learn-co-curriculum / dsc-categorical-variables-regression-lab Public

View license

0 stars 10 forks

Star Watch

Code Issues Pull requests Actions Projects Security Insights

solution

This branch is 10 commits ahead, 14 commits behind master.

hoffm386 differentiation between features, add modeling ... on May 12 15

View code

README.md

Dealing with Categorical Variables - Lab

Introduction

In this lab, you'll explore the Ames Housing dataset and identify numeric and categorical variables. Then you'll transform some categorical data and use it in a multiple regression model.

Objectives

You will be able to:

- Determine whether variables are categorical or numeric
- Use one-hot encoding to create dummy variables

Step 1: Load the Ames Housing Dataset

Import `pandas` , and use it to load the file `ames.csv` into a dataframe called `ames` . If you pass in the argument `index_col=0` this will set the "Id" feature as the index.

```
# Load the dataset

import pandas as pd
ames = pd.read_csv('ames.csv', index_col=0)
```

Visually inspect `ames` (it's ok if you can't see all of the columns).

```
ames
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	Lo
Id							
1	60	RL	65.0	8450	Pave	NaN	Re
2	20	RL	80.0	9600	Pave	NaN	Re
3	60	RL	68.0	11250	Pave	NaN	IR
4	70	RL	60.0	9550	Pave	NaN	IR
5	60	RL	84.0	14260	Pave	NaN	IR
...	...	...	...	...	...	...	...
1456	60	RL	62.0	7917	Pave	NaN	Re
1457	20	RL	85.0	13175	Pave	NaN	Re

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	Lo
Id							
1458	70	RL	66.0	9042	Pave	NaN	Re
1459	20	RL	68.0	9717	Pave	NaN	Re
1460	20	RL	75.0	9937	Pave	NaN	Re

1460 rows × 80 columns

Go ahead and drop all **columns** with missing data, to simplify the problem. Remember that you can use the `dropna` method (documentation here).

```
ames.dropna(axis=1, inplace=True)
ames
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	MSSubClass	MSZoning	LotArea	Street	LotShape	LandContour
Id						
1	60	RL	8450	Pave	Reg	Lvl
2	20	RL	9600	Pave	Reg	Lvl
3	60	RL	11250	Pave	IR1	Lvl
4	70	RL	9550	Pave	IR1	Lvl
5	60	RL	14260	Pave	IR1	Lvl
...	...	...	...	...	...	...
1456	60	RL	7917	Pave	Reg	Lvl

	MSSubClass	MSZoning	LotArea	Street	LotShape	LandContour
Id						
1457	20	RL	13175	Pave	Reg	Lvl
1458	70	RL	9042	Pave	Reg	Lvl
1459	20	RL	9717	Pave	Reg	Lvl
1460	20	RL	9937	Pave	Reg	Lvl

1460 rows × 61 columns

Step 2: Identify Numeric and Categorical Variables

The file `data_description.txt` , located in this repository, has a full description of all variables.

Using this file as well as `pandas` techniques, identify the following predictors:

1. A **continuous numeric** predictor
2. A **discrete numeric** predictor
3. A **string categorical** predictor
4. A **discrete categorical** predictor

(Note that `SalePrice` is the target variable and should not be selected as a predictor.)

For each of these predictors, visualize the relationship between the predictor and `SalePrice` using an appropriate plot.

Finding these will take some digging -- don't be discouraged if they're not immediately obvious! The Ames Housing dataset is a lot more complex than the Auto MPG dataset. There is also no single right answer here.

Continuous Numeric Predictor

```
# A continuous numeric feature should be encoded as some sort of number
ames.select_dtypes("number")
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

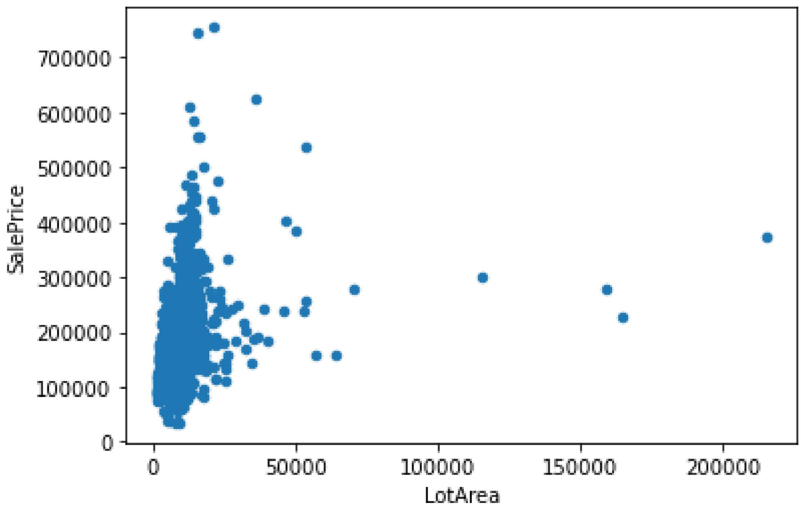
</style>

	MSSubClass	LotArea	OverallQual	OverallCond	YearBuilt	YearRe
Id						
1	60	8450	7	5	2003	2003
2	20	9600	6	8	1976	1976
3	60	11250	7	5	2001	2002
4	70	9550	7	5	1915	1970
5	60	14260	8	5	2000	2000
...	...	...	...	...	...	...
1456	60	7917	6	5	1999	2000
1457	20	13175	6	6	1978	1988
1458	70	9042	7	9	1941	2006
1459	20	9717	5	6	1950	1996
1460	20	9937	5	6	1965	1965

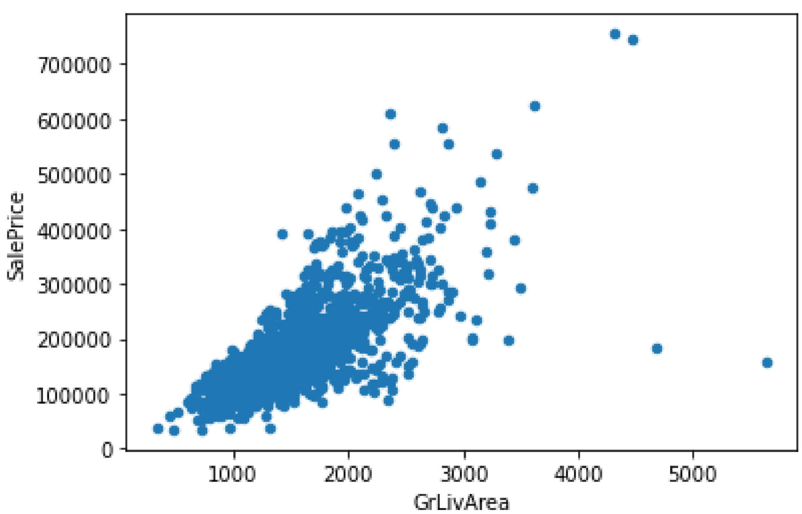
1460 rows × 34 columns

```
# Just reading from left to right,

# MSSubClass doesn't seem like a good choice, because it seems like it's a
# "code" (i.e. categorical) rather than truly numeric

# LotArea does seem like a reasonable choice. The description says:
# LotArea: Lot size in square feet

# That sounds like a number
```

```
ames.plot.scatter(x="LotArea", y="SalePrice");
```




```
# Hmm, that doesn't look quite like a linear relationship. Let's try GrLivArea inste
ames.plot.scatter(x="GrLivArea", y="SalePrice");
```

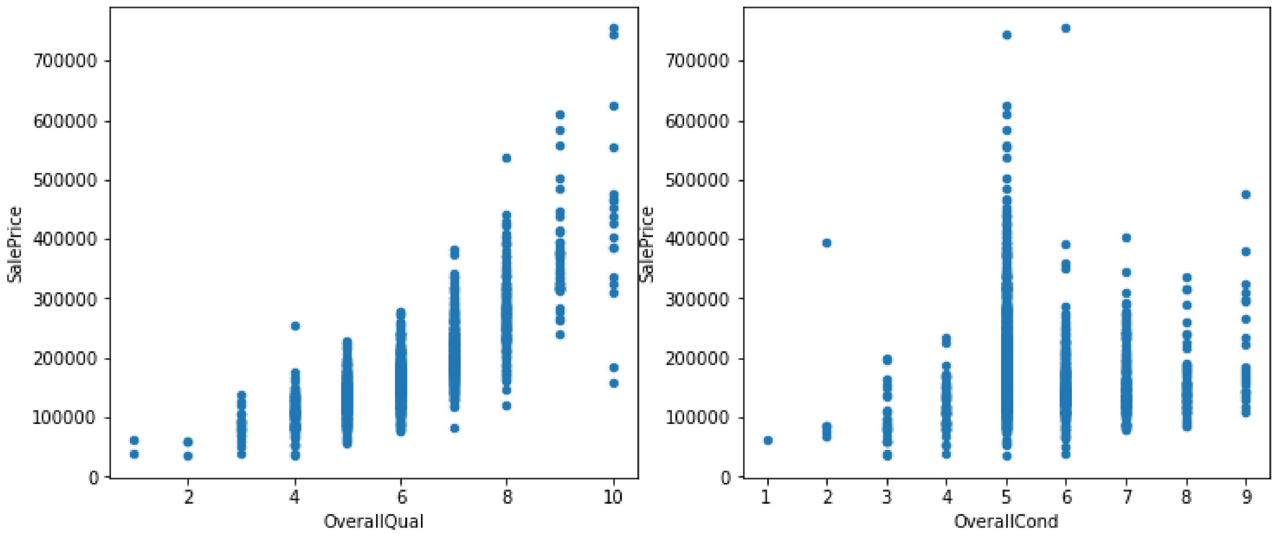



```
# That looks better. Let's go with that.
```

Discrete Numeric Predictor

```
# Continuing left to right, OverallQual and OverallCond are definitely discrete
import matplotlib.pyplot as plt
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12,5))
```

```
ames.plot.scatter(x="OverallQual", y="SalePrice", ax=ax1)
ames.plot.scatter(x="OverallCond", y="SalePrice", ax=ax2);
```




```
# But are they numeric, and linearly related to SalePrice?

# OverallQual looks reasonable. It does seem like a quality of
# 8 is "twice as much" as a quality of 4. Let's call that our
# discrete numeric predictor.
```

String Categorical Predictor

```
# Now let's select all string columns
ames.select_dtypes("object")
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	MSZoning	Street	LotShape	LandContour	Utilities	LotConfig
Id						
1	RL	Pave	Reg	Lvl	AllPub	Inside
2	RL	Pave	Reg	Lvl	AllPub	FR2
3	RL	Pave	IR1	Lvl	AllPub	Inside
4	RL	Pave	IR1	Lvl	AllPub	Corner
5	RL	Pave	IR1	Lvl	AllPub	FR2
...	...	...	...	...	...	...
1456	RL	Pave	Reg	Lvl	AllPub	Inside
1457	RL	Pave	Reg	Lvl	AllPub	Inside
1458	RL	Pave	Reg	Lvl	AllPub	Inside
1459	RL	Pave	Reg	Lvl	AllPub	Inside
1460	RL	Pave	Reg	Lvl	AllPub	Inside

1460 rows × 27 columns

```
# That is still a lot of options. What if we look for options where there are
# only a few different unique categories?
ames.select_dtypes("object").nunique().sort_values()
```

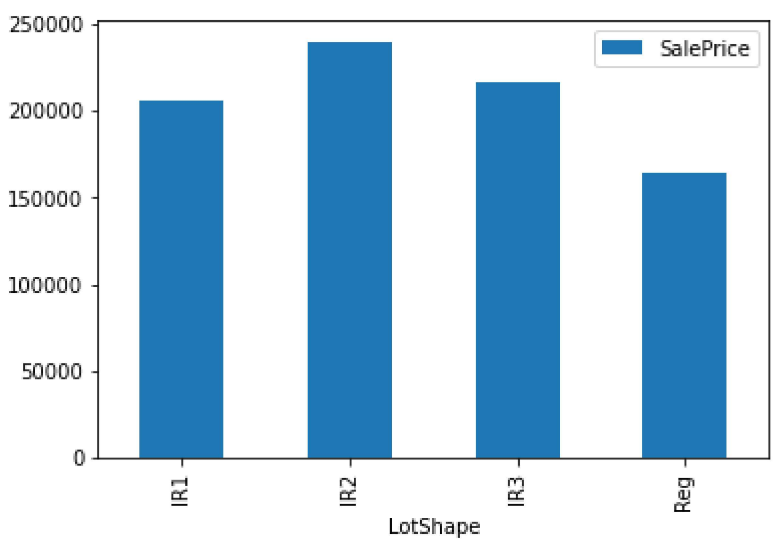
```
Street          2
Utilities       2
CentralAir      2
LandSlope       3
PavedDrive      3
LotShape        4
LandContour     4
KitchenQual     4
ExterQual       4
MSZoning        5
LotConfig       5
BldgType        5
HeatingQC       5
ExterCond       5
Heating         6
Foundation      6
```

```
SaleCondition     6
RoofStyle         6
Functional        7
HouseStyle        8
Condition2        8
RoofMatl          8
SaleType          9
Condition1        9
Exterior1st      15
Exterior2nd      16
Neighborhood     25
dtype: int64
```

```
# LotShape looks good, since there's a good amount of homes in
# multiple categories
ames["LotShape"].value_counts()
```

```
Reg    925
IR1    484
IR2     41
IR3     10
Name: LotShape, dtype: int64
```

```
# Let's plot LotShape
ames.groupby("LotShape").mean().sort_index().plot.bar(y="SalePrice");
```




```
# That looks like a good categorical predictor. Even though
# some of the values have numbers in them, they don't seem to
```

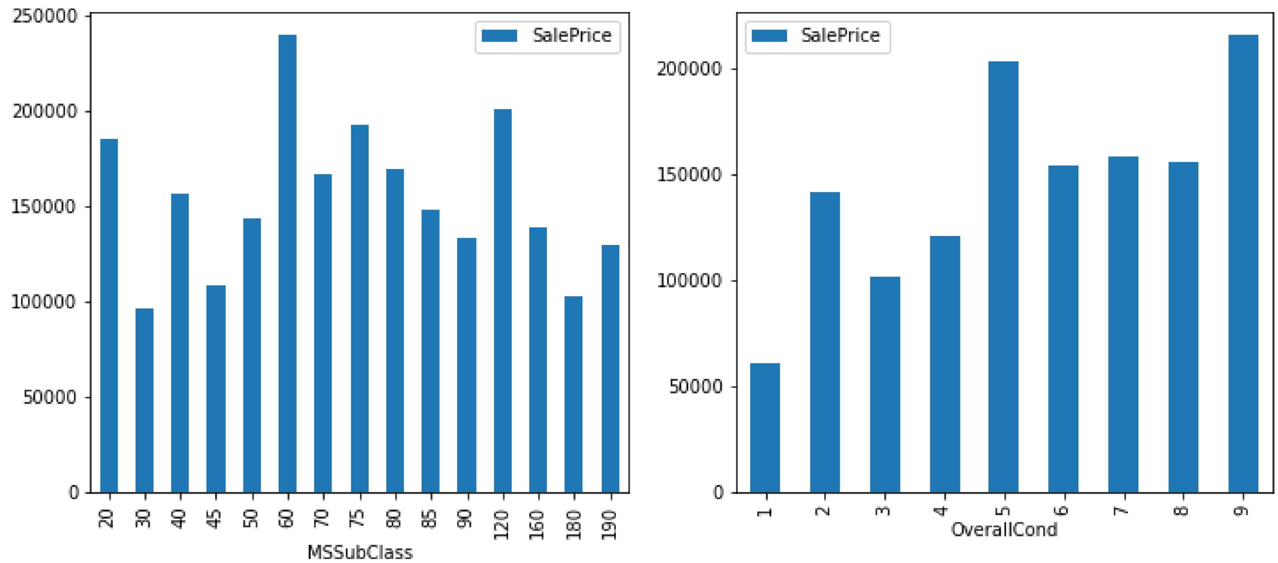
```
# be linearly related
```

Discrete Categorical Predictor

```
# Let's do the same process of looking for the number of unique
# values, except for numeric columns
ames.select_dtypes("number").nunique().sort_values()
```

```
HalfBath            3
BsmtHalfBath        3
FullBath            4
Fireplaces          4
KitchenAbvGr        4
BsmtFullBath        4
GarageCars          5
YrSold              5
PoolArea            8
BedroomAbvGr        8
OverallCond         9
OverallQual        10
MoSold             12
TotRmsAbvGrd       12
MSSubClass         15
3SsnPorch          20
MiscVal            21
LowQualFinSF       24
YearRemodAdd       61
ScreenPorch        76
YearBuilt         112
EnclosedPorch     120
BsmtFinSF2        144
OpenPorchSF       202
WoodDeckSF        274
2ndFlrSF          417
GarageArea        441
BsmtFinSF1        637
SalePrice         663
TotalBsmtSF       721
1stFlrSF          753
BsmtUnfSF         780
GrLivArea         861
LotArea          1073
dtype: int64
```

```
# Hmm, most of those with relatively few categories look like they are actually nume

# Let's actually go back to earlier, when we said that MSSubClass and OverallCond we
# not numeric

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12,5))

ames.groupby("MSSubClass").mean().plot.bar(y="SalePrice", ax=ax1)
ames.groupby("OverallCond").mean().plot.bar(y="SalePrice", ax=ax2);
```




```
# MSSubClass looks least like it could be used as a numeric predictor, so
# let's go with that
```

Step 3: Build a Multiple Regression Model with Your Chosen Predictors

Choose the best-looking 3 out of 4 predictors to include in your model.

Make sure that you one-hot encode your categorical predictor(s) (regardless of whether the current data type is a string or number) first!

```
# MSSubClass is messiest, so exclude that
y = ames["SalePrice"]
X = ames[["GrLivArea", "OverallQual", "LotShape"]]
X
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	GrLivArea	OverallQual	LotShape
Id			
1	1710	7	Reg
2	1262	6	Reg
3	1786	7	IR1
4	1717	7	IR1
5	2198	8	IR1
...	...	...	...
1456	1647	6	Reg
1457	2073	6	Reg
1458	2340	7	Reg
1459	1078	5	Reg
1460	1256	5	Reg

1460 rows × 3 columns

```
X = pd.get_dummies(X, columns=["LotShape"], drop_first=True)
X
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

</style>

	GrLivArea	OverallQual	LotShape_IR2	LotShape_IR3	LotShape_Reg
Id					
1	1710	7	0	0	1
2	1262	6	0	0	1
3	1786	7	0	0	0
4	1717	7	0	0	0
5	2198	8	0	0	0
...	...	...	...	...	...
1456	1647	6	0	0	1
1457	2073	6	0	0	1
1458	2340	7	0	0	1
1459	1078	5	0	0	1
1460	1256	5	0	0	1

1460 rows × 5 columns

```
"""
LotShape IR1 (slightly irregular)
"""
```

```
import statsmodels.api as sm

model = sm.OLS(y, sm.add_constant(X))
results = model.fit()

print(results.summary())
```

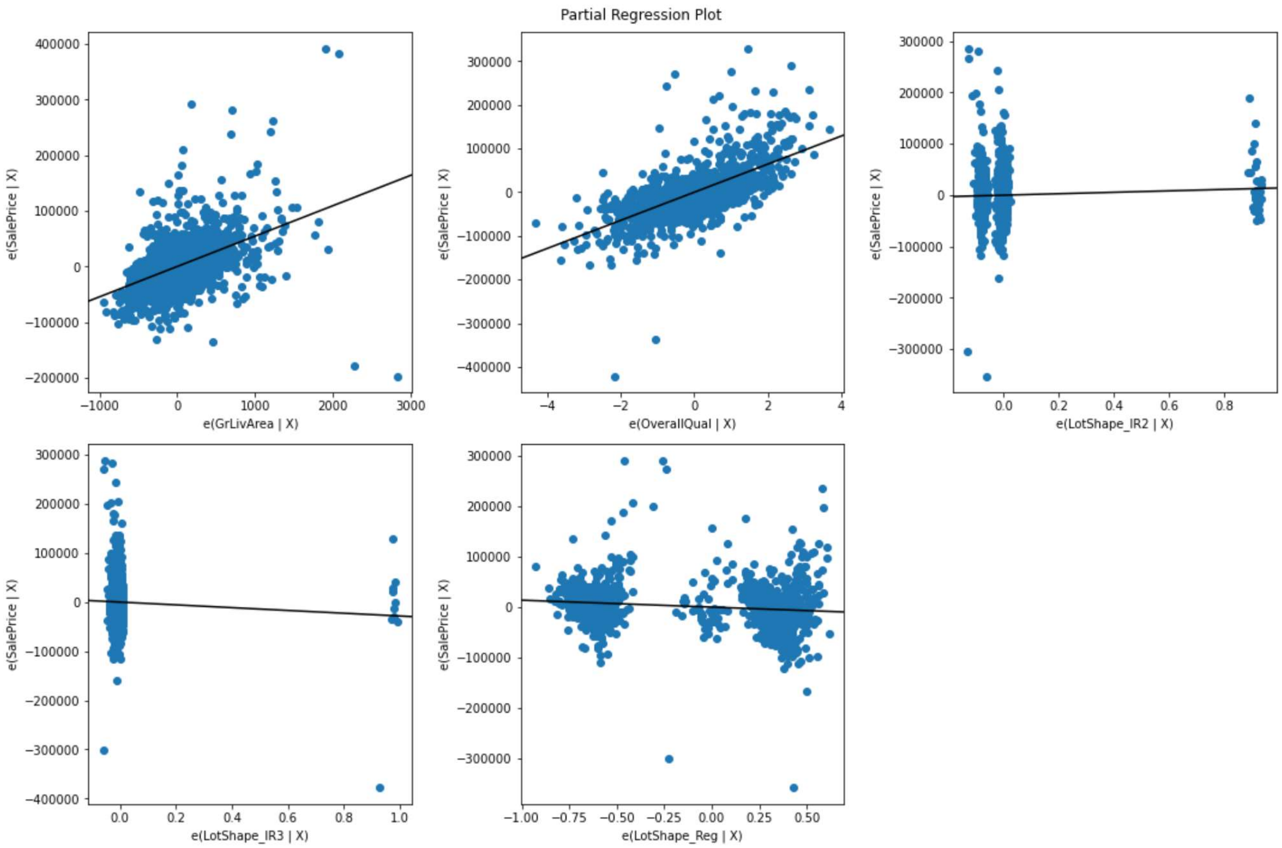
```
                            OLS Regression Results
==============================================================================
Dep. Variable:              SalePrice   R-squared:                       0.723
Model:                            OLS   Adj. R-squared:                  0.722
Method:                 Least Squares   F-statistic:                     759.4
Date:                Wed, 11 May 2022   Prob (F-statistic):               0.00
Time:                        18:38:05   Log-Likelihood:                -17607.
No. Observations:                1460   AIC:                         3.523e+04
Df Residuals:                    1454   BIC:                         3.526e+04
Df Model:                           5
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const         -8.92e+04   5574.151    -16.002      0.000      -1e+05   -7.83e+04
GrLivArea       54.5636      2.613     20.881      0.000      49.438      59.689
OverallQual   3.213e+04    990.676     32.429      0.000    3.02e+04    3.41e+04
LotShape_IR2  1.407e+04   6822.693      2.062      0.039     683.451    2.75e+04
LotShape_IR3  -2.84e+04    1.34e+04    -2.116      0.035   -5.47e+04   -2070.141
LotShape_Reg -1.376e+04   2396.494     -5.742      0.000   -1.85e+04   -9059.887
==============================================================================
Omnibus:                      366.932   Durbin-Watson:                   1.982
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             8050.129
Skew:                           0.620   Prob(JB):                         0.00
Kurtosis:                      14.437   Cond. No.                     1.97e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.97e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Step 4: Create Partial Regression Plots for Features

For each feature of the regression above (including the dummy features), plot the partial regression.

```
fig = plt.figure(figsize=(15,10))
sm.graphics.plot_partregress_grid(
    results,
    exog_idx=list(X.columns),
    grid=(2,3),
    fig=fig)
```

```python
plt.tight_layout()
plt.show()
```



Step 5: Calculate an Error-Based Metric

In addition to the adjusted R-Squared that we can see in the model summary, calculate either MAE or RMSE for this model.

```python
from sklearn.metrics import mean_absolute_error

y_pred = results.predict(sm.add_constant(X))
mean_absolute_error(y, y_pred)
```

```
28396.050798992394
```

Step 6: Summarize Findings

Between the model results, partial regression plots, and error-based metric, what does this model tell you? What would your next steps be to improve the model?

```
"""
Our model is statistically significant overall, and explains about 72% of the
variance in SalePrice. On average it is off by about $28k in its predictions
of home price.

All of our coefficients are statistically significant

So we can say that:

const: When above-grade living area is 0, overall quality is 0, and lot shape
       is slightly irregular, we would expect a home sale price of -$89k

GrLivArea: For each increase of 1 sqft in above-grade living area, we see an
           associated increase of about $55 in sale price

OverallQual: For each increase of 1 in overall quality, we see an associated
             increase of about $32k in sale price

LotShape_IR2: Compared to a slightly irregular lot shape, we see an associated
              increase of about $14k for a moderately irregular lot shape

LotShape_IR3: Compared to a slightly irregular lot shape, we see an associated
              decrease of about $28k for an irregular lot shape

LotShape_Reg: Compared to a slightly irregular lot shape, we see an associated
              decrease of about $14k for a regular lot shape

Looking at the partial regression plots, the dummy variables look fairly
different from the other variables. They tend to have two clusters rather than
a continuous "cloud". Given the relatively small numbers in IR2 and IR3, I
wonder if a better model would have these binned together with IR1 instead.
"""
```

Level Up (Optional)

Try transforming X using scikit-learn *and* fitting a scikit-learn linear regression as well. If there are any differences in the result, investigate them.

```python
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LinearRegression
```

```python
X_sklearn = ames[["GrLivArea", "OverallQual", "LotShape"]].copy()
X_cat = X_sklearn[["LotShape"]]
X_numeric = X_sklearn.drop("LotShape", axis=1)
```

```
ohe = OneHotEncoder(drop="first", sparse=False)
ohe.fit(X_cat)
X_cat_ohe = pd.DataFrame(
    data=ohe.transform(X_cat),
    columns=[f"LotShape_{cat}" for cat in ohe.categories_[0][1:]],
    index=X_cat.index
)
X_cat_ohe
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	LotShape_IR2	LotShape_IR3	LotShape_Reg
Id			
1	0.0	0.0	1.0
2	0.0	0.0	1.0
3	0.0	0.0	0.0
4	0.0	0.0	0.0
5	0.0	0.0	0.0
...	...	...	...
1456	0.0	0.0	1.0
1457	0.0	0.0	1.0
1458	0.0	0.0	1.0
1459	0.0	0.0	1.0
1460	0.0	0.0	1.0

1460 rows × 3 columns

```
X_sklearn_final = pd.concat([X_numeric, X_cat_ohe], axis=1)
X_sklearn_final
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	GrLivArea	OverallQual	LotShape_IR2	LotShape_IR3	LotShape_Reg
Id					
1	1710	7	0.0	0.0	1.0
2	1262	6	0.0	0.0	1.0
3	1786	7	0.0	0.0	0.0
4	1717	7	0.0	0.0	0.0
5	2198	8	0.0	0.0	0.0
...	...	...	...	...	...
1456	1647	6	0.0	0.0	1.0
1457	2073	6	0.0	0.0	1.0
1458	2340	7	0.0	0.0	1.0
1459	1078	5	0.0	0.0	1.0
1460	1256	5	0.0	0.0	1.0

1460 rows × 5 columns

```
lr = LinearRegression()
lr.fit(X_sklearn_final, y)
```

```
LinearRegression()
```

```
import numpy as np
print(results.params.values)
print(np.append(lr.intercept_, lr.coef_))
```

```
[-8.91984448e+04  5.45636496e+01  3.21262730e+04  1.40668251e+04
 -2.84021192e+04 -1.37608433e+04]
[-8.91984448e+04  5.45636496e+01  3.21262730e+04  1.40668251e+04
 -2.84021192e+04 -1.37608433e+04]
```

```
mean_absolute_error(y, lr.predict(X_sklearn_final))
```

```
28396.050798992903
```

Summary

In this lab, you practiced your knowledge of categorical variables on the Ames Housing dataset! Specifically, you practiced distinguishing numeric and categorical data. You then created dummy variables using one hot encoding in order to build a multiple regression model.

Releases

No releases published

Packages

No packages published

Contributors 5

Languages

- Jupyter Notebook 92.6%
- Python 7.4%