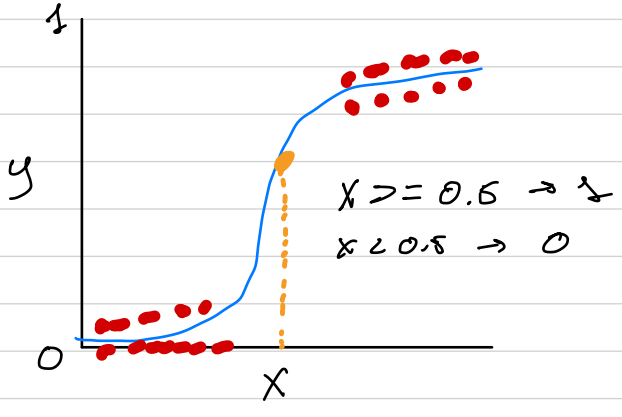
Otro algoritmo que permite clasificar datos en 2 posibles opciones, por ejemplo:

Si en una foto hay un perro o un gato.

si un correo es spam o no, etc.



En lugar de buscar un valor continuo (como en la regresión lineal) ajusta una función en forma de "S" que va desde cero a uno.

Esto nos sirve para calcular la probabilidad de un evento.

Ejercicio Titanic

① Analisis de los datos.

- leer el .csv
- datos.head() → Para hechar un vistazo a los datos.
- datos.describe() → Sumario de información
- graficar en base a sobrevivientes

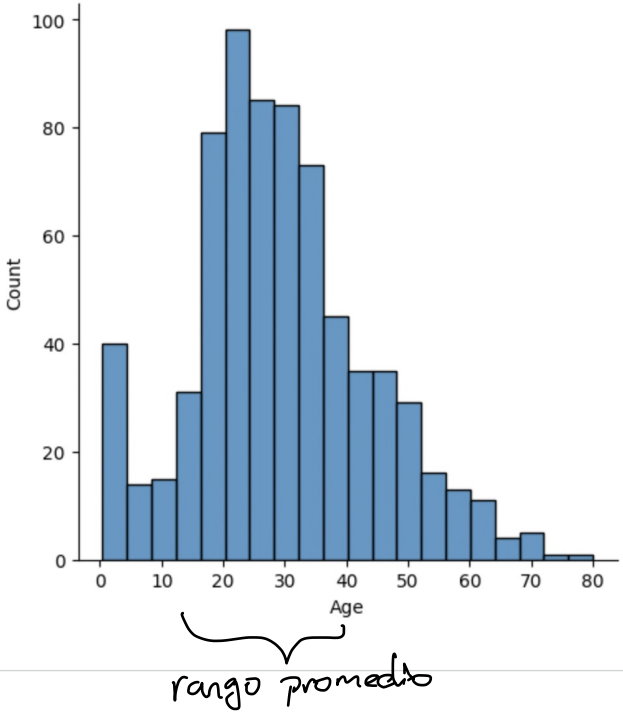
```
#ver la cantidad de gente que sobrevivio
import seaborn as sb

sb.countplot(x='Survived', data = datos)
```

```
# Es importante tener en el conjunto de datos si es hombre o mujer ?
# claramente, era mas facil sobrevivir si eras mujer
sb.countplot(x='Survived', data = datos, hue='Sex')
```

Lo siguiente es evaluar los datos no numericos (datos nulos, y con valores de tipo texto)

```
#ver los datos vacios

# datos.isna()

datos.isna().sum()

# la edad podria calcular un promedio, ya que, son muchos registros (en comparación con los datos)
# para eliminar

# La cabina, no es un dato tan relevante, si podria quitar esa columna

#embarcados, podriamos quitar los registros
```

```
# Si quiero ver cómo se distribuyen los valores de la edad
# puedo graficar usando:

sb.displot(x='Age', data=datos)
```



- Rellenar las filas de la edad

```
#calcular el promedio de edad, entre los datos vacios

datos['Age'].mean()
```

```
29.69911764705882
```

```
datos['Age'].fillna(datos['Age'].mean())
```

Forma más general

Esto no afecta al DF original, así que es necesario modificar el conjunto original.

```
datos['Age'] = datos['Age'].fillna(datos['Age'].mean())
```

Otra forma de hacerlo, separando por clase:

```
# rellenar las edades segun el promedio por clase

mean_age_pclass1 = datos[datos['Pclass'] == 3]['Age'].mean()

datos.loc[datos['Pclass'] == 3, 'Age'] = datos.loc[datos['Pclass'] == 3, 'Age'].fillna(mean_age_pclass1)
```

Tratando los datos de "cabin"

```
#Cabin, es una caracteristica, que la mayoria no lo tiene, los que si lo tienen no son consistentes
datos.isna().sum()

#se puede eliminar
datos = datos.drop(['Cabin'], axis=1)
```

```
# se pueden ver que hay varios datos que si son consistentes, y solo son 2 registros, los puedo eliminar
# sin que afecte mucho al resultado
datos['Embarked'].value_counts()
```

Al final se pueden eliminar los datos NA

datos.dropna()

Eliminar las columnas innecesarias

```
datos = datos.drop(['Ticket','PassengerId','Name'], axis=1)
```

Transformar el sexo en tipo dummies

```
# hay que transformar el Sexo

# en este caso es redundante ya que para la misma columna, solo pueden haber 2 posibles valores

# al condensarlo en una columna, se evita que el modelo se concentre en buscar una relación entre ambas columnas
# generadas por la función get_dummies

dummies_sex = pd.get_dummies(datos['Sex'], drop_first=True)
```

```
# se unen los datos
datos = datos.join(dummies_sex)
```

```
# eliminamos la columna 'Sex'
datos = datos.drop(['Sex'], axis=1)
```

Se puede graficar la información de embarcamiento

```
# se puede gravficar Embarked, para ver si puede ser relevante para el entrenamiento

sb.countplot(x='Survived', data=datos, hue='Embarked')

#la grafica nos dice, segun donde embarcaron, si sobrevivieron o no
```

```
# Creamos los dummies de la columna, quitando el primer elemento
dummies_embarked = pd.get_dummies(datos['Embarked'], drop_first=True)
```

```
# lo unimos al conjunto de datos
datos = datos.join(dummies_embarked)
datos = datos.drop(['Embarked'], axis=1)
```

→ Tambien podemos graficar los datos correlacionados.

```
# podermos ver el mapa de las correlaciones
sb.heatmap(datos.corr(), annot=True, cmap="YlGnBu")
```

Entre menor es el # clase, mayor la probabilidad de sobrevivir.

```
# otra forma de verlo

sb.countplot(x='Survived', data=datos,hue='Pclass')
```

Luego de finalizado el analisis, podemos entrenar el modelo.

```
#serparar las caracteristicas

X = datos.drop(['Survived'],axis=1)
y = datos['Survived']
```

```
#conjutnos de entrenamiento y pruebas
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=.2)
```

```
# Realizar el entrenamiento
from sklearn.linear_model import LogisticRegression

modelo = LogisticRegression(max_iter=1000)
```

```
#iniciar entrenamiento
modelo.fit(X_train,y_train)
```

Procedor con las predicciones

→ modelo.predict(x_test)

Evaluar las predicciones

from sklearn.metrics import accuracy_score

score= accuracy_score(y_test, predicciones)

Obtener la matriz de confusión

from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test, predicciones)

* Se puede crear un DataFrame

```
pd.DataFrame(confusion_matrix( y_test, predicciones)
 columns=['Pred no:', 'Pred si:'], index=['Real no:','Real si:'])
```

Crear otra persona;

```
nueva = [2,29,0,0,150,1,1,0]

pred = modelo.predict([nueva])

if pred[0] == 1:
  print("Sobreviviste")
else:
  print("F bro")
```