In [8]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

In [9]:
```
# Cargar los datos
df = pd.read_csv("train.csv")

# Seleccionamos solo variables numéricas
df_num = df.select_dtypes(include=[np.number])

# Eliminamos las filas con valores nulos (opcional según la estrategia)
df_num = df_num.dropna()

# Separar variables predictoras (X) y variable objetivo (y)
X = df_num.drop(columns=["SalePrice", "Id"])
y = df_num["SalePrice"]

# Normalizar los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# División de los datos
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, ran
```

In [10]:
```
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
```

In [11]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input

model = Sequential()
model.add(Input(shape=(X_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))  # salida de regresión

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)

y_pred_nn = model.predict(X_test).flatten()
```

```
WARNING:tensorflow:5 out of the last 17 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x000002777B088400> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
```

In [12]:
```
# Regresión Lineal
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

# Red neuronal
mse_nn = mean_squared_error(y_test, y_pred_nn)
r2_nn = r2_score(y_test, y_pred_nn)

print(f"Regresión Lineal -> MSE: {mse_lr:.2f}, R²: {r2_lr:.2f}")
print(f"Red Neuronal     -> MSE: {mse_nn:.2f}, R²: {r2_nn:.2f}")
```

```
Regresión Lineal -> MSE: 1588229760.73, R²: 0.80
Red Neuronal     -> MSE: 6286142976.00, R²: 0.20
```

In [13]:
```
nuevos_datos = X_test[:5]
pred_lr = lr.predict(nuevos_datos)
pred_nn = model.predict(nuevos_datos).flatten()

print("Predicciones con regresión lineal:", pred_lr)
print("Predicciones con red neuronal:    ", pred_nn)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
Predicciones con regresión lineal: [149575.3367402  158230.46370932 108496.29825731
141537.83886374
 157079.77923371]
Predicciones con red neuronal:     [48165.188 35407.355 68021.94  42380.375 13463.57
5]
```

In [14]:
```
resultados = pd.DataFrame({
    "Real": y_test[:5].values,
    "Regresión Lineal": pred_lr,
    "Red Neuronal": pred_nn
})
print(resultados)
```

```
     Real  Regresión Lineal  Red Neuronal
0  140000     149575.336740  48165.187500
1  150750     158230.463709  35407.355469
2  157000     108496.298257  68021.937500
3  138000     141537.838864  42380.375000
4  144000     157079.779234  13463.575195
```

In [ ]:

In [ ]: