Module1

April 16, 2022

0.0.1 Grading

The final score that you will receive for your programming assignment is generated in relation to the total points set in your programming assignment item—not the total point value in the nbgrader notebook. When calculating the final score shown to learners, the programming assignment takes the percentage of earned points vs. the total points provided by nbgrader and returns a score matching the equivalent percentage of the point value for the programming assignment. **DO NOT CHANGE VARIABLE OR METHOD SIGNATURES** The autograder will not work properly if your change the variable or method signatures.

0.0.2 Validate Button

Please note that this assignment uses nbgrader to facilitate grading. You will see a **validate button** at the top of your Jupyter notebook. If you hit this button, it will run tests cases for the lab that aren't hidden. It is good to use the validate button before submitting the lab. Do know that the labs in the course contain hidden test cases. The validate button will not let you know whether these test cases pass. After submitting your lab, you can see more information about these hidden test cases in the Grader Output. ***Cells with longer execution times will cause the validate button to time out and freeze. Please know that if you run into Validate time-outs, it will not affect the final submission grading.***

1 Part 1. Data cleaning and Exploratory Data Analysis (EDA)

This part will practice data cleaning and Exploratory Data Analysis (EDA) using a house price dataset and mpg dataset. The first dataset is from a Kaggle competition (https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview), where the task is to predict a house sale price given house features.

[282]:
```
!ls
```

```
data  Module1.ipynb
```

[283]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import math
```

1.1 1. Import data and visually inspect the table [9 pts]

1.1.1 1a) Data import and basic inspection. [5 pts]

We can import the csv data using `pd.read_csv()` function. We can use `df.head()` and `df.tail()` to show the first and last 5 entries. `df.iloc[[3,5,7]]` shows the entries corresponding to the index 3,5,7. What is the maximum value of the feature `MSSubClass` among the last 10 entries? Update the value of `maxval` to the correct integer value.

[284]:
```
df = pd.read_csv('data/house_data.csv') #it is the same data as the kaggle␣
↪competition's train.csv.
# your code here

# uncomment maxval and update the correct integer value
maxval = max(df['MSSubClass'].tail(10))
maxval
```

[284]:
```
180
```

[285]:
```
# this cell tests that you correctly updated maxval
```

1.1.2 1b) df.info() gives the overview of the data frame. Inspect the data using df.info() and answer below questions. [4 pts]

1b-i) Which column is the target?

1b-ii) How many features are in the data? Exclude the target. (Id is not a useful feature, but let's still include)

1b-iii) How many observations (samples) are in the data?

1b-iv) How many features have null values based on the data overview?

[286]:
```
## Fixed 1b2

# your code here
df.info()

# uncomment and update to the correct string value
# copy directly from the uneditd df column name (e.g., 'LandContour')
ANS_1b1 = 'SalePrice'
# uncomment and update to the correct integer value
ANS_1b2 = 80
# uncomment and update to the correct integer value
ANS_1b3 = 1460
```

```python
# uncomment and update to the correct integer value
ANS_1b4 = sum(df.isna().any())
ANS_1b4
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
 24  Exterior2nd    1460 non-null   object
 25  MasVnrType     1452 non-null   object
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object
 28  ExterCond      1460 non-null   object
 29  Foundation     1460 non-null   object
 30  BsmtQual       1423 non-null   object
 31  BsmtCond       1423 non-null   object
 32  BsmtExposure   1422 non-null   object
 33  BsmtFinType1   1423 non-null   object
 34  BsmtFinSF1     1460 non-null   int64
 35  BsmtFinType2   1422 non-null   object
 36  BsmtFinSF2     1460 non-null   int64
 37  BsmtUnfSF      1460 non-null   int64
```

```
 38  TotalBsmtSF    1460 non-null   int64
 39  Heating        1460 non-null   object
 40  HeatingQC      1460 non-null   object
 41  CentralAir     1460 non-null   object
 42  Electrical     1459 non-null   object
 43  1stFlrSF       1460 non-null   int64
 44  2ndFlrSF       1460 non-null   int64
 45  LowQualFinSF   1460 non-null   int64
 46  GrLivArea      1460 non-null   int64
 47  BsmtFullBath   1460 non-null   int64
 48  BsmtHalfBath   1460 non-null   int64
 49  FullBath       1460 non-null   int64
 50  HalfBath       1460 non-null   int64
 51  BedroomAbvGr   1460 non-null   int64
 52  KitchenAbvGr   1460 non-null   int64
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int64
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

[286]: 19

[287]:
```
# this cell will test your solutions to the four questions above
```

1.2 2. Inspect Null values [16 pts]

The empty values in the data are called null values. Null values can take different forms. Have a look at below example. `np.nan` and `None` are native null values in python. They get displayed differently in the pandas dataframe (`pd.DataFrame`) though. But there are other data types such as empty list, empty dictionary, etc and string values that literally says "null" or that are empty spaces. Depending on how messy the data is, sometimes the table may have null values of one or more kinds, and those can be cleaned manually or automatically if you can write a code to include all possible cases which meanings are null values.

[288]:
```
a = [np.nan, None, [], {}, 'NaN', 'Null','NULL','None','NA','?','-', '.','', '␣
↪', '    ']
nulldemo = pd.DataFrame(a)
nulldemo
```

[288]:
```
       0
0    NaN
1   None
2     []
3     {}
4    NaN
5   Null
6   NULL
7   None
8     NA
9      ?
10     -
11     .
12
13
14
```

`.isnull()` method applied to pandas dataframe or series can detect null values. `.dropna()` method in pandas will detect null values and can be specified to drop either rows or columns that contain null values. Below shows that `.isnull()` only detects the python-native null values and cannot detect other forms (string value) of variables that meant null.

[289]:
```
nulldemo.isnull()
```

[289]:
```
       0
0   True
1   True
2  False
3  False
```

```
4    False
5    False
6    False
7    False
8    False
9    False
10   False
11   False
12   False
13   False
14   False
```

Also, sometimes the python-native null values can have an odd data type such as numpy float.

[290]:
```
print(df['MasVnrArea'].iloc[234], df['MasVnrArea'].iloc[234].dtype, type(df['MasVnrArea'].iloc[234]))
print(df['MasVnrArea'].isnull().iloc[234])
print(np.isnan(df['MasVnrArea'].iloc[234]))
print(math.isnan(df['MasVnrArea'].iloc[234]))
print(df['MasVnrArea'].iloc[234]==np.nan)
print(df['MasVnrArea'].iloc[234]==np.float64(np.nan))
```

```
nan float64 <class 'numpy.float64'>
True
True
True
False
False
```

np.isnan() and math.isnan() can detect the nan values with numpy float type, but they will cause errors with native `None` or a string value. Uncomment one of below (one at a time) and run. You'll see error messages.

[291]:
```
# your code here



# print(np.isnan(None))
# print(np.isnan('None'))
# print(math.isnan(None))
# print(math.isnan('None'))
```

1.2.1 2a) Check null values type [5 pts]

Let's check if our data has clean null values (one kind) or messy null values (multiple different representations). Run the codes below and visually inspect the printed results. Which column has string-typed null/none values and how many elements are string-typed null/none values?

[292]:
```python
# prints number of null values detected by .isnull() and string none
for c in df.columns:
    string_null = np.array([x in a[2:] for x in df[c]])
    print(c, df[c].isnull().sum(), string_null.sum())
```

```
Id 0 0
MSSubClass 0 0
MSZoning 0 0
LotFrontage 259 0
LotArea 0 0
Street 0 0
Alley 1369 0
LotShape 0 0
LandContour 0 0
Utilities 0 0
LotConfig 0 0
LandSlope 0 0
Neighborhood 0 0
Condition1 0 0
Condition2 0 0
BldgType 0 0
HouseStyle 0 0
OverallQual 0 0
OverallCond 0 0
YearBuilt 0 0
YearRemodAdd 0 0
RoofStyle 0 0
RoofMatl 0 0
Exterior1st 0 0
Exterior2nd 0 0
MasVnrType 8 864
MasVnrArea 8 0
ExterQual 0 0
ExterCond 0 0
Foundation 0 0
BsmtQual 37 0
BsmtCond 37 0
BsmtExposure 38 0
BsmtFinType1 37 0
BsmtFinSF1 0 0
BsmtFinType2 38 0
BsmtFinSF2 0 0
BsmtUnfSF 0 0
TotalBsmtSF 0 0
Heating 0 0
HeatingQC 0 0
CentralAir 0 0
```

```
Electrical 1 0
1stFlrSF 0 0
2ndFlrSF 0 0
LowQualFinSF 0 0
GrLivArea 0 0
BsmtFullBath 0 0
BsmtHalfBath 0 0
FullBath 0 0
HalfBath 0 0
BedroomAbvGr 0 0
KitchenAbvGr 0 0
KitchenQual 0 0
TotRmsAbvGrd 0 0
Functional 0 0
Fireplaces 0 0
FireplaceQu 690 0
GarageType 81 0
GarageYrBlt 81 0
GarageFinish 81 0
GarageCars 0 0
GarageArea 0 0
GarageQual 81 0
GarageCond 81 0
PavedDrive 0 0
WoodDeckSF 0 0
OpenPorchSF 0 0
EnclosedPorch 0 0
3SsnPorch 0 0
ScreenPorch 0 0
PoolArea 0 0
PoolQC 1453 0
Fence 1179 0
MiscFeature 1406 0
MiscVal 0 0
MoSold 0 0
YrSold 0 0
SaleType 0 0
SaleCondition 0 0
SalePrice 0 0
```

Which column has string-typed null/none values?

[293]:
```python
# your code here

# uncomment and update to the correct string value
col = 'MasVnrType'
```

How many elements are string-typed null/none values?

[294]:
```
# your code here

# uncomment and update to the correct string value
string_null_count = 864
```

[295]:
```
# this cell will test your answer about the column with string-typed null/none␣
↪values
# and the number of string-typed null/none values
```

1.2.2 2b) Inspect observations (rows) with null values. How many observations have at least one missing value? [5 pts]

[296]:
```
# your code here

# uncomment and update to the correct integer value
rows_with_nulls = df.isnull().any(axis=1).sum()
rows_with_nulls
```

[296]:
```
1460
```

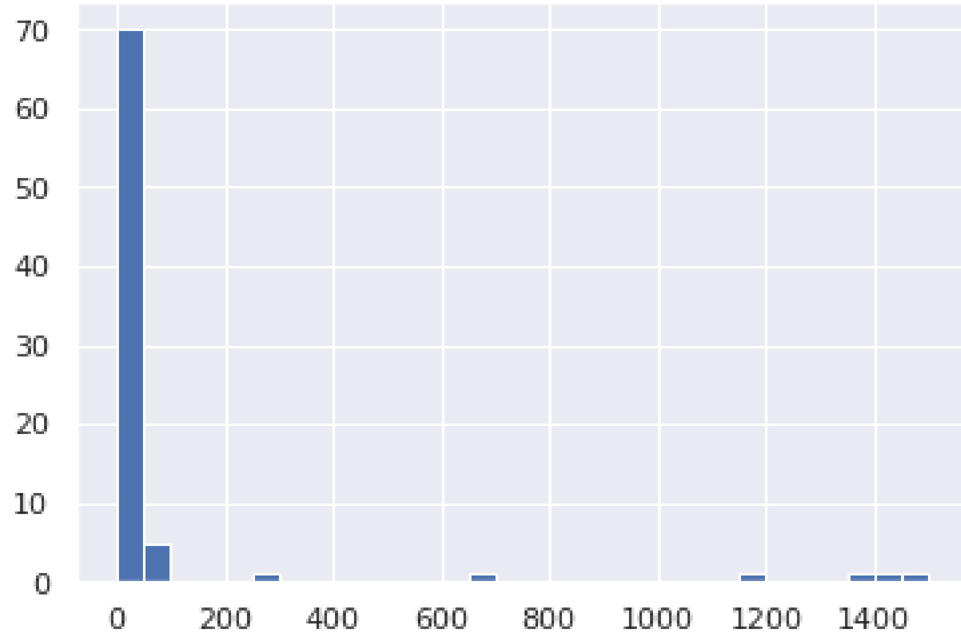
[297]:
```
# this cell will test your answer about the number of rows with null values
```

1.2.3 2c) Make a histogram of null counts [6 pts]

[298]:
```
# your code here



# Please uncomment and update
# do not change the names of the variables from null_counts and histogram
null_counts=pd.Series([])
histogram = None # replace the histogram to be the plt.hist() object.

null_counts = df.isnull().sum(axis=0)
histogram = plt.hist(null_counts, bins=np.arange(0,1550, 50))
histogram
# Hint: Use .isnull() and sum over True values on columns.
# You can make it as short as 2-3 lines of code
```

[298]:
```
(array([70.,  5.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,
         0.,  1.,  1.,  1.]),
 array([   0,   50,  100,  150,  200,  250,  300,  350,  400,  450,  500,
         550,  600,  650,  700,  750,  800,  850,  900,  950, 1000, 1050,
        1100, 1150, 1200, 1250, 1300, 1350, 1400, 1450, 1500]),
```

```
<a list of 30 Patch objects>)
```



[299]:
```python
type(null_counts)
```

[299]:
```
pandas.core.series.Series
```

[300]:
```python
# hidden test 1; tests null_counts
```

[301]:
```python
# hidden test 2; tests histogram
```

1.3 3. Imputing missing values [33 pts]

In this part, we will decide methods to clean the data with missing values.

Complete case analysis (CCA) is to drop any observations (rows) that have null values. It is suitable if the number of observations with null values are very small (say, less than 5%) compared to the total number of observations.

If the data has a large number of features (columns) and the model(s) does not need that many features (some models work better with less number of features), we can consider dropping features that have many missing values. Before dropping features, it is generally a good idea checking whether the feature with missing values is important feature or not (which may need the analyst's judgement). If the feature is very important for the prediction task (for example, a house size when predicting house price) but has a large amount of missing values, we cannot simply drop the feature, or in a rare case, it could mean that the data is not suitable for the analysis. One will have to work with only the observations that has values on that feature given the number of observations is sufficient, or collect more data. If we know that those features are not very important and have a large number of missing values, we can drop the features. As a rule of thumb, features with missing values more than either 5% or 10% can be dropped.

1.3.1 3a) Is the data suitable for complete case analysis or not? [5 pts]

[302]:
```
# your code here



# uncomment and update to string 'no' or 'yes'
suitable_cca = 'no'
```

[303]:
```
# tests solution for whether data is suitable for CCA
```

1.3.2 3b) Dropping feature columns [20 pts]

Let's assume we want to keep columns that have null values 5% or less and discard any column that has null values more than 5%. Treat the string type "None" as a category and not null value. #### 3b-i) According to above condition, how many features can be kept and imputed? [5 pts] #### 3b-ii) Which columns have null values 5% or less of total, so we can impute? [5 pts] #### 3b-iii) Which columns have null vaues more than 5% of total, so we should throw? [5 pts]

[304]:
```
# your code here
perc_missing = df.isnull().sum() / len(df)
perc_df = pd.DataFrame({
    'Column_Name' : df.columns,
    'Perc_Missing' : perc_missing
})
# Complete the codes below by uncommenting and changing the values of␣
 ↪features_to_impute and features_to_throw.
# Each should be a list of feature names (e.g. ['LotFrontage','Alley',...]). Do␣
 ↪not change the variable names.
# There are hidden tests which will grade above three questions.
perc_df.sort_values('Perc_Missing', inplace=True, ascending=False)
perc_df.head(20)

features_to_throw = list(perc_df['Column_Name'][:11])
features_to_impute = perc_df['Column_Name'][11:19]
print(len(features_to_impute), features_to_impute)
print(len(features_to_throw), features_to_throw)
```

```
8 BsmtFinType2    BsmtFinType2
BsmtExposure    BsmtExposure
BsmtQual            BsmtQual
BsmtCond            BsmtCond
BsmtFinType1    BsmtFinType1
```

```
MasVnrArea        MasVnrArea
MasVnrType        MasVnrType
Electrical        Electrical
Name: Column_Name, dtype: object
11 ['PoolQC', 'MiscFeature', 'Alley', 'Fence', 'FireplaceQu', 'LotFrontage',
'GarageYrBlt', 'GarageCond', 'GarageType', 'GarageFinish', 'GarageQual']
```

[305]:
```
# Hidden test for 3b-i
```

[306]:
```
# Hidden test for 3b-ii
```

[307]:
```
# Hidden test for 3b-iii
```

1.3.3 3b-iv) Remove the columns according to the above result. Replace the df with the new result. Also remove Id column as it's not a useful feature. [5 pts]

[308]:
```
# your code here

# remove the columns according to the above result, replace df with the new
↪results
# also remove ID column as it's not a useful feature
features_to_throw.append('Id')
df.drop(features_to_throw, inplace=True, axis=1)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 69 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  MasVnrType     1452 non-null   object
23  MasVnrArea     1452 non-null   float64
24  ExterQual      1460 non-null   object
25  ExterCond      1460 non-null   object
26  Foundation     1460 non-null   object
27  BsmtQual       1423 non-null   object
28  BsmtCond       1423 non-null   object
29  BsmtExposure   1422 non-null   object
30  BsmtFinType1   1423 non-null   object
31  BsmtFinSF1     1460 non-null   int64
32  BsmtFinType2   1422 non-null   object
33  BsmtFinSF2     1460 non-null   int64
34  BsmtUnfSF      1460 non-null   int64
35  TotalBsmtSF    1460 non-null   int64
36  Heating        1460 non-null   object
37  HeatingQC      1460 non-null   object
38  CentralAir     1460 non-null   object
39  Electrical     1459 non-null   object
40  1stFlrSF       1460 non-null   int64
41  2ndFlrSF       1460 non-null   int64
42  LowQualFinSF   1460 non-null   int64
43  GrLivArea      1460 non-null   int64
44  BsmtFullBath   1460 non-null   int64
45  BsmtHalfBath   1460 non-null   int64
46  FullBath       1460 non-null   int64
47  HalfBath       1460 non-null   int64
48  BedroomAbvGr   1460 non-null   int64
49  KitchenAbvGr   1460 non-null   int64
50  KitchenQual    1460 non-null   object
51  TotRmsAbvGrd   1460 non-null   int64
52  Functional     1460 non-null   object
53  Fireplaces     1460 non-null   int64
54  GarageCars     1460 non-null   int64
55  GarageArea     1460 non-null   int64
56  PavedDrive     1460 non-null   object
57  WoodDeckSF     1460 non-null   int64
58  OpenPorchSF    1460 non-null   int64
59  EnclosedPorch  1460 non-null   int64
60  3SsnPorch      1460 non-null   int64
61  ScreenPorch    1460 non-null   int64
62  PoolArea       1460 non-null   int64
63  MiscVal        1460 non-null   int64
```

```
 64  MoSold         1460 non-null   int64
 65  YrSold         1460 non-null   int64
 66  SaleType       1460 non-null   object
 67  SaleCondition  1460 non-null   object
 68  SalePrice      1460 non-null   int64
dtypes: float64(1), int64(34), object(34)
memory usage: 787.2+ KB
```

[309]:
```
# tests that you properly updated df
```

1.3.4 3c) Impute missing data [8 pts]

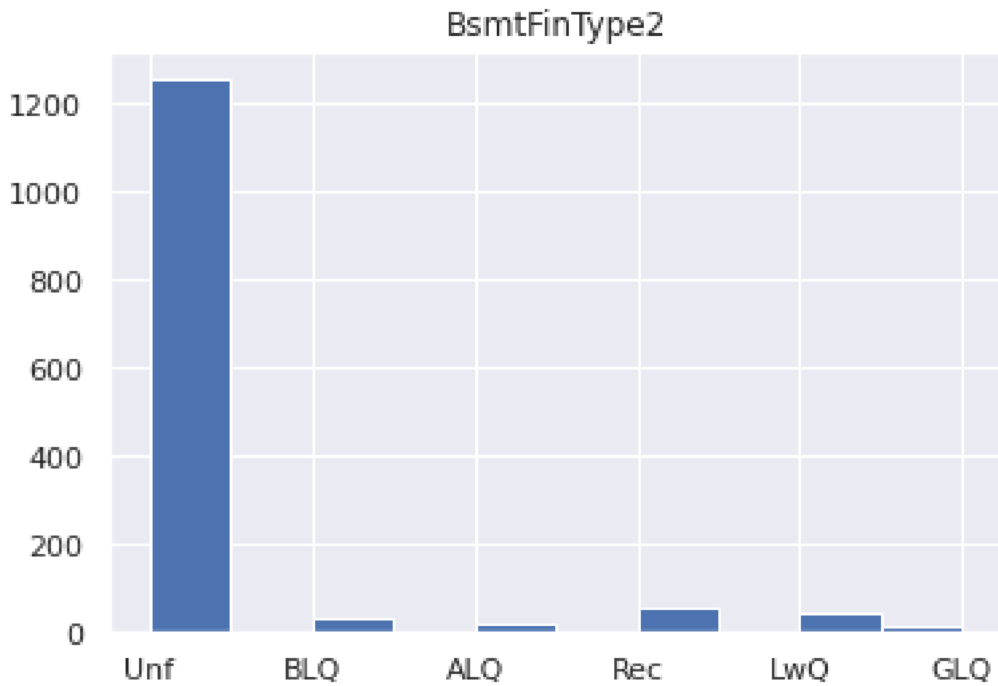
Before imputing columns, we need to think about what methods to use to impute columns. The imputation strategy can be different depending on the variable types and variable value distribution. There are many imputation techniques, but let's use a few simple ones.

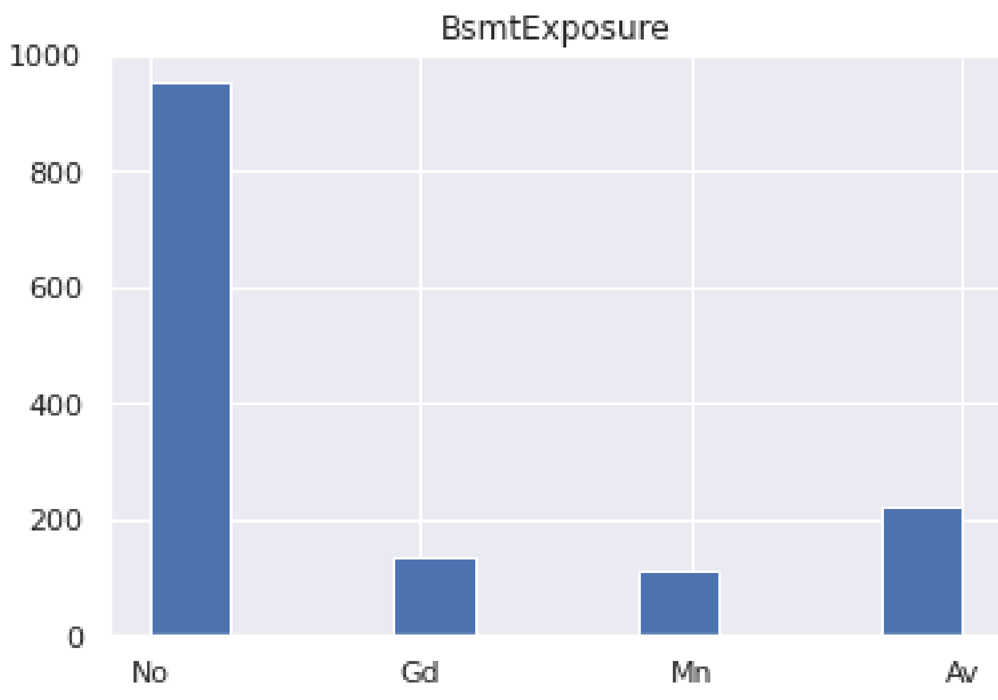
For a numerical variable imputation, we impute mean value if the distribution is symmetric while we use median value to impute when the distribution is skewed. Another method is to assign an arbitrary value that's outside the normal range. Though it can be useful to capture missingness, but it can create outliers. Both mean/median and arbitrary imputation methods are simple to use and suitable when missing values are 5% (no more than 10%) as a rule of thumb. Both methods can distort the original distribution.

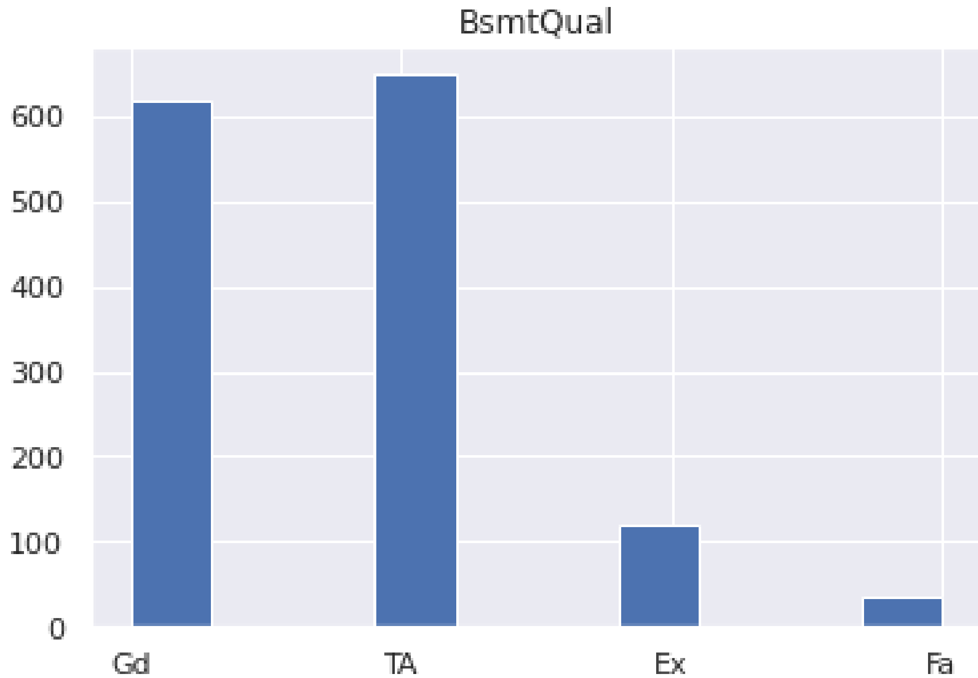
For a categorical variable imputation, we can impute with the most frequent categorical value. It is a simple method but it can distort the original distribution. It is also possible to create a "missing" category to capture missingness. The advantage of using missing category is that it captures missingness but its disadvantage is that it creates another rare category.

Below code shows histograms of feature columns that we can impute.

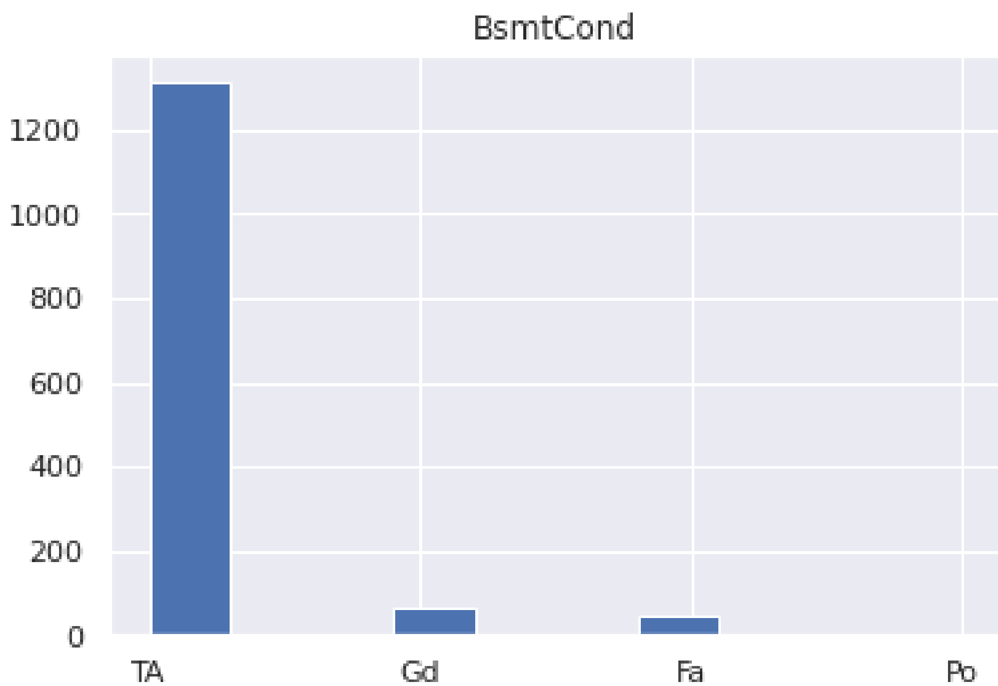
[310]:
```
for c in features_to_impute:
    df[c].hist()
    plt.title(c)
    plt.show()
```


BsmtFinType2
1200
1000
800
600
400
200
0
Unf
BLQ
ALQ
Rec
LwQ
GLQ

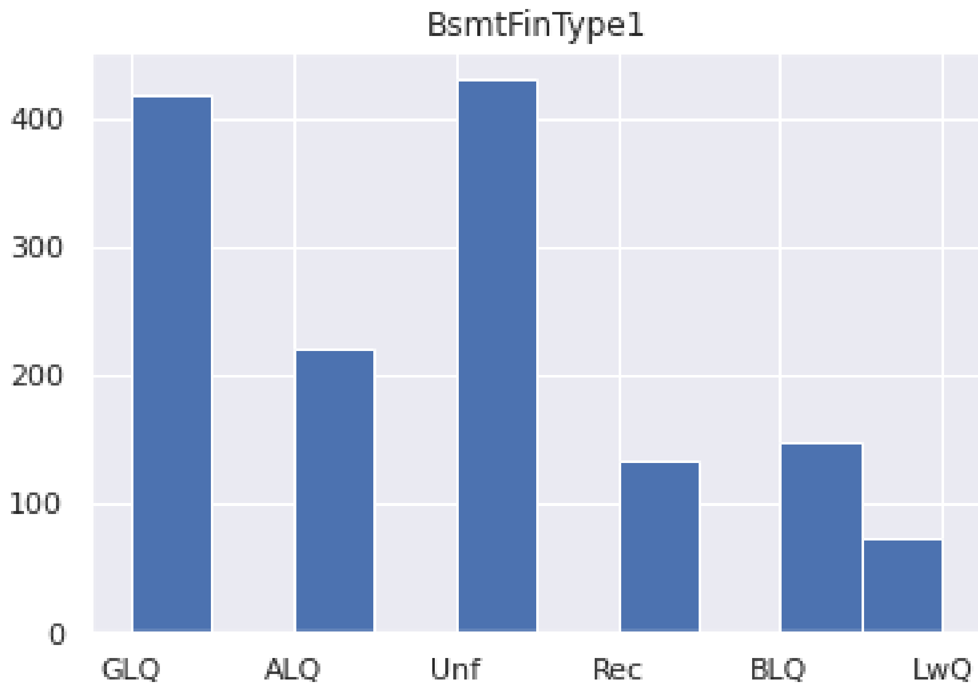

BsmtExposure
1000
800
600
400
200
0
No
Gd
Mn
Av

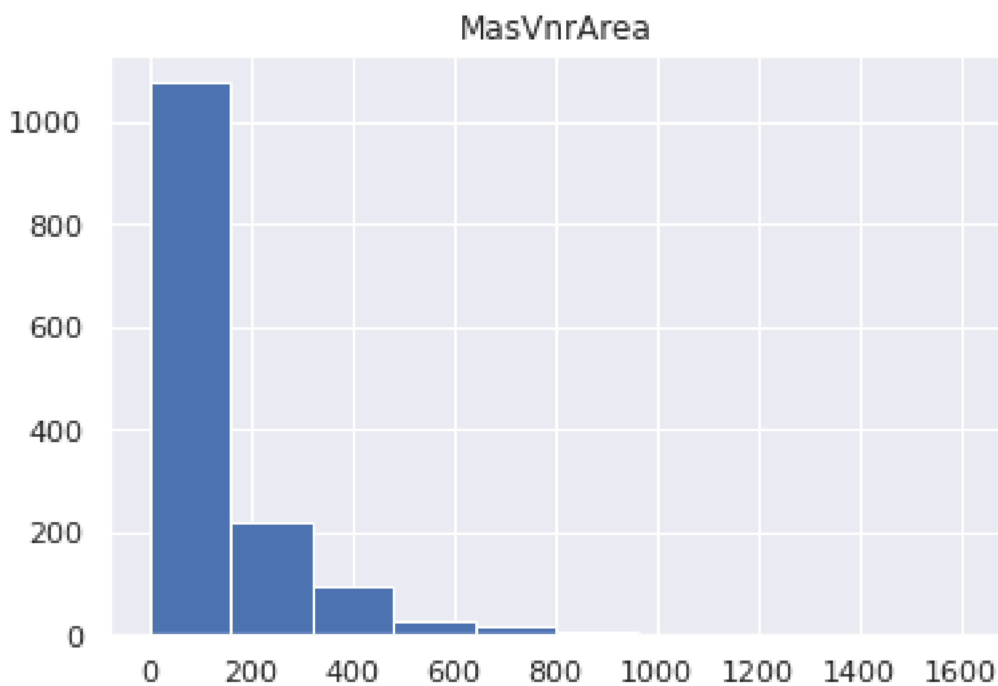


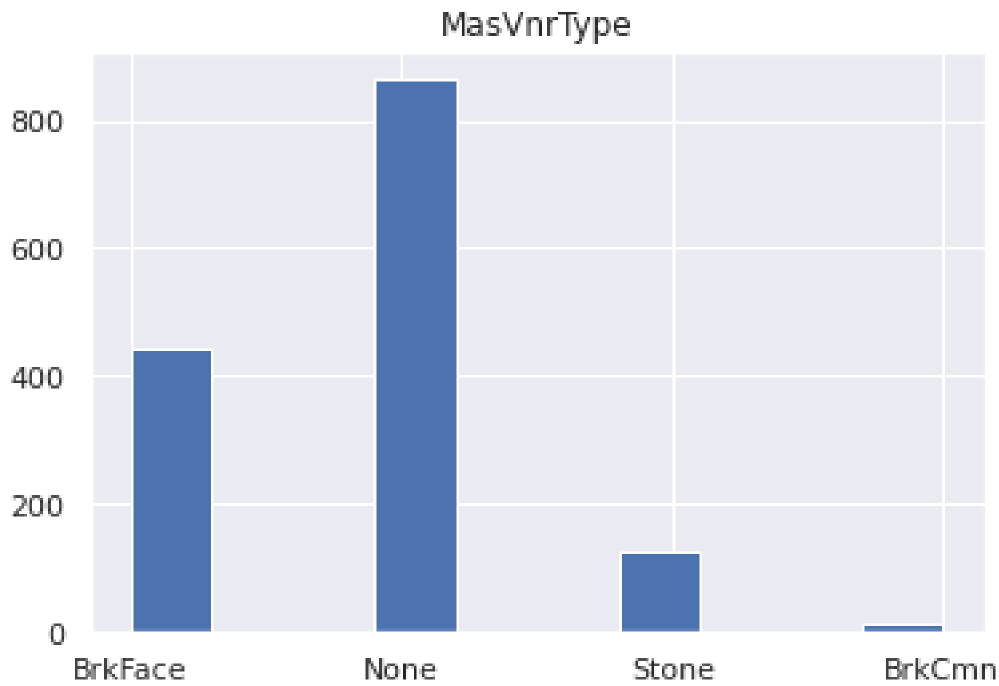
BsmtQual
600
500
400
300
200
100
0
Gd
TA
Ex
Fa

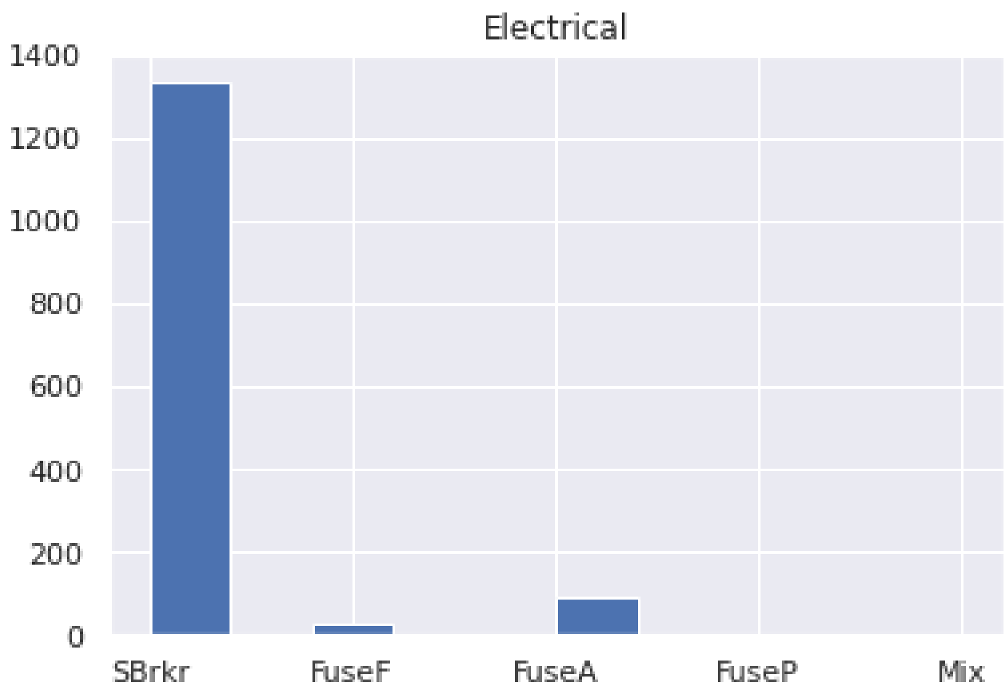


BsmtCond
1200
1000
800
600
400
200
0
TA
Gd
Fa
Po


BsmtFinType1
400
300
200
100
0
GLQ
ALQ
Unf
Rec
BLQ
LwQ


MasVnrArea
1000
800
600
400
200
0
0
200
400
600
800
1000
1200
1400
1600


MasVnrType
800
600
400
200
0
BrkFace
None
Stone
BrkCmn


Electrical
1400
1200
1000
800
600
400
200
0
SBrkr
FuseF
FuseA
FuseP
Mix

1.3.5 3c-i) Impute missing data for features in `features_to_impute`. Choose an appropriate method among mean or median imputation methods for numerical variable(s) and frequentest value imputation for categorical variable(s). [8 pts]

You can inspect variable types by eyes, or use below code as a help. Replace those columns with imputed values. Do not change the column name or the data frame name. Do not add new columns to the data frame.
Hint: You can use .mode() function to find the most frequent value in a Series.
Hint: You may use .fillna() function on each feature Series.

[311]:
```
for c in features_to_impute:
    print(c, len(df[c].unique()), df[c].dtype)
```

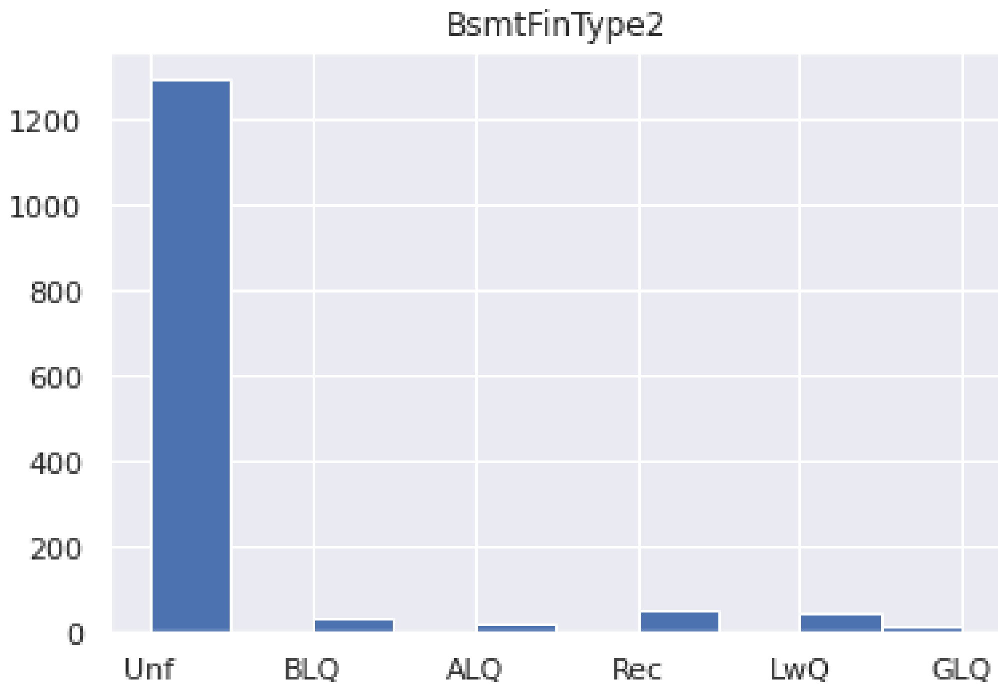
```
BsmtFinType2 7 object
BsmtExposure 5 object
BsmtQual 5 object
BsmtCond 5 object
BsmtFinType1 7 object
MasVnrArea 328 float64
MasVnrType 5 object
Electrical 6 object
```

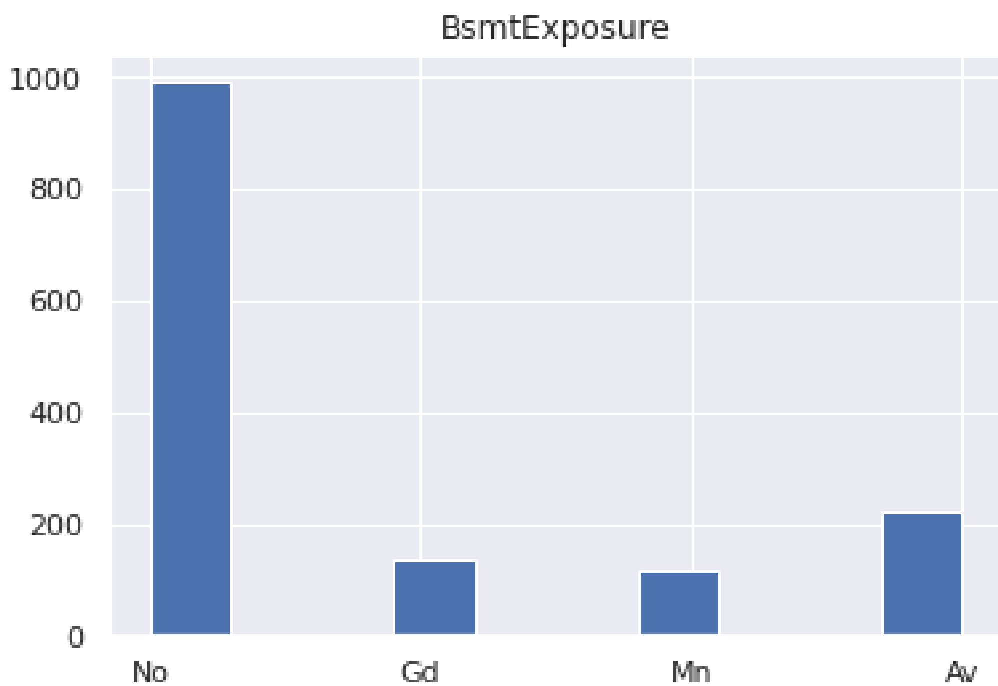
[312]:
```
# your code here
from statistics import mode
print(df['BsmtFinType2'].dtype)
# use this cell for potential debugging
```

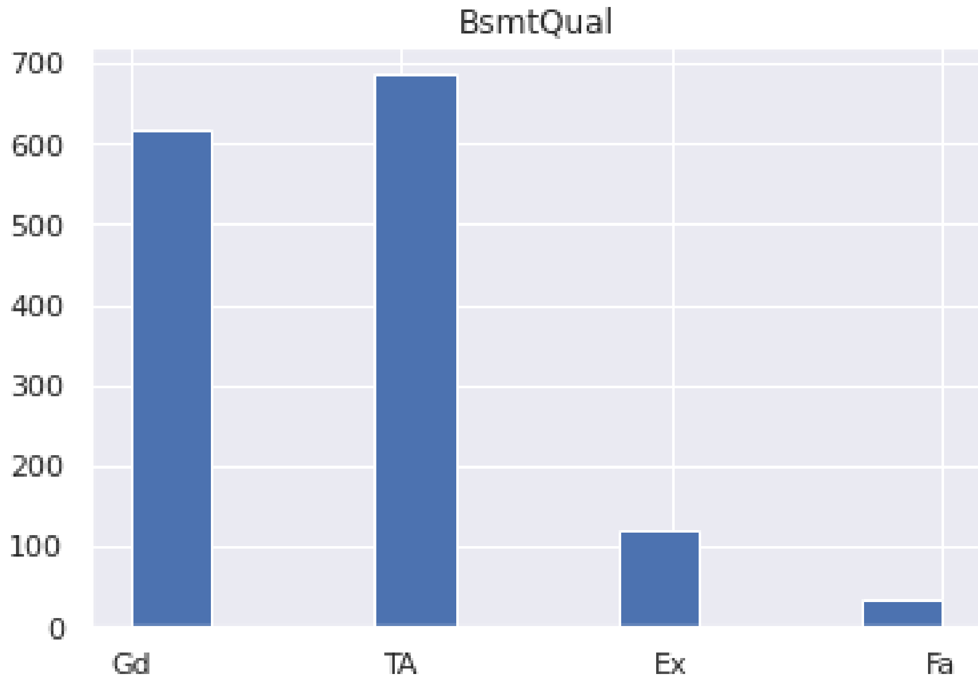
```
object
```

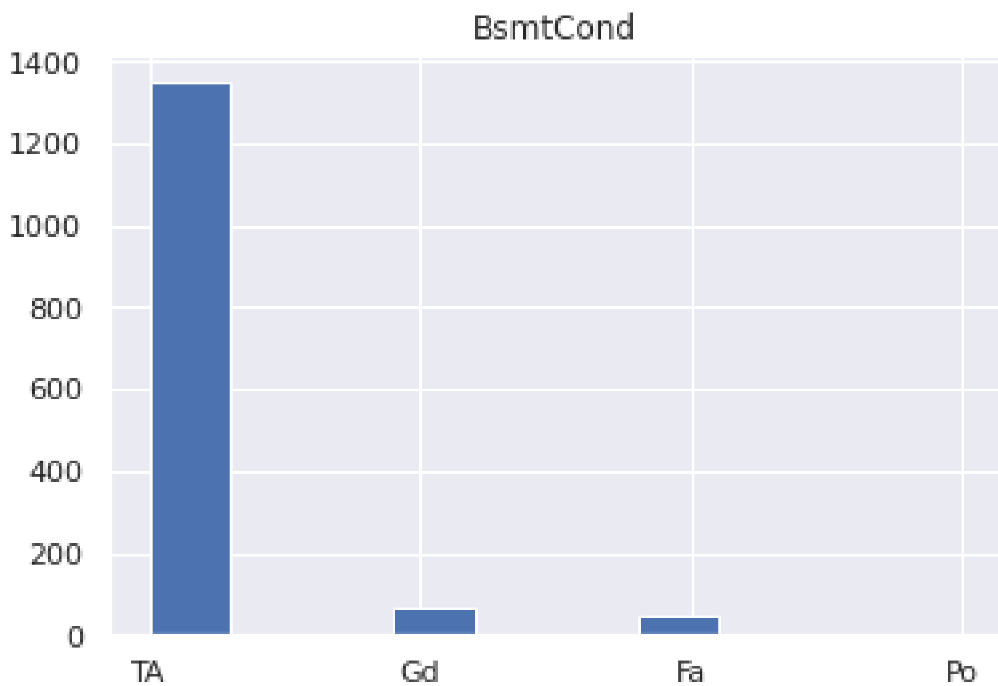
[313]:
```
# impute missing data
# your code here

for col in features_to_impute:
    if df[col].dtype == 'object':
        df[col] = df[col].fillna(mode(df[col]))
    if df[col].dtype == 'float64':
        df[col] = df[col].fillna(df[col].median())
```

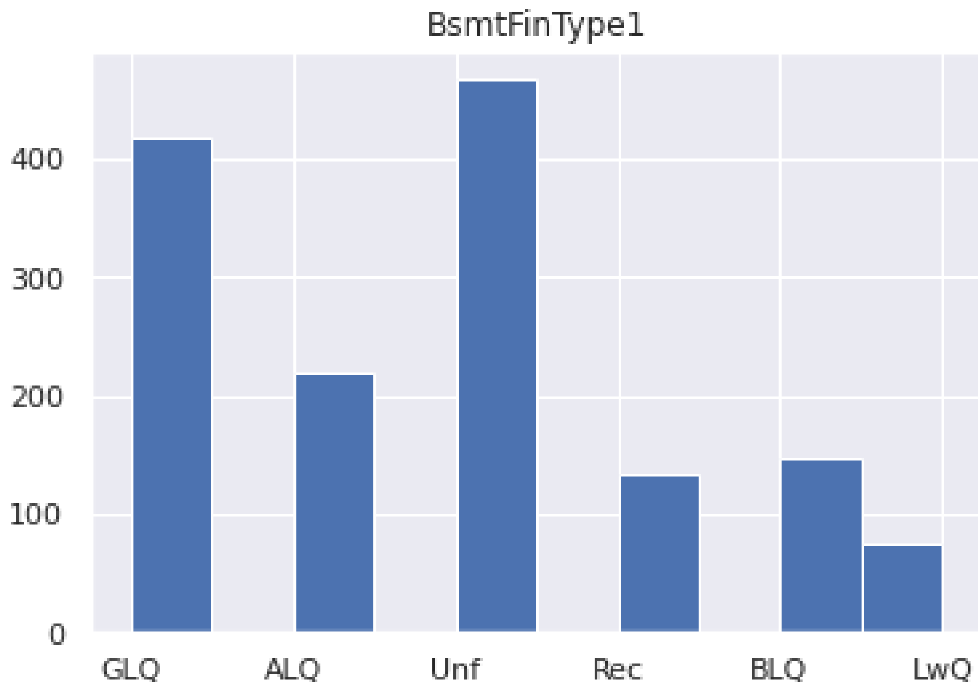
[314]:
```
for c in features_to_impute:
    df[c].hist()
    plt.title(c)
    plt.show()
```

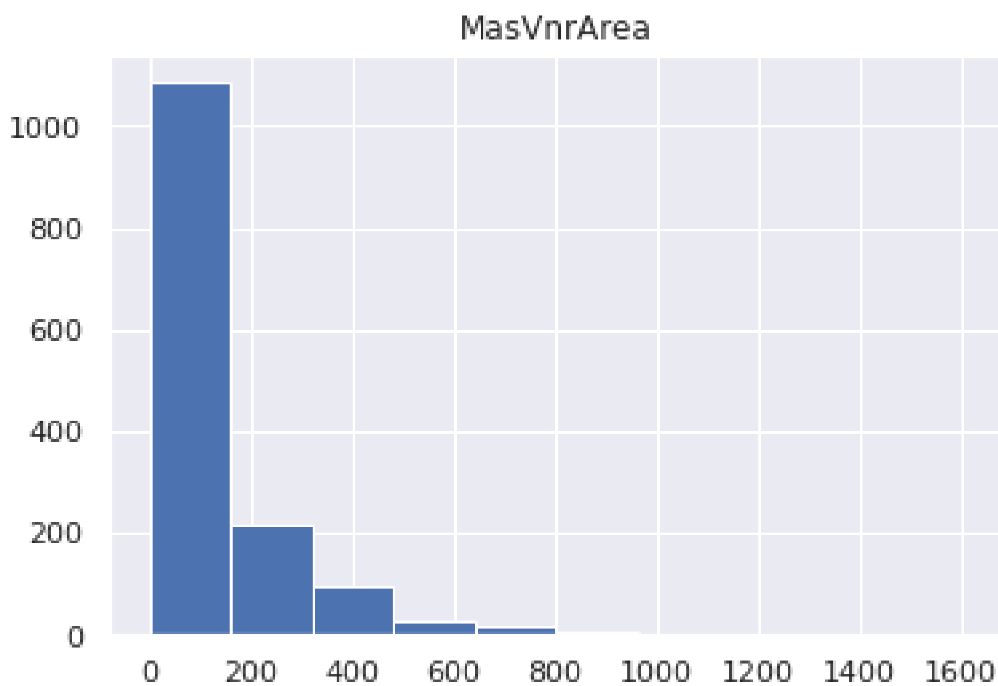

BsmtFinType2
1200
1000
800
600
400
200
0
Unf
BLQ
ALQ
Rec
LwQ
GLQ

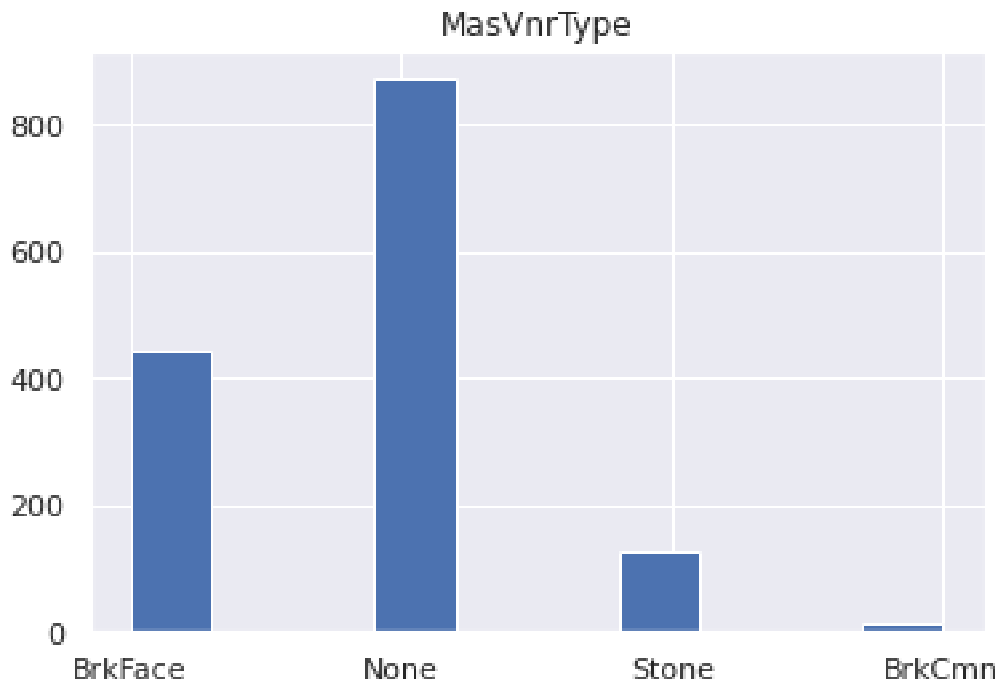

BsmtExposure
1000
800
600
400
200
0
No
Gd
Mn
Av

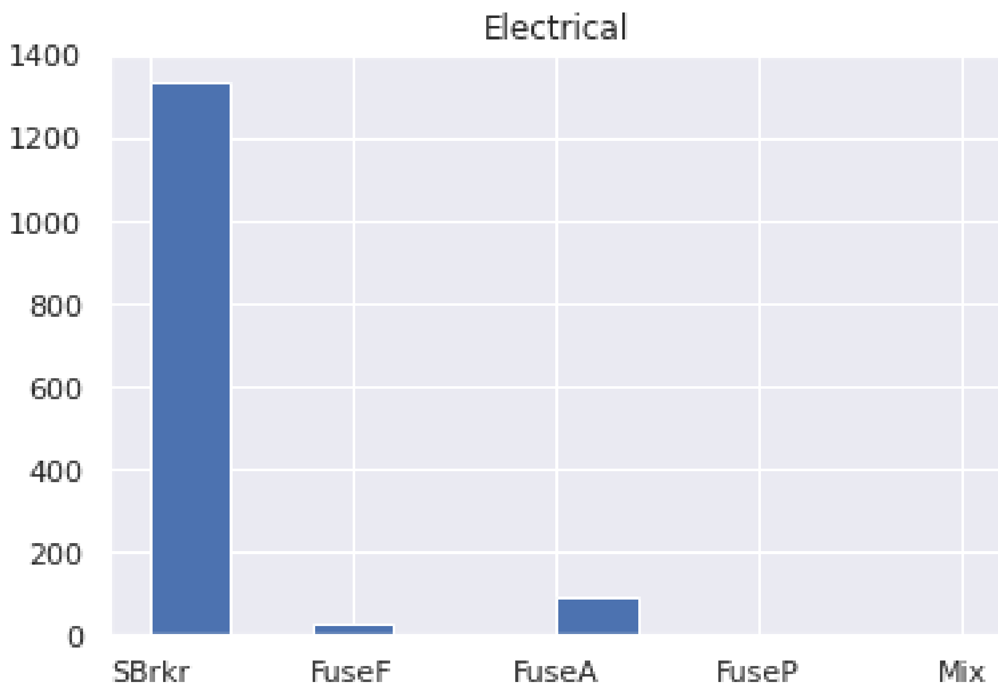

BsmtQual
700
600
500
400
300
200
100
0
Gd
TA
Ex
Fa


BsmtCond
1400
1200
1000
800
600
400
200
0
TA
Gd
Fa
Po


BsmtFinType1
0
100
200
300
400
GLQ
ALQ
Unf
Rec
BLQ
LwQ


MasVnrArea
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600





[315]:
```
# tests 'MasVnrType' and 'MasVnrArea'
```

[316]:
```python
# tsts 'BsmtQual' and 'BsmtCond'
```

[317]:
```python
# tests 'BsmtExposure' and 'BsmtFinType1'
```

[318]:
```python
# tests 'BsmtFinType2' and 'Electrical'
```

2 Part 2. EDA, Simple Linear Regression

In this part, we will use a simplified data and create a simple linear regression model. The dataset can be downloaded from https://www.kaggle.com/harlfoxem/housesalesprediction/download. This dataset contains house sale prices for Kings County, which includes Seattle. It includes homes sold between May 2014 and May 2015. There are several versions of the data. Some additional information about the columns is available here: https://geodacenter.github.io/data-and-lab/KingCounty-HouseSales2015/, some of which are copied below.

Variable	Description
id	Identification
date	Date sold
price	Sale price
bedrooms	Number of bedrooms
bathrooms	Number of bathrooms
sqft_liv	Size of living area in square feet
sqft_lot	Size of the lot in square feet
floors	Number of floors
waterfront	'1' if the property has a waterfront, '0' if not.
view	An index from 0 to 4 of how good the view of the property was
condition	Condition of the house, ranked from 1 to 5
grade	Classification by construction quality which refers to the types of materials used and the quality of workmanship. Buildings of better quality (higher grade) cost more to build per unit of measure and command higher value.
sqft_above	Square feet above ground
sqft_basmt	Square feet below ground
yr_built	Year built
yr_renov	Year renovated. '0' if never renovated
zipcode	5 digit zip code
lat	Latitude
long	Longitude
squft_liv15	Average size of interior housing living space for the closest 15 houses, in square feet
squft_lot15	Average size of land lost for the closest 15 houses, in square feet

[319]:
```
import scipy as sp
import scipy.stats as stats
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import copy
# Set color map to have light blue background
sns.set()
import statsmodels.formula.api as smf
import statsmodels.api as sm
%matplotlib inline
```

[320]:
```
df2 = pd.read_csv('data/house_data_washington.csv')
```

2.1 4. Munging data [15 pts]

2.1.1 4a) Date string to numbers [5 pts]

Inspect the data frame and data type of each column. The column 'date' is the date sold, and has string value. We will extract year and month information from the string. In the data frame df2, create new features 'sales_year' and 'sales_month'.

[321]:
```
# extract year and month info from the string
# create new features 'sales_year' and 'sales_month' in df2
df2['sales_year'] = df2.date.apply(lambda x: int(x[:4]))
df2['sales_month'] = df2.date.apply(lambda x: int(x[4:6]))
print(df2.groupby('sales_month')['id'].count())
print(df2.groupby('sales_year')['id'].count())
```

```
sales_month
1      978
2     1250
3     1875
4     2231
5     2414
6     2180
7     2211
8     1940
9     1774
10    1878
11    1411
12    1471
Name: id, dtype: int64
sales_year
2014    14633
```

```
2015     6980
Name: id, dtype: int64
```

Which month has the most number of sales?

[322]:
```python
# your code here

# uncomment and update string with capitalized month, e.g., 'December'
most_sales = 'May'
```

Which months has the least number of sales?

[323]:
```python
# your code here

# uncomment and update string with capitalized month, e.g., 'December'
least_sales = 'January'
```

[324]:
```python
# tests solutions for most_sales and least_sales
```

[325]:
```python
df2.head(10)
```

[325]:
```
           id             date    price  bedrooms  bathrooms  sqft_living  \
0  7129300520  20141013T000000   221900         3       1.00         1180
1  6414100192  20141209T000000   538000         3       2.25         2570
2  5631500400  20150225T000000   180000         2       1.00          770
3  2487200875  20141209T000000   604000         4       3.00         1960
4  1954400510  20150218T000000   510000         3       2.00         1680
5  7237550310  20140512T000000  1225000         4       4.50         5420
6  1321400060  20140627T000000   257500         3       2.25         1715
7  2008000270  20150115T000000   291850         3       1.50         1060
8  2414600126  20150415T000000   229500         3       1.00         1780
9  3793500160  20150312T000000   323000         3       2.50         1890

   sqft_lot  floors  waterfront  view  …  sqft_basement  yr_built  \
0      5650     1.0           0     0  …              0      1955
1      7242     2.0           0     0  …            400      1951
2     10000     1.0           0     0  …              0      1933
3      5000     1.0           0     0  …            910      1965
4      8080     1.0           0     0  …              0      1987
5    101930     1.0           0     0  …           1530      2001
6      6819     2.0           0     0  …              0      1995
7      9711     1.0           0     0  …              0      1963
8      7470     1.0           0     0  …            730      1960
9      6560     2.0           0     0  …              0      2003

   yr_renovated  zipcode      lat     long  sqft_living15  sqft_lot15  \
0             0    98178  47.5112 -122.257           1340        5650
1          1991    98125  47.7210 -122.319           1690        7639
2             0    98028  47.7379 -122.233           2720        8062
```

```
3            0    98136  47.5208 -122.393          1360         5000
4            0    98074  47.6168 -122.045          1800         7503
5            0    98053  47.6561 -122.005          4760       101930
6            0    98003  47.3097 -122.327          2238         6819
7            0    98198  47.4095 -122.315          1650         9711
8            0    98146  47.5123 -122.337          1780         8113
9            0    98038  47.3684 -122.031          2390         7570

   sales_year  sales_month
0        2014           10
1        2014           12
2        2015            2
3        2014           12
4        2015            2
5        2014            5
6        2014            6
7        2015            1
8        2015            4
9        2015            3

[10 rows x 23 columns]
```

[326]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 23 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  int64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
```

```
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
 21  sales_year     21613 non-null  int64
 22  sales_month    21613 non-null  int64
dtypes: float64(4), int64(18), object(1)
memory usage: 3.8+ MB
```

2.1.2 4b) Variable types [5 pts]

Inspect each feature's data type and variable type. What is the best description for the variable type of following features? Update the string to 'numeric' or 'categorical'.

[327]:
```python
# your code here



# uncomment the feaures below and update the strings with 'numeric' or␣
 ↪'categorical'
price = 'numeric'
bathrooms = 'numeric'
waterfront = 'categorical'
grade = 'numeric'
zipcode = 'categorical'
sales_year = 'numeric'
```

[328]:
```python
# tests that you selected correct variable type for the features in 4b
```

[329]:
```python
# this part is ungraded, but useful to run to check
# your code here

for c in df2.columns[2:]:
    print(c, df2[c].unique())
```

```
price [ 221900  538000  180000 …  610685 1007500  402101]
bedrooms [ 3  2  4  5  1  6  7  0  8  9 11 10 33]
bathrooms [1.   2.25 3.   2.   4.5  1.5  2.5  1.75 2.75 3.25 4.   3.5  0.75 4.75
 5.   4.25 3.75 0.   1.25 5.25 6.   0.5  5.5  6.75 5.75 8.   7.5  7.75
 6.25 6.5 ]
sqft_living [1180 2570  770 … 3087 3118 1425]
sqft_lot [ 5650  7242 10000 …  5813  2388  1076]
floors [1.  2.  1.5 3.  2.5 3.5]
waterfront [0 1]
view [0 3 4 2 1]
condition [3 5 4 1 2]
grade [ 7  6  8 11  9  5 10 12  4  3 13  1]
sqft_above [1180 2170  770 1050 1680 3890 1715 1060 1890 1860  860 1430 1370
1810
```

1980	1600	1200	1250	2330	2270	1070	2450	1710	1750	1400	790	2570	2320
1190	1510	1090	1280	930	2360	890	2620	2600	3595	1570	920	3160	990
2290	2165	1640	1000	2130	2830	2250	2420	3250	1850	1590	1260	2519	1540
1110	1770	2720	2240	3070	2380	2390	880	1040	910	3450	2350	1900	1010
960	2660	1610	765	3520	1290	1960	1160	1210	1270	1440	2190	2920	1460
1170	1240	3140	2030	2310	700	1080	2520	2780	1560	1450	1720	2910	1620
1360	2070	2460	1390	2140	1320	1340	1550	940	1380	3670	2370	1130	980
3540	2500	1760	1030	1780	3400	2680	1670	2590	820	1220	2440	2090	1100
1330	1420	1690	2150	1910	1350	1940	900	1630	2714	850	1870	1950	2760
2020	1120	1480	1230	2280	3760	3530	830	1300	2740	1830	720	2010	3360
800	1730	760	1700	4750	5310	580	2653	2850	2210	2630	3500	1740	1140
2160	2650	970	2040	2180	2220	1660	3370	2690	1930	3150	3030	2050	2490
2560	1275	2580	560	1820	1840	2990	3230	1580	3480	2510	1410	2120	3300
3840	1500	1530	2840	833	2000	6070	950	2200	4040	1920	1490	3470	3130
2610	3260	2260	430	3390	630	4860	3860	2810	870	3180	2770	4030	4410
2400	1520	3040	6050	4740	1970	5403	3350	3580	1790	750	2860	2750	2340
2870	4120	3200	2550	1805	4150	1384	2060	2110	3590	2100	2540	1880	1150
1470	1255	1800	4370	3190	2730	4570	2470	670	2900	4670	4230	2156	1020
2940	2640	2710	3100	3610	4270	840	3090	2300	380	2480	3460	3060	3064
3000	1654	2790	1310	2230	2430	3680	2670	2208	810	740	1422	490	2080
3440	5670	4475	730	3410	3010	600	2960	3570	4300	3990	780	3020	5990
440	4460	4190	2800	2530	1650	3690	2932	3720	4250	3110	2963	4930	2950
5000	2452	2820	1981	640	2495	2403	5320	6720	660	2341	4210	3830	3280
2980	5153	1990	1646	610	710	5450	3504	3210	1782	2930	590	4280	680
3880	3430	3750	4130	5710	3380	3330	4700	3220	3362	3510	3810	620	4490
2410	3050	1008	3488	4070	3420	5770	1605	520	1088	3555	4360	3960	2700
4340	1552	3850	2303	3270	4350	3640	2174	4160	2496	5180	5130	6350	3770
2153	3780	2890	1714	2201	2970	992	3950	3527	2835	3915	1427	4870	3340
3620	4310	3930	4080	5400	570	3310	6110	3320	3490	3859	3710	1798	4600
3560	3940	3600	3800	1105	2305	3290	5050	1556	1553	4000	1657	3001	4220
480	3120	3740	530	3700	5230	5370	3080	4140	4430	3550	1159	1288	2880
4610	1122	3052	1479	7680	3820	1934	5080	2675	2506	5760	2154	4390	3240
1995	1689	2782	2395	4400	6200	3526	4320	2483	4380	4580	4180	2064	3650
1726	2019	4240	1256	500	1355	1747	1678	1833	1414	4115	3597	3170	390
1976	5830	2601	3920	2641	5070	2518	3910	3660	3695	4020	2803	2074	2038
4060	4890	2329	1264	1095	690	4090	1392	2844	902	4560	2811	4720	2168
5610	2683	4900	2095	4290	4050	4260	4440	6220	1175	998	2356	4500	3900
3831	1315	4470	4810	2286	2927	4760	8570	5140	1679	1811	2849	1676	1757
3730	2441	2163	5250	2795	2415	3970	4200	1068	5240	1509	1954	4820	1651
4100	1752	3630	2885	3154	1129	2632	1996	4010	550	410	6430	3790	2031
1652	2434	3316	1899	2331	2497	2216	4170	1341	1961	5584	8860	2507	5220
4850	5844	5530	2145	650	1982	4910	3605	1778	1463	2783	1946	1358	3870
1864	1845	6290	3980	2382	2979	3674	2726	5440	1295	2115	6085	3265	3136
6640	4620	3361	2245	2242	1078	2577	1329	420	4330	1975	7420	1788	2299
1092	4225	1087	1904	470	2966	2192	2253	5550	4133	4285	1216	540	9410
2075	5330	2166	1628	1808	1352	2557	6380	7880	2734	1363	1769	2093	1677
2588	5190	2298	1491	2961	5020	5980	4540	844	6120	2233	4480	4110	4770
2473	995	5160	1494	2007	1048	3002	4780	2155	2014	4980	2665	4830	4790

```
 5010  370 2105 3006 3004 2689 4660 1746 2678 2755 2414  901 4630 2068
 2807 2643 2181 4510 4420 1604 1435 3045 2717 2905 4940 5110 2533 6660
 3485 2659 5090 2375 1964  866 1595  944 5480  809 5040 1764 1656 1802
  460 2692 1544 2044 1212 4083 8020 3905 1502 4590  384 2092 6090 1615
 7320 1396 1484 1765 5490 1453 1643 5300 1381 4065  290 1313 5430 1397
 2793 2475 1936 3028  798 2575 3276 1584 2393 2029 3222 1072 1785 1984
  962 2423 2052 2538 2437 2789 2906 4800 7850 2196 1847 2658 2655 3855
 1728  963 2223 1611 2015 2448 1489 1116 3745 1002 3202 1347 1481 2311
 2544 2584 2217 3569 3181 1921 2612 2671 2598 3284 3266 1076 2594 2718
 1794 2481 3845 1413 1876 3148 2413 1767 5060  806 2547 1834 2024 1165
 2134 1741 2798 1852 2099 3216 1094 2891 2432 2283 2701 1658  893 2009
 1444 2744 3078 3065 1578 2815 4960 1571 6530 4640 1536 3172 6370 3223
 1608 2229 3135 1408 1763 4840 1232 2502 2424 1296 1914  988 3828 3056
 2267 1131 2796 1812 1084 2025 1564 1239 2568 1528 2628 2185 2478 2669
 1912 2828 2425 1446 3206 2406 1419 2056 1144 2456 4950 3192  828 2529
 2732 1987 3906 4073 2578 2738 3691 1061 2846 2542 1889 3336 3236 1451
 1983 2313 1824 1322 1766 2301 3274 1108 2864 2716 1572 3281 2656 2398
 1867 1613 2587 2623  894 1606 2244 2026 2238 2517 2708 2555 1405 4450
 1248 6420 2531 1333 2198 3087 3118 1425]
sqft_basement [   0  400  910 1530  730 1700  300  970  760  720  700  820  780
790
  330 1620  360  588 1510  410  990  600  560  550 1000 1600  500 1040
  880 1010  240  265  290  800  540  380  710  840  770  480  570 1490
  620 1250 1270  120  650  180 1130  450 1640 1460 1020 1030  750  640
 1070  490 1310  630 2000  390  430  850  210 1430 1950  440  220 1160
  860  580 2060 1820 1180  200 1150 1200  680  530 1450 1170 1080  960
 1100  280  870  460 1400 1320  660 1220  900  420 1580 1380  475  690
  270  350  935 1370  980 1470  160  950   50  740 1780 1900  340  470
  370  140 1760  130  610  520  890 1110  150 1720  810  190 1290  670
 1800 1120 1810   60 1050  940  310  930 1390 1830 1300  510 1330 1590
  920 1420 1240 1960 1560 2020 1190 2110 1280  250 2390 1230  170  830
 1260 1410 1340  590 1500 1140  260  100  320 1480 1060 1284 1670 1350
 2570 2590 1090  110 2500   90 1940 1550 2350 2490 1481 1360 1135 1520
 1850 1660 2130 2600 1690  243 1210 2620 1024 1798 1610 1440 1570 1650
  704 1910 1630 2360 1852 2090 2400 1790 2150  230   70 1680 2100 3000
 1870 1710 2030  875 1540 2850 2170  506  906  145 2040  784 1750  374
  518 2720 2730 1840 3480 2160 1920 2330 1860 2050 4820 1913   80 2010
 3260 2200  415 1730  652 2196 1930  515   40 2080 2580 1548 1740  235
  861 1890 2220  792 2070 4130 2250 2240  894 1990  768 2550  435 1008
 2300 2610  666 3500  172 1816 2190 1245 1525 1880  862  946 1281  414
 2180  276 1248  602  516  176  225 1275  266  283   65 2310   10 1770
 2120  295  207  915  556  417  143  508 2810   20  274  248]
yr_built [1955 1951 1933 1965 1987 2001 1995 1963 1960 2003 1942 1927 1977 1900
 1979 1994 1916 1921 1969 1947 1968 1985 1941 1915 1909 1948 2005 1929
 1981 1930 1904 1996 2000 1984 2014 1922 1959 1966 1953 1950 2008 1991
 1954 1973 1925 1989 1972 1986 1956 2002 1992 1964 1952 1961 2006 1988
 1962 1939 1946 1967 1975 1980 1910 1983 1978 1905 1971 2010 1945 1924
 1990 1914 1926 2004 1923 2007 1976 1949 1999 1901 1993 1920 1997 1943
```

```
 1957 1940 1918 1928 1974 1911 1936 1937 1982 1908 1931 1998 1913 2013
 1907 1958 2012 1912 2011 1917 1932 1944 1902 2009 1903 1970 2015 1934
 1938 1919 1906 1935]
yr_renovated [   0 1991 2002 2010 1999 1992 2013 1994 1978 2005 2008 2003 1984
1954
 2014 2011 1974 1983 1945 1990 1988 1957 1977 1981 1995 2000 1998 1970
 1989 2004 1986 2009 2007 1987 1973 2006 1985 2001 1980 1971 1979 1997
 1950 1969 1948 2015 1968 2012 1963 1951 1993 1962 1996 1972 1953 1955
 1982 1956 1940 1976 1946 1975 1958 1964 1959 1960 1967 1965 1934 1944]
zipcode [98178 98125 98028 98136 98074 98053 98003 98198 98146 98038 98007 98115
 98107 98126 98019 98103 98002 98133 98040 98092 98030 98119 98112 98052
 98027 98117 98058 98001 98056 98166 98023 98070 98148 98105 98042 98008
 98059 98122 98144 98004 98005 98034 98075 98116 98010 98118 98199 98032
 98045 98102 98077 98108 98168 98177 98065 98029 98006 98109 98022 98033
 98155 98024 98011 98031 98106 98072 98188 98014 98055 98039]
lat [47.5112 47.721  47.7379 … 47.3906 47.3339 47.6502]
long [-122.257 -122.319 -122.233 -122.393 -122.045 -122.005 -122.327 -122.315
 -122.337 -122.031 -122.145 -122.292 -122.229 -122.394 -122.375 -121.962
 -122.343 -122.21  -122.306 -122.341 -122.169 -122.166 -122.172 -122.218
 -122.36  -122.314 -122.304 -122.11  -122.07  -122.357 -122.368 -122.157
 -122.31  -122.132 -122.362 -122.282 -122.18  -122.027 -122.347 -122.016
 -122.364 -122.175 -121.977 -122.371 -122.151 -122.301 -122.451 -122.322
 -122.189 -122.384 -122.369 -122.281 -122.29  -122.114 -122.122 -122.116
 -122.149 -122.339 -122.335 -122.344 -122.32  -122.297 -122.192 -122.215
 -122.16  -122.179 -122.287 -122.036 -122.073 -121.987 -122.125 -122.34
 -122.025 -122.008 -122.291 -122.365 -122.199 -122.194 -122.387 -122.372
 -122.391 -122.351 -122.386 -122.249 -122.277 -122.378 -121.958 -121.714
 -122.08  -122.196 -122.184 -122.133 -122.38  -122.082 -122.109 -122.053
 -122.349 -122.295 -122.253 -122.248 -122.303 -122.294 -122.226 -122.266
 -122.098 -122.212 -122.244 -122.39  -122.352 -121.85  -122.152 -122.054
 -122.072 -121.998 -122.296 -122.299 -122.381 -122.358 -122.128 -122.171
 -122.174 -122.026 -122.353 -121.943 -122.286 -122.336 -122.359 -122.162
 -122.3   -122.176 -121.996 -122.118 -122.193 -122.023 -122.224 -122.168
 -122.231 -122.331 -122.374 -122.182 -122.308 -122.307 -121.999 -122.376
 -122.2   -122.039 -122.102 -122.188 -122.379 -122.043 -122.153 -122.191
 -122.219 -122.312 -121.911 -121.994 -122.165 -122.37  -122.158 -122.047
 -122.284 -122.017 -122.275 -122.268 -122.367 -122.217 -122.373 -122.013
 -122.214 -122.034 -122.164 -121.899 -122.183 -121.95  -122.324 -122.216
 -122.395 -122.213 -122.345 -122.278 -122.111 -121.711 -122.27  -122.178
 -122.147 -121.772 -122.302 -122.438 -122.223 -122.042 -122.323 -122.255
 -122.4   -122.261 -122.071 -122.206 -122.272 -122.23  -122.144 -122.143
 -122.181 -122.154 -122.311 -122.274 -122.077 -122.    -122.298 -122.058
 -121.837 -122.333 -122.057 -122.252 -122.093 -122.012 -122.052 -122.354
 -122.22  -122.49  -121.875 -122.24  -122.078 -122.173 -121.854 -122.222
 -122.28  -122.137 -122.159 -121.974 -122.141 -122.029 -121.709 -122.19
 -121.97  -122.329 -122.195 -122.06  -121.959 -122.095 -122.148 -122.146
 -122.35  -121.901 -122.241 -122.129 -122.289 -122.305 -122.022 -122.385
 -121.779 -122.032 -122.402 -122.482 -122.227 -121.982 -122.161 -122.046
```

-122.156	-122.127	-122.33	-122.197	-122.041	-122.103	-122.318	-122.382
-122.271	-121.955	-122.211	-122.262	-122.258	-122.121	-122.221	-122.234
-122.089	-122.123	-122.167	-121.909	-122.107	-122.064	-122.066	-122.062
-122.264	-122.186	-122.087	-121.88	-121.864	-122.205	-122.363	-122.139
-122.018	-122.225	-122.285	-122.084	-122.177	-122.056	-122.316	-122.021
-122.348	-122.009	-122.131	-122.411	-122.198	-122.256	-122.117	-122.097
-122.075	-121.845	-122.083	-122.259	-121.87	-122.015	-122.007	-121.86
-122.409	-121.755	-121.972	-122.251	-122.317	-121.776	-122.115	-122.283
-122.242	-122.001	-122.024	-122.309	-122.113	-121.771	-122.239	-122.273
-122.396	-122.094	-122.267	-122.326	-122.13	-122.269	-121.853	-122.05
-122.346	-122.076	-121.826	-122.124	-121.758	-122.202	-121.785	-121.872
-122.006	-122.004	-122.321	-121.882	-122.101	-122.03	-122.185	-122.1
-121.759	-121.965	-122.201	-122.366	-122.313	-122.405	-122.02	-122.279
-122.355	-121.934	-122.15	-122.356	-121.993	-122.044	-122.134	-121.867
-122.01	-121.991	-122.011	-121.983	-122.228	-122.033	-122.276	-122.119
-121.937	-122.361	-122.325	-122.203	-122.136	-122.237	-122.209	-122.049
-122.288	-122.106	-122.037	-122.207	-122.263	-121.915	-122.204	-122.09
-122.069	-121.852	-121.787	-121.976	-122.377	-122.059	-122.383	-121.989
-122.019	-122.208	-121.878	-122.328	-122.25	-122.338	-122.388	-122.265
-122.332	-122.399	-122.397	-122.014	-121.956	-122.092	-122.028	-122.293
-122.12	-122.035	-122.14	-122.04	-122.112	-121.906	-122.17	-122.238
-122.512	-121.997	-121.89	-122.463	-121.908	-122.086	-122.389	-121.913
-122.163	-121.918	-122.108	-122.502	-122.392	-122.236	-121.859	-121.981
-122.342	-121.96	-121.978	-122.47	-121.91	-121.966	-122.065	-122.246
-122.41	-121.879	-122.079	-122.099	-122.187	-121.98	-122.002	-122.138
-121.898	-122.235	-122.126	-121.782	-121.995	-122.401	-121.858	-121.888
-121.752	-122.063	-122.26	-121.78	-121.708	-121.721	-122.403	-121.945
-122.243	-122.45	-121.927	-122.085	-122.088	-121.973	-122.055	-122.398
-121.984	-121.912	-121.903	-121.946	-122.232	-122.412	-122.104	-122.048
-122.479	-122.155	-121.833	-121.778	-122.003	-121.99	-121.926	-122.051
-121.986	-122.245	-121.861	-122.431	-121.964	-122.142	-122.074	-122.247
-122.497	-121.769	-121.827	-121.979	-121.871	-122.091	-121.754	-121.746
-121.92	-121.992	-122.406	-121.359	-121.789	-121.707	-122.068	-122.404
-122.334	-121.799	-121.774	-121.985	-121.865	-121.724	-122.415	-121.756
-121.809	-122.135	-121.691	-122.038	-121.877	-121.94	-121.968	-121.988
-121.315	-121.902	-122.514	-122.414	-121.883	-121.866	-121.744	-122.096
-122.061	-121.881	-121.745	-122.461	-122.067	-121.868	-121.646	-121.93
-122.105	-121.763	-121.718	-121.967	-121.777	-121.957	-121.823	-121.887
-122.408	-122.462	-122.43	-122.456	-121.897	-121.932	-121.969	-121.916
-122.081	-121.975	-121.735	-121.801	-121.761	-121.723	-121.924	-122.475
-121.935	-122.407	-122.448	-122.453	-121.894	-121.936	-121.764	-122.416
-121.905	-122.464	-121.768	-122.484	-121.738	-121.9	-121.82	-122.455
-121.889	-122.496	-121.829	-122.505	-121.951	-121.847	-122.509	-121.961
-121.417	-121.904	-122.503	-121.949	-121.874	-122.432	-121.971	-121.77
-122.473	-121.896	-121.952	-122.254	-121.743	-121.933	-121.892	-121.749
-121.473	-121.857	-122.465	-121.838	-121.954	-122.422	-121.931	-121.963
-122.441	-121.925	-121.352	-122.511	-122.413	-121.876	-121.748	-121.818
-121.8	-121.929	-121.698	-121.886	-121.802	-121.81	-121.762	-121.781

```
 -121.775 -122.44  -121.773 -121.819 -121.726 -122.459 -122.446 -121.855
 -121.736 -122.499 -122.46  -121.786 -122.421 -121.947 -122.439 -121.834
 -121.804 -122.443 -121.716 -121.848 -122.458 -122.515 -121.922 -121.953
 -121.783 -122.472 -121.944 -121.869 -121.828 -122.452 -121.831 -121.737
 -121.739 -121.863 -121.73  -121.856 -121.747 -121.893 -121.733 -121.846
 -121.821 -121.319 -121.765 -121.75  -122.506 -121.948 -121.921 -122.507
 -122.457 -121.914 -122.469 -121.792 -121.907 -121.841 -121.757 -121.788
 -121.731 -122.449 -121.316 -121.321 -122.504 -121.884 -121.803 -121.842
 -121.719 -121.766 -122.433 -122.519 -121.851 -121.402 -122.454 -122.467
 -121.325 -121.815 -121.676 -121.941 -122.445 -121.76  -121.885 -121.742
 -121.822 -121.895 -121.784 -121.701 -121.713 -121.727 -121.849 -121.835
 -122.435 -122.474 -122.444 -121.939 -121.48  -121.364 -121.767 -122.42
 -121.84  -122.425 -122.447 -121.797 -122.491 -121.917 -121.891 -121.942
 -121.862 -121.725 -121.873 -121.405 -122.486 -121.795 -121.734 -121.403]
sqft_living15 [1340 1690 2720 1360 1800 4760 2238 1650 1780 2390 2210 1330 1370
2140
 1890 1610 1060 1280 1400 4110 2240 1220 2200 1030 1760 1860 1520 2630
 2580 1390 1460 1570 2020 1590 2160 1730 1290 2620 2470 2410 3625 1580
 3050 1228 2680  970 1190 1990 1410 1480 2730 1950 2250 2690 2960 2270
 2570 2500 1440 2750 2221 1010 3390 3530 1640 1510 2420 1940 3240 1680
  890 1130 3350 2350 1870 1720 1850 1900 1980 2520 1350 1750 1160 2550
 2370 1240 1270 2990 1380 1540 2090 2640 1830 1620 1880 2340 1710 2700
 3060 2660 1700 1970 1420 2060 2480 1550 1170 2820 1560 2230 2840 1450
 1500 3160 1200 3400 2110 2920 1770 1070 1930 3740 2260 1670 2290 1050
 2540 2190 2030 1230 2330 1300 1430 2770 1250 1630 2590 2130 1100 3836
 1320 2120 3070 1910 2080 1960 2280 1150 3430 2070 2600  830 1260 3120
 2010 1660 1600 2380 3890 4180 2653 2670 3920 2300 2310 2320 3150 1740
 2400 4550 2510 2440 2880 3860 2150 1310 1820 3080  880 2560 3470 1020
 2040 2610 1810 2860 3480 3130 3360 4050 2450 1790 3180 3600 2000 2430
 2850 4680 2360 3930 1490 2460 2077 1920 3630 3220 2100 3230 4300 3850
 2424 2530 3030 2830 2900 2950 1470  940 2740 4210 3340 3980 2180 3715
 2050 1080 2095 1000 3330 2170 1408 1530 2760 3110  950 3000 1307 2220
 4190 3440 3250 1110 2870 1210 2910 1120 4230 1708 3090 3270 2970 1180
 3100 4100 2930 3510 2688 1840 2490 4090 2810 3260 3680 3420 1654 1365
  980 1677 1140 3640 3460 3140 1502 3720 2790 2940  990 2890  860 4750
 1525 3950 5790  760 2234  960 3210 2780 2800 2305 2665 3620 2710 4320
 2650 3370 1509 1277 1981 2434 4640 2242 3040 3970 3200 4600  840 3290
 2214 1162 3010 5600 3820 3540 1975 4800  740 3990 3170 1576 1768 3310
 2980 1429 3900 3380  820 1090 4060 3910 3190 3450 3730  620 3020 3760
 3320 1132 3300 3770 3960  870 3560 4620 3520 1572 3490 1088 3159 4470
 3570 4890 3690 3280 2083 3780  920 1941 1566  850 2496 1040 3410 4240
 4670 4350 1714 5380 4330 3830 5000 2144 1494 1357  930 3580 4250 4080
 3660 1458 3736 1894 2037 1295 4170 3750 3550 4630 1439 3500 2091  900
 3880 3710 1616  720  800 2315 1564 2767 3721 4650 4020  780 1728 2027
 1264 1404 1459 2028 3639 1943 3425 2641 2114 1309 2412 2517 1802 2011
 1466 1414 3193 1845 1156 3670 1696 5340 4440 1745 1884 4690 4920 2406
 4160 3810 4480 2848 1746 2634 2049 5330 1536 2273 3056 4010 4700  910
 2125 1665 2683 3790  700 1855  750 1078 4150 4340 2344 1098 1175 1188
```

```
 3700 3840 4042 2518 3800 2488 3590 2052  810 1528 5030 4740 5070 2967
 4280 2724 3610 3940 4940 4770 1811 4830 2876 1805 1216 5170 1304 2474
 4590 4130 1492 1364 2168 4140 3543 1303 2005 3650 2583 4310 2451 1448
 2955 2142  790 1638 2554 2441 2216 4220 1961 4540  770 4200 3413 1664
 2136 3568 4510 1484 1358 2106 1834 2014 4390 4570 2175 6110 4260  710
 2112 1934 1518 1302 2622 2619 2382 4290 4560 4000 1336 3112 4070 1468
 1571 2605 1138 5110 4850 2165 4410 1678 5610 1984 4660 3870 4370  460
 4610 1914 3515 2246 1786 2109 2326 2728 4400 4950 1767 2054 5500 2555
 3674 2765 1862 1352 4030  399 2415 2901 1815 2236 2253 2004 1356 2403
 1137 1256 4930 4040 2376 4520 4490 2189 2566 2396 1282 2155 1056 2389
 2256 3618 1326 1168 4913  806 1369 2405 2875 1425 5220 1442 2333 3335
 1321 3045 1546 4730 2697 2822 2076 1757 4780  952 4270 2075 2667 1092
 1217 1716 1792 2961 1125 1463 1886  670 4460 2336 3557 5200 2258 1377
 2019 2092 4900 2615 1639 1765 1554 1381 4120 5080 1445 2793 2475  998
 2384 2575 1398 1584 2439 2197 2029 4362 1443 4420 1691 2495 2437 2547
 6210 2009 1847 1346 2578 2879 2255 2815 1608  690 2425 1481 2458 2358
 2056 1921 2419 2996 2502 1798 3087 1076 2981 2363 3191 1763 1876 1949
 2598 1979 1415 2002 2574 2166 3726 2099 2154 1522 1544 2912 2648 1658
 2755 2798 1405 2704 2738 3008 2586 2873 1232 2597 2516 1537 1128 2849
 1399 1131 1569 2381 1084 2304 4530 2297 2279 2303 2669 4225 2513 2725
 1955 2527 4443 2478 1919 1813 2533  828 2015 3078 4495 2673 2316 2647
 3402 3494 2156 3236 2612 2323 2409 2354 1285 2616 1427 1516 2456 2844
 1495 2594 2604 1268 2198 3038 2927]
sqft_lot15 [5650 7639 8062 … 5731 1509 2007]
sales_year [2014 2015]
sales_month [10 12  2  5  6  1  4  3  7  8 11  9]
```

2.1.3 4c) Drop features [5 pts]

Let's drop features that are unnecessary. `id` is not a meaningful feature. `date` string has been coded to `sales_month` and `sales_year`, so we can remove `date`. `zipcode` can be also removed as it's hard to include in a linear regressio model and the location info is included in the `lat` and `long`. Drop the features `id`, `date`, and `zipcode` and replace the df2.

[330]:
```
# drop unnecessary features, replace df2
# your code here
df2.drop(['id', 'date', 'zipcode'],axis=1, inplace=True)
```

[331]:
```
# tests that you droppd the features id, date, and zipcode from df2
```

2.2 5. More inspection; Correlation and pair plot [5 pts and Peer Review]

2.2.1 5a) Get correlation matrix on the data frame. [5 pts]

Which feature may be the best predictor of price based on the correlation? Answer as a string value (e.g. best_guess_predictor = 'price' or best_guess_predictor = 'yr_built')

[332]:

```python
# your code here
print(df2.corr())

# uncomment and update best_guess_predictor with a string value
best_guess_predictor = 'sqft_living'
```

```
                  price  bedrooms  bathrooms  sqft_living  sqft_lot    floors  \
price          1.000000  0.308350   0.525138     0.702035  0.089661  0.256794
bedrooms       0.308350  1.000000   0.515884     0.576671  0.031703  0.175429
bathrooms      0.525138  0.515884   1.000000     0.754665  0.087740  0.500653
sqft_living    0.702035  0.576671   0.754665     1.000000  0.172826  0.353949
sqft_lot       0.089661  0.031703   0.087740     0.172826  1.000000 -0.005201
floors         0.256794  0.175429   0.500653     0.353949 -0.005201  1.000000
waterfront     0.266369 -0.006582   0.063744     0.103818  0.021604  0.023698
view           0.397293  0.079532   0.187737     0.284611  0.074710  0.029444
condition      0.036362  0.028472  -0.124982    -0.058753 -0.008958 -0.263768
grade          0.667434  0.356967   0.664983     0.762704  0.113621  0.458183
sqft_above     0.605567  0.477600   0.685342     0.876597  0.183512  0.523885
sqft_basement  0.323816  0.303093   0.283770     0.435043  0.015286 -0.245705
yr_built       0.054012  0.154178   0.506019     0.318049  0.053080  0.489319
yr_renovated   0.126434  0.018841   0.050739     0.055363  0.007644  0.006338
lat            0.307003 -0.008931   0.024573     0.052529 -0.085683  0.049614
long           0.021626  0.129473   0.223042     0.240223  0.229521  0.125419
sqft_living15  0.585379  0.391638   0.568634     0.756420  0.144608  0.279885
sqft_lot15     0.082447  0.029244   0.087175     0.183286  0.718557 -0.011269
sales_year     0.003576 -0.009838  -0.026596    -0.029038  0.005468 -0.022315
sales_month   -0.010081 -0.001533   0.007392     0.011810 -0.002369  0.014005

               waterfront      view  condition     grade  sqft_above  \
price            0.266369  0.397293   0.036362  0.667434    0.605567
bedrooms        -0.006582  0.079532   0.028472  0.356967    0.477600
bathrooms        0.063744  0.187737  -0.124982  0.664983    0.685342
sqft_living      0.103818  0.284611  -0.058753  0.762704    0.876597
sqft_lot         0.021604  0.074710  -0.008958  0.113621    0.183512
floors           0.023698  0.029444  -0.263768  0.458183    0.523885
waterfront       1.000000  0.401857   0.016653  0.082775    0.072075
view             0.401857  1.000000   0.045990  0.251321    0.167649
condition        0.016653  0.045990   1.000000 -0.144674   -0.158214
grade            0.082775  0.251321  -0.144674  1.000000    0.755923
sqft_above       0.072075  0.167649  -0.158214  0.755923    1.000000
sqft_basement    0.080588  0.276947   0.174105  0.168392   -0.051943
yr_built        -0.026161 -0.053440  -0.361417  0.446963    0.423898
yr_renovated     0.092885  0.103917  -0.060618  0.014414    0.023285
lat             -0.014274  0.006157  -0.014941  0.114084   -0.000816
long            -0.041910 -0.078400  -0.106500  0.198372    0.343803
sqft_living15    0.086463  0.280439  -0.092824  0.713202    0.731870
sqft_lot15       0.030703  0.072575  -0.003406  0.119248    0.194050
```

sales_year	-0.004165	0.001364	-0.045589	-0.030387	-0.023823
sales_month	0.008132	-0.005638	0.021978	0.008376	0.009872

	sqft_basement	yr_built	yr_renovated	lat	long \
price	0.323816	0.054012	0.126434	0.307003	0.021626
bedrooms	0.303093	0.154178	0.018841	-0.008931	0.129473
bathrooms	0.283770	0.506019	0.050739	0.024573	0.223042
sqft_living	0.435043	0.318049	0.055363	0.052529	0.240223
sqft_lot	0.015286	0.053080	0.007644	-0.085683	0.229521
floors	-0.245705	0.489319	0.006338	0.049614	0.125419
waterfront	0.080588	-0.026161	0.092885	-0.014274	-0.041910
view	0.276947	-0.053440	0.103917	0.006157	-0.078400
condition	0.174105	-0.361417	-0.060618	-0.014941	-0.106500
grade	0.168392	0.446963	0.014414	0.114084	0.198372
sqft_above	-0.051943	0.423898	0.023285	-0.000816	0.343803
sqft_basement	1.000000	-0.133124	0.071323	0.110538	-0.144765
yr_built	-0.133124	1.000000	-0.224874	-0.148122	0.409356
yr_renovated	0.071323	-0.224874	1.000000	0.029398	-0.068372
lat	0.110538	-0.148122	0.029398	1.000000	-0.135512
long	-0.144765	0.409356	-0.068372	-0.135512	1.000000
sqft_living15	0.200355	0.326229	-0.002673	0.048858	0.334605
sqft_lot15	0.017276	0.070958	0.007854	-0.086419	0.254451
sales_year	-0.015687	0.003507	-0.023707	-0.029212	0.000270
sales_month	0.006035	-0.006226	0.012827	0.014961	-0.008134

	sqft_living15	sqft_lot15	sales_year	sales_month
price	0.585379	0.082447	0.003576	-0.010081
bedrooms	0.391638	0.029244	-0.009838	-0.001533
bathrooms	0.568634	0.087175	-0.026596	0.007392
sqft_living	0.756420	0.183286	-0.029038	0.011810
sqft_lot	0.144608	0.718557	0.005468	-0.002369
floors	0.279885	-0.011269	-0.022315	0.014005
waterfront	0.086463	0.030703	-0.004165	0.008132
view	0.280439	0.072575	0.001364	-0.005638
condition	-0.092824	-0.003406	-0.045589	0.021978
grade	0.713202	0.119248	-0.030387	0.008376
sqft_above	0.731870	0.194050	-0.023823	0.009872
sqft_basement	0.200355	0.017276	-0.015687	0.006035
yr_built	0.326229	0.070958	0.003507	-0.006226
yr_renovated	-0.002673	0.007854	-0.023707	0.012827
lat	0.048858	-0.086419	-0.029212	0.014961
long	0.334605	0.254451	0.000270	-0.008134
sqft_living15	1.000000	0.183192	-0.021734	0.002449
sqft_lot15	0.183192	1.000000	-0.000085	0.003546
sales_year	-0.021734	-0.000085	1.000000	-0.782389
sales_month	0.002449	0.003546	-0.782389	1.000000

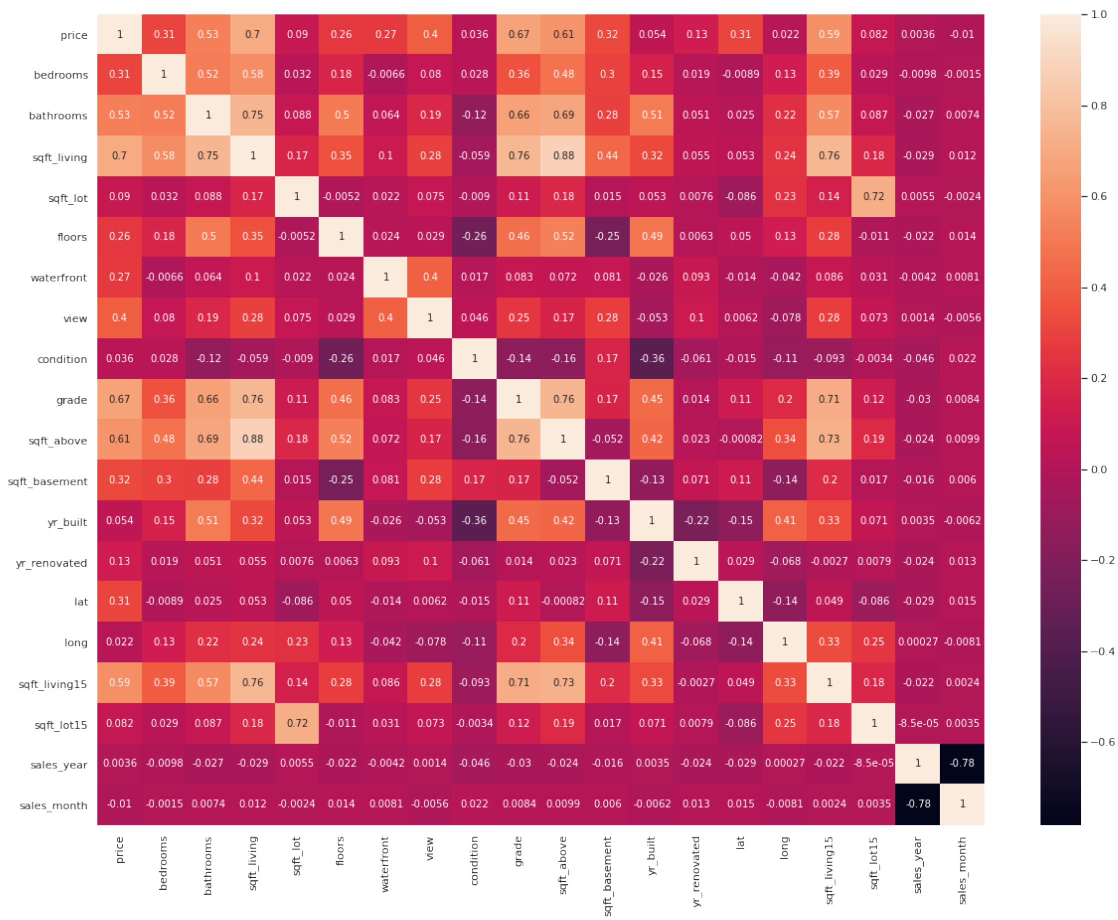
```
[333]: # tests the solution for best_guess_predictor
```

2.2.2 5b) Display the correlation matrix as heat map [Peer Review]

seaborn.heatmap() can visualize a matrix as a heatmap. Visualize the correlation matrix using seaborn.heatmap(). Play with color map, text font size, decimals, text orientation etc. If you find how to make a pretty visualization, please share in the discussion board. You will upload your correlation matrix in the Peer Review assigment for the week. **Note:** your code for this section may cause the Validate button to time out. If you want to run the Validate button prior to submitting, you could comment out the code in this section after completing the Peer Review.

```
[334]: # practice visualizing correlation matrix using a heatmap
# your code here
plt.subplots(figsize=(20,15))
sns.heatmap(df2.corr(), annot=True)
```

```
[334]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1f1bf4bad0>
```

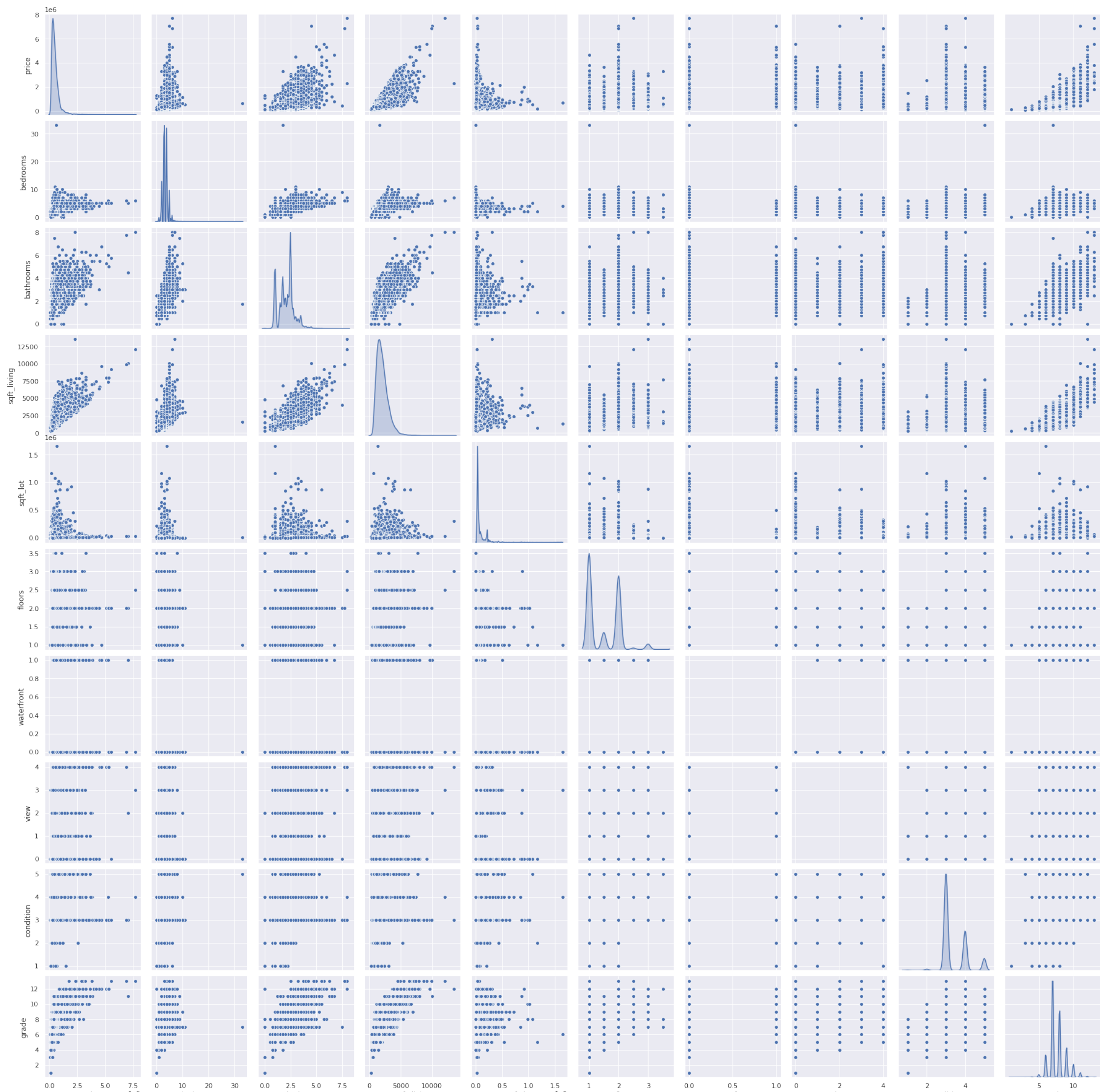


2.2.3 5c) Pair plot [Peer Review]

Pair plot is a fast way to inspect relationships between features. Use seaborn's .pairplot() function to draw a pairplot if the first 10 columns (including price) and inspect their relationships. Set the diagonal elements to be KDE plot. You will upload your pair plot in this week's Peer Review assignment. **Note:** your code for this section may cause the Validate button to time out. If you want to run the Validate button prior to submitting, you could comment out the code in this section after completing the Peer Review.

[335]:
```python
# practice inspecting relationships between features using a pair plot.
# your code here
sns.pairplot(df2.iloc[:, :10], diag_kind='kde')
```

[335]:
```
<seaborn.axisgrid.PairGrid at 0x7f1f1bf529d0>
```



2.3 6. Simple linear regression [Peer Review]

2.3.1 6a) Data preparation [Peer Review]

We will split the data to train and test datasets such that the test dataset is 20% of original data. Use `sklearn.model_selection.train_test_split` function to split the data frame to X_train and X_test. X_train is 80% of observation randomly chosen. X_test is the rest 20%. Both X_train and X_test are `pd.DataFrame` object and include 'price' in the table. Note that the train_test_split can handle data frame as well as array.

[336]:
```
# your code here
from sklearn.model_selection import train_test_split as tst
X_train, X_test = tst(df2, test_size=0.20, random_state=15413)

# use sklearn.model_selecttion.train_test_split to split the data frame
# X_train is 80% of the observations; X_test is 20% of the observations
# print length of X_train and X_test
print(len(X_train))
print(len(X_test))
```

```
17290
4323
```

[337]:
```
# instructor testing cell
# your code here
```

2.3.2 6b) Train a simple linear regression model [Peer Review]

Use the best_guess_predictor as a single predictor and build a simple linear regression model using `statsmodels.formula.api.ols` function (https://www.statsmodels.org/dev/example_formulas.html) Print out the result summary. Train on the X_train portion. What is the adjusted R-squared value?

[338]:
```
# use best_guess_predictor as a single predictor
# build a simple linear regression model, train on the X_train portion
# your code here

model = smf.ols(formula='price ~ sqft_living', data=X_train)
res = model.fit()
print(res.summary())

adj_R2 = 0.493 #update this value according to the result
```

```
                            OLS Regression Results
==============================================================================
```

```
Dep. Variable:                  price   R-squared:                       0.493
Model:                            OLS   Adj. R-squared:                  0.493
Method:                 Least Squares   F-statistic:                 1.684e+04
Date:                Sat, 16 Apr 2022   Prob (F-statistic):               0.00
Time:                        01:33:25   Log-Likelihood:            -2.4028e+05
No. Observations:               17290   AIC:                         4.806e+05
Df Residuals:                   17288   BIC:                         4.806e+05
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
Intercept   -4.468e+04   4923.238     -9.075      0.000   -5.43e+04    -3.5e+04
sqft_living   281.1457      2.167    129.764      0.000     276.899     285.392
==============================================================================
Omnibus:                    11584.466   Durbin-Watson:                   1.997
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           380472.155
Skew:                           2.760   Prob(JB):                         0.00
Kurtosis:                      25.308   Cond. No.                     5.60e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.6e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

2.3.3 6c) Best predictor [Peer Review]

In question 5a, we picked a best guess predictor for price based on the correlation matrix. Now we will consider whether the best_guess_predictor that we used is still the best. Print out a list ranking all of the predictors. Then print out a list of the top three predictors in order.
Hint: Linear regression uses adjusted R squared as fit performance. In this week's Peer Review, answer the following questions: What were your top three predictors? How did you order your list of predictors to select those as the top ones? Is your top predictor for this section the same as the best guess predictor you selected in question 5a?

[339]:
```
# your code here
### Ranking og each predictor and their adjusted R^2

# sqft_living - 0.493
# grade - 0.450
# sqft_above - 0.371
# sqft_living15 - 0.346
# bathrooms - 0.281
# view - 0.16
# sqft_basement - 0.102
```

```python
# bedrooms - 0.098
# lat - 0.094
# floors - 0.066
# waterfront - 0.066
# yr_renovated - 0.016
# sqft_lot - 0.008
# sqft_lot15 - 0.008
# yr_built - 0.003
# condition - 0.001
# long - 0
# sales_year - 0
# sales_month - 0

# uncomment and update top_three
top_three = ['sqft_living', 'grade', 'sqft_above']
```

[340]:
```python
# instructor testing cell
# your code here
```