```
{"cells":[{"metadata":{},"cell_type":"markdown","source":"**This notebook is an exercise in the [Intermediate Machine Learning](https://www.kaggle.com/learn/intermediate-machine-learning) course.  You can reference the tutorial at [this link](https://www.kaggle.com/alexisbcook/categorical-variables).**\n\n---\n"},{"metadata":{},"cell_type":"markdown","source":"By encoding **categorical variables**, you'll obtain your best results thus far!\n\n# Setup\n\nThe questions below will give you feedback on your work. Run the following cell to set up the feedback system."},{"metadata":{"trusted":false},"cell_type":"code","source":"# Set up code checking\nimport os\nif not os.path.exists(\"../input/train.csv\"):\n    os.symlink(\"../input/home-data-for-ml-course/train.csv\", \"../input/train.csv\")  \n    os.symlink(\"../input/home-data-for-ml-course/test.csv\", \"../input/test.csv\") \nfrom learntools.core import binder\nbinder.bind(globals())\nfrom learntools.ml_intermediate.ex3 import *\nprint(\"Setup Complete\")","execution_count":null,"outputs":[]},{"metadata":{},"cell_type":"markdown","source":"In this exercise, you will work with data from the [Housing Prices Competition for Kaggle Learn Users](https://www.kaggle.com/c/home-data-for-ml-course). \n\n![Ames Housing dataset image](https://i.imgur.com/lTJVG4e.png)\n\nRun 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`."},{"metadata":{"trusted":false},"cell_type":"code","source":"import pandas as pd\nfrom sklearn.model_selection import train_test_split\n\n# Read the data\nX = pd.read_csv('../input/train.csv', index_col='Id') \nX_test = pd.read_csv('../input/test.csv', index_col='Id')\n\n# Remove rows with missing target, separate target from predictors\nX.dropna(axis=0, subset=['SalePrice'], inplace=True)\ny = X.SalePrice\nX.drop(['SalePrice'], axis=1, inplace=True)\n\n# To keep things simple, we'll drop columns with missing values\ncols_with_missing = [col for col in X.columns if X[col].isnull().any()] \nX.drop(cols_with_missing, axis=1, inplace=True)\nX_test.drop(cols_with_missing, axis=1, inplace=True)\n\n# Break off validation set from training data\nX_train, X_valid, y_train, y_valid = train_test_split(X, y,\n                                                      train_size=0.8, test_size=0.2,\n                                                      random_state=0)","execution_count":null,"outputs":[]},{"metadata":{},"cell_type":"markdown","source":"Use the next code cell to print the first five rows of the data."},{"metadata":{"trusted":false},"cell_type":"code","source":"X_train.head()","execution_count":null,"outputs":[]},{"metadata":{},"cell_type":"markdown","source":"Notice that the dataset contains both numerical and categorical variables.  You'll need to encode the categorical data before training a model.\n\nTo compare different models, you'll use the same `score_dataset()` function from the tutorial.  This function reports the [mean absolute error](https://en.wikipedia.org/wiki/Mean_absolute_error) (MAE) from a random forest model."},{"metadata":{"trusted":false},"cell_type":"code","source":"from sklearn.ensemble import RandomForestRegressor\nfrom sklearn.metrics import mean_absolute_error\n\n# function for comparing different approaches\ndef score_dataset(X_train, X_valid, y_train, y_valid):\n    model = RandomForestRegressor(n_estimators=100, random_state=0)\n    model.fit(X_train, y_train)\n    preds = model.predict(X_valid)\n    return mean_absolute_error(y_valid, preds)","execution_count":null,"outputs":[]},{"metadata":{},"cell_type":"markdown","source":"# Step 1: Drop columns with categorical data\n\nYou'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.  "},{"metadata":{"trusted":false},"cell_type":"code","source":"# Fill in the lines below: drop columns in training and validation data\ndrop_X_train = X_train.select_dtypes(exclude=['object'])\ndrop_X_valid = X_valid.select_dtypes(exclude=['object'])\n\n# Check your answers\nstep_1.check()","execution_count":null,"outputs":[]},{"metadata":{"trusted":false},"cell_type":"code","source":"# Lines below will give you a hint or solution code\n#step_1.hint()\n#step_1.solution()","execution_count":null,"outputs":[]},{"metadata":{},"cell_type":"markdown","source":"Run the next code cell to get the MAE for this approach."},{"metadata":{"trusted":false},"cell_type":"code","source":"print(\"MAE from Approach 1 (Drop categorical variables):\")\nprint(score_dataset(drop_X_train, drop_X_valid, y_train, y_valid))","execution_count":null,"outputs":[]},{"metadata":{},"cell_type":"markdown","source":"Before jumping into label 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."},{"metadata":{"trusted":false},"cell_type":"code","source":"print(\"Unique values in 'Condition2' column in training data:\", X_train['Condition2'].unique())\nprint(\"\\nUnique values in 'Condition2' column
```

in validation data:\", X_valid['Condition2'].unique())","execution_count":null,"outputs":[]},{"metadata":
{},"cell_type":"markdown","source":"# Step 2: Label encoding\n\n### Part A\n\nIf you now write code to: \n- fit a label encoder to
the training data, and then \n- use it to transform both the training and validation data, \n\nyou'll get an error. Can you see
why this is the case? (_You'll need to use the above output to answer this question._)"},{"metadata":
{"trusted":false},"cell_type":"code","source":"# Check your answer (Run this code cell to receive
credit!)\nstep_2.a.check()","execution_count":null,"outputs":[]},{"metadata":
{"trusted":false},"cell_type":"code","source":"#step_2.a.hint()","execution_count":null,"outputs":[]},{"metadata":
{},"cell_type":"markdown","source":"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 label encoder to deal with new categories. The simplest
approach, however, is to drop the problematic categorical columns. \n\nRun the code cell below to save the problematic columns to
a Python list `bad_label_cols`. Likewise, columns that can be safely label encoded are stored in `good_label_cols`."},{"metadata":
{"trusted":false},"cell_type":"code","source":"# All categorical columns\nobject_cols = [col for col in X_train.columns if
X_train[col].dtype == \"object\"]\n\n# Columns that can be safely label encoded\ngood_label_cols = [col for col in object_cols if
\n   set(X_train[col]) == set(X_valid[col])]\n   \n# Problematic columns that will be dropped from the
dataset\nbad_label_cols = list(set(object_cols)-set(good_label_cols))\n   \nprint('Categorical columns that will be label
encoded:', good_label_cols)\nprint('\\nCategorical columns that will be dropped from the dataset:',
bad_label_cols)","execution_count":null,"outputs":[]},{"metadata":{},"cell_type":"markdown","source":"### Part B\n\nUse the next
code cell to label encode the data in `X_train` and `X_valid`. Set the preprocessed DataFrames to `label_X_train` and
`label_X_valid`, respectively. \n- We have provided code below to drop the categorical columns in `bad_label_cols` from the
dataset. \n- You should label encode the categorical columns in `good_label_cols`. "},{"metadata":
{"trusted":false},"cell_type":"code","source":"from sklearn.preprocessing import LabelEncoder\n\n# Drop categorical columns that
will not be encoded\nlabel_X_train = X_train.drop(bad_label_cols, axis=1)\nlabel_X_valid = X_valid.drop(bad_label_cols,
axis=1)\n\n# Apply label encoder \nlabel_encoder = LabelEncoder()\nfor col in good_label_cols:\n   label_X_train[col] =
label_encoder.fit_transform(label_X_train[col])\n   label_X_valid[col] = label_encoder.transform(label_X_valid[col])\n   \n#
Check your answer\nstep_2.b.check()","execution_count":null,"outputs":[]},{"metadata":
{"trusted":false},"cell_type":"code","source":"# Lines below will give you a hint or solution
code\n#step_2.b.hint()\n#step_2.b.solution()","execution_count":null,"outputs":[]},{"metadata":
{},"cell_type":"markdown","source":"Run the next code cell to get the MAE for this approach."},{"metadata":
{"trusted":false},"cell_type":"code","source":"print(\"MAE from Approach 2 (Label Encoding):\")
\nprint(score_dataset(label_X_train, label_X_valid, y_train, y_valid))","execution_count":null,"outputs":[]},{"metadata":
{},"cell_type":"markdown","source":"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.\n\nSoon, 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.
"},{"metadata":{"trusted":false},"cell_type":"code","source":"# Get number of unique entries in each column with categorical
data\nobject_nunique = list(map(lambda col: X_train[col].nunique(), object_cols))\nd = dict(zip(object_cols, object_nunique))\n\n#
Print number of unique entries by column, in ascending order\nsorted(d.items(), key=lambda x:
x[1])","execution_count":null,"outputs":[]},{"metadata":{},"cell_type":"markdown","source":"# Step 3: Investigating
cardinality\n\n### Part A\n\nThe 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.\n\nWe 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.\n\nUse the output above to
answer the questions below."},{"metadata":{"trusted":false},"cell_type":"code","source":"# Fill in the line below: How many
categorical variables in the training data\n# have cardinality greater than 10?\nhigh_cardinality_numcols = 3\n\n# Fill in the line
below: How many columns are needed to one-hot encode the \n# 'Neighborhood' variable in the training data?\nnum_cols_neighborhood =
25\n\n# Check your answers\nstep_3.a.check()","execution_count":null,"outputs":[]},{"metadata":

{"trusted":false},"cell_type":"code","source":"# Lines below will give you a hint or solution
code\n#step_3.a.hint()\n#step_3.a.solution()","execution_count":null,"outputs":[]},{"metadata":
{},"cell_type":"markdown","source":"### Part B\n\nFor 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 label encoding.\n\nAs an example, consider a dataset with
10,000 rows, and containing one categorical column with 100 unique entries. \n- If this column is replaced with the corresponding
one-hot encoding, how many entries are added to the dataset? \n- If we instead replace the column with the label encoding, how
many entries are added? \n\nUse your answers to fill in the lines below."},{"metadata":
{"trusted":false},"cell_type":"code","source":"# Fill in the line below: How many entries are added to the dataset by \n#
replacing the column with a one-hot encoding?\nOH_entries_added = 990000\n\n# Fill in the line below: How many entries are added to
the dataset by\n# replacing the column with a label encoding?\nlabel_entries_added = 0\n\n# Check your
answers\nstep_3.b.check()","execution_count":null,"outputs":[]},{"metadata":{"trusted":false},"cell_type":"code","source":"# Lines
below will give you a hint or solution code\n#step_3.b.hint()\n#step_3.b.solution()","execution_count":null,"outputs":[]},
{"metadata":{},"cell_type":"markdown","source":"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.\n\nRun 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."},{"metadata":
{"trusted":false},"cell_type":"code","source":"# Columns that will be one-hot encoded\nlow_cardinality_cols = [col for col in
object_cols if X_train[col].nunique() < 10]\n\n# Columns that will be dropped from the dataset\nhigh_cardinality_cols =
list(set(object_cols)-set(low_cardinality_cols))\n\nprint('Categorical columns that will be one-hot encoded:',
low_cardinality_cols)\nprint('\\nCategorical columns that will be dropped from the dataset:',
high_cardinality_cols)","execution_count":null,"outputs":[]},{"metadata":{},"cell_type":"markdown","source":"# Step 4: One-hot
encoding\n\nUse 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. \n- The full list of categorical columns in the dataset can be found in the Python
list `object_cols`.\n- You should only one-hot encode the categorical columns in `low_cardinality_cols`. All other categorical
columns should be dropped from the dataset. "},{"metadata":{"trusted":false},"cell_type":"code","source":"from
sklearn.preprocessing import OneHotEncoder\n\n# Use as many lines of code as you need!\n\none_hot_encoder =
OneHotEncoder(handle_unknown = 'ignore', sparse = False)\nOH_X_train =
pd.DataFrame(one_hot_encoder.fit_transform(X_train[low_cardinality_cols]))\nOH_X_valid =
pd.DataFrame(one_hot_encoder.transform(X_valid[low_cardinality_cols]))\n\nOH_X_train.index = X_train.index\nOH_X_valid.index =
X_valid.index\n\nX_train_num_cols = X_train.drop(object_cols, axis = 1, inplace = False)\nX_valid_num_cols =
X_valid.drop(object_cols, axis = 1, inplace = False)\n\nOH_X_train = pd.concat([X_train_num_cols, OH_X_train], axis =
1)\nOH_X_valid = pd.concat([X_valid_num_cols, OH_X_valid], axis = 1)\n\n# Check your
answer\nstep_4.check()","execution_count":null,"outputs":[]},{"metadata":{"trusted":false},"cell_type":"code","source":"# Lines
below will give you a hint or solution code\n#step_4.hint()\n#step_4.solution()","execution_count":null,"outputs":[]},{"metadata":
{},"cell_type":"markdown","source":"Run the next code cell to get the MAE for this approach."},{"metadata":
{"trusted":false},"cell_type":"code","source":"print(\"MAE from Approach 3 (One-Hot Encoding):\") \nprint(score_dataset(OH_X_train,
OH_X_valid, y_train, y_valid))","execution_count":null,"outputs":[]},{"metadata":{},"cell_type":"markdown","source":"# Generate
test predictions and submit your results\n\nAfter 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.\n\n**This step is completely
optional, and you do not need to submit results to the leaderboard to successfully complete the exercise.**\n\nCheck out the
previous exercise if you need help with remembering how to [join the competition](https://www.kaggle.com/c/home-data-for-ml-course)
or save your results to CSV. Once you have generated a file with your results, follow the instructions below:\n1. Begin by
clicking on the blue **Save Version** button in the top right corner of the window. This will generate a pop-up window. \n2.
Ensure that the **Save and Run All** option is selected, and then click on the blue **Save** button.\n3. 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.\n4. 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 blue **Submit** button to submit your results to the leaderboard.\n\nYou have now successfully submitted to the competition!\n\nIf you want to keep working to improve your performance, select the blue **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.\n"},{"metadata":{"trusted":false},"cell_type":"code","source":"# (Optional) Your code here","execution_count":null,"outputs":[]},{"metadata":{},"cell_type":"markdown","source":"# Keep going\n\nWith 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**. \n\n**[Learn to use pipelines](https://www.kaggle.com/alexisbcook/pipelines)** to preprocess datasets with categorical variables, missing values and any other messiness your data throws at you."},{"metadata":{},"cell_type":"markdown","source":"---\n\n\n\n\n*Have questions or comments? Visit the [Learn Discussion forum](https://www.kaggle.com/learn-forum/161289) to chat with other Learners.*"}],"metadata":{"kernelspec":{"language":"python","display_name":"Python 3","name":"python3"},"language_info":{"pygments_lexer":"ipython3","nbconvert_exporter":"python","version":"3.6.4","file_extension":".py","codemirror_mode":{"name":"ipython","version":3},"name":"python","mimetype":"text/x-python"}},"nbformat":4,"nbformat_minor":4}