This notebook is an exercise in the Intermediate Machine Learning course. You can reference the tutorial at this link.

By encoding **categorical variables**, you'll obtain your best results thus far!

Setup

The questions below will give you feedback on your work. Run the following cell to set up the feedback system.

In [1]:
```
# Set up code checking
import os
if not os.path.exists("../input/train.csv"):
    os.symlink("../input/home-data-for-ml-course/train.csv", "../input/train.csv")
    os.symlink("../input/home-data-for-ml-course/test.csv", "../input/test.csv")
from learntools.core import binder
binder.bind(globals())
from learntools.ml_intermediate.ex3 import *
print("Setup Complete")
```

```
Setup Complete
```

In this exercise, you will work with data from the Housing Prices Competition for Kaggle Learn Users.

Run the next code cell without changes to load the training and validation sets in `X_train`, `X_valid`, `y_train`, and `y_valid`. The test set is loaded in `X_test`.

In [2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Read the data
X = pd.read_csv('../input/train.csv', index_col='Id')
X_test = pd.read_csv('../input/test.csv', index_col='Id')

# Remove rows with missing target, separate target from predictors
X.dropna(axis=0, subset=['SalePrice'], inplace=True)
y = X.SalePrice
X.drop(['SalePrice'], axis=1, inplace=True)

# To keep things simple, we'll drop columns with missing values
cols_with_missing = [col for col in X.columns if X[col].isnull().any()]
X.drop(cols_with_missing, axis=1, inplace=True)
X_test.drop(cols_with_missing, axis=1, inplace=True)

# Break off validation set from training data
```

```
X_train, X_valid, y_train, y_valid = train_test_split(X, y,
                                                      train_size=0.8, test_size=0.2,
                                                      random_state=0)
```

Use the next code cell to print the first five rows of the data.

In [3]:
```
X_train.head()
```

Out[3]:

Id	MSSubClass	MSZoning	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope
619	20	RL	11694	Pave	Reg	Lvl	AllPub	Inside	Gtl
871	20	RL	6600	Pave	Reg	Lvl	AllPub	Inside	Gtl
93	30	RL	13360	Pave	IR1	HLS	AllPub	Inside	Gtl
818	20	RL	13265	Pave	IR1	Lvl	AllPub	CulDSac	Gtl
303	20	RL	13704	Pave	IR1	Lvl	AllPub	Corner	Gtl

5 rows × 60 columns

Notice that the dataset contains both numerical and categorical variables. You'll need to encode the categorical data before training a model.

To compare different models, you'll use the same `score_dataset()` function from the tutorial. This function reports the mean absolute error (MAE) from a random forest model.

In [4]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# function for comparing different approaches
def score_dataset(X_train, X_valid, y_train, y_valid):
    model = RandomForestRegressor(n_estimators=100, random_state=0)
    model.fit(X_train, y_train)
    preds = model.predict(X_valid)
    return mean_absolute_error(y_valid, preds)
```

Step 1: Drop columns with categorical data

You'll get started with the most straightforward approach. Use the code cell below to preprocess the data in `X_train` and `X_valid` to remove columns with categorical data. Set the preprocessed DataFrames to `drop_X_train` and `drop_X_valid`, respectively.

In [5]:
```
# Fill in the lines below: drop columns in training and validation data
drop_X_train = X_train.select_dtypes(exclude= ['object'])
drop_X_valid = X_valid.select_dtypes(exclude= ['object'])

# Check your answers
step_1.check()
```

Correct

In [6]:
```
# Lines below will give you a hint or solution code
#step_1.hint()
# step_1.solution()
```

Run the next code cell to get the MAE for this approach.

In [7]:
```
print("MAE from Approach 1 (Drop categorical variables):")
print(score_dataset(drop_X_train, drop_X_valid, y_train, y_valid))
```

```
MAE from Approach 1 (Drop categorical variables):
17837.82570776256
```

Before jumping into ordinal encoding, we'll investigate the dataset. Specifically, we'll look at the `'Condition2'` column. The code cell below prints the unique entries in both the training and validation sets.

In [8]:
```
print("Unique values in 'Condition2' column in training data:", X_train['Condition2'].u
print("\nUnique values in 'Condition2' column in validation data:", X_valid['Condition2
```

```
Unique values in 'Condition2' column in training data: ['Norm' 'PosA' 'Feedr' 'PosN' 'Ar
tery' 'RRAe']

Unique values in 'Condition2' column in validation data: ['Norm' 'RRAn' 'RRNn' 'Artery'
'Feedr' 'PosN']
```

Step 2: Ordinal encoding

Part A

If you now write code to:

- fit an ordinal encoder to the training data, and then
- use it to transform both the training and validation data,

you'll get an error. Can you see why this is the case? (*You'll need to use the above output to answer this question.*)

In [9]:
```
# Check your answer (Run this code cell to receive credit!)
step_2.a.check()
```

Correct:

Fitting an ordinal encoder to a column in the training data creates a corresponding integer-valued label for each unique value **that appears in the training data**. In the case that the validation data contains values that don't also appear in the training data, the encoder will throw an error, because these values won't have an integer assigned to them. Notice that the `'Condition2'` column in

the validation data contains the values `'RRAn'` and `'RRNn'`, but these don't appear in the training data -- thus, if we try to use an ordinal encoder with scikit-learn, the code will throw an error.

In [10]:
```
#step_2.a.hint()
```

This is a common problem that you'll encounter with real-world data, and there are many approaches to fixing this issue. For instance, you can write a custom ordinal encoder to deal with new categories. The simplest approach, however, is to drop the problematic categorical columns.

Run the code cell below to save the problematic columns to a Python list `bad_label_cols`. Likewise, columns that can be safely ordinal encoded are stored in `good_label_cols`.

In [11]:
```
# Categorical columns in the training data
object_cols = [col for col in X_train.columns if X_train[col].dtype == "object"]

# Columns that can be safely ordinal encoded
good_label_cols = [col for col in object_cols if
                   set(X_valid[col]).issubset(set(X_train[col]))]

# Problematic columns that will be dropped from the dataset
bad_label_cols = list(set(object_cols)-set(good_label_cols))

print('Categorical columns that will be ordinal encoded:', good_label_cols)
print('\nCategorical columns that will be dropped from the dataset:', bad_label_cols)
```

```
Categorical columns that will be ordinal encoded: ['MSZoning', 'Street', 'LotShape', 'La
ndContour', 'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'BldgTy
pe', 'HouseStyle', 'RoofStyle', 'Exterior1st', 'Exterior2nd', 'ExterQual', 'ExterCond',
'Foundation', 'Heating', 'HeatingQC', 'CentralAir', 'KitchenQual', 'PavedDrive', 'SaleTy
pe', 'SaleCondition']

Categorical columns that will be dropped from the dataset: ['Condition2', 'Functional',
'RoofMatl']
```

Part B

Use the next code cell to ordinal encode the data in `X_train` and `X_valid`. Set the preprocessed DataFrames to `label_X_train` and `label_X_valid`, respectively.

- We have provided code below to drop the categorical columns in `bad_label_cols` from the dataset.
- You should ordinal encode the categorical columns in `good_label_cols`.

In [12]:
```
from sklearn.preprocessing import OrdinalEncoder

# Drop categorical columns that will not be encoded
label_X_train = X_train.drop(bad_label_cols, axis=1)
label_X_valid = X_valid.drop(bad_label_cols, axis=1)

# Apply ordinal encoder
enc = OrdinalEncoder()
```

```
label_X_train[good_label_cols]= enc.fit_transform(X_train[good_label_cols])
label_X_valid[good_label_cols]= enc.transform(X_valid[good_label_cols])

# Check your answer
step_2.b.check()
```

Correct

In [13]:
```
# Lines below will give you a hint or solution code
#step_2.b.hint()
#step_2.b.solution()
```

Run the next code cell to get the MAE for this approach.

In [14]:
```
print("MAE from Approach 2 (Ordinal Encoding):")
print(score_dataset(label_X_train, label_X_valid, y_train, y_valid))
```

```
MAE from Approach 2 (Ordinal Encoding):
17098.01649543379
```

So far, you've tried two different approaches to dealing with categorical variables. And, you've seen that encoding categorical data yields better results than removing columns from the dataset.

Soon, you'll try one-hot encoding. Before then, there's one additional topic we need to cover. Begin by running the next code cell without changes.

In [15]:
```
# Get number of unique entries in each column with categorical data
object_nunique = list(map(lambda col: X_train[col].nunique(), object_cols))
d = dict(zip(object_cols, object_nunique))

# Print number of unique entries by column, in ascending order
sorted(d.items(), key=lambda x: x[1])
```

Out[15]:
```
[('Street', 2),
 ('Utilities', 2),
 ('CentralAir', 2),
 ('LandSlope', 3),
 ('PavedDrive', 3),
 ('LotShape', 4),
 ('LandContour', 4),
 ('ExterQual', 4),
 ('KitchenQual', 4),
 ('MSZoning', 5),
 ('LotConfig', 5),
 ('BldgType', 5),
 ('ExterCond', 5),
 ('HeatingQC', 5),
 ('Condition2', 6),
 ('RoofStyle', 6),
 ('Foundation', 6),
 ('Heating', 6),
 ('Functional', 6),
 ('SaleCondition', 6),
 ('RoofMatl', 7),
 ('HouseStyle', 8),
```

```
('Condition1', 9),
('SaleType', 9),
('Exterior1st', 15),
('Exterior2nd', 16),
('Neighborhood', 25)]
```

Step 3: Investigating cardinality

Part A

The output above shows, for each column with categorical data, the number of unique values in the column. For instance, the `'Street'` column in the training data has two unique values: `'Grvl'` and `'Pave'`, corresponding to a gravel road and a paved road, respectively.

We refer to the number of unique entries of a categorical variable as the **cardinality** of that categorical variable. For instance, the `'Street'` variable has cardinality 2.

Use the output above to answer the questions below.

In [16]:
```
# Fill in the line below: How many categorical variables in the training data
# have cardinality greater than 10?
high_cardinality_numcols = 3

# Fill in the line below: How many columns are needed to one-hot encode the
# 'Neighborhood' variable in the training data?
num_cols_neighborhood = 25

# Check your answers
step_3.a.check()
```

Correct

In [17]:
```
# Lines below will give you a hint or solution code
#step_3.a.hint()
#step_3.a.solution()
```

Part B

For large datasets with many rows, one-hot encoding can greatly expand the size of the dataset. For this reason, we typically will only one-hot encode columns with relatively low cardinality. Then, high cardinality columns can either be dropped from the dataset, or we can use ordinal encoding.

As an example, consider a dataset with 10,000 rows, and containing one categorical column with 100 unique entries.

- If this column is replaced with the corresponding one-hot encoding, how many entries are added to the dataset?
- If we instead replace the column with the ordinal encoding, how many entries are added?

Use your answers to fill in the lines below.

In [18]:
```
# Fill in the line below: How many entries are added to the dataset by
# replacing the column with a one-hot encoding?
OH_entries_added = 1e4*100 - 1e4

# Fill in the line below: How many entries are added to the dataset by
# replacing the column with an ordinal encoding?
label_entries_added = 0

# Check your answers
step_3.b.check()
```

Correct

In [19]:
```
# Lines below will give you a hint or solution code
#step_3.b.hint()
#step_3.b.solution()
```

Next, you'll experiment with one-hot encoding. But, instead of encoding all of the categorical variables in the dataset, you'll only create a one-hot encoding for columns with cardinality less than 10.

Run the code cell below without changes to set `low_cardinality_cols` to a Python list containing the columns that will be one-hot encoded. Likewise, `high_cardinality_cols` contains a list of categorical columns that will be dropped from the dataset.

In [20]:
```
# Columns that will be one-hot encoded
low_cardinality_cols = [col for col in object_cols if X_train[col].nunique() < 10]

# Columns that will be dropped from the dataset
high_cardinality_cols = list(set(object_cols)-set(low_cardinality_cols))

print('Categorical columns that will be one-hot encoded:', low_cardinality_cols)
print('\nCategorical columns that will be dropped from the dataset:', high_cardinality_
```

```
Categorical columns that will be one-hot encoded: ['MSZoning', 'Street', 'LotShape', 'La
ndContour', 'Utilities', 'LotConfig', 'LandSlope', 'Condition1', 'Condition2', 'BldgTyp
e', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'ExterQual', 'ExterCond', 'Foundation', 'Heat
ing', 'HeatingQC', 'CentralAir', 'KitchenQual', 'Functional', 'PavedDrive', 'SaleType',
'SaleCondition']

Categorical columns that will be dropped from the dataset: ['Neighborhood', 'Exterior2n
d', 'Exterior1st']
```

Step 4: One-hot encoding

Use the next code cell to one-hot encode the data in `X_train` and `X_valid`. Set the preprocessed DataFrames to `OH_X_train` and `OH_X_valid`, respectively.

- The full list of categorical columns in the dataset can be found in the Python list `object_cols`.
- You should only one-hot encode the categorical columns in `low_cardinality_cols`. All other categorical columns should be dropped from the dataset.

In [21]:
```
from sklearn.preprocessing import OneHotEncoder

# Use as many lines of code as you need!

OH_encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
OH_cols_train = pd.DataFrame(OH_encoder.fit_transform(X_train[low_cardinality_cols]))
OH_cols_valid = pd.DataFrame(OH_encoder.transform(X_valid[low_cardinality_cols]))

# One-hot encoding removed index; put it back
OH_cols_train.index = X_train.index
OH_cols_valid.index = X_valid.index

# Remove categorical columns (will replace with one-hot encoding)
num_X_train = X_train.drop(object_cols, axis=1)
num_X_valid = X_valid.drop(object_cols, axis=1)

# Add one-hot encoded columns to numerical features
OH_X_train = pd.concat([num_X_train, OH_cols_train], axis=1)
OH_X_valid = pd.concat([num_X_valid, OH_cols_valid], axis=1)

# Check your answer
step_4.check()
```

Correct

In [22]:
```
# Lines below will give you a hint or solution code
#step_4.hint()
#step_4.solution()
```

Run the next code cell to get the MAE for this approach.

In [23]:
```
print("MAE from Approach 3 (One-Hot Encoding):")
print(score_dataset(OH_X_train, OH_X_valid, y_train, y_valid))
```

```
MAE from Approach 3 (One-Hot Encoding):
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarning:
Feature names only support names that are all strings. Got feature names with dtypes:
['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
17525.345719178084
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:1692: FutureWarning:
Feature names only support names that are all strings. Got feature names with dtypes:
['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
```

Generate test predictions and submit your results

After you complete Step 4, if you'd like to use what you've learned to submit your results to the leaderboard, you'll need to preprocess the test data before generating predictions.

This step is completely optional, and you do not need to submit results to the leaderboard to successfully complete the exercise.

Check out the previous exercise if you need help with remembering how to join the competition or save your results to CSV. Once you have generated a file with your results, follow the instructions below:

1. Begin by clicking on the **Save Version** button in the top right corner of the window. This will generate a pop-up window.
2. Ensure that the **Save and Run All** option is selected, and then click on the **Save** button.
3. This generates a window in the bottom left corner of the notebook. After it has finished running, click on the number to the right of the **Save Version** button. This pulls up a list of versions on the right of the screen. Click on the ellipsis **(...)** to the right of the most recent version, and select **Open in Viewer**. This brings you into view mode of the same page. You will need to scroll down to get back to these instructions.
4. Click on the **Output** tab on the right of the screen. Then, click on the file you would like to submit, and click on the **Submit** button to submit your results to the leaderboard.

You have now successfully submitted to the competition!

If you want to keep working to improve your performance, select the **Edit** button in the top right of the screen. Then you can change your code and repeat the process. There's a lot of room to improve, and you will climb up the leaderboard as you work.

In [24]:
```
# (Optional) Your code here
```

Keep going

With missing value handling and categorical encoding, your modeling process is getting complex. This complexity gets worse when you want to save your model to use in the future. The key to managing this complexity is something called **pipelines**.

Learn to use pipelines to preprocess datasets with categorical variables, missing values and any other messiness your data throws at you.
