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

Now it's your turn to test your new knowledge of **missing values** handling. You'll probably find it makes a big difference.

Setup

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

In [ ]:
```
# 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.ex2 import *
print("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 [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

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

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

# To keep things simple, we'll use only numerical predictors
X = X_full.select_dtypes(exclude=['object'])
X_test = X_test_full.select_dtypes(exclude=['object'])

# 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 [ ]:
```
X_train.head()
```

You can already see a few missing values in the first several rows. In the next step, you'll obtain a more comprehensive understanding of the missing values in the dataset.

Step 1: Preliminary investigation

Run the code cell below without changes.

In [ ]:
```
# Shape of training data (num_rows, num_columns)
print(X_train.shape)

# Number of missing values in each column of training data
missing_val_count_by_column = (X_train.isnull().sum())
print(missing_val_count_by_column[missing_val_count_by_column > 0])
```

Part A

Use the above output to answer the questions below.

In [ ]:
```
# Fill in the line below: How many rows are in the training data?
num_rows = 1168

# Fill in the line below: How many columns in the training data
# have missing values?
num_cols_with_missing = 3

# Fill in the line below: How many missing entries are contained in
# all of the training data?
tot_missing = 276

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

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

Part B

Considering your answers above, what do you think is likely the best approach to dealing with the missing values?

In [ ]:
```
# Check your answer (Run this code cell to receive credit!)
step_1.b.check()
```

In [ ]:
```
#step_1.b.hint()
```

To compare different approaches to dealing with missing values, 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.

In [ ]:
```
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 2: Drop columns with missing values

In this step, you'll preprocess the data in `X_train` and `X_valid` to remove columns with missing values. Set the preprocessed DataFrames to `reduced_X_train` and `reduced_X_valid`, respectively.

In [ ]:
```
# Fill in the line below: get names of columns with missing values
col_with_missing_values=[col for col in X_train.columns
                        if X_train[col].isnull().any()] # Your code her
e

# Fill in the lines below: drop columns in training and validation data
```

```
reduced_X_train = X_train.drop(col_with_missing_values,axis=1)
reduced_X_valid = X_valid.drop(col_with_missing_values,axis=1)

# Check your answers
step_2.check()
```

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

Run the next code cell without changes to obtain the MAE for this approach.

In [ ]:
```
print("MAE (Drop columns with missing values):")
print(score_dataset(reduced_X_train, reduced_X_valid, y_train, y_valid
))
```

Step 3: Imputation

Part A

Use the next code cell to impute missing values with the mean value along each column. Set the preprocessed DataFrames to `imputed_X_train` and `imputed_X_valid`. Make sure that the column names match those in `X_train` and `X_valid`.

In [ ]:
```
from sklearn.impute import SimpleImputer

# Fill in the lines below: imputation
my_imputer= SimpleImputer(strategy="mean") # Your code here
imputed_X_train = pd.DataFrame(my_imputer.fit_transform(X_train))
imputed_X_valid = pd.DataFrame(my_imputer.transform(X_valid))

# Fill in the lines below: imputation removed column names; put them ba
ck
imputed_X_train.columns = X_train.columns
```

```
imputed_X_valid.columns = X_valid.columns

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

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

Run the next code cell without changes to obtain the MAE for this approach.

In [ ]:
```
print("MAE (Imputation):")
print(score_dataset(imputed_X_train, imputed_X_valid, y_train, y_valid
))
```

Part B

Compare the MAE from each approach. Does anything surprise you about the results? Why do you think one approach performed better than the other?

In [ ]:
```
# Check your answer (Run this code cell to receive credit!)
step_3.b.check()
```

In [ ]:
```
#step_3.b.hint()
```

Step 4: Generate test predictions

In this final step, you'll use any approach of your choosing to deal with missing values. Once you've preprocessed the training and validation features, you'll train and evaluate a random forest model. Then, you'll preprocess the test data before generating predictions that can be submitted to the competition!

Part A

Use the next code cell to preprocess the training and validation data. Set the preprocessed DataFrames to `final_X_train` and `final_X_valid`. **You can use any approach of your choosing here!** in order for this step to be marked as correct, you need only ensure:

- the preprocessed DataFrames have the same number of columns,
- the preprocessed DataFrames have no missing values,
- `final_X_train` and `y_train` have the same number of rows, and
- `final_X_valid` and `y_valid` have the same number of rows.

In [ ]:
```
# Preprocessed training and validation features
final_imputer = SimpleImputer(strategy='median')
final_X_train = pd.DataFrame(final_imputer.fit_transform(X_train))
final_X_valid = pd.DataFrame(final_imputer.transform(X_valid))

# Restoring column names
final_X_train.columns = X_train.columns
final_X_valid.columns = X_valid.columns



# Check your answers
step_4.a.check()
```

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

Run the next code cell to train and evaluate a random forest model. (*Note that we don't use the `score_dataset()` function above, because we will soon use the trained model to generate test predictions!*)

In [ ]:
```
# Define and fit model
model = RandomForestRegressor(n_estimators=100, random_state=0)
model.fit(final_X_train, y_train)
```

```
# Get validation predictions and MAE
preds_valid = model.predict(final_X_valid)
print("MAE (Your approach):")
print(mean_absolute_error(y_valid, preds_valid))
```

Part B

Use the next code cell to preprocess your test data. Make sure that you use a method that agrees with how you preprocessed the training and validation data, and set the preprocessed test features to `final_X_test`.

Then, use the preprocessed test features and the trained model to generate test predictions in `preds_test`.

In order for this step to be marked correct, you need only ensure:

- the preprocessed test DataFrame has no missing values, and
- `final_X_test` has the same number of rows as `X_test`.

In [ ]:
```
# Fill in the line below: preprocess test data
final_X_test = pd.DataFrame(my_imputer.transform(X_test))



# Fill in the line below: get test predictions
preds_test = model.predict(final_X_test)

step_4.b.check()
```

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

Run the next code cell without changes to save your results to a CSV file that can be submitted directly to the competition.

```
In [ ]: # Save test predictions to file
        output = pd.DataFrame({'Id': X_test.index,
                               'SalePrice': preds_test})
        output.to_csv('submission.csv', index=False)
```

Submit your results

Once you have successfully completed Step 4, you're ready to submit your results to the leaderboard! (*You also learned how to do this in the previous exercise. If you need a reminder of how to do this, please use the instructions below.*)

First, you'll need to join the competition if you haven't already. So open a new window by clicking on this link. Then click on the **Join Competition** button.

Next, follow the instructions below:

1. Begin by clicking on the blue **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 blue **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 blue **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 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.

Keep going

Move on to learn what **categorical variables** are, along with how to incorporate them into your machine learning models. Categorical variables are very common in real-world data, but you'll get an error if you try to plug them into your models without processing them first!

Have questions or comments? Visit the Learn Discussion forum to chat with other Learners.