Introducción

Terminado 100 XP 2 minutos Como era de esperar, el papel de un científico de datos implica principalmente explorar y analizar datos. Los resultados de un análisis pueden formar la base de un informe o un modelo de aprendizaje automático, pero todo comienza con los datos, siendo Python el lenguaje de programación más popular para los científicos de datos.

Después de décadas de desarrollo de código abierto, Python proporciona una amplia funcionalidad con potentes bibliotecas estadísticas y numéricas:

NumPy y Pandas simplifican el análisis y la manipulación de datos Matplotlib proporciona visualizaciones de datos atractivas Scikit-learn ofrece un análisis de datos predictivo simple y efectivo TensorFlow y PyTorch ofrecen capacidades de aprendizaje automático y aprendizaje profundo Por lo general, un proyecto de análisis de datos está diseñado para establecer conocimientos sobre un escenario particular o para probar una hipótesis.

Por ejemplo, supongamos que un profesor universitario recopila datos de sus estudiantes, incluido el número de conferencias a las que asistieron, las horas dedicadas al estudio y la calificación final obtenida en el examen de fin de trimestre. El profesor podría analizar los datos para determinar si existe una relación entre la cantidad de estudios que realiza un alumno y la nota final que obtiene. El profesor podría usar los datos para probar la hipótesis de que solo los estudiantes que estudian durante un número mínimo de horas pueden esperar obtener una calificación aprobatoria.

Explore datos con NumPy y Pandas Terminado 100 XP 3 minutos Los científicos de datos pueden utilizar diversas herramientas y técnicas para explorar, visualizar y manipular datos. Una de las formas más comunes en las que los científicos de datos trabajan con datos es utilizar el lenguaje Python y algunos paquetes específicos para el procesamiento de datos.

¿Qué es NumPy? NumPy es una biblioteca de Python que brinda una funcionalidad comparable a las herramientas matemáticas como MATLAB y R. Si bien NumPy simplifica significativamente la experiencia del usuario, también ofrece funciones matemáticas integrales.

¿Qué son los pandas? Pandas es una biblioteca de Python extremadamente popular para el análisis y la manipulación de datos. Pandas es como Excel para Python: proporciona una funcionalidad fácil de usar para tablas de datos.

Explore los datos en un cuaderno de Jupyter Los cuadernos de Jupyter son una forma popular de ejecutar scripts básicos con su navegador web. Por lo general, estos cuadernos son una sola página web, dividida en secciones de texto y secciones de código que se ejecutan en el servidor en lugar de en su máquina local. Esto significa que puede comenzar rápidamente sin necesidad de instalar Python u otras herramientas.

Explorando datos con Python Una parte importante del rol de un científico de datos es explorar, analizar y visualizar datos. Existe una amplia gama de herramientas y lenguajes de programación que pueden usar para hacer esto; y uno de los enfoques más populares es usar cuadernos de Jupyter (como este) y Python.

Python es un lenguaje de programación flexible que se utiliza en una amplia gama de escenarios; desde aplicaciones web hasta programación de dispositivos. Es extremadamente popular en la comunidad de ciencia de datos y aprendizaje automático debido a los muchos paquetes que admite para el análisis y la visualización de datos.

En este cuaderno, exploraremos algunos de estos paquetes y aplicaremos técnicas básicas para analizar datos. No se pretende que sea un ejercicio completo de programación de Python; o incluso una inmersión profunda en el análisis de datos. Más bien, está pensado como un curso acelerado sobre algunas de las formas comunes en las que los científicos de datos pueden usar Python para trabajar con datos.

Explorando matrices de datos con NumPy Comencemos mirando algunos datos simples.

Suponga que una universidad toma una muestra de las calificaciones de los estudiantes para una clase de ciencia de datos.

```
1 data = [50,50,47,97,49,3,53,42,26,74,82,62,37,15,70,27,36,35,48,52,63,64]
2 print(data)
```

```
[50, 50, 47, 97, 49, 3, 53, 42, 26, 74, 82, 62, 37, 15, 70, 27, 36, 35, 48, 52, 6
```

Los datos se han cargado en una estructura de lista de Python , que es un buen tipo de datos para la manipulación de datos en general, pero no está optimizado para el análisis numérico. Para ello, vamos a utilizar el NumPy paquete, que incluye los tipos de datos y funciones específicas para trabajar con Num fibras en Py Thon.

```
1 import numpy as np
2
3 grades = np.array(data)
4 print(grades)
```

```
[50 50 47 97 49  3 53 42 26 74 82 62 37 15 70 27 36 35 48 52 63 64]
```

En caso de que se esté preguntando acerca de las diferencias entre una lista y una matriz NumPy , comparemos cómo se comportan estos tipos de datos cuando los usamos en una expresión que los multiplica por 2.

```
1 print (type(data),'x 2:', data * 2)
2 print('---')
```

```
3 print (type(grades),'x 2:', grades * 2)
```

```
<class 'list'> x 2: [50, 50, 47, 97, 49, 3, 53, 42, 26, 74, 82, 62, 37, 15, 70, 2
---
<class 'numpy.ndarray'> x 2: [100 100  94 194  98   6 106  84  52 148 164 124  74
  96 104 126 128]
```

Tenga en cuenta que multiplicar una lista por 2 crea una nueva lista de dos veces la longitud con la secuencia original de elementos de la lista repetida. Al multiplicar una matriz NumPy, por otro lado, se realiza un cálculo por elementos en el que la matriz se comporta como un vector , por lo que terminamos con una matriz del mismo tamaño en la que cada elemento se ha multiplicado por 2.

La conclusión clave de esto es que las matrices NumPy están diseñadas específicamente para admitir operaciones matemáticas en datos numéricos, lo que las hace más útiles para el análisis de datos que una lista genérica.

Es posible que haya notado que el tipo de clase para la matriz numpy anterior es un numpy.ndarray . La nd indica que esta es una estructura que puede constar de múltiples dimensiones (puede tener n dimensiones). Nuestra instancia específica tiene una sola dimensión de calificaciones de los estudiantes.

Ejecute la celda de abajo para ver la forma de la matriz.

```
1 grades.shape
```

```
(22,)
```

La forma confirma que esta matriz tiene una sola dimensión, que contiene 22 elementos (hay 22 grados en la lista original). Puede acceder a los elementos individuales de la matriz por su posición ordinal basada en cero. Consigamos el primer elemento (el que está en la posición 0).

```
1 grades[0]
```

```
50
```

Muy bien, ahora que conoce el camino alrededor de una matriz NumPy, es hora de realizar un análisis de los datos de calificaciones.

Puede aplicar agregaciones entre los elementos de la matriz, así que busquemos la calificación promedio simple (en otras palabras, el valor medio de la calificación).

```
1 grades.mean()
```

```
49.18181818181818
```

Entonces, la nota media es de alrededor de 50, más o menos en el medio del rango posible de 0 a 100.

Agreguemos un segundo conjunto de datos para los mismos estudiantes, esta vez registrando el número típico de horas por semana que dedicaron a estudiar.

```
# Define an array of study hours
study_hours = [10.0,11.5,9.0,16.0,9.25,1.0,11.5,9.0,8.5,14.5,15.5,
               13.75,9.0,8.0,15.5,8.0,9.0,6.0,10.0,12.0,12.5,12.0]

# Create a 2D array (an array of arrays)
student_data = np.array([study_hours, grades])

# display the array
student_data
```

```
array([[10.  , 11.5 ,  9.  , 16.  ,  9.25,  1.  , 11.5 ,  9.  ,  8.5 ,
        14.5 , 15.5 , 13.75,  9.  ,  8.  , 15.5 ,  8.  ,  9.  ,  6.  ,
        10.  , 12.  , 12.5 , 12.  ],
       [50.  , 50.  , 47.  , 97.  , 49.  ,  3.  , 53.  , 42.  , 26.  ,
        74.  , 82.  , 62.  , 37.  , 15.  , 70.  , 27.  , 36.  , 35.  ,
        48.  , 52.  , 63.  , 64.  ]])
```

Ahora los datos consisten en una matriz bidimensional, una matriz de matrices. Veamos su forma.

```
# Show shape of 2D array
student_data.shape
```

```
(2, 22)
```

La matriz student_data contiene dos elementos, cada uno de los cuales es una matriz que contiene 22 elementos.

Para navegar por esta estructura, debe especificar la posición de cada elemento en la jerarquía. Entonces, para encontrar el primer valor en la primera matriz (que contiene los datos de las horas de estudio), puede usar el siguiente código.

```
# Show the first element of the first element
student_data[0][0]
```

```
10.0
```

Ahora tiene una matriz multidimensional que contiene tanto el tiempo de estudio del estudiante como la información de calificaciones, que puede usar para comparar datos. Por ejemplo, ¿cómo se compara el tiempo medio de estudio con la nota media?

```
# value of each sub-array
```

```
value of each sub-array
dent_data[0].mean()
dent_data[1].mean()

study hours: {:.2f}\nAverage grade: {:.2f}'.format(avg_study, avg_grade))
```

```
Average study hours: 10.52
Average grade: 49.18
```

Explorando datos tabulares con Pandas Si bien NumPy proporciona muchas de las funciones que necesita para trabajar con números, y específicamente matrices de valores numéricos; Cuando comienza a trabajar con tablas de datos bidimensionales, el paquete Pandas ofrece una estructura más conveniente para trabajar: el DataFrame .

Ejecute la siguiente celda para importar la biblioteca Pandas y cree un DataFrame con tres columnas. La primera columna es una lista de los nombres de los estudiantes, y la segunda y tercera columnas son las matrices NumPy que contienen el tiempo de estudio y los datos de las calificaciones.

```
import pandas as pd

df_students = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan
                                     'Rhonda', 'Giovanni', 'Francesca', 'Raja
                                     'Jakeem','Helena','Ismat','Anila','Skye'
                            'StudyHours':student_data[0],
                            'Grade':student_data[1]})

df_students
```

	Name	StudyHours	Grade
0	Dan	10.00	50.0
1	Joann	11.50	50.0
2	Pedro	9.00	47.0
3	Rosie	16.00	97.0
4	Ethan	9.25	49.0
5	Vicky	1.00	3.0
6	Frederic	11.50	53.0
7	Jimmie	9.00	42.0

Tenga en cuenta que, además de las columnas que especificó, el DataFrame incluye un índice para identificar de forma única cada fila. Podríamos haber especificado el índice explícitamente y asignado cualquier tipo de valor apropiado (por ejemplo, una dirección de correo electrónico); pero como no especificamos un índice, se ha creado uno con un valor entero único para cada fila.

Encontrar y filtrar datos en un DataFrame Puede usar el método loc de DataFrame para recuperar datos para un valor de índice específico, como este.

```
1 # Get the data for index value 5
2 df_students.loc[5]
```

```
Name          Vicky
StudyHours        1
Grade             3
Name: 5, dtype: object
```

También puede obtener los datos en un rango de valores de índice, como este:

```
1 # Get the rows with index values from 0 to 5
2 df_students.loc[0:5]
```

	Name	StudyHours	Grade
0	Dan	10.00	50.0
1	Joann	11.50	50.0
2	Pedro	9.00	47.0
3	Rosie	16.00	97.0
4	Ethan	9.25	49.0
5	Vicky	1.00	3.0

Además de poder usar el método loc para buscar filas según el índice, puede usar el método iloc para buscar filas según su posición ordinal en el DataFrame (independientemente del índice):

```
1 # Get data in the first five rows
2 df_students.iloc[0:5]
```

	Name	StudyHours	Grade
0	Dan	10.00	50.0
1	Joann	11.50	50.0
2	Pedro	9.00	47.0
3	Rosie	16.00	97.0
4	Ethan	9.25	49.0

Mire cuidadosamente los iloc[0:5]resultados y compárelos con los loc[0:5]resultados que obtuvo anteriormente. ¿Puedes ver la diferencia?

El método loc devolvió filas con etiqueta de índice en la lista de valores de 0 a 5 , que incluye 0 , 1 , 2 , 3 , 4 y 5 (seis filas). Sin embargo, el método iloc devuelve las filas en las posiciones incluidas en el rango de 0 a 5, y dado que los rangos de números enteros no incluyen el valor del límite superior, esto incluye las posiciones 0 , 1 , 2 , 3 y 4 (cinco filas) .

iloc identifica valores de datos en un DataFrame por posición , que se extiende más allá de las filas a las columnas. Entonces, por ejemplo, puede usarlo para encontrar los valores de las columnas en las posiciones 1 y 2 en la fila 0, así:

```
1 df_students.iloc[0,[1,2]]
```

```
StudyHours    10
Grade         50
Name: 0, dtype: object
```

Volvamos al método loc y veamos cómo funciona con columnas. Recuerde que loc se usa para ubicar elementos de datos basados en valores de índice en lugar de posiciones. En ausencia de una columna de índice explícita, las filas en nuestro marco de datos se indexan como valores enteros, pero las columnas se identifican por su nombre:

```
1 df_students.loc[0,'Grade']
```

```
50.0
```

Aquí hay otro truco útil. Puede usar el método loc para buscar filas indexadas en función de una expresión de filtrado que haga referencia a columnas con nombre distintas del índice, como esta:

```
df_students.loc[df_students['Name']=='Aisha']
```

	Name	StudyHours	Grade
21	Aisha	12.0	64.0

En realidad, no necesita usar explícitamente el método loc para hacer esto; simplemente puede aplicar una expresión de filtrado de DataFrame, como esta:

```
df_students[df_students['Name']=='Aisha']
```

	Name	StudyHours	Grade
21	Aisha	12.0	64.0

Y en buena medida, puede lograr los mismos resultados utilizando el método de consulta de DataFrame , como este:

```
df_students.query('Name=="Aisha"')
```

	Name	StudyHours	Grade
21	Aisha	12.0	64.0

Los tres ejemplos anteriores subrayan una verdad ocasionalmente confusa sobre trabajar con Pandas. A menudo, hay varias formas de lograr los mismos resultados. Otro ejemplo de esto es la forma en que se refiere al nombre de una columna de DataFrame. Puede especificar el nombre de la columna como un valor de índice con nombre (como en los df_students['Name']ejemplos que hemos visto hasta ahora), o puede usar la columna como una propiedad del DataFrame, así:

```
df_students[df_students.Name == 'Aisha']
```

	Name	StudyHours	Grade
21	Aisha	12.0	64.0

Cargando un DataFrame desde un archivo Construimos el DataFrame a partir de algunas matrices existentes. Sin embargo, en muchos escenarios del mundo real, los datos se cargan desde fuentes como archivos. Reemplacemos el DataFrame de calificaciones de los estudiantes con el contenido de un archivo de texto.

```
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to
df_students = pd.read_csv('grades.csv',delimiter=',',header='infer')
df_students.head()
```

```
--2021-10-01 20:08:01--  https://raw.githubusercontent.com/MicrosoftDocs/mslearn-
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.13
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.1
HTTP request sent, awaiting response... 200 OK
Length: 322 [text/plain]
Saving to: ‘grades.csv’

grades.csv          100%[===================>]     322  --.-KB/s    in 0s

2021-10-01 20:08:01 (11.6 MB/s) - ‘grades.csv’ saved [322/322]
```

	Name	StudyHours	Grade
0	Dan	10.00	50.0
1	Joann	11.50	50.0
2	Pedro	9.00	47.0
3	Rosie	16.00	97.0
4	Ethan	9.25	49.0

El método read_csv de DataFrame se usa para cargar datos desde archivos de texto. Como puede ver en el código de ejemplo, puede especificar opciones como el delimitador de columna y qué fila (si hay alguna) contiene encabezados de columna (en este caso, el delimitador es una coma y la primera fila contiene los nombres de columna; estos son los configuración predeterminada, por lo que los parámetros podrían haberse omitido).

Manejo de valores perdidos Uno de los problemas más comunes con los que deben lidiar los científicos de datos es la falta de datos o los datos incompletos. Entonces, ¿cómo sabríamos que el DataFrame contiene valores faltantes? Puede usar el método isnull para identificar qué valores individuales son nulos, como este:

```
1 df_students.isnull()
```

	Name	StudyHours	Grade
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False
4	False	False	False
5	False	False	False
6	False	False	False
7	False	False	False
8	False	False	False
9	False	False	False
10	False	False	False
11	False	False	False
12	False	False	False
13	False	False	False

Por supuesto, con un DataFrame más grande, sería ineficiente revisar todas las filas y columnas individualmente; para que podamos obtener la suma de los valores perdidos para cada columna, así:

```
1 df_students.isnull().sum()
```

```
Name          0
StudyHours    1
Grade         2
dtype: int64
```

Ahora sabemos que falta un valor de StudyHours y dos valores de Grade que faltan .

Para verlos en contexto, podemos filtrar el marco de datos para incluir solo filas donde cualquiera de las columnas (eje 1 del marco de datos) sea nula.

```
1 df_students[df_students.isnull().any(axis=1)]
```

	Name	StudyHours	Grade
22	Bill	8.0	NaN
23	Ted	NaN	NaN

Cuando se recupera el DataFrame, los valores numéricos faltantes se muestran como NaN (no como un número).

Entonces, ahora que hemos encontrado los valores nulos, ¿qué podemos hacer con ellos?

Un enfoque común es imputar valores de reemplazo. Por ejemplo, si falta el número de horas de estudio, podríamos simplemente asumir que el estudiante estudió durante un tiempo promedio y reemplazar el valor faltante con la media de horas de estudio. Para hacer esto, podemos usar el método fillna , así:

```
df_students.StudyHours = df_students.StudyHours.fillna(df_students.StudyHours
df_students
```

	Name	StudyHours	Grade
0	Dan	10.000000	50.0
1	Joann	11.500000	50.0
2	Pedro	9.000000	47.0
3	Rosie	16.000000	97.0
4	Ethan	9.250000	49.0
5	Vicky	1.000000	3.0
6	Frederic	11.500000	53.0
7	Jimmie	9.000000	42.0
8	Rhonda	8.500000	26.0
9	Giovanni	14.500000	74.0
10	Francesca	15.500000	82.0
11	Rajab	13.750000	62.0
12	Naiyana	9.000000	37.0
13	Kian	8.000000	15.0
14	Jenny	15.500000	70.0
15	Jakeem	8.000000	27.0
16	Helena	9.000000	36.0
17	Ismat	6.000000	35.0
18	Anila	10.000000	48.0
19	Skye	12.000000	52.0
20	Daniel	12.500000	63.0
21	Aisha	12.000000	64.0
22	Bill	8.000000	NaN
23	Ted	10.413043	NaN

Alternativamente, puede ser importante asegurarse de que solo use datos que sepa que son absolutamente correctos; para que pueda eliminar filas o columnas que contienen valores nulos mediante el método dropna . En este caso, eliminaremos filas (eje 0 del DataFrame) donde alguna de las columnas contiene valores nulos.

```
1 df_students = df_students.dropna(axis=0, how='any')
2 df_students
```

	Name	StudyHours	Grade
0	Dan	10.00	50.0
1	Joann	11.50	50.0
2	Pedro	9.00	47.0
3	Rosie	16.00	97.0
4	Ethan	9.25	49.0
5	Vicky	1.00	3.0
6	Frederic	11.50	53.0
7	Jimmie	9.00	42.0
8	Rhonda	8.50	26.0
9	Giovanni	14.50	74.0
10	Francesca	15.50	82.0
11	Rajab	13.75	62.0
12	Naiyana	9.00	37.0
13	Kian	8.00	15.0
14	Jenny	15.50	70.0
15	Jakeem	8.00	27.0
16	Helena	9.00	36.0
17	Ismat	6.00	35.0
18	Anila	10.00	48.0
19	Skye	12.00	52.0
20	Daniel	12.50	63.0
21	Aisha	12.00	64.0

Explore los datos en el DataFrame Ahora que hemos limpiado los valores faltantes, estamos listos para explorar los datos en el DataFrame. Comencemos comparando la media de horas de estudio y calificaciones.

```
rs using to column name as an index
['StudyHours'].mean()

ng the column name as a property (just to make the point!)
.Grade.mean()

ours and mean grade
udy hours: {:.2f}\nAverage grade: {:.2f}'.format(mean_study, mean_grade))
```

```
Average weekly study hours: 10.52
Average grade: 49.18
```

De acuerdo, filtremos el DataFrame para encontrar solo los estudiantes que estudiaron por más de la cantidad de tiempo promedio.

```
# Get students who studied for the mean or more hours
df_students[df_students.StudyHours > mean_study]
```

	Name	StudyHours	Grade
1	Joann	11.50	50.0
3	Rosie	16.00	97.0
6	Frederic	11.50	53.0
9	Giovanni	14.50	74.0
10	Francesca	15.50	82.0
11	Rajab	13.75	62.0
14	Jenny	15.50	70.0
19	Skye	12.00	52.0
20	Daniel	12.50	63.0
21	Aisha	12.00	64.0

Tenga en cuenta que el resultado filtrado es en sí mismo un DataFrame, por lo que puede trabajar con sus columnas como cualquier otro DataFrame.

Por ejemplo, busquemos la calificación promedio de los estudiantes que realizaron más tiempo de estudio que el promedio.

```
# What was their mean grade?
df_students[df_students.StudyHours > mean_study].Grade.mean()
```

```
66.7
```

Supongamos que la calificación aprobatoria del curso es 60.

Podemos usar esa información para agregar una nueva columna al DataFrame, indicando si cada estudiante aprobó o no.

Primero, crearemos una Serie Pandas que contiene el indicador de pasa / falla (Verdadero o Falso), y luego concatenamos esa serie como una nueva columna (eje 1) en el DataFrame.

```
passes  = pd.Series(df_students['Grade'] >= 60)
df_students = pd.concat([df_students, passes.rename("Pass")], axis=1)

df_students
```

	Name	StudyHours	Grade	Pass
0	Dan	10.00	50.0	False
1	Joann	11.50	50.0	False
2	Pedro	9.00	47.0	False
3	Rosie	16.00	97.0	True
4	Ethan	9.25	49.0	False
5	Vicky	1.00	3.0	False
6	Frederic	11.50	53.0	False
7	Jimmie	9.00	42.0	False
8	Rhonda	8.50	26.0	False
9	Giovanni	14.50	74.0	True
10	Francesca	15.50	82.0	True
11	Rajab	13.75	62.0	True
12	Naiyana	9.00	37.0	False
13	Kian	8.00	15.0	False
14	Jenny	15.50	70.0	True
15	Jakeem	8.00	27.0	False
16	Helena	9.00	36.0	False
17	Ismat	6.00	35.0	False
18	Anila	10.00	48.0	False
19	Skye	12.00	52.0	False
20	Daniel	12.50	63.0	True
21	Aisha	12.00	64.0	True

Los DataFrames están diseñados para datos tabulares y puede utilizarlos para realizar muchos de los tipos de operaciones de análisis de datos que puede realizar en una base de datos relacional; como agrupar y agregar tablas de datos.

Por ejemplo, puede usar el método groupby para agrupar los datos de los estudiantes en grupos según la columna de Aprobado que agregó anteriormente y contar el número de nombres en cada grupo: en otras palabras, puede determinar cuántos estudiantes aprobaron y reprobaron

```
1 print(df_students.groupby(df_students.Pass).Name.count())
```

```
Pass
False    15
True      7
Name: Name, dtype: int64
```

Puede agregar varios campos en un grupo utilizando cualquier función de agregación disponible. Por ejemplo, puede encontrar el tiempo medio de estudio y la calificación de los grupos de estudiantes que aprobaron y reprobaron el curso.

```
1 rint(df_students.groupby(df_students.Pass)['StudyHours', 'Grade'].mean())
```

```
       StudyHours      Grade
Pass
False    8.783333  38.000000
True    14.250000  73.142857
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: In
  """Entry point for launching an IPython kernel.
```

Los DataFrames son increíblemente versátiles y facilitan la manipulación de datos. Muchas operaciones de DataFrame devuelven una nueva copia del DataFrame; por lo que si desea modificar un DataFrame pero mantener la variable existente, debe asignar el resultado de la operación a la variable existente. Por ejemplo, el siguiente código ordena los datos de los estudiantes en orden descendente de Grade y asigna el DataFrame ordenado resultante a la variable df_students original .

```
1 # Create a DataFrame with the data sorted by Grade (descending)
2 df_students = df_students.sort_values('Grade', ascending=False)
3
4 # Show the DataFrame
5 df_students
```

	Name	StudyHours	Grade	Pass
3	Rosie	16.00	97.0	True
10	Francesca	15.50	82.0	True
9	Giovanni	14.50	74.0	True
14	Jenny	15.50	70.0	True
21	Aisha	12.00	64.0	True
20	Daniel	12.50	63.0	True
11	Rajab	13.75	62.0	True
6	Frederic	11.50	53.0	False
19	Skye	12.00	52.0	False
1	Joann	11.50	50.0	False
0	Dan	10.00	50.0	False
4	Ethan	9.25	49.0	False
18	Anila	10.00	48.0	False
2	Pedro	9.00	47.0	False
7	Jimmie	9.00	42.0	False
12	Naiyana	9.00	37.0	False
16	Helena	9.00	36.0	False
17	Ismat	6.00	35.0	False

Haz doble clic (o pulsa Intro) para editar

Visualizar datos Terminado 100 XP 3 minutos Los científicos de datos visualizan los datos para comprenderlos mejor. Esto puede significar mirar los datos brutos, medidas de resumen como promedios o graficar los datos. Los gráficos son un medio poderoso para ver datos, ya que podemos discernir patrones moderadamente complejos rápidamente sin necesidad de definir medidas de resumen matemáticas.

Representar datos visualmente Representar datos visualmente normalmente significa graficarlos. Esto se hace para proporcionar una evaluación cualitativa rápida de nuestros datos, que puede ser útil para comprender los resultados, encontrar valores atípicos, comprender cómo se distribuyen los números, etc.

Si bien a veces sabemos de antemano qué tipo de gráfico será más útil, otras veces los usamos de manera exploratoria. Para comprender el poder de la visualización de datos, considere los siguientes datos: la ubicación (x, y) de un automóvil autónomo. En su forma cruda, es difícil ver patrones reales. La media o promedio nos dice que su camino se centró alrededor de x = 0.2 ey = 0.3, y el rango de números parece estar entre -2 y 2.

Los gráficos no se limitan a diagramas de dispersión 2D como los anteriores, sino que se pueden usar para explorar otros tipos de datos, como proporciones, que se muestran a través de gráficos circulares, gráficos de barras apiladas, cómo se distribuyen los datos, con histogramas, diagramas de caja y bigotes, y cómo se diferencian dos conjuntos de datos. A menudo, cuando intentamos comprender datos o resultados sin procesar, es posible que experimentemos con diferentes tipos de gráficos hasta que encontremos uno que explique los datos de una manera visualmente intuitiva.

Ejercicio: visualizar datos con Matplotlib

Explorar datos con Python: visualizar datos En este cuaderno, aplicaremos técnicas básicas para analizar datos con estadísticas básicas y visualizar mediante gráficos.

Cargando nuestros datos Antes de empezar, carguemos los mismos datos sobre las horas de estudio que analizamos en el cuaderno anterior. También volveremos a calcular quién pasó de la misma manera que la última vez. Ejecute el código en la celda de abajo haciendo clic en el botón ► Ejecutar para ver los datos.

```
import pandas as pd

# Load data from a text file
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-tc
df_students = pd.read_csv('grades.csv',delimiter=',',header='infer')

# Remove any rows with missing data
df_students = df_students.dropna(axis=0, how='any')

# Calculate who passed, assuming '60' is the grade needed to pass
passes  = pd.Series(df_students['Grade'] >= 60)

# Save who passed to the Pandas dataframe
df_students = pd.concat([df_students, passes.rename("Pass")], axis=1)


# Print the result out into this notebook
df_students
```

```
--2021-10-01 21:52:40--  https://raw.githubusercontent.com/MicrosoftDocs/mslearn-
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.13
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.1
HTTP request sent, awaiting response... 200 OK
Length: 322 [text/plain]
Saving to: ‘grades.csv.1’

grades.csv.1        100%[===================>]     322  --.-KB/s    in 0s

2021-10-01 21:52:40 (9.44 MB/s) - ‘grades.csv.1’ saved [322/322]
```

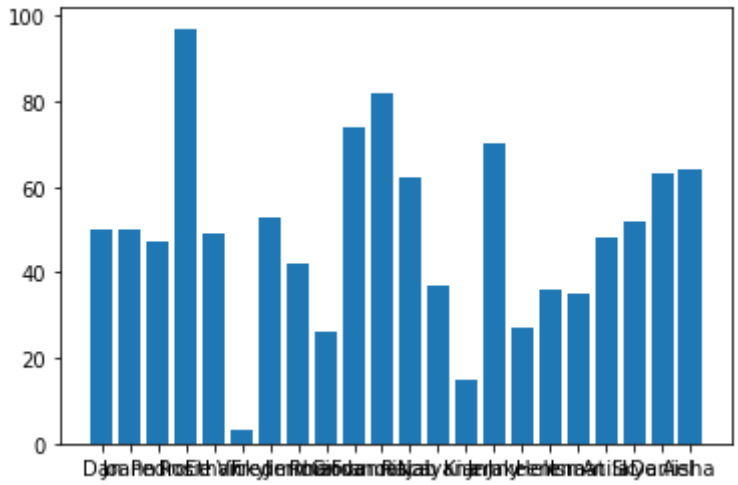
	Name	StudyHours	Grade	Pass
0	Dan	10.00	50.0	False
1	Joann	11.50	50.0	False
2	Pedro	9.00	47.0	False
3	Rosie	16.00	97.0	True
4	Ethan	9.25	49.0	False
5	Vicky	1.00	3.0	False
6	Frederic	11.50	53.0	False
7	Jimmie	9.00	42.0	False
8	Rhonda	8.50	26.0	False
9	Giovanni	14.50	74.0	True
10	Francesca	15.50	82.0	True
11	Rajab	13.75	62.0	True
12	Naiyana	9.00	37.0	False
13	Kian	8.00	15.0	False
14	Jenny	15.50	70.0	True
15	Jakeem	8.00	27.0	False
16	Helena	9.00	36.0	False

Visualización de datos con Matplotlib Los DataFrames brindan una excelente manera de explorar y analizar datos tabulares, pero a veces una imagen vale más que mil filas y columnas. La biblioteca Matplotlib proporciona la base para trazar visualizaciones de datos que pueden mejorar en gran medida su capacidad para analizar los datos.

Comencemos con una gráfica de barras simple que muestra la calificación de cada estudiante.

```
# Ensure plots are displayed inline in the notebook
%matplotlib inline

from matplotlib import pyplot as plt

# Create a bar plot of name vs grade
plt.bar(x=df_students.Name, height=df_students.Grade)
```

```
 8
 9 # Display the plot
10 plt.show()
```

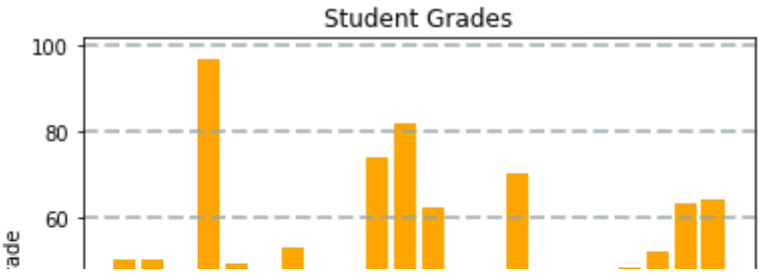


Bueno, eso funcionó; pero el gráfico podría necesitar algunas mejoras para aclarar lo que estamos viendo.

Tenga en cuenta que utilizó la clase pyplot de Matplotlib para trazar el gráfico. Esta clase proporciona un montón de formas de mejorar los elementos visuales de la trama. Por ejemplo, el siguiente código:

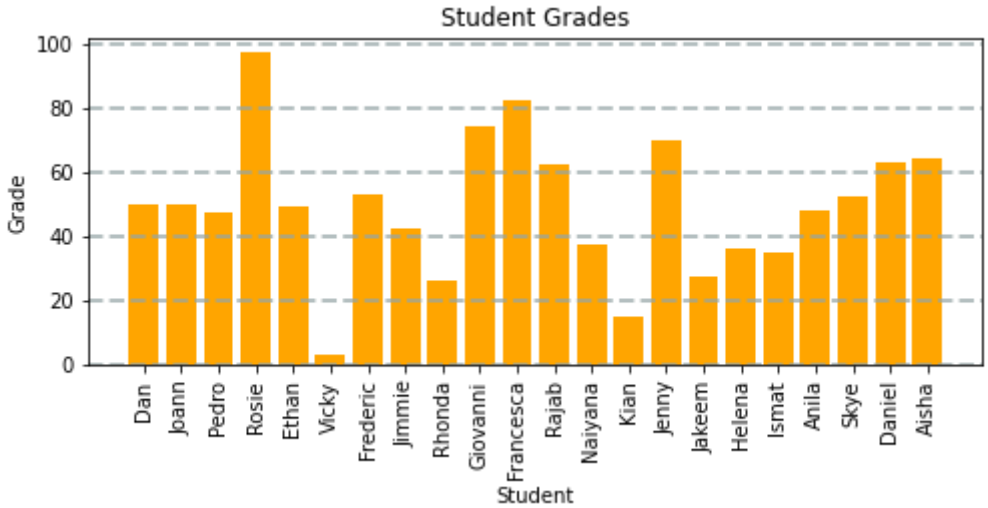
Especifica el color del gráfico de barras. Agrega un título al gráfico (para que sepamos lo que representa) Agrega etiquetas a X e Y (para que sepamos qué eje muestra qué datos) Agrega una cuadrícula (para facilitar la determinación

```
 1 # Create a bar plot of name vs grade
 2 plt.bar(x=df_students.Name, height=df_students.Grade, color='orange')
 3
 4 # Customize the chart
 5 plt.title('Student Grades')
 6 plt.xlabel('Student')
 7 plt.ylabel('Grade')
 8 plt.grid(color='#95a5a6', linestyle='--', linewidth=2, axis='y', alpha=0.7)
 9 plt.xticks(rotation=90)
10
11 # Display the plot
12 plt.show()
```



Una trama está técnicamente contenida con una figura . En los ejemplos anteriores, la figura se creó implícitamente para usted; pero puedes crearlo explícitamente. Por ejemplo, el siguiente código crea una figura con un tamaño específico.

```
# Create a Figure
fig = plt.figure(figsize=(8,3))

# Create a bar plot of name vs grade
plt.bar(x=df_students.Name, height=df_students.Grade, color='orange')

# Customize the chart
plt.title('Student Grades')
plt.xlabel('Student')
plt.ylabel('Grade')
plt.grid(color='#95a5a6', linestyle='--', linewidth=2, axis='y', alpha=0.7)
plt.xticks(rotation=90)

# Show the figure
plt.show()
```

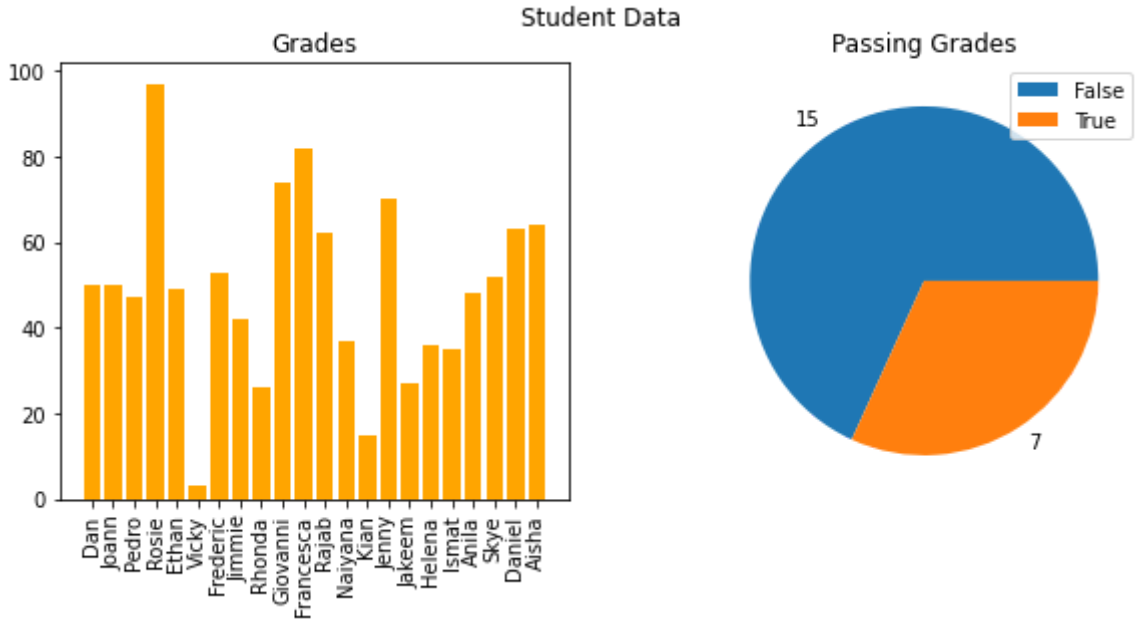


Una figura puede contener múltiples subtramas, cada una en su propio eje .

Por ejemplo, el siguiente código crea una figura con dos subtramas: una es un gráfico de barras que muestra las calificaciones de los estudiantes y el otro es un gráfico circular que compara el número de calificaciones aprobatorias con las no aprobadas.

```
# Create a figure for 2 subplots (1 row, 2 columns)
fig, ax = plt.subplots(1, 2, figsize = (10,4))
```

```
# Create a bar plot of name vs grade on the first axis
ax[0].bar(x=df_students.Name, height=df_students.Grade, color='orange')
ax[0].set_title('Grades')
ax[0].set_xticklabels(df_students.Name, rotation=90)

# Create a pie chart of pass counts on the second axis
pass_counts = df_students['Pass'].value_counts()
ax[1].pie(pass_counts, labels=pass_counts)
ax[1].set_title('Passing Grades')
ax[1].legend(pass_counts.keys().tolist())

# Add a title to the Figure
fig.suptitle('Student Data')

# Show the figure
fig.show()
```



Saltar al contenido principal

Aprender Navegación global Docs Learn Browse Explore and analyze data with Python Exercise - Visualize data with Matplotlib

Unit 5 of 9 Anterior próximo Ejercicio: visualizar datos con Matplotlib Terminado 100 XP 10 minutos Este módulo requiere una caja de arena para completarse. Ha utilizado 1 de 10 sandboxes por hoy. Mañana habrá más sandboxes disponibles.

Activar sandbox

Show variables active in Jupyter kernel

Edición Celda de descuento seleccionada en la posición 11 Explorar datos con Python: visualizar datos En este cuaderno, aplicaremos técnicas básicas para analizar datos con estadísticas básicas y visualizar mediante gráficos.

Cargando nuestros datos Antes de empezar, carguemos los mismos datos sobre las horas de estudio que analizamos en el cuaderno anterior. También volveremos a calcular quién pasó de la misma manera que la última vez. Ejecute el código en la celda de abajo haciendo clic en el botón ► Ejecutar para ver los datos.

[] importar · pandas · como · pd# · Cargar · datos · desde · un · archivo · de · texto! wget · https : //raw.githubusercontent.com/MicrosoftDocs/mslearn- introducción-al-aprendizaje-automático / main / Data / ml-basi cs / grade.csvdf_students · = · pd.read_csv ('grados.csv' , delimitador = ',' , encabezado = 'inferir')# · Eliminar · cualquier · fila · con · datos · faltantesdf_students · = · df_students.dropna (eje = 0 , · cómo = 'cualquiera')# · Calcular · quién · aprobó, · asumiendo · '60' · es · la · calificación · necesaria · para · aprobarpasa ·· = · pd.Series (df_students ['Grade'] · > = · 60)# · Guardar · que · pasado · a · la · pandas · trama de datosdf_students · = · pd.concat ([df_students , · pasa.rename ("Aprobado")], · eje = 1)# · Imprimir · la · resultado · cabo · en · este · portátildf_students ·

Presione shift + enter para ejecutar celdas Visualización de datos con Matplotlib Los DataFrames brindan una excelente manera de explorar y analizar datos tabulares, pero a veces una imagen vale más que mil filas y columnas. La biblioteca Matplotlib proporciona la base para trazar visualizaciones de datos que pueden mejorar en gran medida su capacidad para analizar los datos.

Comencemos con una gráfica de barras simple que muestra la calificación de cada estudiante.

[]

· Asegúrese · de · que · los · gráficos · se · muestran · en línea · en · el · cuaderno% matplotlib · en líneade · matplotlib · importar · pyplot · como · plt# · Crear · una · barra · gráfica · de · nombre · vs · gradoplt.bar (x = df_students.Name , · altura = df_students.Grade)# · Mostrar · la · tramaplt.show ()

Presione shift + enter para ejecutar celdas Bueno, eso funcionó; pero el gráfico podría necesitar algunas mejoras para aclarar lo que estamos viendo.

Tenga en cuenta que utilizó la clase pyplot de Matplotlib para trazar el gráfico. Esta clase proporciona un montón de formas de mejorar los elementos visuales de la trama. Por ejemplo, el siguiente código:

Especifica el color del gráfico de barras. Agrega un título al gráfico (para que sepamos lo que representa) Agrega etiquetas a X e Y (para que sepamos qué eje muestra qué datos) Agrega una cuadrícula (para facilitar la determinación de los valores de las barras) Gira los marcadores X (para que podamos leerlos) []

```
· Crear · una · barra · gráfica · de · nombre · vs · gradoplt.bar ( x = df_students.Name , · altura = df_students.Grade , · color = 'naranja' )# · Personalizar · el · gráficoplt.title ( 'Estudiante · Calificaciones' )plt.xlabel ( 'Estudiante' )plt.ylabel ( 'Grado' )plt.grid ( color = '# 95a5a6' , · estilo de línea = '-' , · ancho de línea = 2 , · eje = 'y' , · alfa = 0,7 )plt.xticks ( rotación = 90 )# · Mostrar · la · tramaplt.show ()
```

Presione shift + enter para ejecutar celdas Una trama está técnicamente contenida con una figura . En los ejemplos anteriores, la figura se creó implícitamente para usted; pero puedes crearlo explícitamente. Por ejemplo, el siguiente código crea una figura con un tamaño específico.

Ejecutar celda[]

```
· Crear · una · figurafig · = · plt.figure ( figsize = ( 8 , 3 ))# · Crear · una · barra · gráfica · de · nombre · vs · gradoplt.bar ( x = df_students.Name , · altura = df_students.Grade , · color = 'naranja' )# · Personalizar · el · gráficoplt.title ( 'Estudiante · Calificaciones' )plt.xlabel ( 'Estudiante' )plt.ylabel ( 'Grado' )plt.grid ( color = '# 95a5a6' , · estilo de línea = '-' , · ancho de línea = 2 , · eje = 'y' , · alfa = 0,7 )plt.xticks ( rotación = 90 )# · Mostrar · la · figuraplt.show ()
```

Presione shift + enter para ejecutar celdas Una figura puede contener múltiples subtramas, cada una en su propio eje .

Por ejemplo, el siguiente código crea una figura con dos subtramas: una es un gráfico de barras que muestra las calificaciones de los estudiantes y el otro es un gráfico circular que compara el número de calificaciones aprobatorias con las no aprobadas.

[]

```
· Crear · una · figura · para · 2 · subparcelas · (1 · fila, · 2 · columnas)fig , · ax · = · plt.subplots ( 1 , · 2 , · figsize · = · ( 10 , 4 ))# · Crear · una · barra · gráfica · de · nombre · vs · calificación · en · el · primer · ejeax [ 0 ] .bar ( x = df_students.Name , · altura = df_students.Grade , · color = 'naranja' )ax [ 0 ] .set_title ( 'Calificaciones' )ax [ 0 ] .set_xticklabels ( df_students.Name , · rotación = 90 )# · Crear · una · pastel · diagrama · de · paso · conteos · sobre · el · segundo · ejepass_counts · = · df_students [ 'Pass' ] .value_counts ()ax [ 1 ] .pie ( pass_counts , · etiquetas = pass_counts )ax [ 1 ] .set_title ( ' Aprobado · Calificaciones' )ax [ 1 ] .legend ( pass_counts.keys () .tolist ())# · Agregar · un · título · a · la · Figurafig.suptitle ( 'Estudiante · Datos' )# · Mostrar · la · figurafig.show ()
```

Presione shift + enter para ejecutar celdas Hasta ahora, ha utilizado métodos del objeto Matplotlib.pyplot para trazar gráficos. Sin embargo, Matplotlib es tan fundamental para los gráficos en Python que muchos paquetes, incluido Pandas, proporcionan métodos que abstraen las funciones subyacentes de Matplotlib y simplifican el trazado. Por ejemplo, el DataFrame proporciona sus propios métodos para trazar datos, como se muestra en el siguiente ejemplo para trazar un gráfico de barras de horas de estudio.

Código

Reducción [] df_students.plot.bar (x = 'Nombre' , y = 'StudyHours' , color = 'teal' , figsize = (6 , 4))

Presione shift + enter para ejecutar celdas Empezando con el análisis estadístico Ahora que sabe cómo usar Python para manipular y visualizar datos, puede comenzar a analizarlos.

Gran parte de la ciencia de datos tiene sus raíces en las estadísticas , por lo que exploraremos algunas técnicas estadísticas básicas.

Nota : esto esno tiene la intención de enseñarle estadísticas, es un tema demasiado grande para este cuaderno. Sin embargo, le presentará algunos conceptos y técnicas estadísticos que los científicos de datos utilizan mientras exploran datos en preparación para el modelado de aprendizaje automático. Estadística descriptiva y distribución de datos Al examinar una variable (por ejemplo, una muestra de las calificaciones de los estudiantes), los científicos de datos están particularmente interesados en su distribución (en otras palabras, cómo se distribuyen los diferentes valores de calificación en la muestra). El punto de partida de esta exploración suele ser visualizar los datos como un histograma y ver con qué frecuencia se produce cada valor de la variable.

[]

Obtener la variable para examinar

var_data = df_students ['Grade']

Crea una figura

fig = plt.figure (figsize = (10,4))

Plot a histogram

plt.hist(var_data)

Add titles and labels

plt.title('Data Distribution') plt.xlabel('Value') plt.ylabel('Frequency')

Show the figure

fig.show() The histogram for grades is a symmetric shape, where the most frequently occurring grades tend to be in the middle of the range (around 50), with fewer grades at the extreme ends of the scale.

Measures of central tendency To understand the distribution better, we can examine so-called measures of central tendency; which is a fancy way of describing statistics that represent the

"middle" of the data. The goal of this is to try to find a "typical" value. Common ways to define the middle of the data include:

The mean: A simple average based on adding together all of the values in the sample set, and then dividing the total by the number of samples. The median: The value in the middle of the range of all of the sample values. The mode: The most commonly occuring value in the sample set*. Let's calculate these values, along with the minimum and maximum values for comparison, and show them on the histogram.

*Of course, in some sample sets , there may be a tie for the most common value - in which case the dataset is described as bimodal or even multimodal. [ ]

Get the variable to examine

var = df_students['Grade']

Get statistics

min_val = var.min() max_val = var.max() mean_val = var.mean() med_val = var.median() mod_val = var.mode()[0]

print('Minimum:{:.2f}\nMean:{:.2f}\nMedian:{:.2f}\nMode:{:.2f}\nMaximum: {:.2f}\n'.format(min_val, mean_val, med_val, mod_val, max_val))

Create a Figure

fig = plt.figure(figsize=(10,4))

Plot a histogram

plt.hist(var)

Add lines for the statistics

plt.axvline(x=min_val, color = 'gray', linestyle='dashed', linewidth = 2) plt.axvline(x=mean_val, color = 'cyan', linestyle='dashed', linewidth = 2) plt.axvline(x=med_val, color = 'red', linestyle='dashed', linewidth = 2) plt.axvline(x=mod_val, color = 'yellow', linestyle='dashed', linewidth = 2) plt.axvline(x=max_val, color = 'gray', linestyle='dashed', linewidth = 2)

Add titles and labels

plt.title('Data Distribution') plt.xlabel('Value') plt.ylabel('Frequency')

Show the figure

fig.show() For the grade data, the mean, median, and mode all seem to be more or less in the middle of the minimum and maximum, at around 50.

Another way to visualize the distribution of a variable is to use a box plot (sometimes called a box-and-whiskers plot). Let's create one for the grade data.

[ ]

Get the variable to examine

var = df_students['Grade']

Create a Figure

fig = plt.figure(figsize=(10,4))

Plot a histogram

plt.boxplot(var)

Add titles and labels

plt.title('Data Distribution')

Show the figure

fig.show() The box plot shows the distribution of the grade values in a different format to the histogram. The box part of the plot shows where the inner two quartiles of the data reside - so in this case, half of the grades are between approximately 36 and 63. The whiskers extending from the box show the outer two quartiles; so the other half of the grades in this case are between 0 and 36 or 63 and 100. The line in the box indicates the median value.

For learning, it can be useful to combine histograms and box plots, with the box plot's orientation changed to align it with the histogram (in some ways, it can be helpful to think of the histogram as a "front elevation" view of the distribution, and the box plot as a "plan" view of the distribution from above.)

[ ]

Create a function that we can re-use

def show_distribution(var_data): from matplotlib import pyplot as plt

```
# Get statistics
min_val = var_data.min()
max_val = var_data.max()
```

```
    mean_val = var_data.mean()
    med_val = var_data.median()
    mod_val = var_data.mode()[0]

    print('Minimum:{:.2f}\nMean:{:.2f}\nMedian:{:.2f}\nMode:{:.2f}\nMaximum:{:.2f}\n'.format(min_val,
                                                                                            mean_val,
                                                                                            med_val,
                                                                                            mod_val,
                                                                                            max_val))

    # Create a figure for 2 subplots (2 rows, 1 column)
    fig, ax = plt.subplots(2, 1, figsize = (10,4))

    # Plot the histogram
    ax[0].hist(var_data)
    ax[0].set_ylabel('Frequency')

    # Add lines for the mean, median, and mode
    ax[0].axvline(x=min_val, color = 'gray', linestyle='dashed', linewidth = 2)
    ax[0].axvline(x=mean_val, color = 'cyan', linestyle='dashed', linewidth = 2)
    ax[0].axvline(x=med_val, color = 'red', linestyle='dashed', linewidth = 2)
    ax[0].axvline(x=mod_val, color = 'yellow', linestyle='dashed', linewidth = 2)
    ax[0].axvline(x=max_val, color = 'gray', linestyle='dashed', linewidth = 2)

    # Plot the boxplot
    ax[1].boxplot(var_data, vert=False)
    ax[1].set_xlabel('Value')

    # Add a title to the Figure
    fig.suptitle('Data Distribution')

    # Show the figure
    fig.show()
```

Get the variable to examine

col = df_students['Grade']

Call the function

show_distribution(col) All of the measurements of central tendency are right in the middle of the data distribution, which is symmetric with values becoming progressively lower in both directions from the middle.

To explore this distribution in more detail, you need to understand that statistics is fundamentally about taking samples of data and using probability functions to extrapolate information about the full population of data.

What does this mean? Samples refer to the data we have on hand - such as information about these 22 students' study habits and grades. The population refers to all possible data we could collect - such as every student's grades and study habits across every educational institution throughout the history of time. Usually we're interested in the population but it's simply not practical to collect all of that data. Instead, we need to try estimate what the population is like from the small amount of data (samples) that we have.

If we have enough samples, we can calculate something called a probability density function, which estimates the distribution of grades for the full population.

The Pandas DataFrame class provides a helpful plot function to show this density.

[ ] def show_density(var_data): from matplotlib import pyplot as plt

```
fig = plt.figure(figsize=(10,4))

# Plot density
var_data.plot.density()

# Add titles and labels
plt.title('Data Density')

# Show the mean, median, and mode
plt.axvline(x=var_data.mean(), color = 'cyan', linestyle='dashed', linewidth = 2)
plt.axvline(x=var_data.median(), color = 'red', linestyle='dashed', linewidth = 2)
plt.axvline(x=var_data.mode()[0], color = 'yellow', linestyle='dashed', linewidth = 2)

# Show the figure
plt.show()
```

Get the density of Grade

col = df_students['Grade'] show_density(col) As expected from the histogram of the sample, the density shows the characteristic "bell curve" of what statisticians call a normal distribution with the mean and mode at the center and symmetric tails.

Summary Well done! There were a number of new concepts in here, so let's summarise.

Here we have:

Made graphs with matplotlib Seen how to customise these graphs Calculated basic statistics, such as medians Looked at the spread of data using box plots and histograms Learned about samples vs populations Estimated what the population of graphse might look like from a sample

of grades. In our next notebook we will look at spotting unusual data, and finding relationships between data.

Further Reading To learn more about the Python packages you explored in this notebook, see the following documentation:

NumPy Pandas Matplotlib Celda de rebaja vacía. Haga doble clic o presione enter para agregar contenido.

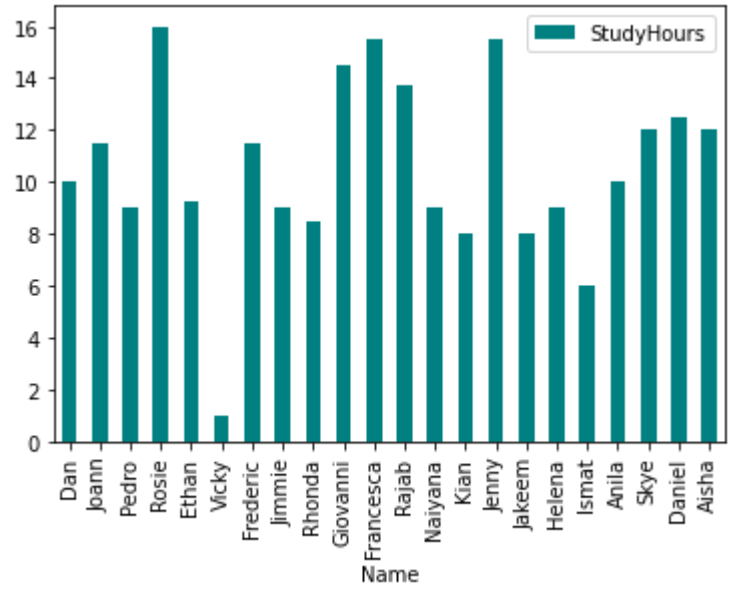
¿Necesitas ayuda? Consulte nuestra guía de solución de problemas o proporcione comentarios específicos informando un problema .

Inglés Estados Unidos)

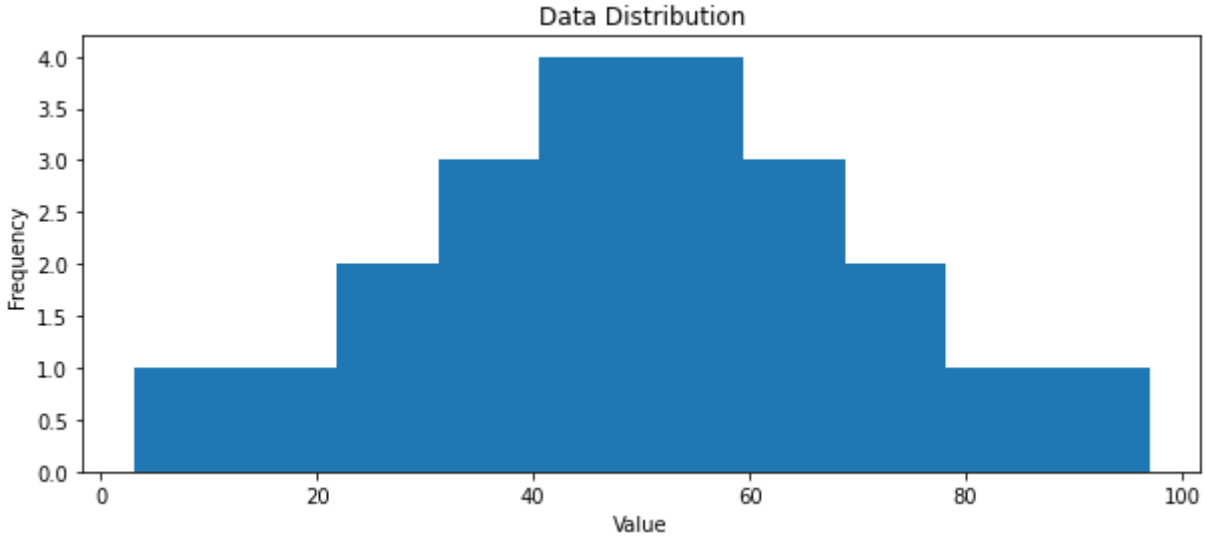
Tema Documentos de la versión anterior Blog Contribuir Privacidad y cookies Condiciones de uso Marcas comerciales © Microsoft 2021

```
1 _students.plot.bar(x='Name', y='StudyHours', color='teal', figsize=(6,4))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f13cb64bc50>
```

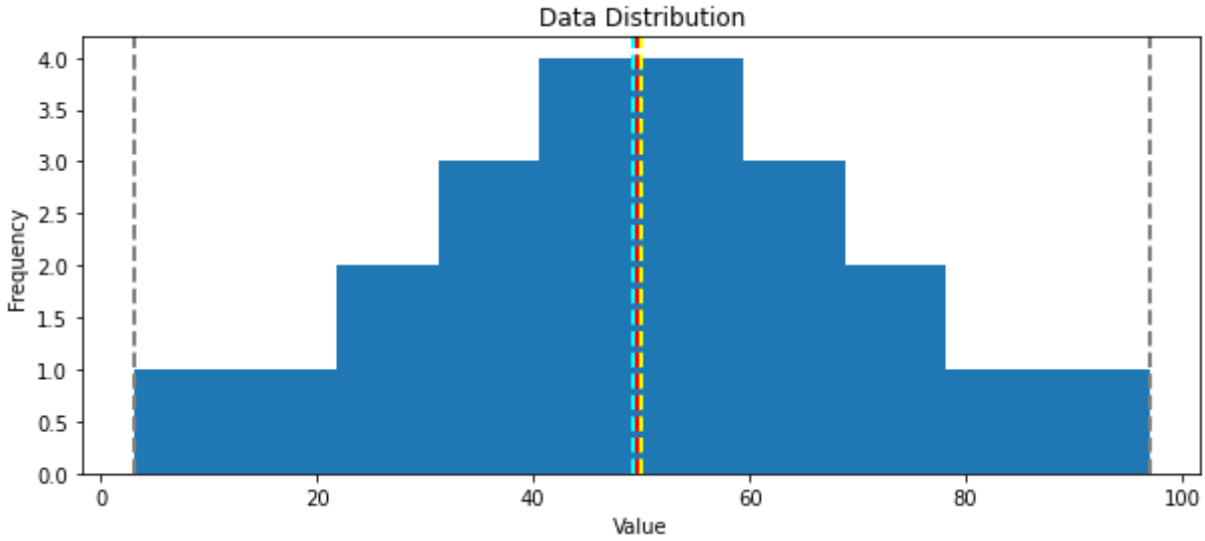



```
 1 # Get the variable to examine
 2 var_data = df_students['Grade']
 3
 4 # Create a Figure
 5 fig = plt.figure(figsize=(10,4))
 6
 7 # Plot a histogram
 8 plt.hist(var_data)
 9
10 # Add titles and labels
11 plt.title('Data Distribution')
12 plt.xlabel('Value')
13 plt.ylabel('Frequency')
14
15 # Show the figure
16 fig.show()
```




```
# Get the variable to examine
var = df_students['Grade']

# Get statistics
min_val = var.min()
max_val = var.max()
mean_val = var.mean()
med_val = var.median()
mod_val = var.mode()[0]

print('Minimum:{:.2f}\nMean:{:.2f}\nMedian:{:.2f}\nMode:{:.2f}\nMaximum:{:.2f



# Create a Figure
fig = plt.figure(figsize=(10,4))

# Plot a histogram
plt.hist(var)

# Add lines for the statistics
plt.axvline(x=min_val, color = 'gray', linestyle='dashed', linewidth = 2)
plt.axvline(x=mean_val, color = 'cyan', linestyle='dashed', linewidth = 2)
plt.axvline(x=med_val, color = 'red', linestyle='dashed', linewidth = 2)
plt.axvline(x=mod_val, color = 'yellow', linestyle='dashed', linewidth = 2)
plt.axvline(x=max_val, color = 'gray', linestyle='dashed', linewidth = 2)

# Add titles and labels
plt.title('Data Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Show the figure
fig.show()
```

```
Minimum:3.00
Mean:49.18
Median:49.50
Mode:50.00
Maximum:97.00
```

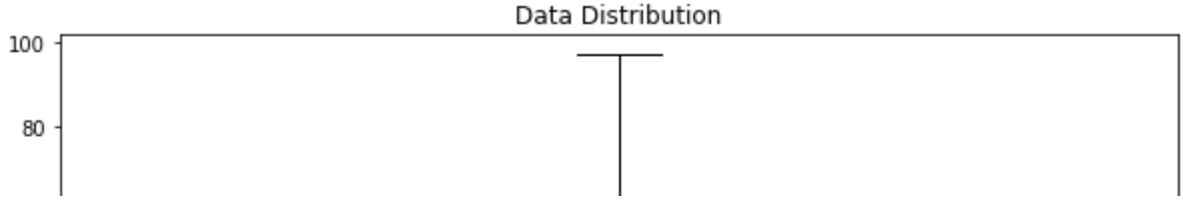


Para los datos de calificaciones, la media, la mediana y la moda parecen estar más o menos en el medio del mínimo y el máximo, alrededor de 50.

Otra forma de visualizar la distribución de una variable es usar un diagrama de caja (a veces llamado diagrama de caja y bigotes). Creemos uno para los datos de calificaciones.

```
1
```

```
# Get the variable to examine
var = df_students['Grade']

# Create a Figure
fig = plt.figure(figsize=(10,4))

# Plot a histogram
plt.boxplot(var)

# Add titles and labels
plt.title('Data Distribution')

# Show the figure
fig.show()
```



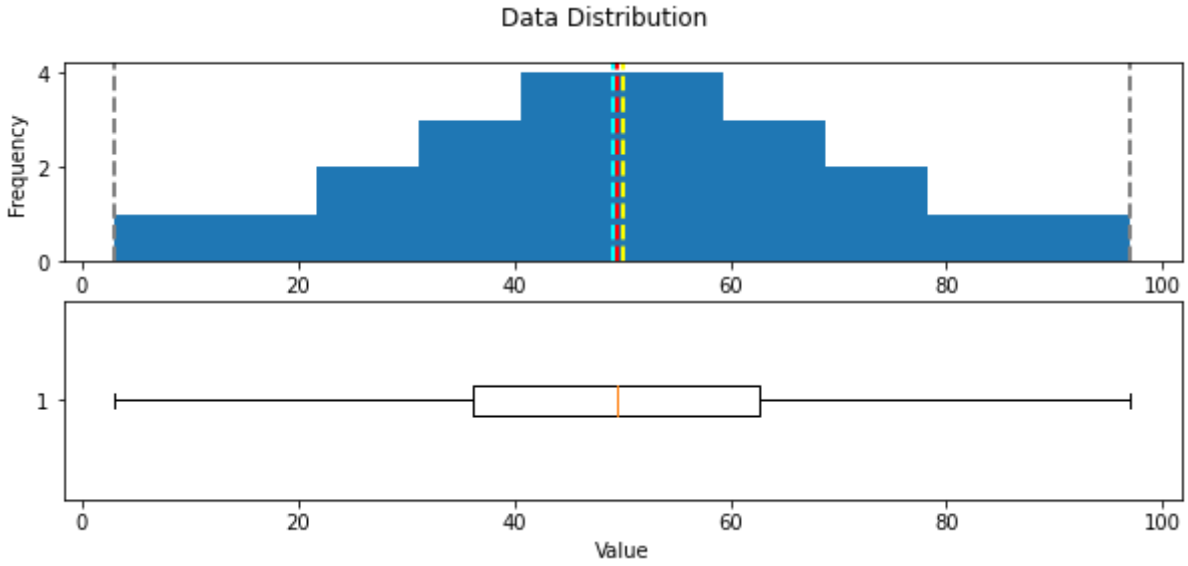
El diagrama de caja muestra la distribución de los valores de calificación en un formato diferente al histograma. La parte del cuadro del gráfico muestra dónde residen los dos cuartiles internos de los datos, por lo que en este caso, la mitad de las calificaciones están entre aproximadamente 36 y 63. Los bigotes que se extienden desde el cuadro muestran los dos cuartiles externos; por lo que la otra mitad de las calificaciones en este caso están entre 0 y 36 o 63 y 100. La línea en el cuadro indica el valor mediano .

Para aprender, puede ser útil combinar histogramas y diagramas de caja, con la orientación del diagrama de caja cambiada para alinearla con el histograma (de alguna manera, puede ser útil pensar en el histograma como una vista de "elevación frontal" de la distribución y el diagrama de caja como una vista en "planta" de la distribución desde arriba).

```
# Create a function that we can re-use
def show_distribution(var_data):
    from matplotlib import pyplot as plt

    # Get statistics
    min_val = var_data.min()
    max_val = var_data.max()
    mean_val = var_data.mean()
    med_val = var_data.median()
    mod_val = var_data.mode()[0]

    print('Minimum:{:.2f}\nMean:{:.2f}\nMedian:{:.2f}\nMode:{:.2f}\nMaximum:{



    # Create a figure for 2 subplots (2 rows, 1 column)
    fig, ax = plt.subplots(2, 1, figsize = (10,4))

    # Plot the histogram
    ax[0].hist(var_data)
    ax[0].set_ylabel('Frequency')

    # Add lines for the mean, median, and mode
    ax[0].axvline(x=min_val, color = 'gray', linestyle='dashed', linewidth =
    ax[0].axvline(x=mean_val, color = 'cyan', linestyle='dashed', linewidth =
    ax[0].axvline(x=med_val, color = 'red', linestyle='dashed', linewidth = 2
    ax[0].axvline(x=mod_val, color = 'yellow', linestyle='dashed', linewidth
    ax[0].axvline(x=max_val, color = 'gray', linestyle='dashed', linewidth =

    # Plot the boxplot
    ax[1].boxplot(var_data, vert=False)
    ax[1].set_xlabel('Value')
```

```
    ax[1].set_xlabel('Value')

    # Add a title to the Figure
    fig.suptitle('Data Distribution')

    # Show the figure
    fig.show()

# Get the variable to examine
col = df_students['Grade']
# Call the function
show_distribution(col)
```

```
Minimum:3.00
Mean:49.18
Median:49.50
Mode:50.00
Maximum:97.00
```



Todas las medidas de tendencia central están justo en el medio de la distribución de datos, que es simétrica con valores que se vuelven progresivamente más bajos en ambas direcciones desde el medio.

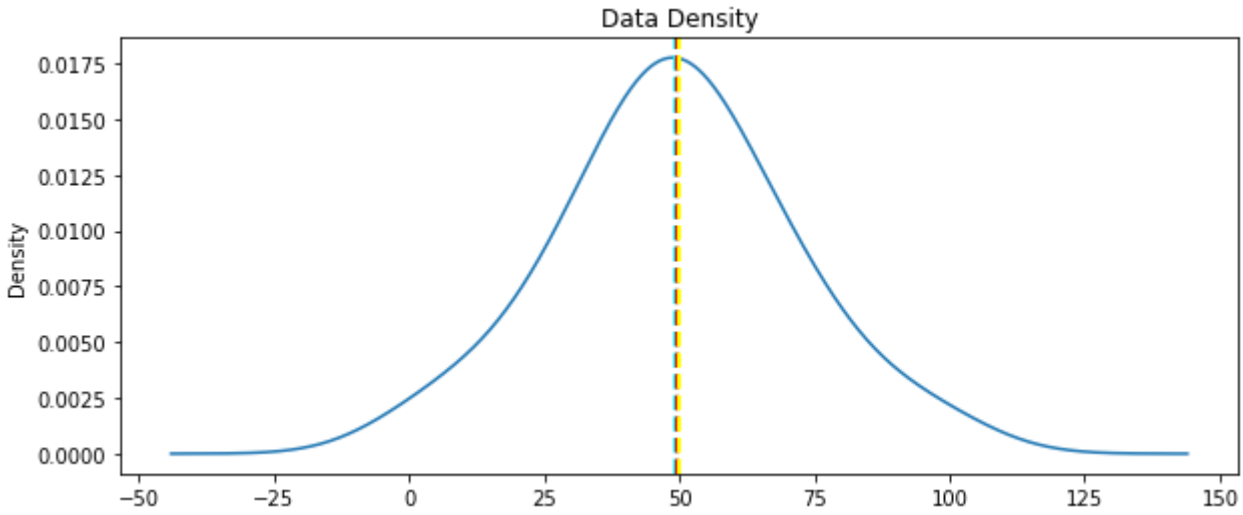
Para explorar esta distribución con más detalle, debe comprender que la estadística se trata fundamentalmente de tomar muestras de datos y utilizar funciones de probabilidad para extrapolar información sobre la población completa de datos.

¿Qué significa esto? Las muestras se refieren a los datos que tenemos a mano, como información sobre los hábitos de estudio y las calificaciones de estos 22 estudiantes. La población se refiere a todos los datos posibles que pudimos recopilar, como las calificaciones y los hábitos de estudio de cada estudiante en todas las instituciones educativas a lo largo de la historia. Por lo general, estamos interesados en la población, pero simplemente no es práctico recopilar todos esos datos. En cambio, debemos intentar estimar cómo es la población a partir de la pequeña cantidad de datos (muestras) que tenemos.

Si tenemos suficientes muestras, podemos calcular algo llamado función de densidad de probabilidad , que estima la distribución de calificaciones para la población completa.

La clase Pandas DataFrame proporciona una función de trazado útil para mostrar esta densidad.

```
def show_density(var_data):
    from matplotlib import pyplot as plt

    fig = plt.figure(figsize=(10,4))

    # Plot density
    var_data.plot.density()

    # Add titles and labels
    plt.title('Data Density')

    # Show the mean, median, and mode
    plt.axvline(x=var_data.mean(), color = 'cyan', linestyle='dashed', linewi
    plt.axvline(x=var_data.median(), color = 'red', linestyle='dashed', linew
    plt.axvline(x=var_data.mode()[0], color = 'yellow', linestyle='dashed', l

    # Show the figure
    plt.show()

# Get the density of Grade
col = df_students['Grade']
show_density(col)
```



Examinar datos del mundo real Terminado 100 XP 3 minutos Los datos presentados en el material educativo suelen ser notablemente perfectos y están diseñados para mostrar a los estudiantes cómo encontrar relaciones claras entre las variables. Los datos del 'mundo real' son un poco menos simples.

Debido a la complejidad de los datos del 'mundo real', los datos sin procesar deben inspeccionarse para detectar problemas antes de usarlos.

Como tal, la mejor práctica es inspeccionar los datos sin procesar y procesarlos antes de usarlos, lo que reduce los errores o problemas, generalmente eliminando puntos de datos erróneos o modificando los datos en una forma más útil.

Explorando datos con Python: datos del mundo real La última vez, analizamos las calificaciones de los datos de nuestros estudiantes y lo investigamos visualmente con histogramas y diagramas de caja. Ahora veremos casos más complejos, describiremos los datos de manera más completa y discutiremos cómo hacer comparaciones básicas entre los datos.

Distribuciones de datos del mundo real La última vez, analizamos las calificaciones de los datos de nuestros estudiantes y estimamos a partir de esta muestra cómo se vería la población completa de calificaciones. Solo para actualizar, echemos un vistazo a estos datos nuevamente.

Ejecute el siguiente código para imprimir los datos y hacer un histograma + diagrama de caja que muestre las calificaciones de nuestra muestra de estudiantes.

```
import pandas as pd
from matplotlib import pyplot as plt

# Load data from a text file
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to
df_students = pd.read_csv('grades.csv',delimiter=',',header='infer')

# Remove any rows with missing data
df_students = df_students.dropna(axis=0, how='any')

# Calculate who passed, assuming '60' is the grade needed to pass
passes  = pd.Series(df_students['Grade'] >= 60)

# Save who passed to the Pandas dataframe
df_students = pd.concat([df_students, passes.rename("Pass")], axis=1)


# Print the result out into this notebook
print(df_students)


# Create a function that we can re-use
def show_distribution(var_data):
    '''
    This function will make a distribution (graph) and display it
    '''

    # Get statistics
    min_val = var_data.min()
    max_val = var_data.max()
    mean_val = var_data.mean()
    med_val = var_data.median()
    mod_val = var_data.mode()[0]

    print('Minimum:{:.2f}\nMean:{:.2f}\nMedian:{:.2f}\nMode:{:.2f}\nMaximum:{
```

```
36
37
38
39
40
41     # Create a figure for 2 subplots (2 rows, 1 column)
42     fig, ax = plt.subplots(2, 1, figsize = (10,4))
43
44     # Plot the histogram
45     ax[0].hist(var_data)
46     ax[0].set_ylabel('Frequency')
47
48     # Add lines for the mean, median, and mode
49     ax[0].axvline(x=min_val, color = 'gray', linestyle='dashed', linewidth =
50     ax[0].axvline(x=mean_val, color = 'cyan', linestyle='dashed', linewidth =
51     ax[0].axvline(x=med_val, color = 'red', linestyle='dashed', linewidth = 2
52     ax[0].axvline(x=mod_val, color = 'yellow', linestyle='dashed', linewidth
53     ax[0].axvline(x=max_val, color = 'gray', linestyle='dashed', linewidth =
54
55     # Plot the boxplot
56     ax[1].boxplot(var_data, vert=False)
57     ax[1].set_xlabel('Value')
58
59     # Add a title to the Figure
60     fig.suptitle('Data Distribution')
61
62     # Show the figure
63     fig.show()
64
65
66 show_distribution(df_students['Grade'])
```

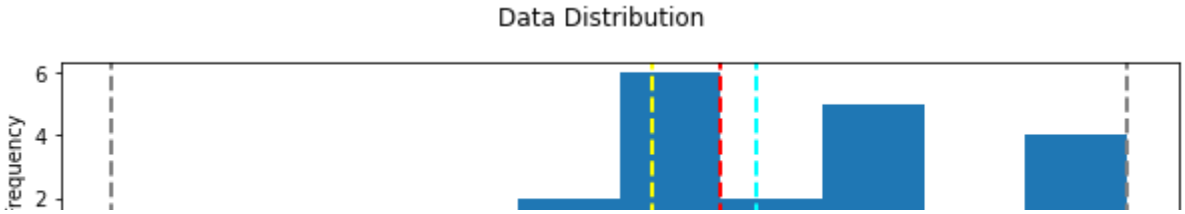
```
--2021-10-01 22:55:58--  https://raw.githubusercontent.com/MicrosoftDocs/mslearn-
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.13
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.1
HTTP request sent, awaiting response... 200 OK
Length: 322 [text/plain]
Saving to: ‘grades.csv.2’

grades.csv.2        100%[===================>]     322  --.-KB/s    in 0s

2021-10-01 22:55:58 (22.6 MB/s) - ‘grades.csv.2’ saved [322/322]

         Name  StudyHours  Grade   Pass
0         Dan       10.00   50.0  False
1       Joann       11.50   50.0  False
2       Pedro        9.00   47.0  False
3       Rosie       16.00   97.0   True
4       Ethan        9.25   49.0  False
5       Vicky        1.00    3.0  False
6    Frederic       11.50   53.0  False
7      Jimmie        9.00   42.0  False
8      Rhonda        8.50   26.0  False
9    Giovanni       14.50   74.0   True
10  Francesca       15.50   82.0   True
11      Rajab       13.75   62.0   True
12    Naiyana        9.00   37.0  False
13       Kian        8.00   15.0  False
14      Jenny       15.50   70.0   True
15     Jakeem        8.00   27.0  False
16     Helena        9.00   36.0  False
17      Ismat        6.00   35.0  False
18      Anila       10.00   48.0  False
19       Skye       12.00   52.0  False
20     Daniel       12.50   63.0   True
21      Aisha       12.00   64.0   True
Minimum:3.00
Mean:49.18
```

Como recordará, nuestros datos tenían la media y la moda en el centro, con datos distribuidos simétricamente desde allí.

Ahora echemos un vistazo a la distribución de los datos de las horas de estudio.

```
# Get the variable to examine
col = df_students['StudyHours']
# Call the function
show_distribution(col)
```

```
Minimum:1.00
Mean:10.52
Median:10.00
Mode:9.00
Maximum:16.00
```



La distribución de los datos del tiempo de estudio es significativamente diferente a la de los grados.

Tenga en cuenta que los bigotes del diagrama de caja solo comienzan alrededor de 6.0, lo que indica que la gran mayoría del primer trimestre de los datos está por encima de este valor. El mínimo está marcado con una o , lo que indica que es estadísticamente un valor atípico , un valor que se encuentra significativamente fuera del rango del resto de la distribución.

Los valores atípicos pueden ocurrir por muchas razones. Tal vez un estudiante tenía la intención de registrar "10" horas de tiempo de estudio, pero ingresó "1" y perdió el "0". ¡O tal vez el estudiante era anormalmente perezoso cuando se trata de estudiar! De cualquier manera, es una anomalía estadística que no representa a un estudiante típico. Veamos cómo se ve la distribución sin él.

```
# Get the variable to examine
# We will only get students who have studied more than one hour
col = df_students[df_students.StudyHours>1]['StudyHours']

# Call the function
show_distribution(col)
```

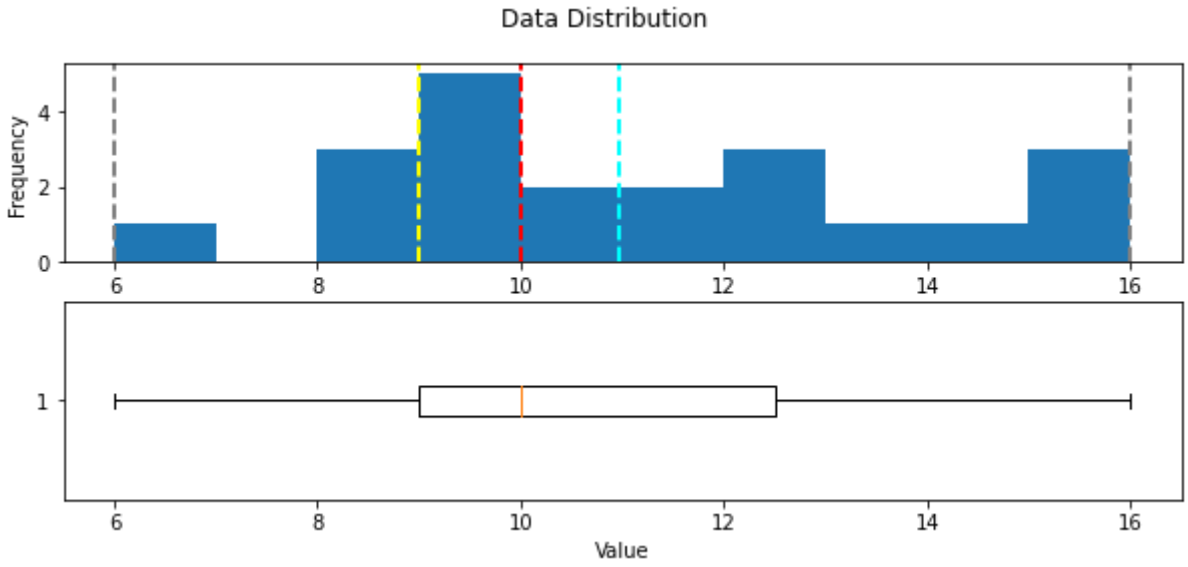
```
Minimum:6.00
Mean:10.98
Median:10.00
```

Para fines de aprendizaje, acabamos de tratar que el valor 1 es un valor atípico y lo excluimos. En el mundo real, sin embargo, sería inusual excluir datos en los extremos sin más justificación cuando el tamaño de nuestra muestra es tan pequeño. Esto se debe a que cuanto menor sea el tamaño de nuestra muestra, más probable es que nuestra muestra sea una mala representación de toda la población (aquí, la población significa calificaciones para todos los estudiantes, no solo para nuestros 22). Por ejemplo, si tomamos una muestra del tiempo de estudio de otros 1000 estudiantes, ¡podríamos encontrar que en realidad es bastante común no estudiar mucho!

Cuando tenemos más datos disponibles, nuestra muestra se vuelve más confiable. Esto hace que sea más fácil considerar valores atípicos como valores que caen por debajo o por encima de los percentiles dentro de los cuales se encuentran la mayoría de los datos. Por ejemplo, el siguiente código usa la función de cuantil de Pandas para excluir observaciones por debajo del percentil 0.01 (el valor por encima del cual reside el 99% de los datos).

```
# calculate the 0.01th percentile
q01 = df_students.StudyHours.quantile(0.01)
# Get the variable to examine
col = df_students[df_students.StudyHours>q01]['StudyHours']
# Call the function
show_distribution(col)
```

```
Minimum:6.00
Mean:10.98
Median:10.00
Mode:9.00
Maximum:16.00
```

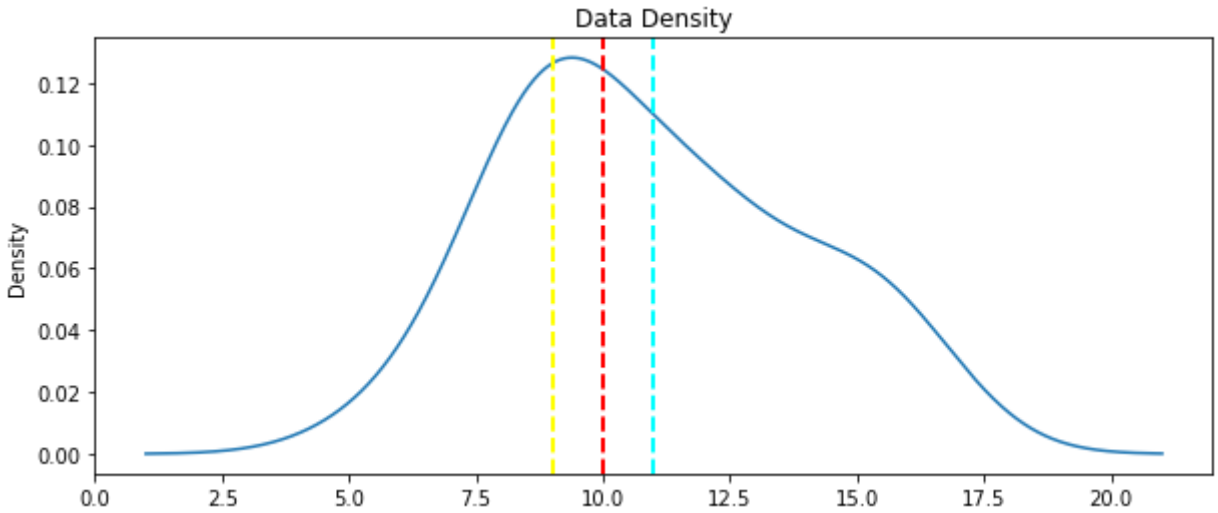


Con los valores atípicos eliminados, el diagrama de caja muestra todos los datos dentro de los cuatro cuartiles. Sin embargo, tenga en cuenta que la distribución no es simétrica como lo es para los datos de calificaciones: hay algunos estudiantes con tiempos de estudio muy altos de

alrededor de 16 horas, pero la mayor parte de los datos está entre 7 y 13 horas; Los pocos valores extremadamente altos empujan la media hacia el extremo superior de la escala.

Veamos la densidad de esta distribución.

```
def show_density(var_data):
    fig = plt.figure(figsize=(10,4))

    # Plot density
    var_data.plot.density()

    # Add titles and labels
    plt.title('Data Density')

    # Show the mean, median, and mode
    plt.axvline(x=var_data.mean(), color = 'cyan', linestyle='dashed', linewi
    plt.axvline(x=var_data.median(), color = 'red', linestyle='dashed', linew
    plt.axvline(x=var_data.mode()[0], color = 'yellow', linestyle='dashed', l

    # Show the figure
    plt.show()

# Get the density of StudyHours
show_density(col)
```



Este tipo de distribución se llama sesgada a la derecha . La masa de los datos está en el lado izquierdo de la distribución, creando una cola larga a la derecha debido a los valores en el extremo superior; que tiran de la media hacia la derecha.

Medidas de varianza Así que ahora tenemos una buena idea de dónde están las distribuciones de datos de la mitad de la calificación y las horas de estudio. Sin embargo, hay otro aspecto de las distribuciones que deberíamos examinar: ¿cuánta variabilidad hay en los datos?

Las estadísticas típicas que miden la variabilidad en los datos incluyen:

Rango : la diferencia entre el máximo y el mínimo. No hay incorporado en función de esto, pero es fácil de calcular utilizando el min y max funciones. Varianza : el promedio de la diferencia al

cuadrado de la media. Puede usar la función var incorporada para encontrar esto. Desviación estándar : la raíz cuadrada de la varianza. Puede usar la función estándar incorporada para encontrar esto.

```
']:


 - Variance: {:.2f}\n - Std.Dev: {:.2f}'.format(col_name, rng, var, std))
```

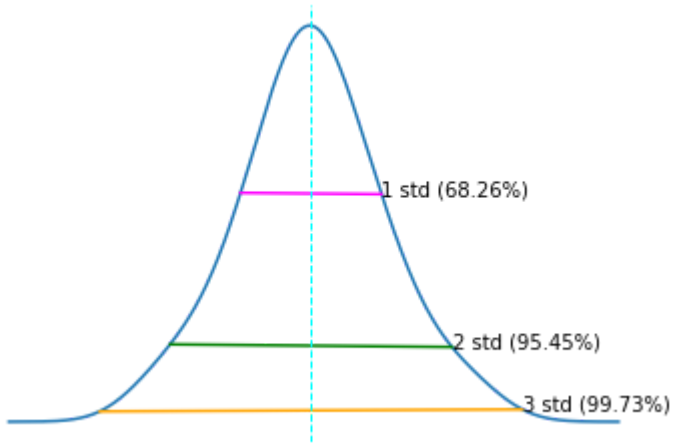
```
Grade:
 - Range: 94.00
 - Variance: 472.54
 - Std.Dev: 21.74

StudyHours:
 - Range: 15.00
 - Variance: 12.16
 - Std.Dev: 3.49
```

De estas estadísticas, la desviación estándar es generalmente la más útil. Proporciona una medida de la varianza de los datos en la misma escala que los propios datos (por lo tanto, puntos de calificación para la distribución de calificaciones y horas para la distribución de horas de estudio). Cuanto mayor es la desviación estándar, mayor es la varianza cuando se comparan los valores de la distribución con la media de la distribución; en otras palabras, los datos están más dispersos.

Cuando se trabaja con una distribución normal , la desviación estándar trabaja con las características particulares de una distribución normal para proporcionar una comprensión aún mayor. Ejecute la celda a continuación para ver la relación entre las desviaciones estándar y los datos en la distribución normal.

```
import scipy.stats as stats

# Get the Grade column
col = df_students['Grade']

# get the density
density = stats.gaussian_kde(col)

# Plot the density
col.plot.density()

# Get the mean and standard deviation
s = col.std()
m = col.mean()

```

```
# Annotate 1 stdev
x1 = [m-s, m+s]
y1 = density(x1)
plt.plot(x1,y1, color='magenta')
plt.annotate('1 std (68.26%)', (x1[1],y1[1]))

# Annotate 2 stdevs
x2 = [m-(s*2), m+(s*2)]
y2 = density(x2)
plt.plot(x2,y2, color='green')
plt.annotate('2 std (95.45%)', (x2[1],y2[1]))

# Annotate 3 stdevs
x3 = [m-(s*3), m+(s*3)]
y3 = density(x3)
plt.plot(x3,y3, color='orange')
plt.annotate('3 std (99.73%)', (x3[1],y3[1]))

# Show the location of the mean
plt.axvline(col.mean(), color='cyan', linestyle='dashed', linewidth=1)

plt.axis('off')

plt.show()
```



Las líneas horizontales muestran el porcentaje de datos dentro de 1, 2 y 3 desviaciones estándar de la media (más o menos).

En cualquier distribución normal:

Aproximadamente el 68,26% de los valores se encuentran dentro de una desviación estándar de la media. Aproximadamente el 95,45% de los valores se encuentran dentro de dos desviaciones estándar de la media. Aproximadamente el 99,73% de los valores se encuentran dentro de tres desviaciones estándar de la media. Entonces, como sabemos que la nota media es 49,18, la desviación estándar es 21,74 y la distribución de las notas es aproximadamente normal; podemos calcular que el 68,26% de los alumnos debería alcanzar una nota entre 27,44 y 70,92.

Las estadísticas descriptivas que hemos utilizado para comprender la distribución de las variables de datos de los estudiantes son la base del análisis estadístico; y debido a que son una

parte tan importante de la exploración de sus datos, hay un método Describe incorporado del objeto DataFrame que devuelve las principales estadísticas descriptivas para todas las columnas numéricas.

```
1 df_students.describe()
```

	StudyHours	Grade
count	22.000000	22.000000
mean	10.522727	49.181818
std	3.487144	21.737912
min	1.000000	3.000000
25%	9.000000	36.250000
50%	10.000000	49.500000
75%	12.375000	62.750000
max	16.000000	97.000000

Comparando datos Ahora que sabe algo sobre la distribución estadística de los datos en su conjunto de datos, está listo para examinar sus datos para identificar cualquier relación aparente entre las variables.

En primer lugar, eliminemos las filas que contengan valores atípicos para tener una muestra que sea representativa de una clase típica de estudiantes. Identificamos que la columna StudyHours contiene algunos valores atípicos con valores extremadamente bajos, por lo que eliminaremos esas filas.

```
1 df_sample = df_students[df_students['StudyHours']>1]
2 df_sample
```

	Name	StudyHours	Grade	Pass
0	Dan	10.00	50.0	False
1	Joann	11.50	50.0	False
2	Pedro	9.00	47.0	False
3	Rosie	16.00	97.0	True
4	Ethan	9.25	49.0	False
6	Frederic	11.50	53.0	False
7	Jimmie	9.00	42.0	False
8	Rhonda	8.50	26.0	False
9	Giovanni	14.50	74.0	True
10	Francesca	15.50	82.0	True
11	Rajab	13.75	62.0	True
12	Naiyana	9.00	37.0	False
13	Kian	8.00	15.0	False

Comparación de variables numéricas y categóricas Los datos incluyen dos variables numéricas (StudyHours y Grade) y dos variables categóricas (Name y Pass). Comencemos comparando la columna numérica StudyHours con la columna categórica Aprobado para ver si existe una relación aparente entre la cantidad de horas estudiadas y una calificación aprobatoria.

Para hacer esta comparación, creemos diagramas de caja que muestren la distribución de StudyHours para cada posible valor de Aprobación (verdadero y falso).

19	Skye	12.00	52.0	False

```
1 df_sample.boxplot(column='StudyHours', by='Pass', figsize=(8,5))
```

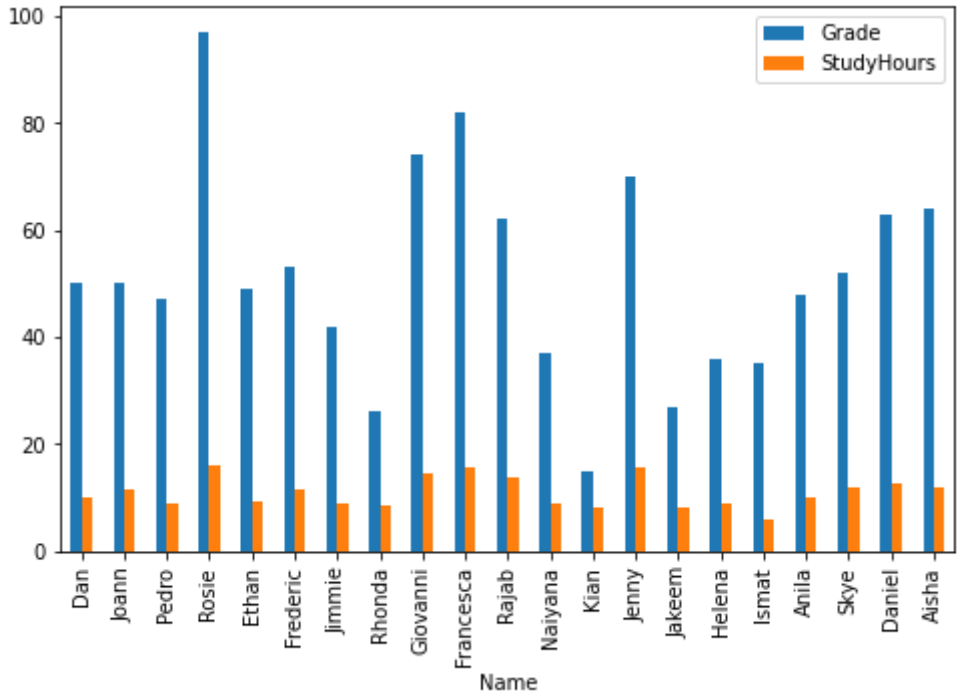
```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecat
  return array(a, dtype, copy=False, order=order)
<matplotlib.axes._subplots.AxesSubplot at 0x7f13b3211750>
```

Al comparar las distribuciones de StudyHours, es inmediatamente evidente (si no particularmente sorprendente) que los estudiantes que aprobaron el curso tendieron a estudiar más horas que los estudiantes que no lo hicieron. Entonces, si desea predecir si es probable que un estudiante apruebe o no el curso, la cantidad de tiempo que pasa estudiando puede ser una buena característica de predicción.

Comparar variables numéricas Ahora comparemos dos variables numéricas. Comenzaremos creando un gráfico de barras que muestre las calificaciones y las horas de estudio.


```
eate a bar plot of name vs grade and study hours
ample.plot(x='Name', y=['Grade','StudyHours'], kind='bar', figsize=(8,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f13b3396090>
```



Haz doble clic (o pulsa Intro) para editar

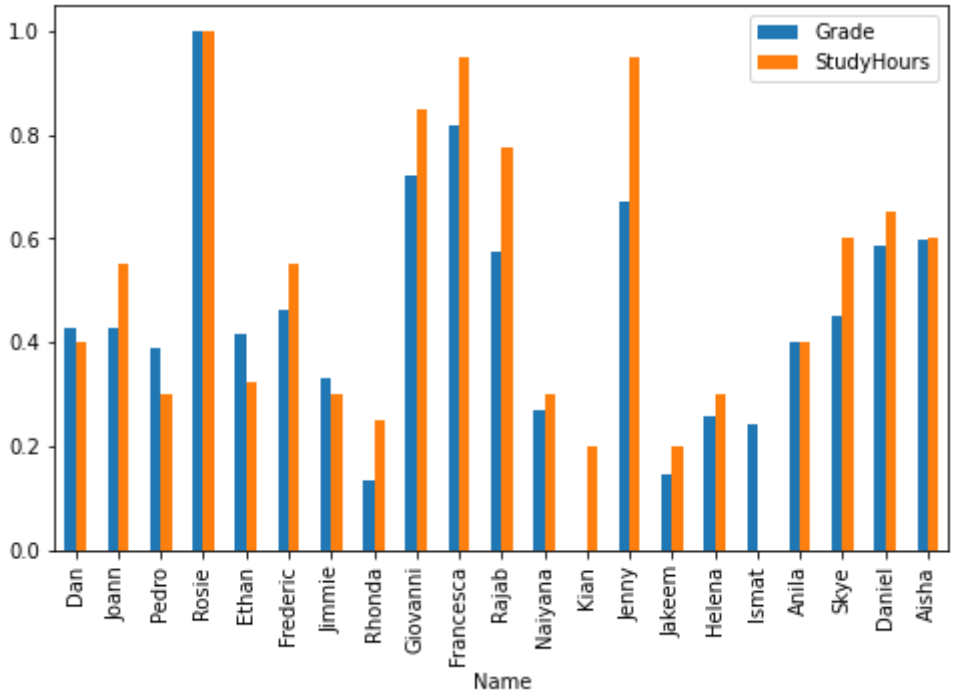
El gráfico muestra barras para el grado y las horas de estudio de cada estudiante; pero no es fácil de comparar porque los valores están en diferentes escalas. Las calificaciones se miden en puntos de calificación y varían de 3 a 97; mientras que el tiempo de estudio se mide en horas y varía de 1 a 16.

Una técnica común cuando se trabaja con datos numéricos en diferentes escalas es normalizar los datos para que los valores mantengan su distribución proporcional, pero se midan en la misma escala. Para lograr esto, usaremos una técnica llamada escala MinMax que distribuye los valores proporcionalmente en una escala de 0 a 1. Puede escribir el código para aplicar esta

transformación; pero la biblioteca Scikit-Learn proporciona un escalador para que lo haga por

```
 1 earn.preprocessing import MinMaxScaler
 2
 3 scaler object
 4  MinMaxScaler()
 5
 6  a new dataframe for the scaled values
 7 lized = df_sample[['Name', 'Grade', 'StudyHours']].copy()
 8
 9 ize the numeric columns
10 lized[['Grade','StudyHours']] = scaler.fit_transform(df_normalized[['Grade','
11
12 he normalized values
13 lized.plot(x='Name', y=['Grade','StudyHours'], kind='bar', figsize=(8,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f13b1de1290>
```



Con los datos normalizados, es más fácil ver una relación aparente entre la calificación y el tiempo de estudio. No es una coincidencia exacta, pero definitivamente parece que los estudiantes con calificaciones más altas tienden a haber estudiado más.

Entonces parece haber una correlación entre el tiempo de estudio y la calificación; y de hecho, existe una medida de correlación estadística que podemos utilizar para cuantificar la relación entre estas columnas.

```
1 df_normalized.Grade.corr(df_normalized.StudyHours)
```

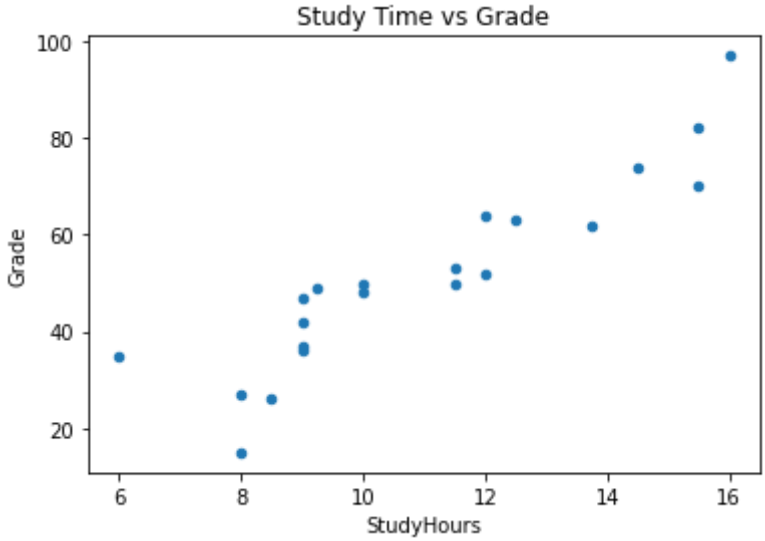
```
0.9117666413789677
```

La estadística de correlación es un valor entre -1 y 1 que indica la fuerza de una relación. Los valores por encima de 0 indican una correlación positiva (los valores altos de una variable

tienden a coincidir con los valores altos de la otra), mientras que los valores por debajo de 0 indican una correlación negativa (los valores altos de una variable tienden a coincidir con los valores bajos de la otra). En este caso, el valor de correlación es cercano a 1; mostrando una correlación fuertemente positiva entre el tiempo de estudio y el grado.

```
ate a scatter plot
mple.plot.scatter(title='Study Time vs Grade', x='StudyHours', y='Grade')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f13b30ebc90>
```

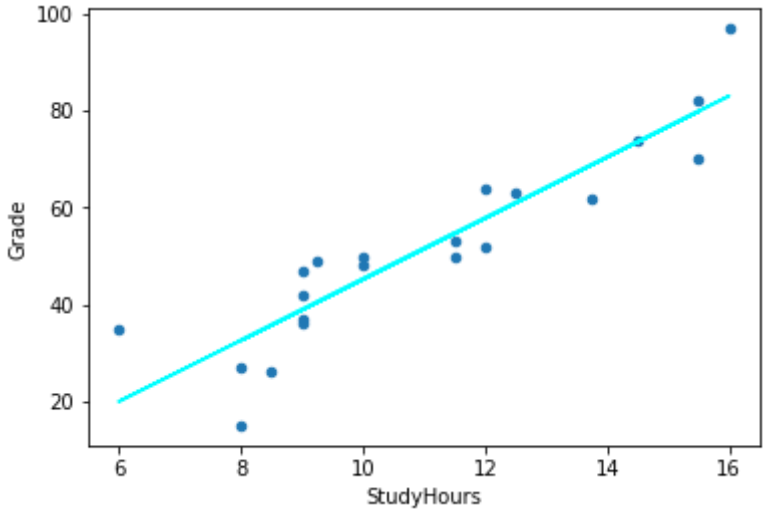


Afortunadamente, no necesita codificar el cálculo de regresión usted mismo: el paquete SciPy incluye una clase de estadísticas que proporciona un método linregress para hacer el trabajo duro por usted. Esto devuelve (entre otras cosas) los coeficientes que necesita para la ecuación de pendiente: pendiente (m) e intersección (b) en función de un par de muestras variables que desea comparar.

```
from scipy import stats

#
df_regression = df_sample[['Grade', 'StudyHours']].copy()

# Get the regression slope and intercept
m, b, r, p, se = stats.linregress(df_regression['StudyHours'], df_regression[
print('slope: {:.4f}\ny-intercept: {:.4f}'.format(m,b))
print('so...\n f(x) = {:.4f}x + {:.4f}'.format(m,b))

# Use the function (mx + b) to calculate f(x) for each x (StudyHours) value
df_regression['fx'] = (m * df_regression['StudyHours']) + b

# Calculate the error between f(x) and the actual y (Grade) value
df_regression['error'] = df_regression['fx'] - df_regression['Grade']

# Create a scatter plot of Grade vs Salary
df_regression.plot.scatter(x='StudyHours', y='Grade')

```

```
# Plot the regression line
plt.plot(df_regression['StudyHours'],df_regression['fx'], color='cyan')

# Display the plot
plt.show()
```

```
slope: 6.3134
y-intercept: -17.9164
so...
 f(x) = 6.3134x + -17.9164
```



Tenga en cuenta que esta vez, el código trazó dos cosas distintas: el diagrama de dispersión de las horas de estudio de muestra y las calificaciones se trazan como antes, y luego se traza una línea de mejor ajuste basada en los coeficientes de regresión de mínimos cuadrados.

Los coeficientes de pendiente e intersección calculados para la línea de regresión se muestran encima del gráfico.

La línea se basa en los valores f (x) calculados para cada valor de StudyHours . Ejecute la siguiente celda para ver una tabla que incluye los siguientes valores:

Las horas de estudio para cada alumno. La nota obtenida por cada alumno. El valor de f (x) calculado utilizando los coeficientes de la línea de regresión. El error entre el valor calculado de f (x) y el valor real de la calificación . Algunos de los errores, particularmente en los extremos, y bastante grandes (hasta más de 17,5 puntos); pero en general, la línea se acerca bastante a las calificaciones reales.

```
# Show the original x,y values, the f(x) value, and the error
df_regression[['StudyHours', 'Grade', 'fx', 'error']]
```

	StudyHours	Grade	fx	error
0	10.00	50.0	45.217846	-4.782154
1	11.50	50.0	54.687985	4.687985
2	9.00	47.0	38.904421	-8.095579
3	16.00	97.0	83.098400	-13.901600
4	9.25	49.0	40.482777	-8.517223
6	11.50	53.0	54.687985	1.687985
7	9.00	42.0	38.904421	-3.095579
8	8.50	26.0	35.747708	9.747708
9	14.50	74.0	73.628262	-0.371738
10	15.50	82.0	79.941687	-2.058313
11	13.75	62.0	68.893193	6.893193
12	9.00	37.0	38.904421	1.904421
13	8.00	15.0	32.590995	17.590995
14	15.50	70.0	79.941687	9.941687
15	8.00	27.0	32.590995	5.590995
16	9.00	36.0	38.904421	2.904421
17	6.00	35.0	19.964144	-15.035856

Usando los coeficientes de regresión para la predicción Ahora que tiene los coeficientes de regresión para el tiempo de estudio y la relación de calificaciones, puede usarlos en una función para estimar la calificación esperada para una determinada cantidad de estudio.

```
egression coefficients




re than 100
diction))


week may result in a grade of {:.0f}'.format(study_time, expected_grade))
```

```
Studying for 14 hours per week may result in a grade of 70
```

Entonces, al aplicar estadísticas a los datos de muestra, ha determinado una relación entre el tiempo de estudio y la calificación; y encapsuló esa relación en una función general que puede usarse para predecir una calificación para una cantidad determinada de tiempo de estudio.

Esta técnica es, de hecho, la premisa básica del aprendizaje automático. Puede tomar un conjunto de datos de muestra que incluye una o más características (en este caso, la cantidad de horas estudiadas) y un valor de etiqueta conocido (en este caso, la calificación obtenida) y usar los datos de muestra para derivar una función que calcule valores de etiqueta predichos para cualquier conjunto de características.

Verificación de conocimientos Terminado 200 XP 3 minutos Responda las siguientes preguntas para comprobar su aprendizaje.

1. Tiene una matriz NumPy con la forma (2,20). ¿Qué te dice esto sobre los elementos de la matriz?

La matriz es bidimensional y consta de dos matrices cada una con 20 elementos.

La matriz contiene 2 elementos, con los valores 2 y 20

La matriz contiene 20 elementos, todos con el valor 2

2. Tiene un DataFrame de Pandas llamado df_sales que contiene datos de ventas diarias. El DataFrame contiene las siguientes columnas: año, mes, día_de_month, sales_total. Desea encontrar el valor total de ventas promedio. ¿Qué código deberías usar?

df_sales ['sales_total']. avg ()

df_sales ['sales_total']. mean ()

mean (df_sales ['sales_total'])

3. Tiene un DataFrame que contiene datos sobre las ventas diarias de helados. Utiliza el método corr para comparar las columnas avg_temp y units_sold, y obtiene un resultado de 0.97. ¿Qué indica este resultado?

En el día con el valor máximo de unidades vendidas, el valor avg_temp fue 0.97

Los días con valores altos de avg_temp tienden a coincidir con días que tienen valores altos de unit_sold

El valor de unit_sold es, en promedio, el 97% del valor avg_temp

Resumen Terminado 100 XP 1 minuto En este módulo, aprendió a usar Python para explorar, visualizar y manipular datos. La exploración de datos es el núcleo de la ciencia de datos y es un elemento clave en el análisis de datos y el aprendizaje automático.

El aprendizaje automático es un subconjunto de la ciencia de datos que se ocupa del modelado predictivo. En otras palabras, el aprendizaje automático utiliza datos para crear modelos

predictivos, con el fin de predecir valores desconocidos. Puede utilizar el aprendizaje automático para predecir cuánta comida necesita pedir un supermercado o para identificar plantas en fotografías.

El aprendizaje automático funciona identificando relaciones entre valores de datos que describen características de algo, sus características , como la altura y el color de una planta, y el valor que queremos predecir, la etiqueta , como la especie de planta. Estas relaciones se integran en un modelo a través de un proceso de formación .

✓ 0 s completado a las 14:03