1. Crea un espacio dentro del repositorio de GitHub que creaste para el portafolio de análisis (por ejemplo, una carpeta para el módulo de ML con una subcarpeta para este entregable).

In [ ]:
```
#Primero definiremos las librerias que se utilizaran
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDRegressor
```

2. Define una semilla que corresponda con los últimos cuatro dígitos de tu matrícula

In [ ]:
```
#La matricula es 01284657, asi que se escogen los ultimos 4 digitos
seed = 4657
```

3. Carga el set de datos de Valhalla y divide el set de datos en entrenamiento (40%), validación (40%), y prueba (20%), utilizando el método train_test_split. y la semilla definida arriba

In [ ]:
```
#Primero haremos la lectura del dataset y haremos la division de las columna
v = pd.read_csv('Valhalla23.csv')
x = v.drop(columns=['Valks'])
y = v['Valks']

#Primero haremos la division de dos variables con el train_test_split, una v
#test, la cual tendra el 20% de los datos, y la otra variable sera v_1, la c
x_1, x_test, y_1, y_test = train_test_split(x, y, test_size=0.2, random_stat

#Ahora con los datos restantes haremos la division de 40 y 40 a las variable
x_train, x_val, y_train, y_val = train_test_split(x_1, y_1, test_size=0.5, r

#Confirmamos los valores de las variables
print('x_train:', x_train.shape)
print('x_val:', x_val.shape)
print('x_test:', x_test.shape)
```

```
x_train: (40, 1)
x_val: (40, 1)
x_test: (20, 1)
```

4 .Entrena un modelo base de tipo SGDRegressor que utilice una tasa de aprendizaje de 1E-4, un máximo de iteraciones de un millón, y que utilice la semilla definida arriba

In [ ]:
```
#Se hace el entrenamiento con el uso de SGDRegressor y las especificaciones
model = SGDRegressor(learning_rate='constant', eta0=1E-4, max_iter=1000000,
model.fit(x_train, y_train)
```

Out[ ]:
```
SGDRegressor
SGDRegressor(eta0=0.0001, learning_rate='constant', max_iter=100000
0,
             random_state=4657)
```

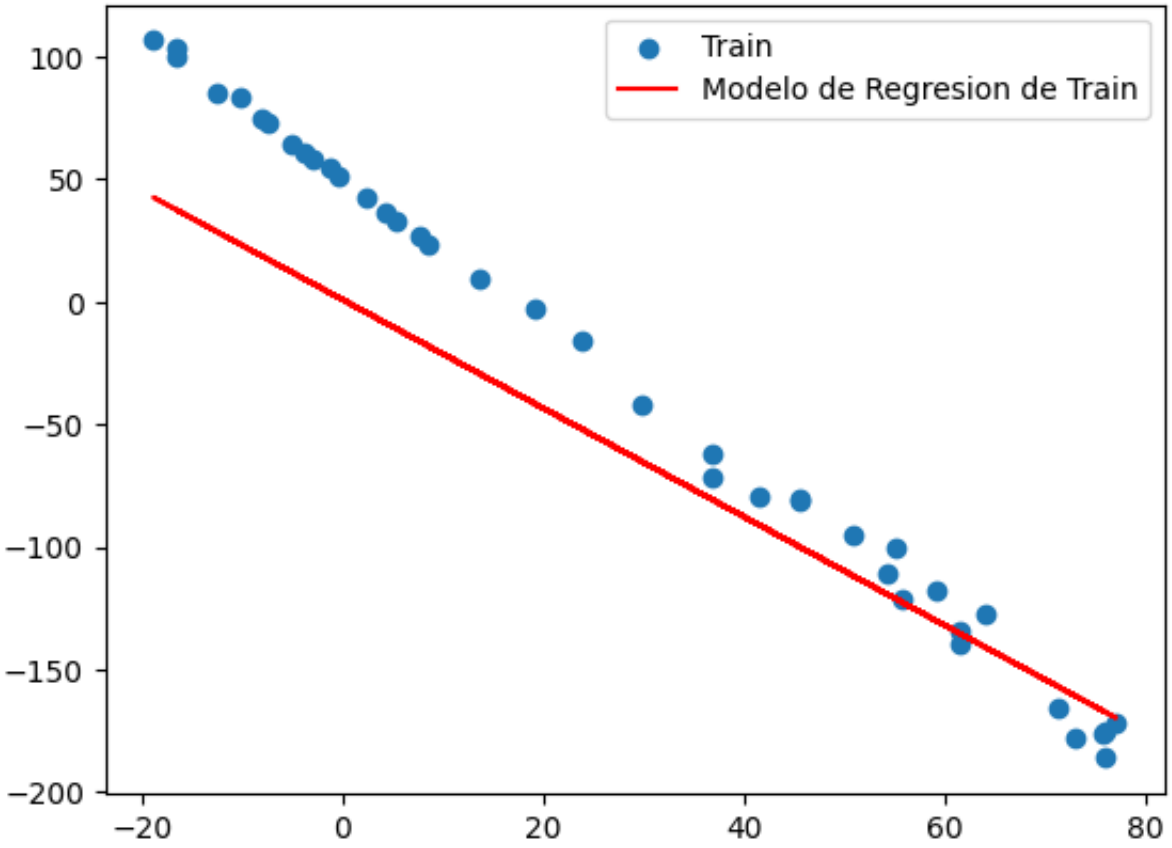
5. Calcula el error cuadrático medio para este modelo, sobre los datos de entrenamiento, validación, y prueba. Estos datos servirán como línea base.

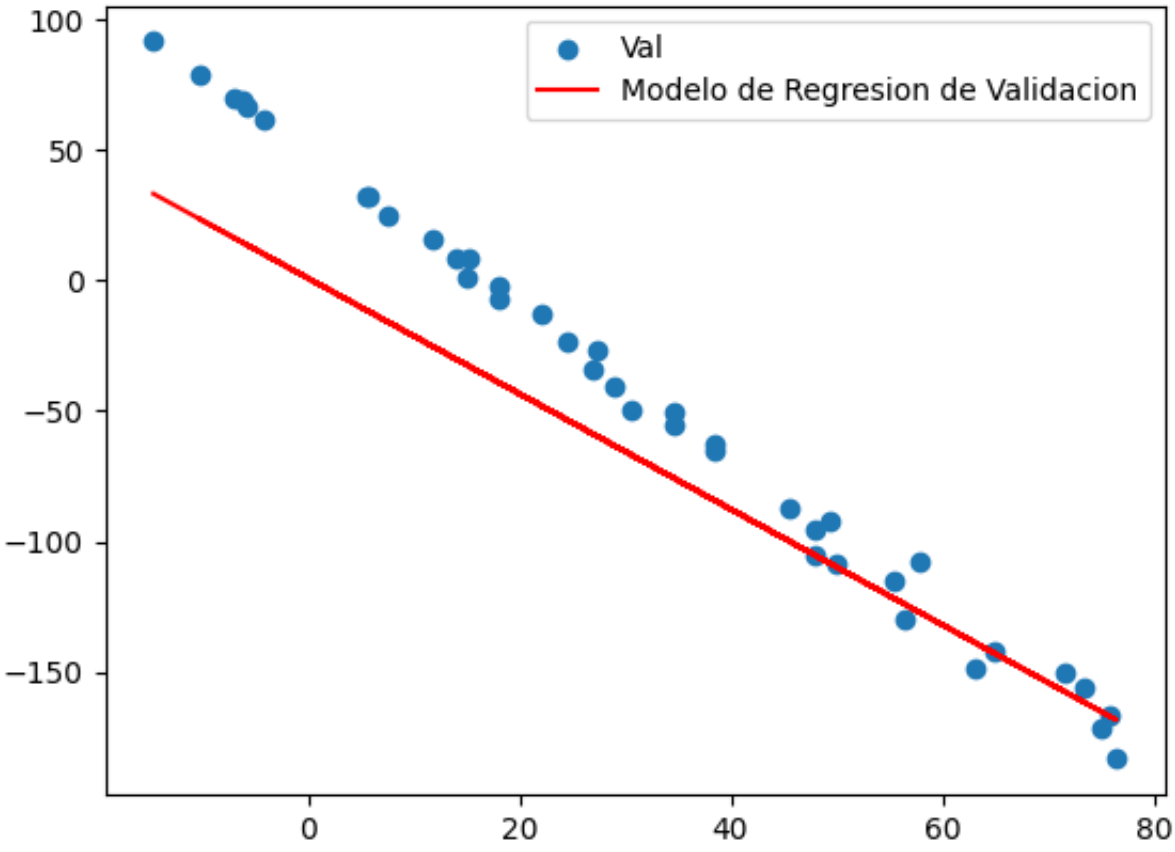
In [ ]:
```
#Primero hacemos la importacion de la funcion mean_squared_error
from sklearn.metrics import mean_squared_error

#Ahora sacaremos las errores cuadraticos medios.

y_val_pred = model.predict(x_val)
mse_val = mean_squared_error(y_val, y_val_pred)
print('El error cuadratico medio de la variable de valoracion es: ', mse_val

y_train_pred = model.predict(x_train)
mse_train = mean_squared_error(y_train, y_train_pred)
print('El error cuadratico medio de la variable de train es: ', mse_train)

y_test_pred = model.predict(x_test)
mse_test = mean_squared_error(y_test, y_test_pred)
print('El error cuadratico medio de la variable de test es: ', mse_test)
```

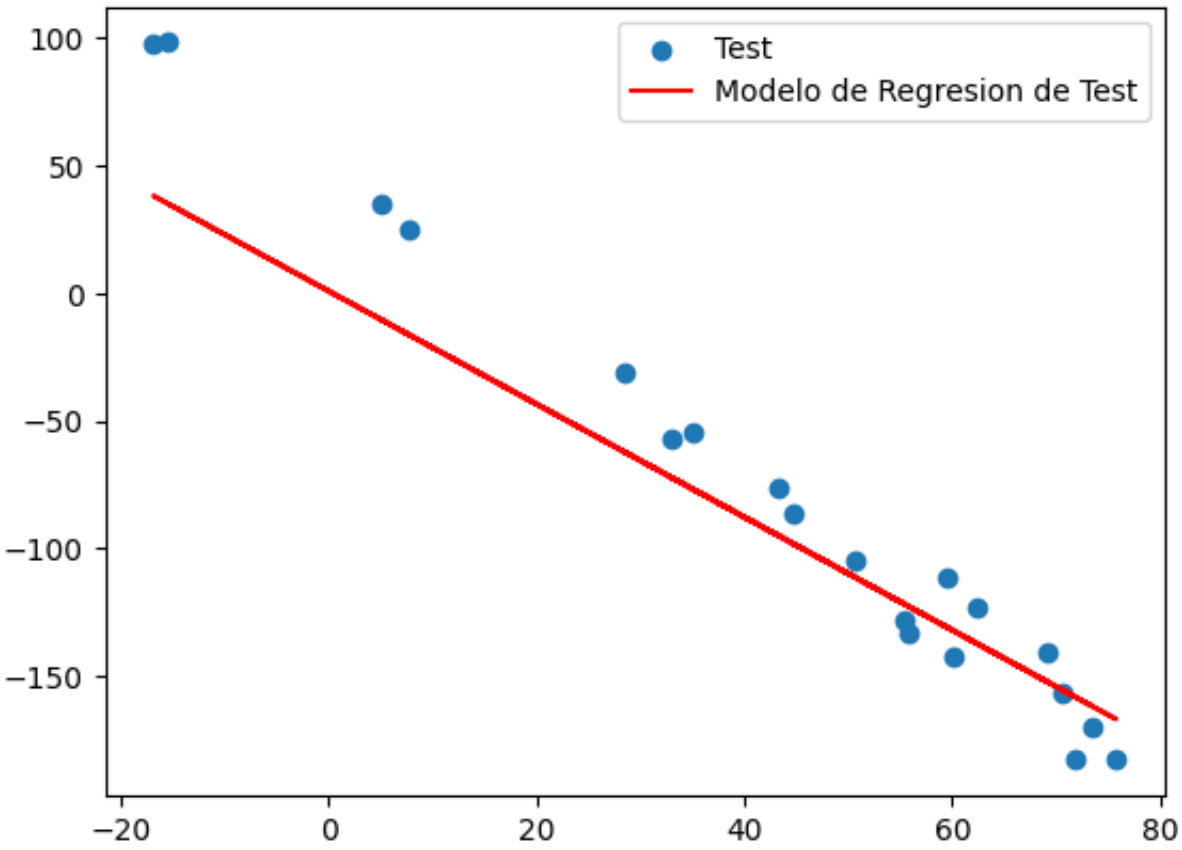
```
El error cuadratico medio de la variable de valoracion es:  991.981452306486
6
El error cuadratico medio de la variable de train es:  1406.1411863842443
El error cuadratico medio de la variable de test es:  775.7414893474745
```

6. Realiza una gráfica donde muestres cada subconjunto de datos (entrenamiento, validación, prueba) y el modelo de regresión obtenido (como una recta)

In [ ]:
```
#Primero importaremos plt para hacer los scatter plots
import matplotlib.pyplot as plt

#Haremos el scatter plot de train
plt.scatter(x_train, y_train, label='Train')
plt.plot(x_train, model.predict(x_train), color='red', label='Modelo de Regr
plt.legend()
plt.show()
plt.close()

#Haremos el scatter plot de val
plt.scatter(x_val, y_val, label='Val')
plt.plot(x_val, model.predict(x_val), color='red', label='Modelo de Regresic
plt.legend()
plt.show()
plt.close()

#Haremos el scatter plot de test
plt.scatter(x_test, y_test, label='Test')
plt.plot(x_test, model.predict(x_test), color='red', label='Modelo de Regres
plt.legend()
plt.show()
plt.close()
```


Train
Modelo de Regresion de Train
100
50
0
−50
−100
−150
−200
−20
0
20
40
60
80


Val
Modelo de Regresion de Validacion
100
50
0
−50
−100
−150
0
20
40
60
80



7. Crea una lista que contenga 20 elementos (enteros) entre 2 y 39 (sin repetición, y que incluyan el número 2). Estos valores representarán la cantidad de instancias que se usarán para el análisis

In [ ]:
```
#Primero importamos la libreria random
import random

#Luego creamos la lista empezando del 3 para luego insertar el valor de 2
#asi asegurandonos de que la lista lo contenga pero no se repita
lista = random.sample(range(3, 40), 19)

#Ahora insertamos el valor de 2 a la lista
lista.insert(0,2)

print(lista)
```

```
[2, 7, 15, 28, 21, 24, 37, 34, 33, 22, 17, 12, 27, 32, 14, 29, 16, 39, 38,
6]
```

8. Para cada uno de los tamaños del punto anterior, entrena 100 modelos usando un subconjunto aleatorio del set de entrenamiento que contenga esa cantidad de muestras. Por ejemplo, para el tamaño de 2 muestras, se deben entrenar 100 modelos utilizando 2 muestras seleccionadas aleatoriamente de las 40 muestras disponibles en el set de entrenamiento

In [ ]:
```
import numpy as np

#Primero asignaremos una variable que contenga los resultados

resultados = {}

#Utilizaremos un for para hacer los entrenamientos
for i in lista:
    modelos_train = []
    pred_train = []

    for _ in range(100):
        valor_r = random.sample(range(len(x_train)), i)
        x_train_1 = x_train.iloc[valor_r]
        y_train_1 = y_train.iloc[valor_r]

        modelo1 = SGDRegressor(learning_rate='constant', eta0=1E-4, max_iter
        modelo1.fit(x_train_1, y_train_1)

        modelos_train.append(modelo1)

    resultados[i] = {'modelos': modelos_train, 'x_train_1': x_train_1, 'y_tr
```

9. Para cada uno de los modelos del punto anterior, calcula el error cuadrático medio en el subconjunto de entrenamiento (el que tiene un número cambiante de muestras), y en el subconjunto de validación

In [ ]:
```
#Primero crearemos mse_resultados para almacenar los valores de MSE, tambien
mse_resultados = {}

e_train = {tamaño: [] for tamaño in lista}
e_val = {tamaño: [] for tamaño in lista}
```

```
#Creamos un for el cual calcula el MSE de los datos de train y validacion
for tamaño,data in resultados.items():
  modelos = data['modelos']
  x_train_1 = data['x_train_1']
  y_train_1 = data['y_train_1']
  x_train = data['x_train']
  y_train = data['y_train']

  mse_train_tamaño = []
  mse_val_tamaño = []

  for modelo in modelos:
        #Predicción de entrenamiento
        pred_train = modelo.predict(x_train_1)
        mse_train = mean_squared_error(y_train_1, pred_train)
        mse_train_tamaño.append(mse_train)

        #Predicción de validación
        pred_val = modelo.predict(x_val)
        mse_val = mean_squared_error(y_val, pred_val)
        mse_val_tamaño.append(mse_val)

  #Asignamos los promedios de los valores de MSE tanto de train como de vali
  mse_resultados[tamaño] = {'mse_train': np.mean(mse_train_tamaño),'mse_val'
```

10. Calcula el promedio de las 100 repeticiones para cada uno de los modelos y sus errores. Esto debería generar dos listas de 20 valores cada uno, donde cada elemento representa el error promedio de las 100 repeticiones que se hicieron para cada subconjunto de entrenamiento

In [ ]:
```
#Generamos variables en las cuales se van a guardar los promedios de train y
promedio_train = {}
promedio_val = {}

#Hacemos un for el cual guarde el valor de cada MSE de train y validacion en
for tamaño in lista:
    promedio_train[tamaño] = mse_resultados[tamaño]['mse_train']
    promedio_val[tamaño] = mse_resultados[tamaño]['mse_val']


print(promedio_train)
print(promedio_val)
print(len(promedio_train))
```

```
print(len(promedio_val))
```

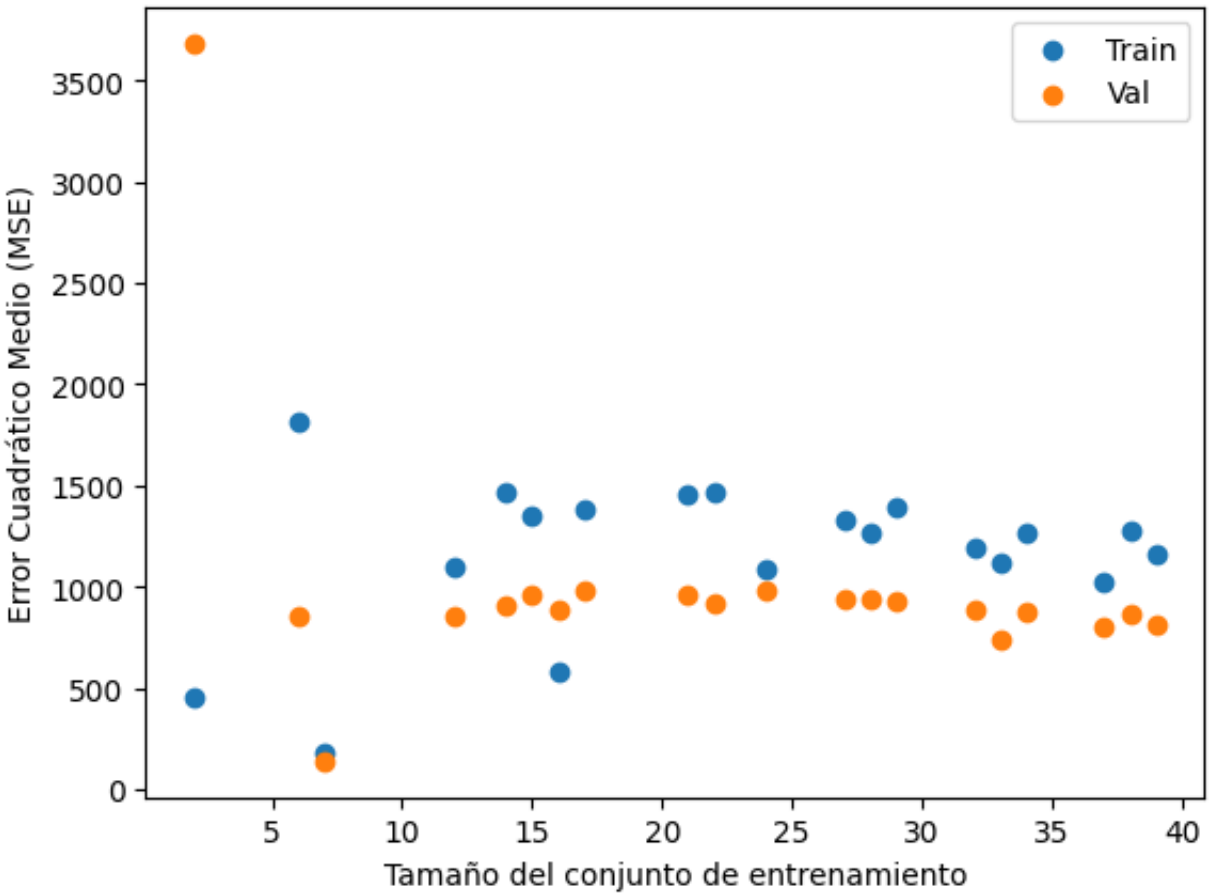
```
{2: 449.4282346801361, 7: 178.24820143779655, 15: 1346.6403062966012, 28: 12
66.1612051316388, 21: 1460.5352380440722, 24: 1087.1701115704877, 37: 1022.3
905407064549, 34: 1263.1173907636207, 33: 1118.5192851095385, 22: 1471.93792
25513123, 17: 1383.191387215354, 12: 1093.8647984476133, 27: 1327.4182575366
93, 32: 1195.936436649951, 14: 1465.199943811292, 29: 1389.0705428947676, 1
6: 577.0715223432177, 39: 1162.5691437025441, 38: 1280.7697458054158, 6: 181
1.2910815842167}
{2: 3683.50394836514, 7: 138.39154388317695, 15: 960.6258141518887, 28: 943.
9349382207241, 21: 959.585404113065, 24: 985.5773191322482, 37: 804.16637883
54325, 34: 880.6738486153878, 33: 735.8825863869477, 22: 920.0668109646006,
17: 982.2013859191512, 12: 855.025197574265, 27: 941.3013255075482, 32: 889.
1232312390401, 14: 907.8759655081906, 29: 925.2756362847368, 16: 887.0010849
923038, 39: 812.9631996682413, 38: 868.3305831452459, 6: 851.8538756031571}
20
20
```

11. Agrega a las listas anteriores los errores de entrenamiento y validación de la línea base

In [ ]:
```
#Hacemos un append el cual agregue los valores
e_train[tamaño].append(mse_train)
e_val[tamaño].append(mse_val)
```

12. Haz una gráfica donde se muestre la evolución del error promedio de entrenamiento y validación, para cada uno de los diferentes tamaños de entrenamiento

In [ ]:
```
plt.scatter(x=lista, y=promedio_train.values(), label='Train')
plt.scatter(x=lista, y=promedio_val.values(), label='Val')
plt.xlabel('Tamaño del conjunto de entrenamiento')
plt.ylabel('Error Cuadrático Medio (MSE)')
plt.legend()
plt.show()
```



13. Con base en la grafica anterior, explica el tipo de ajuste obtenido para el primer modelo (el entrenado sobre 2 muestras) y para el modelo final (el entrenado sobre 40 muestras). También explica como cambia el tipo de ajuste a medida que se incrementa el número de muestras del entrenamiento. Incluye también en tu análisis el grado de sesgo y de varianza para los diferentes modelos.

En el primer modelo podemos ver que la prediccion fue poco acertada, esto se debe a que el entrenamiento fue de 2, lo cual es muy poco y no le permite a un aprendizaje automatizado hacer la correlacion entre las variables, por lo cual fue muy poco eficiente, en cambio con el ultimo modelo se puede ver un menor valor de MSE y una cercania entre las variables mucho mejor, lo cual demuestra la importancia del entrenamiento optimo, pero tampoco se puede decir que la mayor cantidad siempre es la mejor, se tiene que encontrar un intervalo de vairables donde sea vea un mejor valor tanto de

cercania de prediccion y menor MSE. En el primero modelo se puede ver un sobreajuste sobre las variables debido a su poco entrenamiento, demostrando que tiene una poca capacidad de hacer predicciones, y conforme van aumentando el entrenamiento este sobreajuste mejora visiblemente, teniendo menor MSE y distancia en general.

En cuanto a la varianza el sesgo, el primer modelo muestra sobreajuste, lo cual indica una varianza alta, esto siendo malo para un modelo de ML, despues podemos ver como rapidamente ambos aspectos mejoran con el aumento de los modelos.

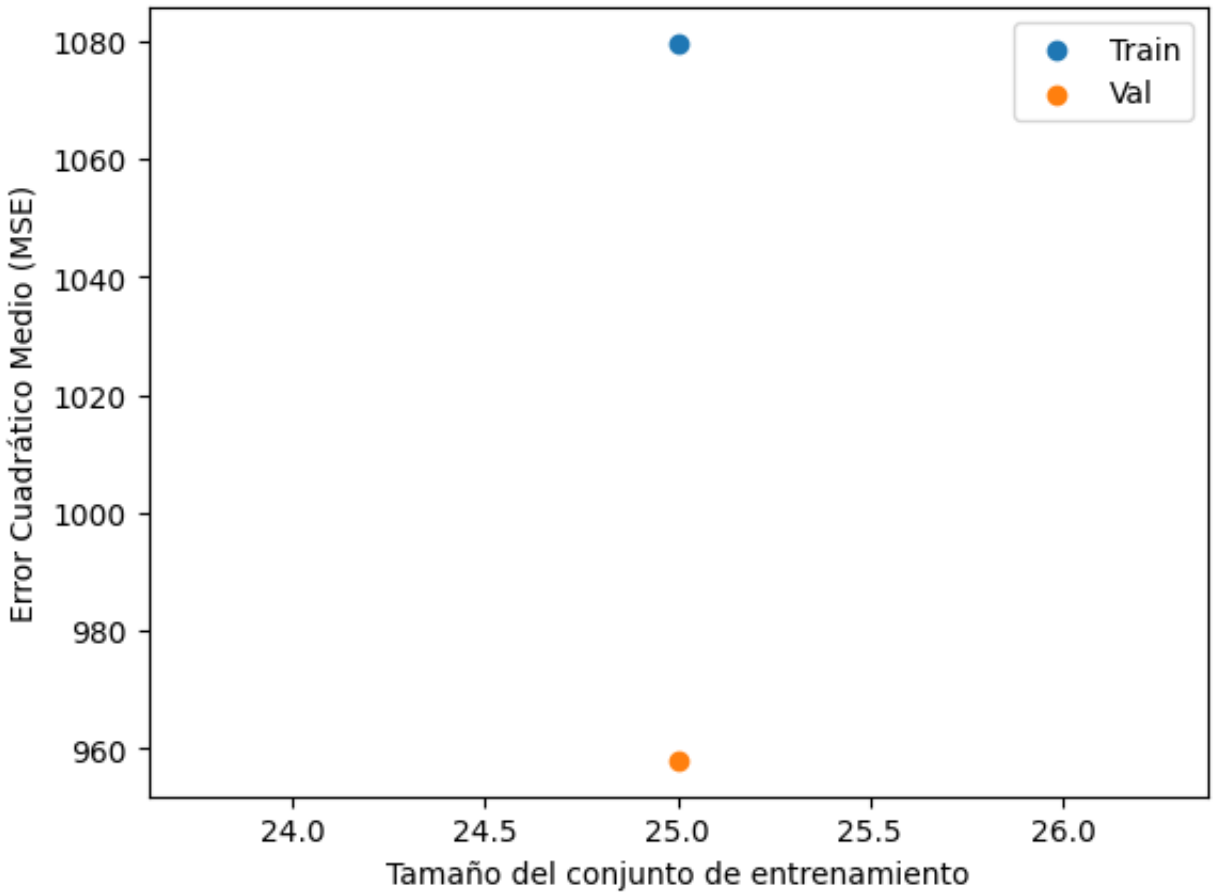
14. Con base en la gráfica y los datos, identifica la cantidad de muestras más adecuada para realizar el entrenamiento. Justifica tu selección.

Un valor optimo de muestras podria ser 25, ya que muestra tanto un valor de MSE bajo en comparacion a los demas modelos y tambien muestra una mejor prediccion, aunque los modelos cercanos tambien se pueden considerar buenos, pero el 25 se muestra como el mejor.

15. Entrena un nuevo modelo utilizando esa cantidad de muestras, y calcula su error cuadrático medio sobre el subconjunto de entrenamiento (el de la cantidad de muestras seleccionadas), el de validación, y el de prueba.

In [ ]:
```
#Primero haremos la nueva muestra con el valor de 25 ya que ese fue el mas c
muestras_nuevo = random.sample(range(len(x_train)), 25)
x_train1 = x_train.iloc[muestras_nuevo]
y_train1 = y_train.iloc[muestras_nuevo]

#Ahora usaremos SGDRRegressor para hacer el nuevo entrenamiento
n_modelo = SGDRegressor(learning_rate='constant', eta0=1E-4, max_iter=100000
n_modelo.fit(x_train1, y_train1)

#Calcularemos el MSE para el entrenamiento
y_train1_pred = n_modelo.predict(x_train1)
mse_train1 = mean_squared_error(y_train1, y_train1_pred)

#Calcularemos el MSE para la validacion
y_val_pred1 = n_modelo.predict(x_val)
mse_val1 = mean_squared_error(y_val, y_val_pred1)
```

```
#Calcularemos el MSE para el test
y_test_pred1 = n_modelo.predict(x_test)
mse_test1 = mean_squared_error(y_test, y_test_pred1)

#Se hara la grafica para tener representacion visual y poder hacer uin anali
plt.scatter(x=25 , y=mse_train1, label='Train')
plt.scatter(x=25 , y=mse_val1, label='Val')
plt.xlabel('Tamaño del conjunto de entrenamiento')
plt.ylabel('Error Cuadrático Medio (MSE)')
plt.legend()
plt.show()
```



16. Compara los valores del punto anterior contra los errores obtenidos para la línea base (ver punto 5)

In [ ]:
```
print("MSE de Entrenamiento")
print("Nuevo modelo:", mse_train1)
print("Modelo base: 775.7414893474745")

print("MSE de Validacion")
```

```
print("Nuevo modelo:", mse_val1)
print("Modelo base: 991.9814523064866")

print("MSE de Test")
print("Nuevo modelo:", mse_test1)
print("Modelo base: 1406.1411863842443")
```

```
MSE de Entrenamiento
Nuevo modelo: 1079.6077835164435
Modelo base: 775.7414893474745
MSE de Validacion
Nuevo modelo: 957.9395325503432
Modelo base: 991.9814523064866
MSE de Test
Nuevo modelo: 784.7579365017518
Modelo base: 1406.1411863842443
```

17. Argumenta cuál configuración funcionó mejor, y por qué

Viendo la comparacion de los MSE, podemos ver que aunque el entrenamiento tuvo un valor mas alto en el modelo nuevo, se pueden observar mejores valores tanto el la validacion y en test, lo cual indica que se esta haciendo una mejor prediccion que en el modelo base, dando un claro ejemplo de la importancia de un entrenamiento correcto de modelos de ML.

La razon por la cual se puede llegar a tener mejores valores de MSE es que se entreno con el valor mas optimo de valores, por lo cual se encontro el balance entre muy poco entrenamiento y sobre entrenamiento, normalizando lo mayor posible la varianza y el sesgo.