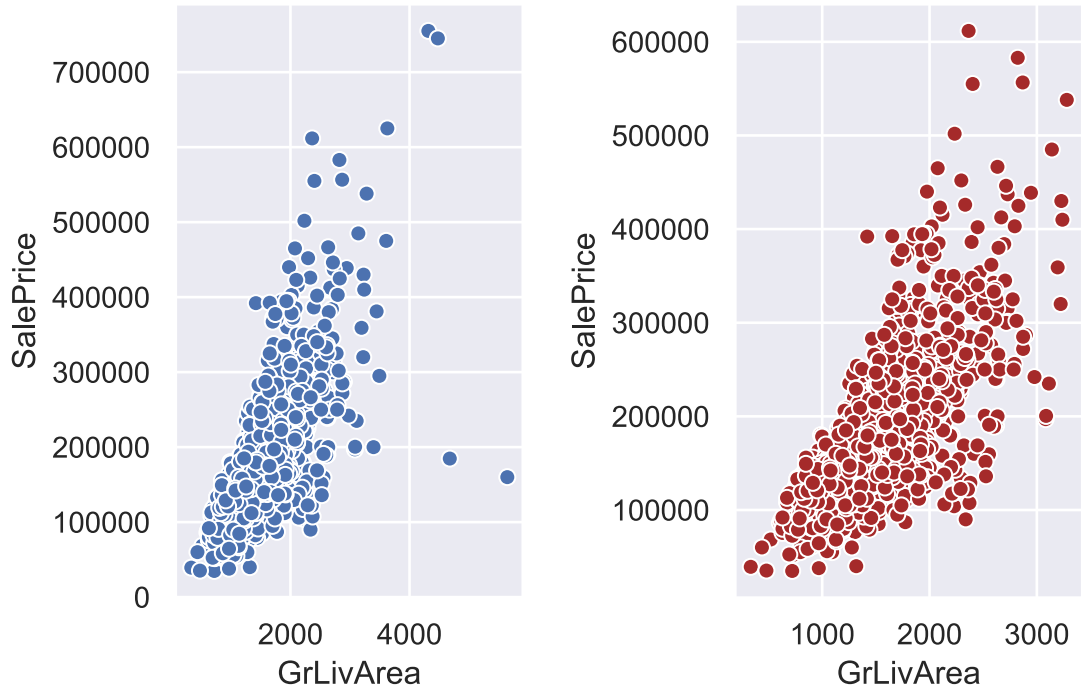
In [18]:
```
%matplotlib inline
import d2l
from mxnet import autograd, gluon, init, nd
from mxnet.gluon import data as gdata, loss as gloss, nn, utils
import numpy as np
import pandas as pd
import seaborn as sns
import scipy
sns.set()
import matplotlib.pyplot as plt



train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')


ig, axes = plt.subplots(nrows = 1, ncols = 2)
sns.scatterplot(x = train_data.GrLivArea, y = train_data.SalePrice, data
= train_data, ax = axes[0])

train_data = train_data[~(np.abs(train_data.GrLivArea - train_data.GrLiv
Area.mean()) > (3.5 * train_data.GrLivArea.std()))]
#print(f"Updated shape of the dataset: {train.shape}")
sns.scatterplot(x = train_data.GrLivArea, y = train_data.SalePrice, data
= train_data, ax = axes[1], color = "brown")

plt.tight_layout()

#ASSISTED WITH CODE FROM VAGHULB1992
```



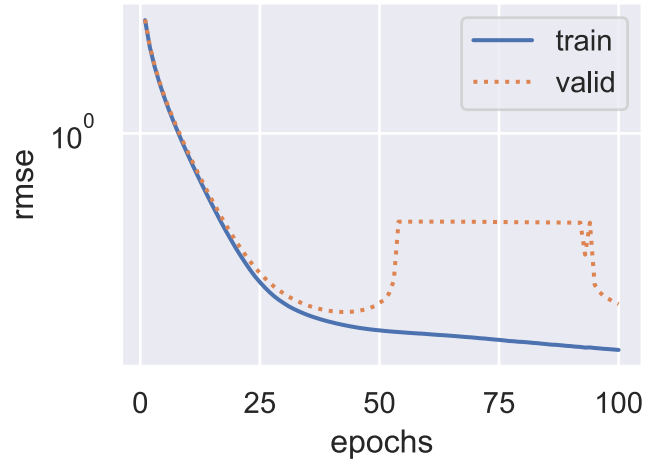
In [19]:
```
all_features = pd.concat((train_data.iloc[:, 1:-1], test_data.iloc[:, 1
:]))
numeric_features = all_features.dtypes[all_features.dtypes != 'object'].
index
all_features[numeric_features] = all_features[numeric_features].apply(
    lambda x: (x - x.mean()) / (x.std()))
# after standardizing the data all means vanish, hence we can set missin
g values to 0
all_features = all_features.fillna(0)
all_features = pd.get_dummies(all_features, dummy_na=True)
n_train = train_data.shape[0]
train_features = nd.array(all_features[:n_train].values)
test_features = nd.array(all_features[n_train:].values)
train_labels = nd.array(train_data.SalePrice.values).reshape((-1, 1))
train_data["SalePrice"] = np.log1p(train_data["SalePrice"])
```

In [20]:
```
loss = gloss.L2Loss()

def get_net():
    net = nn.Sequential()
    net.add(nn.Dense(1))
    net.initialize()
    return net
def log_rmse(net, features, labels):
    # To further stabilize the value when the logarithm is taken, set th
e value less than 1 as 1.
    clipped_preds = nd.clip(net(features), 1, float('inf'))
    rmse = nd.sqrt(2 * loss(clipped_preds.log(), labels.log()).mean())
    return rmse.asscalar()
def train(net, train_features, train_labels, test_features, test_labels,
          num_epochs, learning_rate, weight_decay, batch_size):
    train_ls, test_ls = [], []
    train_iter = gdata.DataLoader(gdata.ArrayDataset(
        train_features, train_labels), batch_size, shuffle=True)
    # The Adam optimization algorithm is used here.
    trainer = gluon.Trainer(net.collect_params(), 'adam', {
        'learning_rate': learning_rate, 'wd': weight_decay})
    for epoch in range(num_epochs):
        for X, y in train_iter:
            with autograd.record():
                l = loss(net(X), y)
            l.backward()
            trainer.step(batch_size)
        train_ls.append(log_rmse(net, train_features, train_labels))
        if test_labels is not None:
            test_ls.append(log_rmse(net, test_features, test_labels))
    return train_ls, test_ls
def get_k_fold_data(k, i, X, y):
    assert k > 1
    fold_size = X.shape[0] // k
    X_train, y_train = None, None
    for j in range(k):
        idx = slice(j * fold_size, (j + 1) * fold_size)
        X_part, y_part = X[idx, :], y[idx]
        if j == i:
            X_valid, y_valid = X_part, y_part
        elif X_train is None:
            X_train, y_train = X_part, y_part
        else:
            X_train = nd.concat(X_train, X_part, dim=0)
            y_train = nd.concat(y_train, y_part, dim=0)
    return X_train, y_train, X_valid, y_valid
def k_fold(k, X_train, y_train, num_epochs,
           learning_rate, weight_decay, batch_size):
    train_l_sum, valid_l_sum = 0, 0
    for i in range(k):
        data = get_k_fold_data(k, i, X_train, y_train)
        net = get_net()
        train_ls, valid_ls = train(net, *data, num_epochs, learning_rate
,
                                   weight_decay, batch_size)
        train_l_sum += train_ls[-1]
```

```
        valid_l_sum += valid_ls[-1]
        if i == 0:
            d2l.semilogy(range(1, num_epochs + 1), train_ls, 'epochs',
'rmse',
                         range(1, num_epochs + 1), valid_ls,
                         ['train', 'valid'])
        print('fold %d, train rmse: %f, valid rmse: %f' % (
            i, train_ls[-1], valid_ls[-1]))
    return train_l_sum / k, valid_l_sum / k
def train_and_pred(train_features, test_feature, train_labels, test_data
,
                   num_epochs, lr, weight_decay, batch_size):
    net = get_net()
    train_ls, _ = train(net, train_features, train_labels, None, None,
                        num_epochs, lr, weight_decay, batch_size)
    d2l.semilogy(range(1, num_epochs + 1), train_ls, 'epochs', 'rmse')
    print('train rmse %f' % train_ls[-1])
    # apply the network to the test set
    preds = net(test_features).asnumpy()
    # reformat it for export to Kaggle
    test_data['SalePrice'] = pd.Series(preds.reshape(1, -1)[0])
    submission = pd.concat([test_data['Id'], test_data['SalePrice']], ax
is=1)
    submission.to_csv('submission.csv', index=False)
```

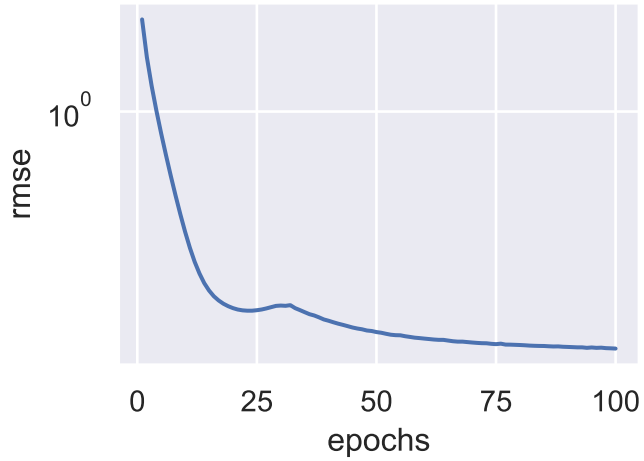
In [22]:

```
k, num_epochs, lr, weight_decay, batch_size = 2, 100, 3, 0, 10
train_l, valid_l = k_fold(k, train_features, train_labels, num_epochs, l
r,
                          weight_decay, batch_size)
print('%d-fold validation: avg train rmse: %f, avg valid rmse: %f'
      % (k, train_l, valid_l))
```




```
fold 0, train rmse: 0.125822, valid rmse: 0.195271
fold 1, train rmse: 0.136007, valid rmse: 0.137343
2-fold validation: avg train rmse: 0.130915, avg valid rmse: 0.166307
```

In [23]:
```
train_and_pred(train_features, test_features, train_labels, test_data,
               num_epochs, lr, weight_decay, batch_size)
```




```
train rmse 0.117930
```

In [ ]:
```

```

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission.csv	just now	1 seconds	0 seconds	0.14613

Complete

Jump to your position on the leaderboard

2541	**CJ Jang**		0.14613	5	now

Your Best Entry ↑

You advanced 263 places on the leaderboard!

Your submission scored 0.14613, which is an improvement of your previous score of 0.15296. Great job!

Tweet this!