index

February 3, 2022

1 Preprocessing with scikit-learn - Cumulative Lab

github link is:

`https://github.com/miladshiraniUCB/dsc-sklearn-preprocessing-lab.git`

1.1 Introduction

In this cumulative lab, you'll practice applying various preprocessing techniques with scikit-learn (`sklearn`) to the Ames Housing dataset in order to prepare the data for predictive modeling. The main emphasis here is on preprocessing (not EDA or modeling theory), so we will skip over most of the visualization and metrics steps that you would take in an actual modeling process.

1.2 Objectives

You will be able to:

- Practice identifying which preprocessing technique to use
- Practice filtering down to relevant columns
- Practice applying `sklearn.impute` to fill in missing values
- Practice applying `sklearn.preprocessing`:
  - `OrdinalEncoder` for converting binary categories to 0 and 1 within a single column
  - `OneHotEncoder` for creating multiple "dummy" columns to represent multiple categories

1.3 Your Task: Prepare the Ames Housing Dataset for Modeling

Photo by Kyle Kempt on Unsplash

1.3.1 Requirements

1. Drop Irrelevant Columns For the purposes of this lab, we will only be using a subset of all of the features present in the Ames Housing dataset. In this step you will drop all irrelevant columns.

2. Handle Missing Values Often for reasons outside of a data scientist's control, datasets are missing some values. In this step you will assess the presence of NaN values in our subset of data, and use `MissingIndicator` and `SimpleImputer` from the `sklearn.impute` submodule to handle any missing values.

3. Convert Categorical Features into Numbers A built-in assumption of the scikit-learn library is that all data being fed into a machine learning model is already in a numeric format, otherwise you will get a `ValueError` when you try to fit a model. In this step you will use an `OrdinalEncoder` to replace data within individual non-numeric columns with 0s and 1s, and a `OneHotEncoder` to replace columns containing more than 2 categories with multiple "dummy" columns containing 0s and 1s.

At this point, a scikit-learn model should be able to run without errors!

4. Preprocess Test Data Apply Steps 1-3 to the test data in order to perform a final model evaluation.

1.4 Lab Setup

1.4.1 Getting the Data

In the cell below, we import the `pandas` library, open the CSV containing the Ames Housing data as a pandas `DataFrame`, and inspect its contents.

[59]:
```python
# Run this cell without changes
import pandas as pd
df = pd.read_csv("data/ames.csv")
df
```

[59]:
```
        Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0        1          60       RL         65.0     8450   Pave   NaN      Reg
1        2          20       RL         80.0     9600   Pave   NaN      Reg
2        3          60       RL         68.0    11250   Pave   NaN      IR1
3        4          70       RL         60.0     9550   Pave   NaN      IR1
4        5          60       RL         84.0    14260   Pave   NaN      IR1
…     …          …       …          …       …    …   …    …
1455  1456          60       RL         62.0     7917   Pave   NaN      Reg
1456  1457          20       RL         85.0    13175   Pave   NaN      Reg
1457  1458          70       RL         66.0     9042   Pave   NaN      Reg
1458  1459          20       RL         68.0     9717   Pave   NaN      Reg
1459  1460          20       RL         75.0     9937   Pave   NaN      Reg

     LandContour Utilities  … PoolArea PoolQC  Fence MiscFeature MiscVal  \
0            Lvl    AllPub  …        0    NaN    NaN         NaN       0
1            Lvl    AllPub  …        0    NaN    NaN         NaN       0
2            Lvl    AllPub  …        0    NaN    NaN         NaN       0
3            Lvl    AllPub  …        0    NaN    NaN         NaN       0
4            Lvl    AllPub  …        0    NaN    NaN         NaN       0
…          …       … …     …      …    …         …       …
1455         Lvl    AllPub  …        0    NaN    NaN         NaN       0
1456         Lvl    AllPub  …        0    NaN  MnPrv         NaN       0
1457         Lvl    AllPub  …        0    NaN  GdPrv        Shed    2500
1458         Lvl    AllPub  …        0    NaN    NaN         NaN       0
1459         Lvl    AllPub  …        0    NaN    NaN         NaN       0

     MoSold YrSold  SaleType  SaleCondition  SalePrice
0         2   2008        WD         Normal     208500
1         5   2007        WD         Normal     181500
2         9   2008        WD         Normal     223500
3         2   2006        WD        Abnorml     140000
4        12   2008        WD         Normal     250000
…      …    …       …            …       …
1455      8   2007        WD         Normal     175000
1456      2   2010        WD         Normal     210000
1457      5   2010        WD         Normal     266500
```

```
1458       4    2010         WD         Normal      142125
1459       6    2008         WD         Normal      147500

[1460 rows x 81 columns]
```

[60]:
```python
# Run this cell without changes
df.describe()
```

[60]:
```
                Id    MSSubClass  LotFrontage        LotArea  OverallQual  \
count  1460.000000  1460.000000  1201.000000    1460.000000  1460.000000
mean    730.500000    56.897260    70.049958   10516.828082     6.099315
std     421.610009    42.300571    24.284752    9981.264932     1.382997
min       1.000000    20.000000    21.000000    1300.000000     1.000000
25%     365.750000    20.000000    59.000000    7553.500000     5.000000
50%     730.500000    50.000000    69.000000    9478.500000     6.000000
75%    1095.250000    70.000000    80.000000   11601.500000     7.000000
max    1460.000000   190.000000   313.000000  215245.000000    10.000000

       OverallCond    YearBuilt  YearRemodAdd   MasVnrArea   BsmtFinSF1  …  \
count  1460.000000  1460.000000   1460.000000  1452.000000  1460.000000  …
mean      5.575342  1971.267808   1984.865753   103.685262   443.639726  …
std       1.112799    30.202904     20.645407   181.066207   456.098091  …
min       1.000000  1872.000000   1950.000000     0.000000     0.000000  …
25%       5.000000  1954.000000   1967.000000     0.000000     0.000000  …
50%       5.000000  1973.000000   1994.000000     0.000000   383.500000  …
75%       6.000000  2000.000000   2004.000000   166.000000   712.250000  …
max       9.000000  2010.000000   2010.000000  1600.000000  5644.000000  …

        WoodDeckSF  OpenPorchSF  EnclosedPorch    3SsnPorch  ScreenPorch  \
count  1460.000000  1460.000000    1460.000000  1460.000000  1460.000000
mean     94.244521    46.660274      21.954110     3.409589    15.060959
std     125.338794    66.256028      61.119149    29.317331    55.757415
min       0.000000     0.000000       0.000000     0.000000     0.000000
25%       0.000000     0.000000       0.000000     0.000000     0.000000
50%       0.000000    25.000000       0.000000     0.000000     0.000000
75%     168.000000    68.000000       0.000000     0.000000     0.000000
max     857.000000   547.000000     552.000000   508.000000   480.000000

          PoolArea       MiscVal       MoSold       YrSold      SalePrice
count  1460.000000   1460.000000  1460.000000  1460.000000    1460.000000
mean      2.758904     43.489041     6.321918  2007.815753  180921.195890
std      40.177307    496.123024     2.703626     1.328095   79442.502883
min       0.000000      0.000000     1.000000  2006.000000   34900.000000
25%       0.000000      0.000000     5.000000  2007.000000  129975.000000
50%       0.000000      0.000000     6.000000  2008.000000  163000.000000
75%       0.000000      0.000000     8.000000  2009.000000  214000.000000
max     738.000000  15500.000000    12.000000  2010.000000  755000.000000
```

```
[8 rows x 38 columns]
```

The prediction target for this analysis is the sale price of the home, so we separate the data into `X` and `y` accordingly:

[61]:
```
# Run this cell without changes
y = df["SalePrice"]
X = df.drop("SalePrice", axis=1)
```

Next, we separate the data into a train set and a test set prior to performing any preprocessing steps:

[62]:
```
# Run this cell without changes
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

(If you are working through this lab and you just want to start over with the original value for `X_train`, re-run the cell above.)

[63]:
```
# Run this cell without changes
print(f"X_train is a DataFrame with {X_train.shape[0]} rows and {X_train.
 ↪shape[1]} columns")
print(f"y_train is a Series with {y_train.shape[0]} values")

# We always should have the same number of rows in X as values in y
assert X_train.shape[0] == y_train.shape[0]
```

```
X_train is a DataFrame with 1095 rows and 80 columns
y_train is a Series with 1095 values
```

Fitting a Model For this lab we will be using a `LinearRegression` model from scikit-learn (documentation here).

Right now, we have not done any preprocessing, so we expect that trying to fit a model will fail:

[64]:
```
# Run this cell without changes
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-64-cc9cd14c8d9e> in <module>
      3
      4 model = LinearRegression()
----> 5 model.fit(X_train, y_train)
```

```
/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/linear_model/_base.py in fit(self, X, y, sample_weight)
    503
    504         n_jobs_ = self.n_jobs
--> 505         X, y = self._validate_data(X, y, accept_sparse=['csr', 'csc', 'coo'],
    506                                    y_numeric=True, multi_output=True)
    507

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/base.py in _validate_data(self, X, y, reset, validate_separately, **check_params)
    430                 y = check_array(y, **check_y_params)
    431             else:
--> 432                 X, y = check_X_y(X, y, **check_params)
    433             out = X, y
    434

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/utils/validation.py in inner_f(*args, **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/utils/validation.py in check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, ensure_min_features, y_numeric, estimator)
    793         raise ValueError("y cannot be None")
    794
--> 795     X = check_array(X, accept_sparse=accept_sparse,
    796                     accept_large_sparse=accept_large_sparse,
    797                     dtype=dtype, order=order, copy=copy,

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/utils/validation.py in inner_f(*args, **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/utils/validation.py in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator)
```

```
    596                     array = array.astype(dtype, casting="unsafe",␣
 ↪copy=False)
    597                 else:
--> 598                     array = np.asarray(array, order=order, dtype=dtype)
    599             except ComplexWarning:
    600                 raise ValueError("Complex data not supported\n"

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/numpy/core/_asarray.p␣
 ↪in asarray(a, dtype, order)
     83
     84     """
---> 85     return array(a, dtype, copy=False, order=order)
     86
     87

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/pandas/core/generic.p␣
 ↪in __array__(self, dtype)
   1779
   1780     def __array__(self, dtype=None) -> np.ndarray:
-> 1781         return np.asarray(self._values, dtype=dtype)
   1782
   1783     def __array_wrap__(self, result, context=None):

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/numpy/core/_asarray.p␣
 ↪in asarray(a, dtype, order)
     83
     84     """
---> 85     return array(a, dtype, copy=False, order=order)
     86
     87

ValueError: could not convert string to float: 'RL'
```

As you can see, we got `ValueError: could not convert string to float: 'RL'`.

In order to fit a scikit-learn model, all values must be numeric, and the third column of our full dataset (`MSZoning`) contains values like `'RL'` and `'RH'`, which are strings. So this error was expected, but after some preprocessing, this model will work!

1.5 1. Drop Irrelevant Columns

For the purpose of this analysis, we'll only use the following columns, described by `relevant_columns`. You can find the full description of their values in the file `data/data_description.txt` included in this repository.

In the cell below, reassign `X_train` so that it only contains the columns in `relevant_columns`.

Hint: Even though we describe this as "dropping" irrelevant columns, it's easier if you invert the logic, so that we are only keeping relevant columns, rather than using the `.drop()` method. It is possible to use the `.drop()` method if you really want to, but first you would need to create a list of the column names that you don't want to keep.

[65]:
```python
# Replace None with appropriate code

# Declare relevant columns
relevant_columns = [
    'LotFrontage',  # Linear feet of street connected to property
    'LotArea',      # Lot size in square feet
    'Street',       # Type of road access to property
    'OverallQual',  # Rates the overall material and finish of the house
    'OverallCond',  # Rates the overall condition of the house
    'YearBuilt',    # Original construction date
    'YearRemodAdd', # Remodel date (same as construction date if no remodeling␣
 ↪or additions)
    'GrLivArea',    # Above grade (ground) living area square feet
    'FullBath',     # Full bathrooms above grade
    'BedroomAbvGr', # Bedrooms above grade (does NOT include basement bedrooms)
    'TotRmsAbvGrd', # Total rooms above grade (does not include bathrooms)
    'Fireplaces',   # Number of fireplaces
    'FireplaceQu',  # Fireplace quality
    'MoSold',       # Month Sold (MM)
    'YrSold'        # Year Sold (YYYY)
]

# Reassign X_train so that it only contains relevant columns
X_train = X_train[relevant_columns]

# Visually inspect X_train
X_train
```

[65]:
```
      LotFrontage  LotArea Street  OverallQual  OverallCond  YearBuilt  \
1023         43.0     3182   Pave            7            5       2005
810          78.0    10140   Pave            6            6       1974
1384         60.0     9060   Pave            6            5       1939
626           NaN    12342   Pave            5            5       1960
813          75.0     9750   Pave            6            6       1958
…            …        …      …              …            …          …
1095         78.0     9317   Pave            6            5       2006
1130         65.0     7804   Pave            4            3       1928
1294         60.0     8172   Pave            5            7       1955
860          55.0     7642   Pave            7            8       1918
1126         53.0     3684   Pave            7            5       2007

      YearRemodAdd  GrLivArea  FullBath  BedroomAbvGr  TotRmsAbvGrd  \
1023          2006       1504         2             2             7
810           1999       1309         1             3             5
```

```
1384          1950        1258          1             2             6
626           1978        1422          1             3             6
813           1958        1442          1             4             7
…            …         …        …            …            …
1095          2006        1314          2             3             6
1130          1950        1981          2             4             7
1294          1990         864          1             2             5
860           1998        1426          1             3             7
1126          2007        1555          2             2             7

      Fireplaces FireplaceQu  MoSold  YrSold
1023           1          Gd       5    2008
810            1          Fa       1    2006
1384           0         NaN      10    2009
626            1          TA       8    2007
813            0         NaN       4    2007
…           …          …      …      …
1095           1          Gd       3    2007
1130           2          TA      12    2009
1294           0         NaN       4    2006
860            1          Gd       6    2007
1126           1          TA       6    2009

[1095 rows x 15 columns]
```

Check that the new shape is correct:

[66]:
```python
# Run this cell without changes

# X_train should have the same number of rows as before
assert X_train.shape[0] == 1095

# Now X_train should only have as many columns as relevant_columns
assert X_train.shape[1] == len(relevant_columns)
```

1.6 2. Handle Missing Values

In the cell below, we check to see if there are any NaNs in the selected subset of data:

[67]:
```python
# Run this cell without changes
X_train.isna().sum()
```

[67]:
```
LotFrontage    200
LotArea          0
Street           0
OverallQual      0
OverallCond      0
```

```
YearBuilt         0
YearRemodAdd      0
GrLivArea         0
FullBath          0
BedroomAbvGr      0
TotRmsAbvGrd      0
Fireplaces        0
FireplaceQu     512
MoSold            0
YrSold            0
dtype: int64
```

Ok, it looks like we have some NaNs in `LotFrontage` and `FireplaceQu`.

Before we proceed to fill in those values, we need to ask: **do these NaNs actually represent *missing* values, or is there some real value/category being represented by NaN?**

1.6.1 Fireplace Quality

To start with, let's look at `FireplaceQu`, which means "Fireplace Quality". Why might we have NaN fireplace quality?

Well, some properties don't have fireplaces!

Let's confirm this guess with a little more analysis.

First, we know that there are 512 records with NaN fireplace quality. How many records are there with zero fireplaces?

[68]:
```python
## For Myself
X_train["Fireplaces"].value_counts()
```

[68]:
```
0    512
1    491
2     89
3      3
Name: Fireplaces, dtype: int64
```

[69]:
```python
# Run this cell without changes
X_train[X_train["Fireplaces"] == 0]
```

[69]:
```
      LotFrontage  LotArea Street  OverallQual  OverallCond  YearBuilt  \
1384         60.0     9060   Pave            6            5       1939
813          75.0     9750   Pave            6            6       1958
839          70.0    11767   Pave            5            6       1946
430          21.0     1680   Pave            6            5       1971
513          71.0     9187   Pave            6            5       1983
…               …    …    …            …            …         …
87           40.0     3951   Pave            6            5       2009
330           NaN    10624   Pave            5            4       1964
```

```
1238          63.0    13072   Pave            6            5       2005
121           50.0     6060   Pave            4            5       1939
1294          60.0     8172   Pave            5            7       1955

      YearRemodAdd  GrLivArea  FullBath  BedroomAbvGr  TotRmsAbvGrd  \
1384          1950       1258         1             2             6
813           1958       1442         1             4             7
839           1995       1200         1             3             6
430           1971        987         1             2             4
513           1983       1080         1             3             5
…              …         …        …            …            …
87            2009       1224         2             2             4
330           1964       1728         2             6            10
1238          2005       1141         1             3             6
121           1950       1123         1             3             4
1294          1990        864         1             2             5

      Fireplaces FireplaceQu  MoSold  YrSold
1384           0         NaN      10    2009
813            0         NaN       4    2007
839            0         NaN       5    2008
430            0         NaN       7    2008
513            0         NaN       6    2007
…            …          …      …      …
87             0         NaN       6    2009
330            0         NaN      11    2007
1238           0         NaN       3    2006
121            0         NaN       6    2007
1294           0         NaN       4    2006

[512 rows x 15 columns]
```

Ok, that's 512 rows, same as the number of NaN `FireplaceQu` records. To double-check, let's query for that combination of factors (zero fireplaces and `FireplaceQu` is NaN):

[70]:
```
# Run this cell without changes
X_train[
    (X_train["Fireplaces"] == 0) &
    (X_train["FireplaceQu"].isna())
]
```

[70]:
```
      LotFrontage  LotArea Street  OverallQual  OverallCond  YearBuilt  \
1384         60.0     9060   Pave            6            5       1939
813          75.0     9750   Pave            6            6       1958
839          70.0    11767   Pave            5            6       1946
430          21.0     1680   Pave            6            5       1971
513          71.0     9187   Pave            6            5       1983
```

```
…           …     …    …            …           …           …
87          40.0     3951   Pave            6            5        2009
330          NaN    10624   Pave            5            4        1964
1238        63.0    13072   Pave            6            5        2005
121         50.0     6060   Pave            4            5        1939
1294        60.0     8172   Pave            5            7        1955

      YearRemodAdd  GrLivArea  FullBath  BedroomAbvGr  TotRmsAbvGrd  \
1384          1950       1258         1             2             6
813           1958       1442         1             4             7
839           1995       1200         1             3             6
430           1971        987         1             2             4
513           1983       1080         1             3             5
…              …        …        …             …             …
87            2009       1224         2             2             4
330           1964       1728         2             6            10
1238          2005       1141         1             3             6
121           1950       1123         1             3             4
1294          1990        864         1             2             5

      Fireplaces FireplaceQu  MoSold  YrSold
1384           0         NaN      10    2009
813            0         NaN       4    2007
839            0         NaN       5    2008
430            0         NaN       7    2008
513            0         NaN       6    2007
…            …          …      …      …
87             0         NaN       6    2009
330            0         NaN      11    2007
1238           0         NaN       3    2006
121            0         NaN       6    2007
1294           0         NaN       4    2006

[512 rows x 15 columns]
```

Looks good, still 512 records. So, NaN fireplace quality is not actually information that is missing from our dataset, it is a genuine category which means "fireplace quality is not applicable". This interpretation aligns with what we see in `data/data_description.txt`:

```
...
FireplaceQu: Fireplace quality

       Ex   Excellent - Exceptional Masonry Fireplace
       Gd   Good - Masonry Fireplace in main level
       TA   Average - Prefabricated Fireplace in main living area or Masonry Fireplace in basem
       Fa   Fair - Prefabricated Fireplace in basement
       Po   Poor - Ben Franklin Stove
       NA   No Fireplace
```

...

So, let's replace those NaNs with the string "N/A" to indicate that this is a real category, not missing data:

[71]:
```python
# Run this cell without changes
X_train["FireplaceQu"] = X_train["FireplaceQu"].fillna("N/A")
X_train["FireplaceQu"].value_counts()
```

```
<ipython-input-71-7d9f18d4b622>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train["FireplaceQu"] = X_train["FireplaceQu"].fillna("N/A")
```

[71]:
```
N/A    512
Gd     286
TA     236
Fa      26
Ex      19
Po      16
Name: FireplaceQu, dtype: int64
```

Eventually we will still need to perform some preprocessing to prepare the `FireplaceQu` column for modeling (because models require numeric inputs rather than inputs of type `object`), but we don't need to worry about filling in missing values.

1.6.2 Lot Frontage

Now let's look at `LotFrontage` — it's possible that NaN is also a genuine category here, and it's possible that it's just missing data instead. Let's apply some domain understanding to understand whether it's possible that lot frontage can be N/A just like fireplace quality can be N/A.

Lot frontage is defined as the "Linear feet of street connected to property", i.e. how much of the property runs directly along a road. The amount of frontage required for a property depends on its zoning. Let's look at the zoning of all records with NaN for `LotFrontage`:

[72]:
```python
# Run this cell without changes
df[df["LotFrontage"].isna()]["MSZoning"].value_counts()
```

[72]:
```
RL    229
RM     19
FV      8
RH      3
Name: MSZoning, dtype: int64
```

So, we have RL (residential low density), RM (residential medium density), FV (floating village residential), and RH (residential high density). Looking at the building codes from the City of

Ames, it appears that all of these zones require at least 24 feet of frontage.

Nevertheless, we can't assume that all properties have frontage just because the zoning regulations require it. Maybe these properties predate the regulations, or they received some kind of variance permitting them to get around the requirement. **It's still not as clear here as it was with the fireplaces whether this is a genuine "not applicable" kind of NaN or a "missing information" kind of NaN.**

In a case like this, we can take a double approach:

1. Make a new column in the dataset that simply represents whether `LotFrontage` was originally NaN
2. Fill in the NaN values of `LotFrontage` with median frontage in preparation for modeling

1.6.3 Missing Indicator for `LotFrontage`

First, we import `sklearn.impute.MissingIndicator` (documentation here). The goal of using a `MissingIndicator` is creating a new column to represent which values were NaN (or some other "missing" value) in the original dataset, in case NaN ends up being a meaningful indicator rather than a random missing bit of data.

A `MissingIndicator` is a scikit-learn transformer, meaning that we will use the standard steps for any scikit-learn transformer:

1. Identify data to be transformed (typically not every column is passed to every transformer)
2. Instantiate the transformer object
3. Fit the transformer object (on training data only)
4. Transform data using the transformer object
5. Add the transformed data to the other data that was not transformed

[73]:
```
# Replace None with appropriate code
from sklearn.impute import MissingIndicator

# (1) Identify data to be transformed
# We only want missing indicators for LotFrontage
frontage_train = X_train[["LotFrontage"]]

# (2) Instantiate the transformer object
missing_indicator = MissingIndicator()

# (3) Fit the transformer object on frontage_train
missing_indicator.fit(frontage_train)

# (4) Transform frontage_train and assign the result
# to frontage_missing_train
frontage_missing_train = missing_indicator.transform(frontage_train)

# Visually inspect frontage_missing_train
frontage_missing_train
```

[73]:
```
array([[False],
       [False],
       [False],
       …,
       [False],
       [False],
       [False]])
```

The result of transforming `frontage_train` should be an array of arrays, each containing `True` or `False`. Make sure the `asserts` pass before moving on to the next step.

[74]:
```python
# Run this cell without changes
import numpy as np

# frontage_missing_train should be a NumPy array
assert type(frontage_missing_train) == np.ndarray

# We should have the same number of rows as the full X_train
assert frontage_missing_train.shape[0] == X_train.shape[0]

# But we should only have 1 column
assert frontage_missing_train.shape[1] == 1
```

Now let's add this new information as a new column of `X_train`:

[75]:
```python
# Run this cell without changes

# (5) add the transformed data to the other data
X_train["LotFrontage_Missing"] = frontage_missing_train
X_train
```

```
<ipython-input-75-1e115aae470c>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train["LotFrontage_Missing"] = frontage_missing_train
```

[75]:
```
      LotFrontage  LotArea Street  OverallQual  OverallCond  YearBuilt  \
1023         43.0     3182   Pave            7            5       2005
810          78.0    10140   Pave            6            6       1974
1384         60.0     9060   Pave            6            5       1939
626           NaN    12342   Pave            5            5       1960
813          75.0     9750   Pave            6            6       1958
…              …        …    …              …            …          …
1095         78.0     9317   Pave            6            5       2006
1130         65.0     7804   Pave            4            3       1928
```

```
1294          60.0     8172   Pave             5            7        1955
860           55.0     7642   Pave             7            8        1918
1126          53.0     3684   Pave             7            5        2007

      YearRemodAdd  GrLivArea  FullBath  BedroomAbvGr  TotRmsAbvGrd  \
1023          2006       1504         2             2             7
810           1999       1309         1             3             5
1384          1950       1258         1             2             6
626           1978       1422         1             3             6
813           1958       1442         1             4             7
…             …          …        …           …            …
1095          2006       1314         2             3             6
1130          1950       1981         2             4             7
1294          1990        864         1             2             5
860           1998       1426         1             3             7
1126          2007       1555         2             2             7

      Fireplaces FireplaceQu  MoSold  YrSold  LotFrontage_Missing
1023           1          Gd       5    2008                False
810            1          Fa       1    2006                False
1384           0         N/A      10    2009                False
626            1          TA       8    2007                 True
813            0         N/A       4    2007                False
…          …          …     …     …                   …
1095           1          Gd       3    2007                False
1130           2          TA      12    2009                False
1294           0         N/A       4    2006                False
860            1          Gd       6    2007                False
1126           1          TA       6    2009                False

[1095 rows x 16 columns]
```

[76]:
```python
# Run this cell without changes

# Now we should have 1 extra column compared to
# our original subset
assert X_train.shape[1] == len(relevant_columns) + 1
```

1.6.4 Imputing Missing Values for LotFrontage

Now that we have noted where missing values were originally present, let's use a `SimpleImputer` (documentation here) to fill in those NaNs in the `LotFrontage` column.

The process is very similar to the `MissingIndicator` process, except that we want to replace the original `LotFrontage` column with the transformed version instead of just adding a new column on.

In the cell below, create and use a `SimpleImputer` with `strategy="median"` to transform the value of frontage_train (declared above).

[77]:
```python
# Replace None with appropriate code

from sklearn.impute import SimpleImputer

# (1) frontage_train was created previously, so we don't
# need to extract the relevant data again

# (2) Instantiate a SimpleImputer with strategy="median"
imputer = SimpleImputer(strategy="median")

# (3) Fit the imputer on frontage_train
imputer.fit(frontage_train)

# (4) Transform frontage_train using the imputer and
# assign the result to frontage_imputed_train
frontage_imputed_train = imputer.transform(frontage_train)

# Visually inspect frontage_imputed_train
frontage_imputed_train
```

[77]:
```
array([[43.],
       [78.],
       [60.],
       …,
       [60.],
       [55.],
       [53.]])
```

Now we can replace the original value of LotFrontage in X_train with the new value:

[78]:
```python
# Run this cell without changes

# (5) Replace value of LotFrontage
X_train["LotFrontage"] = frontage_imputed_train

# Visually inspect X_train
X_train
```

```
<ipython-input-78-a75161a2f420>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train["LotFrontage"] = frontage_imputed_train
```

[78]:
```
      LotFrontage  LotArea Street  OverallQual  OverallCond  YearBuilt  \
1023         43.0     3182   Pave            7            5       2005
810          78.0    10140   Pave            6            6       1974
1384         60.0     9060   Pave            6            5       1939
626          70.0    12342   Pave            5            5       1960
813          75.0     9750   Pave            6            6       1958
…             …       …    …          …            …          …
1095         78.0     9317   Pave            6            5       2006
1130         65.0     7804   Pave            4            3       1928
1294         60.0     8172   Pave            5            7       1955
860          55.0     7642   Pave            7            8       1918
1126         53.0     3684   Pave            7            5       2007

      YearRemodAdd  GrLivArea  FullBath  BedroomAbvGr  TotRmsAbvGrd  \
1023          2006       1504         2             2             7
810           1999       1309         1             3             5
1384          1950       1258         1             2             6
626           1978       1422         1             3             6
813           1958       1442         1             4             7
…              …         …        …            …            …
1095          2006       1314         2             3             6
1130          1950       1981         2             4             7
1294          1990        864         1             2             5
860           1998       1426         1             3             7
1126          2007       1555         2             2             7

      Fireplaces FireplaceQu  MoSold  YrSold  LotFrontage_Missing
1023           1          Gd       5    2008                False
810            1          Fa       1    2006                False
1384           0         N/A      10    2009                False
626            1          TA       8    2007                 True
813            0         N/A       4    2007                False
…          …          …     …     …                  …
1095           1          Gd       3    2007                False
1130           2          TA      12    2009                False
1294           0         N/A       4    2006                False
860            1          Gd       6    2007                False
1126           1          TA       6    2009                False

[1095 rows x 16 columns]
```

Now the shape of `X_train` should still be the same as before:

[79]:
```python
# Run this cell without changes
assert X_train.shape == (1095, 16)
```

And now our `X_train` no longer contains any NaN values:

[80]:
```
# Run this cell without changes
X_train.isna().sum()
```

[80]:
```
LotFrontage          0
LotArea              0
Street               0
OverallQual          0
OverallCond          0
YearBuilt            0
YearRemodAdd         0
GrLivArea            0
FullBath             0
BedroomAbvGr         0
TotRmsAbvGrd         0
Fireplaces           0
FireplaceQu          0
MoSold               0
YrSold               0
LotFrontage_Missing  0
dtype: int64
```

Great! Now we have completed Step 2.

1.7 3. Convert Categorical Features into Numbers

Despite dropping irrelevant columns and filling in those NaN values, if we feed the current `X_train` into our scikit-learn `LinearRegression` model, it will crash:

[81]:
```
# Run this cell without changes
model.fit(X_train, y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-81-585a53321d2e> in <module>
      1 # Run this cell without changes
----> 2 model.fit(X_train, y_train)

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/linear_model/
 ↪_base.py in fit(self, X, y, sample_weight)
    503
    504         n_jobs_ = self.n_jobs
--> 505         X, y = self._validate_data(X, y, accept_sparse=['csr', 'csc',␣
 ↪'coo'],
    506                                    y_numeric=True, multi_output=True)
    507

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/base.py in␣
 ↪_validate_data(self, X, y, reset, validate_separately, **check_params)
```

```
    430                 y = check_array(y, **check_y_params)
    431             else:
--> 432                 X, y = check_X_y(X, y, **check_params)
    433             out = X, y
    434

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/utils/
 ↪validation.py in inner_f(*args, **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/utils/
 ↪validation.py in check_X_y(X, y, accept_sparse, accept_large_sparse, dtype,␣
 ↪order, copy, force_all_finite, ensure_2d, allow_nd, multi_output,␣
 ↪ensure_min_samples, ensure_min_features, y_numeric, estimator)
    793         raise ValueError("y cannot be None")
    794
--> 795     X = check_array(X, accept_sparse=accept_sparse,

    796                     accept_large_sparse=accept_large_sparse,
    797                     dtype=dtype, order=order, copy=copy,

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/utils/
 ↪validation.py in inner_f(*args, **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklearn/utils/
 ↪validation.py in check_array(array, accept_sparse, accept_large_sparse, dtype,␣
 ↪order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples,␣
 ↪ensure_min_features, estimator)
    596                     array = array.astype(dtype, casting="unsafe",␣
 ↪copy=False)
    597                 else:
--> 598                     array = np.asarray(array, order=order, dtype=dtype)
    599             except ComplexWarning:
    600                 raise ValueError("Complex data not supported\n"

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/numpy/core/_asarray.py␣
 ↪in asarray(a, dtype, order)
     83
     84     """
---> 85     return array(a, dtype, copy=False, order=order)
```

```
     86
     87

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/pandas/core/generic.p
 ↪in __array__(self, dtype)
   1779
   1780     def __array__(self, dtype=None) -> np.ndarray:
-> 1781         return np.asarray(self._values, dtype=dtype)
   1782
   1783     def __array_wrap__(self, result, context=None):

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/numpy/core/_asarray.p
 ↪in asarray(a, dtype, order)
     83
     84     """
---> 85     return array(a, dtype, copy=False, order=order)
     86
     87

ValueError: could not convert string to float: 'Pave'
```

Now the first column to cause a problem is `Street`, which is documented like this:

```
...
Street: Type of road access to property

       Grvl Gravel
       Pave Paved
...
```

Let's look at the full `X_train`:

[82]:
```
# Run this cell without changes
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1095 entries, 1023 to 1126
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   LotFrontage     1095 non-null   float64
 1   LotArea         1095 non-null   int64
 2   Street          1095 non-null   object
 3   OverallQual     1095 non-null   int64
 4   OverallCond     1095 non-null   int64
 5   YearBuilt       1095 non-null   int64
 6   YearRemodAdd    1095 non-null   int64
 7   GrLivArea       1095 non-null   int64
```

```
 8   FullBath              1095 non-null   int64
 9   BedroomAbvGr          1095 non-null   int64
 10  TotRmsAbvGrd          1095 non-null   int64
 11  Fireplaces            1095 non-null   int64
 12  FireplaceQu           1095 non-null   object
 13  MoSold                1095 non-null   int64
 14  YrSold                1095 non-null   int64
 15  LotFrontage_Missing   1095 non-null   bool
dtypes: bool(1), float64(1), int64(12), object(2)
memory usage: 137.9+ KB
```

So, our model is crashing because some of the columns are non-numeric.

Anything that is already `float64` or `int64` will work with our model, but these features need to be converted:

- `Street` (currently type `object`)
- `FireplaceQu` (currently type `object`)
- `LotFrontage_Missing` (currently type `bool`)

There are two main approaches to converting these values, depending on whether there are 2 values (meaning the categorical variable can be converted into a single binary number) or more than 2 values (meaning we need to create extra columns to represent all categories).

(If there is only 1 value, this is not a useful feature for the purposes of predictive analysis because every single row contains the same information.)

In the cell below, we inspect the value counts of the specified features:

[83]:
```python
# Run this cell without changes

print(X_train["Street"].value_counts())
print()
print(X_train["FireplaceQu"].value_counts())
print()
print(X_train["LotFrontage_Missing"].value_counts())
```

```
Pave    1091
Grvl       4
Name: Street, dtype: int64

N/A    512
Gd     286
TA     236
Fa      26
Ex      19
Po      16
Name: FireplaceQu, dtype: int64

False    895
```

```
True      200
Name: LotFrontage_Missing, dtype: int64
```

So, it looks like `Street` and `LotFrontage_Missing` have only 2 categories and can be converted into binary in place, whereas `FireplaceQu` has 6 categories and will need to be expanded into multiple columns.

1.7.1 Binary Categories

For binary categories, we will use an `OrdinalEncoder` (documentation here) to convert the categories of `Street` and `LotFrontage_Missing` into binary values (0s and 1s).

Just like in Step 2 when we used the `MissingIndicator` and `SimpleImputer`, we will follow these steps:

1. Identify data to be transformed
2. Instantiate the transformer object
3. Fit the transformer object (on training data only)
4. Transform data using the transformer object
5. Add the transformed data to the other data that was not transformed

Let's start with transforming `Street`:

[84]:
```python
# Replace None with appropriate code

# (0) import OrdinalEncoder from sklearn.preprocessing
from sklearn.preprocessing import OrdinalEncoder

# (1) Create a variable street_train that contains the
# relevant column from X_train
# (Use double brackets [[]] to get the appropriate shape)
street_train = X_train[["Street"]]

# (2) Instantiate an OrdinalEncoder
encoder_street = OrdinalEncoder()

# (3) Fit the encoder on street_train
encoder_street.fit(street_train)

# Inspect the categories of the fitted encoder
encoder_street.categories_[0]
```

[84]:
```
array(['Grvl', 'Pave'], dtype=object)
```

The `.categories_` attribute of `OrdinalEncoder` is only present once the `.fit` method has been called. (The trailing `_` indicates this convention.)

What this tells us is that when `encoder_street` is used to transform the street data into 1s and 0s, `0` will mean `'Grvl'` (gravel) in the original data, and `1` will mean `'Pave'` (paved) in the original data.

The eventual scikit-learn model only cares about the 1s and 0s, but this information can be useful for us to understand what our code is doing and help us debug when things go wrong.

Now let's transform `street_train` with the fitted encoder:

[85]:
```
# Replace None with appropriate code

# (4) Transform street_train using the encoder and
# assign the result to street_encoded_train
street_encoded_train = encoder_street.transform(street_train)

# Flatten for appropriate shape
street_encoded_train = street_encoded_train.flatten()

# Visually inspect street_encoded_train
street_encoded_train
```

[85]:
```
array([1., 1., 1., …, 1., 1., 1.])
```

All of the values we see appear to be `1` right now, but that makes sense since there were only 4 properties with gravel (`0`) streets in `X_train`.

Now let's replace the original `Street` column with the encoded version:

[86]:
```
# Replace None with appropriate code

# (5) Replace value of Street
X_train["Street"] = street_encoded_train

# Visually inspect X_train
X_train
```

```
<ipython-input-86-95ad13a2aad2>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train["Street"] = street_encoded_train
```

[86]:
```
      LotFrontage  LotArea  Street  OverallQual  OverallCond  YearBuilt  \
1023         43.0     3182     1.0            7            5       2005
810          78.0    10140     1.0            6            6       1974
1384         60.0     9060     1.0            6            5       1939
626          70.0    12342     1.0            5            5       1960
813          75.0     9750     1.0            6            6       1958
…             …       …        …            …            …          …
1095         78.0     9317     1.0            6            5       2006
1130         65.0     7804     1.0            4            3       1928
```

```
1294         60.0     8172     1.0              5             7          1955
860          55.0     7642     1.0              7             8          1918
1126         53.0     3684     1.0              7             5          2007

      YearRemodAdd  GrLivArea  FullBath  BedroomAbvGr  TotRmsAbvGrd  \
1023          2006       1504         2             2             7
810           1999       1309         1             3             5
1384          1950       1258         1             2             6
626           1978       1422         1             3             6
813           1958       1442         1             4             7
…             …          …        …           …            …
1095          2006       1314         2             3             6
1130          1950       1981         2             4             7
1294          1990        864         1             2             5
860           1998       1426         1             3             7
1126          2007       1555         2             2             7

      Fireplaces FireplaceQu  MoSold  YrSold  LotFrontage_Missing
1023           1          Gd       5    2008                False
810            1          Fa       1    2006                False
1384           0         N/A      10    2009                False
626            1          TA       8    2007                 True
813            0         N/A       4    2007                False
…          …          …       …     …                   …
1095           1          Gd       3    2007                False
1130           2          TA      12    2009                False
1294           0         N/A       4    2006                False
860            1          Gd       6    2007                False
1126           1          TA       6    2009                False

[1095 rows x 16 columns]
```

[87]:

```python
# Run this cell without changes
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1095 entries, 1023 to 1126
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   LotFrontage          1095 non-null   float64
 1   LotArea              1095 non-null   int64
 2   Street               1095 non-null   float64
 3   OverallQual          1095 non-null   int64
 4   OverallCond          1095 non-null   int64
 5   YearBuilt            1095 non-null   int64
 6   YearRemodAdd         1095 non-null   int64
```

```
 7   GrLivArea             1095 non-null   int64
 8   FullBath              1095 non-null   int64
 9   BedroomAbvGr          1095 non-null   int64
 10  TotRmsAbvGrd          1095 non-null   int64
 11  Fireplaces            1095 non-null   int64
 12  FireplaceQu           1095 non-null   object
 13  MoSold                1095 non-null   int64
 14  YrSold                1095 non-null   int64
 15  LotFrontage_Missing   1095 non-null   bool
dtypes: bool(1), float64(2), int64(12), object(1)
memory usage: 137.9+ KB
```

Perfect! Now `Street` should by type `int64` instead of `object`.

Now, repeat the same process with `LotFrontage_Missing`:

[88]:
```
# Replace None with appropriate code

# (1) We already have a variable frontage_missing_train
# from earlier, no additional step needed

# (2) Instantiate an OrdinalEncoder for missing frontage
encoder_frontage_missing = OrdinalEncoder()

# (3) Fit the encoder on frontage_missing_train
encoder_frontage_missing.fit(frontage_missing_train)

# Inspect the categories of the fitted encoder
encoder_frontage_missing.categories_[0]
```

[88]:
```
array([False,  True])
```

[89]:
```
# Replace None with appropriate code

# (4) Transform frontage_missing_train using the encoder and
# assign the result to frontage_missing_encoded_train
frontage_missing_encoded_train = encoder_frontage_missing.
 ↪transform(frontage_missing_train)

# Flatten for appropriate shape
frontage_missing_encoded_train = frontage_missing_encoded_train.flatten()

# Visually inspect frontage_missing_encoded_train
frontage_missing_encoded_train
```

[89]:
```
array([0., 0., 0., …, 0., 0., 0.])
```

[90]:
```python
# Replace None with appropriate code

# (5) Replace value of LotFrontage_Missing
X_train["LotFrontage_Missing"] = frontage_missing_encoded_train

# Visually inspect X_train
X_train
```

```
<ipython-input-90-470619138265>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train["LotFrontage_Missing"] = frontage_missing_encoded_train
```

[90]:

	LotFrontage	LotArea	Street	OverallQual	OverallCond	YearBuilt	\
1023	43.0	3182	1.0	7	5	2005	
810	78.0	10140	1.0	6	6	1974	
1384	60.0	9060	1.0	6	5	1939	
626	70.0	12342	1.0	5	5	1960	
813	75.0	9750	1.0	6	6	1958	
…	…	…	…	…	…	…	
1095	78.0	9317	1.0	6	5	2006	
1130	65.0	7804	1.0	4	3	1928	
1294	60.0	8172	1.0	5	7	1955	
860	55.0	7642	1.0	7	8	1918	
1126	53.0	3684	1.0	7	5	2007	

	YearRemodAdd	GrLivArea	FullBath	BedroomAbvGr	TotRmsAbvGrd	\
1023	2006	1504	2	2	7	
810	1999	1309	1	3	5	
1384	1950	1258	1	2	6	
626	1978	1422	1	3	6	
813	1958	1442	1	4	7	
…	…	…	…	…	…	
1095	2006	1314	2	3	6	
1130	1950	1981	2	4	7	
1294	1990	864	1	2	5	
860	1998	1426	1	3	7	
1126	2007	1555	2	2	7	

	Fireplaces	FireplaceQu	MoSold	YrSold	LotFrontage_Missing
1023	1	Gd	5	2008	0.0
810	1	Fa	1	2006	0.0
1384	0	N/A	10	2009	0.0
626	1	TA	8	2007	1.0

```
813             0          N/A       4     2007                     0.0
…          …          …      …      …                   …
1095            1           Gd       3     2007                     0.0
1130            2           TA      12     2009                     0.0
1294            0          N/A       4     2006                     0.0
860             1           Gd       6     2007                     0.0
1126            1           TA       6     2009                     0.0

[1095 rows x 16 columns]
```

[91]:
```python
# Run this cell without changes
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1095 entries, 1023 to 1126
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   LotFrontage          1095 non-null   float64
 1   LotArea              1095 non-null   int64
 2   Street               1095 non-null   float64
 3   OverallQual          1095 non-null   int64
 4   OverallCond          1095 non-null   int64
 5   YearBuilt            1095 non-null   int64
 6   YearRemodAdd         1095 non-null   int64
 7   GrLivArea            1095 non-null   int64
 8   FullBath             1095 non-null   int64
 9   BedroomAbvGr         1095 non-null   int64
 10  TotRmsAbvGrd         1095 non-null   int64
 11  Fireplaces           1095 non-null   int64
 12  FireplaceQu          1095 non-null   object
 13  MoSold               1095 non-null   int64
 14  YrSold               1095 non-null   int64
 15  LotFrontage_Missing  1095 non-null   float64
dtypes: float64(3), int64(12), object(1)
memory usage: 145.4+ KB
```

Great, now we only have 1 column remaining that isn't type `float64` or `int64`!

Note on Preprocessing Boolean Values For binary values like `LotFrontage_Missing`, you might see a few different approaches to preprocessing. Python treats `True` and `1` as equal:

[92]:
```python
# Run this cell without changes
print(True == 1)
print(False == 0)
```

```
True
True
```

This means that if your model is purely using Python, you actually might just be able to leave columns as type `bool` without any issues. You will likely see examples that do this. However if your model relies on C or Java "under the hood", this might cause problems.

There is also a technique using `pandas` rather than scikit-learn for this particular conversion of boolean values to integers:

[93]:
```
# Run this cell without changes
df_example = pd.DataFrame(frontage_missing_train,␣
↪columns=["LotFrontage_Missing"])
df_example
```

[93]:
```
      LotFrontage_Missing
0                   False
1                   False
2                   False
3                    True
4                   False
…                       …
1090                False
1091                False
1092                False
1093                False
1094                False

[1095 rows x 1 columns]
```

[94]:
```
# Run this cell without changes
df_example["LotFrontage_Missing"] = df_example["LotFrontage_Missing"].
↪astype(int)
df_example
```

[94]:
```
      LotFrontage_Missing
0                       0
1                       0
2                       0
3                       1
4                       0
…                     …
1090                    0
1091                    0
1092                    0
1093                    0
1094                    0

[1095 rows x 1 columns]
```

This code is casting every value in the `LotFrontage_Missing` column to an integer, achieving the same result as the `OrdinalEncoder` example with less code.

The downside of using this approach is that it doesn't fit into a scikit-learn pipeline as neatly because it is using `pandas` to do the transformation instead of scikit-learn.

In the future, you will need to make your own determination of which strategy to use!

1.7.2 Multiple Categories

Unlike `Street` and `LotFrontage_Missing`, `FireplaceQu` has more than two categories. Therefore the process for encoding it numerically is a bit more complicated, because we will need to create multiple "dummy" columns that are each representing one category.

To do this, we can use a `OneHotEncoder` from `sklearn.preprocessing` (documentation here).

The first several steps are very similar to all of the other transformers we've used so far, although the process of combining the data with the original data differs.

In the cells below, complete steps `(0)-(4)` of preprocessing the `FireplaceQu` column using a `OneHotEncoder`:

[95]:
```
# Replace None with appropriate code

# (0) import OneHotEncoder from sklearn.preprocessing
from sklearn.preprocessing import OneHotEncoder

# (1) Create a variable fireplace_qu_train
# extracted from X_train
# (double brackets due to shape expected by OHE)
fireplace_qu_train = X_train[["FireplaceQu"]]

# (2) Instantiate a OneHotEncoder with categories="auto",
# sparse=False, and handle_unknown="ignore"
ohe = OneHotEncoder(categories = "auto", sparse = False,
                    handle_unknown = "ignore")

# (3) Fit the encoder on fireplace_qu_train
ohe.fit(fireplace_qu_train)

# Inspect the categories of the fitted encoder
ohe.categories_
```

[95]:
```
[array(['Ex', 'Fa', 'Gd', 'N/A', 'Po', 'TA'], dtype=object)]
```

[96]:
```
# Replace None with appropriate code

# (4) Transform fireplace_qu_train using the encoder and
# assign the result to fireplace_qu_encoded_train
fireplace_qu_encoded_train = ohe.transform(fireplace_qu_train)

# Visually inspect fireplace_qu_encoded_train
```

```
fireplace_qu_encoded_train
```

[96]:
```
array([[0., 0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0.],
       …,
       [0., 0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1.]])
```

Notice that this time, unlike with `MissingIndicator`, `SimpleImputer`, or `OrdinalEncoder`, we have created multiple columns of data out of a single column. The code below converts this unlabeled NumPy array into a readable pandas dataframe in preparation for merging it back with the rest of `X_train`:

[97]:
```
# Run this cell without changes

# (5a) Make the transformed data into a dataframe
fireplace_qu_encoded_train = pd.DataFrame(
    # Pass in NumPy array
    fireplace_qu_encoded_train,
    # Set the column names to the categories found by OHE
    columns=ohe.categories_[0],
    # Set the index to match X_train's index
    index=X_train.index
)

# Visually inspect new dataframe
fireplace_qu_encoded_train
```

[97]:

	Ex	Fa	Gd	N/A	Po	TA
1023	0.0	0.0	1.0	0.0	0.0	0.0
810	0.0	1.0	0.0	0.0	0.0	0.0
1384	0.0	0.0	0.0	1.0	0.0	0.0
626	0.0	0.0	0.0	0.0	0.0	1.0
813	0.0	0.0	0.0	1.0	0.0	0.0
…	…	…	…	…	…	…
1095	0.0	0.0	1.0	0.0	0.0	0.0
1130	0.0	0.0	0.0	0.0	0.0	1.0
1294	0.0	0.0	0.0	1.0	0.0	0.0
860	0.0	0.0	1.0	0.0	0.0	0.0
1126	0.0	0.0	0.0	0.0	0.0	1.0

[1095 rows x 6 columns]

A couple notes on the code above:

- The main goal of converting this into a dataframe (rather than converting `X_train` into a

NumPy array, which would also allow them to be combined) is **readability** — to help you and others understand what your code is doing, and to help you debug. Eventually when you write this code as a pipeline, it will be NumPy arrays "under the hood".

- We are using just the **raw categories** from `FireplaceQu` as our new dataframe columns, but you'll also see examples where a lambda function or list comprehension is used to create column names indicating the original column name, e.g. `FireplaceQu_Ex`, `FireplaceQu_Fa` rather than just `Ex`, `Fa`. This is a design decision based on readability — the scikit-learn model will work the same either way.
- It is very important that **the index of the new dataframe matches the index of the main `X_train` dataframe**. Because we used `train_test_split`, the index of `X_train` is shuffled, so it goes `1023`, `810`, `1384` etc. instead of `0`, `1`, `2`, etc. If you don't specify an index for the new dataframe, it will assign the first record to the index `0` rather than `1023`. If you are ever merging encoded data like this and a bunch of NaNs start appearing, make sure that the indexes are lined up correctly! You also may see examples where the index of `X_train` has been reset, rather than specifying the index of the new dataframe — either way works.

Next, we want to drop the original `FireplaceQu` column containing the categorical data:

(For previous transformations we didn't need to drop anything because we were replacing 1 column with 1 new column in place, but one-hot encoding works differently.)

[98]:
```
# Run this cell without changes

# (5b) Drop original FireplaceQu column
X_train.drop("FireplaceQu", axis=1, inplace=True)

# Visually inspect X_train
X_train
```

```
/opt/anaconda3/envs/learn-env/lib/python3.8/site-
packages/pandas/core/frame.py:4163: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

[98]:

	LotFrontage	LotArea	Street	OverallQual	OverallCond	YearBuilt	\
1023	43.0	3182	1.0	7	5	2005	
810	78.0	10140	1.0	6	6	1974	
1384	60.0	9060	1.0	6	5	1939	
626	70.0	12342	1.0	5	5	1960	
813	75.0	9750	1.0	6	6	1958	
…	…	…	…	…	…	…	
1095	78.0	9317	1.0	6	5	2006	
1130	65.0	7804	1.0	4	3	1928	
1294	60.0	8172	1.0	5	7	1955	
860	55.0	7642	1.0	7	8	1918	
1126	53.0	3684	1.0	7	5	2007	

```
      YearRemodAdd  GrLivArea  FullBath  BedroomAbvGr  TotRmsAbvGrd  \
1023          2006       1504         2             2             7
810           1999       1309         1             3             5
1384          1950       1258         1             2             6
626           1978       1422         1             3             6
813           1958       1442         1             4             7
…              …          …         …            …            …
1095          2006       1314         2             3             6
1130          1950       1981         2             4             7
1294          1990        864         1             2             5
860           1998       1426         1             3             7
1126          2007       1555         2             2             7

      Fireplaces  MoSold  YrSold  LotFrontage_Missing
1023           1       5    2008                  0.0
810            1       1    2006                  0.0
1384           0      10    2009                  0.0
626            1       8    2007                  1.0
813            0       4    2007                  0.0
…            …      …     …                    …
1095           1       3    2007                  0.0
1130           2      12    2009                  0.0
1294           0       4    2006                  0.0
860            1       6    2007                  0.0
1126           1       6    2009                  0.0

[1095 rows x 15 columns]
```

Finally, we want to concatenate the new dataframe together with the original `X_train`:

[99]:
```python
# Run this cell without changes

# (5c) Concatenate the new dataframe with current X_train
X_train = pd.concat([X_train, fireplace_qu_encoded_train], axis=1)

# Visually inspect X_train
X_train
```

[99]:
```
      LotFrontage  LotArea  Street  OverallQual  OverallCond  YearBuilt  \
1023         43.0     3182     1.0            7            5       2005
810          78.0    10140     1.0            6            6       1974
1384         60.0     9060     1.0            6            5       1939
626          70.0    12342     1.0            5            5       1960
813          75.0     9750     1.0            6            6       1958
…             …       …      …           …           …          …
1095         78.0     9317     1.0            6            5       2006
```

```
1130         65.0     7804    1.0              4              3         1928
1294         60.0     8172    1.0              5              7         1955
860          55.0     7642    1.0              7              8         1918
1126         53.0     3684    1.0              7              5         2007

      YearRemodAdd  GrLivArea  FullBath  BedroomAbvGr  …  Fireplaces  \
1023          2006       1504         2             2  …           1
810           1999       1309         1             3  …           1
1384          1950       1258         1             2  …           0
626           1978       1422         1             3  …           1
813           1958       1442         1             4  …           0
…              …          …        …            …   …          …
1095          2006       1314         2             3  …           1
1130          1950       1981         2             4  …           2
1294          1990        864         1             2  …           0
860           1998       1426         1             3  …           1
1126          2007       1555         2             2  …           1

      MoSold  YrSold  LotFrontage_Missing   Ex   Fa   Gd  N/A   Po   TA
1023       5    2008                  0.0  0.0  0.0  1.0  0.0  0.0  0.0
810        1    2006                  0.0  0.0  1.0  0.0  0.0  0.0  0.0
1384      10    2009                  0.0  0.0  0.0  0.0  1.0  0.0  0.0
626        8    2007                  1.0  0.0  0.0  0.0  0.0  0.0  1.0
813        4    2007                  0.0  0.0  0.0  0.0  1.0  0.0  0.0
…        …     …                    …   …   …   …   …   …   …
1095       3    2007                  0.0  0.0  0.0  1.0  0.0  0.0  0.0
1130      12    2009                  0.0  0.0  0.0  0.0  0.0  0.0  1.0
1294       4    2006                  0.0  0.0  0.0  0.0  1.0  0.0  0.0
860        6    2007                  0.0  0.0  0.0  1.0  0.0  0.0  0.0
1126       6    2009                  0.0  0.0  0.0  0.0  0.0  0.0  1.0

[1095 rows x 21 columns]
```

[100]:

```
# Run this cell without changes
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1095 entries, 1023 to 1126
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   LotFrontage          1095 non-null   float64
 1   LotArea              1095 non-null   int64
 2   Street               1095 non-null   float64
 3   OverallQual          1095 non-null   int64
 4   OverallCond          1095 non-null   int64
 5   YearBuilt            1095 non-null   int64
```

```
 6   YearRemodAdd         1095 non-null   int64
 7   GrLivArea            1095 non-null   int64
 8   FullBath             1095 non-null   int64
 9   BedroomAbvGr         1095 non-null   int64
 10  TotRmsAbvGrd         1095 non-null   int64
 11  Fireplaces           1095 non-null   int64
 12  MoSold               1095 non-null   int64
 13  YrSold               1095 non-null   int64
 14  LotFrontage_Missing  1095 non-null   float64
 15  Ex                   1095 non-null   float64
 16  Fa                   1095 non-null   float64
 17  Gd                   1095 non-null   float64
 18  N/A                  1095 non-null   float64
 19  Po                   1095 non-null   float64
 20  TA                   1095 non-null   float64
dtypes: float64(9), int64(12)
memory usage: 188.2 KB
```

Ok, everything is numeric now! We have completed the minimum necessary preprocessing to use these features in a scikit-learn model!

[101]:
```python
# Run this cell without changes
model.fit(X_train, y_train)
```

[101]:
```
LinearRegression()
```

Great, no error this time.

Let's use cross validation to take a look at the model's performance:

[102]:
```python
# Run this cell without changes
from sklearn.model_selection import cross_val_score

cross_val_score(model, X_train, y_train, cv=3)
```

[102]:
```
array([0.75131297, 0.66405511, 0.80347971])
```

Not terrible, we are explaining between 66% and 80% of the variance in the target with our current feature set. Let's say that this is our final model and move on to preparing the test data.

1.8 4. Preprocess Test Data

Apply Steps 1-3 to the test data in order to perform a final model evaluation.

This part is done for you, and it should work automatically, assuming you didn't change the names of any of the transformer objects. Note that we are intentionally **not instantiating or fitting the transformers** here, because you always want to fit transformers on the training data only.

Step 1: Drop Irrelevant Columns

[103]:
```python
# Run this cell without changes
X_test = X_test.loc[:, relevant_columns]
```

Step 2: Handle Missing Values

[104]:
```python
# Run this cell without changes

# Replace FireplaceQu NaNs with "N/A"s
X_test["FireplaceQu"] = X_test["FireplaceQu"].fillna("N/A")

# Add missing indicator for lot frontage
frontage_test = X_test[["LotFrontage"]]
frontage_missing_test = missing_indicator.transform(frontage_test)
X_test["LotFrontage_Missing"] = frontage_missing_test

# Impute missing lot frontage values
frontage_imputed_test = imputer.transform(frontage_test)
X_test["LotFrontage"] = frontage_imputed_test

# Check that there are no more missing values
X_test.isna().sum()
```

[104]:
```
LotFrontage            0
LotArea                0
Street                 0
OverallQual            0
OverallCond            0
YearBuilt              0
YearRemodAdd           0
GrLivArea              0
FullBath               0
BedroomAbvGr           0
TotRmsAbvGrd           0
Fireplaces             0
FireplaceQu            0
MoSold                 0
YrSold                 0
LotFrontage_Missing    0
dtype: int64
```

Step 3: Convert Categorical Features into Numbers

[105]:
```python
# Run this cell without changes

# Encode street type
street_test = X_test[["Street"]]
street_encoded_test = encoder_street.transform(street_test).flatten()
X_test["Street"] = street_encoded_test
```

```python
# Encode frontage missing
frontage_missing_test = X_test[["LotFrontage_Missing"]]
frontage_missing_encoded_test = encoder_frontage_missing.
 ↪transform(frontage_missing_test).flatten()
X_test["LotFrontage_Missing"] = frontage_missing_encoded_test

# One-hot encode fireplace quality
fireplace_qu_test = X_test[["FireplaceQu"]]
fireplace_qu_encoded_test = ohe.transform(fireplace_qu_test)
fireplace_qu_encoded_test = pd.DataFrame(
    fireplace_qu_encoded_test,
    columns=ohe.categories_[0],
    index=X_test.index
)
X_test.drop("FireplaceQu", axis=1, inplace=True)
X_test = pd.concat([X_test, fireplace_qu_encoded_test], axis=1)

# Visually inspect X_test
X_test
```

[105]:

```
      LotFrontage  LotArea  Street  OverallQual  OverallCond  YearBuilt  \
892          70.0     8414     1.0            6            8       1963
1105         98.0    12256     1.0            8            5       1994
413          56.0     8960     1.0            5            6       1927
522          50.0     5000     1.0            6            7       1947
1036         89.0    12898     1.0            9            5       2007
…               …        …       …            …            …          …
988          70.0    12046     1.0            6            6       1976
243          75.0    10762     1.0            6            6       1980
1342         70.0     9375     1.0            8            5       2002
1057         70.0    29959     1.0            7            6       1994
1418         71.0     9204     1.0            5            5       1963

      YearRemodAdd  GrLivArea  FullBath  BedroomAbvGr  …  Fireplaces  \
892           2003       1068         1             3  …           0
1105          1995       2622         2             3  …           2
413           1950       1028         1             2  …           1
522           1950       1664         2             3  …           2
1036          2008       1620         2             2  …           1
…                …          …         …             …  …           …
988           1976       2030         2             4  …           1
243           1980       1217         1             3  …           1
1342          2002       2169         2             3  …           1
1057          1994       1850         2             3  …           1
1418          1963       1144         1             3  …           0
```

```
      MoSold  YrSold  LotFrontage_Missing   Ex   Fa   Gd  N/A   Po   TA
892        2    2006                  0.0  0.0  0.0  0.0  1.0  0.0  0.0
1105       4    2010                  0.0  0.0  0.0  0.0  0.0  0.0  1.0
413        3    2010                  0.0  0.0  0.0  1.0  0.0  0.0  0.0
522       10    2006                  0.0  0.0  0.0  1.0  0.0  0.0  0.0
1036       9    2009                  0.0  1.0  0.0  0.0  0.0  0.0  0.0
…       …     …                   …   …   …   …   …   …   …
988        6    2007                  1.0  0.0  0.0  0.0  0.0  0.0  1.0
243        4    2009                  0.0  0.0  0.0  0.0  0.0  0.0  1.0
1342       8    2007                  1.0  0.0  0.0  1.0  0.0  0.0  0.0
1057       1    2009                  1.0  0.0  0.0  1.0  0.0  0.0  0.0
1418       8    2008                  0.0  0.0  0.0  0.0  1.0  0.0  0.0

[365 rows x 21 columns]
```

Fit the model on the full training set, evaluate on test set:

[106]:
```python
# Run this cell without changes
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

[106]: 0.8016639002688316

Great, that worked! Now we have completed the full process of preprocessing the Ames Housing data in preparation for machine learning!

1.9 Summary

In this cumulative lab, you used various techniques to prepare the Ames Housing data for modeling. You filtered down the full dataset to only relevant columns, filled in missing values, and converted categorical data into numeric data. Each time, you practiced the scikit-learn transformer workflow by instantiating the transformer, fitting on the relevant training data, transforming the training data, and transforming the test data at the end (without re-instantiating or re-fitting the transformer object).