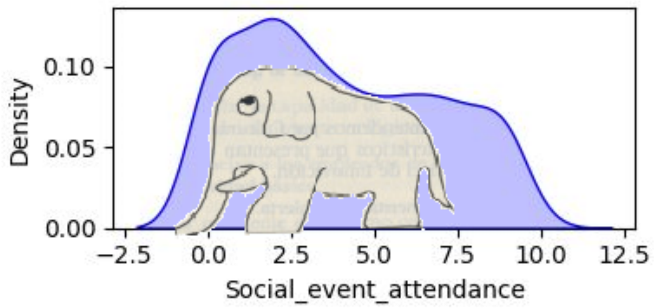
[64]:
```
plt.figure(figsize=(4, 2))
sns.kdeplot(data.Social_event_attendance, fill = True, color = 'blue',bw_adjust=1.2)
plt.tight_layout()
plt.savefig("figure1.png")
```



Valeria Valentina
Cabra Flórez

Matplotlib

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# '%matplotlib inline'used in Jupyter notebooks to display matplotlib plots directl
```

In [2]:
```
%matplotlib inline
```

In [3]:
```
data = pd.read_csv('personality_dataset.csv')
data_nums = data[['Time_spent_Alone', 'Social_event_attendance','Going_outside']]
data_nums.head(2)
```

Out[3]:

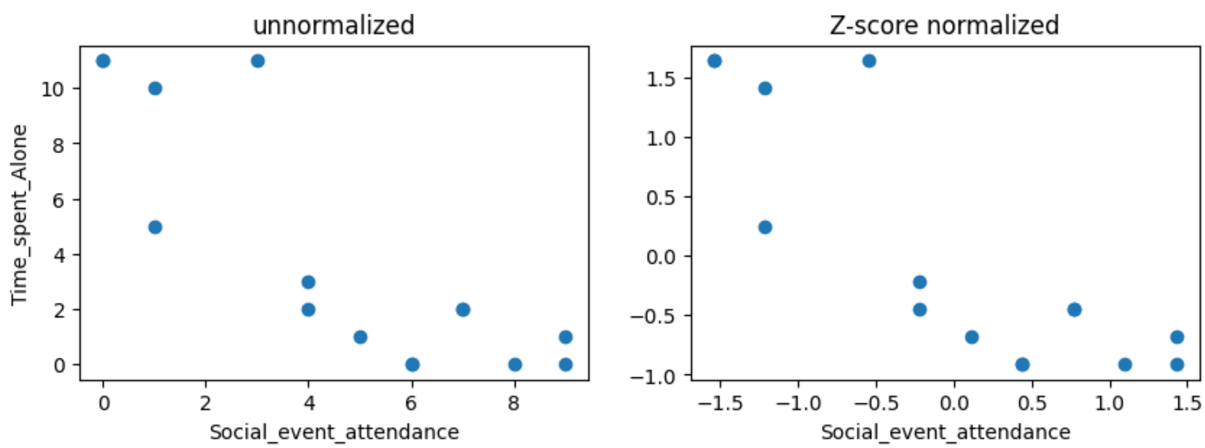
	Time_spent_Alone	Social_event_attendance	Going_outside
0	4.0	4.0	6.0
1	9.0	0.0	0.0

In [4]:
```
def zscore_normalize_features(X):
    mu     = np.mean(X, axis=0)
    sigma  = np.std(X, axis=0)
    X_norm = (X - mu) / sigma
    return (X_norm, mu, sigma)

grafir = data_nums.sample(n=15)
# O también
mu     = np.mean(grafir,axis=0)
sigma  = np.std(grafir,axis=0)
X_norm = (grafir - mu)/sigma
```

In [5]:
```
columns = data_nums.columns

fig,ax=plt.subplots(1, 2, figsize=(10, 3))
```

```
ax[0].scatter(grafir['Social_event_attendance'], grafir['Time_spent_Alone'])
ax[0].set_xlabel(columns[1]); ax[0].set_ylabel(columns[0]);
ax[0].set_title("unnormalized")

ax[1].scatter(X_norm['Social_event_attendance'], X_norm['Time_spent_Alone'])
ax[1].set_xlabel(columns[1]); ax[0].set_ylabel(columns[0]);
ax[1].set_title(r"Z-score normalized")
```

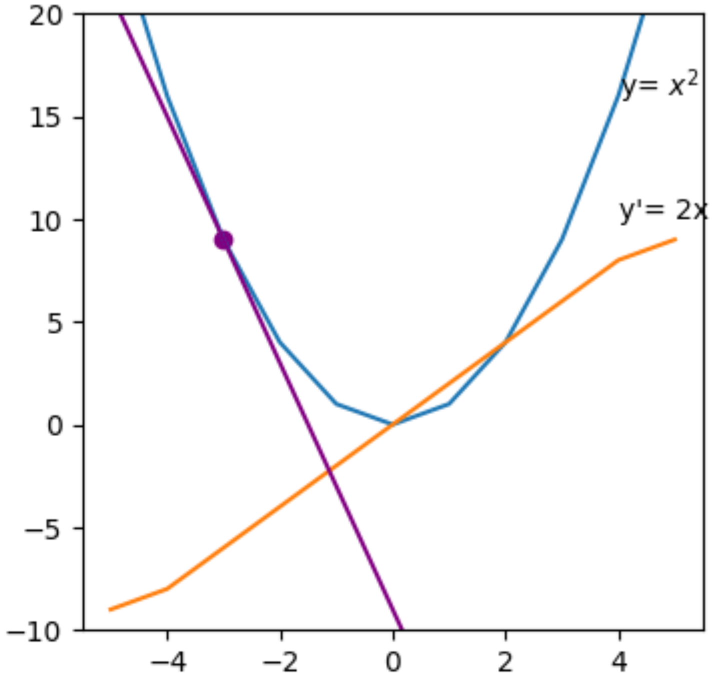
Out[5]: Text(0.5, 1.0, 'Z-score normalized')



Adding multiple lines to the same plot. This is useful when you want to compare different datasets on the same axes.

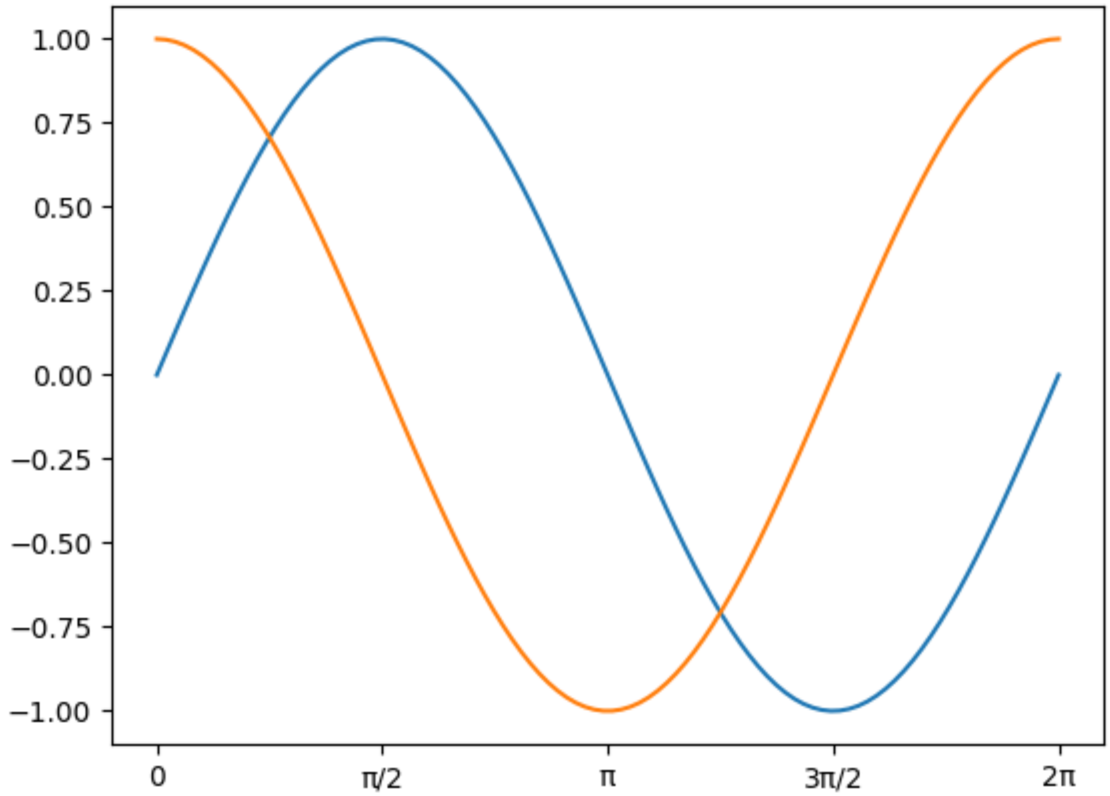
In [6]:
```
# Graficar usando método .figure, y luego otros .plot, . hist, etc
plt.figure(figsize=(4, 4)) # plt.figure(figsize=(width, height)) in inches
x = np.arange(-5,6)
y = np.arange(-5,6) ** 2
plt.plot(x, y)
plt.annotate(r'y= $x^2$', xy=(4.0, 16))
deriv = np.gradient(y)
deriv
plt.plot(x, deriv)
plt.annotate('y\'= 2x', xy=(4.0, 10))
plt.scatter (-3, 9, color = 'purple')
m = deriv[2]; b = y[2] - m*x[2]
y_purp = m * x + b
plt.plot(x, y_purp, color='purple')
plt.ylim(-10,20)
```

Out[6]: (-10.0, 20.0)



In [7]:
```
x = x = np.linspace(0, 2 * np.pi, 100)
y_sen = np.sin(x); y_cos = np.cos(x)
plt.plot(x, y_sen)
plt.plot(x, y_cos)
plt.xticks([0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi], ["0", "π/2", "π", "3π/2", "2π"]
```

Out[7]:
```
([<matplotlib.axis.XTick at 0x24cf61e19a0>,
  <matplotlib.axis.XTick at 0x24cf613a990>,
  <matplotlib.axis.XTick at 0x24cf61e1eb0>,
  <matplotlib.axis.XTick at 0x24cf6209b80>,
  <matplotlib.axis.XTick at 0x24cf620a930>],
 [Text(0.0, 0, '0'),
  Text(1.5707963267948966, 0, 'π/2'),
  Text(3.141592653589793, 0, 'π'),
  Text(4.71238898038469, 0, '3π/2'),
  Text(6.283185307179586, 0, '2π')])
```

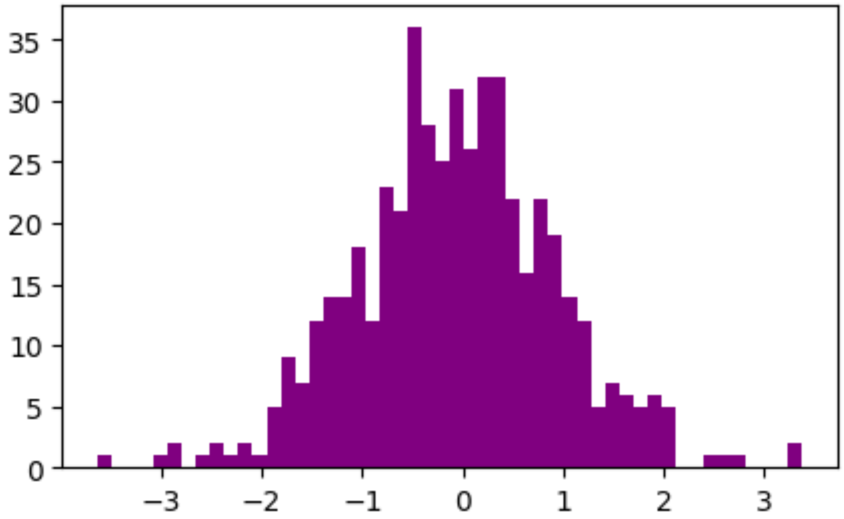


.figure, y luego otros .plot, . hist, etc

In [8]:
```
values = np.random.standard_normal(500)
print('El rango es: ', min(values), max(values))
#rtg: print('Los values son: ', values)
plt.figure(figsize=(5,3))
plt.hist(values, bins=50, color= 'purple')
```

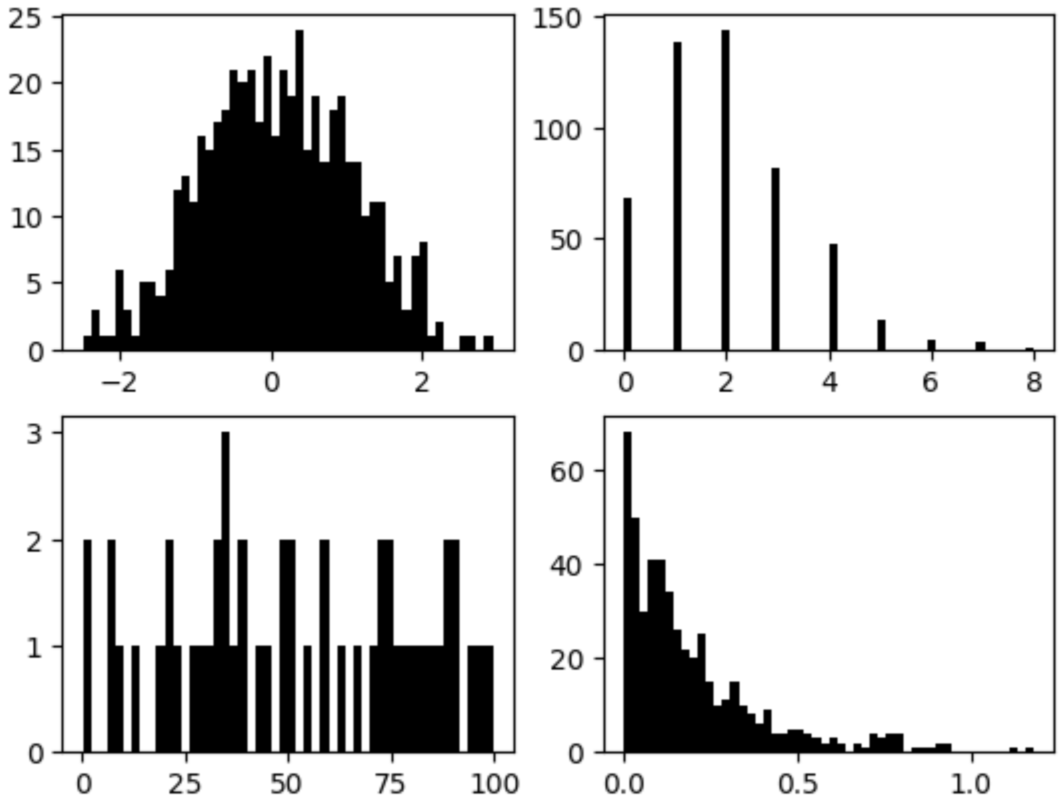
```
El rango es:  -3.6346958886092717 3.3744767538185534
```

Out[8]:
```
(array([ 1.,  0.,  0.,  0.,  1.,  2.,  0.,  1.,  2.,  1.,  2.,  1.,  5.,
         9.,  7., 12., 14., 14., 18., 12., 23., 21., 36., 28., 25., 31.,
        26., 32., 32., 22., 16., 22., 19., 14., 12.,  5.,  7.,  6.,  5.,
         6.,  5.,  0.,  0.,  1.,  1.,  1.,  0.,  0.,  0.,  2.]),
 array([-3.63469589, -3.49451244, -3.35432898, -3.21414553, -3.07396208,
        -2.93377862, -2.79359517, -2.65341172, -2.51322827, -2.37304481,
        -2.23286136, -2.09267791, -1.95249445, -1.812311  , -1.67212755,
        -1.5319441 , -1.39176064, -1.25157719, -1.11139374, -0.97121028,
        -0.83102683, -0.69084338, -0.55065993, -0.41047647, -0.27029302,
        -0.13010957,  0.01007389,  0.15025734,  0.29044079,  0.43062424,
         0.5708077 ,  0.71099115,  0.8511746 ,  0.99135806,  1.13154151,
         1.27172496,  1.41190841,  1.55209187,  1.69227532,  1.83245877,
         1.97264223,  2.11282568,  2.25300913,  2.39319258,  2.53337604,
         2.67355949,  2.81374294,  2.9539264 ,  3.09410985,  3.2342933 ,
         3.37447675]),
 <BarContainer object of 50 artists>)
```



La generación de la gráfica outputs 2 arrays con números que se usan para hacer el histograma

In [9]:
```
# Graficar creando fig y axes, con método.subplots
fig, axes = plt.subplots(2,2, sharex= False, sharey= False) # crear figura/ blanco _
# 2 columnas en el espacio

# Generar 500 valores en cada tipo de distribución, salvo la uniforme
distribuciones = [[np.random.standard_normal(500), np.random.poisson(2, size= 500)]
                  [np.random.uniform(0, 100,50), np.random.exponential(0.2, size= 5
for i in range (2):
    for j in range(2):
        axes[i][j].hist(distribuciones[i][j], bins=50, color='black')
plt.show()
```

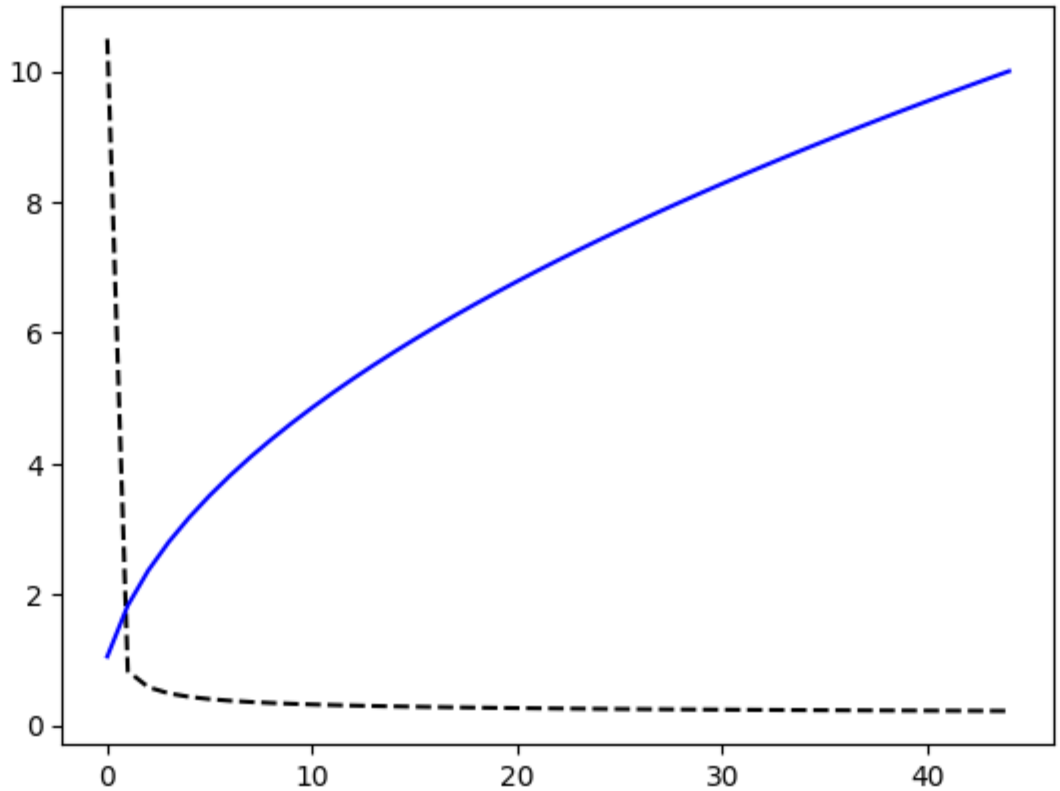


Sobre las distribuciones: np.random.standard_normal(500), the 500 specifies the number of random samples to generate from the standard normal; np.random.binomial(n,p); np.random.poisson(lam, size = number of samples desired); np.random.exponential(scale, size = No samples desired); np.random.uniform(low, high, size) will generate random numbers between low and high.

subplots allow you to create multiple plots within the same figure. This is useful when you want to display different datasets separately. The visual result might look similar if you plot multiple lines on the same axes or use subplots, but the structure of your code is different

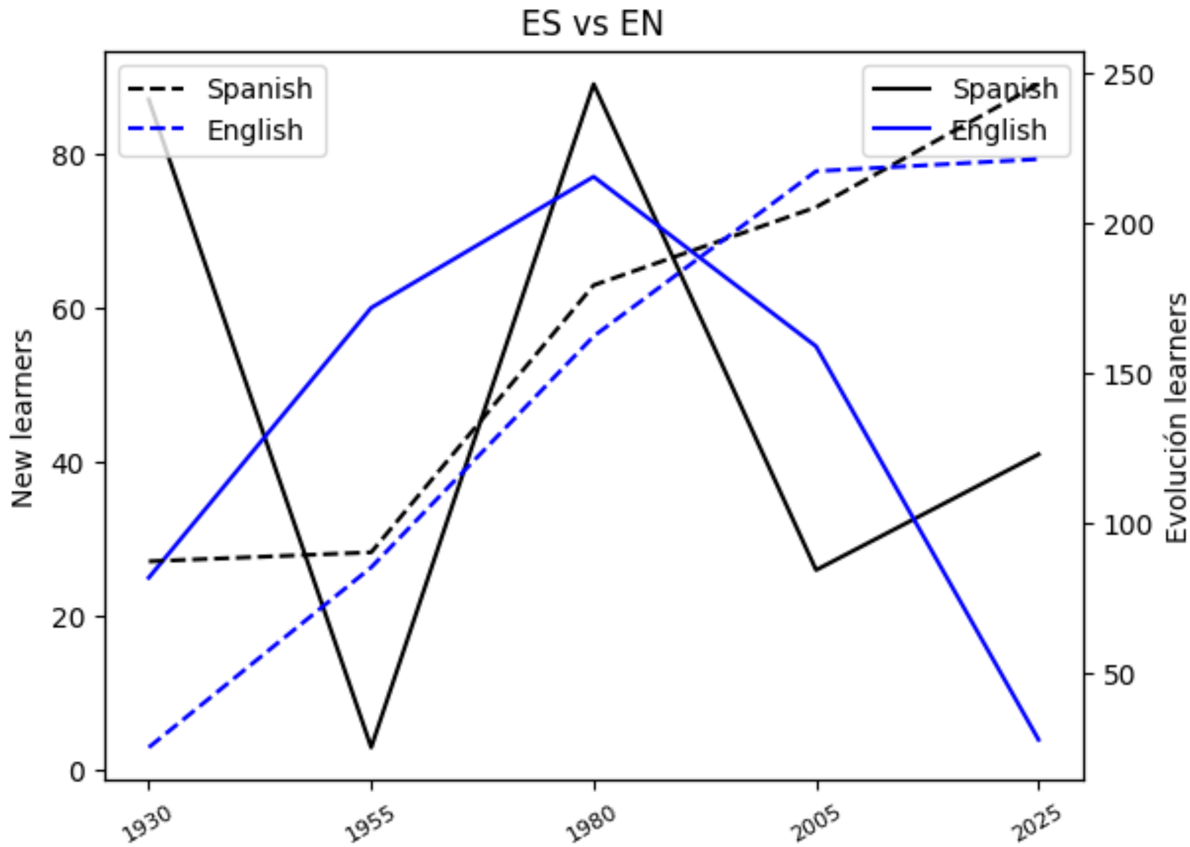
In [10]:
```
# graficar ln(x) y x**1/2. Usar método .add_subplot to a figure
absci = np.linspace(1.1,100,45) # crea 45 valores entre 1.1 y 100; igualmente dispe
logarit = 1/np.log(absci)
raiz= np.sqrt(absci)

fig = plt.figure()
ax = fig.add_subplot()

ax.plot(logarit, color='black', linestyle='dashed')
ax.plot(raiz, color='blue', linestyle='-')
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x24cf87dfe90>]
```



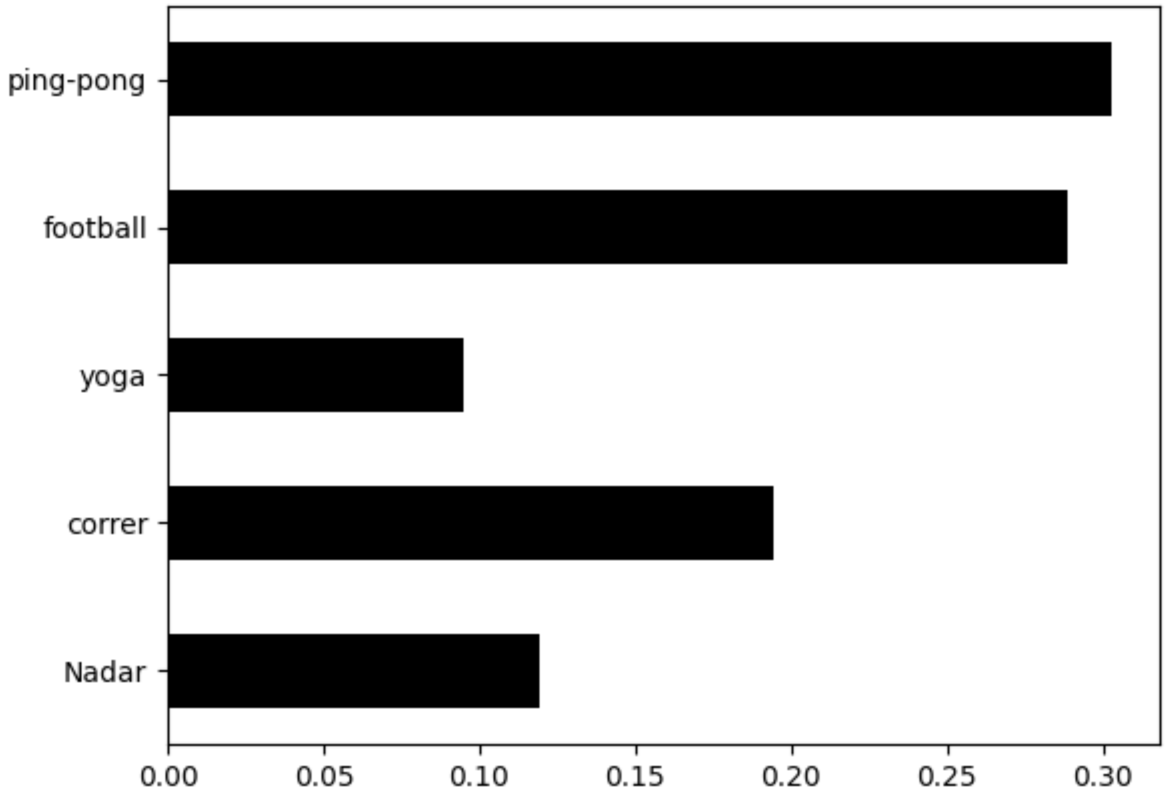
In [11]:
```
fig, ax = plt.subplots()
espagnol = np.random.randint(0,100, size=5)
print(espagnol)
english = np.random.randint(0,100, size=5)
print(english)

ax.plot(espagnol, color='black', label= 'Spanish', linestyle='-')
ax.plot(english, color='blue', label= 'English', linestyle='-')# Generar 5 values a

ticks = ax.set_xticks([0, 1, 2, 3, 4]) # dividir el eje en 5 partes iguales
labels = ax.set_xticklabels(['1930','1955','1980','2005','2025'], rotation=30, font

# Para las acumulativas...
cum_espagnol = np.cumsum(espagnol)
print(cum_espagnol)
cum_english = np.cumsum(english)
print(cum_english)

ax2 = ax.twinx() # crea un segundo eje y, que comparte el eje x de antes
ax2.plot(cum_espagnol, color='black', label= 'Spanish', linestyle='--')
ax2.plot(cum_english, color='blue', label= 'English', linestyle='--')

ax.legend()
ax2.legend()

ax.set_ylabel('New learners')
ax2.set_ylabel('Evolución learners')
ax.set_title('ES vs EN')
plt.show()
```

```
[87  3 89 26 41]
[25 60 77 55  4]
[ 87  90 179 205 246]
[ 25  85 162 217 221]
```



In [12]:
```
# Generar 5 valores aleatorios que sumen 100% o 1
random_values = np.random.rand(5)
suma = np.sum(random_values)
probab = random_values/ suma # This is a Numpy array
data = pd.Series(probab, index = ['Nadar', 'correr', 'yoga', 'football', 'ping-pong

fig, ax = plt.subplots()
data.plot.barh(color='black')
```

Out[12]: <Axes: >



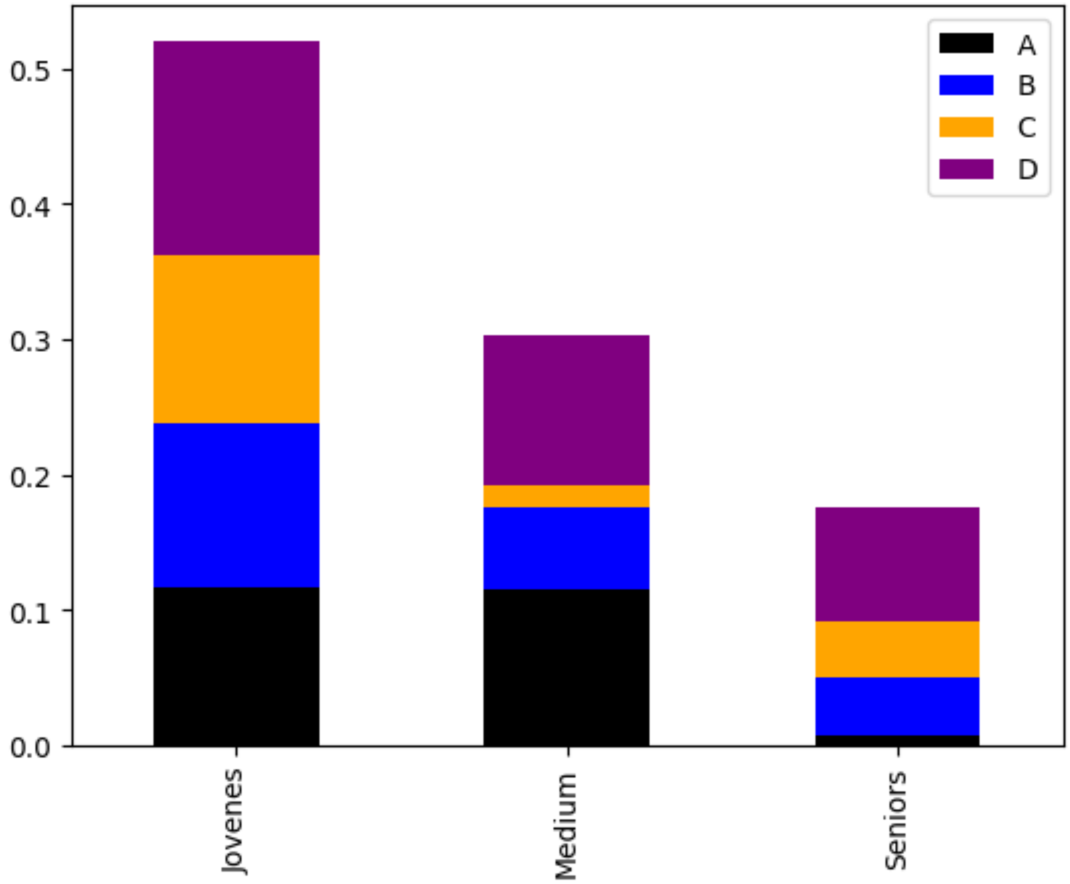
In [13]:
```
values = np.random.rand(12)
probab = values/np.sum(values)
val_tabl = probab.reshape(3,4)
df = pd.DataFrame(val_tabl, index= ['Jovenes', 'Medium', 'Seniors'],
                  columns = pd.Index(['A', 'B', 'C', 'D']))
df
```

Out[13]:

	A	B	C	D
Jovenes	0.115876	0.122584	0.124037	0.158424
Medium	0.115027	0.060817	0.016858	0.110722
Seniors	0.007549	0.041973	0.041392	0.084740

In [14]:
```
df.plot.bar(stacked=True, color=['black', 'blue', 'orange', 'purple'])
```

Out[14]: `<Axes: >`



Seaborn

In [15]:
```
import seaborn as sns
```

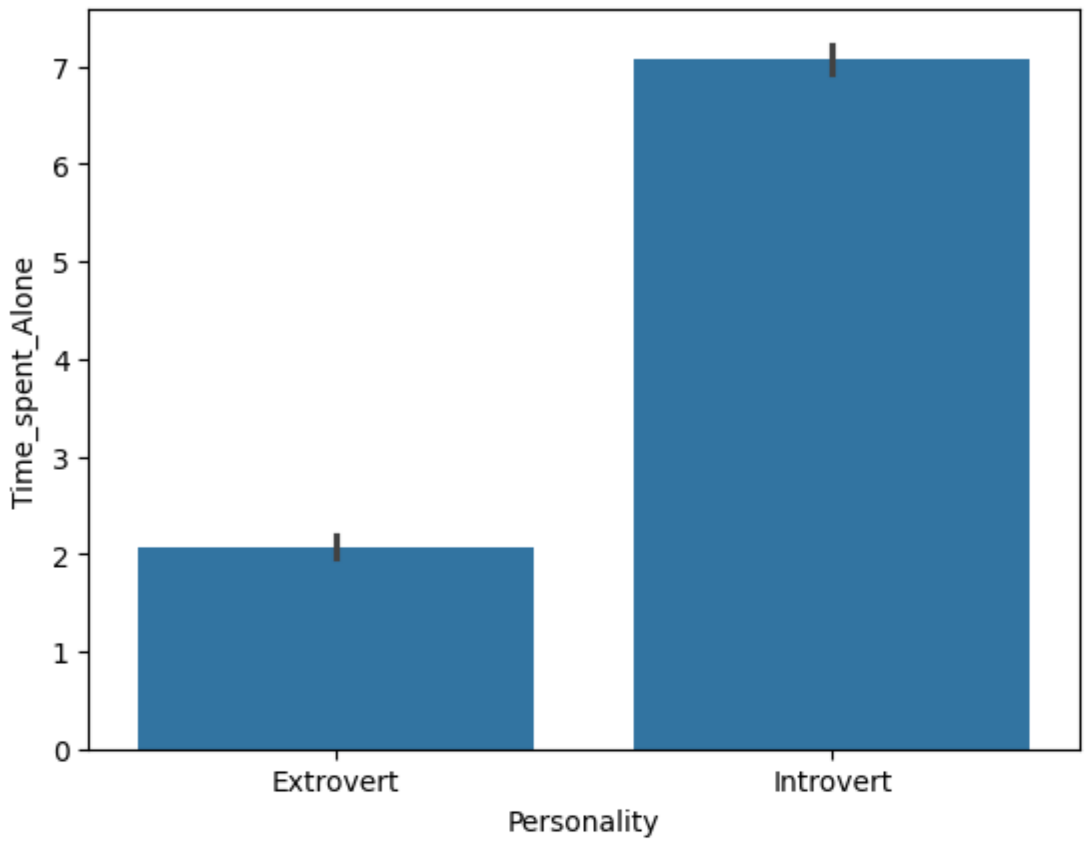
In [16]:
```
data = pd.read_csv('personality_dataset.csv')
data.head()
```

Out[16]:

	Time_spent_Alone	Stage_fear	Social_event_attendance	Going_outside	Drained_after_soc
0	4.0	No	4.0	6.0	
1	9.0	Yes	0.0	0.0	
2	9.0	Yes	1.0	2.0	
3	0.0	No	6.0	7.0	
4	3.0	No	9.0	4.0	

In [17]:
```
sns.barplot(x='Personality', y='Time_spent_Alone', data=data)
```

Out[17]: <Axes: xlabel='Personality', ylabel='Time_spent_Alone'>



Dan los 7.08 de media de Introvertidos vs los 2.06 hrs de media de los extroverts, que hallaremos con datis.groupby('Personality')[['Time_spent_Alone','...]].mean()

```
- Time_spent_Alone: Hours spent alone daily (0-11).
- Stage_fear: Presence of stage fright (Yes/No).
- Social_event_attendance: Frequency of social events (0-10).
- Going_outside: Frequency of going outside (0-7).
- Drained_after_socializing: Feeling drained after socializing (Yes/No).
- Friends_circle_size: Number of close friends (0-15).
- Post_frequency: Social media post frequency (0-10).
- Personality: Target variable (Extrovert/Introvert).*
```

In [18]:

```
# Implementación en Matplotlib sería
grafi = data.groupby('Personality')[['Time_spent_Alone']].mean()
print(type(grafi))
grafi
```

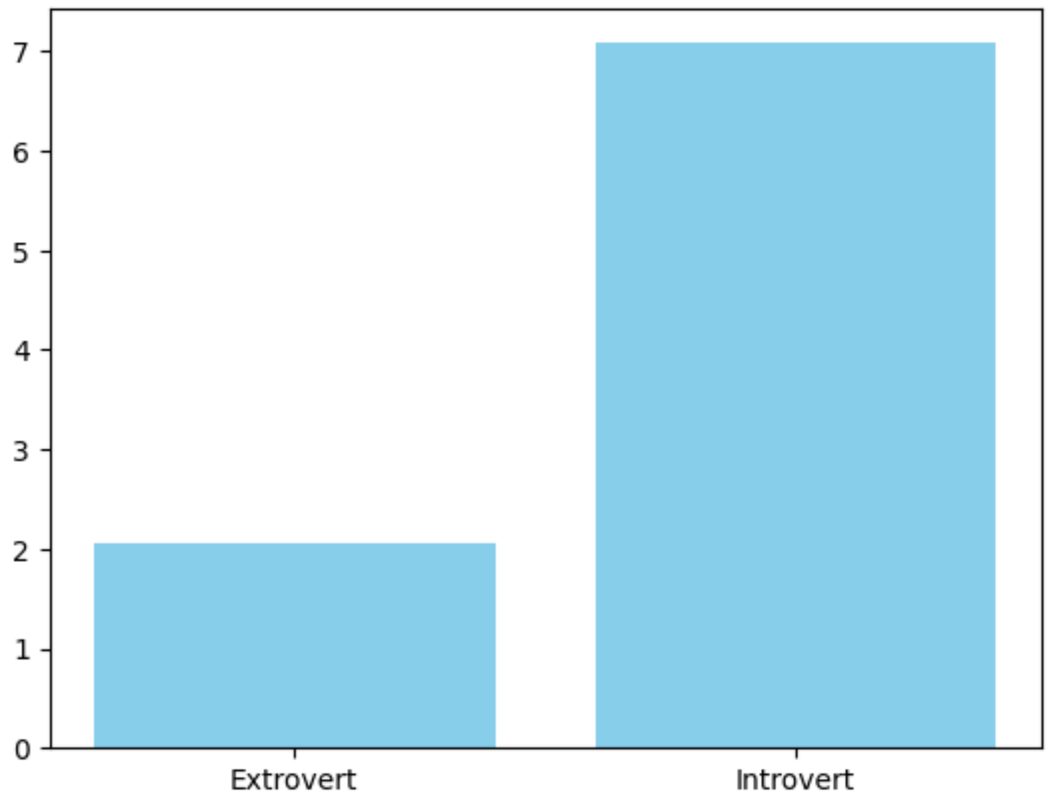
```
<class 'pandas.core.frame.DataFrame'>
```

Out[18]:

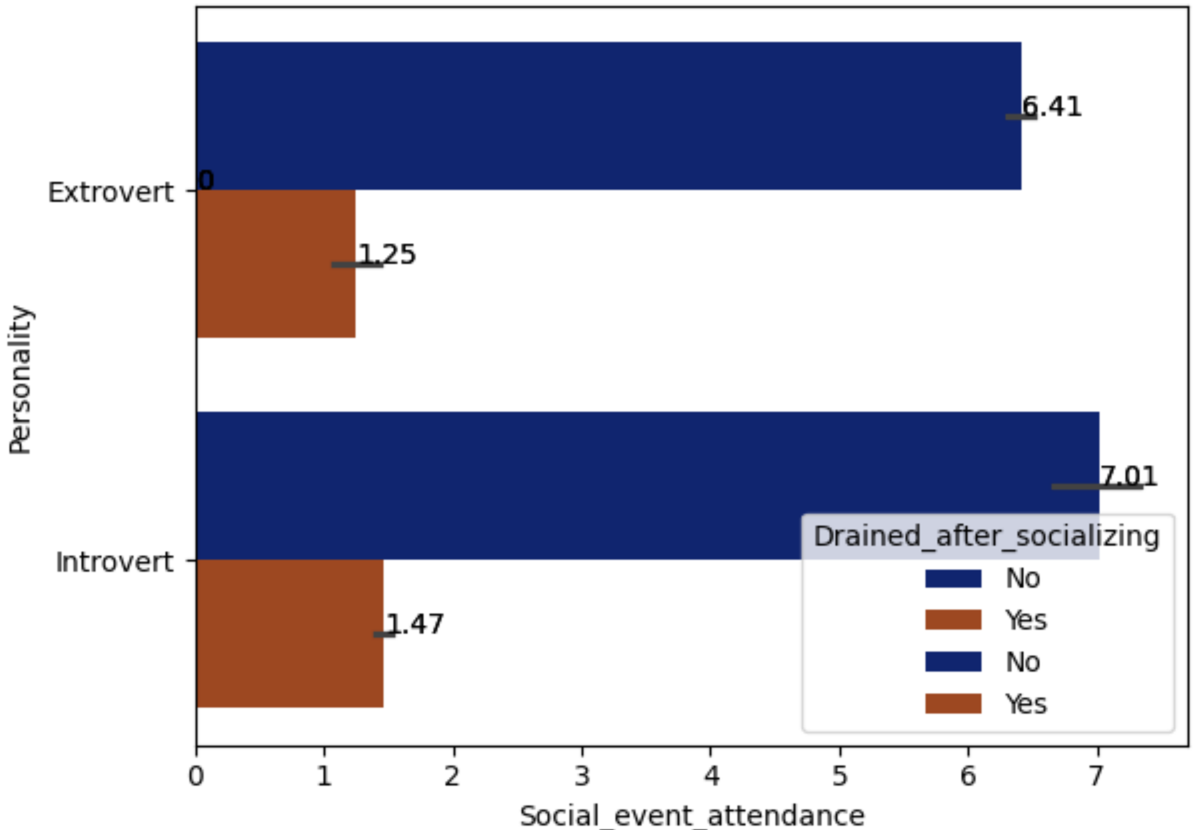
	Time_spent_Alone
Personality	
Extrovert	2.067261
Introvert	7.080435

In [19]:
```
# bar function needs two lists or arrays: one for the x values and one for the y va
print(grafi.index) # This is an Index object , it works with plt.bar
print(grafi.values) # This needs to be flatten since it is a 2D array
plt.bar(grafi.index, grafi.values.flatten(), color = 'skyblue')
plt.show()
```

```
Index(['Extrovert', 'Introvert'], dtype='object', name='Personality')
[[2.0672615 ]
 [7.08043478]]
```



In [20]:
```
sns.barplot(x='Social_event_attendance', y = 'Personality', hue = 'Drained_after_so
            palette= 'dark')
# hue means a third category used to color the bars
# get the axes object
ax = sns.barplot(x='Social_event_attendance', y = 'Personality', hue = 'Drained_aft
            palette= 'dark')
# Iterar sobre cada barra para poner el valor
for p in ax.patches:
# annotate is a function used to add text to the plot.The first argument is the tex
    ax.annotate(round(p.get_width(),2),
                xy=(p.get_width(), p.get_y() + p.get_height() / 2)) # el texto en e
```



In [21]:
```python
# groupby method with multiple columns
data.groupby(['Personality', 'Drained_after_socializing'])['Social_event_attendance
```

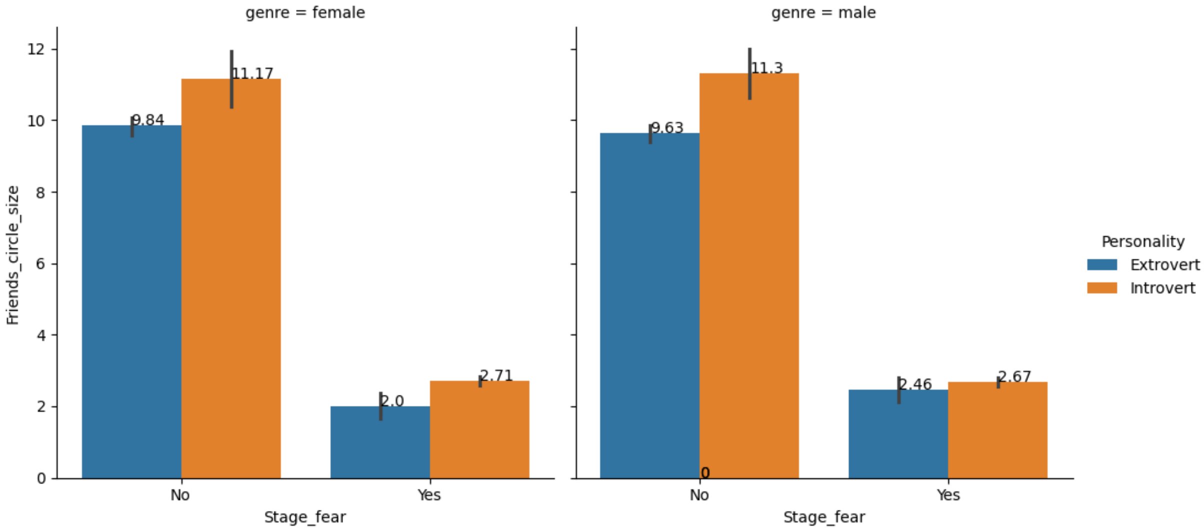
Out[21]:
```
Personality  Drained_after_socializing
Extrovert    No                           6.410795
             Yes                          1.252252
Introvert    No                           7.012821
             Yes                          1.465981
Name: Social_event_attendance, dtype: float64
```

In [22]:
```python
# Crear virtualmente la categoria género
import numpy as np
# Saber a cuántas filas les tengo que inventar el value. data.shape
genre = np.random.choice(["male", "female"], 2900)
data['genre'] = genre

data.head()
```

Out[22]:

	Time_spent_Alone	Stage_fear	Social_event_attendance	Going_outside	Drained_after_soc
0	4.0	No	4.0	6.0	
1	9.0	Yes	0.0	0.0	
2	9.0	Yes	1.0	2.0	
3	0.0	No	6.0	7.0	
4	3.0	No	9.0	4.0	

In [23]:
```
# get the axes object
g = sns.catplot(x='Stage_fear', y='Friends_circle_size', hue='Personality', col='ge

# Loops through each bar in each subplot and adds the value as a label.
for ax in g.axes.flat:
    for bar in ax.patches: # for the catplot isnt patches but axes
        ax.annotate(round(bar.get_height(),2), xy= (bar.get_x() + bar.get_width() /
```



Para posicionar las etiquetas: se ponen iterando sobre el atributo 'patches' que tiene cada bar con el método annotate que tiene cada bar. para el método.annotate: El primer argumento es el hight(si la gráfica es vert) o width de la bar (si la gráf es en horizontal) el segundo argumento xy, es un punto con coordenadas xy. Si la gráfica es vertical en x se promedia la posi en x y el width de la bar; en y va la hight de la bar. Si la gráfica es horz en x se pone el width de la bar ; en y e promedia la posi en y y la hight de la bar

In [24]:
```
data.groupby(['Personality', 'Stage_fear','genre'])['Friends_circle_size'].mean()
```

Out[24]:
```
Personality  Stage_fear  genre
Extrovert    No          female     9.843227
                         male       9.634526
             Yes         female     2.000000
                         male       2.457627
Introvert    No          female    11.171429
                         male      11.295455
             Yes         female     2.706815
                         male       2.670886
Name: Friends_circle_size, dtype: float64
```

Salaries vs. years of experience

In [25]:
```
data_s = pd.read_csv('Salary_Data.csv')
genre = np.random.choice(["male", "female"], 30)
data_s['genre'] = genre
data_s.head(5)
```

Out[25]:

	YearsExperience	Salary	genre
0	1.1	39343	male
1	1.3	46205	female
2	1.5	37731	male
3	2.0	43525	female
4	2.2	39891	female

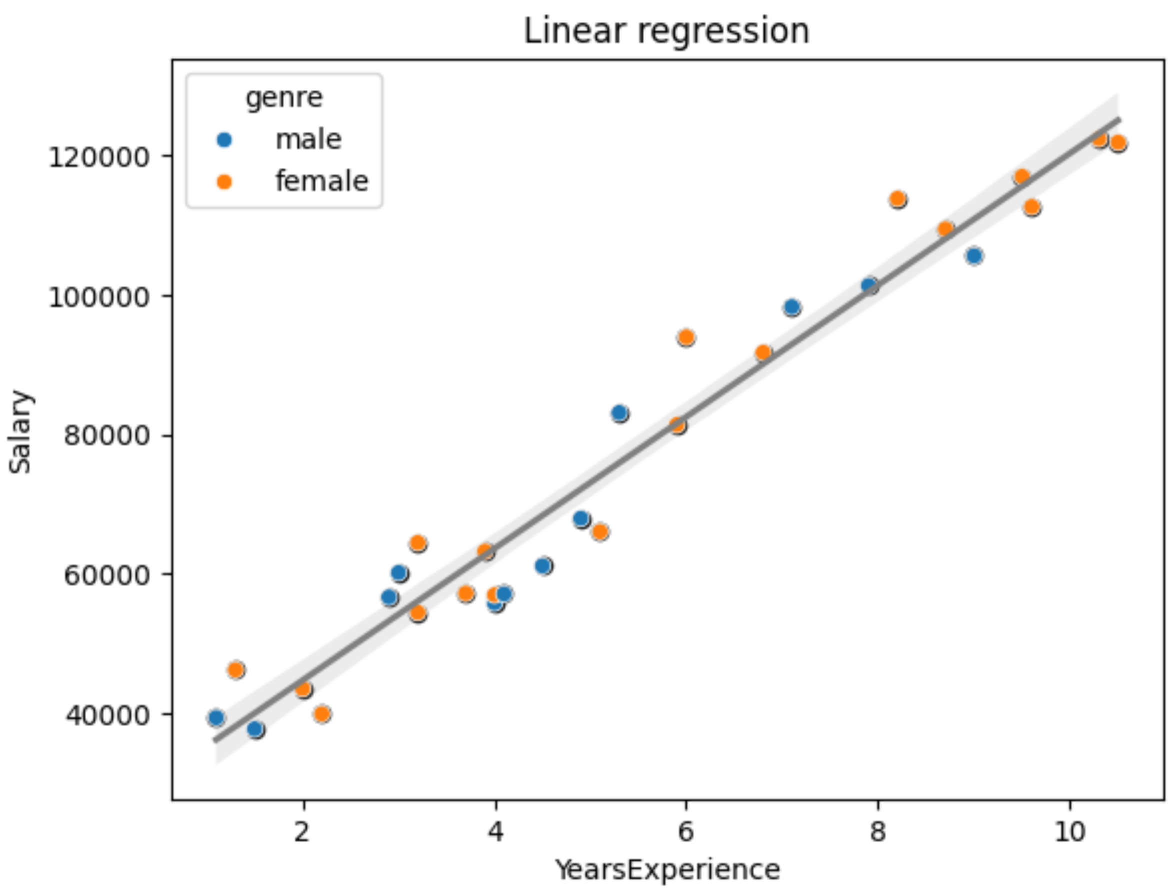
In [26]:
```
print("Shape: ",data_s.shape)
data_s.isna().sum()
```

```
Shape:  (30, 3)
```

Out[26]:
```
YearsExperience    0
Salary             0
genre              0
dtype: int64
```

In [27]:
```
fig, ax = plt.subplots()
reg = sns.regplot(x='YearsExperience', y='Salary', data=data_s, scatter_kws={'color
# los parámetros (,x_jitter=0.4, y_jitter = 0.4,) jitter sirven para agregar fluctu
sns.scatterplot(x='YearsExperience', y='Salary', data=data_s, hue='genre', ax=ax)
reg.set_title('Linear regression')
```

Out[27]:
```
Text(0.5, 1.0, 'Linear regression')
```



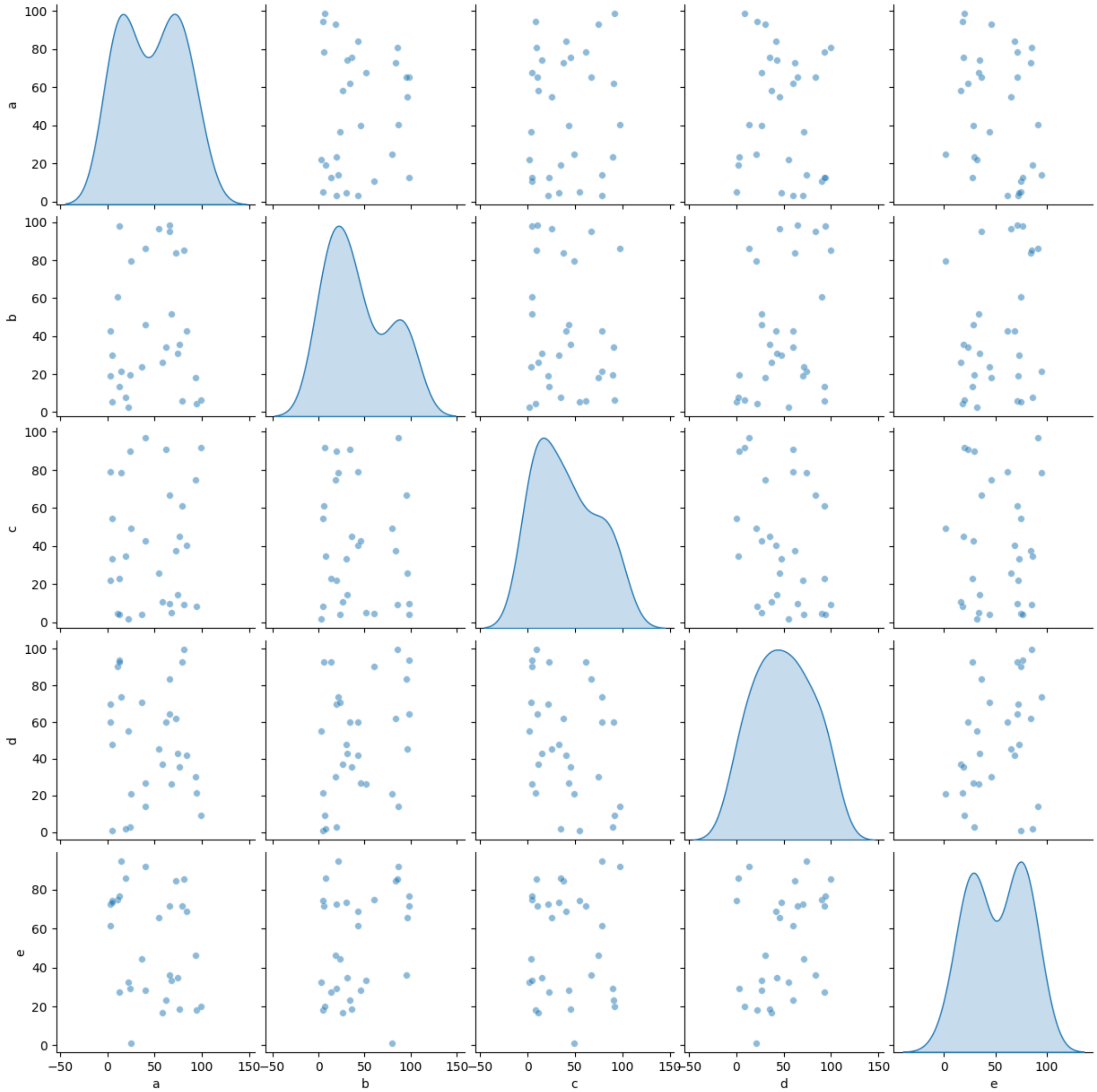
In [28]:
```
# Crear data frame ficticio con floats entre 0 y 100. en random.rand los genera ent
dati = pd.DataFrame(np.random.rand(150).reshape(30,5)*100, columns=['a','b','c','d'
dati.head(5)
```

Out[28]:

	a	b	c	d	e
0	54.740679	96.442175	25.601516	45.469567	65.433959
1	10.730672	60.530744	4.422005	90.195058	74.729211
2	14.132730	21.347989	78.726215	73.698439	94.682759
3	12.486991	98.104714	4.352024	93.837673	76.893502
4	78.661344	5.925726	61.217270	92.817974	71.668106

In [29]:
```
sns.pairplot(dati, diag_kind='kde', plot_kws={'alpha':0.5, 's':30})
# sets the transparency of the scatter plots to 0.5, s is a parameter that controls
```

Out[29]:
```
<seaborn.axisgrid.PairGrid at 0x24ced1f7800>
```


a
b
c
d
e
a
b
c
d
e

Numpy

In [1]:
```
import numpy as np
data = np.array([[88, 89, 90],
                 [91, 92, 93]])
data
print(data.shape, data.ndim)

data_1 = data.astype(np.float64)
data_1
```

```
(2, 3) 2
```

Out[1]:
```
array([[88., 89., 90.],
       [91., 92., 93.]])
```

In [2]:
```
matri3D = np.array([[[1,2,3],
                      [4,5,6],
                      [7,8,9]],
                     [[10,11,12],
                      [13,14,15],
                      [16,17,18]]])
matri3D
matri3D[1,0,2]
```

Out[2]:
```
np.int64(12)
```

In [3]:
```
reshapi = np.arange(30).reshape(6,5)
reshapi
```

Out[3]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24],
       [25, 26, 27, 28, 29]])
```

In [4]:
```
transponer = reshapi.T
transponer
```

Out[4]:
```
array([[ 0,  5, 10, 15, 20, 25],
       [ 1,  6, 11, 16, 21, 26],
       [ 2,  7, 12, 17, 22, 27],
       [ 3,  8, 13, 18, 23, 28],
       [ 4,  9, 14, 19, 24, 29]])
```

In [5]:
```
verdades = np.array(['a', 'b', 'a'])
arrai = np.array([[1,2,3],[4,5,6], [7,8,9]])
arrai[verdades == 'a']
```

Out[5]:
```
array([[1, 2, 3],
       [7, 8, 9]])
```

In [6]:
```
transponer>13
```

Out[6]:
```
array([[False, False, False,  True,  True,  True],
       [False, False, False,  True,  True,  True],
       [False, False, False,  True,  True,  True],
       [False, False, False,  True,  True,  True],
       [False, False,  True,  True,  True,  True]])
```

In [7]:
```
np.where(transponer>13, 1,0)
```

Out[7]:
```
array([[0, 0, 0, 1, 1, 1],
       [0, 0, 0, 1, 1, 1],
       [0, 0, 0, 1, 1, 1],
       [0, 0, 0, 1, 1, 1],
       [0, 0, 1, 1, 1, 1]])
```

In [8]:
```
transponer
```

Out[8]:
```
array([[ 0,  5, 10, 15, 20, 25],
       [ 1,  6, 11, 16, 21, 26],
       [ 2,  7, 12, 17, 22, 27],
       [ 3,  8, 13, 18, 23, 28],
       [ 4,  9, 14, 19, 24, 29]])
```

In [9]:
```
transponer.mean()# Media general
```

Out[9]:
```
np.float64(14.5)
```

In [10]:
```
transponer.mean (axis=0) # Media columnas
```

Out[10]:
```
array([ 2.,  7., 12., 17., 22., 27.])
```

In [11]:
```
transponer.mean(axis=1) # Media filas
```

Out[11]:
```
array([12.5, 13.5, 14.5, 15.5, 16.5])
```

- Tbm se puede transponer.cumsum(), .cumsum(axis=0 o 1);
- tambien transponer.sort(), .sort(axis= 0 o 1)

In [12]:
```
transponer.cumsum(axis=0)
```

Out[12]:
```
array([[  0,   5,  10,  15,  20,  25],
       [  1,  11,  21,  31,  41,  51],
       [  3,  18,  33,  48,  63,  78],
       [  6,  26,  46,  66,  86, 106],
       [ 10,  35,  60,  85, 110, 135]])
```

In [13]:
```
normali = np.random.standard_normal((8,3))
normali
```

Out[13]:
```
array([[-0.04441088,  1.22344868, -0.02810148],
       [-1.25389542,  1.08176346,  0.51474356],
       [ 0.85674811,  0.40564538,  2.31270661],
       [ 0.09215684, -0.54825059,  2.03111159],
       [-0.49882715, -1.3402853 , -0.4048148 ],
       [-0.45681639,  0.19256157, -0.26399653],
       [ 0.24134812,  0.96943478, -0.65514648],
       [-0.23717841, -1.77271972, -0.9497772 ]])
```

Pandas

Series

In [14]:
```
import pandas as pd
```

In [15]:
```
seri = pd.Series([1,2,3,4])
seri
```

Out[15]:
```
0    1
1    2
2    3
3    4
dtype: int64
```

In [16]:
```
colores = pd.Series([10,11,25], index = ['azul', 'amarillo', 'rojo'])
colores
```

Out[16]:
```
azul        10
amarillo    11
rojo        25
dtype: int64
```

In [17]:
```
colores.index
# dtype = 'object' in pandas means string
```

Out[17]:
```
Index(['azul', 'amarillo', 'rojo'], dtype='object')
```

As in normal python: colores['azul'] outputs 10; colores['azul'] = 1000 cambia el 10 por 1000

In [18]:
```
colores[['azul', 'rojo']]
```

Out[18]:
```
azul    10
rojo    25
dtype: int64
```

In [19]:
```
colores[colores>10]
```

Out[19]:
```
amarillo    11
rojo        25
dtype: int64
```

si se pusiera colores*10, multiplica cada elemento de la serie por 10. 'azul' in colores da True

In [20]:
```
contact = {'name': 'Valeria', 'second name': 'Valentina', 'apellido': 'Cabra', 'tel
serie_contact = pd.Series(contact)
serie_contact
```

Out[20]:
```
name              Valeria
second name     Valentina
apellido            Cabra
telefono        +34658223
dtype: object
```

In [21]:
```
serie_contact.to_dict()
```

Out[21]:
```
{'name': 'Valeria',
 'second name': 'Valentina',
 'apellido': 'Cabra',
 'telefono': '+34658223'}
```

In [22]:
```
tags = [ 'apellido','second name','name', 'telefono', 'Estatura']
objeto = pd.Series(contact, index = tags)
objeto
```

Out[22]:
```
apellido            Cabra
second name     Valentina
name              Valeria
telefono        +34658223
Estatura              NaN
dtype: object
```

In [23]:
```
pd.isna(objeto) # tbm funciona pd.notna(objeto) ; pd.isna(objeto).sum() daría 1
```

Out[23]:
```
apellido       False
second name    False
name           False
telefono       False
Estatura        True
dtype: bool
```

DataFrames

In [24]:
```
dicti = {
    "fruits": ["apple", "banana", "mango", "grape", "kiwi"],
    "colors": ["red", "blue", "green", "yellow", "purple"],
    "animals": ["cat", "dog", "lion", "tiger", "elephant"],
    "cities": ["London", "Paris", "Tokyo", "New York", "Berlin"],
    "numeri": [100,115,20,3,69]
}
dicti_frame = pd.DataFrame(dicti)
dicti_frame
```

Out[24]:

	fruits	colors	animals	cities	numeri
0	apple	red	cat	London	100
1	banana	blue	dog	Paris	115
2	mango	green	lion	Tokyo	20
3	grape	yellow	tiger	New York	3
4	kiwi	purple	elephant	Berlin	69

*Se podría poner dicti.head(2) o dicti.tail(1)

In [25]:

```
# otra forma de crear DataFrames
frame_2 = pd.DataFrame(dicti, columns= ["animals","cities", "fruits", "colors",'num
frame_2
```

Out[25]:

	animals	cities	fruits	colors	numeri	amperios
0	cat	London	apple	red	100	NaN
1	dog	Paris	banana	blue	115	NaN
2	lion	Tokyo	mango	green	20	NaN
3	tiger	New York	grape	yellow	3	NaN
4	elephant	Berlin	kiwi	purple	69	NaN

In [26]:

```
# frame_2 ['fruits'] selecciona columna frutas ; tbm frame_2.fruits
# frame_2 [['fruits','animals']] selección 2 columnas
frame_2['amperios']= 400
frame_2
```

Out[26]:

	animals	cities	fruits	colors	numeri	amperios
0	cat	London	apple	red	100	400
1	dog	Paris	banana	blue	115	400
2	lion	Tokyo	mango	green	20	400
3	tiger	New York	grape	yellow	3	400
4	elephant	Berlin	kiwi	purple	69	400

In [27]:

```
# crear columna en base al valor de otra
frame_2 ['positivos'] = frame_2['numeri']> 19
frame_2
```

Out[27]:

	animals	cities	fruits	colors	numeri	amperios	positivos
0	cat	London	apple	red	100	400	True
1	dog	Paris	banana	blue	115	400	True
2	lion	Tokyo	mango	green	20	400	True
3	tiger	New York	grape	yellow	3	400	False
4	elephant	Berlin	kiwi	purple	69	400	True

In [28]:

```
frame_3 = frame_2.copy()
frame_3
```

Out[28]:

	animals	cities	fruits	colors	numeri	amperios	positivos
0	cat	London	apple	red	100	400	True
1	dog	Paris	banana	blue	115	400	True
2	lion	Tokyo	mango	green	20	400	True
3	tiger	New York	grape	yellow	3	400	False
4	elephant	Berlin	kiwi	purple	69	400	True

In [29]:

```
frame_3 = frame_3.drop(index = [1]) # quitó fila 1
frame_3
```

Out[29]:

	animals	cities	fruits	colors	numeri	amperios	positivos
0	cat	London	apple	red	100	400	True
2	lion	Tokyo	mango	green	20	400	True
3	tiger	New York	grape	yellow	3	400	False
4	elephant	Berlin	kiwi	purple	69	400	True

In [30]:

```
# buscar por rango
frame_3[frame_3['numeri']> 3]
```

Out[30]:

	animals	cities	fruits	colors	numeri	amperios	positivos
0	cat	London	apple	red	100	400	True
2	lion	Tokyo	mango	green	20	400	True
4	elephant	Berlin	kiwi	purple	69	400	True

In [31]:

```
# buscar por un valor específico
frame_3 [frame_3['fruits'] == 'apple']
```

Out[31]:

	animals	cities	fruits	colors	numeri	amperios	positivos
0	cat	London	apple	red	100	400	True

Indexar, iloc, loc

In [32]:

```
# indexar diferente
indices = ['Sujeto1', 'Sujeto2', 'Sujeto3', 'Sujeto4']
frame_3.index = indices
frame_3
```

Out[32]:

	animals	cities	fruits	colors	numeri	amperios	positivos
Sujeto1	cat	London	apple	red	100	400	True
Sujeto2	lion	Tokyo	mango	green	20	400	True
Sujeto3	tiger	New York	grape	yellow	3	400	False
Sujeto4	elephant	Berlin	kiwi	purple	69	400	True

In [33]:

```
# Devuelve la fila 1 del frame_2 cuya indexacion todavia era numero
# loc is label-based, meaning you use row and column labels to access data.
# access the row with the label 1.
frame_2.loc[1]
```

Out[33]:

```
animals         dog
cities        Paris
fruits       banana
colors         blue
numeri          115
amperios        400
positivos      True
Name: 1, dtype: object
```

In [34]:

```
# Devuelve la fila Sujeto2 del frame_3, si añadir más , seguir dentro del []
frame_3.loc['Sujeto2']
```

Out[34]:

```
animals       lion
cities       Tokyo
fruits       mango
colors       green
numeri          20
amperios       400
positivos     True
Name: Sujeto2, dtype: object
```

In [35]:

```
# Devuelve la fila 1 del frame_2 cuya indexacion todavia era numero
# iloc is integer-based, meaning you use row and column indices to access data.
# access the second row of the DataFrame, since indexing starts at 0
frame_2.iloc[1]
```

Out[35]:
```
animals         dog
cities        Paris
fruits       banana
colors         blue
numeri          115
amperios        400
positivos      True
Name: 1, dtype: object
```

In [36]:
```
# Tbm se puede especificar por filas y columnas a seleccionar
frame_3.loc[['Sujeto2', 'Sujeto3'], ['cities', 'fruits']]
```

Out[36]:

	cities	fruits
Sujeto2	Tokyo	mango
Sujeto3	New York	grape

In [37]:
```
frame_2.iloc[[1,2],[1,2]]
```

Out[37]:

	cities	fruits
1	Paris	banana
2	Tokyo	mango

In [38]:
```
frame_3.loc[frame_3.numeri < 21, ['cities', 'fruits']]
```

Out[38]:

	cities	fruits
Sujeto2	Tokyo	mango
Sujeto3	New York	grape

In [39]:
```
frame_3.numeri.describe()
```

Out[39]:
```
count      4.000000
mean      48.000000
std       44.549598
min        3.000000
25%       15.750000
50%       44.500000
75%       76.750000
max      100.000000
Name: numeri, dtype: float64
```

Personality_dataset

In [40]:
```
datis = pd.read_csv('personality_dataset.csv')
datis.head()
```

Out[40]:

	Time_spent_Alone	Stage_fear	Social_event_attendance	Going_outside	Drained_after_soc
0	4.0	No	4.0	6.0	
1	9.0	Yes	0.0	0.0	
2	9.0	Yes	1.0	2.0	
3	0.0	No	6.0	7.0	
4	3.0	No	9.0	4.0	

In [41]:
```
datis.shape
```

Out[41]: (2900, 8)

In [42]:
```
datis.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2900 entries, 0 to 2899
Data columns (total 8 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Time_spent_Alone           2837 non-null   float64
 1   Stage_fear                 2827 non-null   object 
 2   Social_event_attendance    2838 non-null   float64
 3   Going_outside              2834 non-null   float64
 4   Drained_after_socializing  2848 non-null   object 
 5   Friends_circle_size        2823 non-null   float64
 6   Post_frequency             2835 non-null   float64
 7   Personality                2900 non-null   object 
dtypes: float64(5), object(3)
memory usage: 181.4+ KB
```

In [43]:
```
# Resumen estadístico de las columnas numéricas
datis.describe()
```

Out[43]:

	Time_spent_Alone	Social_event_attendance	Going_outside	Friends_circle_size	Post_f
count	2837.000000	2838.000000	2834.000000	2823.000000	28:
mean	4.505816	3.963354	3.000000	6.268863	
std	3.479192	2.903827	2.247327	4.289693	
min	0.000000	0.000000	0.000000	0.000000	
25%	2.000000	2.000000	1.000000	3.000000	
50%	4.000000	3.000000	3.000000	5.000000	
75%	8.000000	6.000000	5.000000	10.000000	
max	11.000000	10.000000	7.000000	15.000000	

In [44]:
```
datis.columns
```

Out[44]:
```
Index(['Time_spent_Alone', 'Stage_fear', 'Social_event_attendance',
       'Going_outside', 'Drained_after_socializing', 'Friends_circle_size',
       'Post_frequency', 'Personality'],
      dtype='object')
```

In [45]:
```
solo_nums = datis.loc[:, ['Time_spent_Alone', 'Social_event_attendance',
                          'Going_outside', 'Friends_circle_size', 'Post_frequency']]
solo_nums
```

Out[45]:

	Time_spent_Alone	Social_event_attendance	Going_outside	Friends_circle_size	Post_fr
0	4.0	4.0	6.0	13.0	
1	9.0	0.0	0.0	0.0	
2	9.0	1.0	2.0	5.0	
3	0.0	6.0	7.0	14.0	
4	3.0	9.0	4.0	8.0	
...	...	...	...	...	
2895	3.0	7.0	6.0	6.0	
2896	3.0	8.0	3.0	14.0	
2897	4.0	1.0	1.0	4.0	
2898	11.0	1.0	NaN	2.0	
2899	3.0	6.0	6.0	6.0	

2900 rows × 5 columns

In [46]:
```
# Datos nulos x column
datis.isnull().sum()
```

Out[46]:
```
Time_spent_Alone             63
Stage_fear                   73
Social_event_attendance      62
Going_outside                66
Drained_after_socializing    52
Friends_circle_size          77
Post_frequency               65
Personality                   0
dtype: int64
```

In [47]:
```
# Un mejor resumen estadístico general, calculations are performed only on the non-
solo_nums.agg(['mean','std','max'])
```

Out[47]:

	Time_spent_Alone	Social_event_attendance	Going_outside	Friends_circle_size	Post_f
mean	4.505816	3.963354	3.000000	6.268863	
std	3.479192	2.903827	2.247327	4.289693	
max	11.000000	10.000000	7.000000	15.000000	

In [48]:
```
datis['Personality'].value_counts()
```

Out[48]:
```
Personality
Extrovert    1491
Introvert    1409
Name: count, dtype: int64
```

In [49]:
```
# nested value_counts
datis[['Personality', 'Stage_fear']].value_counts()
```

Out[49]:
```
Personality  Stage_fear
Extrovert    No            1338
Introvert    Yes           1299
Extrovert    Yes            111
Introvert    No              79
Name: count, dtype: int64
```

In [50]:
```
# Crear nueva columna con prom de eventos y posts
datis['PromedioEvntPost'] = datis[['Social_event_attendance', 'Post_frequency']].me
datis.head()
```

Out[50]:

	Time_spent_Alone	Stage_fear	Social_event_attendance	Going_outside	Drained_after_soc
0	4.0	No	4.0	6.0	
1	9.0	Yes	0.0	0.0	
2	9.0	Yes	1.0	2.0	
3	0.0	No	6.0	7.0	
4	3.0	No	9.0	4.0	

In [51]:
```
# Crear función de categorización en base a Going_outside
def categorizar(Going_outside):
    if Going_outside >= 5: return 'Outdoorsy'
    else: return 'Cozy'
# Crear columna con nueva categorización
datis['Cozy or Outdoorsy?'] = datis['Going_outside'].apply(categorizar)

datis.head()
```

Out[51]:

	Time_spent_Alone	Stage_fear	Social_event_attendance	Going_outside	Drained_after_soc
0	4.0	No	4.0	6.0	
1	9.0	Yes	0.0	0.0	
2	9.0	Yes	1.0	2.0	
3	0.0	No	6.0	7.0	
4	3.0	No	9.0	4.0	

In [52]:

```
datis.groupby('Personality')[['Time_spent_Alone','Social_event_attendance','Going_o
                              'Friends_circle_size','Post_frequency']].mean()
```

Out[52]:

	Time_spent_Alone	Social_event_attendance	Going_outside	Friends_circle_size	I
Personality					
Extrovert	2.067261	6.016405	4.634615	9.173673	
Introvert	7.080435	1.778909	1.272859	3.196793	

In [53]:

```
# Correlación entre 'Time_spent_Alone'&'Social_event_attendance'
#.corr() method calculates the correlation between numerical columns.
#If a column contains strings or non-numeric data, it will be ignored in the correl
datis[['Time_spent_Alone','Social_event_attendance']].corr()
```

Out[53]:

	Time_spent_Alone	Social_event_attendance
Time_spent_Alone	1.000000	-0.733011
Social_event_attendance	-0.733011	1.000000

Conclusiones generales: Maso 50, 50 extra e introvertidos. 4.5 hrs de media general spended alone, 6.2 amigos cercanos. 111/1491 extrovertidos tienen pánico escénico. Tous las columnas menos el Time_spent_Alone los extroverts tienen medias superiores. Corr Asociación fuerte inversa, más tiempo solo menos eventos sociales attendance.

Merge, Concatenation

In [54]:

```
import random
dados = pd.DataFrame({'participante': ['Valeria', 'Valentina', 'Sofia'], 'dados1':
                      'dados2': np.random.randint(1,7, size=3)})
# np.random.randint(1,7, size=3) generar num aleatorio entre 1 y 6 por cada partici
dados
```

Out[54]:

	participante	dados1	dados2
0	Valeria	6	6
1	Valentina	4	4
2	Sofia	1	1

In [55]:
```
colores = pd.DataFrame ({'participante': ['Valeria', 'Valentina', 'Sofia','Antonio'
                         'color1': ['Azul', 'Verde', 'Amarillo', 'Blanco']})
colores.head(2)
```

Out[55]:

	participante	color1
0	Valeria	Azul
1	Valentina	Verde

In [56]:
```
# Merge fusiona solo la intersección entre dos conjuntos
pd.merge(dados, colores) # Lo mismo que pd.merge(dados, colores, on='participante')
```

Out[56]:

	participante	dados1	dados2	color1
0	Valeria	6	6	Azul
1	Valentina	4	4	Verde
2	Sofia	1	1	Amarillo

In [57]:
```
# Merging DataFrames on columns with different names
colores2 = pd.DataFrame ({'nombre': ['Valeria', 'Valentina', 'Sofia','Antonio'],
                         'color1': ['Azul', 'Verde', 'Amarillo', 'Blanco']})
pd.merge(dados, colores2, left_on='participante', right_on = 'nombre' ) # Los datos
# los de 'colores2' en base a 'nombre'
```

Out[57]:

	participante	dados1	dados2	nombre	color1
0	Valeria	6	6	Valeria	Azul
1	Valentina	4	4	Valentina	Verde
2	Sofia	1	1	Sofia	Amarillo

Equivalente a lo de arriba pero especificando con el parámetro how='inner'. only the rows with matching values in both dataframes will be included in the result. If there is no match, the row will not be included

In [58]:
```
pd.merge(dados, colores2, left_on='participante', right_on = 'nombre', how='inner')
```

Out[58]:

	participante	dados1	dados2	nombre	color1
0	Valeria	6	6	Valeria	Azul
1	Valentina	4	4	Valentina	Verde
2	Sofia	1	1	Sofia	Amarillo

In [59]:

```
# Para Unión. En el caso dados U colores2
# outer merge, also known as a full outer join,
# returns all rows from both dataframes, with NaN values in places where there is n
pd.merge(dados, colores2, left_on='participante', right_on = 'nombre', how='outer')
```

Out[59]:

	participante	dados1	dados2	nombre	color1
0	NaN	NaN	NaN	Antonio	Blanco
1	Sofia	1.0	1.0	Sofia	Amarillo
2	Valentina	4.0	4.0	Valentina	Verde
3	Valeria	6.0	6.0	Valeria	Azul

In [60]:

```
# all rows from the left dataframe (dados in this case)
# will be included in the result, and only the matching rows from the right datafra
# If there is no match, the result will have NaN
pd.merge(dados, colores2, left_on='participante', right_on = 'nombre', how='left')
```

Out[60]:

	participante	dados1	dados2	nombre	color1
0	Valeria	6	6	Valeria	Azul
1	Valentina	4	4	Valentina	Verde
2	Sofia	1	1	Sofia	Amarillo

In [61]:

```
# all rows from the right dataframe (colores2 in this case)
# will be included in the result, and only the matching rows from the left datafram
# If there is no match, the result will have NaN
pd.merge(dados, colores2, left_on='participante', right_on = 'nombre', how='right')
```

Out[61]:

	participante	dados1	dados2	nombre	color1
0	Valeria	6.0	6.0	Valeria	Azul
1	Valentina	4.0	4.0	Valentina	Verde
2	Sofia	1.0	1.0	Sofia	Amarillo
3	NaN	NaN	NaN	Antonio	Blanco

Pandas

In [65]:
```
import pandas as pd
import numpy as np
```

Datos faltantes, series

In [66]:
```
data = pd.Series([1.5, np.nan, 2, None, 4.5, 6.8, 25.3, 98, 65, np.nan])
data
```

Out[66]:
```
0     1.5
1     NaN
2     2.0
3     NaN
4     4.5
5     6.8
6    25.3
7    98.0
8    65.0
9     NaN
dtype: float64
```

In [67]:
```
data.isna() # also .notna
```

Out[67]:
```
0    False
1     True
2    False
3     True
4    False
5    False
6    False
7    False
8    False
9     True
dtype: bool
```

In [68]:
```
data.dropna()
```

Out[68]:
```
0     1.5
2     2.0
4     4.5
5     6.8
6    25.3
7    98.0
8    65.0
dtype: float64
```

In [69]:
```
# .dropna equivalent to
data[data.notna()]
```

Out[69]:
```
0     1.5
2     2.0
4     4.5
5     6.8
6    25.3
7    98.0
8    65.0
dtype: float64
```

Datos faltantes, dfs

In [70]:
```
data = pd.DataFrame([
    [81, 32, 6, 56],
    [np.nan,np.nan,np.nan, np.nan],
    [0, np.nan, np.nan, 1],
    [np.nan, 6.5, 8, 14],
    [np.nan, 45, 8, 68]
])
data
```

Out[70]:

	0	1	2	3
0	81.0	32.0	6.0	56.0
1	NaN	NaN	NaN	NaN
2	0.0	NaN	NaN	1.0
3	NaN	6.5	8.0	14.0
4	NaN	45.0	8.0	68.0

In [71]:
```
data.dropna()
```

Out[71]:

	0	1	2	3
0	81.0	32.0	6.0	56.0

In [72]:
```
data.dropna(how='all')
```

Out[72]:

	0	1	2	3
0	81.0	32.0	6.0	56.0
2	0.0	NaN	NaN	1.0
3	NaN	6.5	8.0	14.0
4	NaN	45.0	8.0	68.0

In [73]:
```
data[4] = np.nan
data
```

Out[73]:

	0	1	2	3	4
0	81.0	32.0	6.0	56.0	NaN
1	NaN	NaN	NaN	NaN	NaN
2	0.0	NaN	NaN	1.0	NaN
3	NaN	6.5	8.0	14.0	NaN
4	NaN	45.0	8.0	68.0	NaN

In [74]:
```
data.dropna(axis='columns', how='all')
```

Out[74]:

	0	1	2	3
0	81.0	32.0	6.0	56.0
1	NaN	NaN	NaN	NaN
2	0.0	NaN	NaN	1.0
3	NaN	6.5	8.0	14.0
4	NaN	45.0	8.0	68.0

In [75]:
```
data.fillna(-23)
```

Out[75]:

	0	1	2	3	4
0	81.0	32.0	6.0	56.0	-23.0
1	-23.0	-23.0	-23.0	-23.0	-23.0
2	0.0	-23.0	-23.0	1.0	-23.0
3	-23.0	6.5	8.0	14.0	-23.0
4	-23.0	45.0	8.0	68.0	-23.0

In [76]:
```
data.fillna({1:-23, 2:-24, 3:-25, 4:-26})
```

Out[76]:

	0	1	2	3	4
0	81.0	32.0	6.0	56.0	-26.0
1	NaN	-23.0	-24.0	-25.0	-26.0
2	0.0	-23.0	-24.0	1.0	-26.0
3	NaN	6.5	8.0	14.0	-26.0
4	NaN	45.0	8.0	68.0	-26.0

In [77]:
```
datai = data[3]
```

```
datai
```

Out[77]:
```
0    56.0
1     NaN
2     1.0
3    14.0
4    68.0
Name: 3, dtype: float64
```

In [78]:
```
datai.mean() # sum/4, i.e. sin contar el nan
```

Out[78]:
```
np.float64(34.75)
```

In [79]:
```
datai = datai.fillna(0)
datai
```

Out[79]:
```
0    56.0
1     0.0
2     1.0
3    14.0
4    68.0
Name: 3, dtype: float64
```

In [80]:
```
datai.mean() # sum/5, i.e. contando en nan convertido en 0
```

Out[80]:
```
np.float64(27.8)
```

In [81]:
```
datai_2 = data[3]
datai_2
```

Out[81]:
```
0    56.0
1     NaN
2     1.0
3    14.0
4    68.0
Name: 3, dtype: float64
```

In [82]:
```
datai_2.mean() # sum/4, i.e. sin contar el nan
```

Out[82]:
```
np.float64(34.75)
```

In [83]:
```
datai_2.fillna(datai_2.mean())
```

Out[83]:
```
0    56.00
1    34.75
2     1.00
3    14.00
4    68.00
Name: 3, dtype: float64
```

In [84]:
```
datai_2.mean()
# nan convertido en la media de los nonan , no afecta la media. cuando nan se convi
```

Out[84]:
```
np.float64(34.75)
```

data transformation

In [85]:
```
data = pd.DataFrame({
    'Cl1': [0, 1, 0, 1, 0, 1, 0, 1],
    'Cl2': ['a', 'b', 'b', 'b', 'd', 'd', 'b', 'b']
})
data
```

Out[85]:

	Cl1	Cl2
0	0	a
1	1	b
2	0	b
3	1	b
4	0	d
5	1	d
6	0	b
7	1	b

In [86]:
```
data.duplicated()
```

Out[86]:
```
0    False
1    False
2    False
3     True
4    False
5    False
6     True
7     True
dtype: bool
```

In [87]:
```
data.drop_duplicates()
```

Out[87]:

	Cl1	Cl2
0	0	a
1	1	b
2	0	b
4	0	d
5	1	d

In [88]:
```
data1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'], 'data1': range(7)

data2 = pd.DataFrame({
```

```
    'group_val': [5, 6.7],
}, index=['a', 'b'])
```

In [89]:

```
data1.head(7)
```

Out[89]:

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	a	5
6	b	6

In [90]:

```
data2.head()
```

Out[90]:

	group_val
a	5.0
b	6.7

In [91]:

```
pd.merge(data1, data2, left_on='key', right_index=True)
# the left_on argument specifies the column in the left DataFrame (data1 in this ca
# In your example, left_on='key' means that the merge will use the 'key' column fro
# The right_index=True argument means that the merge will use the index of the righ
# instead of a specific column
```

Out[91]:

	key	data1	group_val
0	b	0	6.7
1	b	1	6.7
2	a	2	5.0
4	a	4	5.0
5	a	5	5.0
6	b	6	6.7

In [92]:

```
pd.merge(data1, data2, left_on='key', right_index=True, how='outer')
```

Out[92]:

	key	data1	group_val
2	a	2	5.0
4	a	4	5.0
5	a	5	5.0
0	b	0	6.7
1	b	1	6.7
6	b	6	6.7
3	c	3	NaN

In [93]:
```
pd.merge(data1, data2, left_on='key', right_index=True, how='inner')
```

Out[93]:

	key	data1	group_val
0	b	0	6.7
1	b	1	6.7
2	a	2	5.0
4	a	4	5.0
5	a	5	5.0
6	b	6	6.7

In [94]:
```
data1.join(data2, on='key') # data1 join with data 2 en base a la columna 'key'. De
# automáticamente que los values de la columna key son los index de data2
```

Out[94]:

	key	data1	group_val
0	b	0	6.7
1	b	1	6.7
2	a	2	5.0
3	c	3	NaN
4	a	4	5.0
5	a	5	5.0
6	b	6	6.7

In [95]:
```
arr = np.arange(12).reshape((3, 4))
arr
```

Out[95]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [96]:
```
np.concatenate([arr, arr], axis=1)
# the axis parameter is used in various functions to specify the direction of the o
# When axis=1, the operation is performed across columns (horizontally).
```

Out[96]:
```
array([[ 0,  1,  2,  3,  0,  1,  2,  3],
       [ 4,  5,  6,  7,  4,  5,  6,  7],
       [ 8,  9, 10, 11,  8,  9, 10, 11]])
```

In [97]:
```
np.concatenate([arr, arr], axis=0)
```

Out[97]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [98]:
```
df1 = pd.DataFrame(np.arange(6).reshape(3,2), index=['a', 'b', 'c'], columns=['one'
df2 = pd.DataFrame(5 + np.arange(4).reshape(2,2), index=['a', 'c'], columns=['three
```

In [99]:
```
df1.head()
```

Out[99]:

	one	two
a	0	1
b	2	3
c	4	5

In [100…
```
df2.head()
```

Out[100…

	three	four
a	5	6
c	7	8

In [101…
```
pd.concat([df1, df2], axis=1, keys=['level1', 'level2'])
```

Out[101…

	level1		level2	
	one	two	three	four
a	0	1	5.0	6.0
b	2	3	NaN	NaN
c	4	5	7.0	8.0

In [102…

```
pd.concat([df1, df2], axis=0)
```

Out[102…

	one	two	three	four
a	0.0	1.0	NaN	NaN
b	2.0	3.0	NaN	NaN
c	4.0	5.0	NaN	NaN
a	NaN	NaN	5.0	6.0
c	NaN	NaN	7.0	8.0

Reshape h_indx

In [103…

```
data = pd.DataFrame(
    np.random.randint(0,10,(2,3)), # array 2x3 con random enteros etre 0 y 9
    index=pd.Index(['P1', 'P2'], name='participant'),
    columns=pd.Index(['Manzana', 'Pera', 'Fresa'], name='fruit')
)

data.head()
```

Out[103…

fruit	Manzana	Pera	Fresa
participant			
P1	5	6	5
P2	7	2	3

In [104…

```
data = data.stack()
# a cada fila, se le crean 3 filas que antes eran las columnas
data
```

Out[104…

```
participant  fruit
P1           Manzana    5
             Pera       6
             Fresa      5
P2           Manzana    7
             Pera       2
             Fresa      3
dtype: int32
```

In [105…
```
data = data.unstack()
data
```

Out[105…

fruit	Manzana	Pera	Fresa
participant			
P1	5	6	5
P2	7	2	3

Pivot

In [106…
```
df = pd.DataFrame({
    'Mes': ['Ene', 'Feb', 'Marz', 'Abril', 'May', 'Jun'],
    'Sí/No': [0, 0, 1, 0, 1, 1],
    'Producto': ['A', 'B', 'B', 'C', 'A', 'B'],
    'Revenue': [120, 140, 60, 300, 20, 800]
})
df.head()
```

Out[106…

	Mes	Sí/No	Producto	Revenue
0	Ene	0	A	120
1	Feb	0	B	140
2	Marz	1	B	60
3	Abril	0	C	300
4	May	1	A	20

In [107…
```
df = df.set_index('Mes')
df.head()
```

Out[107…

	Sí/No	Producto	Revenue
Mes			
Ene	0	A	120
Feb	0	B	140
Marz	1	B	60
Abril	0	C	300
May	1	A	20

In [108…
```
df_long = df.stack().reset_index().rename(columns={0:'value'})
# .rename(columns={0:'value'}) sin esta parte arriba de la columna aparece '0', est
df_long
```

Out[108…

	Mes	level_1	value
0	Ene	Sí/No	0
1	Ene	Producto	A
2	Ene	Revenue	120
3	Feb	Sí/No	0
4	Feb	Producto	B
5	Feb	Revenue	140
6	Marz	Sí/No	1
7	Marz	Producto	B
8	Marz	Revenue	60
9	Abril	Sí/No	0
10	Abril	Producto	C
11	Abril	Revenue	300
12	May	Sí/No	1
13	May	Producto	A
14	May	Revenue	20
15	Jun	Sí/No	1
16	Jun	Producto	B
17	Jun	Revenue	800

In [109…

```
pivot_data = df_long.pivot(index='Mes', columns='level_1', values='value')
pivot_data.head()
```

Out[109…

level_1	Producto	Revenue	Sí/No
Mes			
Abril	C	300	0
Ene	A	120	0
Feb	B	140	0
Jun	B	800	1
Marz	B	60	1