los coeficientes de la publicidad y las ofertas son positivos (si se realizan, el número de unidades aumenta), mientras que el del precio es negativo (al subir el precio, disminuyen las unidades vendidas).

In [43]: 1 regresion_lineal.summary()

Out[43]: OLS Regression Results

Dep. Variable:	unidades_vendidas	R-squared:	0.830			
Model:	OLS	Adj. R-squared:	0.818			
Method:	Least Squares	F-statistic:	67.81			
Date:	Mon, 14 Jun 2021	Prob (F-statistic):	1.48e-34			
Time:	16:19:48	Log-Likelihood:	-827.81			
No. Observations:	105	AIC:	1672.			
Df Residuals:	97	BIC:	1693.			
Df Model:	7					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
precio	-1.116e+04	1360.205	-8.201	0.000	-1.39e+04	-8455.820
publi1	2758.1041	309.190	8.920	0.000	2144.448	3371.760
publi2	1071.8374	393.864	2.721	0.008	290.127	1853.548
publi3	2439.8374	393.864	6.195	0.000	1658.127	3221.548
oferta1	1.031e+04	672.890	15.323	0.000	8975.004	1.16e+04
oferta2	383.6667	771.559	0.497	0.620	-1147.665	1914.998
oferta3	148.6667	771.559	0.193	0.848	-1382.665	1679.998
intercept	2.811e+04	2078.857	13.523	0.000	2.4e+04	3.22e+04

Con esto hemos concluido nuestro primer acercamiento a la modelización en Python a través de la regresión lineal. En asignaturas posteriores profundizaréis mucho más sobre este tema, aprendiendo tanto técnicas como modelos avanzados, y haciendo uso de librerías potentes como *scikit-learn*, *prophet* o *xgboost*.

Conclusiones

 Edix Educación

En este fastbook hemos aprendido a realizar un proceso de análisis exploratorio de datos. Hemos comenzado por el principio de cualquier proyecto de datos: **importando** las tablas necesarias y **comprobando** que los datos son correctos. Una vez que están cargados, deberemos asegurarnos de que no tenemos valores perdidos que puedan interferir en nuestro análisis, y tratarlos adecuadamente si existen.

Hemos continuado por la **exploración gráfica de los datos**: una forma sencilla e intuitiva de entender las variables de las que disponemos y cómo se distribuyen. En este apartado hemos dado un rápido repaso a los principales gráficos univariantes: histogramas, gráficos de densidad, de evolución, de barras y de caja (o box plot). Hemos continuado con los outliers, u observaciones extremas dentro de una variable, y aprendido a detectarlos tanto gráficamente como mediante umbrales.

Finalmente, nos hemos adentrado en las **relaciones entre variables**, y hemos visto cómo las correlaciones pueden darnos indicaciones sobre el impacto de algunas de ellas sobre nuestra variable objetivo. Por último, hemos visto cómo realizar nuestro primer modelo en Python: la **regresión lineal**.

Con este fastbook ya hemos finalizado nuestro aprendizaje de la programación en Python centrada en la analítica de datos, y dedicaremos el último fastbook a conceptos más generales de Python como la orientación a objetos o los tests unitarios.

¡Enhорabuena! Fastbook superado

edix

Creamos Digital Workers