Clasificación Multicategórica con Regresión Logística

Entender el problema

Para mostrar la creación de un modelo multiclase y su evaluación vamos a tratar un problema en el que queremos detectar cuál es el sistema operativo de los usuarios que acceden a una dirección web específica (y que no es importante para el problema) en la que pasan un tiempo determinado, navegan a través de las páginas del site, realizan un conjunto de acciones y dejan una valoración de la misma. Encontrarás una variable "clase" que contiene el target, esta vez no tendrás que buscarlo, y que toma tres posibles valores en función del OS: Windows, Linux, Mac.

Creemos ese modelo predictor a partir de una regresión logística.

Preparación Datos: Primer vistazo

In [2]:
```
import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

plt.rc('font', size=14)
plt.rc('axes', labelsize=14, titlesize=14)
plt.rc('legend', fontsize=14)
plt.rc('xtick', labelsize=10)
plt.rc('ytick', labelsize=10)
```

In [2]:
```
df_os = pd.read_csv("data/usuarios_win_mac_lin.csv")

clases = {
    0: 'Windows',
    1: 'Linux',
```

```
    2: 'Mac'
}

df_os.head()
```

Out[2]:

	duracion	paginas	acciones	valor	clase
0	7.0	2	4	8	2
1	21.0	2	6	6	2
2	57.0	2	4	4	2
3	101.0	3	6	12	2
4	109.0	2	6	12	2

In [3]:
```
df_os.info()
```

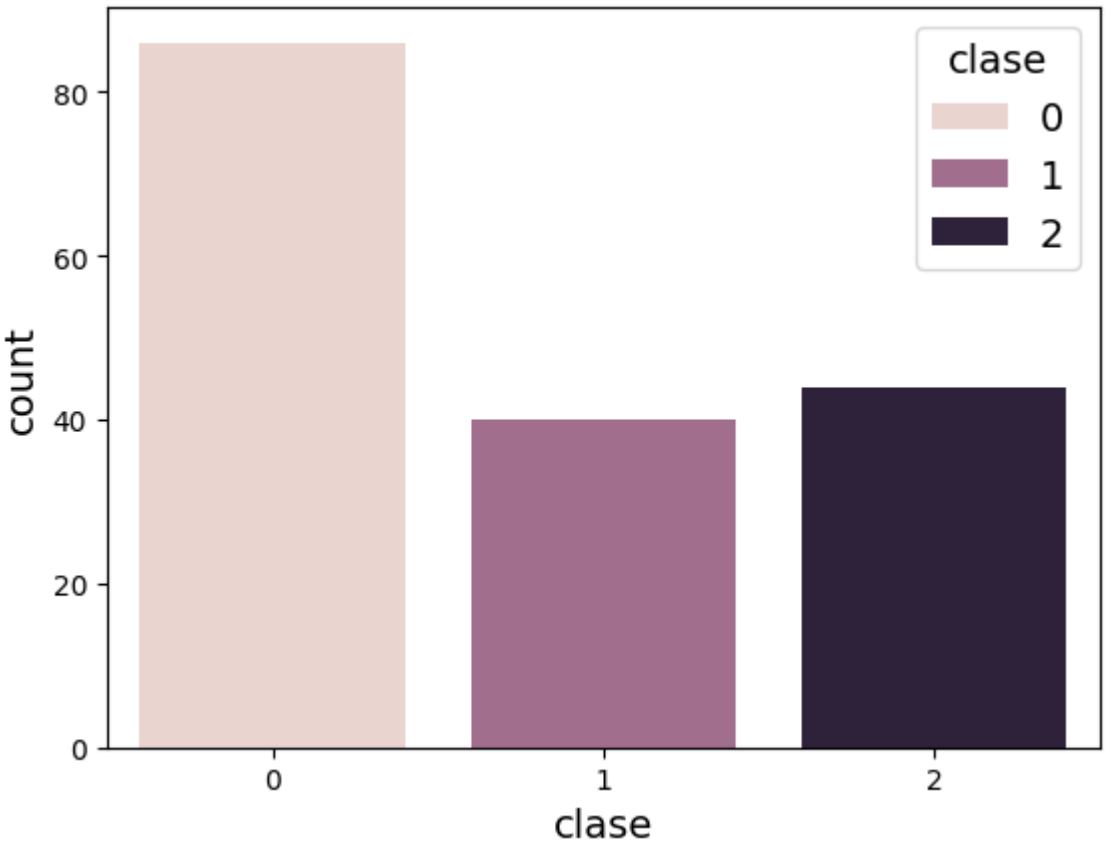
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 170 entries, 0 to 169
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   duracion  170 non-null    float64
 1   paginas   170 non-null    int64
 2   acciones  170 non-null    int64
 3   valor     170 non-null    int64
 4   clase     170 non-null    int64
dtypes: float64(1), int64(4)
memory usage: 6.8 KB
```

Mostramos el target, y su balanceo o desbalanceo

In [4]:
```
df_os.clase.value_counts(True)
```

Out[4]:
```
clase
0    0.505882
2    0.258824
1    0.235294
Name: proportion, dtype: float64
```

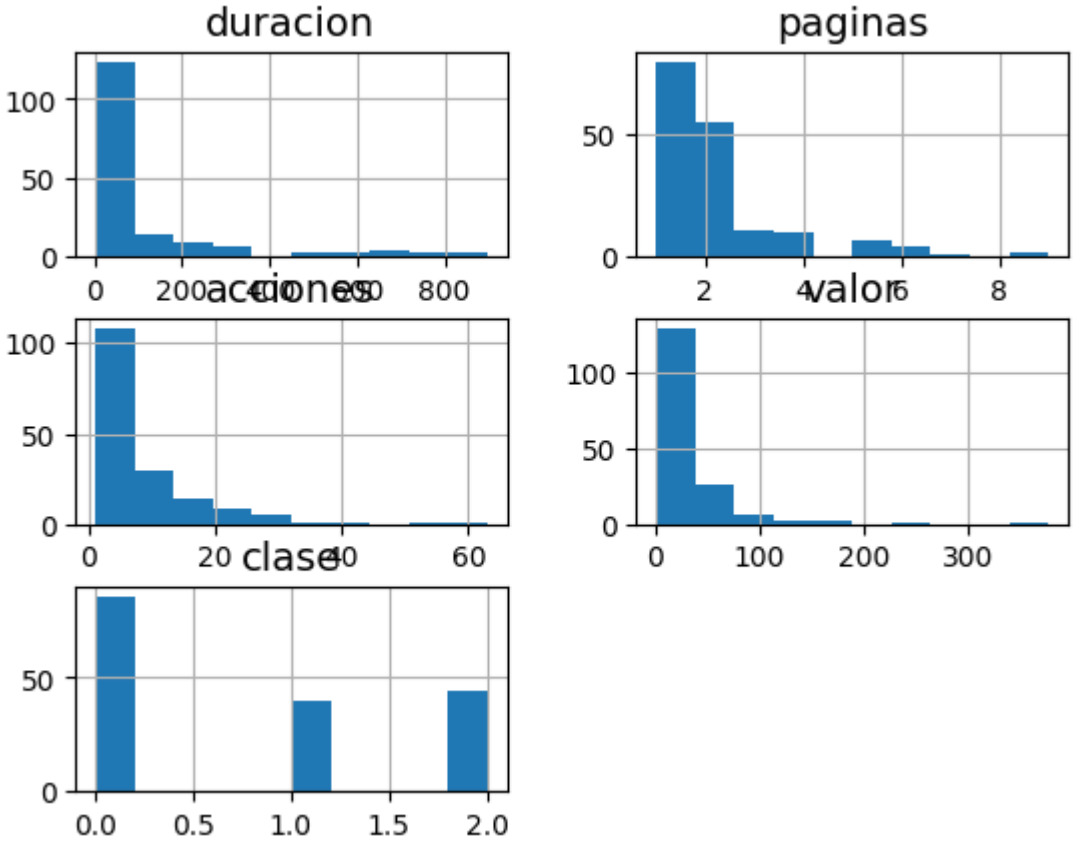
In [5]:
```
sns.countplot(x = "clase", data = df_os, hue = "clase");
```



Windows es el sistema dominante y habrá que tenerlo en cuenta a la hora de las medidas y tendríamos que tenerlo en cuenta a la hora de modelar pero lo veremos en sesiones posteriores.

Echamos un vistazo y sólo eso a la distribución de las variables contínuas (o sea casi todas) y nos fijamos en escala y figura

In [6]:
```
df_os.hist();
```



Tiene pinta de que vamos a tener que escalar y hacer alguna transformación para "normalizar" las distribuciones.

Split

Son muy pocos datos (170) pero vamos a hacer el split igual para que vayamos cogiendo el hábito. Además estratificamos sobre el target (pero ojo esto es para hacer más limpio el ejercicio, por ejemplo porque suponemos que es algo que se va a mantener debido al marketshare de cada sistema operativo)

In [7]:
```
train_set, test_set = train_test_split(df_os, test_size= 0.2, stratify= df_os["c
```

In [8]:
```
train_set["clase"].value_counts(normalize = True)
```

Out[8]:
```
clase
0    0.507353
2    0.257353
1    0.235294
Name: proportion, dtype: float64
```

In [9]:
```
test_set["clase"].value_counts(normalize = True)
```

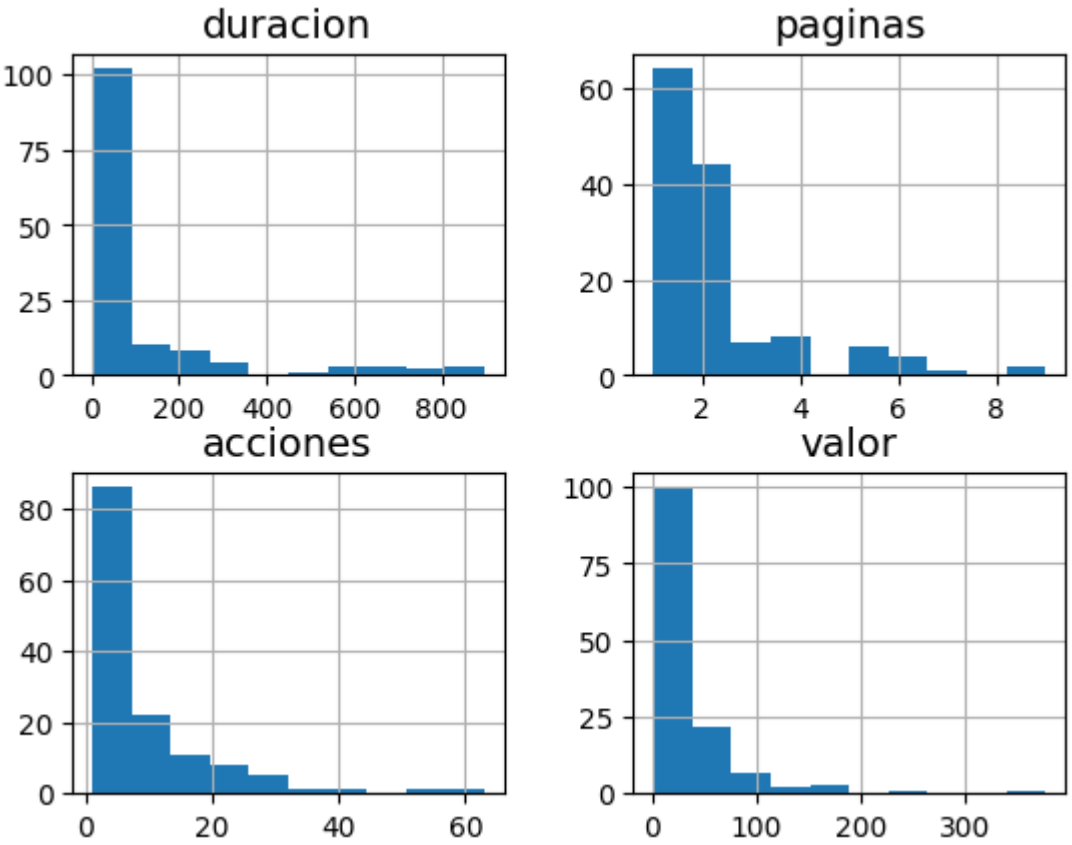
Out[9]:
```
clase
0    0.500000
2    0.264706
1    0.235294
Name: proportion, dtype: float64
```

No es perfecto, pero prueba a hacer el split sin estratificar.

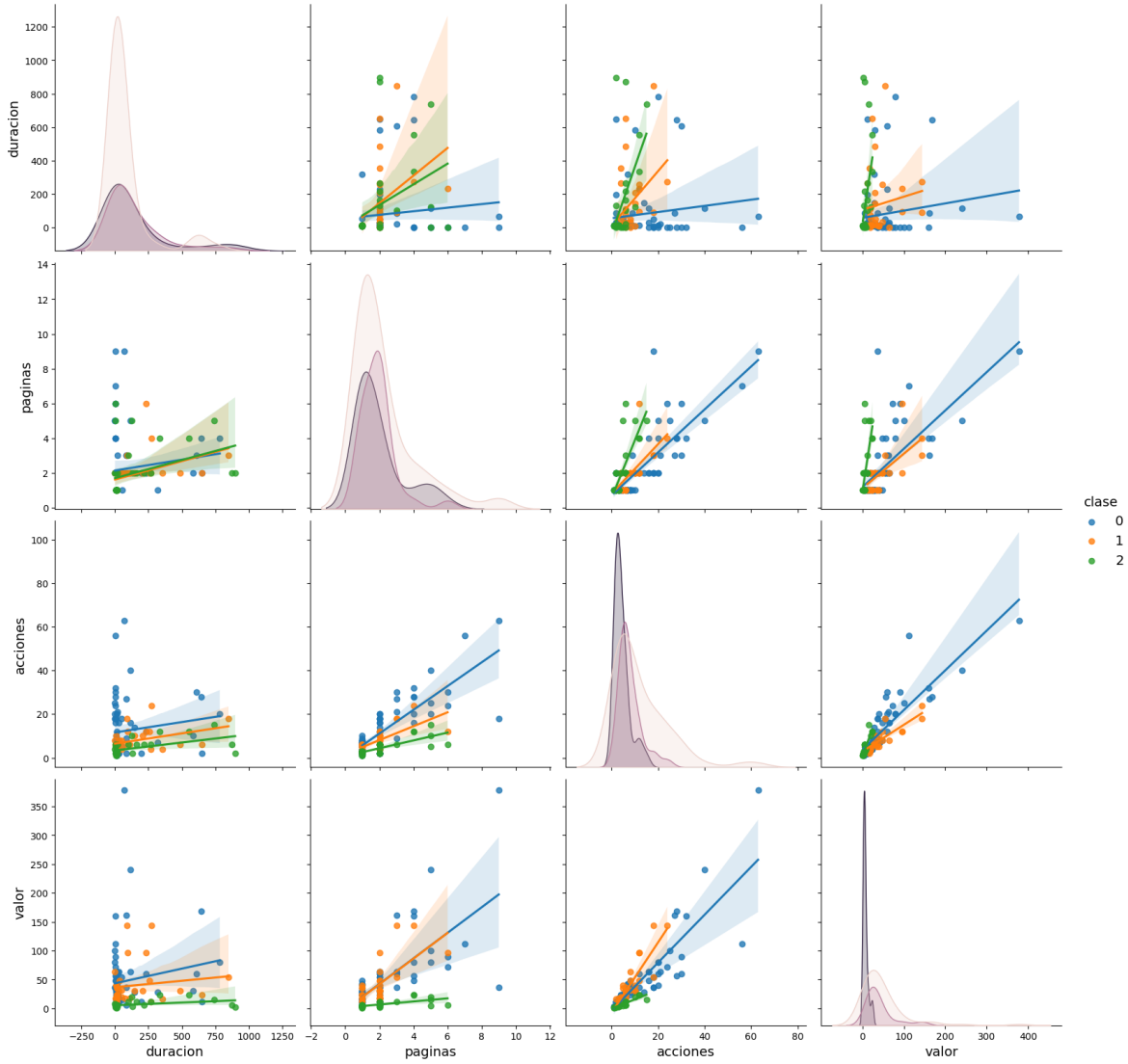
Mini-EDA

Vamos a ver la correlación de las variables con el target, pero realmente son tan pocas que no vamos a quitar ninguna:

In [10]:
```
train_set.drop("clase", axis = 1).hist()
plt.show()
```



In [11]:
```
sns.pairplot(train_set,
             hue='clase',
             height=4,
             vars=["duracion", "paginas","acciones","valor"],
             kind='reg'); # "reg" de regresión lineal
```



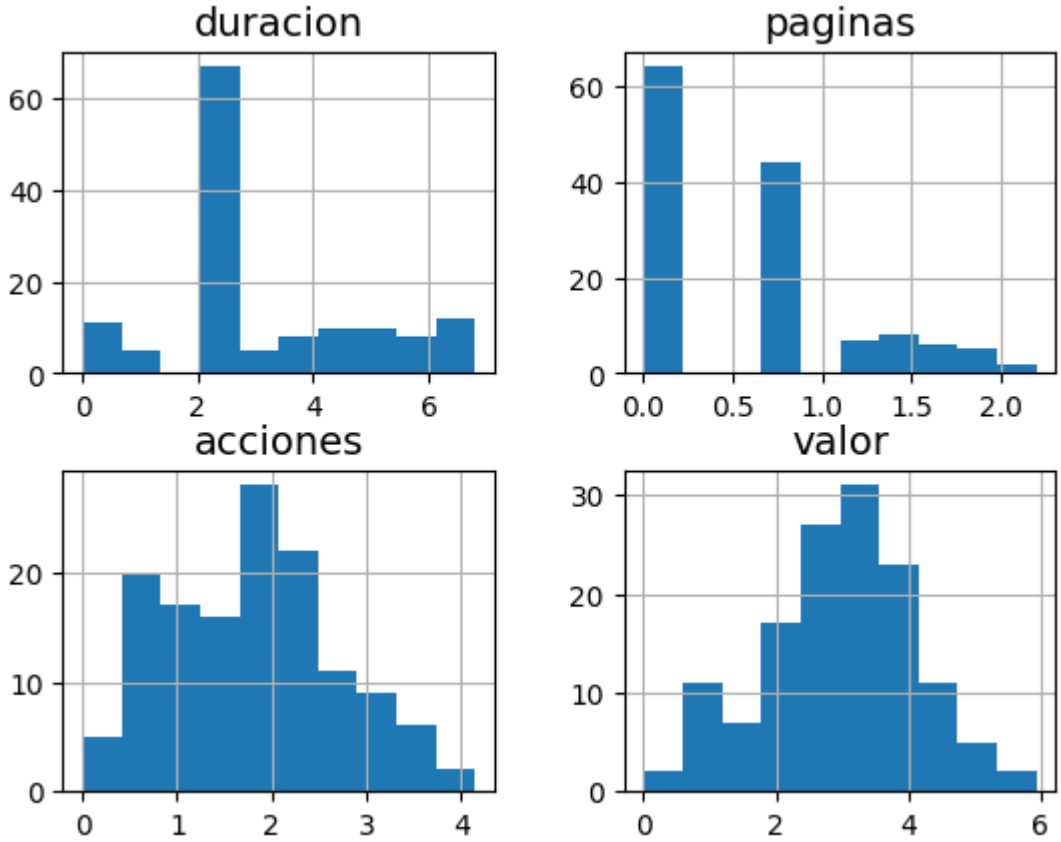
In [12]:
```
features = train_set.drop("clase", axis = 1).columns.to_list()
print(features)
```

```
['duracion', 'paginas', 'acciones', 'valor']
```

Procesamiento de Features

In [13]:
```
for col in features:
    train_set[col] = train_set[col].apply(np.log)
train_set[features].hist()
```

Out[13]:
```
array([[<Axes: title={'center': 'duracion'}>,
        <Axes: title={'center': 'paginas'}>],
       [<Axes: title={'center': 'acciones'}>,
        <Axes: title={'center': 'valor'}>]], dtype=object)
```



In [14]:
```
for col in features:
    test_set[col] = test_set[col].apply(np.log)
# Ojo ni lo mires :-), pero tienes que aplicarle las mismas transformaciones que
```

In [15]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(train_set[features]) # Adaptamos el StandarScaler al Train, pero lueg
X_train = pd.DataFrame(scaler.transform(train_set[features]), columns = features
X_test = pd.DataFrame(scaler.transform(test_set[features]), columns= features)
X_train.describe()
```

Out[15]:

	duracion	paginas	acciones	valor
count	1.360000e+02	1.360000e+02	1.360000e+02	1.360000e+02
mean	-1.632681e-16	-4.571507e-17	2.416368e-16	-3.395976e-16
std	1.003697e+00	1.003697e+00	1.003697e+00	1.003697e+00
min	-1.878357e+00	-8.947527e-01	-2.004312e+00	-2.501146e+00
25%	-4.781410e-01	-8.947527e-01	-7.831545e-01	-7.258228e-01
50%	-3.805922e-01	2.695613e-01	-1.268982e-02	2.121177e-01
75%	7.491348e-01	2.695613e-01	7.577748e-01	6.482352e-01
max	2.092503e+00	2.796035e+00	2.600971e+00	2.565771e+00

In [16]:
```
y_train = train_set["clase"]
y_test = test_set["clase"]
```

Creamos el modelo

In [17]:
```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(max_iter=10000)

clf.fit(X_train, y_train) # No hay que decir nada en especial, el detecta el tip
```

Out[17]:
```
LogisticRegression(max_iter=10000)
```

Si mostramos los coeficientes o pesos del modelo veremos que en realidad ha entrenado un modelo de regresión logística para cada clase como vimos en la sesión de teoría. Ha hecho un one-vs-rest

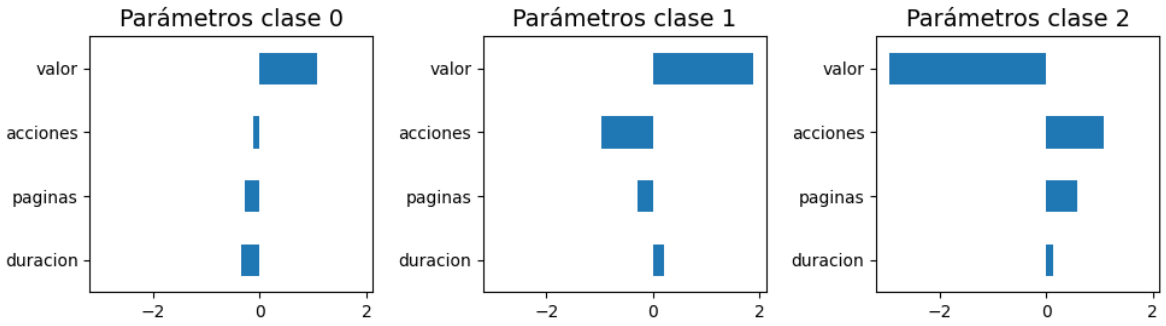
In [18]:
```
clf.coef_
```

Out[18]:
```
array([[-0.33875784, -0.28243018, -0.11998396,  1.06520743],
       [ 0.20873356, -0.2993436 , -0.96504166,  1.8810649 ],
       [ 0.13002428,  0.58177378,  1.08502562, -2.94627234]])
```

In [19]:
```
df_coefs = pd.DataFrame(clf.coef_, columns=X_train.columns)

fig, ax = plt.subplots(1, 3, figsize=(10, 3), sharex=True)
for i, subdf in df_coefs.iterrows():
    subdf.plot(kind="barh", ax=ax[i])
    ax[i].set_title(f"Parámetros clase {i}")

fig.tight_layout()
```



Ten en cuenta que estos coeficientes están aplicados sobre el logaritmo de las variables, aún así puedes ver que influye en la selección de cada clase.

Podemos ver las predicciones

In [20]:
```
clf.predict(X_train)
```

Out[20]:
```
array([0, 2, 0, 0, 2, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       2, 2, 0, 1, 0, 0, 0, 2, 1, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 0, 0, 2,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0, 0, 0, 2, 0, 0, 0, 0, 1,
       0, 2, 2, 0, 0, 2, 0, 2, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 2, 2, 0,
       0, 0, 0, 0, 0, 2, 2, 2, 2, 0, 0, 2, 0, 0, 0, 2, 0, 2, 0, 2, 0, 1,
       0, 0, 1, 1, 0, 2, 0, 0, 0, 0, 0, 2, 0, 1, 2, 0, 2, 0, 2, 2, 0, 2,
       2, 0, 2, 0])
```

Y las probabilidades para cada instancia, nos da un vector de tres probabilidades una para cada clase y puedes comprobar que escoge la de mayor probabilidad:

In [21]:
```
clf.predict_proba(X_train)
```

Out[21]:
```
array([[0.71646107, 0.27437523, 0.0091637 ],
       [0.39537849, 0.10494105, 0.49968046],
       [0.62557227, 0.33522879, 0.03919894],
       [0.67445683, 0.3220793 , 0.00346387],
       [0.19058325, 0.1395637 , 0.66985306],
       [0.38953368, 0.10630614, 0.50416017],
       [0.53891527, 0.45985535, 0.00122939],
       [0.7588464 , 0.2158235 , 0.0253301 ],
       [0.4699667 , 0.05470718, 0.47532612],
       [0.5911302 , 0.24190819, 0.16696161],
       [0.68803294, 0.30935528, 0.00261178],
       [0.70699895, 0.28786924, 0.00513181],
       [0.7488766 , 0.21916641, 0.03195698],
       [0.71317964, 0.2665672 , 0.02025316],
       [0.8085818 , 0.1752283 , 0.0161899 ],
       [0.70441951, 0.21905707, 0.07652343],
       [0.60960877, 0.38184107, 0.00855016],
       [0.5761585 , 0.38260778, 0.04123372],
       [0.8861692 , 0.06840359, 0.04542722],
       [0.59449295, 0.3296183 , 0.07588875],
       [0.4812114 , 0.49446707, 0.02432153],
       [0.33802073, 0.52311691, 0.13886235],
       [0.28764809, 0.11842382, 0.59392809],
       [0.23603862, 0.06913009, 0.69483128],
       [0.71754845, 0.26013093, 0.02232062],
       [0.26014751, 0.59132396, 0.14852853],
       [0.50099832, 0.30689334, 0.19210834],
       [0.47953791, 0.40353689, 0.1169252 ],
       [0.86703611, 0.09835412, 0.03460977],
       [0.39537849, 0.10494105, 0.49968046],
       [0.45930464, 0.53778153, 0.00291384],
       [0.61745081, 0.3802173 , 0.00233188],
       [0.5807746 , 0.35921815, 0.06000725],
       [0.67632189, 0.27894137, 0.04473674],
       [0.00780089, 0.00156449, 0.99063463],
       [0.73843508, 0.25032618, 0.01123874],
       [0.02979674, 0.00553675, 0.96466652],
       [0.61403063, 0.38437365, 0.00159572],
       [0.23603862, 0.06913009, 0.69483128],
       [0.71317964, 0.2665672 , 0.02025316],
       [0.61254389, 0.36863563, 0.01882048],
       [0.47110697, 0.18300289, 0.34589014],
       [0.79349453, 0.11677401, 0.08973146],
       [0.04730525, 0.01665552, 0.93603923],
       [0.86737531, 0.10222483, 0.03039985],
       [0.69382274, 0.29937659, 0.00680066],
       [0.62475331, 0.06431709, 0.3109296 ],
       [0.71877874, 0.26128953, 0.01993173],
       [0.88960389, 0.10353151, 0.0068646 ],
       [0.77569341, 0.1997806 , 0.02452599],
       [0.59343592, 0.30691105, 0.09965302],
       [0.59902142, 0.24082103, 0.16015754],
       [0.53320513, 0.40003568, 0.06675919],
       [0.50788604, 0.49059776, 0.0015162 ],
       [0.34714436, 0.1885783 , 0.46427733],
       [0.63957779, 0.34987041, 0.0105518 ],
       [0.00705896, 0.0015923 , 0.99134874],
       [0.58778296, 0.21827204, 0.193945  ],
       [0.61674078, 0.38164656, 0.00161266],
       [0.59385381, 0.31503242, 0.09111377],
```

```
[0.06636734, 0.01117606, 0.9224566 ],
[0.6848089 , 0.31378616, 0.00140494],
[0.63169918, 0.36444097, 0.00385985],
[0.5322331 , 0.2855693 , 0.1821976 ],
[0.64214636, 0.32266133, 0.03519231],
[0.43894227, 0.55273604, 0.00832169],
[0.64765417, 0.29223258, 0.06011325],
[0.22776964, 0.07036786, 0.7018625 ],
[0.36116676, 0.20856257, 0.43027067],
[0.5506327 , 0.36316346, 0.08620384],
[0.53554273, 0.44177985, 0.02267743],
[0.00344511, 0.00249393, 0.99406096],
[0.64606836, 0.27790081, 0.07603084],
[0.21068015, 0.10215062, 0.68716924],
[0.79418324, 0.20418257, 0.0016342 ],
[0.66882202, 0.32766696, 0.00351102],
[0.06786154, 0.01111417, 0.92102429],
[0.46483193, 0.18565872, 0.34950934],
[0.87089303, 0.12759693, 0.00151004],
[0.69287044, 0.2777135 , 0.02941607],
[0.03278126, 0.01800493, 0.94921381],
[0.47692077, 0.40570798, 0.11737125],
[0.89218646, 0.08983905, 0.01797449],
[0.54403362, 0.44143035, 0.01453603],
[0.38953368, 0.10630614, 0.50416017],
[0.23170608, 0.06977543, 0.69851849],
[0.39537849, 0.10494105, 0.49968046],
[0.68424679, 0.30650357, 0.00924964],
[0.71317964, 0.2665672 , 0.02025316],
[0.57183679, 0.27141581, 0.1567474 ],
[0.63368306, 0.3556308 , 0.01068615],
[0.61471318, 0.35645396, 0.02883286],
[0.72360603, 0.22135295, 0.05504102],
[0.06786154, 0.01111417, 0.92102429],
[0.38953368, 0.10630614, 0.50416017],
[0.33079372, 0.29482187, 0.37438441],
[0.08614733, 0.01437601, 0.89947666],
[0.49812597, 0.44129006, 0.06058397],
[0.77615201, 0.20398026, 0.01986773],
[0.24189695, 0.10684913, 0.65125391],
[0.89206067, 0.06981049, 0.03812884],
[0.51303871, 0.47714706, 0.00981424],
[0.67204862, 0.31466411, 0.01328727],
[0.0180746 , 0.00185707, 0.98006834],
[0.71920575, 0.26565678, 0.01513747],
[0.28112668, 0.1711979 , 0.54767542],
[0.89620657, 0.07365821, 0.03013522],
[0.05920629, 0.0240568 , 0.91673691],
[0.63987963, 0.28300272, 0.07711765],
[0.39271884, 0.5162229 , 0.09105826],
[0.88094317, 0.10823101, 0.01082582],
[0.61911031, 0.3693165 , 0.01157319],
[0.40055995, 0.55955312, 0.03988693],
[0.48208263, 0.49181294, 0.02610443],
[0.47056177, 0.43171186, 0.09772637],
[0.06636734, 0.01117606, 0.9224566 ],
[0.72978203, 0.25554448, 0.01467349],
[0.74571389, 0.23479339, 0.01949272],
[0.65470904, 0.22672355, 0.11856742],
[0.79463876, 0.1910897 , 0.01427155],
```

```
       [0.80869267, 0.13652576, 0.05478157],
       [0.31528418, 0.20103705, 0.48367877],
       [0.51722419, 0.47918926, 0.00358654],
       [0.37713872, 0.45040324, 0.17245804],
       [0.07225801, 0.02504117, 0.90270082],
       [0.61855717, 0.36284948, 0.01859335],
       [0.31050661, 0.16136695, 0.52812643],
       [0.65470904, 0.22672355, 0.11856742],
       [0.03908222, 0.00266026, 0.95825752],
       [0.02979674, 0.00553675, 0.96466652],
       [0.7590572 , 0.20212666, 0.03881615],
       [0.06786154, 0.01111417, 0.92102429],
       [0.05692699, 0.036267  , 0.90680601],
       [0.65970069, 0.33552315, 0.00477616],
       [0.03145444, 0.00255682, 0.96598874],
       [0.57819641, 0.33979744, 0.08200616]])
```

In [22]:
```
from sklearn.metrics import accuracy_score

acc_train = accuracy_score(y_train, clf.predict(X_train))
print(acc_train)
```

```
0.7279411764705882
```

En este caso vemos que el accuracy es superior a la contribución de la clase mayoritaria, no tiene mala pinta (pero recuerda que es el train). No vamos a ver el test, antes vamos a usar la validación cruzada y luego veremos las métricas para multiclase.

Validamos el modelo

Usemos la validación cruzada no tanto para comparar con otros modelos, que no tenemos, sino como para adelantarnos a la evaluación con el test y de nuevo ir cogiendo hábito.

In [23]:
```
from sklearn import model_selection
name='Logistic Regression'
cv_results = model_selection.cross_val_score(clf, X_train, y_train, cv=5, scorin

msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
print(cv_results)
print(msg)
```

```
[0.64285714 0.62962963 0.66666667 0.66666667 0.77777778]
Logistic Regression: 0.676720 (0.052496)
```

Ya vemos que sale menor que el 72% del train.

Evaluación del modelo

In [24]:
```
from sklearn.metrics import precision_score, recall_score, accuracy_score, balan
                            roc_auc_score, roc_curve, precision_recall_curve, co
```

In [25]:
```
y_proba = clf.predict_proba(X_test)
y_pred = clf.predict(X_test)
```

Matriz de confusion

A mano, como un dataframe a partir de los datos que nos da el `confusion_matrix` de sklearn:

In [26]:
```
cm = pd.DataFrame(confusion_matrix(y_test, y_pred), index=[f"Real {i}" for i in
                  columns=[f"Predicho {i}" for i in clf.classes_])

cm
```

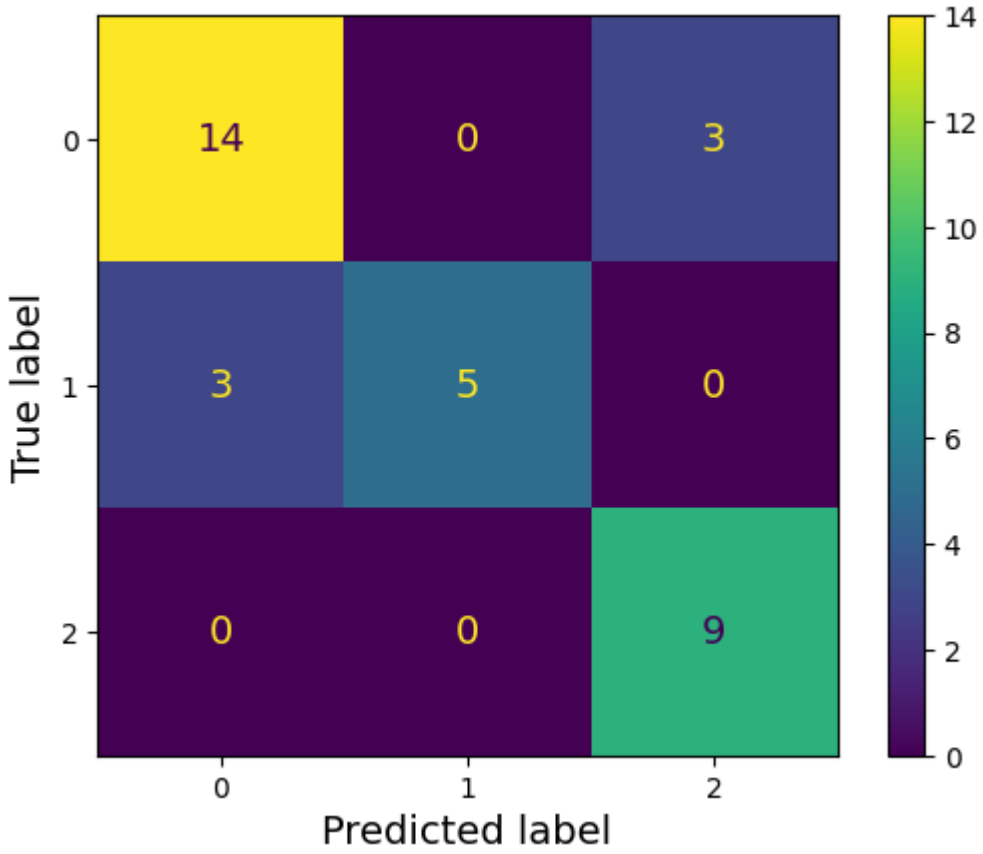
Out[26]:

	Predicho 0	Predicho 1	Predicho 2
Real 0	14	0	3
Real 1	3	5	0
Real 2	0	0	9

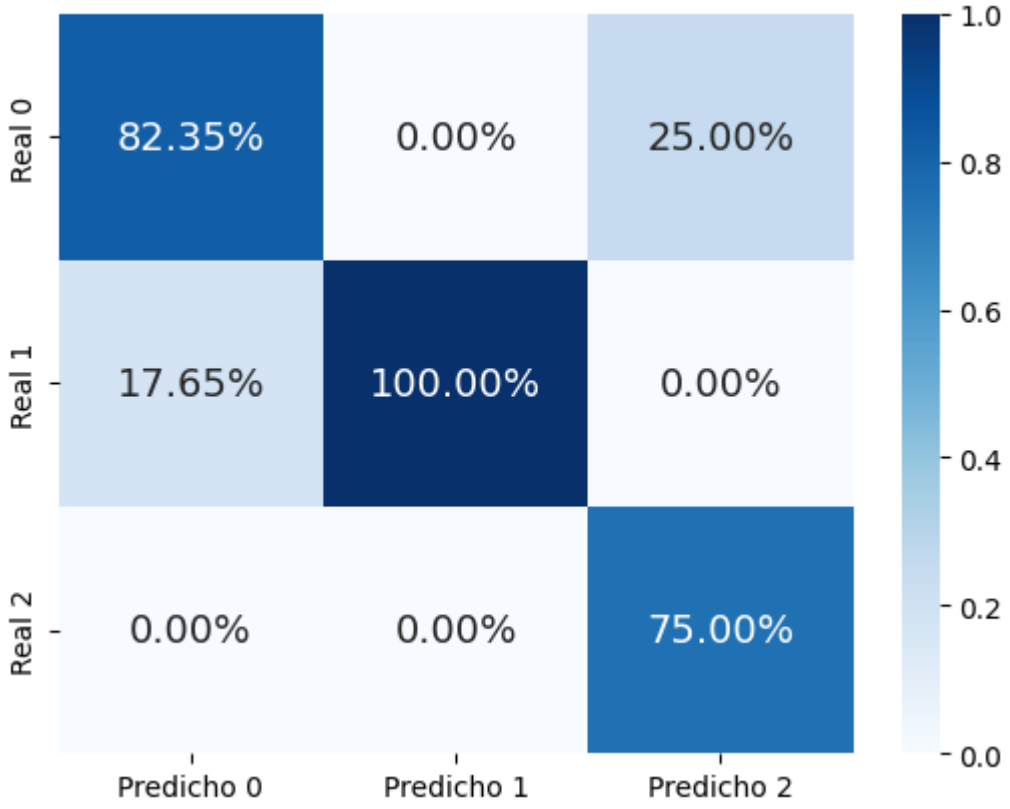
Gráficamente:

In [27]:
```
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
```

Out[27]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fb64e63f07
0>
```



In [28]:
```
sns.heatmap(cm/np.sum(cm), annot=True,
            fmt='.2%', cmap='Blues');
```



Métricas MACRO y MICRO

In [29]:
```
micro_prec = precision_score(y_test, y_pred, average="micro")
print("MICRO PRECISION:", micro_prec)

macro_prec = precision_score(y_test, y_pred, average="macro")
print("MACRO PRECISION:", macro_prec)

micro_rec = recall_score(y_test, y_pred, average="micro")
print("MICRO RECALL:", micro_rec)

macro_rec = recall_score(y_test, y_pred, average="macro")
print("MACRO RECALL:", macro_rec)

micro_acc = accuracy_score(y_test, y_pred)
print("MICRO ACCURACY:", micro_acc)
```

```
MICRO PRECISION: 0.8235294117647058
MACRO PRECISION: 0.8578431372549019
MICRO RECALL: 0.8235294117647058
MACRO RECALL: 0.8161764705882352
MICRO ACCURACY: 0.8235294117647058
```

In [30]:
```
acierto = accuracy_score(y_test, y_pred)

error = 1 - acierto
print("Acierto:", round(acierto*100, 2), "%")
print("Error:", round(error*100, 2), "%")
```

```
Acierto: 82.35 %
Error: 17.65 %
```

Informe de Resultados

In [31]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.82      0.82        17
           1       1.00      0.62      0.77         8
           2       0.75      1.00      0.86         9

    accuracy                           0.82        34
   macro avg       0.86      0.82      0.82        34
weighted avg       0.85      0.82      0.82        34
```

Clasificación de nuevos registros

Para terminar veamos como sería la predicción para un usuario que dedica una duración de 1, ve 1 página, ejecuta 1 acción y valora con 2 el site

In [3]:
```
new_data = pd.DataFrame(scaler.transform(np.log(np.array([[1, 1, 1, 2]]))),colum
print(new_data)
print(clf.predict_proba(new_data))
print(clf.predict(new_data))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 new_data = pd.DataFrame(scaler.transform(np.log(np.array([[1, 1, 1,
2]]))),columns = features)
      2 print(new_data)
      3 print(clf.predict_proba(new_data))

NameError: name 'scaler' is not defined
```

In [ ]:
```
```