Selecciona cualquiera de los Challenge vistos en clase y programa un algoritmo que permita resolver el problema. Dicho algoritmo debe ser uno de los algoritmos vistos en el módulo (o que tu profesor de módulo autorice) haciendo uso de Scikit-learn. Lo que se busca es que demuestres tu conocimiento sobre el framework y como configurar el algoritmo.

In [32]:
```
#Importar pandas para poder leer el archivo

import pandas as pd

#Asignar una variable en la cual se almacenan todo el dataset, luego dos var
#una columna cada para poder asignar nuestra 'x' y 'y'

v = pd.read_csv("Valhalla23.csv")
X = v.drop('Celsius', axis=1)
y = v['Celsius']
```

Divide el set de datos del problema en dos subconjuntos, uno para entrenamiento y otro para prueba. Entrena tu modelo sobre el primer subconjunto, y por un mínimo de 100 iteraciones. Selecciona valores para la tasa de aprendizaje y para los parámetros iniciales, según tu criterio.

In [33]:
```
#Para hacer la division de los datos en dos subconjuntos haremos uso de trai
#Primero importaremos la libreria necesaria

from sklearn.model_selection import train_test_split

#Ahora haremos la division de las variables, utilizando 70% de los datos par
# y 30% de los datos para la variable de prueba

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

#Ahora haremos un entrenamiento con 100 iteraciones usando linear_model, ahc
# la libreria necesaria

from sklearn import linear_model
from sklearn.linear_model import LinearRegression

#Ahora asignaremos el entrenamiento a la variable modelo y usaremos la funci
# llevar a cabo el entrenamiento

modelo = linear_model.SGDRegressor(penalty= None, max_iter= 100, eta0=0.0001
modelo.fit(X_train, y_train)
```

Out[33]:

SGDRegressor

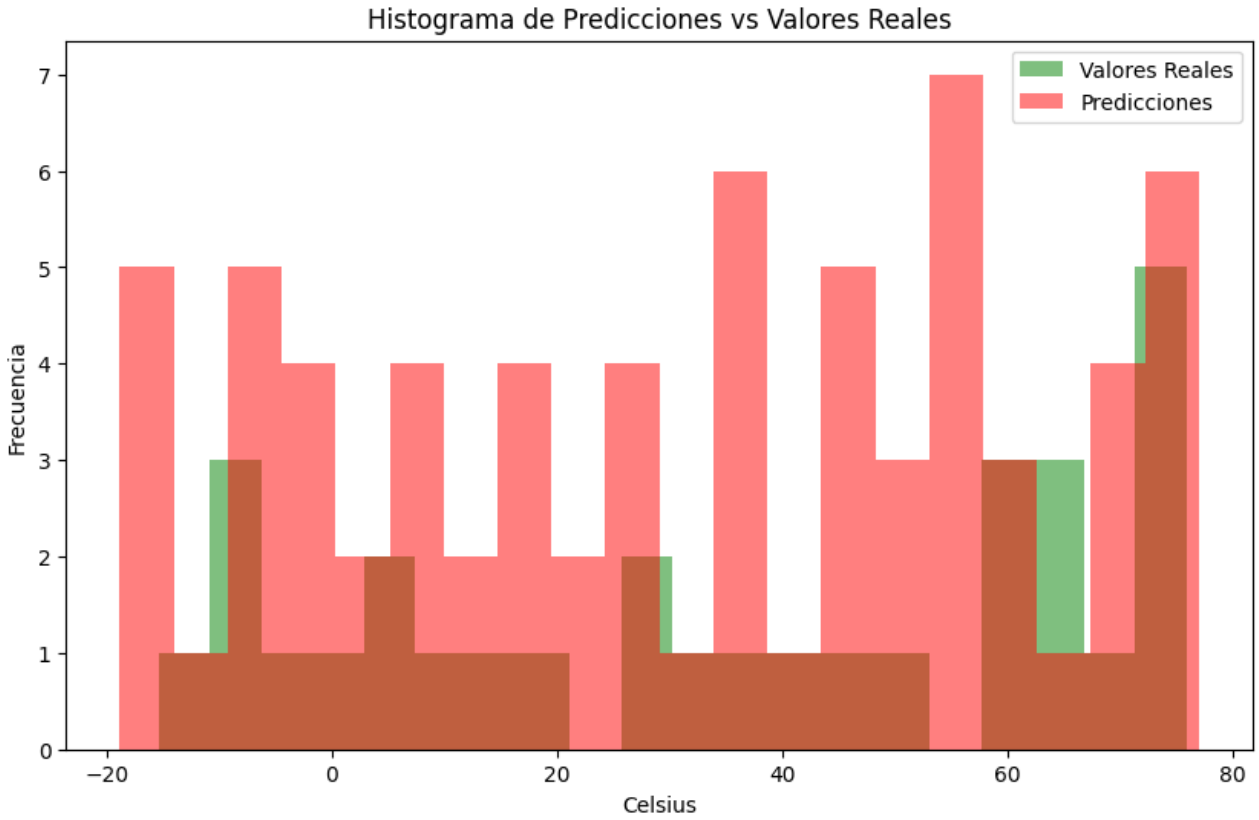
```
SGDRegressor(eta0=0.0001, max_iter=100, penalty=None)
```

Prueba tu implementación. Para ello, utiliza el modelo entrenado para hacer predecir las salidas del subconjunto de prueba, y compara contra los datos reales en una gráfica.

In [34]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.hist(y_test, bins=20, alpha=0.5, color='green', label='Valores Reales')
plt.hist(y_train, bins=20, alpha=0.5, color='red', label='Predicciones')
plt.xlabel('Celsius')
plt.ylabel('Frecuencia')
plt.title('Histograma de Predicciones vs Valores Reales')
plt.legend()
plt.show()
```



Calcula una métrica acorde a tu modelo, tanto para el subconjunto de entrenamiento, como para el subconjunto de prueba. Por ejemplo, si implementaste un algoritmo de regresión debes calcular el error cuadrático medio; si fue uno de clasificación, debes reportar las matrices de confusión y el f1-score.

In [36]:
```
#Para la revision de nuestras predicciones usaremos mean_squared_error, por
# improtaremos primero las librerias

from sklearn.metrics import mean_squared_error

#Ahora aplicaremos la funcion para ver el desempeño de nuestra variable

p = modelo.predict(X_test)
mse = mean_squared_error(y_test, p)
print('El mean squared error del modelo es:')
print(mse)
```

```
El mean squared error del modelo es:
200.84865380209976
```

Para facilitar la revisión, entrega dos archivos. El primero debe ser un Jupyter Notebook con todo el desarrollo (código comentado). El segundo debe ser un PDF del Jupyter Notebook. Revisa las instrucciones del entregable pasado para ver cómo exportar el archivo HTML y posteriormente pasarlo a PDF.

In [37]:
```
from google.colab import drive
drive.mount('/content/drive')
!jupyter nbconvert --to html '/content/drive/MyDrive/ColabNotebooks/Uso_de_f
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, cal
l drive.mount("/content/drive", force_remount=True).
[NbConvertApp] Converting notebook /content/drive/MyDrive/ColabNotebooks/Uso
_de_framework.ipynb to html
[NbConvertApp] Writing 626438 bytes to /content/drive/MyDrive/ColabNotebook
s/Uso_de_framework.html
```