final

1 DATA1030 Final

1.0.1 Due 12/15/18 at 11:59 pm

Student Name: Shiyun Zou

Student Email: shiyun_zou@brown.edu

Student Github Name: IreneSZ

Link to student Github Account:https://github.com/IreneSZ/1030final

Student Kaggle Account Name: Irene Zou

Link to student Kaggle Account: https://www.kaggle.com/irenesz

---

Directions: 1. This is an open computer, open book, and open web exam. You are encouraged to review concepts from lectures, labs and the textbooks for definitions and technical help. 2. However, you are expressly forbidden from searching for actual or similar problem solutions. 3. All work on this exam must be entirely your own. No talking or sharing with your classmates or anyone else. 4. You can use PyCharm, Pythontutor and any other tools you like to work on various problems in this exam. 5. Submission: Create a directory called `final` at the top level of your data1030 student GitHub folder.
6. Place a notebook called `final.ipynb` in it that contains the exam tasks below. Include all necessary code. 7. Be sure to organize your notebook so that it is clear what each cell is doing, and which question it relates to or answers. 8. **Important:** your `final` directory must include all additional files that your notebook requires. ** The grading process automatically uses the file names provided, so please spell and capitalize them exactly as given.**

Notebooks that cannot be run from start to finish will be scored a zero.

1.1 Guided Kaggle Competition

For your final exam in DATA 1030, you will create a submission for the

House Prices: Advanced Regression Techniques description Kaggle competition.

In order to speed your work, and to give you an example of a detailed analysis of this dataset that leads to a reasonable model, your work will be guided by Erik Bruin's kernel analysis and submission described in this Kaggle R Kernel House prices: Lasso, XGBoost, and a detailed EDA.

Below is a linked table of contents to a copy of this kernel.

Your task for this exam will be to use Python, sklearn and the plloting libraries of your choice, to recreate the critical aspects of his analysis (including ETL and EDA) in order for you to develop your own submission. While it is possible to work online using the Kaggle platform, it will probably be more efficient for you to work locally by modifying this notebook.

1.2 Below are the required sections for your notebook.

- Include an appropriate narratives where appropriate, also try and fully develop most of the techniques he employed to visualize, analyze and improve the data.

- Along the way be sure to do appropriate ETL on the final model variables model, but in order to save time, you can skip data cleaning and other steps on irrelevant variables.

- For Section 9, you should use sklearn gridsearch to try and improve his final model.

- Extra Credit [10]: Review the sklearn Preprocessing Material and implement your feature transformations using appropriate Transformer functions, e.g. the preprocessing module further provides a utility class StandardScaler that implements the Transformer API to compute the mean and standard deviation on a training set so as to be able to later reapply the same transformation on the testing set.

1.2.1 Final Hand-in steps:

Kaggle competition entry

- Design your notebook so that when run top to bottom it will generate a copy of your final `final_submission.csv`. Include a copy of this file in the **`final`** directory that you turn in.
- Save a copy of the final copy of your notebook as a regular `.ipynb` and as a `.pdf`
- Remeber to also participate in the competition and to submit your final submission.

1.2.2 Additional Resources:

The Kaggle machine learning tutorial is quite good, and also uses the Ames Housing dataset in many of its kernels. For example,

- https://www.kaggle.com/dansbecker/xgboost
- https://www.kaggle.com/dansbecker/submitting-from-a-kernel

You are also encouraged to look at, as needed, at the other kernels related to this competition (even the ones in Python)

```
### BEGIN SOLUTION
```

1.3 1 Executive Summary [10]

1.4 EDA

1.4.1 I used histogram, correlation matrix, scatterplot and box-whisker to examine the distribution of sale price, the correlation between independent variables, and the behavior of the selected independent variables, which have the highest correlation with sale price.

1.5 Handling the missing data, non-numerical data

1.5.1 For variables with missing data, I used different methods to treat them, based on the context of the problem. For example, all the homes with NAN fireplace quality has 0 fireplace, which makes sense and I changed the NAN into None. For numerical series, I tried interpolating numerically, with mean or median when approporiate.

1.5.2 In order to run later tests, I also converted the non-numerical variables into category, and then used one hot encoding to prepare them for regression models.

1.5.3 Also, I normalized the features.

1.6 Handling outlier, skewness, etc.

1.6.1 I also checked and chose to remove 2 outliers, checked for normality (qq plot) and converted price to log scale to reduce the fat tail.

1.7 Regression models

1.7.1 I used Lasso and XGBoost models, both tuned with sklearn gridsearch, cross validation.

1.7.2 The best parameters are chosen for the two models, and are used to predict the y_test.

1.7.3 The final submission is a weighted average of the two prediction series.

1.8 2 Introduction

1.9 3 Loading and Exploring Data [10]

1.9.1 3.1 Loading libraries required and reading the data into Python

1.9.2 3.2 Data size and structure

```
In [1]: #3.1 load the data
        import pandas as pd
        train = pd.read_csv("train.csv")
        test = pd.read_csv("test.csv")
```

```
In [2]: #3.2 data size
        train_size = train.shape
        test_size = test.shape
        print("The train set size is", train_size, "; ", "the test size is ", test_size)
        #combine the two sets
        df = pd.concat([train, test], ignore_index=True)
```

```
The train set size is (1460, 81) ;  the test size is  (1459, 80)


/home/shiyun/.local/lib/python3.6/site-packages/ipykernel_launcher.py:6: FutureWarning: Sorting
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.
```

```
In [3]: df.tail()
```

```
Out[3]:       1stFlrSF  2ndFlrSF  3SsnPorch Alley  BedroomAbvGr BldgType BsmtCond  \
        2914       546       546          0   NaN             3    Twnhs       TA
        2915       546       546          0   NaN             3   TwnhsE       TA
        2916      1224         0          0   NaN             4     1Fam       TA
        2917       970         0          0   NaN             3     1Fam       TA
        2918       996      1004          0   NaN             3     1Fam       TA

             BsmtExposure  BsmtFinSF1  BsmtFinSF2  ...   SaleType ScreenPorch  Street  \
        2914           No         0.0         0.0  ...         WD           0    Pave
        2915           No       252.0         0.0  ...         WD           0    Pave
        2916           No      1224.0         0.0  ...         WD           0    Pave
        2917           Av       337.0         0.0  ...         WD           0    Pave
        2918           Av       758.0         0.0  ...         WD           0    Pave

              TotRmsAbvGrd TotalBsmtSF  Utilities WoodDeckSF YearBuilt YearRemodAdd  \
        2914             5       546.0     AllPub          0      1970         1970
        2915             6       546.0     AllPub          0      1970         1970
        2916             7      1224.0     AllPub        474      1960         1996
        2917             6       912.0     AllPub         80      1992         1992
        2918             9       996.0     AllPub        190      1993         1994

             YrSold
        2914   2006
        2915   2006
        2916   2006
        2917   2006
        2918   2006
```

```
[5 rows x 81 columns]
```

```
In [4]: # The first 10 rows
        df.head()
```

```
Out[4]:    1stFlrSF  2ndFlrSF  3SsnPorch Alley  BedroomAbvGr BldgType BsmtCond  \
        0       856       854          0   NaN             3     1Fam       TA
        1      1262         0          0   NaN             3     1Fam       TA
        2       920       866          0   NaN             3     1Fam       TA
        3       961       756          0   NaN             3     1Fam       Gd
        4      1145      1053          0   NaN             4     1Fam       TA

          BsmtExposure  BsmtFinSF1  BsmtFinSF2  ...   SaleType ScreenPorch  Street  \
        0           No       706.0         0.0  ...         WD           0    Pave
        1           Gd       978.0         0.0  ...         WD           0    Pave
        2           Mn       486.0         0.0  ...         WD           0    Pave
        3           No       216.0         0.0  ...         WD           0    Pave
        4           Av       655.0         0.0  ...         WD           0    Pave

           TotRmsAbvGrd TotalBsmtSF  Utilities WoodDeckSF YearBuilt YearRemodAdd  \
        0             8       856.0     AllPub          0      2003         2003
        1             6      1262.0     AllPub        298      1976         1976
        2             6       920.0     AllPub          0      2001         2002
        3             7       756.0     AllPub          0      1915         1970
        4             9      1145.0     AllPub        192      2000         2000

          YrSold
        0   2008
        1   2007
        2   2008
        3   2006
        4   2008

        [5 rows x 81 columns]
```

```
In [5]: #datatype of the first 10 variables
        df.dtypes[1:11]
```

```
Out[5]: 2ndFlrSF          int64
        3SsnPorch         int64
        Alley            object
        BedroomAbvGr      int64
        BldgType         object
        BsmtCond         object
        BsmtExposure     object
        BsmtFinSF1      float64
        BsmtFinSF2      float64
        BsmtFinType1     object
        dtype: object
```

1.10 4 Exploring some of the most important variables [10]

1.10.1 4.1 The response variable; SalePrice

1.10.2 4.2 The most important numeric predictors

1.10.3 4.2.1 Correlations with SalePrice

1.10.4 4.2.2 Overall Quality

1.10.5 4.2.3 Above Grade (Ground) Living Area (square feet)

In [6]:
```
#4.1 the distribution of the response variable
import matplotlib.pyplot as plt
plt.hist(train['SalePrice'],bins=80)
plt.xlabel("SalePrice")
plt.ylabel("count")
plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

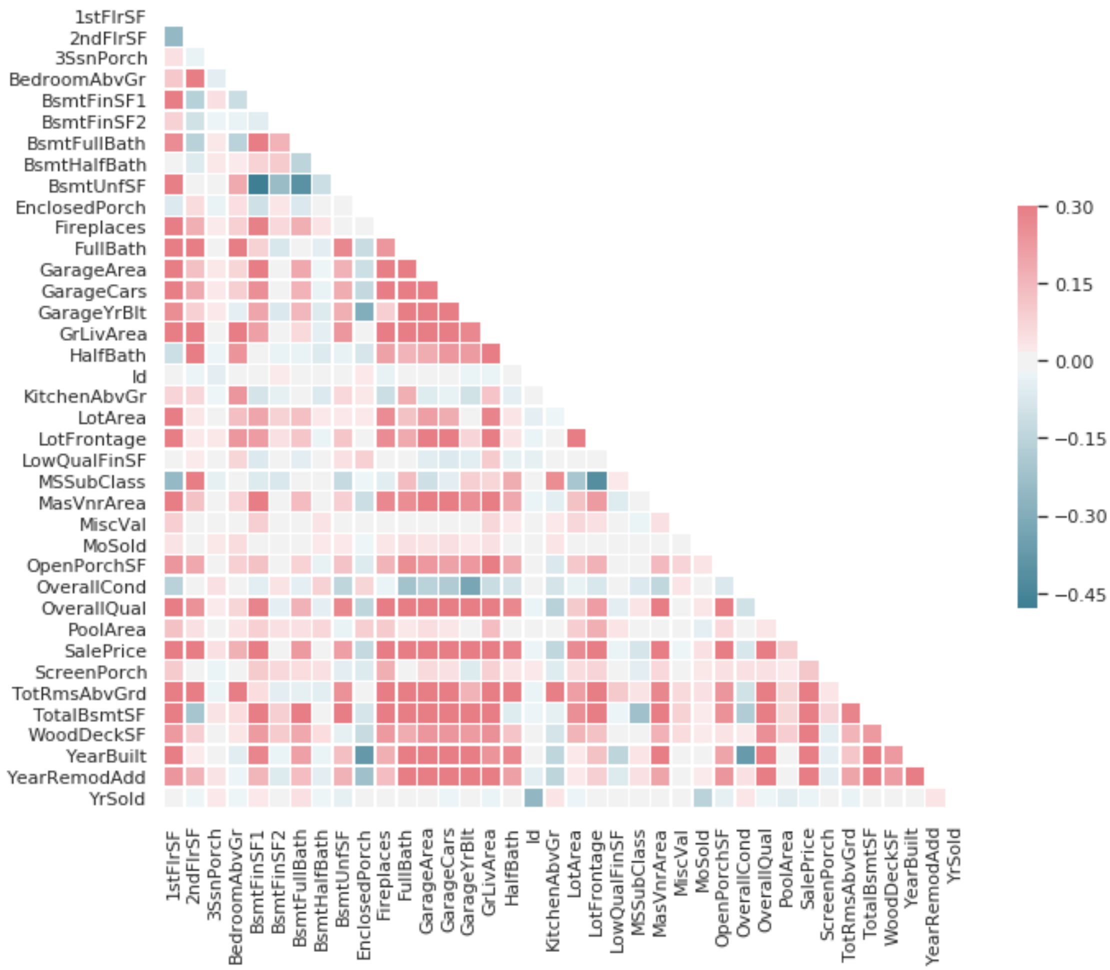
In [7]:
```
#statistics of saleprice
df['SalePrice'].describe()
```

Out[7]:
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

In [8]:
```
#4.2.1 correlations
import numpy as np
import seaborn as sns
df_numeric = df.select_dtypes(include=np.number)

sns.set(style="white")
corr = df_numeric.corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)
```

```
# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2cb5a5e128>

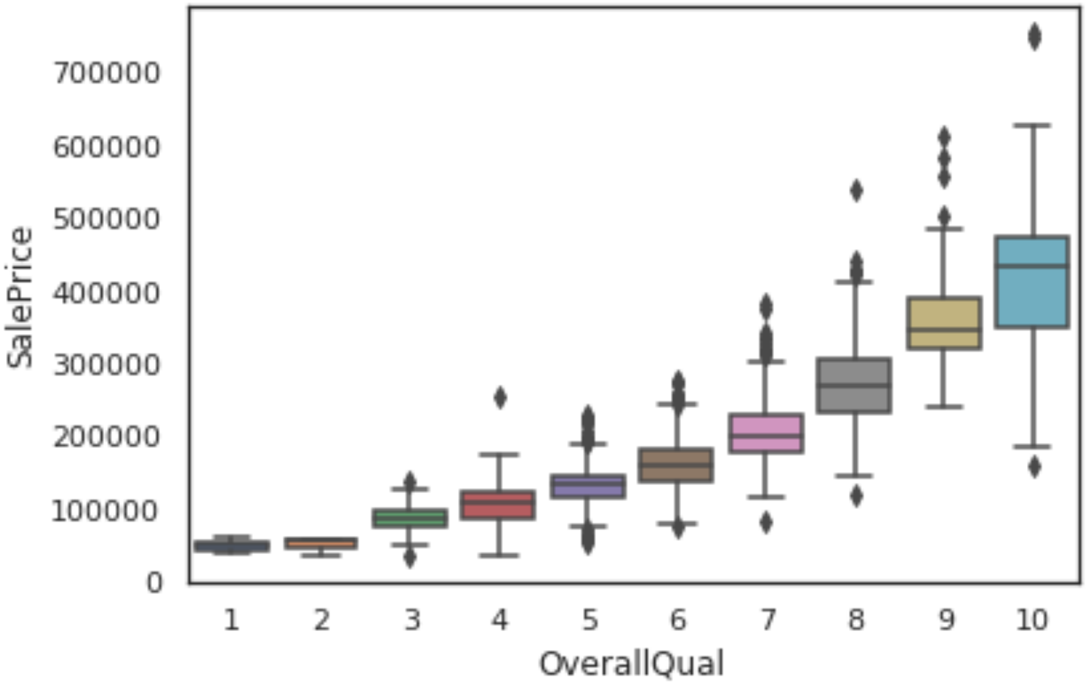



```
In [9]: #find the 10 variables with the highest corr with saleprice
        #the top 10 variables's correlations with saleprice are all greater than 0.5
        corr['SalePrice'].sort_values(ascending=False)[1:11]
```

```
Out[9]: OverallQual     0.790982
        GrLivArea       0.708624
        GarageCars      0.640409
        GarageArea      0.623431
        TotalBsmtSF     0.613581
        1stFlrSF        0.605852
        FullBath        0.560664
```

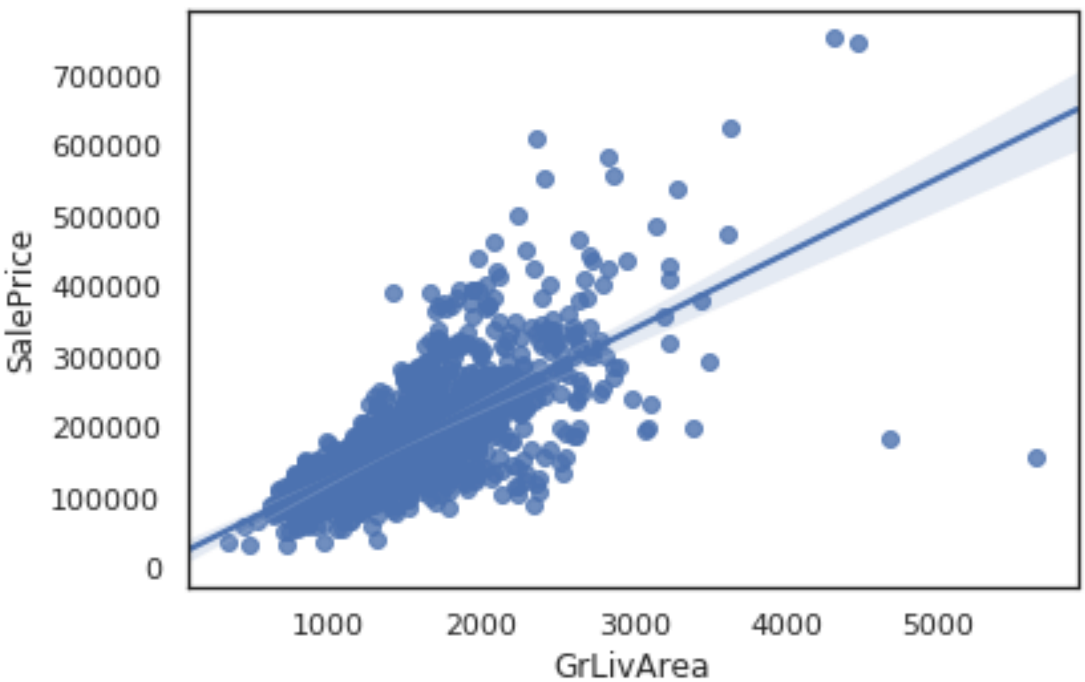
```
TotRmsAbvGrd     0.533723
YearBuilt        0.522897
YearRemodAdd     0.507101
Name: SalePrice, dtype: float64
```

```
In [10]: #4.2.2
         ax = sns.boxplot(x="OverallQual", y="SalePrice", data=df)
```




```
In [11]: #4.2.3 Above Grade (Ground) Living Area (square feet)
         ax = sns.regplot(x="GrLivArea", y="SalePrice", data=df.loc[df['SalePrice']!=np.nan])
```

```
/home/shiyun/.local/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



1.11 5 Missing data, label encoding, and factorizing variables [5]

1.11.1 5.1 Completeness of the data

1.11.2 5.2 Imputing missing data

5.2.1 Pool variables

5.2.2 Miscellaneous Feature

5.2.3 Alley

5.2.4 Fence

5.2.5 Fireplace variables

5.2.6 Lot variables

5.2.7 Garage variables

5.2.8 Basement Variables

5.2.9 Masonry variables

5.2.10 MS Zoning

5.2.11 Kitchen variables

5.2.12 Utilities

5.2.13 Home functionality

5.2.14 Exterior variables

5.2.15 Electrical system

5.2.16 Sale Type and Condition

1.12 5.3 Label encoding/factorizing the remaining character variables [5]

5.3.1 Foundation

5.3.2 Heating and airco

5.3.3 Roof

5.3.4 Land

5.3.5 Dwelling

5.3.6 Neighborhood and Conditions

5.3.7 Pavement of Street & Driveway

1.12.1 5.4 Changing some numeric variables into factors

5.4.1 Year and Month Sold

5.4.2 MSSubClass

In [12]:
```
#5.1 check for missing values
df.isnull().sum().sort_values(ascending=False)
```

Out[12]:
```
PoolQC          2909
MiscFeature     2814
Alley           2721
Fence           2348
SalePrice       1459
FireplaceQu     1420
LotFrontage      486
```

```
GarageFinish      159
GarageCond        159
GarageQual        159
GarageYrBlt       159
GarageType        157
BsmtCond           82
BsmtExposure       82
BsmtQual           81
BsmtFinType2       80
BsmtFinType1       79
MasVnrType         24
MasVnrArea         23
MSZoning            4
BsmtFullBath        2
BsmtHalfBath        2
Utilities           2
Functional          2
Electrical          1
Exterior2nd         1
KitchenQual         1
Exterior1st         1
GarageCars          1
TotalBsmtSF         1
                 ...
Neighborhood        0
YearBuilt           0
WoodDeckSF          0
TotRmsAbvGrd        0
Street              0
ScreenPorch         0
SaleCondition       0
RoofStyle           0
RoofMatl            0
PoolArea            0
PavedDrive          0
OverallQual         0
OverallCond         0
OpenPorchSF         0
MoSold              0
HalfBath            0
MiscVal             0
MSSubClass          0
LowQualFinSF        0
LotShape            0
LotConfig           0
LotArea             0
LandSlope           0
LandContour         0
```

```
KitchenAbvGr        0
Id                  0
HouseStyle          0
HeatingQC           0
YearRemodAdd        0
1stFlrSF            0
Length: 81, dtype: int64
```

In [13]:
```
#5.2.1 pool variables: pool quality
#assign 'no pool' to NAs
df['PoolQC'].fillna('no pool', inplace=True)
```

In [14]:
```
#encode the labels as ordinal
df['PoolQC'].replace('no pool', 0, inplace=True)
df['PoolQC'].replace('Po', 1, inplace=True)
df['PoolQC'].replace('TA', 3, inplace=True)
df['PoolQC'].replace('Fa', 2, inplace=True)
df['PoolQC'].replace('Gd', 4, inplace=True)
df['PoolQC'].replace('Ex', 5, inplace=True)

print(df.PoolQC.unique())
```

```
[0 5 2 4]
```

In [15]:
```
#5.2.1 pool area
df[['PoolArea','PoolQC','OverallQual']].loc[(df['PoolArea'] > 0) & (df['PoolQC'] == 0)
```

Out[15]:

	PoolArea	PoolQC	OverallQual
2420	368	0	4
2503	444	0	6
2599	561	0	3

In [16]:
```
df['PoolQC'][2420] = 2
df['PoolQC'][2503] = 3
df['PoolQC'][2600] = 2
```

```
/home/shiyun/.local/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.
/home/shiyun/.local/lib/python3.6/site-packages/ipykernel_launcher.py:2: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html

/home/shiyun/.local/lib/python3.6/site-packages/ipykernel_launcher.py:3: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  This is separate from the ipykernel package so we can avoid doing imports until
```

In [17]:
```
#5.2.2 Misc Features
#count na
df['MiscFeature'].isnull().sum()
```

Out[17]: 2814

In [18]:
```
df.groupby(['MiscFeature']).count()
```

Out[18]:
```
             1stFlrSF  2ndFlrSF  3SsnPorch  Alley  BedroomAbvGr  BldgType  \
MiscFeature
Gar2                5         5          5      1             5         5
Othr                4         4          4      0             4         4
Shed               95        95         95      7            95        95
TenC                1         1          1      0             1         1

             BsmtCond  BsmtExposure  BsmtFinSF1  BsmtFinSF2   ...    SaleType  \
MiscFeature                                                   ...
Gar2                5             5           5           5   ...           5
Othr                3             3           4           4   ...           4
Shed               91            91          95          95   ...          95
TenC                1             1           1           1   ...           1

             ScreenPorch  Street  TotRmsAbvGrd  TotalBsmtSF  Utilities  \
MiscFeature
Gar2                   5       5             5            5          5
Othr                   4       4             4            4          4
Shed                  95      95            95           95         94
TenC                   1       1             1            1          1

             WoodDeckSF  YearBuilt  YearRemodAdd  YrSold
MiscFeature
Gar2                  5          5             5       5
Othr                  4          4             4       4
Shed                 95         95            95      95
TenC                  1          1             1       1

[4 rows x 80 columns]
```

In [19]:
```
#5.2.3 Alley
#check for NA
df['Alley'].isnull().sum()
```

Out[19]: 2721

In [20]:
```
#convert NA into None
df['Alley'].fillna(value='None', inplace=True)
df['Alley'].isnull().sum()
```

Out[20]: 0

In [21]:
```
#5.2.4 fence: nan -> None
df['Fence'].isnull().sum()
```

Out[21]: 2348

In [22]:
```
#convert fence into categories
df['Fence'].fillna(value='None', inplace=True)
df['Fence'] = df['Fence'].astype('category')
```

In [23]:
```
# 5.2.5 fireplace
#fire place quality na -> no fireplace
#change into integers of level of quality
df['FireplaceQu'].fillna(value='None', inplace=True)
df['FireplaceQu'].replace('None', 0, inplace=True)
df['FireplaceQu'].replace('Po', 1, inplace=True)
df['FireplaceQu'].replace('Fa', 2, inplace=True)
df['FireplaceQu'].replace('TA', 3, inplace=True)
df['FireplaceQu'].replace('Gd', 4, inplace=True)
df['FireplaceQu'].replace('Ex', 5, inplace=True)
```

In [24]:
```
df['FireplaceQu'].unique()
```

Out[24]: array([0, 3, 4, 2, 5, 1])

In [25]:
```
#number of fireplaces
#check for missing value
df['Fireplaces'].isnull().sum()
```

Out[25]: 0

In [26]:
```
#5.2.6 Lot
df['LotFrontage'].isnull().sum()
```

Out[26]: 486

In [27]:
```
#interpolate the missing data using neighbor median
m = df.groupby(['Neighborhood'])['LotFrontage'].median()
for i in range(0,len(df)):
    if pd.isnull(df['LotFrontage'][i]):
        df['LotFrontage'][i] = m[df['Neighborhood'][i]]
    else:
        pass
df['LotFrontage'].isnull().sum()
```

```
/home/shiyun/.local/lib/python3.6/site-packages/ipykernel_launcher.py:5: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """
```

Out[27]: 0

In [28]:
```
#interpolate all the missing value in numerical variables
#using mean to fill the missing data of each numerical column
num_cols = list(df.select_dtypes(include=['int64']).columns)
for value in num_cols:
    avg = df[value].mean()
    df[value].fillna(avg, inplace=True)
```

In [29]:
```
for value in num_cols:
    print(value, df[value].isnull().sum())
```

```
1stFlrSF 0
2ndFlrSF 0
3SsnPorch 0
BedroomAbvGr 0
EnclosedPorch 0
FireplaceQu 0
Fireplaces 0
FullBath 0
GrLivArea 0
HalfBath 0
Id 0
KitchenAbvGr 0
LotArea 0
LowQualFinSF 0
MSSubClass 0
MiscVal 0
MoSold 0
OpenPorchSF 0
OverallCond 0
OverallQual 0
PoolArea 0
PoolQC 0
ScreenPorch 0
TotRmsAbvGrd 0
WoodDeckSF 0
YearBuilt 0
YearRemodAdd 0
YrSold 0
```

```
In [30]: #5.3 label encoding
         #5.3.1 foundation --> category
         df['Foundation'] = df['Foundation'].astype('category')
         df['Foundation'].isnull().sum()
```

```
Out[30]: 0
```

```
In [31]: #5.3.2 heating and airco
         #heating type --> category
         df['Heating'] = df['Heating'].astype('category')

         #heating QC --> quantiles
         revalue = {'None': 0, 'Po':1, 'Fa':2, 'TA': 3, 'Gd':4, 'Ex':5}
         df['HeatingQC'] = df['HeatingQC'].map(revalue)

         #centralair --> 0 or 1
         binary = {'N': 0, 'Y': 1}
         df['CentralAir'] = df['CentralAir'].map(binary)
```

```
In [32]: df['HeatingQC'].isnull().sum()
         df['Foundation'].isnull().sum()
         df['Heating'].isnull().sum()
         df['CentralAir'].isnull().sum()
```

```
Out[32]: 0
```

```
In [33]: #5.3.3 roof
         #roof style --> category
         df['RoofStyle'] = df['RoofStyle'].astype('category')

         #roof material --> category
         df['RoofMatl'] = df['RoofMatl'].astype('category')
```

```
In [34]: print(df['HeatingQC'].isnull().sum(), df['HeatingQC'].isnull().sum())
```

```
0 0
```

```
In [35]: #5.3.4
         #land contour --> category
         df['LandContour'] = df['LandContour'].astype('category')

         #land slope --> 0,1,2
         tripple = {'Sev':0, 'Mod':1, 'Gtl':2}
         df['LandSlope'] = df['LandSlope'].map(tripple)
```

```
In [36]: print(df['LandContour'].isnull().sum(), df['LandSlope'].isnull().sum())
```

```
0 0
```

In [37]:
```
#5.3.5 dwelling
#building type --> category
df['BldgType'] = df['BldgType'].astype('category')

#housetyle --> category
df['HouseStyle'] = df['HouseStyle'].astype('category')
```

In [38]:
```
print(df['BldgType'].isnull().sum(), df['HouseStyle'].isnull().sum())
```

```
0 0
```

In [39]:
```
#5.3.6 neighborhood
#neighborhood --> category
df['Neighborhood'] = df['Neighborhood'].astype('category')

#condition 1 --> category
df['Condition1'] = df['Condition1'].astype('category')

#condition 2 --> category
df['Condition2'] = df['Condition2'].astype('category')
```

In [40]:
```
print(df['Neighborhood'].isnull().sum(), df['Condition1'].isnull().sum(), df['Conditi
```

```
0 0 0
```

In [41]:
```
#5.3.7 pavement
# street --> 0, 1
binary2 = {'Grvl': 0, 'Pave': 1}
df['Street'] = df['Street'].map(binary2)

#paveddrive --> 0,1,2
tripple2 = {'N':0, 'P':1, 'Y':2}
df['PavedDrive'] = df['PavedDrive'].map(tripple2)
```

In [42]:
```
print(df['Street'].isnull().sum(), df['PavedDrive'].isnull().sum())
```

```
0 0
```

In [43]:
```
#5.4
#5.4.1
df['YrSold'] = df['YrSold'].astype('category')
df['MoSold'] = df['MoSold'].astype('category')
df['YearBuilt'] = df['YearBuilt'].astype('category')
df['YearRemodAdd'] = df['YearRemodAdd'].astype('category')
```

In [44]:
```
#5.4.2 type of dwelling
#change into category
df['MSSubClass'] = df['MSSubClass'].astype('category')
df['MSSubClass'].unique()
```

```
Out[44]: [60, 20, 70, 50, 190, ..., 160, 75, 180, 40, 150]
         Length: 16
         Categories (16, int64): [60, 20, 70, 50, ..., 75, 180, 40, 150]
```

```
In [ ]:
```

1.13 6 Visualization of important variables [20]

1.13.1 6.1 Correlations again

1.13.2 6.2 Finding variable importance with a quick Random Forest

6.2.1 Above Ground Living Area, and other surface related variables (in square feet)

6.2.2 The most important categorical variable; Neighborhood

6.2.3 Overall Quality, and other Quality variables

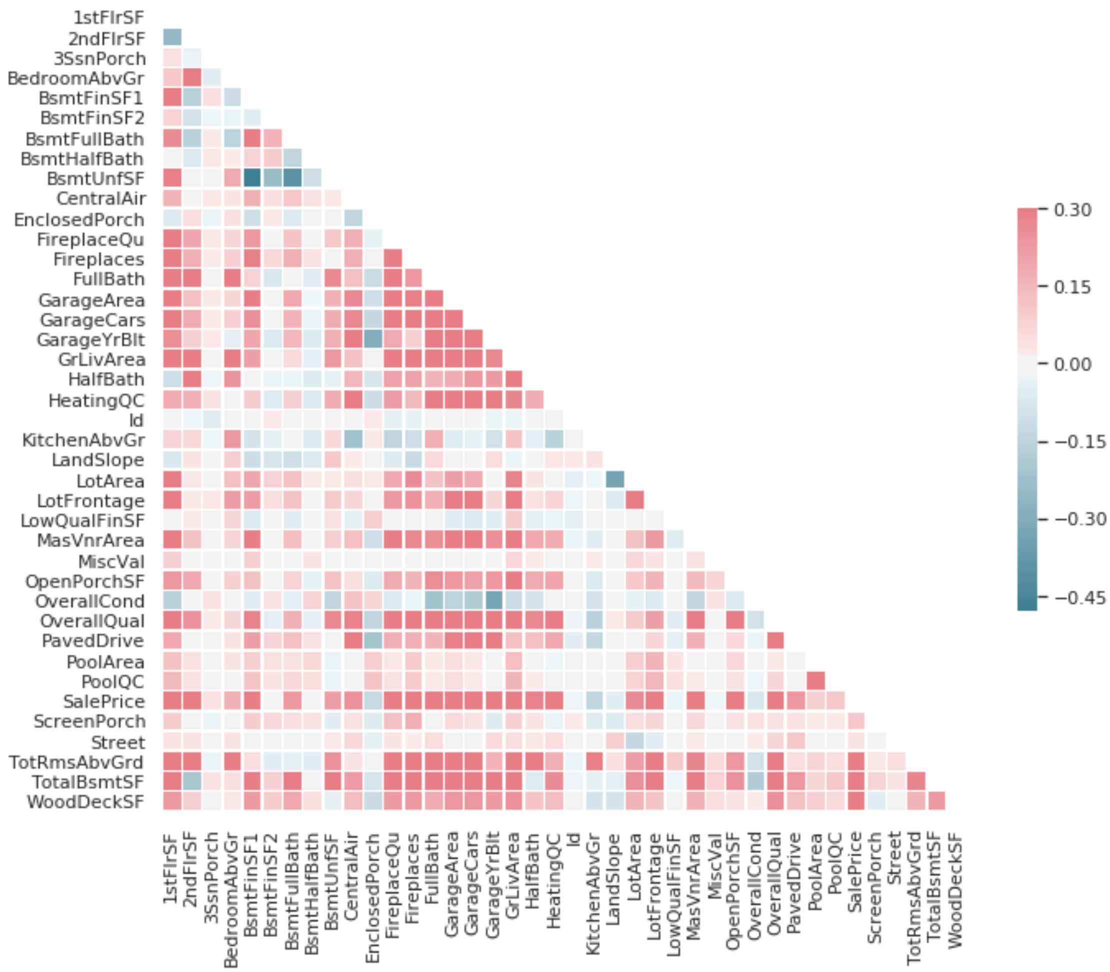
6.2.4 The second most important categorical variable; MSSubClass

6.2.5 Garage variables

6.2.6 Basement variables

```
In [45]: # 6.1
         #corrlation
         df_numeric = df.select_dtypes(include=np.number)

         corr1 = df_numeric.corr()
         f, ax = plt.subplots(figsize=(11, 9))
         mask = np.zeros_like(corr1, dtype=np.bool)
         mask[np.triu_indices_from(mask)] = True
         # Generate a custom diverging colormap
         cmap = sns.diverging_palette(220, 10, as_cmap=True)

         # Draw the heatmap with the mask and correct aspect ratio
         sns.heatmap(corr1, mask=mask, cmap=cmap, vmax=.3, center=0,
                     square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

```
Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2ca9da5eb8>
```

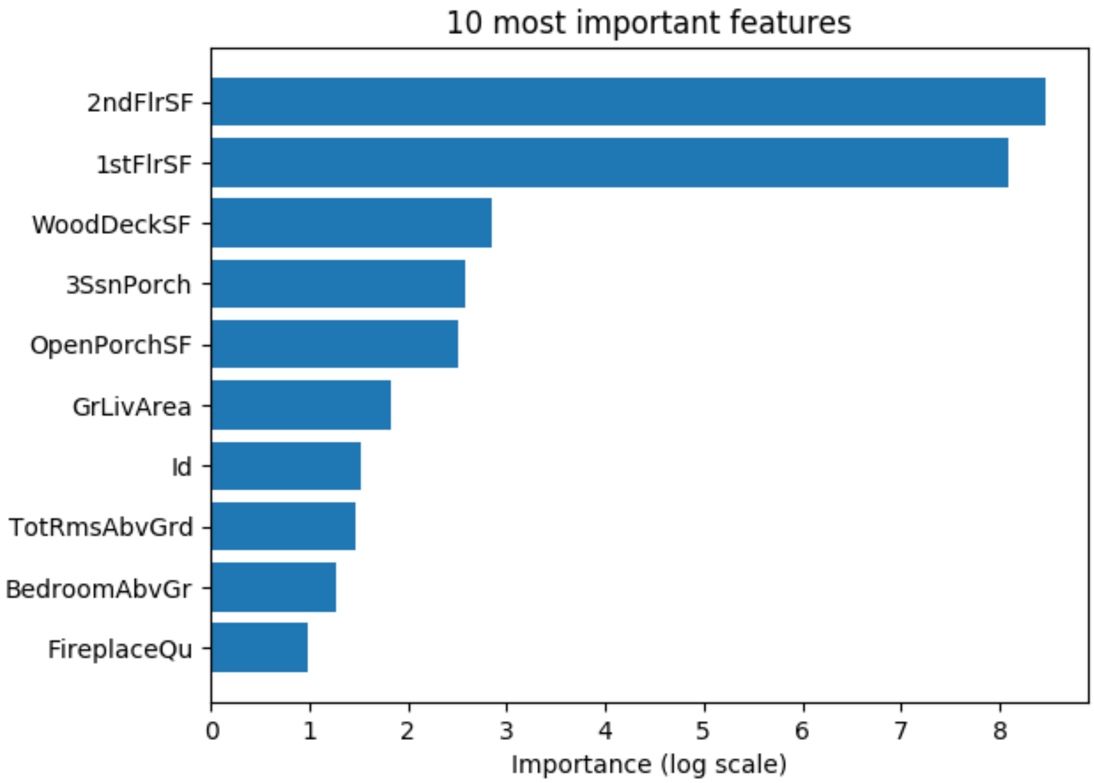



```
In [46]: #6.2 find variable importance
         #due to lack of time, I have not cleaned all the varibles as in the sample notebook
         #therefore, I am doning a random forest on the features that I have cleaned
         num_cols = list(df.select_dtypes(include=['int64']).columns)
         df3 = pd.DataFrame(df[num_cols[0]])

         for columns in num_cols:
             df3[columns] = df[columns]

         #random forest
         from sklearn.ensemble import RandomForestRegressor
         from sklearn.datasets import make_regression

         X = np.asmatrix(df3)
         y = np.asarray(df['SalePrice'])

         X, y = make_regression(n_features=28, n_informative=2,
```

```
                                random_state=0, shuffle=False)
        regr = RandomForestRegressor(max_depth=2, random_state=0,
                                     n_estimators=100)
        regr.fit(X, y)
        #find the 10 highest importance
        importances = regr.feature_importances_
        tmp = list(df3.columns)
        df_10 = pd.DataFrame(importances, columns=['importance'])
        df_10['feature'] = tmp
        df_10 = df_10.sort_values(by='importance', ascending=False).head(10)
```

```
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/ensemble/weight_boosting.py:29: Depreca
  from numpy.core.umath_tests import inner1d
```

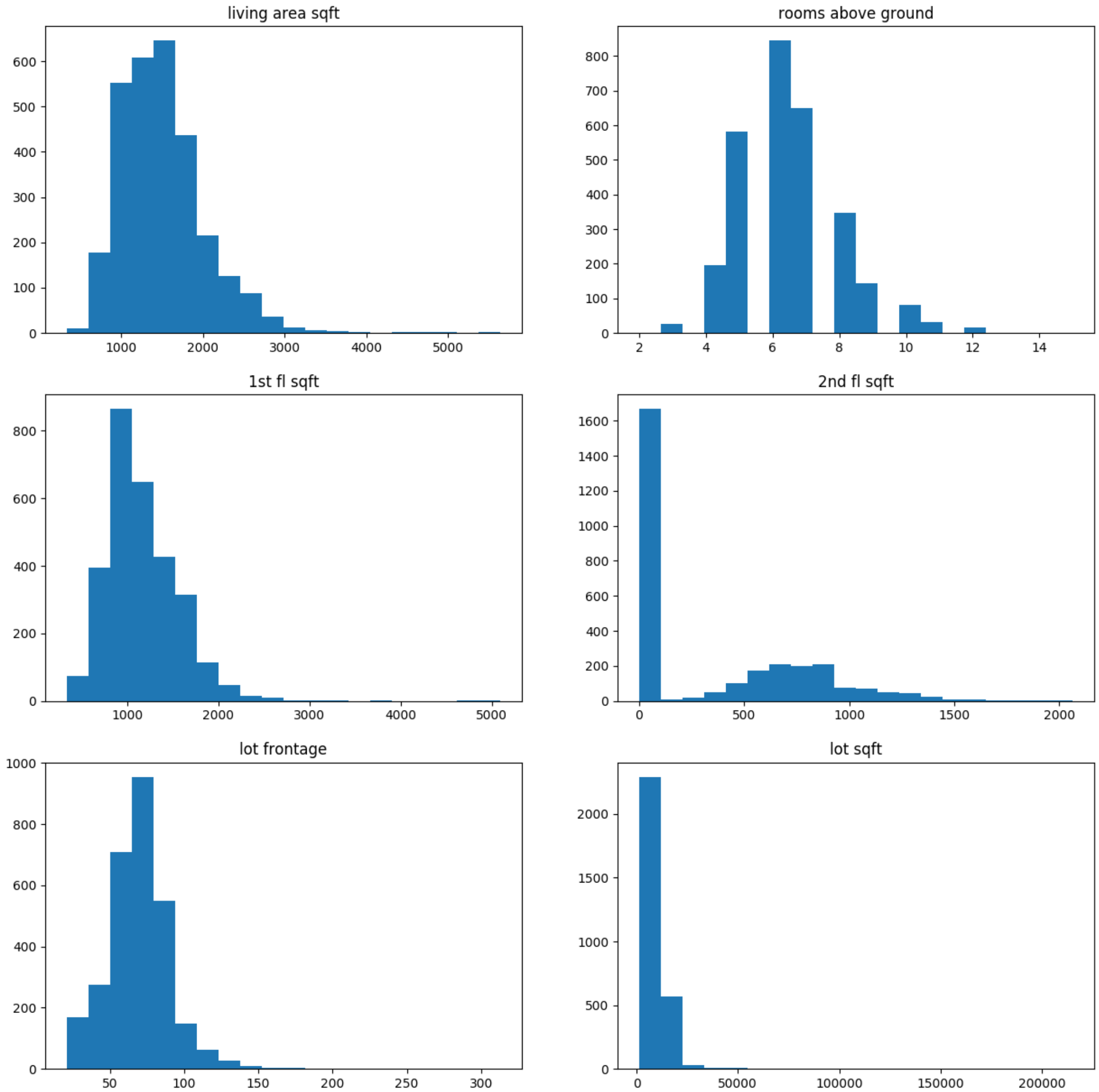
In [47]:
```
plt.rcdefaults()
fig, ax = plt.subplots()

feature = list(df_10['feature'])
y_pos = np.arange(len(feature))
importance = np.log(np.asarray(df_10['importance']))+9


ax.barh(y_pos, importance)
ax.set_yticks(y_pos)
ax.set_yticklabels(feature)
ax.invert_yaxis()
ax.set_xlabel('Importance (log scale)')
ax.set_title('10 most important features')

plt.show()
```




```
In [48]: # 6.2.1
         s1 = df['GrLivArea'];
         s2 = df['TotRmsAbvGrd'];
         s3 = df['1stFlrSF']
         s4 = df['2ndFlrSF']
         s5 = df['LotFrontage']
         s6 = df['LotArea']

         f, ((ax1, ax2), (ax3, ax4), (ax5, ax6)) = plt.subplots(3, 2, sharex=False, sharey=Fal
         f.set_figheight(15)
         f.set_figwidth(15)

         ax1.hist(s1, bins=20)
         ax1.set_title('living area sqft')
         ax2.hist(s2, bins=20)
         ax2.set_title('rooms above ground')
         ax3.hist(s3, bins=20)
         ax3.set_title('1st fl sqft')
         ax4.hist(s4, bins=20)
         ax4.set_title('2nd fl sqft')
         ax5.hist(s5, bins=20)
         ax5.set_title('lot frontage')
```

```
ax6.hist(s6, bins=20)
ax6.set_title('lot sqft')
```

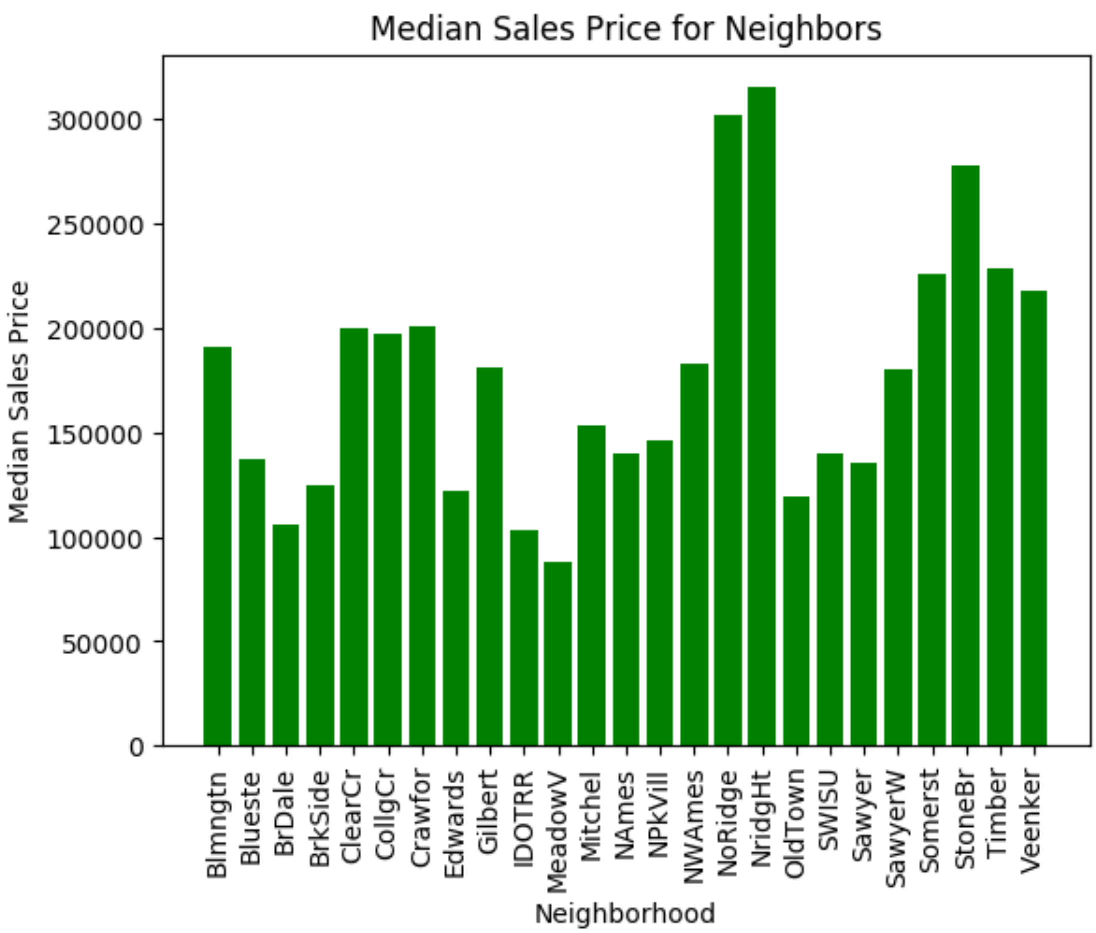
Out[48]: Text(0.5, 1.0, 'lot sqft')



In [49]:
```
#6.2.2 the most important categorical variable
#median sales price of different neighborhood
df4 = pd.DataFrame(df.groupby('Neighborhood')['SalePrice'].median())
```

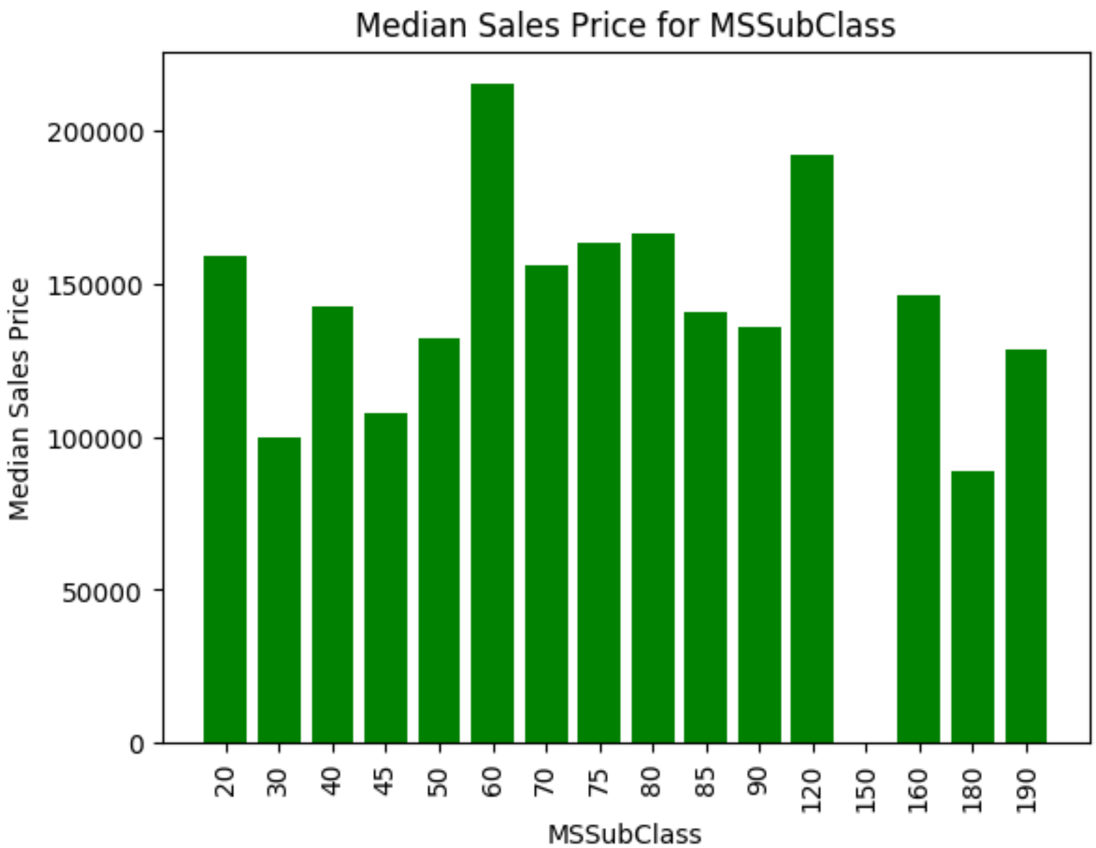
In [50]:
```
neighbor = np.asarray(df4.index)
price = np.asarray(df4['SalePrice'])

x_pos = [i for i, _ in enumerate(neighbor)]
```

```
plt.bar(x_pos,price,color='green')
plt.xlabel("Neighborhood")
plt.ylabel("Median Sales Price")
plt.title("Median Sales Price for Neighbors")

plt.xticks(x_pos, neighbor, rotation='vertical')

plt.show()
```




```
In [51]: #6.2.4 the MSSubClass
         df5 = pd.DataFrame(df.groupby('MSSubClass')['SalePrice'].median())
         ms = np.asarray(df5.index)
         price = np.asarray(df5['SalePrice'])

         x_pos = [i for i, _ in enumerate(ms)]

         plt.bar(x_pos,price,color='green')
         plt.xlabel("MSSubClass")
```

```
plt.ylabel("Median Sales Price")
plt.title("Median Sales Price for MSSubClass")

plt.xticks(x_pos, ms, rotation='vertical')

plt.show()
```



1.14 7 Feature engineering [5]

1.14.1 7.1 Total number of Bathrooms

1.14.2 7.2 Adding 'House Age', 'Remodeled (Yes/No)', and IsNew variables

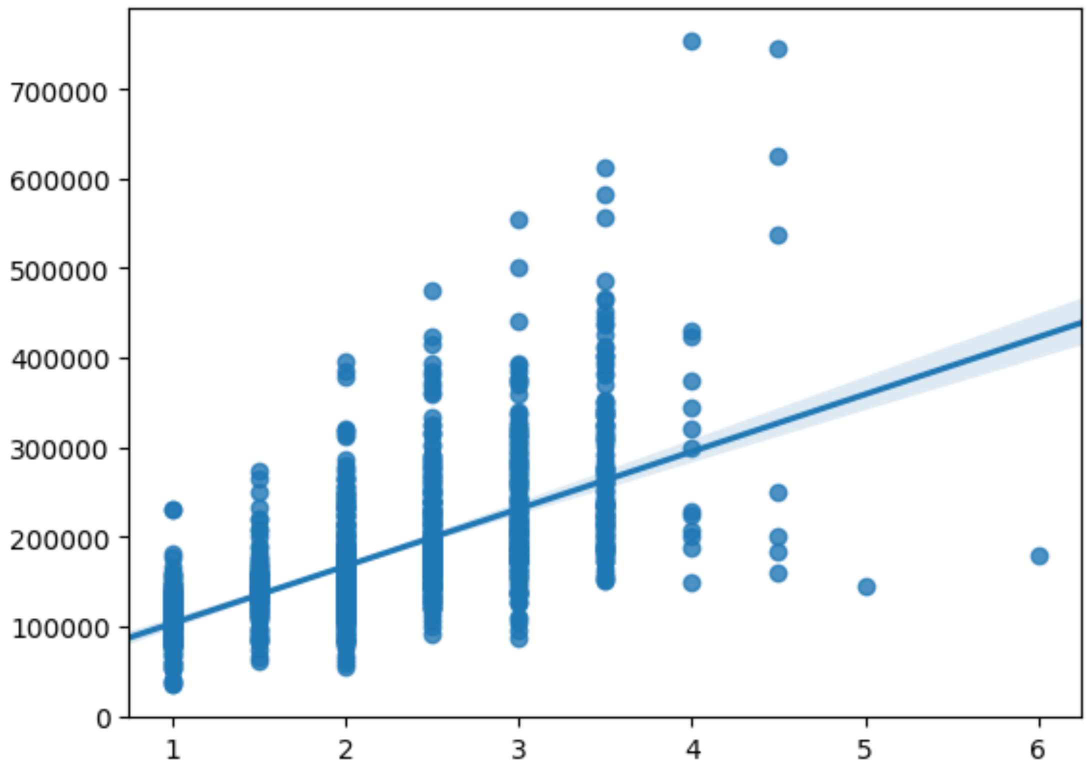
1.14.3 7.3 Binning Neighborhood

1.14.4 7.4 Total Square Feet

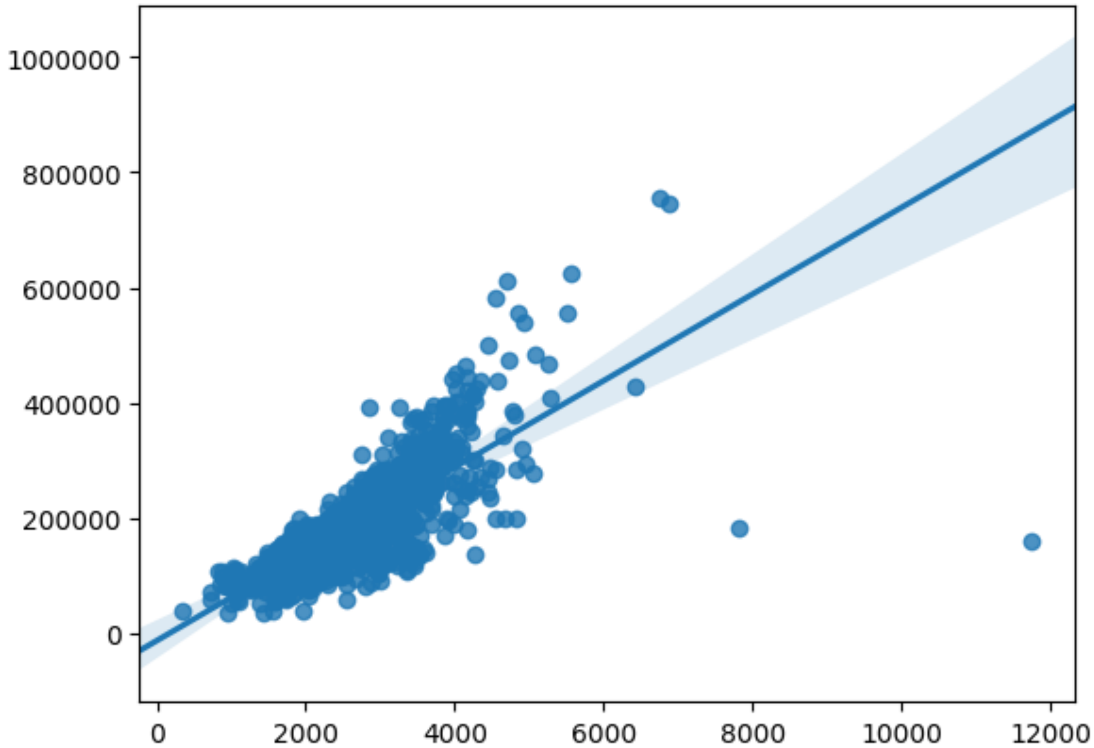
1.14.5 7.5 Consolidating Porch variables

In [52]:
```
#7.1
df['TotBathrooms'] = df['FullBath'] + df['HalfBath']*0.5 + df['BsmtHalfBath']*0.5 + d
ax = sns.regplot(x=np.asarray(df['TotBathrooms']), y=np.asarray(df["SalePrice"]))
```

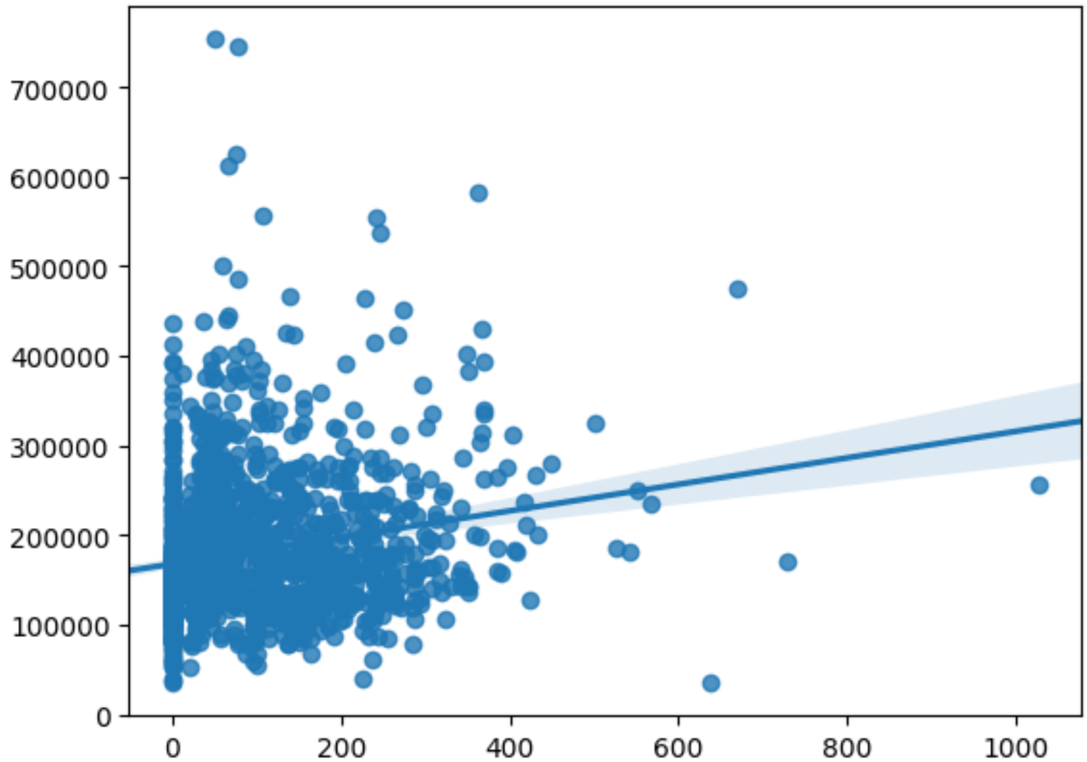
```
/home/shiyun/.local/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```




```
In [53]: #7.4 total square feet
         #as expected, total square feet is strongly correlated with sale price
         df['TotalSqFeet'] = df['GrLivArea'] + df['TotalBsmtSF']
         ax = sns.regplot(x=np.asarray(df['TotalSqFeet']), y=np.asarray(df["SalePrice"]))
```




```
In [54]: #7.5 total porch
         df['TotalPorchSF'] = df['OpenPorchSF'] + df['EnclosedPorch'] + df['3SsnPorch'] + df['S
         ax = sns.regplot(x=np.asarray(df['TotalPorchSF']), y=np.asarray(df["SalePrice"]))
```



1.15 8 Preparing data for modeling [20]

1.15.1 8.1 Dropping highly correlated variables

1.15.2 8.2 Removing outliers

1.15.3 8.3 PreProcessing predictor variables

8.3.1 Skewness and normalizing of the numeric predictors

8.3.2 One hot encoding the categorical variables

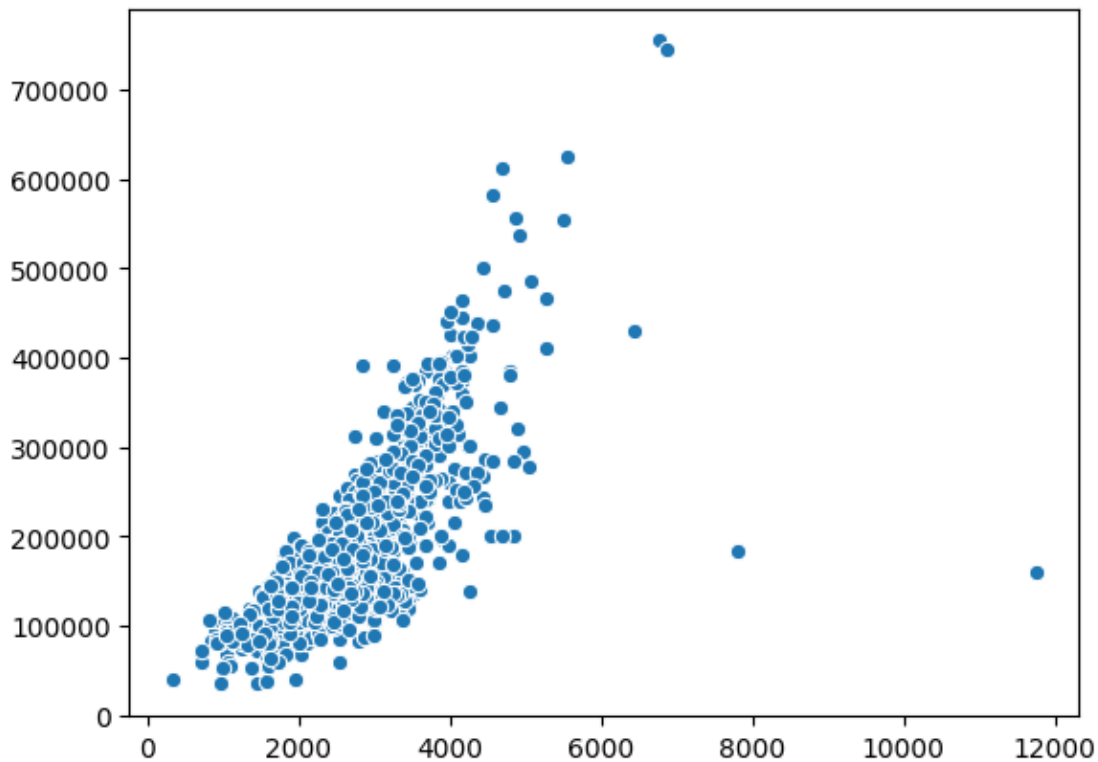
8.3.3 Removing levels with few or no observations in train or test

1.15.4 8.4 Dealing with skewness of response variable

1.15.5 8.5 Composing train and test sets

In [55]:
```
#8.1 use the correlation matrx to drop variables that are highly correlated
to_drop = ['YearRemodAdd', 'GarageYrBlt', 'GarageArea', 'GarageCond', 'TotalBsmtSF','
df = df.drop(columns=to_drop)
```

```
In [56]: #8.2 remove outliers
         #remove the two big houses with low prices, mannually
         ax = sns.scatterplot(x=np.asarray(df['TotalSqFeet']), y=np.asarray(df["SalePrice"]))
```




```
In [57]: #get the 2 largest sqft
         df.sort_values(by='TotalSqFeet', ascending=False).head(2)
```

```
Out[57]:       1stFlrSF  2ndFlrSF  3SsnPorch Alley  BedroomAbvGr BldgType BsmtCond  \
         1298      4692       950          0  None             3     1Fam       TA
         2549      5095         0          0  None             2     1Fam       TA

              BsmtExposure  BsmtFinSF2 BsmtFinType1      ...       ScreenPorch  Street  \
         1298           Gd         0.0          GLQ      ...                 0       1
         2549           Gd         0.0          GLQ      ...                 0       1

               TotRmsAbvGrd Utilities  WoodDeckSF  YearBuilt YrSold TotBathrooms  \
         1298            12    AllPub         214       2008   2008          4.5
         2549            15    AllPub         546       2008   2007          4.0

              TotalSqFeet  TotalPorchSF
         1298     11752.0           292
         2549     10190.0           484
```

```
[2 rows x 78 columns]
```

```
In [58]: df = df[df['TotalSqFeet'] < 5095]
```

```
In [59]: #8.3.1 skewness
         #first, normalize the data
         from sklearn import preprocessing
         tmp = df3.values
         min_max_scaler = preprocessing.MinMaxScaler()
         tmp_scaled = min_max_scaler.fit_transform(tmp)
         df_norm = pd.DataFrame(tmp_scaled)
```

```
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/utils/validation.py:475: DataConversio
  warnings.warn(msg, DataConversionWarning)
```

```
In [60]: #8.3.2 one hot encoding
         df_dummy = pd.get_dummies(df)
```

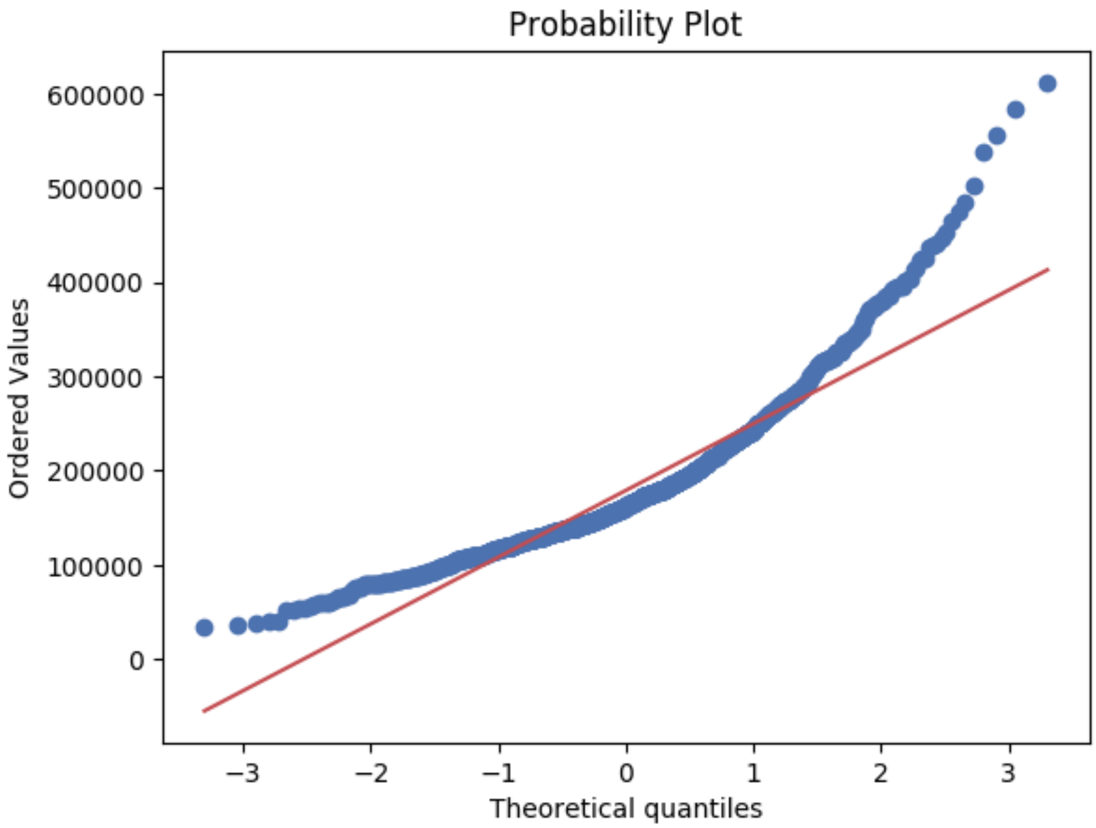
```
In [61]: #8.3.3 remove levels with few or no observation
         #check which values are absent in the test test
         col_not_in_test = test.isnull().sum().sort_values(ascending=False).head(10)
         col_not_in_test = np.asarray(col_not_in_test.index)
         col_not_in_test = [ 'MiscFeature', 'Alley', 'Fence', 'FireplaceQu',
                'LotFrontage', 'GarageQual',
                'GarageFinish']
```

```
In [62]: for col in col_not_in_test:
             df_notintest = df.drop(columns=col)
         df = df_notintest
```

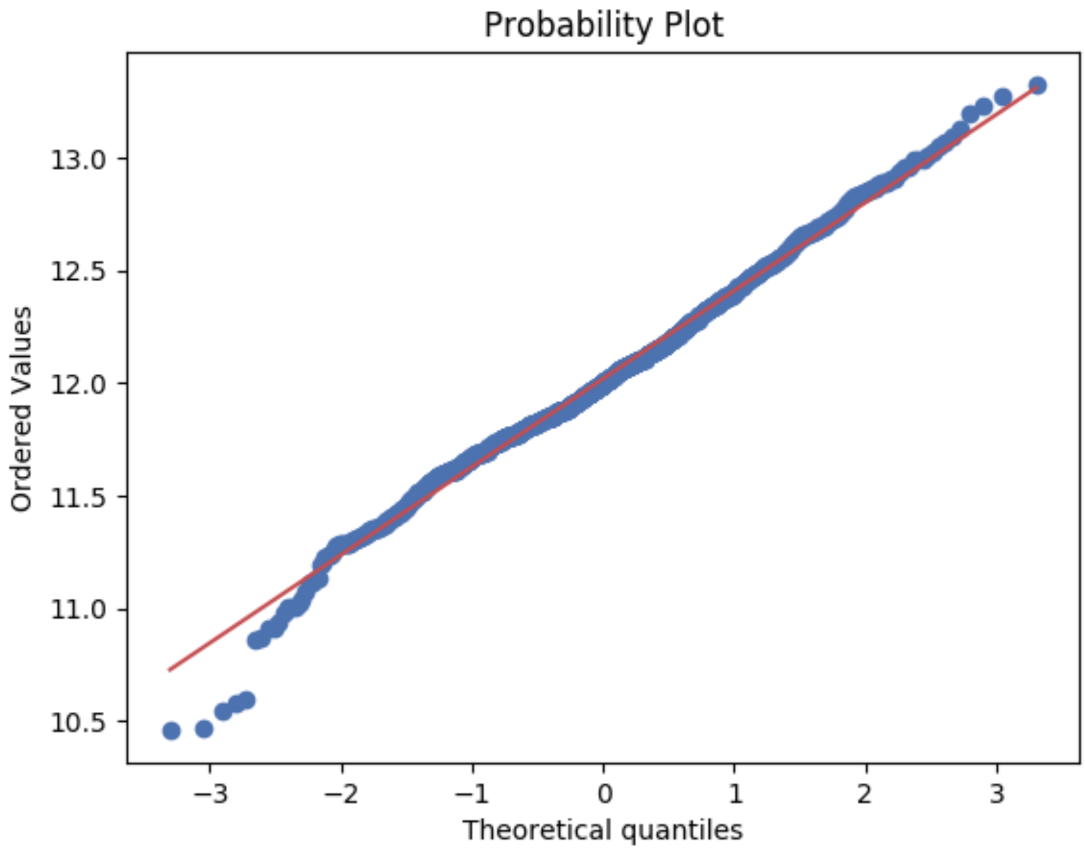
```
In [63]: df.shape
```

```
Out[63]: (2903, 77)
```

```
In [64]: #skewness & take log of saleprice to reduce skewness
         df['SalePrice'].kurtosis()
         from scipy import stats
         from matplotlib import pylab
         fig = plt.figure()
         #ax = fig.add_subplot(111)
         x = np.asarray(df['SalePrice'].dropna())
         res = stats.probplot(x, dist="norm", plot=pylab)
         pylab.show()
```




```
In [65]: df['SalePrice'] = df['SalePrice'].apply(np.log)
         fig = plt.figure()
         x = np.asarray(df['SalePrice'].dropna())
         res = stats.probplot(x, dist="norm", plot=pylab)
         pylab.show()
```



1.16 9 Modeling [20]

1.16.1 9.1 Lasso regression model

1.16.2 9.2 XGBoost model

1.16.3 9.3 Averaging predictions

In [66]:
```
#using the cleaned dataset
train_cleaned = pd.read_csv("eb_train1.csv")
train_cleaned.drop(columns='Unnamed: 0', inplace=True)
train_cleanedy = pd.read_csv("eb_train1_y.csv")
train_cleanedy.drop(columns='Unnamed: 0', inplace=True)

test_cleaned = pd.read_csv("eb_test1.csv")
test_cleaned.drop(columns='Unnamed: 0', inplace=True)
```

In [67]:
```
#training set
X = np.asmatrix(train_cleaned)
y = np.asarray(train_cleanedy)
```

```
In [68]: #test set
         X_test = np.asmatrix(test_cleaned)
```

```
In [69]: #9.1 lasso regression, parameter tuning with gridsearchcv
         from sklearn import linear_model
         from sklearn.model_selection import GridSearchCV
         import numpy as np

         parameters = {'alpha':np.logspace(-5,-2,30)}
         lasso = linear_model.Lasso(alpha="alpha")
         reg = GridSearchCV(lasso, parameters, cv=5)
         reg.fit(X, y)
```

```
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
```

```
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491
  ConvergenceWarning)
```

```
Out[69]: GridSearchCV(cv=5, error_score='raise',
                estimator=Lasso(alpha='alpha', copy_X=True, fit_intercept=True, max_iter=1000,
            normalize=False, positive=False, precompute=False, random_state=None,
            selection='cyclic', tol=0.0001, warm_start=False),
                fit_params=None, iid=True, n_jobs=1,
                param_grid={'alpha': array([1.00000e-05, 1.26896e-05, 1.61026e-05, 2.04336e-05
                3.29034e-05, 4.17532e-05, 5.29832e-05, 6.72336e-05, 8.53168e-05,
                1.08264e-04, 1.37382e-04, 1.74333e-04, 2.21222e-04, 2.80722e-04,
                3.56225e-04, 4.52035e-04, 5.73615e-04, 7.27895e-04, 9.23671e-04,
                1.17210e-03, 1.48735e-03, 1.88739e-03, 2.39503e-03, 3.03920e-03,
                3.85662e-03, 4.89390e-03, 6.21017e-03, 7.88046e-03, 1.00000e-02])},
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring=None, verbose=0)
```

```
In [71]: sorted(reg.cv_results_.keys())
```

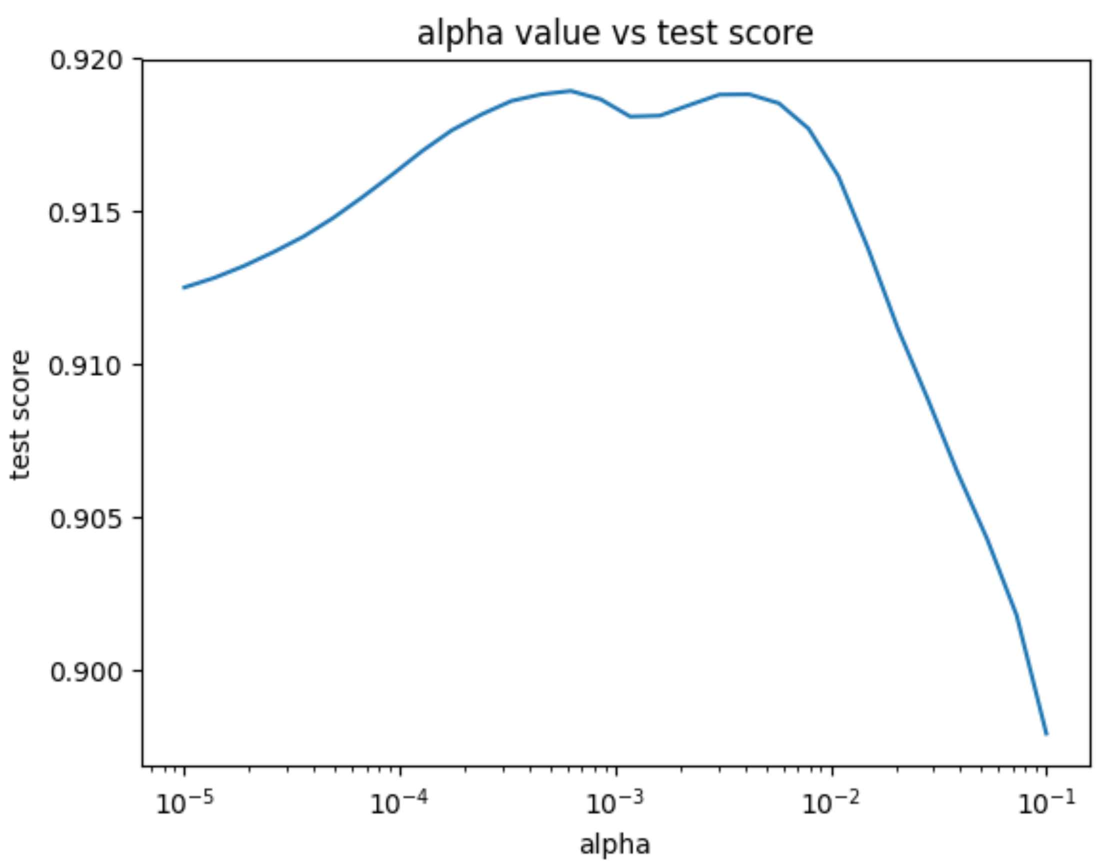
```
Out[71]: ['mean_fit_time',
          'mean_score_time',
          'mean_test_score',
          'mean_train_score',
          'param_alpha',
          'params',
          'rank_test_score',
          'split0_test_score',
          'split0_train_score',
```

```
 'split1_test_score',
 'split1_train_score',
 'split2_test_score',
 'split2_train_score',
 'split3_test_score',
 'split3_train_score',
 'split4_test_score',
 'split4_train_score',
 'std_fit_time',
 'std_score_time',
 'std_test_score',
 'std_train_score']
```

```
In [72]: test_score = reg.cv_results_['mean_test_score']
         alpha_choice = np.asarray(np.logspace(-5,-1, 30))
```

```
In [73]: #plot different test scores for different alpha choices
         #the x axis has taken log
         plt.semilogx(alpha_choice, test_score)
         plt.title('alpha value vs test score')
         plt.xlabel("alpha")
         plt.ylabel("test score")
```

```
Out[73]: Text(0, 0.5, 'test score')
```



In [74]:
```
#to find the best alpha that associates with the best score
test_score.max()
```

Out[74]: 0.9189270180233196

In [75]:
```
list(test_score).index(test_score.max())
```

Out[75]: 13

In [76]:
```
best_alpha = alpha_choice[13]
```

In [77]:
```
#use the best alpha to fit the train data, calculate the RMSE
from sklearn.metrics import mean_squared_error
clf = linear_model.Lasso(alpha = best_alpha)
clf.fit(X, y)
y_predict = clf.predict(X)
train_mse = mean_squared_error(y, y_predict)
import math
rmse = math.sqrt(train_mse)
rmse
```

Out[77]: 0.10456320721478549

In [78]:
```
lasso_pred = reg.predict(X_test)
lasso_pred = np.exp(lasso_pred)
lasso_pred
```

Out[78]:
```
array([113986.64840258, 162576.70269041, 179373.85741767, ...,
       173359.59922796, 120912.75740255, 218890.09226122])
```

In [79]:
```
#9.2 XGBoost
import xgboost as xgb
from xgboost.sklearn import XGBRegressor
from sklearn import cross_validation, metrics
from sklearn.grid_search import GridSearchCV

import matplotlib.pylab as plt
%matplotlib inline
from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 12, 4
```

```
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWarn
  "This module will be removed in 0.20.", DeprecationWarning)
/home/shiyun/.local/lib/python3.6/site-packages/sklearn/grid_search.py:42: DeprecationWarning:
  DeprecationWarning)
```

In [80]:
```
parameters = {
    'n_estimators': [1000],
    'learning_rate': [0.1, 0.05, 0.01],
    'max_depth': list(range(2, 7)),
    'gamma': [0],
    'colsample_bytree': [1],
    'min_child_weight': list(range(1, 6)),
    'subsample': [1]
}
```

In [82]:
```
model = XGBRegressor(silent=True, n_jobs=10)
reg = GridSearchCV(model, parameters, cv=5)
reg.fit(X, y)
```

Out[82]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=10, nthread=None, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'n_estimators': [1000], 'learning_rate': [0.1, 0.05, 0.01], 'max_de
       pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=0)
```

In [83]:
```
#find the best model
best_model = reg.best_estimator_
```

In [84]:
```
best_model
```

Out[84]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.05, max_delta_step=0,
       max_depth=2, min_child_weight=5, missing=None, n_estimators=1000,
       n_jobs=10, nthread=None, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [85]:
```
xgb_pred = best_model.predict(X_test)
```

In [86]:
```
xgb_pred = np.exp(xgb_pred)
xgb_pred
```

Out[86]:
```
array([116424.695, 164930.27 , 188358.78 , ..., 172915.   , 118149.67 ,
       210353.58 ], dtype=float32)
```

In [87]:
```
#9.3 averaging predictions
sub_avg = (lasso_pred*2 + xgb_pred)/3
submission = pd.DataFrame(test['Id'], columns=['Id'])
submission['SalePrice'] = sub_avg
```

In [88]:
```
submission.to_csv("Submission.csv", index=False)
```

END SOLUTION

In [ ]: