Import the libraries

```
In [ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, SGDRegressor
from sklearn.model_selection import train_test_split
np.random.seed(42)
```

Read the data

```
In [ ]: df = pd.read_csv('Valhalla23.csv')
df.head()
```

Out[ ]:

	Celsius	Valks
0	61.4720	-139.740
1	70.5790	-156.600
2	-7.3013	73.269
3	71.3380	-165.420
4	43.2360	-75.835

```
In [ ]: X = df["Celsius"]
y = df["Valks"]
```

Split the data into train and test sets

```
In [ ]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

Scale the data

```
In [ ]: scaler = StandardScaler()
X_train = scaler.fit_transform(np.array(X_train).reshape(-1, 1))
X_test = scaler.transform(np.array(X_test).reshape(-1, 1))
```

Create the model

```
In [ ]: learning_rate = 0.01 # is sufficent to avoid some divergences in the model and star
initial_intercept = 0.5 # it dont assume bias between positive and negative values
initial_coef = 0.1 # small value

model = SGDRegressor(
    max_iter=1000,
    n_iter_no_change=100,  # Min 100 iterations
    tol=1e-5,  # Tolerance for early stopping
    learning_rate='constant',
```

```
    eta0=learning_rate,
    random_state=42
)

model.intercept_ = initial_intercept
model.coef_ = initial_coef

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

The explanation of the hyperparameters:

Learning rate of 0.01: It's a compromise between speed and stability. Smaller values like 0.001 might be too slow, while larger values like 0.1 might cause instability.

Initial intercept of 0.5: This value is chosen to start the model at a neutral point. It's halfway between 0 and 1, which are common ranges for normalized data. It avoids introducing any initial bias towards positive or negative predictions.

Initial coefficient of 0.1: This small value allows the model to start with weak feature influences and gradually strengthen them. It helps prevent the model from making strong initial predictions before learning from the data.

Max iterations of 1000: This is just for the model to converge without excessive computational cost.

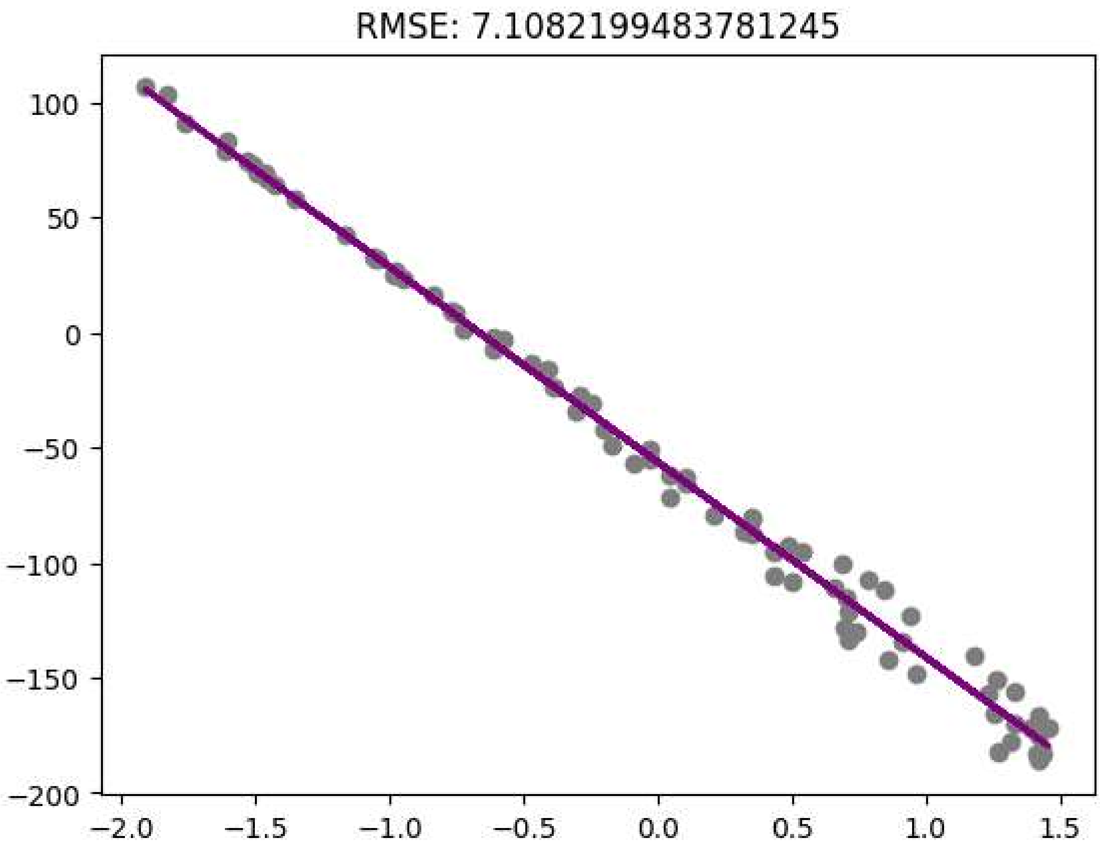
N_iter_no_change of 100: This forces the model to train for at least 100 iterations, giving it a chance to learn even if early improvements are small. (A parameter given by the professor).

Tolerance of 1e-5: This small value ensures the model continues training until very minor improvements stop occurring. It strikes a balance between precision and avoiding overfitting.

Train results

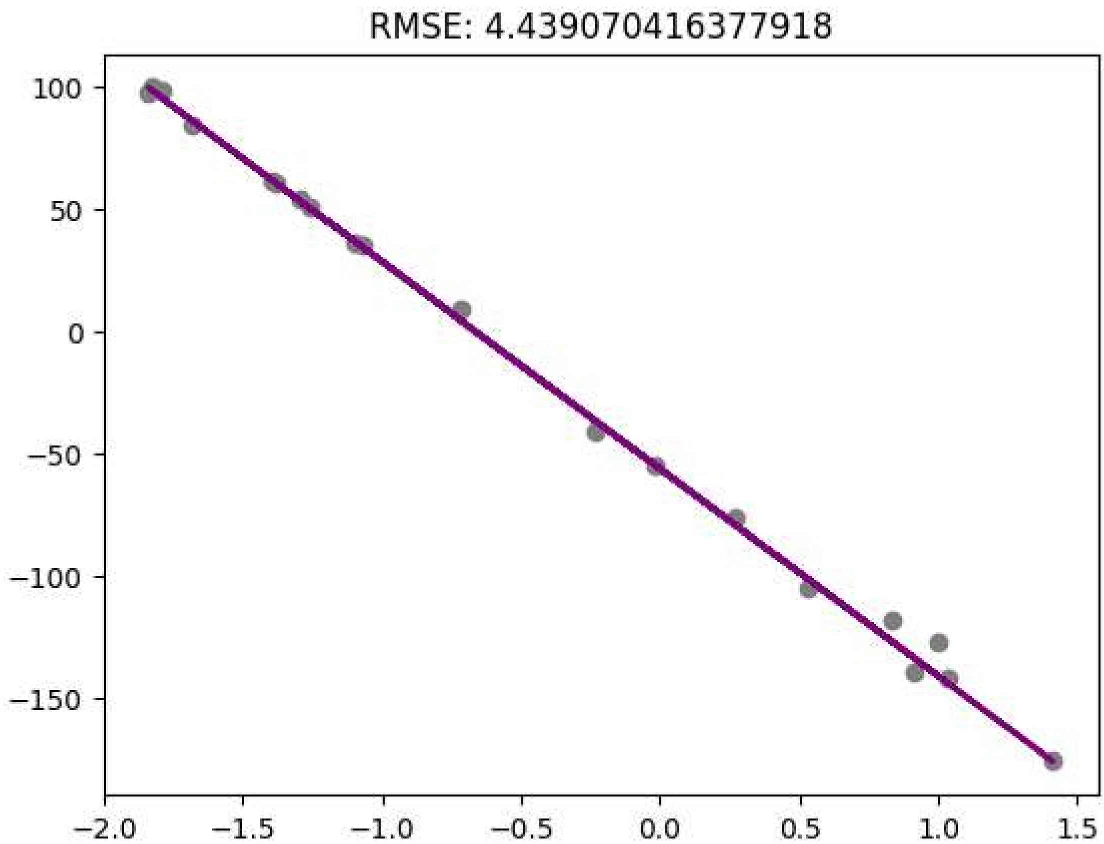
In [ ]:

```
y_train_pred = model.predict(X_train)

rmse = np.sqrt(np.mean((y_train_pred - y_train) ** 2))

plt.scatter(X_train, y_train, color='gray')
plt.plot(X_train, y_train_pred, color='purple', linewidth=2)
plt.title(f'RMSE: {rmse}')
plt.show()
```



Test results

In [ ]:
```
rmse = np.sqrt(np.mean((y_pred - y_test)**2))
plt.scatter(X_test, y_test, color='gray')
plt.plot(X_test, y_pred, color='purple', linewidth=2)
plt.title(f'RMSE: {rmse}')
plt.show()
```


RMSE: 4.439070416377918
100
50
0
−50
−100
−150
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5