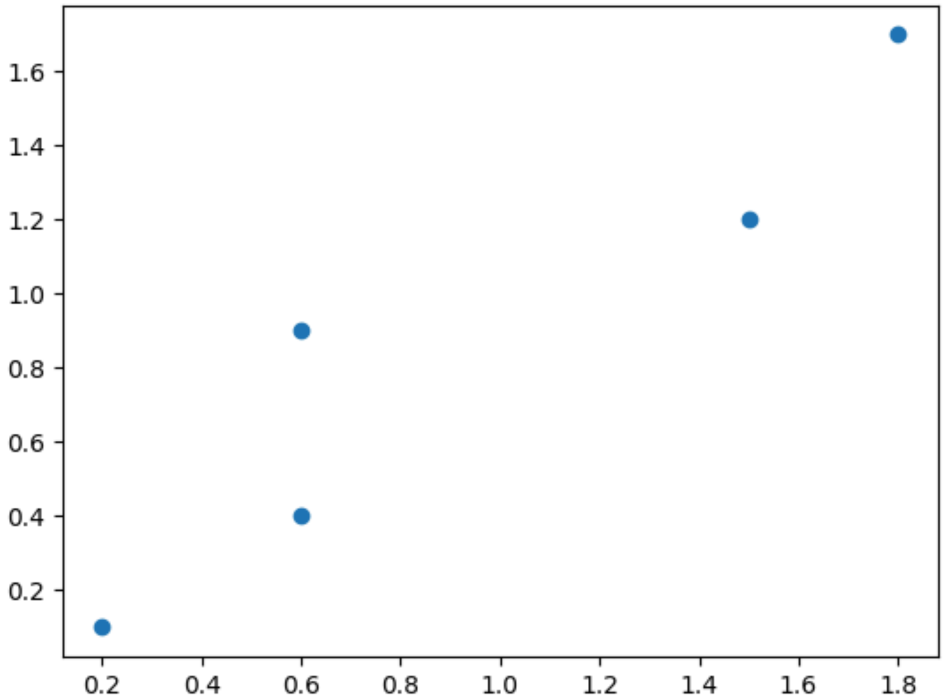
```
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.array([0.2,0.6,0.6,1.5,1.8])
y = np.array([0.1,0.4,0.9,1.2,1.7])
plt.scatter(x,y)
```

```
<matplotlib.collections.PathCollection at 0x7f7c812b5210>
```




```
#from sklearn.model_selection import train_test_split
#X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.6)
X_train,y_train = np.array([0.6,1.8]), np.array([0.4,1.7])
X_test,y_test = np.array([0.2,0.6,1.5]), np.array([0.1,0.9,1.2])
X_train.shape
```

```
(2,)
```

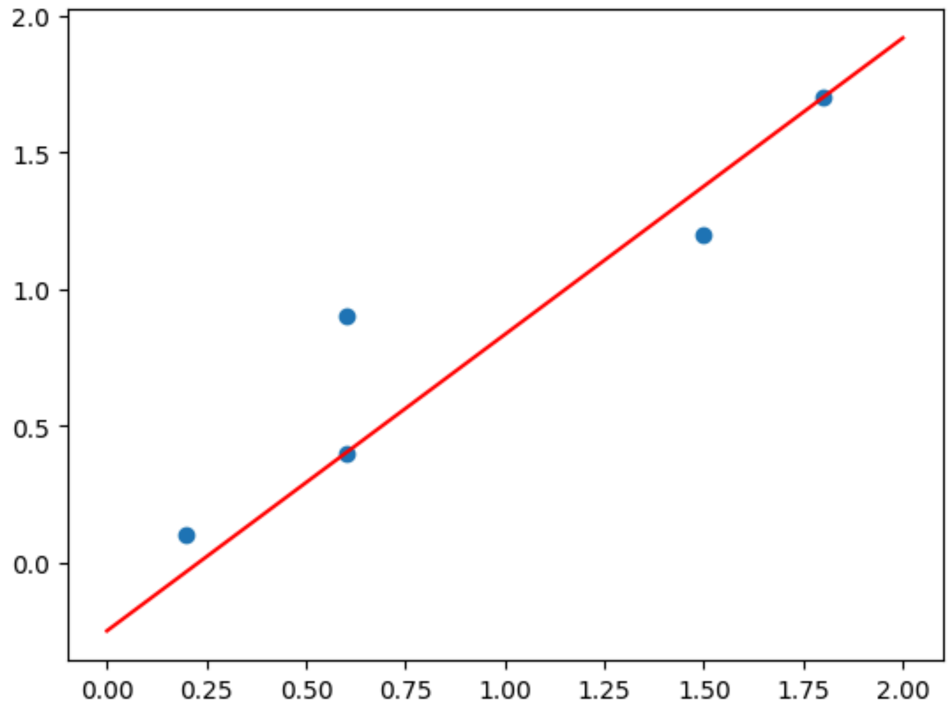
```
from sklearn.linear_model import LinearRegression
X_train.resize(len(X_train),1)
y_train.resize(len(y_train),1)
lr = LinearRegression()
lr.fit(X_train,y_train)
coefs = lr.coef_[0]
intercept = lr.intercept_[0]
print("y = {:.4f} + {:.4f}x".format(intercept,coefs[0]))
X_train.shape
```

```
y = -0.2500 + 1.0833x
(2, 1)
```

```
xx = np.linspace(0,2,2)
yy = lr.predict(xx.reshape(len(xx),1))
```

```
plt.scatter(x,y)
```

```
[<matplotlib.lines.Line2D at 0x7f7bcb4e2d10>]
```




```
from sklearn.metrics import mean_squared_error as mse
mse1 = mse(y_test, lr.predict(X_test.reshape(len(X_test),1)))
print('Error : {}'.format(mse1))
```

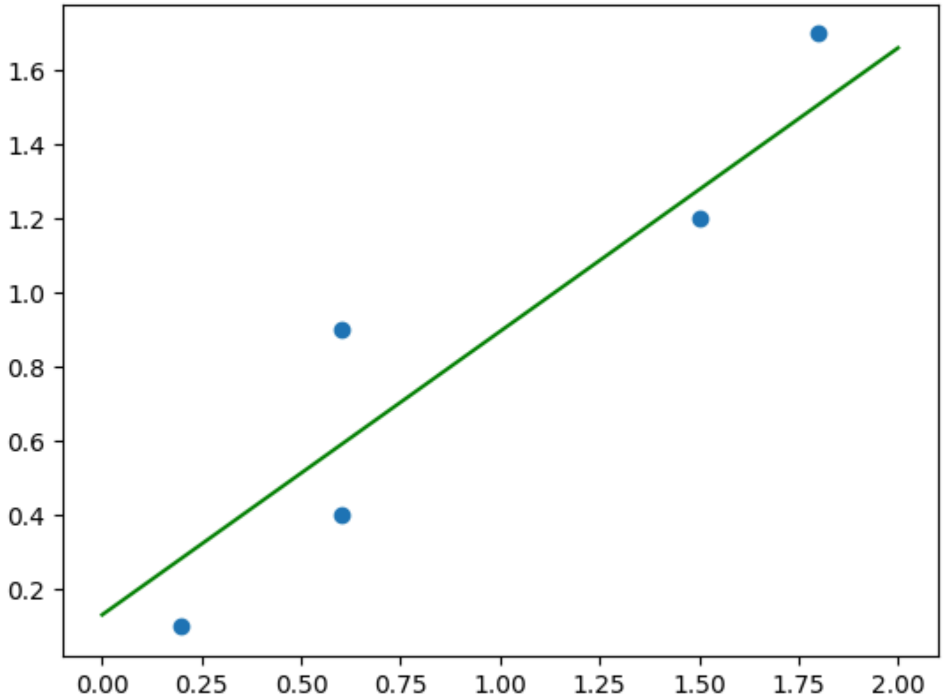
```
Error : 0.09946759259259248
```

```
# Regularización Ridge
from sklearn.linear_model import Ridge
ridge = Ridge(alpha = 0.3)
ridge.fit(X_train,y_train)
coefs = ridge.coef_[0]
intercept = ridge.intercept_[0]
print("y = {:.4f} + {:.4f}x".format(intercept,coefs[0]))
```

```
y = 0.1324 + 0.7647x
```

```
xx = np.linspace(0.0,2,2)
yy2 = ridge.predict(xx.reshape(len(xx),1))
plt.scatter(x,y)
plt.plot(xx,yy2,c='g')
```

```
[<matplotlib.lines.Line2D at 0x7f7bcae47050>]
```




```
from sklearn.metrics import mean_squared_error as mse
mse2 = mse(y_test, ridge.predict(X_test.reshape(len(X_test), 1)))
print('Error : {}'.format(mse2))
```

```
Error : 0.04533737024221454
```

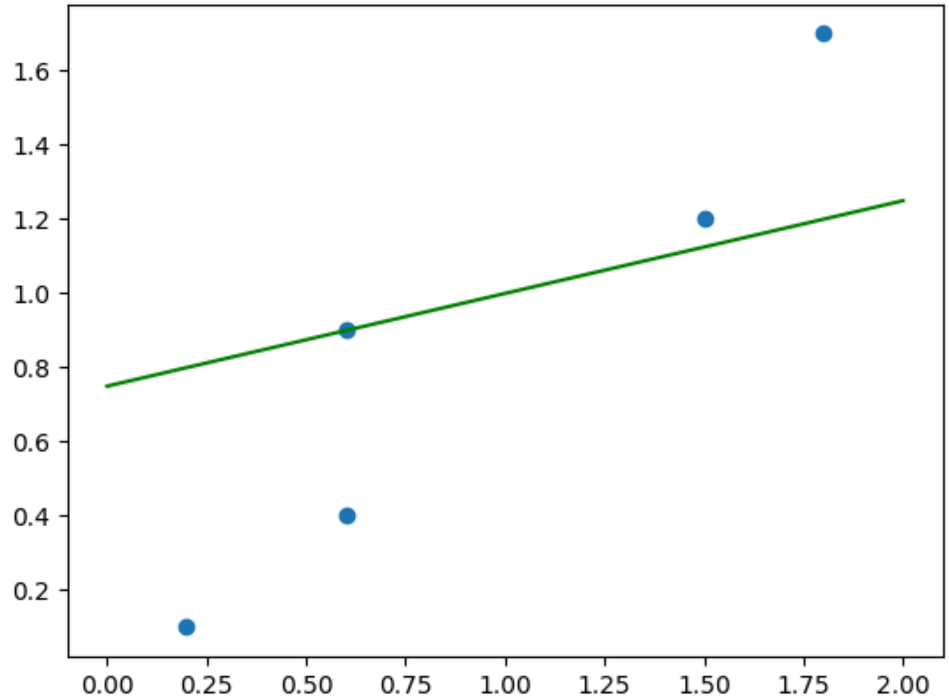
```

```

```
# Regularización Lasso
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.3)
lasso.fit(X_train,y_train)
coefs = lasso.coef_[0]
intercept = lasso.intercept_[0]
print("y = {:.4f} + {:.4f}x".format(intercept,coefs))
coefs
```

```
y = 0.7500 + 0.2500x
np.float64(0.25000000000000006)
```

```
xx = np.linspace(0.0,2,2)
yy3 = lasso.predict(xx.reshape(len(xx),1))
plt.scatter(x,y)
plt.plot(xx,yy3,c='g')
```

```
[<matplotlib.lines.Line2D at 0x7f7bcae2e450>]
```




```
from sklearn.metrics import mean_squared_error as mse
mse2 = mse(y_test, lasso.predict(X_test.reshape(len(X_test), 1)))
print('Error : {}'.format(mse2))
```

```
Error : 0.16520833333333337
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

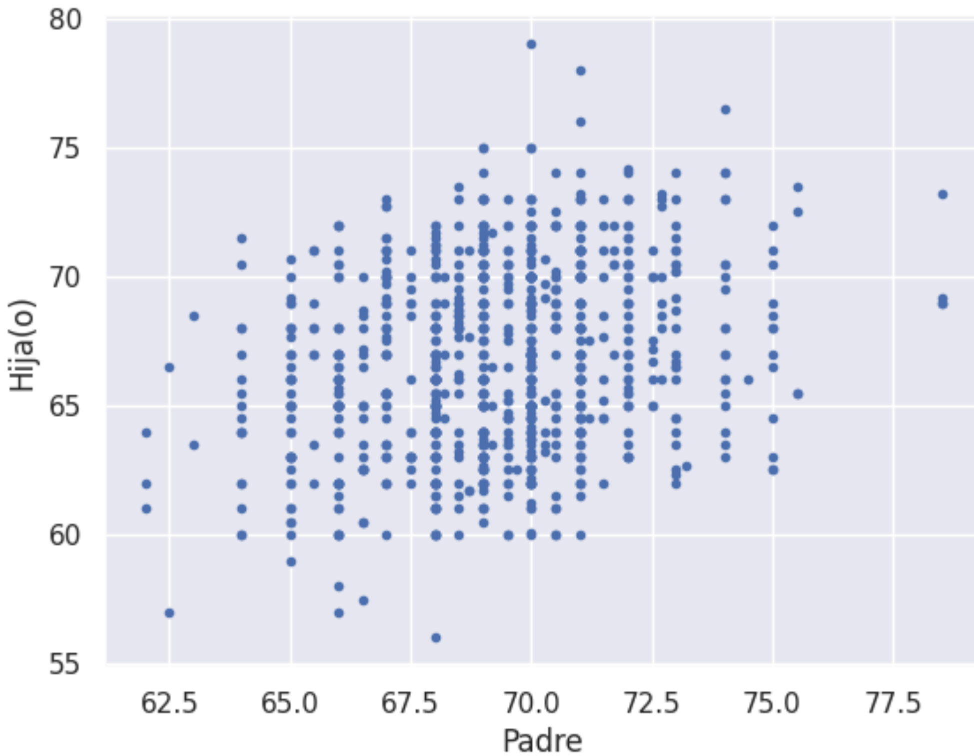
```
df = pd.read_csv('families.csv')
df.tail(2)
```

	family	father	mother	midparentHeight	children	childNum	gender	childHeight
932	204	62.5	63.0	65.27	2	1	male	66.5
933	204	62.5	63.0	65.27	2	2	female	57.0

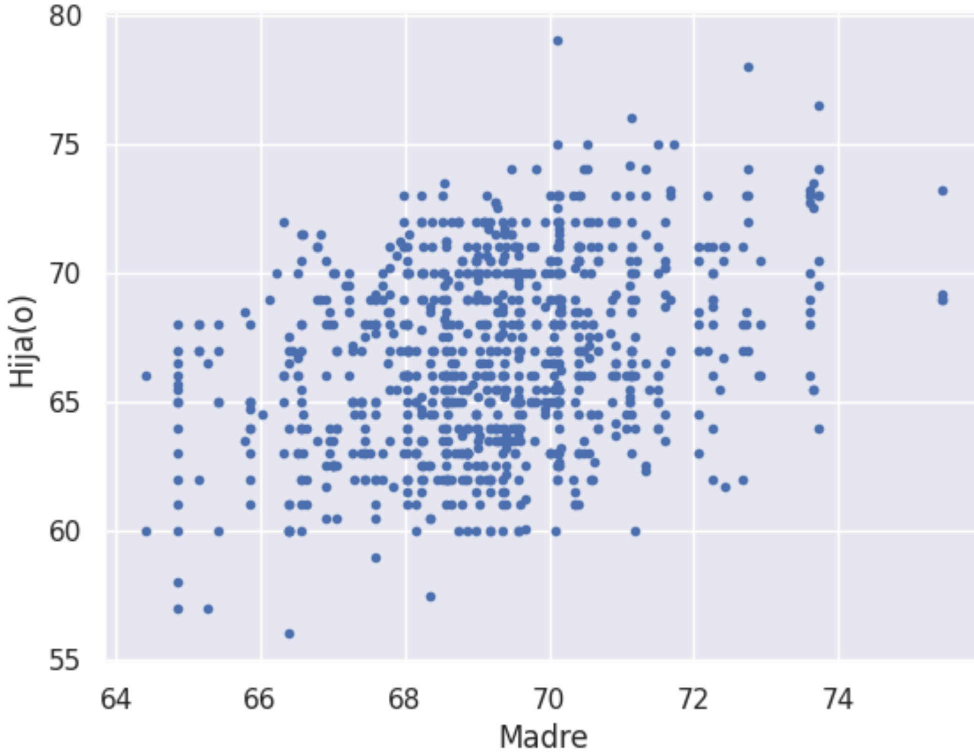
```
#import plotly.express as px
#fig = px.scatter_3d(df, x='father', y='mother', z='childHeight')
#fig.show()
```

```
X = df.iloc[:, [1,3]].values
y = df.iloc[:, 7].values
```

```
plt.plot(X[:,0], y, 'b.')
plt.xlabel('Padre')
plt.ylabel('Hija(o)')
plt.show()
```




```
plt.plot(X[:,1], y, 'b.')
plt.xlabel('Madre')
plt.ylabel('Hija(o)')
plt.show()
```




```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
```

```
# conjunto de entrenamiento y pruebas
X_train,X_test,y_train,y_test=train_test_split(X, y, test_size=0.2,
                                                random_state=2)
```

```
# modelo lineaal y su rendimiento
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
print('  R2 : ',r2_score(y_test, y_pred))
print('RMSE : ',np.sqrt(mean_squared_error(y_test, y_pred)))
```

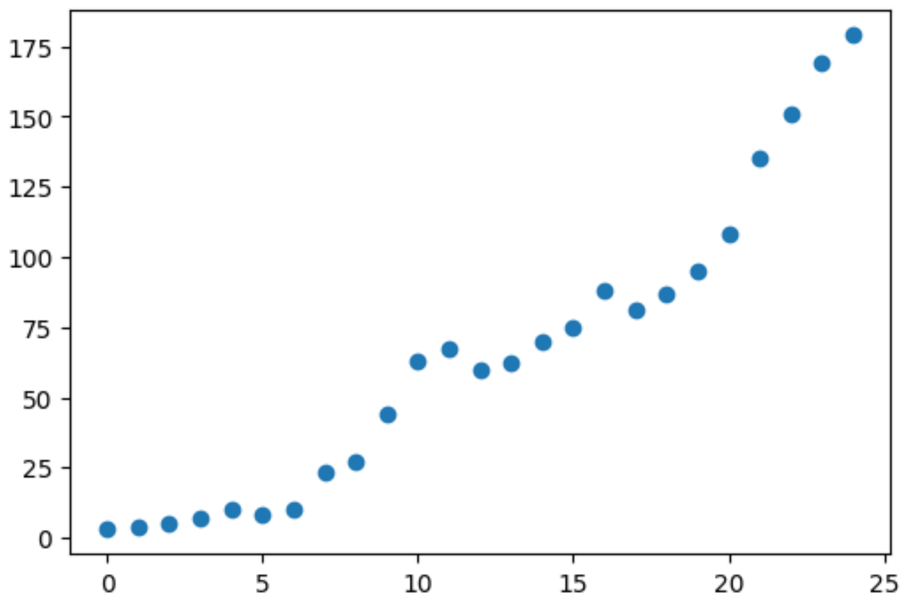
```
  R2 :  0.05375000577834388
RMSE :  3.3605503188369497
```

```
print(lr.coef_)
print(lr.intercept_)
```

```
[0.07286771 0.55935083]
23.14535713417917
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
y = [3, 4, 5, 7, 10, 8, 10, 23, 27, 44, 63, 67,  60, 62, 70, 75, 88, 81,
     87, 95, 108, 135, 151, 169, 179]
x = np.arange(len(y))
plt.figure(figsize=(6,4))
plt.scatter(x, y)
plt.show()
```




```
# Características polinomiales
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2, include_bias=False) # bias en el modelo lineal
```

```
# Generación de características nuevas
poly_features = poly.fit_transform(x.reshape(-1, 1))
poly_features.shape
```

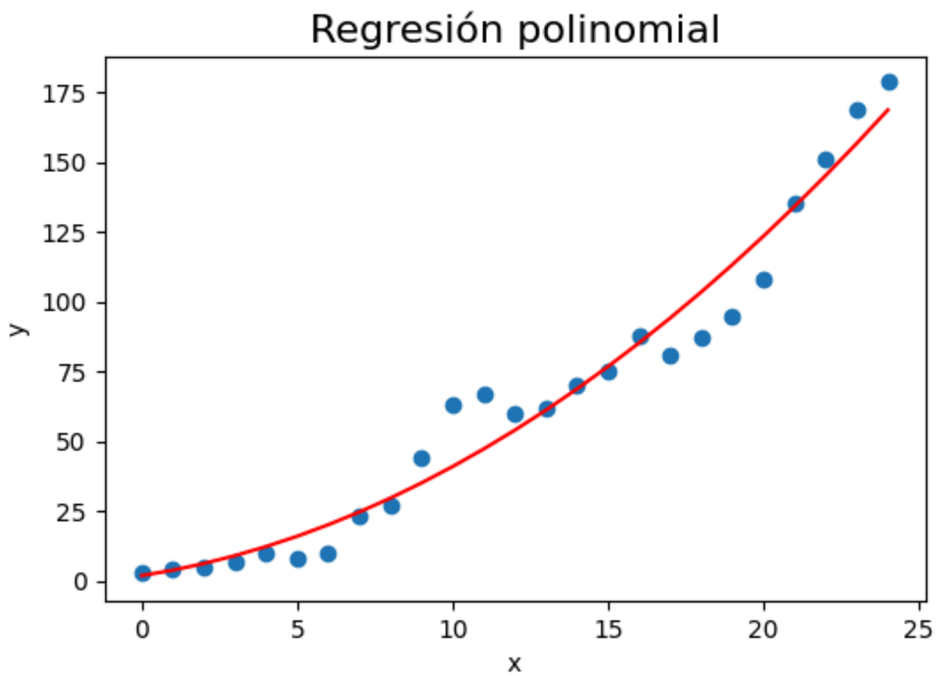
```
(25, 2)
```

```
# Modelo lineal
from sklearn.linear_model import LinearRegression
poly_reg_model = LinearRegression()
```

```
# Ajuste y predicción
poly_reg_model.fit(poly_features, y)
y_pred = poly_reg_model.predict(poly_features)
```

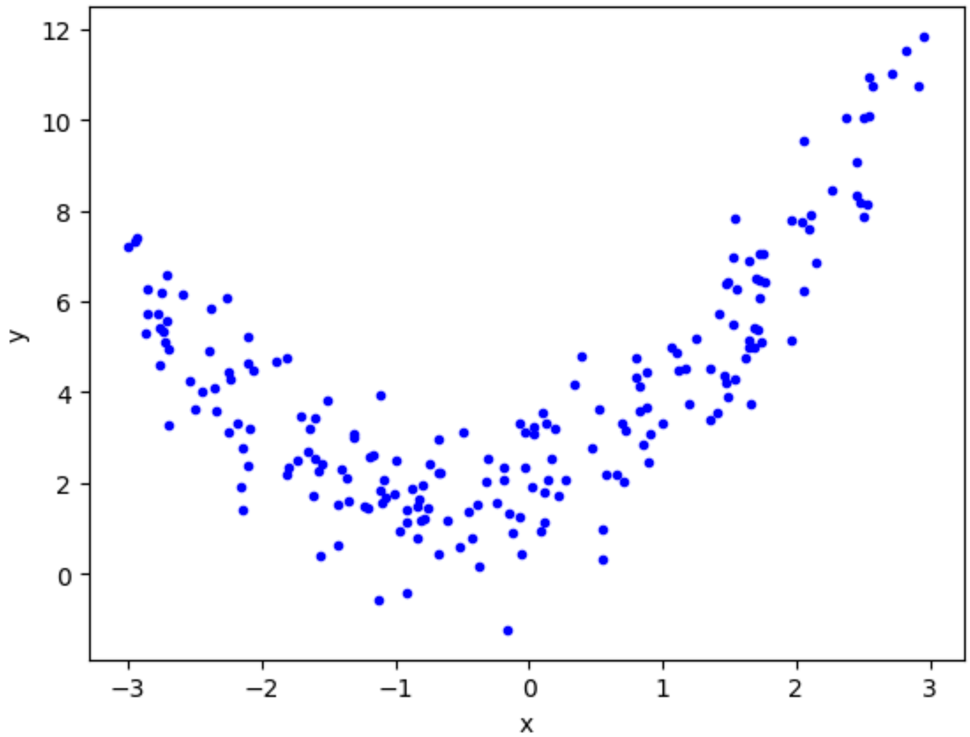
```
plt.figure(figsize=(6, 4))
plt.scatter(x, y)
plt.plot(x, y_pred, c="red")
plt.title('Regresión polinomial', size=16)
```

```
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```




```
# Determinar el grado del polinomio
```

```
import numpy as np
import matplotlib.pyplot as plt
x = 6 * np.random.rand(200, 1) - 3
y = 0.8*x**2 + 0.9*x + 2 + np.random.randn(200, 1)
#ecuación -> y = 0.8x^2 + 0.9x + 2
plt.plot(x, y, 'b.')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



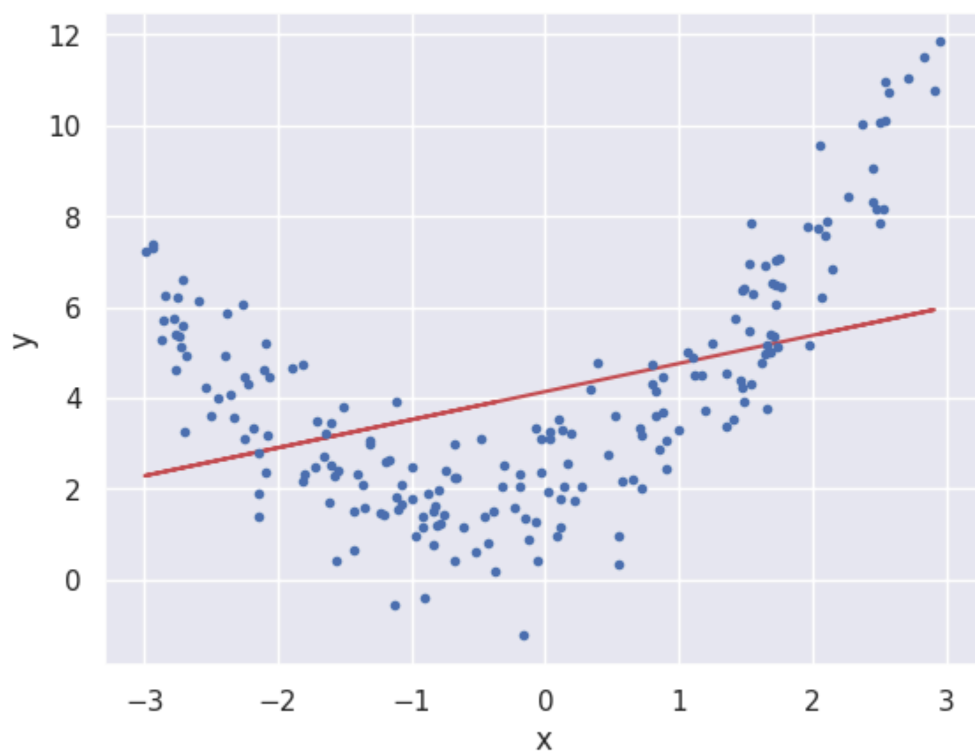

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score, mean_squared_error
import seaborn as sns
sns.set()
```

```
# conjunto de entrenamiento y pruebas
x_train,x_test,y_train,y_test=train_test_split(x, y, test_size=0.2,
                                               random_state=2)
```

```
# modelo lineaal y su rendimiento
lr = LinearRegression()
lr.fit(x_train, y_train)
y_pred = lr.predict(x_test)
print('  R2 : ',r2_score(y_test, y_pred))
print('RMSE : ',np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
  R2 :  0.348786016337134
RMSE :  2.182948912340432
```

```
plt.plot(x_train, lr.predict(x_train), color='r')
plt.plot(x, y, 'b.')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```




```
# Polinomio de grado 2
poly = PolynomialFeatures(degree=2, include_bias=False)
x_train_poly = poly.fit_transform(x_train)
x_test_poly = poly.transform(x_test)
lr = LinearRegression()
lr.fit(x_train_poly, y_train)
y_pred = lr.predict(x_test_poly)
print('  R2 : ',r2_score(y_test, y_pred))
print('RMSE : ',np.sqrt(mean_squared_error(y_test, y_pred)))
# Visualización
"""
x_new = np.linspace(-3, 3, 200).reshape(200, 1)
x_new_poly = poly.transform(x_new)
y_new_pred = lr.predict(x_new_poly)
plt.plot(x_new, y_new_pred, "r", linewidth=2, label='Curva de ajuste')
plt.plot(x_train, y_train, "b.",label='Entrenamiento')
plt.plot(x_test, y_test, "g*",label='Pruebas')
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
"""
```

```
  R2 :  0.9177735823112784
RMSE :  0.7756885809730303
'\nx_new = np.linspace(-3, 3, 200).reshape(200, 1)\nx_new_poly =
poly.transform(x_new)\ny_new_pred = lr.predict(x_new_poly)\nplt.plot(x_new, y_new_pred,
"r", linewidth=2, label=\'Curva de ajuste\')\nplt.plot(x_train, y_train,
"b.",label=\'Entrenamiento\')\nplt.plot(x_test, y_test,
"g*",label=\'Pruebas\')\nplt.xlabel("x")\nplt.ylabel("y")\nplt.legend()\nplt.show()\n'
```

```
# Polinomio de grado 2
poly = PolynomialFeatures(degree=2, include_bias=False)
```

```
x_train_poly = poly.fit_transform(x_train)
x_test_poly = poly.transform(x_test)
lr = LinearRegression()
lr.fit(x_train_poly, y_train)
y_pred = lr.predict(x_test_poly)
print('  R2 : ',r2_score(y_test, y_pred))
print('RMSE : ',np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
  R2 :  0.9177735823112784
RMSE :  0.7756885809730303
```

```
# Polinomio de grado 3
poly = PolynomialFeatures(degree=3, include_bias=False)
x_train_poly = poly.fit_transform(x_train)
x_test_poly = poly.transform(x_test)
lr = LinearRegression()
lr.fit(x_train_poly, y_train)
y_pred = lr.predict(x_test_poly)
print('  R2 : ',r2_score(y_test, y_pred))
print('RMSE : ',np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
  R2 :  0.9158615394486871
RMSE :  0.7846554471212009
```

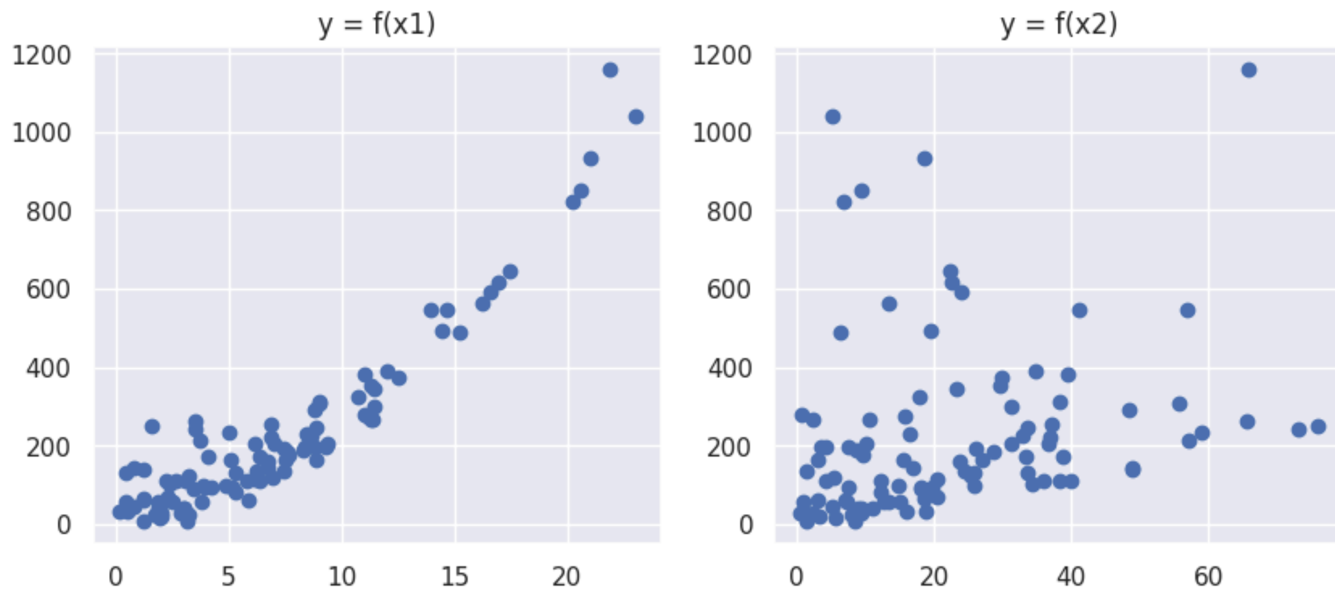
```
# Polinomio de grado 10
poly = PolynomialFeatures(degree=10, include_bias=False)
x_train_poly = poly.fit_transform(x_train)
x_test_poly = poly.transform(x_test)
lr = LinearRegression()
lr.fit(x_train_poly, y_train)
y_pred = lr.predict(x_test_poly)
print('  R2 : ',r2_score(y_test, y_pred))
print('RMSE : ',np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
  R2 :  0.9008513193120585
RMSE :  0.8517755762987763
```

```
# PolyReg múltiples características
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Generador de datos
np.random.seed(1)
x1 = np.absolute(np.random.randn(100, 1) * 10)
x2 = np.absolute(np.random.randn(100, 1) * 30)
y = 2*x1**2 + 3*x2 + 2 + np.random.randn(100, 1)*20
#ecuación -> y = 2*x1^2 + 3x2 + 2
```

```
# Gráficas
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))
axes[0].scatter(x1, y)
axes[1].scatter(x2, y)
axes[0].set_title("y = f(x1)")
axes[1].set_title("y = f(x2)")
plt.show()
```




```
# DataFrame de los datos
df = pd.DataFrame({'x1':x1.reshape(100,), 'x2':x2.reshape(100,),
                   'y':y.reshape(100,)}, index=range(0,100))
df.tail(2)
```

	x1	x2	y
98	6.200008	24.328550	132.757355
99	6.980320	31.333263	205.167741

```
# Características polinomiales y conjuntos de entrenamiento/prueba
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
X, y = df[['x1', 'x2']], df['y']
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly_features = poly.fit_transform(X)
X_poly_features.shape
```

```
(100, 5)
```

```
# Entrenamiento y pruebas
X_train,X_test,y_train,y_test=train_test_split(X_poly_features, y,
                                               test_size=0.3, random_state=42)
```

```
# Modelo lineal
from sklearn.linear_model import LinearRegression
```

```python
poly_reg_model = LinearRegression()
```

```python
# Ajuste, predicción y evaluación
poly_reg_model.fit(X_train, y_train)
y_pred = poly_reg_model.predict(X_test)
from sklearn.metrics import mean_squared_error
poly_reg_rmse = np.sqrt(mean_squared_error(y_test, y_pred))
poly_reg_rmse
```

```
np.float64(20.937707839078698)
```

```python
""" Para dos variables independientes se convierte en:
indep + a·x1 + b·x2 + c·x1^2+ d·x1·x2 + e·x2^2
#ecuación -> y = 2*x1^2 + 3x2 + 2
con: indep = intercept
a = coefs[0]
...
e = coefs[4]
"""
print(poly_reg_model.intercept_, poly_reg_model.coef_)
```

```
14.123436038978923 [0.61945509 1.9140045  1.89905813 0.0207338  0.01300394]
```

```python
# Características polinomiales y conjuntos de entrenamiento/prueba
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
X, y = df[['x1', 'x2']], df['y']
```

```python
# Entrenamiento y pruebas
X_train,X_test,y_train,y_test=train_test_split(X, y,
                                               test_size=0.3, random_state=42)
```

```python
# Vs modelo lineal puro
lin_reg_model = LinearRegression()
lin_reg_model.fit(X_train, y_train)
lin_reg_y_pred = lin_reg_model.predict(X_test)
lin_reg_rmse = np.sqrt(mean_squared_error(y_test, lin_reg_y_pred))
lin_reg_rmse
```

```
np.float64(62.302487453878506)
```

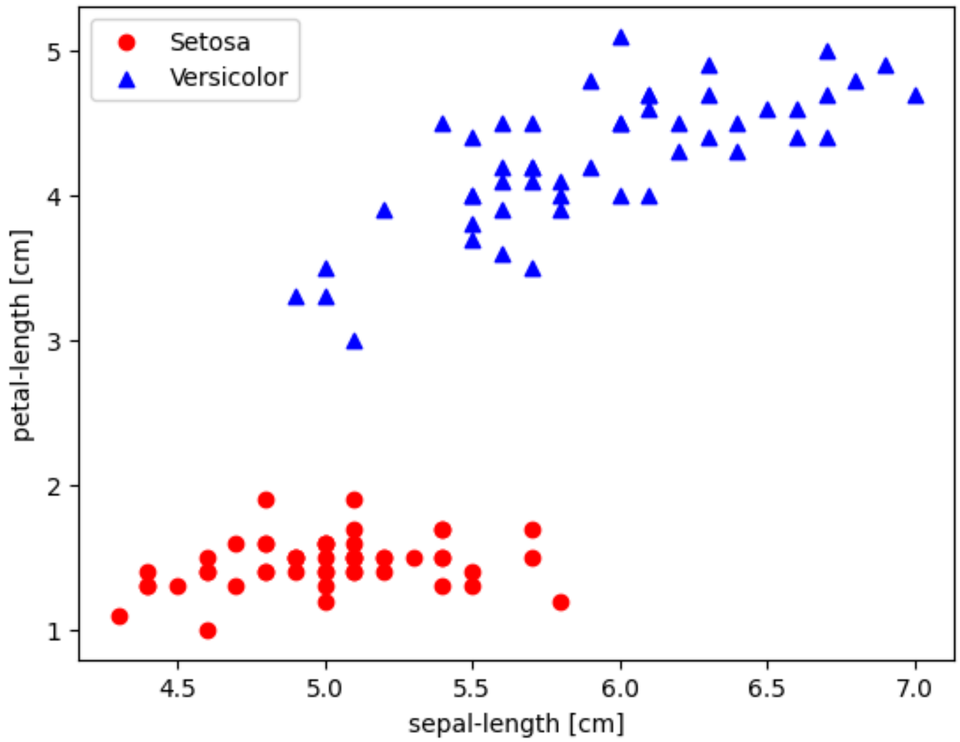
```
import pandas as pd
import numpy as np
# Conjunto Iris
# https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
df = pd.read_csv('https://bit.ly/38XWXS4', header=None)
df.tail(2)
```

	0	1	2	3	4
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

```
# sepal length & petal length de las primeras 100 entradas
# Iris-setosa & Iris-versicolor
X = df.iloc[:100, [0,2]].values
# Cambiando etiquetas de texto a números
y = df.iloc[:100, 4].values
y = np.array(np.where(y=='Iris-setosa',-1,1))
y
```

```
array([-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1])
```

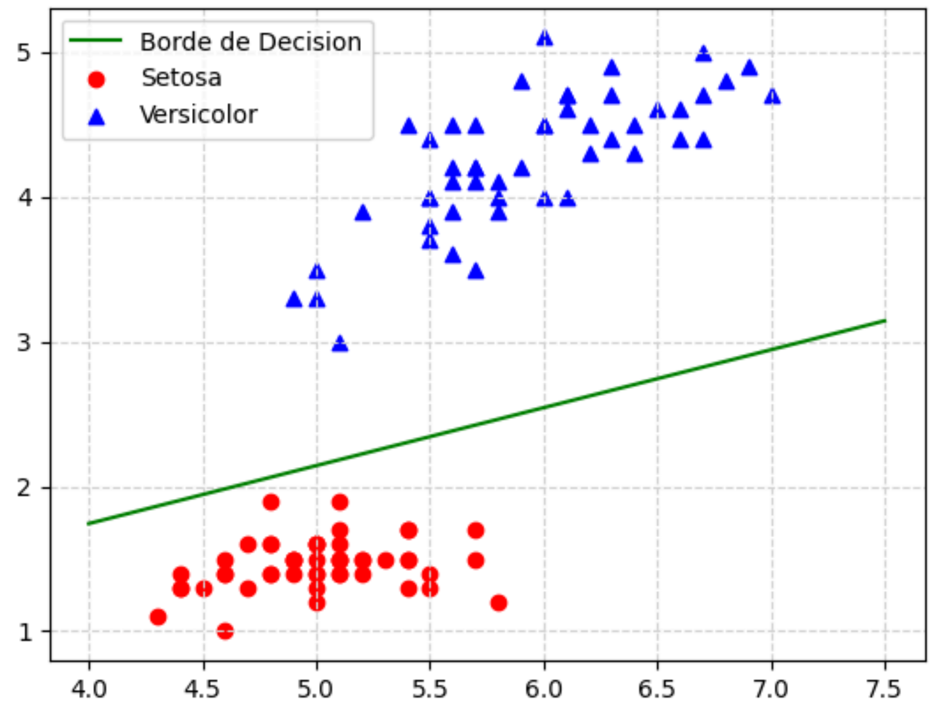
```
import matplotlib.pyplot as plt
plt.scatter(X[:50,0],X[:50,1],color='red',marker='o',label='Setosa')
plt.scatter(X[50:,0],X[50:,1],color='blue',marker='^',label='Versicolor')
plt.xlabel('sepal-length [cm]')
plt.ylabel('petal-length [cm]')
plt.legend(loc='upper left')
plt.show()
```




```
# Perceptrón
from sklearn.linear_model import Perceptron
ppn = Perceptron(max_iter=40, eta0=0.1, random_state=1)
ppn.fit(X, y)
print(ppn.intercept_, ppn.coef_)
```

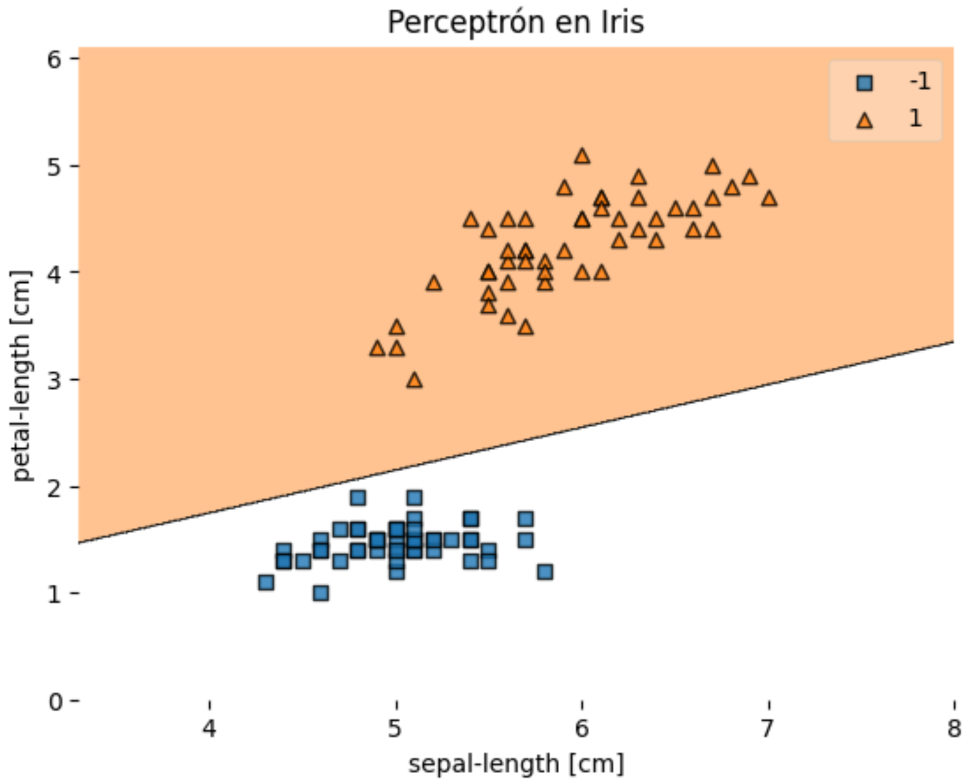
```
[-0.1] [[-0.28  0.7 ]]
```

```
#        w0   +   w1 * x1   +    w2 * x2  =  0
# intercept_ + coef_[0][0]*x1 + coef_[0][1]*x2
# => x2 = -(w0 + w1*x1) / w2
# Borde de decision
"""
x1 = np.linspace(4, 7.5, 2)
x2 = -(ppn.intercept_ + ppn.coef_[0][0]*x1) / ppn.coef_[0][1]
plt.plot(x1, x2, 'g', label = "Borde de Decision")
# Clase -1 : setosa
plt.scatter(X[y==-1][:,0],X[y==-1][:,1],color='red',marker='o',label='Setosa')
# Clase 1 : versicolor
plt.scatter(X[y==1][:,0],X[y==1][:,1],color='blue',marker='^',label='Versicolor')
plt.legend()
plt.grid(color='lightgray', linestyle='--')
"""
x1 = np.linspace(4, 7.5, 2)
x2 = -(ppn.intercept_ + ppn.coef_[0][0]*x1) / ppn.coef_[0][1]
plt.plot(x1, x2, 'g', label = "Borde de Decision")
# Clase -1 : setosa
plt.scatter(X[y==-1][:,0],X[y==-1][:,1],color='red',marker='o',label='Setosa')
# Clase 1 : versicolor
plt.scatter(X[y==1][:,0],X[y==1][:,1],color='blue',marker='^',label='Versicolor')
plt.legend()
plt.grid(color='lightgray', linestyle='--')
```




```
#!pip install mlxtend
```

```
from mlxtend.plotting import plot_decision_regions
# Regiones de decisión
plot_decision_regions(X, y, clf=ppn)
plt.title('Perceptrón en Iris')
plt.xlabel('sepal-length [cm]')
plt.ylabel('petal-length [cm]')
plt.show()
```




```
# Con sklearn
from sklearn import datasets
import numpy as np
iris = datasets.load_iris()
```

```
X = iris.data[:, [2,3]]
y = iris.target[:]
print('Etiquetas : ',np.unique(y))
```

```
Etiquetas :  [0 1 2]
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,
                                               random_state=1,stratify=y)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(105, 2) (105,)
(45, 2) (45,)
```

```
print('Total de etiquetas en y       :', np.bincount(y))
print('Total de etiquetas en y_train :', np.bincount(y_train))
print('Total de etiquetas en y_test  :', np.bincount(y_test))
```

```
Total de etiquetas en y       : [50 50 50]
Total de etiquetas en y_train : [35 35 35]
Total de etiquetas en y_test  : [15 15 15]
```

```
# Estandarización
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
#sc.fit(X_train)
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

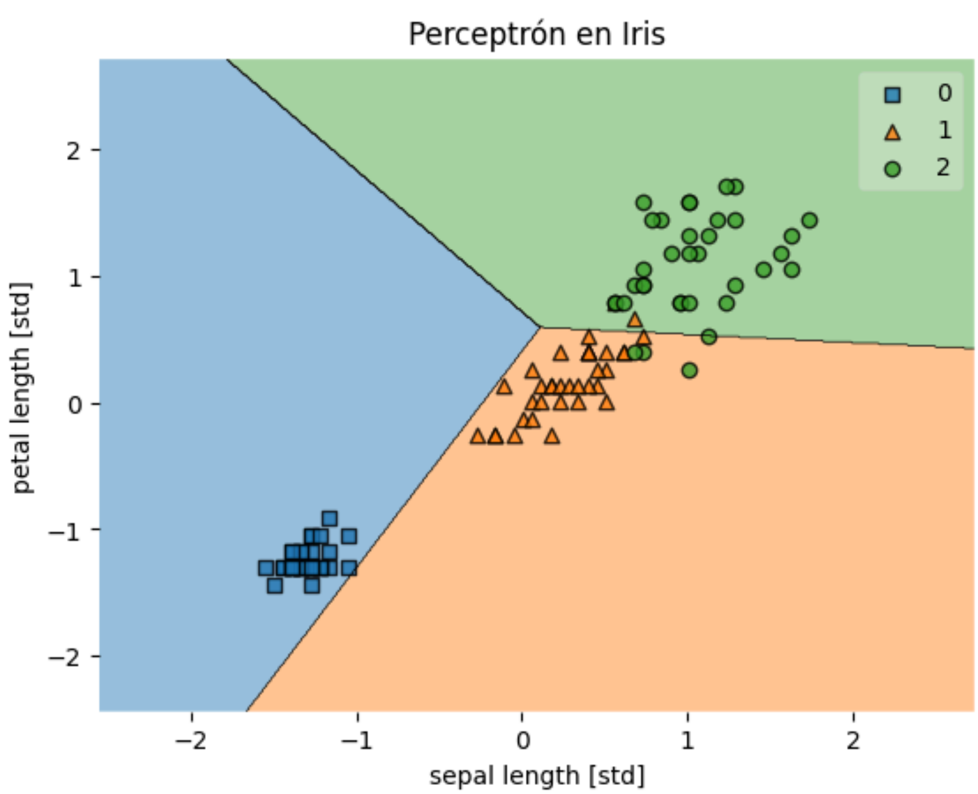
```
# Perceptrón
from sklearn.linear_model import Perceptron
ppn = Perceptron(max_iter=40, eta0=0.1, random_state=1)
# Ajuste y evaluación
ppn.fit(X_train_std, y_train)
print('Exactitud : ',ppn.score(X_test_std,y_test))
```

```
Exactitud :  0.9777777777777777
```

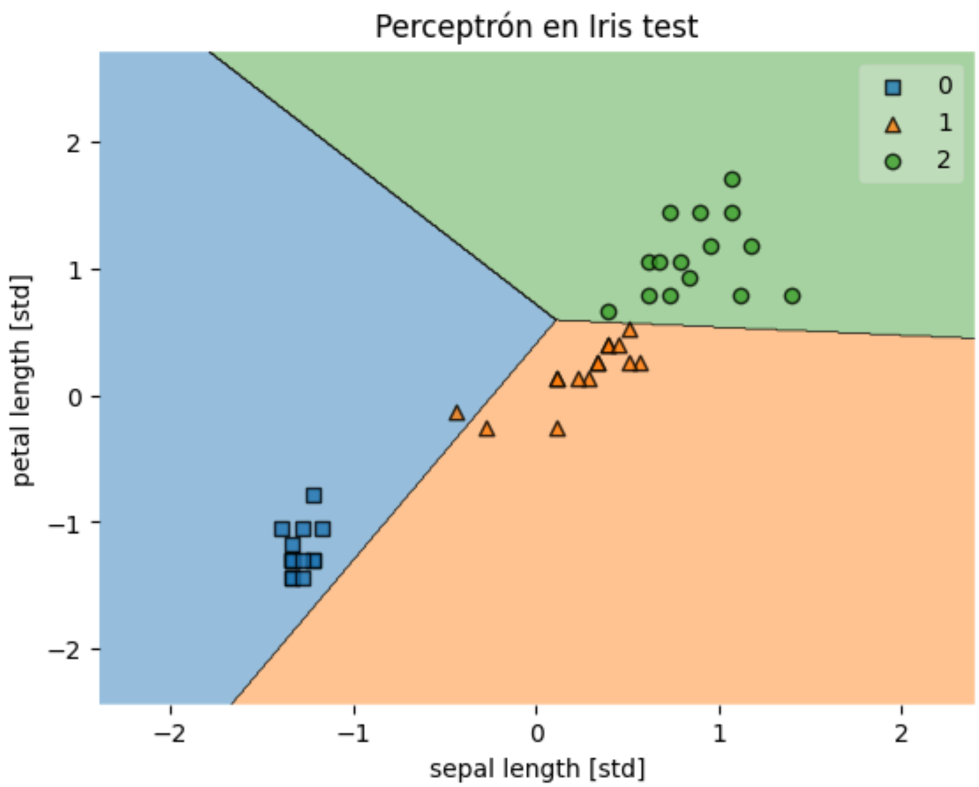
```
from sklearn.metrics import accuracy_score
y_pred = ppn.predict(X_test_std)
print('Exactitud : ',accuracy_score(y_test,y_pred))
```

```
Exactitud :  0.9777777777777777
```

```
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt
# Regiones de decisión (entrenamiento)
plot_decision_regions(X_train_std, y_train, clf=ppn)
plt.xlabel('sepal length [std]')
plt.ylabel('petal length [std]')
plt.title('Perceptrón en Iris')
plt.show()
```



```python
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt
# Regiones de decisión (pruebas)
plot_decision_regions(X_test_std, y_test, clf=ppn)
plt.xlabel('sepal length [std]')
plt.ylabel('petal length [std]')
plt.title('Perceptrón en Iris test')
plt.show()
```

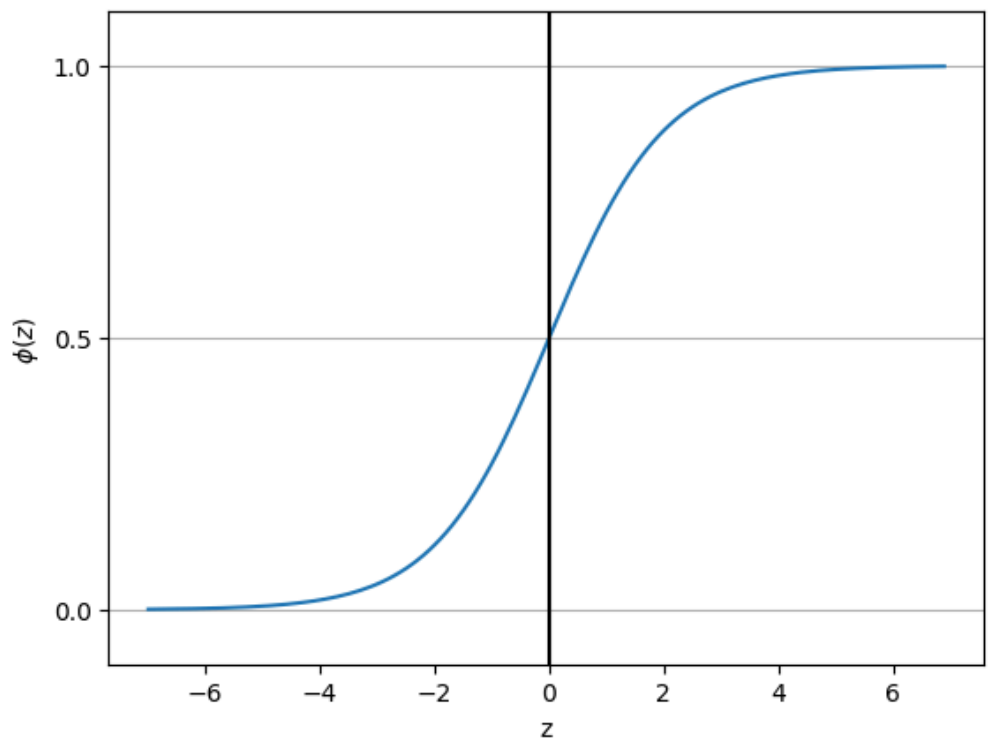



```python

```

```python
# Regresión Logística
```

```python
import numpy as np
import matplotlib.pyplot as plt
# Sigmoide
def sigmoide(z):
  return 1/(1+np.exp(-z))
```

```python
z = np.arange(-7,7,0.1)
phi_z = sigmoide(z)
plt.plot(z, phi_z)
plt.axvline(0.0, color='k')
plt.ylim(-0.1,1.1)
plt.xlabel('z')
plt.ylabel('$\phi (z)$')
plt.yticks([0.0,0.5,1])
ax = plt.gca() # gca =  get current axis
ax.yaxis.grid(True)
plt.show()
```




```python
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C=100, random_state=1) # C => inverso de la regularización
                                  # valor pequeño => reglurarización más fuerte
lr.fit(X_train_std, y_train)
```

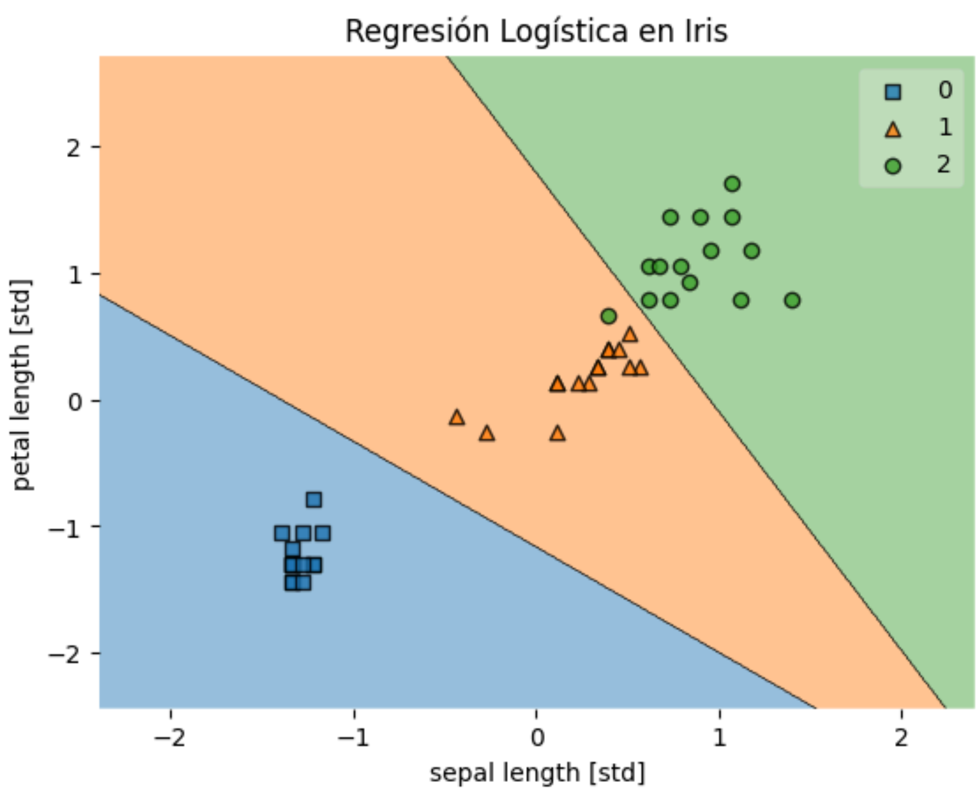
```
LogisticRegression
LogisticRegression(C=100, random_state=1)
```

```python
y_pred = lr.predict(X_test_std)
print('Exactitud : ',lr.score(X_test_std,y_test))
```

```
Exactitud :  0.9777777777777777
```

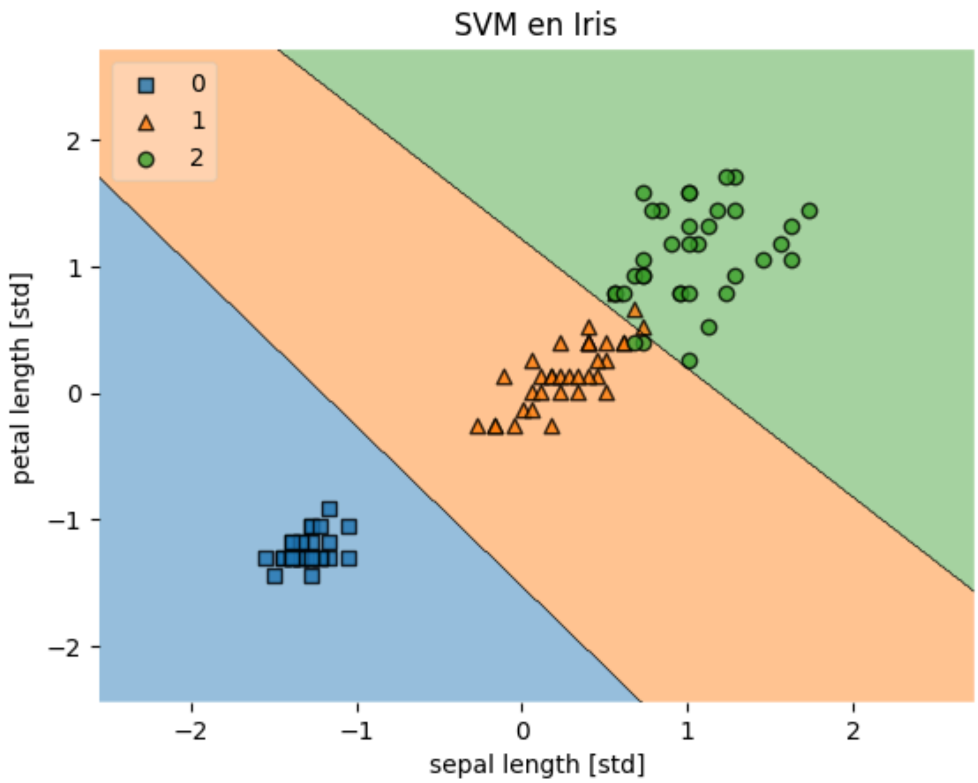
```python
# Regiones de decisión (entrenamiento)
plot_decision_regions(X_train_std, y_train, clf=lr)
plt.xlabel('sepal length [std]')
plt.ylabel('petal length [std]')
plt.title('Regresión Logística en Iris')
plt.show()
```


```
# Regiones de decisión (pruebas)
plot_decision_regions(X_test_std, y_test, clf=lr)
plt.xlabel('sepal length [std]')
plt.ylabel('petal length [std]')
plt.title('Regresión Logística en Iris')
plt.show()
```

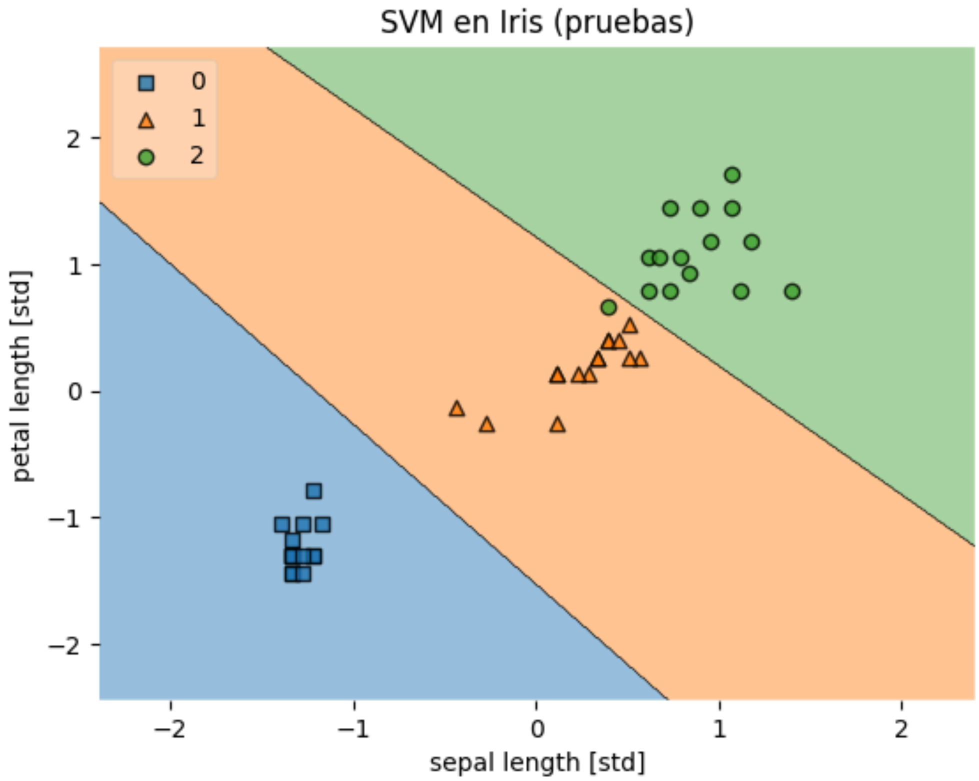


```
# Máquina de soporte vectorial
```

```
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt
from sklearn.svm import SVC
# Modelo y ajuste
svm = SVC(C=0.5, kernel='linear')
svm.fit(X_train_std, y_train)
# Regiones de decisión (entrenamiento)
plot_decision_regions(X_train_std, y_train, clf=svm, legend=2)
plt.xlabel('sepal length [std]')
plt.ylabel('petal length [std]')
plt.title('SVM en Iris')
plt.show()
```




```
# Regiones de decisión (pruebas)
plot_decision_regions(X_test_std, y_test, clf=svm, legend=2)
plt.xlabel('sepal length [std]')
plt.ylabel('petal length [std]')
plt.title('SVM en Iris (pruebas)')
plt.show()
```



1

1

1

1

1

1

1

1

1

1

1

1

```
# KNN
# bibliotecas
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

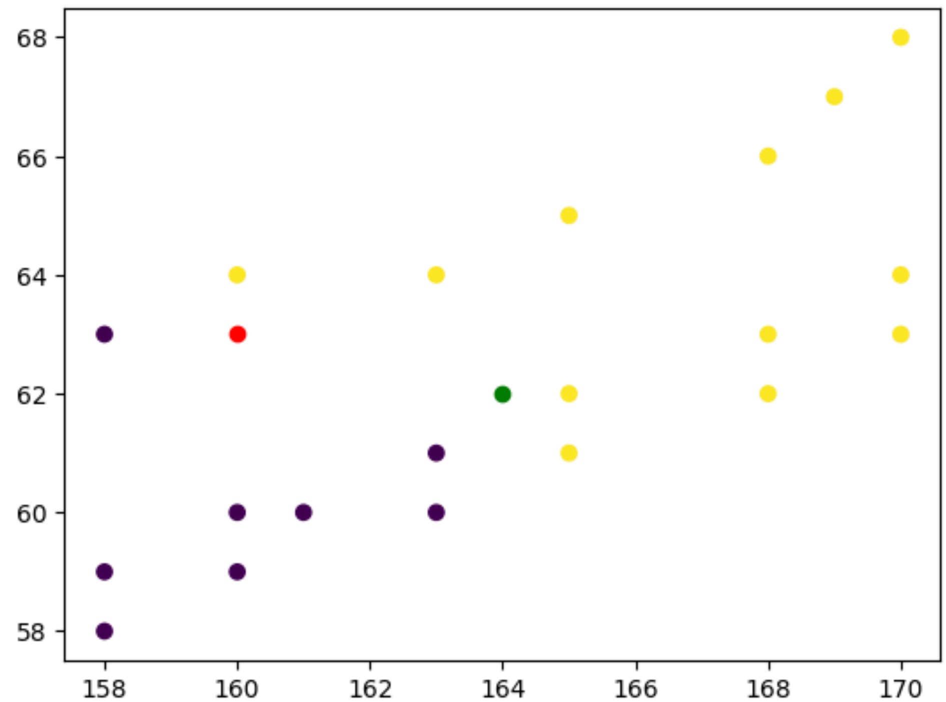
```
data = pd.read_csv('tallas.csv')
data.T
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
altura	170	168	163	168	158	160	168	165	160	158	169	158	170	165	161	170	163	160	165	1
peso	64	62	60	63	63	60	66	61	59	59	67	58	63	65	60	68	61	64	62	
talla	L	L	M	L	M	M	L	L	M	M	L	M	L	L	M	L	M	L	L	

```
talla_map = {'L':1, 'M':0}
data['color'] = data.talla.map(talla_map)
data.T
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
altura	170	168	163	168	158	160	168	165	160	158	169	158	170	165	161	170	163	160	165	1
peso	64	62	60	63	63	60	66	61	59	59	67	58	63	65	60	68	61	64	62	
talla	L	L	M	L	M	M	L	L	M	M	L	M	L	L	M	L	M	L	L	
color	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0	1	0	1	1	

```
plt.scatter
new = pd.DataFrame([ [160,63,None,None] ])
new.columns = ['altura','peso','talla','color']
plt.scatter(new.altura,new.peso,color='r')
new2 = pd.DataFrame([ [164,62,None,None] ])
new2.columns = ['altura','peso','talla','color']
plt.scatter(new2.altura,new2.peso,color='g')
plt.scatter(data.altura, data.peso, c=data.color)
plt.show()
```



```python
# knn propio
def get_closest_points(data, point, k=3):
  X = data.iloc[:,0:2].values
  p = point.iloc[:,0:2].values
  talla = data.talla.values
  dist=[[i,np.linalg.norm(X[i]-p),talla[i]]
        for i in range(len(X))]
  dist = pd.DataFrame(dist)
  dist.columns = ['index','dist','talla']
  return dist.sort_values(by='dist').head(k)

def show_closest_points(data, point, cercanos, color='k'):
  plt.scatter(data.altura,data.peso,c=data.color)
  plt.scatter(point.altura,point.peso,color=color)
  for c in cercanos.values:
    p = data.loc[c[0],:]
    plt.plot([point.altura[0],p.altura],[point.peso[0],p.peso])
  plt.show()
```

```python
c = get_closest_points(data, new, 5)
print(c)
#show_closest_points(data, new, c, color='r')
```

```
    index      dist talla
17     17  1.000000     L
4       4  2.000000     M
5       5  3.000000     M
14     14  3.162278     M
19     19  3.162278     L
```

```python
c = get_closest_points(data, new2, 5)
print(c)
#show_closest_points(data, new2, c, color='g')
```

	index	dist	talla
18	18	1.000000	L
7	7	1.414214	L
16	16	1.414214	M
2	2	2.236068	M
19	19	2.236068	L

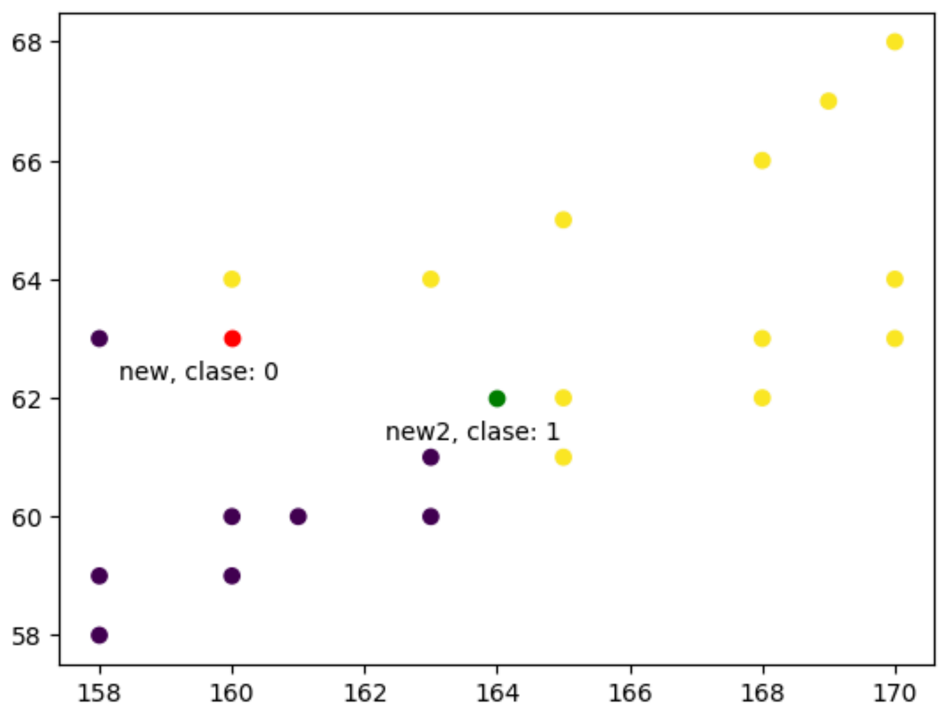
```
# Con sklearn
from sklearn.neighbors import KNeighborsClassifier
X = data.iloc[:,:2].values
y = data.iloc[:,3].values
```

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X, y)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

```
new = np.array([160,63]).reshape(1,2)
new_pred = knn.predict(new)[0]
new2 = np.array([164,62]).reshape(1,2)
new2_pred = knn.predict(new2)[0]
plt.scatter(X[:,0], X[:,1], c=y)
plt.scatter(new[:,0], new[:,1], c='r')
plt.text(x=new[:,0]-1.7, y=new[:,1]-0.7, s=f"new, clase: {new_pred}")
plt.scatter(new2[:,0], new2[:,1], c='g')
plt.text(x=new2[:,0]-1.7, y=new2[:,1]-0.7, s=f"new2, clase: {new2_pred}")
```

```
Text([162.3], [61.3], 'new2, clase: 1')
```



```python
# Ejemplo 2 ~> regresión
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

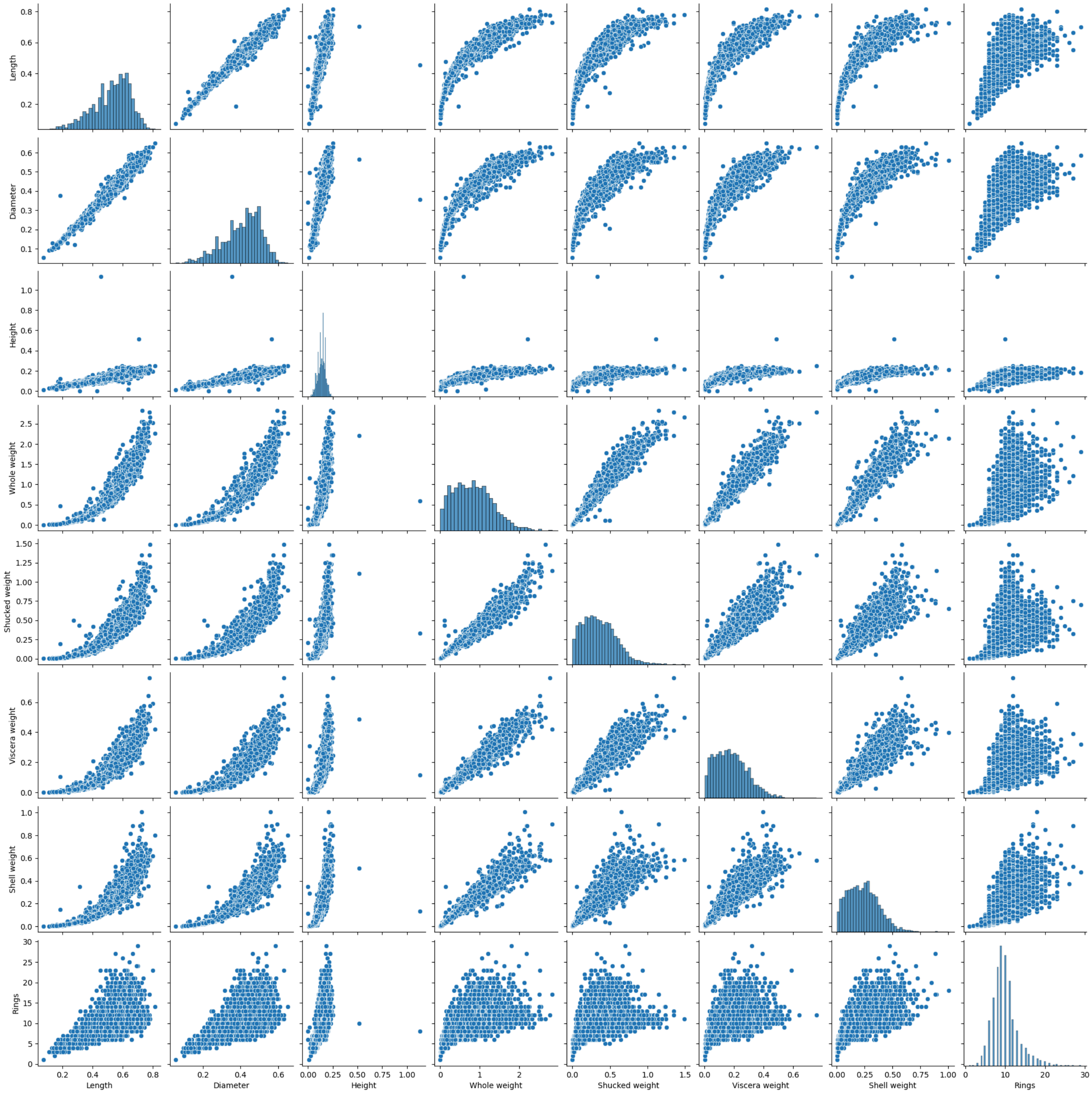
```python
# https://archive.ics.uci.edu/dataset/1/abalone
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data"
abulon = pd.read_csv(url, header=None)
abulon.columns = ["Sex", "Length", "Diameter", "Height", "Whole weight",
                  "Shucked weight", "Viscera weight", "Shell weight", "Rings"]
abulon.tail(3)
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
4174	M	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.308	9
4175	F	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.296	10
4176	M	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.495	12

```python
# https://laroussecocina.mx/palabra/abulon/
# Predecir la edad => No sirve la columna sex
abulon = abulon.drop("Sex", axis=1)
```

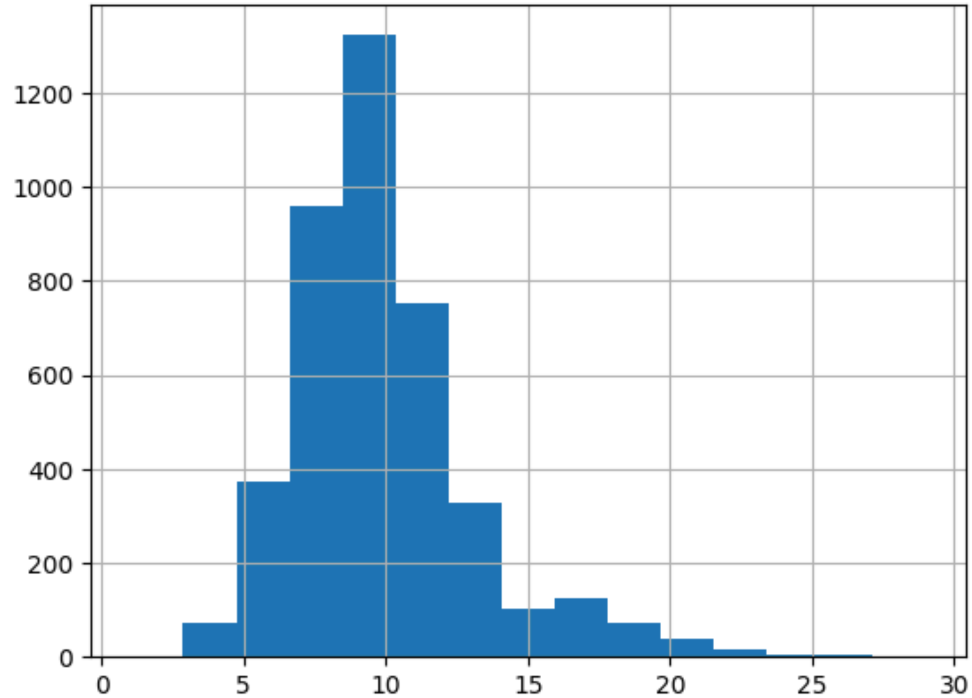
```python
sns.pairplot(abulon) # (8x8)
```

```
<seaborn.axisgrid.PairGrid at 0x7fbfc87cbd90>
```



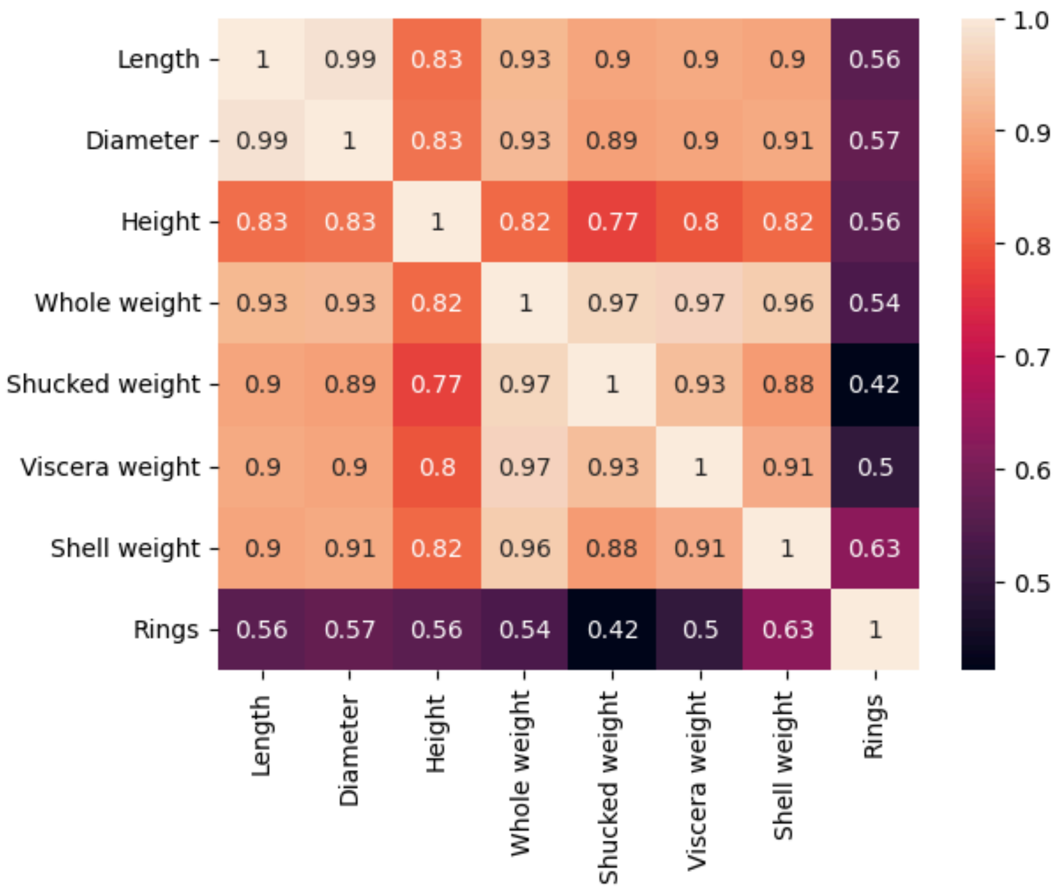

```
# número de anillos ~ edad <= Variable objetivo
# en este conjunto la mayoría están entre 5 y 15 años
abulon.Rings.hist(bins=15)
```

<Axes: >




```
1 #Correlaciones
2 corr = abulon.corr()
3 sns.heatmap(corr, annot=True)
```

<Axes: >



```
corr.Rings
```

```
Length            0.556720
Diameter          0.574660
Height            0.557467
Whole weight      0.540390
Shucked weight    0.420884
Viscera weight    0.503819
Shell weight      0.627574
Rings             1.000000
Name: Rings, dtype: float64
```

```
X = abulon.drop('Rings',axis=1).values
y = abulon.Rings.values
X[:2,:],y[:2]
```

```
(array([[0.455 , 0.365 , 0.095 , 0.514 , 0.2245, 0.101 , 0.15  ],
        [0.35  , 0.265 , 0.09  , 0.2255, 0.0995, 0.0485, 0.07  ]]),
 array([15,  7]))
```

```
# Predecir el número de anillo => edad
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
X_train.shape, X_test.shape
```

```
((3341, 7), (836, 7))
```

```
# Modelo regresor
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors=3)
knn.fit(X_train, y_train)
```

```
▾ KNeighborsRegressor ⓘ ?
KNeighborsRegressor(n_neighbors=3)
```

```
from sklearn.metrics import mean_squared_error
from math import sqrt
y_train_pred = knn.predict(X_train)
mse = mean_squared_error(y_train, y_train_pred)
sqrt(mse)
```

```
1.6643104166366538
```

```
y_pred = knn.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
sqrt(mse)
```

```
2.407209400382251
```

```
# Ejemplo Bag of Words
documents = ['Hello, how are you!',
             'Win money, win from home.',
             'Call me now.',
             'Hello, Call hello you tomorrow?']
```

```
# Importar CountVectorizer y crear un objeto
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
cv.fit(documents)
names = cv.get_feature_names_out()
names
```

```
array(['are', 'call', 'from', 'hello', 'home', 'how', 'me', 'money',
       'now', 'tomorrow', 'win', 'you'], dtype=object)
```

```
# Transformar y convertir en arreglo
docs = cv.transform(documents).toarray()
docs
```

```
array([[1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1],
       [0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 2, 0],
       [0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
       [0, 1, 0, 2, 0, 0, 0, 0, 0, 1, 0, 1]])
```

```
# Convertir en DataFrame
import pandas as pd
freq = pd.DataFrame(data=docs, columns=names)
freq
```

	are	call	from	hello	home	how	me	money	now	tomorrow	win	you
0	1	0	0	1	0	1	0	0	0	0	0	1
1	0	0	1	0	1	0	0	1	0	0	2	0
2	0	1	0	0	0	0	1	0	1	0	0	0
3	0	1	0	2	0	0	0	0	0	1	0	1

```
# Naive Bayes
import pandas as pd
df = pd.read_csv('spam.csv', names = ['label', 'sms_message'])
df.head(3)
```

	label	sms_message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...

```
# Preprocesamiento
df.label = df.label.map({'ham':0,'spam':1})
df.head(3)
```

	label	sms_message
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(df.sms_message, df.label,
                                               random_state=1)
print('Conjunto completo: {}'.format(df.shape[0]))
print('Conjunto de entrenamiento: {}'.format(X_train.shape[0]))
print('Conjunto de pruebas: {}'.format(X_test.shape[0]))
```

```
Conjunto completo: 5572
Conjunto de entrenamiento: 4179
Conjunto de pruebas: 1393
```

```
# Aplicar BoW == Bolsa de palabras
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
# entrenamos el objeto
train_data = cv.fit_transform(X_train)
test_data = cv.transform(X_test)
train_data.shape, test_data.shape
```

```
((4179, 7464), (1393, 7464))
```

```
# Modelo Bayes multinomial
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
# Entrenar y predecir
nb.fit(train_data, y_train)
y_pred = nb.predict(test_data)
```

```
# Hay otras metricas
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
print('Exactitud: ', accuracy_score(y_test,y_pred) )
print('Precisión: ', precision_score(y_test,y_pred) )
print('   Recall: ', recall_score(y_test,y_pred)  )
print('       F1: ', f1_score(y_test,y_pred) )
```

```
Exactitud:  0.9856424982053122
Precisión:  0.9545454545454546
   Recall:  0.9333333333333333
       F1:  0.9438202247191011
```

```
# XGB Clasificador => Predecir diabetes
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
#!pip install xgboost
```

```
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
# Datos
df = pd.read_csv('diabetes.csv')
df.tail(2)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFuncti
766	1	126	60	0	0	30.1	0.3
767	1	93	70	31	0	30.4	0.3

```
# Separar X, y
X = df.iloc[:,:-1].values
y = df.iloc[:,-1].values
```

```
# Entrenamiento y prueba
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=5)
X_train.shape, X_test.shape
```

```
((614, 8), (154, 8))
```

```
# Modelo
xgb = XGBClassifier()
xgb.fit(X_train, y_train)
```

```
XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              feature_weights=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, ...)
```

```
y_pred = xgb.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy : ",accuracy)
```

```
Accuracy :  0.7922077922077922
```

```

```

```
#!pip install xgboost
```

```
# Regresor => Predecir calorías
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn import metrics
```

```
# Datos
exercise = pd.read_csv('exercise.csv')
exercise.tail(3)
```

	User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp
14997	17271188	female	43	159.0	58.0	16.0	90.0	40.1
14998	18643037	male	78	193.0	97.0	2.0	84.0	38.3
14999	11751526	male	63	173.0	79.0	18.0	92.0	40.5

```
# Datos
calories = pd.read_csv('calories.csv')
calories.tail(3)
```

	User_ID	Calories
14997	17271188	75.0
14998	18643037	11.0
14999	11751526	98.0

```
# Combinando los dos DFs
calories_data = pd.concat([exercise, calories.Calories], axis=1)
calories_data.head(3)
```

	User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories
0	14733363	male	68	190.0	94.0	29.0	105.0	40.8	231.0
1	14861698	female	20	166.0	60.0	14.0	94.0	40.3	66.0
2	11179863	male	69	179.0	79.0	5.0	88.0	38.7	26.0

```python
# Número de filas y columnas
calories_data.shape
```

```
(15000, 9)
```

```python
# Información del conjunto
calories_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   User_ID     15000 non-null  int64
 1   Gender      15000 non-null  object
 2   Age         15000 non-null  int64
 3   Height      15000 non-null  float64
 4   Weight      15000 non-null  float64
 5   Duration    15000 non-null  float64
 6   Heart_Rate  15000 non-null  float64
 7   Body_Temp   15000 non-null  float64
 8   Calories    15000 non-null  float64
dtypes: float64(6), int64(2), object(1)
memory usage: 1.0+ MB
```

```python
# ¿Hay nulos?
calories_data.isnull().sum()
```

```
User_ID       0
Gender        0
Age           0
Height        0
Weight        0
Duration      0
Heart_Rate    0
Body_Temp     0
Calories      0
dtype: int64
```

```python
# Algunas estadísticas de las columnas
calories_data.describe()
```

	User_ID	Age	Height	Weight	Duration	Heart_Rate	Body_Temp
count	1.500000e+04	15000.000000	15000.000000	15000.000000	15000.000000	15000.000000	15000.000000
mean	1.497736e+07	42.789800	174.465133	74.966867	15.530600	95.518533	40.025453
std	2.872851e+06	16.980264	14.258114	15.035657	8.319203	9.583328	0.779230
min	1.000116e+07	20.000000	123.000000	36.000000	1.000000	67.000000	37.100000
25%	1.247419e+07	28.000000	164.000000	63.000000	8.000000	88.000000	39.600000
50%	1.499728e+07	39.000000	175.000000	74.000000	16.000000	96.000000	40.200000
75%	1.744928e+07	56.000000	185.000000	87.000000	23.000000	103.000000	40.600000
max	1.999965e+07	79.000000	222.000000	132.000000	30.000000	128.000000	41.500000

```
# Visualizaciones
sns.set()
```

```
# Conteo por género
#sns.countplot(x=calories_data.Gender)
```

```
# Distribución de edades
#sns.kdeplot(calories_data.Age, fill=True)
```

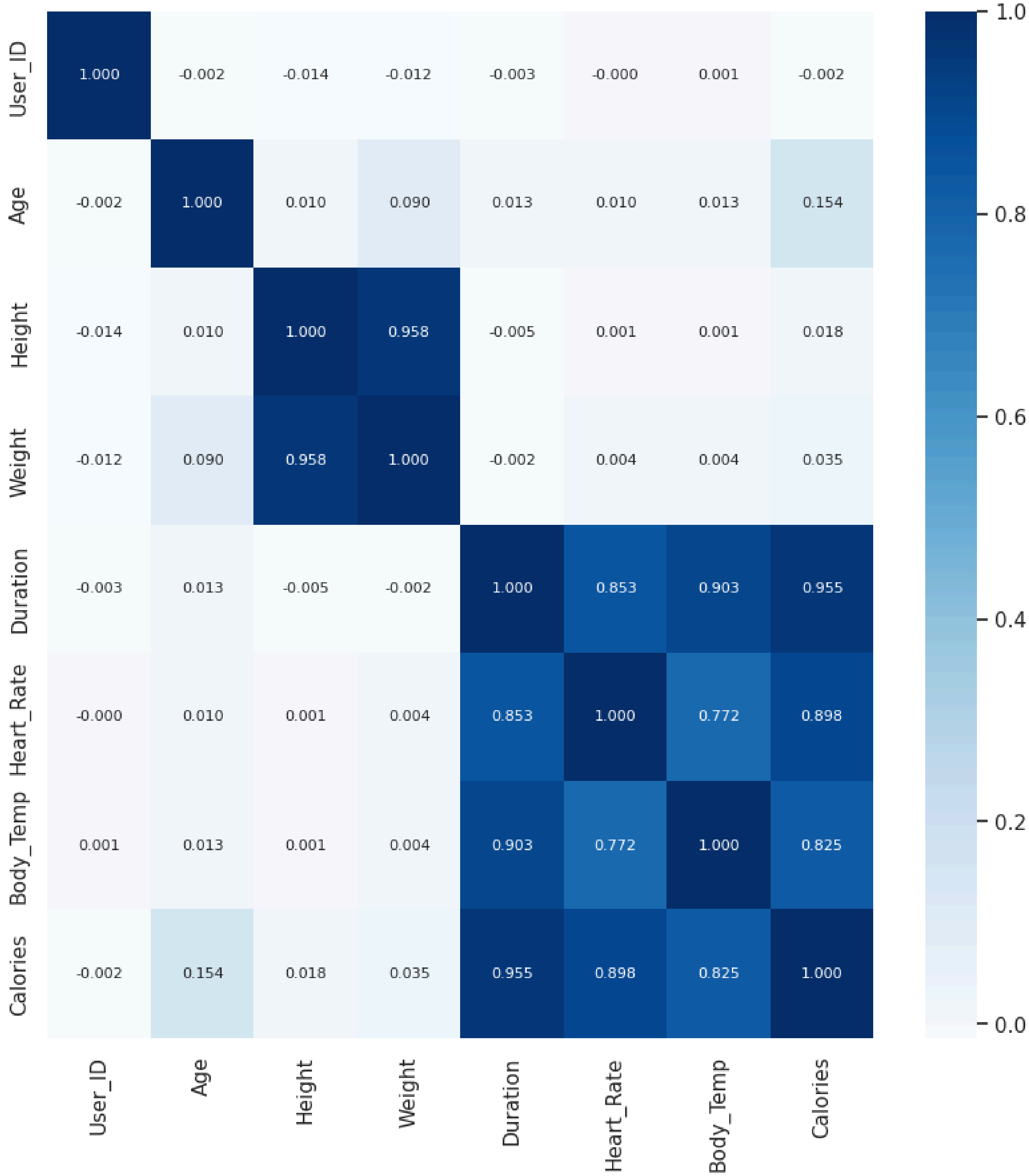
```
# Histograma de edades
#sns.histplot(calories_data.Age, kde=True)
```

```
# # Distribución de Peso
#sns.histplot(calories_data.Weight, kde=True)
```

```
# Correlación entre las columnas (positiva / negativa)
corr = calories_data.drop(['Gender'],axis=1).corr()
#corr
```

```
# Visualización con un heatmap
plt.figure(figsize=(10,10))
sns.heatmap(corr, cbar=True, annot=True, fmt='.3f',
            annot_kws={'size':8}, cmap='Blues')
```

<Axes: >




```
# Convertir texto a número
calories_data.replace({ 'Gender':{'male':0,'female':1} },inplace=True)
calories_data.head()
```

```
/tmp/ipykernel_660/941247895.py:2: FutureWarning: Downcasting behavior in `replace` is depre
  calories_data.replace({ 'Gender':{'male':0,'female':1} },inplace=True)
```

	User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories
0	14733363	0	68	190.0	94.0	29.0	105.0	40.8	231.0
1	14861698	1	20	166.0	60.0	14.0	94.0	40.3	66.0
2	11179863	0	69	179.0	79.0	5.0	88.0	38.7	26.0
3	16180408	1	34	179.0	71.0	13.0	100.0	40.5	71.0
4	17771927	1	27	154.0	58.0	10.0	81.0	39.8	35.0

```
# Separar predictoras y objetivo
X = calories_data.drop(columns=['User_ID','Calories'], axis=1)
y = calories_data.Calories
```

```
# Entrenamiento y pruebas
X_train,X_test,y_train,y_test=train_test_split(X, y, test_size=0.2,
                                               random_state=2)
```

```
X.shape, X_train.shape, X_test.shape
```

```
((15000, 7), (12000, 7), (3000, 7))
```

```
# Modelo XGBRegressor
xgb = XGBRegressor()
xgb.fit(X_train, y_train)
```

```
XGBRegressor

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=None, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=None,
             n_jobs=None, num_parallel_tree=None, ...)
```

```
# Evaluación
y_hat = xgb.predict(X_test)
```

```
# Error absoluto medio
mae = metrics.mean_absolute_error(y_test, y_hat)
print("Mean Absolute Error = ", mae)
```

```
Mean Absolute Error =  1.4833678883314132
```