Dealing with Categorical Variables Lab

August 6, 2024

1 Dealing with Categorical Variables - Lab

1.1 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.

1.2 Objectives

You will be able to:

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

1.3 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.

[1]:
```
# Your code here - load the dataset
import pandas as pd
import matplotlib.pyplot as plt
import warnings

ames = pd.read_csv("ames.csv", index_col=0)
```

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

[2]:
```
# Your code here
ames.head()
```

[2]:
```
    MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
Id
1           60       RL         65.0     8450   Pave   NaN      Reg
2           20       RL         80.0     9600   Pave   NaN      Reg
3           60       RL         68.0    11250   Pave   NaN      IR1
4           70       RL         60.0     9550   Pave   NaN      IR1
5           60       RL         84.0    14260   Pave   NaN      IR1

   LandContour Utilities LotConfig  … PoolArea PoolQC Fence MiscFeature  \
```

```
Id                                    …
1          Lvl     AllPub    Inside   …         0     NaN    NaN          NaN
2          Lvl     AllPub       FR2   …         0     NaN    NaN          NaN
3          Lvl     AllPub    Inside   …         0     NaN    NaN          NaN
4          Lvl     AllPub    Corner   …         0     NaN    NaN          NaN
5          Lvl     AllPub       FR2   …         0     NaN    NaN          NaN

   MiscVal MoSold  YrSold  SaleType  SaleCondition  SalePrice
Id
1        0      2    2008        WD         Normal     208500
2        0      5    2007        WD         Normal     181500
3        0      9    2008        WD         Normal     223500
4        0      2    2006        WD        Abnorml     140000
5        0     12    2008        WD         Normal     250000

[5 rows x 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).

[3]:
```
ames.dropna(axis=1, inplace=True)
```

[4]:
```
ames.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1460 entries, 1 to 1460
Data columns (total 61 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   MSSubClass      1460 non-null   int64
 1   MSZoning        1460 non-null   object
 2   LotArea         1460 non-null   int64
 3   Street          1460 non-null   object
 4   LotShape        1460 non-null   object
 5   LandContour     1460 non-null   object
 6   Utilities       1460 non-null   object
 7   LotConfig       1460 non-null   object
 8   LandSlope       1460 non-null   object
 9   Neighborhood    1460 non-null   object
 10  Condition1      1460 non-null   object
 11  Condition2      1460 non-null   object
 12  BldgType        1460 non-null   object
 13  HouseStyle      1460 non-null   object
 14  OverallQual     1460 non-null   int64
 15  OverallCond     1460 non-null   int64
 16  YearBuilt       1460 non-null   int64
 17  YearRemodAdd    1460 non-null   int64
 18  RoofStyle       1460 non-null   object
```

```
 19  RoofMatl       1460 non-null   object
 20  Exterior1st    1460 non-null   object
 21  Exterior2nd    1460 non-null   object
 22  ExterQual      1460 non-null   object
 23  ExterCond      1460 non-null   object
 24  Foundation     1460 non-null   object
 25  BsmtFinSF1     1460 non-null   int64
 26  BsmtFinSF2     1460 non-null   int64
 27  BsmtUnfSF      1460 non-null   int64
 28  TotalBsmtSF    1460 non-null   int64
 29  Heating        1460 non-null   object
 30  HeatingQC      1460 non-null   object
 31  CentralAir     1460 non-null   object
 32  1stFlrSF       1460 non-null   int64
 33  2ndFlrSF       1460 non-null   int64
 34  LowQualFinSF   1460 non-null   int64
 35  GrLivArea      1460 non-null   int64
 36  BsmtFullBath   1460 non-null   int64
 37  BsmtHalfBath   1460 non-null   int64
 38  FullBath       1460 non-null   int64
 39  HalfBath       1460 non-null   int64
 40  BedroomAbvGr   1460 non-null   int64
 41  KitchenAbvGr   1460 non-null   int64
 42  KitchenQual    1460 non-null   object
 43  TotRmsAbvGrd   1460 non-null   int64
 44  Functional     1460 non-null   object
 45  Fireplaces     1460 non-null   int64
 46  GarageCars     1460 non-null   int64
 47  GarageArea     1460 non-null   int64
 48  PavedDrive     1460 non-null   object
 49  WoodDeckSF     1460 non-null   int64
 50  OpenPorchSF    1460 non-null   int64
 51  EnclosedPorch  1460 non-null   int64
 52  3SsnPorch      1460 non-null   int64
 53  ScreenPorch    1460 non-null   int64
 54  PoolArea       1460 non-null   int64
 55  MiscVal        1460 non-null   int64
 56  MoSold         1460 non-null   int64
 57  YrSold         1460 non-null   int64
 58  SaleType       1460 non-null   object
 59  SaleCondition  1460 non-null   object
 60  SalePrice      1460 non-null   int64
dtypes: int64(34), object(27)
memory usage: 707.2+ KB
```

1.4 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.

1.4.1 Continuous Numeric Predictor

[5]:
```
ames = ames.drop(columns=['BsmtFinSF2', 'LowQualFinSF', 'EnclosedPorch',␣
↪'3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal'])
```

[6]:
```
# Your code here - continuous numeric predictor (float)
continuous_Numeric = ames.select_dtypes("number")
```

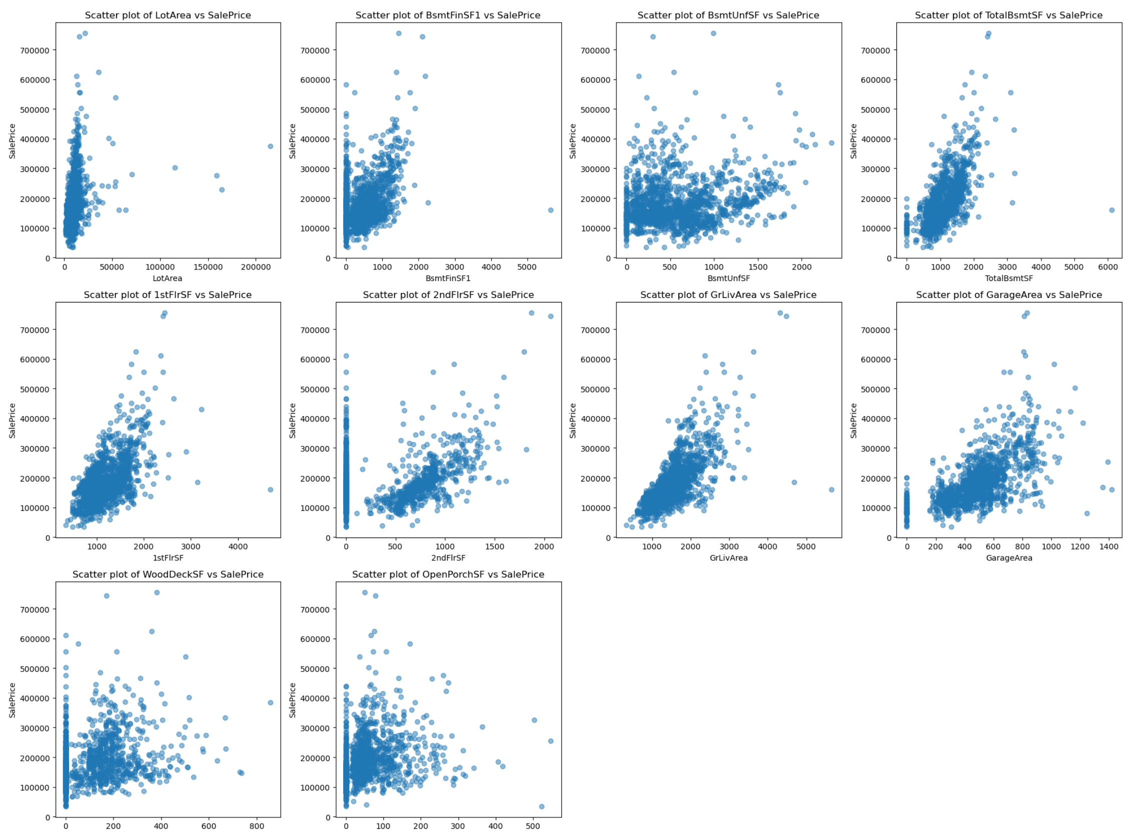
[7]:
```
continuous_Numeric.columns
```

[7]:
```
Index(['MSSubClass', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt',
       'YearRemodAdd', 'BsmtFinSF1', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF',
       '2ndFlrSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd',
       'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
```

[8]:
```
continuous_columns = ['LotArea', 'BsmtFinSF1', 'BsmtUnfSF', 'TotalBsmtSF',␣
↪'1stFlrSF', '2ndFlrSF', 'GrLivArea',
                      'GarageArea', 'WoodDeckSF', 'OpenPorchSF']
```

[9]:
```
def visualize_columns(df, columns, target_column, plots_per_row=4):
    num_columns = len(columns)
    # Calculate the number of rows needed
    num_rows = (num_columns + plots_per_row - 1) // plots_per_row

    fig, axes = plt.subplots(num_rows, plots_per_row, figsize=(plots_per_row *␣
↪5, num_rows * 5))
    # Flatten the axes array for easy iteration
    axes = axes.flatten()

    for i, col in enumerate(columns):
```

```
        axes[i].scatter(df[col], df[target_column], alpha=0.5)
        axes[i].set_title(f'Scatter plot of {col} vs {target_column}')
        axes[i].set_xlabel(col)
        axes[i].set_ylabel(target_column)

    # Hide any unused subplots
    for j in range(i + 1, len(axes)):
        fig.delaxes(axes[j])

    plt.tight_layout()
    plt.show()

# Visualize continous columns
visualize_columns(ames, continuous_columns, 'SalePrice')
```

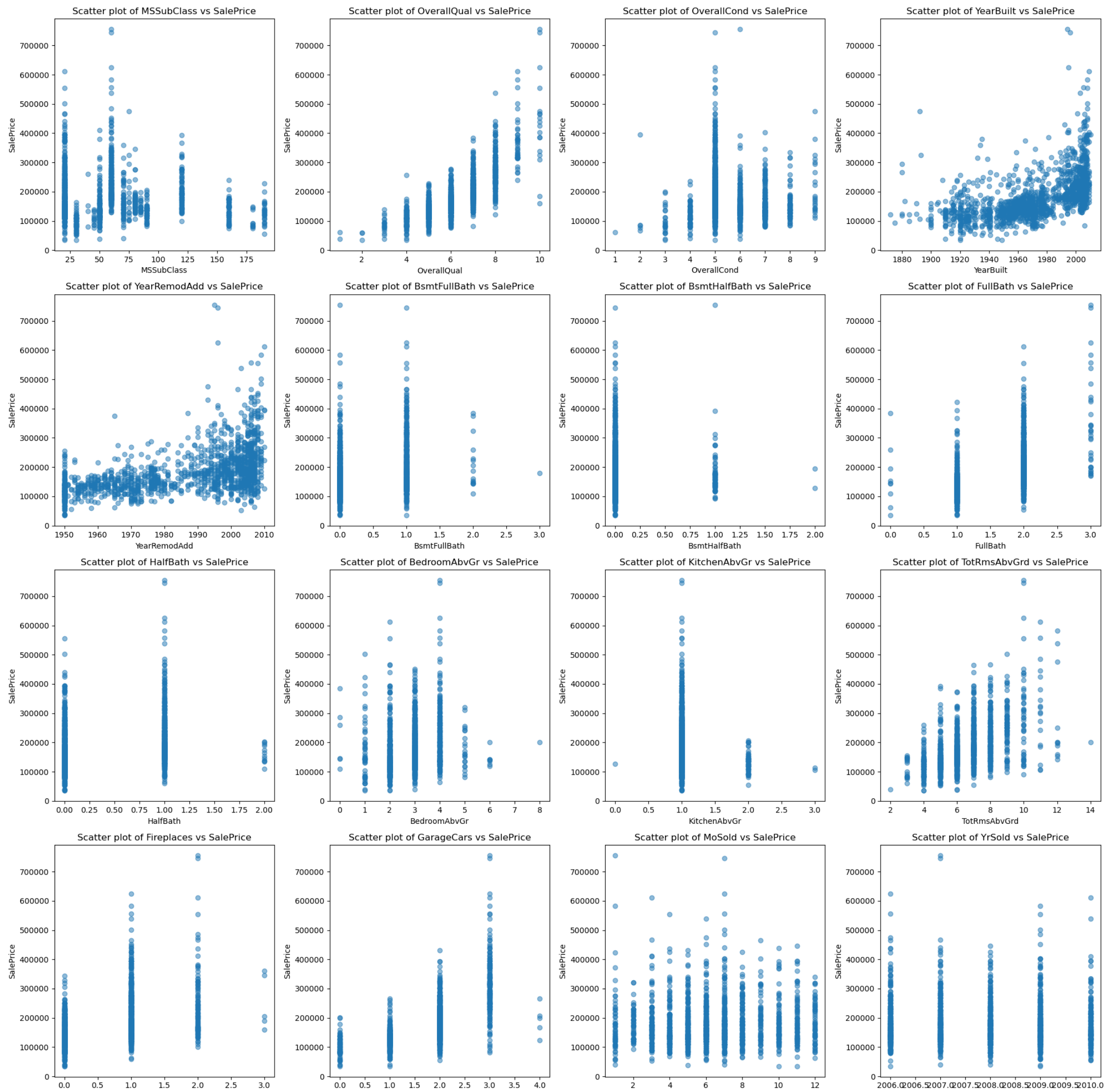


1.4.2 Discrete Numeric Predictor

[10]:
```
# Your code here - discrete numeric predictor
drop_col = ['SalePrice']
drop_col.extend(continuous_columns)
```

```python
discrete_Numeric = ames.select_dtypes("number").copy()
discrete_Numeric = discrete_Numeric.drop(columns=drop_col)
discrete_Numeric = list(discrete_Numeric.columns)
```

[11]:
```python
visualize_columns(ames, discrete_Numeric, 'SalePrice')
```



1.4.3 String Categorical Predictor

[12]:
```python
# Selecting all string columns
string_Categorical = ames.select_dtypes("object")
string_Categorical.head()
```

[12]:

```
   MSZoning Street LotShape LandContour Utilities LotConfig LandSlope  \
Id
1        RL   Pave      Reg         Lvl    AllPub    Inside       Gtl
2        RL   Pave      Reg         Lvl    AllPub       FR2       Gtl
3        RL   Pave      IR1         Lvl    AllPub    Inside       Gtl
4        RL   Pave      IR1         Lvl    AllPub    Corner       Gtl
5        RL   Pave      IR1         Lvl    AllPub       FR2       Gtl

   Neighborhood Condition1 Condition2  … ExterCond Foundation Heating  \
Id                                     …
1       CollgCr       Norm       Norm  …        TA      PConc    GasA
2       Veenker      Feedr       Norm  …        TA     CBlock    GasA
3       CollgCr       Norm       Norm  …        TA      PConc    GasA
4       Crawfor       Norm       Norm  …        TA     BrkTil    GasA
5       NoRidge       Norm       Norm  …        TA      PConc    GasA

   HeatingQC CentralAir KitchenQual Functional PavedDrive SaleType  \
Id
1         Ex          Y          Gd        Typ          Y       WD
2         Ex          Y          TA        Typ          Y       WD
3         Ex          Y          Gd        Typ          Y       WD
4         Gd          Y          Gd        Typ          Y       WD
5         Ex          Y          Gd        Typ          Y       WD

   SaleCondition
Id
1         Normal
2         Normal
3         Normal
4        Abnorml
5         Normal

[5 rows x 27 columns]
```

[13]:

```
string_Categorical.nunique().sort_values(ascending=False)
```

[13]:

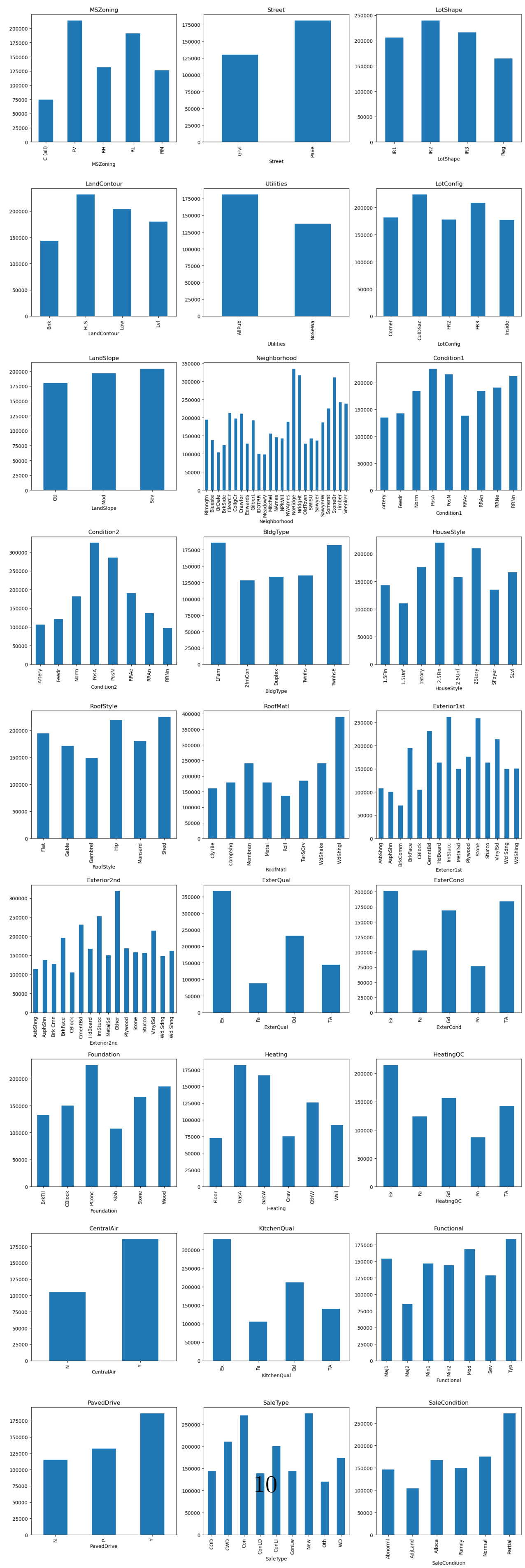
```
Neighborhood     25
Exterior2nd      16
Exterior1st      15
Condition1        9
SaleType          9
RoofMatl          8
HouseStyle        8
Condition2        8
Functional        7
Foundation        6
RoofStyle         6
```

```
Heating           6
SaleCondition     6
HeatingQC         5
ExterCond         5
LotConfig         5
MSZoning          5
BldgType          5
LotShape          4
ExterQual         4
KitchenQual       4
LandContour       4
LandSlope         3
PavedDrive        3
Street            2
Utilities         2
CentralAir        2
dtype: int64
```

[14]:

```python
# Suppress the FutureWarning
warnings.filterwarnings('ignore', category=FutureWarning)

stringList = list(string_Categorical.columns)
# Ensure SalePrice is numeric
ames['SalePrice'] = pd.to_numeric(ames['SalePrice'], errors='coerce')

# Determine the number of rows and columns for the subplots
num_cols = 3
num_rows = (len(stringList) + num_cols - 1) // num_cols

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, num_rows * 5))

# Flatten the axes array for easy iteration
axes = axes.flatten()

for i, col in enumerate(stringList):
    # Ensure the column is treated as a category
    ames[col] = ames[col].astype('category')

    # Group by the column and calculate the mean SalePrice
    mean_prices = ames.groupby(col)['SalePrice'].mean().sort_index()

    # Plot the bar chart
    mean_prices.plot.bar(ax=axes[i])
    axes[i].set_title(col)

# Hide any unused subplots
for j in range(i + 1, len(axes)):
```

```
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show();

# Re-enable warnings if needed
warnings.filterwarnings('default', category=FutureWarning)
```


MSZoning
C (all) FV RH RL RM
Street
Grvl Pave
LotShape
IR1 IR2 IR3 Reg
LandContour
Bnk HLS Low Lvl
Utilities
AllPub NoSeWa
LotConfig
Corner CulDSac FR2 FR3 Inside
LandSlope
Gtl Mod Sev
Neighborhood
Condition1
Artery Feedr Norm PosA PosN RRAe RRAn RRNe RRNn
Condition2
Artery Feedr Norm PosA PosN RRAe RRAn RRNn
BldgType
1Fam 2fmCon Duplex Twnhs TwnhsE
HouseStyle
1.5Fin 1.5Unf 1Story 2.5Fin 2.5Unf 2Story SFoyer SLvl
RoofStyle
Flat Gable Gambrel Hip Mansard Shed
RoofMatl
ClyTile CompShg Membran Metal Roll Tar&Grv WdShake WdShngl
Exterior1st
Exterior2nd
ExterQual
Ex Fa Gd TA
ExterCond
Ex Fa Gd Po TA
Foundation
BrkTil CBlock PConc Slab Stone Wood
Heating
Floor GasA GasW Grav OthW Wall
HeatingQC
Ex Fa Gd Po TA
CentralAir
N Y
KitchenQual
Ex Fa Gd TA
Functional
Maj1 Maj2 Min1 Min2 Mod Sev Typ
PavedDrive
N P Y
SaleType
COD CWD Con ConLD ConLI ConLw New Oth WD
SaleCondition
Abnorml AdjLand Alloca Family Normal Partial

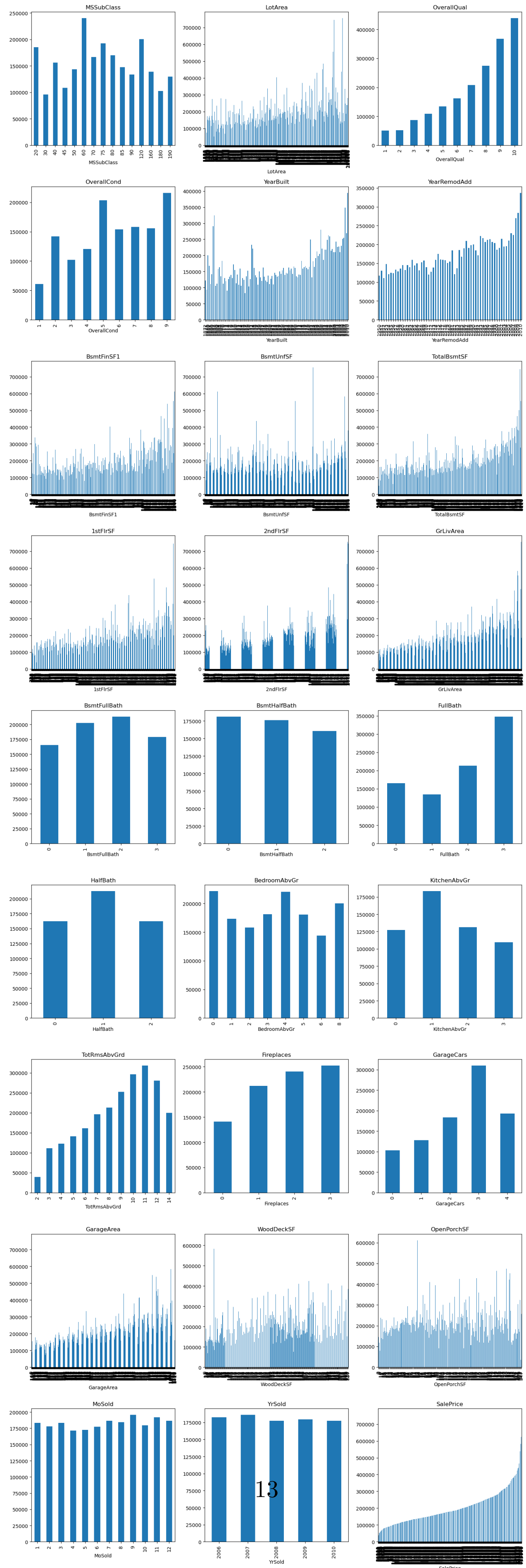
1.4.4 Discrete Categorical Predictor

[15]:
```
# Your code here - discrete categorical predictor
discrete_Categorical = ames.select_dtypes("number")
discrete_Categorical.nunique().sort_values()
```

[15]:
```
BsmtHalfBath       3
HalfBath           3
BsmtFullBath       4
FullBath           4
Fireplaces         4
KitchenAbvGr       4
YrSold             5
GarageCars         5
BedroomAbvGr       8
OverallCond        9
OverallQual       10
TotRmsAbvGrd      12
MoSold            12
MSSubClass        15
YearRemodAdd      61
YearBuilt        112
OpenPorchSF      202
WoodDeckSF       274
2ndFlrSF         417
GarageArea       441
BsmtFinSF1       637
SalePrice        663
TotalBsmtSF      721
1stFlrSF         753
BsmtUnfSF        780
GrLivArea        861
LotArea         1073
dtype: int64
```

[16]:
```
# Ensure SalePrice is numeric
ames['SalePrice'] = pd.to_numeric(ames['SalePrice'], errors='coerce')

discreteList = list(discrete_Categorical.columns)
# Determine the number of rows and columns for the subplots
num_cols = 3
num_rows = (len(discreteList) + num_cols - 1) // num_cols

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, num_rows * 5))
```

```python
# Flatten the axes array for easy iteration
axes = axes.flatten()

for i, col in enumerate(discreteList):
    # Group by the column and calculate the mean SalePrice
    mean_prices = ames.groupby(col)['SalePrice'].mean().sort_index()

    # Plot the bar chart
    mean_prices.plot.bar(ax=axes[i])
    axes[i].set_title(col)

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show();
```


MSSubClass
LotArea
OverallQual
OverallCond
YearBuilt
YearRemodAdd
BsmtFinSF1
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
GrLivArea
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageCars
GarageArea
WoodDeckSF
OpenPorchSF
MoSold
YrSold
SalePrice

1.5 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.

[17]:
```
# Your code here - prepare X and y, including one-hot encoding
# MSSubClass is messiest
y = ames["SalePrice"]
X = ames[["GrLivArea", "OverallQual", "LotShape"]]
X.head()
```

[17]:

Id	GrLivArea	OverallQual	LotShape
1	1710	7	Reg
2	1262	6	Reg
3	1786	7	IR1
4	1717	7	IR1
5	2198	8	IR1

[18]:
```
# Your answer here - which category or categories were dropped?
X = pd.get_dummies(X, columns=["LotShape"], drop_first=True, dtype=int)
X.head()
```

[18]:

Id	GrLivArea	OverallQual	LotShape_IR2	LotShape_IR3	LotShape_Reg
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

[19]:
```
# Your code here - build a regression model and display results
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
```

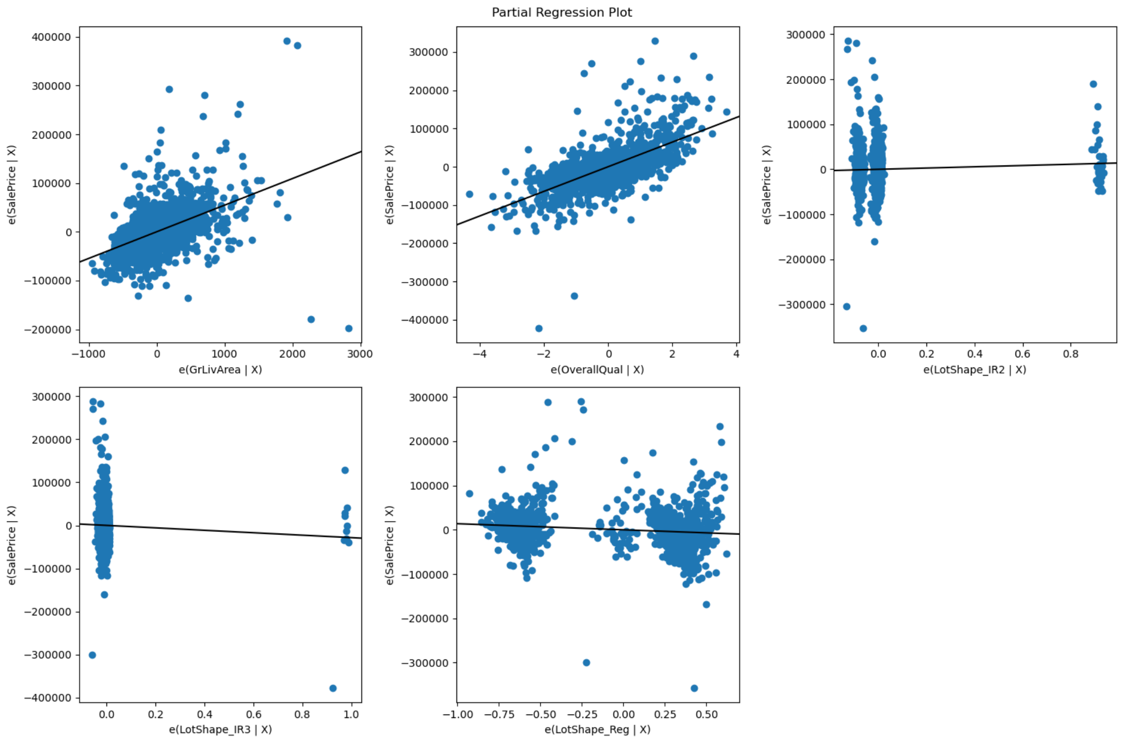
```
Date:                Tue, 06 Aug 2024   Prob (F-statistic):                0.00
Time:                        23:49:51   Log-Likelihood:                 -17607.
No. Observations:                1460   AIC:                          3.523e+04
Df Residuals:                    1454   BIC:                          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.
```

1.6 Step 4: Create Partial Regression Plots for Features

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

[20]:

```python
# Your code here - create partial regression plots
fig = plt.figure(figsize=(15,10))
sm.graphics.plot_partregress_grid(
    results,
    exog_idx=list(X.columns),
    grid=(2,3),
    fig=fig)
plt.tight_layout()
plt.show()
```



1.7 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.

[21]:
```
# Your code here - calculate an error-based metric
from sklearn.metrics import mean_absolute_error

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

[21]: 28396.050798992394

1.8 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?

[22]:
```
# Your answer here
"""
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.
""";
```

1.9 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.

[23]:
```
# Your code here
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LinearRegression
```

[24]:
```
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_output=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.head()
```

[24]:

	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

[25]:
```
X_sklearn_final = pd.concat([X_numeric, X_cat_ohe], axis=1)
X_sklearn_final.head()
```

[25]:

	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

[26]:
```
lr = LinearRegression()
lr.fit(X_sklearn_final, y)
```

[26]: LinearRegression()

[27]:
```
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]
```

[28]:
```
mean_absolute_error(y, lr.predict(X_sklearn_final))
```

[28]: 28396.050798992903

1.10 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.