Branch: master ▾ Find file Copy path

IS540-Project-2 / Decision Tree, Random Forest & SVM.ipynb

danielle-ezzo Update DE - 1 May 2018

8ae8234 on May 1, 2018

1 contributor

Download History

2.01 MB

Competition #2: Data Audit Report

Research Question & Goal:

Is it possible to predict the sale price for each house in our data set? It is our job to predict the sales price for each house. For each Id in the test set, we must predict the value of the SalePrice variable.

Business Understanding:

If you ask a home buyer to describe their dream house, they probably won't begin with the height of the basement ceiling or the proximity to an east-west railroad. But the dataset in this analysis proves that much more influences price negotiations than the number of bedrooms or whether there's a white-picket fence.

A house is a building that functions as a home, ranging from simple dwellings such as rudimentary huts of nomadic tribes and the improvised shacks in shantytowns, to complex, fixed structures of wood, brick, concrete or other materials containing plumbing, ventilation and electrical systems. Houses use a range of different roofing systems to keep precipitation such as rain from getting into the dwelling space. Houses may have doors or locks to secure the dwelling space and protect its inhabitants and contents from burglars or other trespassers. Most conventional modern houses in Western cultures will contain one or more bedrooms and bathrooms, a kitchen or cooking area, and a living room. A house may have a separate dining room, or the eating area may be integrated into another room. Some large houses in North America also have a recreation room.

With all the various ways a house can be constructed, and with all the different materials that can be used in its construction, how can one accurately determine the price of a house? Often when we refer to price we refer to sale price of a house. Architecture, foundations, floor space, and number of rooms all could play a part in determining the sale price of a house. The dataset that has been gathered for the purposes of this report contains 81 variables - 1 ID variable, 1 Target variable (SalePrice) and 79 Predictor variables, all listed below.

- MSSubClass: The building class
- MSZoning: The general zoning classification
- LotFrontage: Linear feet of street connected to property

- LotArea: Lot size in square feet
- Street: Type of road access
- Alley: Type of alley access
- LotShape: General shape of property
- LandContour: Flatness of the property
- Utilities: Type of utilities available
- LotConfig: Lot configuration
- LandSlope: Slope of property
- Neighborhood: Physical locations within Ames city limits
- Condition1: Proximity to main road or railroad
- Condition2: Proximity to main road or railroad (if a second is present)
- BldgType: Type of dwelling
- HouseStyle: Style of dwelling
- OverallQual: Overall material and finish quality
- OverallCond: Overall condition rating
- YearBuilt: Original construction date
- YearRemodAdd: Remodel date
- RoofStyle: Type of roof
- RoofMatl: Roof material
- Exterior1st: Exterior covering on house
- Exterior2nd: Exterior covering on house (if more than one material)
- MasVnrType: Masonry veneer type
- MasVnrArea: Masonry veneer area in square feet
- ExterQual: Exterior material quality
- ExterCond: Present condition of the material on the exterior
- Foundation: Type of foundation
- BsmtQual: Height of the basement
- BsmtCond: General condition of the basement
- BsmtExposure: Walkout or garden level basement walls
- BsmtFinType1: Quality of basement finished area
- BsmtFinSF1: Type 1 finished square feet
- BsmtFinType2: Quality of second finished area (if present)
- BsmtFinSF2: Type 2 finished square feet
- BsmtUnfSF: Unfinished square feet of basement area
- TotalBsmtSF: Total square feet of basement area
- Heating: Type of heating
- HeatingQC: Heating quality and condition
- CentralAir: Central air conditioning
- Electrical: Electrical system
- 1stFlrSF: First Floor square feet
- 2ndFlrSF: Second floor square feet
- LowQualFinSF: Low quality finished square feet (all floors)
- GrLivArea: Above grade (ground) living area square feet
- BsmtFullBath: Basement full bathrooms
- BsmtHalfBath: Basement half bathrooms

- BsmtHalfBath: Basement half bathrooms
- FullBath: Full bathrooms above grade
- HalfBath: Half baths above grade
- Bedroom: Number of bedrooms above basement level
- Kitchen: Number of kitchens
- KitchenQual: Kitchen quality
- TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)
- Functional: Home functionality rating
- Fireplaces: Number of fireplaces
- FireplaceQu: Fireplace quality
- GarageType: Garage location
- GarageYrBlt: Year garage was built
- GarageFinish: Interior finish of the garage
- GarageCars: Size of garage in car capacity
- GarageArea: Size of garage in square feet
- GarageQual: Garage quality
- GarageCond: Garage condition
- PavedDrive: Paved driveway
- WoodDeckSF: Wood deck area in square feet
- OpenPorchSF: Open porch area in square feet
- EnclosedPorch: Enclosed porch area in square feet
- 3SsnPorch: Three season porch area in square feet
- ScreenPorch: Screen porch area in square feet
- PoolArea: Pool area in square feet
- PoolQC: Pool quality
- Fence: Fence quality
- MiscFeature: Miscellaneous feature not covered in other categories
- MiscVal: Value of miscellaneous feature
- MoSold: Month Sold
- YrSold: Year Sold
- SaleType: Type of sale
- SaleCondition: Condition of sale

Data Understanding

Training Set

Our data set is divided into two parts, a training set and a testing set. To begin, we examine the training set. The data set contains 81 columns and 1460 rows. Our variables have the following breakdown: 36 are quantitative, 43 categorical and then Id and SalePrice are viewed separately. ID offers no predicitve value and SalePrice is our target variable.

In [1]:

```
# Importing useful packages
```

```
import numpy as np
from scipy import stats
import pandas as pd
import sklearn as sk
import seaborn as sb
import datetime as dt
import pylab
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

%matplotlib inline
from sklearn import datasets
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegress
ion
import statsmodels.api as sm

from sklearn import metrics
from sklearn import model_selection
from sklearn.model_selection import cross_val_sc
ore

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.cross_validation import train_test_
split
from sklearn.cluster import KMeans
from sklearn import metrics

from sklearn.metrics import accuracy_score
from sklearn import tree
from sklearn.metrics import precision_recall_fsc
ore_support


# Read in Data file and define NaN values
housetrain = pd.read_csv("train.csv",header=0,na
_values='None')
housetrain.MSSubClass = housetrain.MSSubClass.as
type(str)
```

```
C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\sklearn\cross_validat
ion.py:44: DeprecationWarning: This module was d
eprecated in version 0.18 in favor of the model_
selection module into which all the refactored c
lasses and functions are moved. Also note that t
he interface of the new CV iterators are differe
nt from that of this module. This module will be
removed in 0.20.
  "This module will be removed in 0.20.", Deprec
ationWarning)
```

After reading in the data into our python workspace, we had to change one of our integer variables to be a string for ease as it was not an ordinal categorical variarble. We then printed out our data types to make sure we were happy with them.

In [2]:

```
# Print types
pd.set_option('display.max_rows', 82)
print(housetrain.dtypes)
```

```
Id                int64
MSSubClass       object
MSZoning         object
LotFrontage     float64
LotArea           int64
Street           object
Alley            object
LotShape         object
LandContour      object
Utilities        object
LotConfig        object
LandSlope        object
Neighborhood     object
Condition1       object
Condition2       object
BldgType         object
HouseStyle       object
OverallQual       int64
OverallCond       int64
YearBuilt         int64
YearRemodAdd      int64
RoofStyle        object
RoofMatl         object
Exterior1st      object
Exterior2nd      object
MasVnrType       object
MasVnrArea      float64
ExterQual        object
ExterCond        object
Foundation       object
BsmtQual         object
BsmtCond         object
BsmtExposure     object
BsmtFinType1     object
BsmtFinSF1        int64
BsmtFinType2     object
BsmtFinSF2        int64
BsmtUnfSF         int64
TotalBsmtSF       int64
Heating          object
HeatingQC        object
CentralAir       object
Electrical       object
1stFlrSF          int64
2ndFlrSF          int64
LowQualFinSF      int64
GrLivArea         int64
BsmtFullBath      int64
BsmtHalfBath      int64
FullBath          int64
HalfBath          int64
BedroomAbvGr      int64
```

```
KitchenAbvGr       int64
KitchenQual       object
TotRmsAbvGrd       int64
Functional        object
Fireplaces         int64
FireplaceQu       object
GarageType        object
GarageYrBlt      float64
GarageFinish      object
GarageCars         int64
GarageArea         int64
GarageQual        object
GarageCond        object
PavedDrive        object
WoodDeckSF         int64
OpenPorchSF        int64
EnclosedPorch      int64
3SsnPorch          int64
ScreenPorch        int64
PoolArea           int64
PoolQC            object
Fence             object
MiscFeature       object
MiscVal            int64
MoSold             int64
YrSold             int64
SaleType          object
SaleCondition     object
SalePrice          int64
dtype: object
```

Next, we perform a data describe to see the summary statistics of our data. As we can see below, some of our data has missing values.

In [3]:

```
# Data describe
pd.set_option('display.max_columns', 500)
print(housetrain.describe())
```

```
                Id  LotFrontage        LotArea  
OverallQual  OverallCond  \
count  1460.000000  1201.000000    1460.000000  
1460.000000  1460.000000  
mean    730.500000    70.049958   10516.828082  
6.099315     5.575342  
std     421.610009    24.284752    9981.264932  
1.382997     1.112799  
min       1.000000    21.000000    1300.000000  
1.000000     1.000000  
25%     365.750000    59.000000    7553.500000  
5.000000     5.000000  
50%     730.500000    69.000000    9478.500000  
6.000000     5.000000  
75%    1095.250000    80.000000   11601.500000  
7.000000     6.000000  
```

```
max    1460.000000   313.000000  215245.000000
10.000000     9.000000

         YearBuilt  YearRemodAdd   MasVnrArea
BsmtFinSF1   BsmtFinSF2  \
count  1460.000000   1460.000000  1452.000000  1
460.000000  1460.000000
mean   1971.267808   1984.865753   103.685262
443.639726    46.549315
std      30.202904     20.645407   181.066207
456.098091   161.319273
min    1872.000000   1950.000000     0.000000
0.000000     0.000000
25%    1954.000000   1967.000000     0.000000
0.000000     0.000000
50%    1973.000000   1994.000000     0.000000
383.500000     0.000000
75%    2000.000000   2004.000000   166.000000
712.250000     0.000000
max    2010.000000   2010.000000  1600.000000  5
644.000000  1474.000000

         BsmtUnfSF  TotalBsmtSF     1stFlrSF
2ndFlrSF  LowQualFinSF  \
count  1460.000000  1460.000000  1460.000000  14
60.000000   1460.000000
mean    567.240411  1057.429452  1162.626712   3
46.992466      5.844521
std     441.866955   438.705324   386.587738   4
36.528436     48.623081
min       0.000000     0.000000   334.000000
0.000000      0.000000
25%     223.000000   795.750000   882.000000
0.000000      0.000000
50%     477.500000   991.500000  1087.000000
0.000000      0.000000
75%     808.000000  1298.250000  1391.250000   7
28.000000      0.000000
max    2336.000000  6110.000000  4692.000000  20
65.000000    572.000000

         GrLivArea  BsmtFullBath  BsmtHalfBath
FullBath     HalfBath  \
count  1460.000000   1460.000000   1460.000000
1460.000000  1460.000000
mean   1515.463699      0.425342      0.057534
1.565068     0.382877
std     525.480383      0.518911      0.238753
0.550916     0.502885
min     334.000000      0.000000      0.000000
0.000000     0.000000
25%    1129.500000      0.000000      0.000000
1.000000     0.000000
50%    1464.000000      0.000000      0.000000
2.000000     0.000000
75%    1776.750000      1.000000      0.000000
2.000000     1.000000
max    5642.000000      3.000000      2.000000
```

```
3.000000     2.000000

       BedroomAbvGr  KitchenAbvGr  TotRmsAbvGrd
Fireplaces  GarageYrBlt  \
count   1460.000000   1460.000000   1460.000000
1460.000000  1379.000000
mean       2.866438      1.046575      6.517808
0.613014  1978.506164
std        0.815778      0.220338      1.625393
0.644666    24.689725
min        0.000000      0.000000      2.000000
0.000000  1900.000000
25%        2.000000      1.000000      5.000000
0.000000  1961.000000
50%        3.000000      1.000000      6.000000
1.000000  1980.000000
75%        3.000000      1.000000      7.000000
1.000000  2002.000000
max        8.000000      3.000000     14.000000
3.000000  2010.000000

        GarageCars   GarageArea   WoodDeckSF  Op
enPorchSF  EnclosedPorch  \
count  1460.000000  1460.000000  1460.000000  14
60.000000    1460.000000
mean      1.767123   472.980137    94.244521
46.660274       21.954110
std       0.747315   213.804841   125.338794
66.256028       61.119149
min       0.000000     0.000000     0.000000
0.000000       0.000000
25%       1.000000   334.500000     0.000000
0.000000       0.000000
50%       2.000000   480.000000     0.000000
25.000000       0.000000
75%       2.000000   576.000000   168.000000
68.000000       0.000000
max       4.000000  1418.000000   857.000000   5
47.000000     552.000000

         3SsnPorch  ScreenPorch     PoolArea
MiscVal       MoSold  \
count  1460.000000  1460.000000  1460.000000   1
460.000000  1460.000000
mean      3.409589    15.060959     2.758904
43.489041     6.321918
std      29.317331    55.757415    40.177307
496.123024     2.703626
min       0.000000     0.000000     0.000000
0.000000     1.000000
25%       0.000000     0.000000     0.000000
0.000000     5.000000
50%       0.000000     0.000000     0.000000
0.000000     6.000000
75%       0.000000     0.000000     0.000000
0.000000     8.000000
max     508.000000   480.000000   738.000000  15
500.000000    12.000000
```

```
            YrSold      SalePrice
count  1460.000000    1460.000000
mean   2007.815753  180921.195890
std       1.328095   79442.502883
min    2006.000000   34900.000000
25%    2007.000000  129975.000000
50%    2008.000000  163000.000000
75%    2009.000000  214000.000000
max    2010.000000  755000.000000
```

Summary statistics are important to observe obvious outliers and initial trends.

- Some of our variables contain missing data. This is by and large due to the formatting of the data in its use of "NA" to show when a house doesn't contain a feature. Nonetheless, it was decided to use it as missing initially to investigate if any variables contained imbalances due to missing data. Additionally, some variables contain all records (1460) but have zero as the minimum. Based on our analysis, this is more than likely due to the fact the house doesn't have this feature. For example, if we look at *TotBsmtSF*, which is the total square feet of the basement, we see that it is missing no records but has zero as a minimum. This more than likely means that the house does not have a basement.
- We notice on average, there is more unfinished basement space than finished basement space.
- There is on average 200 square feet less space upstairs than downstairs in houses. This makes sense as some homes don't have a complete second floor, and most houses are not built as a perfect square but reduce size on the second floor for structural requirements.
- Some of our summary statistic variables are actually ordinal data so their summary statistics do not reveal much other than that they have no erroneous values (*OverallQual, OverallCond, YearBuilt, YearRemodAdd, MasVnrArea, GarageYrBlt, MoSold, YrSold*)

Next, we look to quantify the missingness of our data.

In [4]:

```
# Get numeric value to missing features
for i in range(len(housetrain.columns)):
    j = housetrain.columns[i]
    miss=((1460-housetrain[str(j)].count())/1460
)*100
    print("The missingness of variable {}".forma
t(j))
    print("{0:.2f}%".format(miss))
```

The missingness of variable Id

```
The missingness of variable Id
0.00%
The missingness of variable MSSubClass
0.00%
The missingness of variable MSZoning
0.00%
The missingness of variable LotFrontage
17.74%
The missingness of variable LotArea
0.00%
The missingness of variable Street
0.00%
The missingness of variable Alley
93.77%
The missingness of variable LotShape
0.00%
The missingness of variable LandContour
0.00%
The missingness of variable Utilities
0.00%
The missingness of variable LotConfig
0.00%
The missingness of variable LandSlope
0.00%
The missingness of variable Neighborhood
0.00%
The missingness of variable Condition1
0.00%
The missingness of variable Condition2
0.00%
The missingness of variable BldgType
0.00%
The missingness of variable HouseStyle
0.00%
The missingness of variable OverallQual
0.00%
The missingness of variable OverallCond
0.00%
The missingness of variable YearBuilt
0.00%
The missingness of variable YearRemodAdd
0.00%
The missingness of variable RoofStyle
0.00%
The missingness of variable RoofMatl
0.00%
The missingness of variable Exterior1st
0.00%
The missingness of variable Exterior2nd
0.00%
The missingness of variable MasVnrType
59.73%
The missingness of variable MasVnrArea
0.55%
The missingness of variable ExterQual
0.00%
The missingness of variable ExterCond
0.00%
```

```
The missingness of variable Foundation
0.00%
The missingness of variable BsmtQual
2.53%
The missingness of variable BsmtCond
2.53%
The missingness of variable BsmtExposure
2.60%
The missingness of variable BsmtFinType1
2.53%
The missingness of variable BsmtFinSF1
0.00%
The missingness of variable BsmtFinType2
2.60%
The missingness of variable BsmtFinSF2
0.00%
The missingness of variable BsmtUnfSF
0.00%
The missingness of variable TotalBsmtSF
0.00%
The missingness of variable Heating
0.00%
The missingness of variable HeatingQC
0.00%
The missingness of variable CentralAir
0.00%
The missingness of variable Electrical
0.07%
The missingness of variable 1stFlrSF
0.00%
The missingness of variable 2ndFlrSF
0.00%
The missingness of variable LowQualFinSF
0.00%
The missingness of variable GrLivArea
0.00%
The missingness of variable BsmtFullBath
0.00%
The missingness of variable BsmtHalfBath
0.00%
The missingness of variable FullBath
0.00%
The missingness of variable HalfBath
0.00%
The missingness of variable BedroomAbvGr
0.00%
The missingness of variable KitchenAbvGr
0.00%
The missingness of variable KitchenQual
0.00%
The missingness of variable TotRmsAbvGrd
0.00%
The missingness of variable Functional
0.00%
The missingness of variable Fireplaces
0.00%
The missingness of variable FireplaceQu
47.26%
```

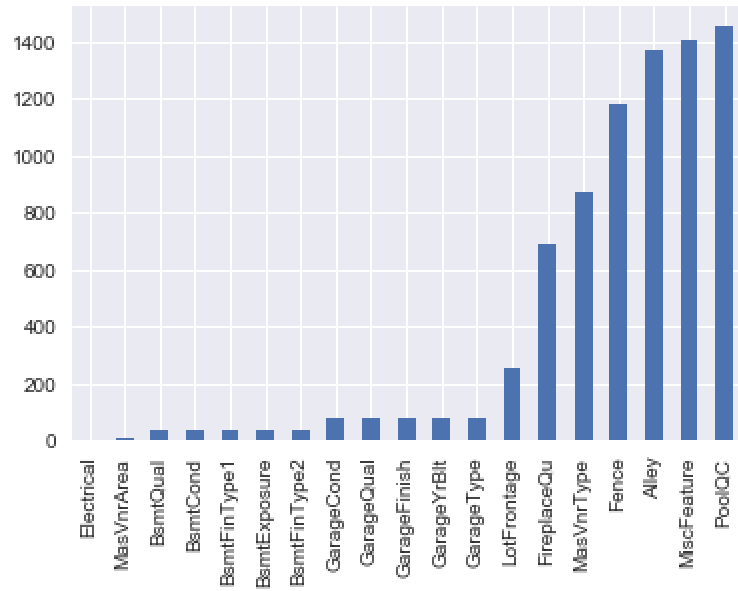
```
The missingness of variable GarageType
5.55%
The missingness of variable GarageYrBlt
5.55%
The missingness of variable GarageFinish
5.55%
The missingness of variable GarageCars
0.00%
The missingness of variable GarageArea
0.00%
The missingness of variable GarageQual
5.55%
The missingness of variable GarageCond
5.55%
The missingness of variable PavedDrive
0.00%
The missingness of variable WoodDeckSF
0.00%
The missingness of variable OpenPorchSF
0.00%
The missingness of variable EnclosedPorch
0.00%
The missingness of variable 3SsnPorch
0.00%
The missingness of variable ScreenPorch
0.00%
The missingness of variable PoolArea
0.00%
The missingness of variable PoolQC
99.52%
The missingness of variable Fence
80.75%
The missingness of variable MiscFeature
96.30%
The missingness of variable MiscVal
0.00%
The missingness of variable MoSold
0.00%
The missingness of variable YrSold
0.00%
The missingness of variable SaleType
0.00%
The missingness of variable SaleCondition
0.00%
The missingness of variable SalePrice
0.00%
```

In [5]:

```
missing = housetrain.isnull().sum()
missing = missing[missing > 0]
missing.sort_values(inplace=True)
missing.plot.bar()
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xc355
be0>
```



The most obvious example of missing is *PoolQC*. Looking at the graph it has the most number of missing values, as most houses in Ames do not have a pool on the property. Looking at the main culprits of missing values, we actaully see it makes sense that these variables have so many missing. *MiscFeature* is for features like tennis courts, second garages, elevators.Not many families can afford these types of add ons to their home, so the missingness of this variable makes sense. For the moment, it was decided to keep them in our dataset, as the few houses they do affect would see a dramatic increase in their sale price due to these features. Anything with around 50% of the data missing should be removed from further analysis, meaning we remove the following variables:

- *PoolQC*
- *MiscFeature*
- *Alley*
- *Fence*
- *MasVnrType*
- *FireplaceQu*

However, despite the large number of missing values, these could be construed as rare noise so we will keep them for the moment. Later, we will use Principal Component Analysis and let PCA decide which variables should be kept.

Imputation

As we saw, the biggest culprits of our missing data have perfectly logical reasons behind it. With that in mind, it was decided to fill in the NA values using either string representations of what was really going on (i.e. no pool) or zero.

In [6]:

```
# Alley : data description says NA means "no all
ey access"
housetrain.loc[:, "Alley"] = housetrain.loc[:,
"Alley"].fillna("None")
# BedroomAbvGr : NA most likely means 0
housetrain.loc[:, "BedroomAbvGr"] = housetrain.l
oc[:, "BedroomAbvGr"].fillna(0)
# BsmtQual etc : data description says NA for ba
sement features is "no basement"
housetrain.loc[:, "BsmtQual"] = housetrain.loc
[:, "BsmtQual"].fillna("No")
housetrain.loc[:, "BsmtCond"] = housetrain.loc
[:, "BsmtCond"].fillna("No")
housetrain.loc[:, "BsmtExposure"] = housetrain.l
oc[:, "BsmtExposure"].fillna("No")
housetrain.loc[:, "BsmtFinType1"] = housetrain.l
oc[:, "BsmtFinType1"].fillna("No")
housetrain.loc[:, "BsmtFinType2"] = housetrain.l
oc[:, "BsmtFinType2"].fillna("No")
housetrain.loc[:, "BsmtFullBath"] = housetrain.l
oc[:, "BsmtFullBath"].fillna(0)
housetrain.loc[:, "BsmtHalfBath"] = housetrain.l
oc[:, "BsmtHalfBath"].fillna(0)
housetrain.loc[:, "BsmtUnfSF"] = housetrain.loc
[:, "BsmtUnfSF"].fillna(0)
# CentralAir : NA most likely means No
housetrain.loc[:, "CentralAir"] = housetrain.loc
[:, "CentralAir"].fillna("N")
# Condition : NA most likely means Normal
housetrain.loc[:, "Condition1"] = housetrain.loc
[:, "Condition1"].fillna("Norm")
housetrain.loc[:, "Condition2"] = housetrain.loc
[:, "Condition2"].fillna("Norm")
# EnclosedPorch : NA most likely means no enclos
ed porch
housetrain.loc[:, "EnclosedPorch"] = housetrain.
loc[:, "EnclosedPorch"].fillna(0)
# External stuff : NA most likely means average
housetrain.loc[:, "ExterCond"] = housetrain.loc
[:, "ExterCond"].fillna("TA")
housetrain.loc[:, "ExterQual"] = housetrain.loc
[:, "ExterQual"].fillna("TA")
# Fence : data description says NA means "no fen
ce"
housetrain.loc[:, "Fence"] = housetrain.loc[:,
"Fence"].fillna("No")
# FireplaceQu : data description says NA means
 "no fireplace"
housetrain.loc[:, "FireplaceQu"] = housetrain.lo
c[:, "FireplaceQu"].fillna("No")
housetrain.loc[:, "Fireplaces"] = housetrain.loc
[:, "Fireplaces"].fillna(0)
# Functional : data description says NA means ty
pical
housetrain.loc[:, "Functional"] = housetrain.loc
[:, "Functional"].fillna("Typ")
# GarageType etc : data description says NA for
```

```
# GarageType etc : data description says NA for
 garage features is "no garage"
housetrain.loc[:, "GarageType"] = housetrain.loc
[:, "GarageType"].fillna("No")
housetrain.loc[:, "GarageFinish"] = housetrain.l
oc[:, "GarageFinish"].fillna("No")
housetrain.loc[:, "GarageQual"] = housetrain.loc
[:, "GarageQual"].fillna("No")
housetrain.loc[:, "GarageCond"] = housetrain.loc
[:, "GarageCond"].fillna("No")
housetrain.loc[:, "GarageArea"] = housetrain.loc
[:, "GarageArea"].fillna(0)
housetrain.loc[:, "GarageCars"] = housetrain.loc
[:, "GarageCars"].fillna(0)
# HalfBath : NA most likely means no half baths
 above grade
housetrain.loc[:, "HalfBath"] = housetrain.loc
[:, "HalfBath"].fillna(0)
# HeatingQC : NA most likely means typical
housetrain.loc[:, "HeatingQC"] = housetrain.loc
[:, "HeatingQC"].fillna("TA")
# KitchenAbvGr : NA most likely means 0
housetrain.loc[:, "KitchenAbvGr"] = housetrain.l
oc[:, "KitchenAbvGr"].fillna(0)
# KitchenQual : NA most likely means typical
housetrain.loc[:, "KitchenQual"] = housetrain.lo
c[:, "KitchenQual"].fillna("TA")
# LotFrontage : NA most likely means no lot fron
tage
housetrain.loc[:, "LotFrontage"] = housetrain.lo
c[:, "LotFrontage"].fillna(0)
# LotShape : NA most likely means regular
housetrain.loc[:, "LotShape"] = housetrain.loc
[:, "LotShape"].fillna("Reg")
# MasVnrType : NA most likely means no veneer
#housetrain.loc[:, "MasVnrType"] = housetrain.lo
c[:, "MasVnrType"].fillna("None")
housetrain.loc[:, "MasVnrArea"] = housetrain.loc
[:, "MasVnrArea"].fillna(0)
# MiscFeature : data description says NA means
 "no misc feature"
housetrain.loc[:, "MiscFeature"] = housetrain.lo
c[:, "MiscFeature"].fillna("No")
housetrain.loc[:, "MiscVal"] = housetrain.loc[:,
"MiscVal"].fillna(0)
# OpenPorchSF : NA most likely means no open por
ch
housetrain.loc[:, "OpenPorchSF"] = housetrain.lo
c[:, "OpenPorchSF"].fillna(0)
# PavedDrive : NA most likely means not paved
housetrain.loc[:, "PavedDrive"] = housetrain.loc
[:, "PavedDrive"].fillna("N")
# PoolQC : data description says NA means "no po
ol"
housetrain.loc[:, "PoolQC"] = housetrain.loc[:,
"PoolQC"].fillna("No")
housetrain.loc[:, "PoolArea"] = housetrain.loc
[:, "PoolArea"].fillna(0)
# SaleCondition : NA most likely means normal sa
```

```
# SaleCondition : NA most likely means normal sa
le
housetrain.loc[:, "SaleCondition"] = housetrain.
loc[:, "SaleCondition"].fillna("Normal")
# ScreenPorch : NA most likely means no screen p
orch
housetrain.loc[:, "ScreenPorch"] = housetrain.lo
c[:, "ScreenPorch"].fillna(0)
# TotRmsAbvGrd : NA most likely means 0
housetrain.loc[:, "TotRmsAbvGrd"] = housetrain.l
oc[:, "TotRmsAbvGrd"].fillna(0)
# Utilities : NA most likely means all public ut
ilities
housetrain.loc[:, "Utilities"] = housetrain.loc
[:, "Utilities"].fillna("AllPub")
# WoodDeckSF : NA most likely means no wood deck
housetrain.loc[:, "WoodDeckSF"] = housetrain.loc
[:, "WoodDeckSF"].fillna(0)
```

Outliers

Outliers are tricky when it comes to houses as bidding wars can drive up a price of a house, additionaly abstract features like tennis courts also greatly affect the selling price of a house, where the house is located can have serious implications as to the value of a house. Additionally with so many variables to maintain and manage, tracking down outliers is a difficult business. For simplicity, we will examine sale price against the above grade square feet as this variable tells us a very important feature of a house.

In [7]:

```
# Plotting scatter plot of the two variables
plt.scatter(housetrain.GrLivArea, housetrain.Sal
ePrice, c = "blue", marker = "s")
plt.title("Looking for outliers")
plt.xlabel("GrLivArea")
plt.ylabel("SalePrice")
plt.show()
```

As we can see, we have four outliers. Two houses that sold for far less than they should have based on the square footage, and two that sold for far more than the average. It was decided, based discussions and advice from the project brief on Kaggle to remove house that have a square footage of more than 4000 feet. This action removes 4 data points.

In [8]:

```
housetrain = housetrain[housetrain.GrLivArea < 4
000]
```

Recoding

Recoding involves substituting the values of a variable with values that are more useful. Recoding is done for a number of reasons; to create a more balanced variable by grouping small occurances, to reduce the number of distinct values, to group similar values together and so on. It is an important feature in data analysis as it helps to reduce the curse of dimensionaility later when we create dummy variables for our categorical variables.

In this section, we will be conducting an initial recoding of our variables. This will be based on trying to keep a variable having no more than 5 distinct values. Groupings will be done using the methods listed above. The first task is to get the frequency counts of our current values in each variable as we will see below.

In [9]:

```
# Graphing missing data
group = housetrain.columns.to_series().groupby(h
ousetrain.dtypes).groups # grouping columns by t
ype
groups={k.name: v for k, v in group.items()} #
 creating as dictionary

# Taking only the object type col names
objects=housetrain[groups['object'].values]
#print(objects.head(5))
# Printing freqiency counts
for i in objects.columns:
        #print('{} \n' .format(objects[i]))
        print(objects[i].value_counts())
        print('\n')
```

```
20     536
60     295
50     144
120     87
```

```
120      87
30       69
160      63
70       60
80       58
90       52
190      30
85       20
75       16
45       12
180      10
40        4
Name: MSSubClass, dtype: int64


RL          1147
RM           218
FV            65
RH            16
C (all)       10
Name: MSZoning, dtype: int64


Pave    1450
Grvl       6
Name: Street, dtype: int64


None    1365
Grvl      50
Pave      41
Name: Alley, dtype: int64


Reg    925
IR1    481
IR2     41
IR3      9
Name: LotShape, dtype: int64


Lvl    1309
Bnk      61
HLS      50
Low      36
Name: LandContour, dtype: int64


AllPub    1455
NoSeWa       1
Name: Utilities, dtype: int64


Inside     1051
Corner      260
CulDSac      94
FR2          47
FR3           4
```

```
Name: LotConfig, dtype: int64


Gtl    1378
Mod      65
Sev      13
Name: LandSlope, dtype: int64


NAmes      225
CollgCr    150
OldTown    113
Edwards     98
Somerst     86
Gilbert     79
NridgHt     77
Sawyer      74
NWAmes      73
SawyerW     59
BrkSide     58
Crawfor     51
Mitchel     49
NoRidge     39
Timber      38
IDOTRR      37
ClearCr     28
StoneBr     25
SWISU       25
MeadowV     17
Blmngtn     17
BrDale      16
Veenker     11
NPkVill      9
Blueste      2
Name: Neighborhood, dtype: int64


Norm      1258
Feedr       80
Artery      48
RRAn        26
PosN        18
RRAe        11
PosA         8
RRNn         5
RRNe         2
Name: Condition1, dtype: int64


Norm      1442
Feedr        6
RRNn         2
Artery       2
RRAn         1
PosA         1
PosN         1
RRAe         1
Name: Condition2, dtype: int64
```

```
1Fam      1216
TwnhsE     114
Duplex      52
Twnhs       43
2fmCon      31
Name: BldgType, dtype: int64


1Story    726
2Story    441
1.5Fin    154
SLvl       65
SFoyer     37
1.5Unf     14
2.5Unf     11
2.5Fin      8
Name: HouseStyle, dtype: int64


Gable      1140
Hip         283
Flat         13
Gambrel      11
Mansard       7
Shed          2
Name: RoofStyle, dtype: int64


CompShg    1432
Tar&Grv      11
WdShngl       5
WdShake       5
Membran       1
Roll          1
Metal         1
Name: RoofMatl, dtype: int64


VinylSd    515
HdBoard    221
MetalSd    220
Wd Sdng    205
Plywood    108
CemntBd     60
BrkFace     50
WdShing     26
Stucco      24
AsbShng     20
BrkComm      2
Stone        2
ImStucc      1
CBlock       1
AsphShn      1
Name: Exterior1st, dtype: int64
```

```
VinylSd    504
MetalSd    214
HdBoard    206
Wd Sdng    197
Plywood    142
CmentBd     59
Wd Shng     38
BrkFace     25
Stucco      25
AsbShng     20
ImStucc      9
Brk Cmn      7
Stone        5
AsphShn      3
Other        1
CBlock       1
Name: Exterior2nd, dtype: int64


BrkFace    444
Stone      126
BrkCmn      15
Name: MasVnrType, dtype: int64


TA    906
Gd    487
Ex     49
Fa     14
Name: ExterQual, dtype: int64


TA    1278
Gd     146
Fa      28
Ex       3
Po       1
Name: ExterCond, dtype: int64


PConc     643
CBlock    634
BrkTil    146
Slab       24
Stone       6
Wood        3
Name: Foundation, dtype: int64


TA    649
Gd    618
Ex    117
No     37
Fa     35
Name: BsmtQual, dtype: int64
```

```
TA    1307
Gd      65
Fa      45
No      37
Po       2
Name: BsmtCond, dtype: int64


No    991
Av    220
Gd    131
Mn    114
Name: BsmtExposure, dtype: int64


Unf    430
GLQ    414
ALQ    220
BLQ    148
Rec    133
LwQ     74
No      37
Name: BsmtFinType1, dtype: int64


Unf    1252
Rec      54
LwQ      46
No       38
BLQ      33
ALQ      19
GLQ      14
Name: BsmtFinType2, dtype: int64


GasA     1424
GasW       18
Grav        7
Wall        4
OthW        2
Floor       1
Name: Heating, dtype: int64


Ex    737
TA    428
Gd    241
Fa     49
Po      1
Name: HeatingQC, dtype: int64


Y    1361
N      95
Name: CentralAir, dtype: int64


SBrkr    1330
```

```
FuseA      94
FuseF      27
FuseP       3
Mix         1
Name: Electrical, dtype: int64


TA    735
Gd    586
Ex     96
Fa     39
Name: KitchenQual, dtype: int64


Typ     1356
Min2      34
Min1      31
Mod       15
Maj1      14
Maj2       5
Sev        1
Name: Functional, dtype: int64


No    690
Gd    378
TA    312
Fa     33
Ex     23
Po     20
Name: FireplaceQu, dtype: int64


Attchd     867
Detchd     387
BuiltIn     87
No          81
Basment     19
CarPort      9
2Types       6
Name: GarageType, dtype: int64


Unf    605
RFn    422
Fin    348
No      81
Name: GarageFinish, dtype: int64


TA    1307
No      81
Fa      48
Gd      14
Po       3
Ex       3
Name: GarageQual, dtype: int64
```

```
TA    1322
No      81
Fa      35
Gd       9
Po       7
Ex       2
Name: GarageCond, dtype: int64


Y    1336
N      90
P      30
Name: PavedDrive, dtype: int64


No    1451
Fa       2
Gd       2
Ex       1
Name: PoolQC, dtype: int64


No       1176
MnPrv     156
GdPrv      59
GdWo       54
MnWw       11
Name: Fence, dtype: int64


No      1402
Shed      49
Othr       2
Gar2       2
TenC       1
Name: MiscFeature, dtype: int64


WD       1265
New       120
COD        43
ConLD       9
ConLw       5
ConLI       5
CWD         4
Oth         3
Con         2
Name: SaleType, dtype: int64


Normal     1197
Partial     123
Abnorml     100
Family       20
Alloca       12
AdjLand       4
Name: SaleCondition, dtype: int64
```

```
Name: SaleCondition, dtype: int64
```

As we can observe, there are over a dozen varaibles that have more than 5 distinct values. There will be a lot of work involved in completing this recoding. Below find the list of variables, how we rocoded and why. If a variable is not listed, it was deemed that no changes were necessary to the variable.

- MSSubClass: Complex - Requires Specilised case
- MSZoning: Complex - Requires Specilised case
- LotShape: Grouped the irregular options together to create more balanced variable
- LandContour: Changed to binary in order to create a more balanced variable
- LotConfig: Grouped Frontage together to create a more balanced variable
- Neighborhood: Complex - Requires Specilised case
- Condition1: Grouped railrowad and positive features to create a more balanced variable
- Condition2: Grouped railrowad and positive features to create a more balanced variable
- HouseStyle: Grouped 1 story and 1.5 story together, 2 story+ together to reduce number of distinct values
- OverallQual: Recoded to reduce number of distinct values/add numerical order
- OverallCond: Recoded to reduce number of distinct values/add numerical order
- RoofStyle: Regrouped everything not Gable or Hip to other to create more balanced variable
- RoofMatl: Made binary of standard vs not standard to reduce number of distinct values
- Exterior1st: Complex - Requires Specilised case
- Exterior2nd: Complex - Requires Specilised case
- ExterQual: Recoded to reduce number of distinct values/add numerical order
- ExterCond: Recoded to reduce number of distinct values/add numerical order
- Foundation: Grouped non standard to other to reduce number of distinct variables
- BsmtQual: Recoded to reduce number of distinct values/add numerical order
- BsmtCond: Recoded to reduce number of distinct values/add numerical order
- BsmtExposure: Recoded to reduce number of distinct values/add numerical orders
- BsmtFinType1: Grouped like values together to reduce number of distinct values
- BsmtFinType2: Grouped like values together to reduce number of distinct values

number of distinct values
- Heating: Grouped Gas together to reduce number of distinct values
- HeatingQC: Recoded to reduce number of distinct values/add numerical order
- KitchenQual: Recoded to reduce number of distinct values/add numerical order
- Functional: Recoded to reduce number of distinct values/add numerical order
- GarageType: Complex - Requires Specilised case
- GarageQual: Recoded to reduce number of distinct values/add numerical order
- GarageCond: Recoded to reduce number of distinct values/add numerical order
- SaleType: Grouped similar contracts together to reduce number of distinct values
- SaleCondition: Complex - Requires Specilised case

The next blocks of code execute the above descriptions. We replace all the "Excellent" and "Good" ratings with 3, "Average" with 2 and so on. We group frontage on either 2 sides and frontage on 3 sides to just frontage for *LotConfig* and many other changes in order to make the data more manageable, reduce the curse of dimensionaility, and ultimately, create a better model.

In [10]:

```
# reg or irreg
housetrain['LotShape']=housetrain['LotShape'].replace(['IR1','IR2','IR3'],'IRReg')
#print(housetrain['LotShape'].value_counts())

# flat or not flat
housetrain['LandContour']=housetrain['LandContour'].replace(['Bnk','HLS','Low'],'NotFlat')
#print(housetrain['LandContour'].value_counts())

# combined frontage
housetrain['LotConfig']=housetrain['LotConfig'].replace(['FR2','FR3'],'Frontage')
#print(housetrain['LotConfig'].value_counts())

# combined rail and pos
housetrain['Condition1']=housetrain['Condition1'].replace(['RRNn','RRAn','RRNe','RRAe'],'Rail')
housetrain['Condition1']=housetrain['Condition1'].replace(['PosN','PosA'],'Pos')
#print(housetrain['Condition1'].value_counts())

# combined rail and pos
housetrain['Condition2']=housetrain['Condition2'].replace(['RRNn','RRAn','RRNe','RRAe'],'Rail')
housetrain['Condition2']=housetrain['Condition2'].replace(['PosN','PosA'],'Pos')
```

```
#print(housetrain['Condition2'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetrain['ExterQual']=housetrain['ExterQual'].
replace(['Ex','Gd'],'Above Average')
housetrain['ExterQual']=housetrain['ExterQual'].
replace(['Fa','Po'],'Below Average')
#print(housetrain['ExterQual'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetrain['ExterCond']=housetrain['ExterCond'].
replace(['Ex','Gd'],'Above Average')
housetrain['ExterCond']=housetrain['ExterCond'].
replace(['Fa','Po'],'Below Average')
#print(housetrain['ExterCond'].value_counts())

housetrain['HouseStyle']=housetrain['HouseStyle'
].replace(['1Story','1.5Unf','1.5Fin'],'1to2Stor
y')
housetrain['HouseStyle']=housetrain['HouseStyle'
].replace(['2Story','2.5Unf','2.5Fin'],'2+Story'
)
#print(housetrain['HouseStyle'].value_counts())

housetrain['RoofStyle']=housetrain['RoofStyle'].
replace(['Flat','Gambrel','Mansard','Shed'],'Oth
er')
#print(housetrain['RoofStyle'].value_counts())

housetrain['RoofMatl']=housetrain['RoofMatl'].re
place(['ClyTile','Membran','Metal','Roll','Tar&G
rv','WdShake','WdShngl'],'Other')
#print(housetrain['RoofMatl'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetrain['SaleType']=housetrain['SaleType'].re
place(['WD','CWD','VWD'],'Warrenty Deed')
housetrain['SaleType']=housetrain['SaleType'].re
place(['Con','ConLw','ConLI','ConLD'],'Contract'
)
#print(housetrain['SaleType'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetrain['GarageCond']=housetrain['GarageCond'
].replace(['Ex','Gd'],'Above Average')
housetrain['GarageCond']=housetrain['GarageCond'
].replace(['Fa','Po'],'Below Average')
#print(housetrain['GarageCond'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetrain['GarageQual']=housetrain['GarageQual'
].replace(['Ex','Gd'],'Above Average')
housetrain['GarageQual']=housetrain['GarageQual'
```

```
].replace(['Fa','Po'],'Below Average')
#print(housetrain['GarageQual'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetrain['Functional']=housetrain['Functional'
].replace(['Min1','Min2'],'Min')
housetrain['Functional']=housetrain['Functional'
].replace(['Maj1','Maj2','Sev','Sal'],'Maj')
#print(housetrain['Functional'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetrain['KitchenQual']=housetrain['KitchenQua
l'].replace(['Ex','Gd'],'Above Average')
housetrain['KitchenQual']=housetrain['KitchenQua
l'].replace(['Fa','Po'],'Below Average')
#print(housetrain['KitchenQual'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetrain['HeatingQC']=housetrain['HeatingQC'].
replace(['Ex','Gd'],'Above Average')
housetrain['HeatingQC']=housetrain['HeatingQC'].
replace(['Fa','Po'],'Below Average')
#print(housetrain['HeatingQC'].value_counts())

# Merging Gas
housetrain['Heating']=housetrain['Heating'].repl
ace(['GasA','GasW'],'Gas')
#print(housetrain['Heating'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetrain['BsmtFinType2']=housetrain['BsmtFinTy
pe2'].replace(['ALQ','Rec'],'Average')
housetrain['BsmtFinType2']=housetrain['BsmtFinTy
pe2'].replace(['BLQ','LwQ'],'Below Average')
#print(housetrain['BsmtFinType2'].value_counts
())

# Recoding to have less options and grouping sim
ilar
housetrain['BsmtFinType1']=housetrain['BsmtFinTy
pe1'].replace(['ALQ','Rec'],'Average')
housetrain['BsmtFinType1']=housetrain['BsmtFinTy
pe1'].replace(['BLQ','LwQ'],'Below Average')
#print(housetrain['BsmtFinType1'].value_counts
())

# Recoding to have less options and grouping sim
ilar
housetrain['BsmtCond']=housetrain['BsmtCond'].re
place(['Ex','Gd'],'Above Average')
housetrain['BsmtCond']=housetrain['BsmtCond'].re
place(['Fa','Po'],'Below Average')
#print(housetrain['BsmtCond'].value_counts())
```

```
# Recoding to have less options and grouping sim
ilar
housetrain['BsmtQual']=housetrain['BsmtQual'].re
place(['Ex','Gd'],'Above Average')
housetrain['BsmtQual']=housetrain['BsmtQual'].re
place(['Fa','Po'],'Below Average')
#print(housetrain['BsmtQual'].value_counts())

# Foundation: One of the more standard options o
r other
housetrain['Foundation']=housetrain['Foundation'
].replace(['BrkTil','Slab','Stone','Wood'],'Othe
r')
#print(housetrain['Foundation'].value_counts())
group = housetrain.columns.to_series().groupby(h
ousetrain.dtypes).groups # grouping columns by t
ype
groups={k.name: v for k, v in group.items()} #
 creating as dictionary

# Taking only the object type col names
objects=housetrain[groups['object'].values]
for i in objects.columns:
        #print('{} \n' .format(objects[i]))
        print(objects[i].value_counts())
        print('\n')
```

```
20     536
60     295
50     144
120     87
30      69
160     63
70      60
80      58
90      52
190     30
85      20
75      16
45      12
180     10
40       4
Name: MSSubClass, dtype: int64


RL         1147
RM          218
FV           65
RH           16
C (all)      10
Name: MSZoning, dtype: int64


Pave    1450
Grvl       6
Name: Street, dtype: int64
```

```
None    1365
Grvl      50
Pave      41
Name: Alley, dtype: int64


Reg      925
IRReg    531
Name: LotShape, dtype: int64


Lvl        1309
NotFlat     147
Name: LandContour, dtype: int64


AllPub    1455
NoSeWa       1
Name: Utilities, dtype: int64


Inside      1051
Corner       260
CulDSac       94
Frontage      51
Name: LotConfig, dtype: int64


Gtl    1378
Mod      65
Sev      13
Name: LandSlope, dtype: int64


NAmes      225
CollgCr    150
OldTown    113
Edwards     98
Somerst     86
Gilbert     79
NridgHt     77
Sawyer      74
NWAmes      73
SawyerW     59
BrkSide     58
Crawfor     51
Mitchel     49
NoRidge     39
Timber      38
IDOTRR      37
ClearCr     28
StoneBr     25
SWISU       25
MeadowV     17
Blmngtn     17
BrDale      16
Veenker     11
NPkVill      9
```

```
NPkVill       9
Blueste       2
Name: Neighborhood, dtype: int64


Norm      1258
Feedr       80
Artery      48
Rail        44
Pos         26
Name: Condition1, dtype: int64


Norm      1442
Feedr        6
Rail         4
Pos          2
Artery       2
Name: Condition2, dtype: int64


1Fam      1216
TwnhsE     114
Duplex      52
Twnhs       43
2fmCon      31
Name: BldgType, dtype: int64


1to2Story    894
2+Story      460
SLvl          65
SFoyer        37
Name: HouseStyle, dtype: int64


Gable    1140
Hip       283
Other      33
Name: RoofStyle, dtype: int64


CompShg    1432
Other        24
Name: RoofMatl, dtype: int64


VinylSd    515
HdBoard    221
MetalSd    220
Wd Sdng    205
Plywood    108
CemntBd     60
BrkFace     50
WdShing     26
Stucco      24
AsbShng     20
BrkComm      2
```

```
Stone        2
ImStucc      1
CBlock       1
AsphShn      1
Name: Exterior1st, dtype: int64


VinylSd    504
MetalSd    214
HdBoard    206
Wd Sdng    197
Plywood    142
CmentBd     59
Wd Shng     38
BrkFace     25
Stucco      25
AsbShng     20
ImStucc      9
Brk Cmn      7
Stone        5
AsphShn      3
Other        1
CBlock       1
Name: Exterior2nd, dtype: int64


BrkFace    444
Stone      126
BrkCmn      15
Name: MasVnrType, dtype: int64


TA               906
Above Average    536
Below Average     14
Name: ExterQual, dtype: int64


TA               1278
Above Average     149
Below Average      29
Name: ExterCond, dtype: int64


PConc     643
CBlock    634
Other     179
Name: Foundation, dtype: int64


Above Average    735
TA               649
No                37
Below Average     35
Name: BsmtQual, dtype: int64


TA               1307
```

```
Above Average      65
Below Average      47
No                 37
Name: BsmtCond, dtype: int64


No    991
Av    220
Gd    131
Mn    114
Name: BsmtExposure, dtype: int64


Unf              430
GLQ              414
Average          353
Below Average    222
No                37
Name: BsmtFinType1, dtype: int64


Unf              1252
Below Average      79
Average            73
No                 38
GLQ                14
Name: BsmtFinType2, dtype: int64


Gas      1442
Grav        7
Wall        4
OthW        2
Floor       1
Name: Heating, dtype: int64


Above Average    978
TA               428
Below Average     50
Name: HeatingQC, dtype: int64


Y    1361
N      95
Name: CentralAir, dtype: int64


SBrkr    1330
FuseA      94
FuseF      27
FuseP       3
Mix         1
Name: Electrical, dtype: int64


TA               735
Above Average    682
```

```
Below Average     39
Name: KitchenQual, dtype: int64


Typ    1356
Min      65
Maj      20
Mod      15
Name: Functional, dtype: int64


No    690
Gd    378
TA    312
Fa     33
Ex     23
Po     20
Name: FireplaceQu, dtype: int64


Attchd     867
Detchd     387
BuiltIn     87
No          81
Basment     19
CarPort      9
2Types       6
Name: GarageType, dtype: int64


Unf    605
RFn    422
Fin    348
No      81
Name: GarageFinish, dtype: int64


TA               1307
No                 81
Below Average      51
Above Average      17
Name: GarageQual, dtype: int64


TA               1322
No                 81
Below Average      42
Above Average      11
Name: GarageCond, dtype: int64


Y    1336
N      90
P      30
Name: PavedDrive, dtype: int64
```

```
No    1451
Fa       2
Gd       2
Ex       1
Name: PoolQC, dtype: int64


No       1176
MnPrv     156
GdPrv      59
GdWo       54
MnWw       11
Name: Fence, dtype: int64


No      1402
Shed      49
Othr       2
Gar2       2
TenC       1
Name: MiscFeature, dtype: int64


Warrenty Deed    1269
New               120
COD                43
Contract           21
Oth                 3
Name: SaleType, dtype: int64


Normal     1197
Partial     123
Abnorml     100
Family       20
Alloca       12
AdjLand       4
Name: SaleCondition, dtype: int64
```

In [11]:

```
# Encode some categorical features as ordered nu
mbers when there is information in the order
housetrain = housetrain.replace({"BsmtCond" : {
"No" : 0, "Below Average" : 1, "TA" : 2, "Above
 Average":3},
                       "BsmtExposure" : {"No" :
0, "Mn" : 1, "Av": 2, "Gd" : 3},
                        "Fence":{"GdPrv":2,"GdW
o":2,"MnPrv":1,"MnWw":1,"No":0},
                        "LotShape":{"IRReg":0,"R
eg":1},
                        "CentralAir":{"N":0,"Y":
1},
                        "LandContour":{"NotFlat"
:0,"Lvl":1},
```

```
                        "PavedDrive":{"N":0,"Y":
1,"P":1},
                       "BsmtQual" : {"No" : 0,
"Below Average" : 1, "TA" : 2, "Above Average":3
},
                       "ExterCond" : {"Below Ave
rage" : 1, "TA" : 2, "Above Average":3},
                       "ExterQual" : {"Below Ave
rage" : 1, "TA" : 2, "Above Average":3},
                       "BsmtFinType1":{"No":0,"U
nf":1,"Below Average":1,"Average":2,"GLQ":3},
                        "BsmtFinType2":{"No":0,
"Unf":1,"Below Average":1,"Average":2,"GLQ":3},
                       "Functional" : {"Maj" : 1
, "Mod" : 2, "Min" : 3, "Typ" : 4},
                       "GarageCond" : {"No" : 0,
"Below Average" : 1, "TA" : 2, "Above Average":3
},
                       "GarageQual" : {"No" : 0,
"Below Average" : 1, "TA" : 2, "Above Average":3
},
                       "HeatingQC" : {"Below Ave
rage" : 1, "TA" : 2, "Above Average":3},
                       "KitchenQual" : {"Below A
verage" : 1, "TA" : 2, "Above Average":3},
                       "LandSlope" : {"Sev" : 1,
"Mod" : 2, "Gtl" : 3}}
                      )

# Create new features
# 1* Simplifications of existing features
housetrain["OverallQual"] = housetrain.OverallQu
al.replace({1 : 1, 2 : 1, 3 : 1, # bad

4 : 2, 5 : 2, 6 : 2, # average

7 : 3, 8 : 3, 9 : 3, 10 : 3 # good

})
housetrain["OverallCond"] = housetrain.OverallCo
nd.replace({1 : 1, 2 : 1, 3 : 1, # bad

4 : 2, 5 : 2, 6 : 2, # average

7 : 3, 8 : 3, 9 : 3, 10 : 3 # good

})



group = housetrain.columns.to_series().groupby(h
ousetrain.dtypes).groups # grouping columns by t
ype
groups={k.name: v for k, v in group.items()} #
 creating as dictionary

# Taking only the object type col names
objects=housetrain[groups['object'].values]
```

```
for i in objects.columns:
        #print('{} \n' .format(objects[i]))
        print(objects[i].value_counts())
        print('\n')
```

```
20     536
60     295
50     144
120     87
30      69
160     63
70      60
80      58
90      52
190     30
85      20
75      16
45      12
180     10
40       4
Name: MSSubClass, dtype: int64


RL         1147
RM          218
FV           65
RH           16
C (all)      10
Name: MSZoning, dtype: int64


Pave    1450
Grvl       6
Name: Street, dtype: int64


None    1365
Grvl      50
Pave      41
Name: Alley, dtype: int64


AllPub    1455
NoSeWa       1
Name: Utilities, dtype: int64


Inside      1051
Corner       260
CulDSac       94
Frontage      51
Name: LotConfig, dtype: int64


NAmes      225
CollgCr    150
OldTown    113
Edwards     98
```

```
Somerst     86
Gilbert     79
NridgHt     77
Sawyer      74
NWAmes      73
SawyerW     59
BrkSide     58
Crawfor     51
Mitchel     49
NoRidge     39
Timber      38
IDOTRR      37
ClearCr     28
StoneBr     25
SWISU       25
MeadowV     17
Blmngtn     17
BrDale      16
Veenker     11
NPkVill      9
Blueste      2
Name: Neighborhood, dtype: int64


Norm      1258
Feedr       80
Artery      48
Rail        44
Pos         26
Name: Condition1, dtype: int64


Norm      1442
Feedr        6
Rail         4
Pos          2
Artery       2
Name: Condition2, dtype: int64


1Fam      1216
TwnhsE     114
Duplex      52
Twnhs       43
2fmCon      31
Name: BldgType, dtype: int64


1to2Story    894
2+Story      460
SLvl          65
SFoyer        37
Name: HouseStyle, dtype: int64


Gable    1140
Hip       283
Other      33
```

```
Name: RoofStyle, dtype: int64


CompShg    1432
Other        24
Name: RoofMatl, dtype: int64


VinylSd    515
HdBoard    221
MetalSd    220
Wd Sdng    205
Plywood    108
CemntBd     60
BrkFace     50
WdShing     26
Stucco      24
AsbShng     20
BrkComm      2
Stone        2
ImStucc      1
CBlock       1
AsphShn      1
Name: Exterior1st, dtype: int64


VinylSd    504
MetalSd    214
HdBoard    206
Wd Sdng    197
Plywood    142
CmentBd     59
Wd Shng     38
BrkFace     25
Stucco      25
AsbShng     20
ImStucc      9
Brk Cmn      7
Stone        5
AsphShn      3
Other        1
CBlock       1
Name: Exterior2nd, dtype: int64


BrkFace    444
Stone      126
BrkCmn      15
Name: MasVnrType, dtype: int64


PConc     643
CBlock    634
Other     179
Name: Foundation, dtype: int64
```

```
Gas       1442
Grav         7
Wall         4
OthW         2
Floor        1
Name: Heating, dtype: int64


SBrkr    1330
FuseA      94
FuseF      27
FuseP       3
Mix         1
Name: Electrical, dtype: int64


No    690
Gd    378
TA    312
Fa     33
Ex     23
Po     20
Name: FireplaceQu, dtype: int64


Attchd     867
Detchd     387
BuiltIn     87
No          81
Basment     19
CarPort      9
2Types       6
Name: GarageType, dtype: int64


Unf    605
RFn    422
Fin    348
No      81
Name: GarageFinish, dtype: int64


No    1451
Fa       2
Gd       2
Ex       1
Name: PoolQC, dtype: int64


No      1402
Shed      49
Othr       2
Gar2       2
TenC       1
Name: MiscFeature, dtype: int64


Warrenty Deed    1269
```

```
New                120
COD                 43
Contract            21
Oth                  3
Name: SaleType, dtype: int64


Normal     1197
Partial     123
Abnorml     100
Family       20
Alloca       12
AdjLand       4
Name: SaleCondition, dtype: int64
```

Specialized Recoding

In [12]:

```
# Specialized Recoding
# Neighborhood (based off average)
housetrain['Neighborhood']=housetrain['Neighborh
ood'].replace(['MeadowV', 'IDOTRR', 'BrDale', 'B
rkSide', 'Edwards', 'OldTown',

'Sawyer', 'Blueste'],'Low')
housetrain['Neighborhood']=housetrain['Neighborh
ood'].replace(['SWISU', 'NPkVill', 'NAmes','Mitc
hel', 'SawyerW', 'NWAmes',

'Gilbert', 'Blmngtn'],'Mid')
housetrain['Neighborhood']=housetrain['Neighborh
ood'].replace(['CollgCr','Crawfor','ClearCr', 'S
omerst', 'Veenker', 'Timber',

'StoneBr', 'NridgHt', 'NoRidge'],'High')

# Exterior1st
housetrain['Exterior1st']=housetrain['Exterior1s
t'].replace(['Plywood', 'CmentBd', 'Wd Shng', 'S
tucco', 'BrkFace', 'AsbShng',

'ImStucc', 'Brk Cmn', 'Stone', 'AsphShn', 'CBloc
k', 'Other',

'CemntBd', 'WdShing', 'BrkComm'],'AllOther')
# Exterior2nd
housetrain['Exterior2nd']=housetrain['Exterior2n
d'].replace(['Plywood', 'CmentBd', 'Wd Shng', 'S
tucco', 'BrkFace', 'AsbShng',

'ImStucc', 'Brk Cmn', 'Stone', 'AsphShn', 'CBloc
k', 'Other',

'CemntBd', 'WdShing', 'BrkComm'],'AllOther')
```

```
# Garage Type
housetrain['GarageType']=housetrain['GarageType'
].replace(['BuiltIn', 'NA', 'Basment', 'CarPort'
, '2Types'],'Other')

# Sale Condition
housetrain['SaleCondition']=housetrain['SaleCond
ition'].replace(['Alloca'],'Normal')
housetrain['SaleCondition']=housetrain['SaleCond
ition'].replace(['Family','AdjLand'],'Other')
```

In [13]:

```
# Combining variables
housetrain['BsmtFinSF']=housetrain['BsmtFinSF1']
+housetrain['BsmtFinSF2']
housetrain['PorchSF']=housetrain['OpenPorchSF']+
housetrain['EnclosedPorch']+housetrain['3SsnPorc
h']+housetrain['ScreenPorch']

housetrain['hasPool'] = np.where(housetrain['Poo
lArea']>0, 1, 0)
```

Normalising

Most statistical methods (the parametric methods) include the assumption that the sample is drawn from a population where the values have a Normal distribution. One of the first steps of statistical analysis of your data is therefore to check the distribution of the different variables.

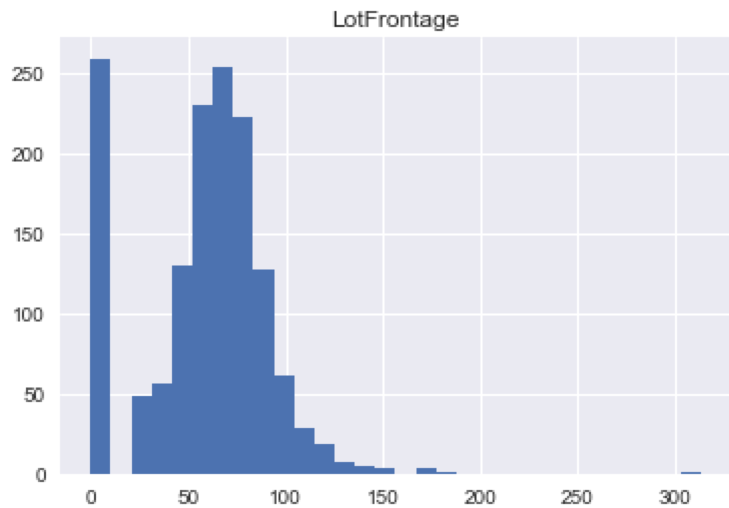
Upon completing the task of dealing with missing values and errors in the data, it was decided to move on to normalizing our data.

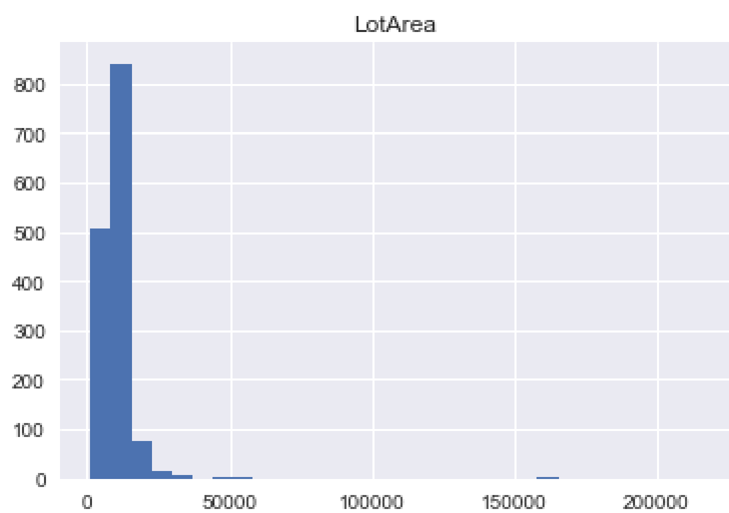
The Normal distribution is symmetrical, not very peaked or very flat-topped, and if we examine the charts below, we can see that our data is often skewed. We have selected the log method for the moment, and will improve on this method before the final model.

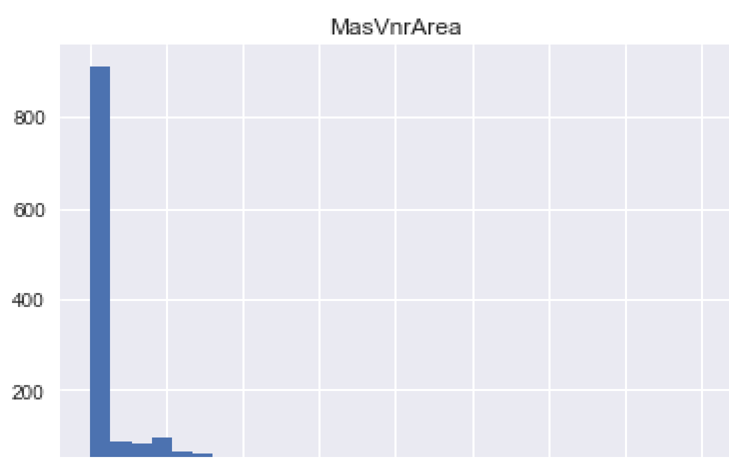
In [14]:

```
# Printing plots for int 64 and float64
# #quantvar = ['LotFrontage','LotArea','MasVnrAr
ea','BsmtFinSF1','BsmtFinSF2','BsmtUnfSF','Total
BsmtSF','1stFlrSF','2ndFlrSF',
#              'LowQualFinSF','GrLivArea','BsmtFu
llBath','BsmtHalfBath','FullBath','HalfBath','Be
droomAbvGr','KitchenAbvGr','TotRmsAbvGrd',
#              'Fireplaces','GarageCars','GarageA
rea','WoodDeckSF','OpenPorchSF','EnclosedPorc
h','3SsnPorch','ScreenPorch','PoolArea'
#              ,'MiscVal','SalePrice']
```

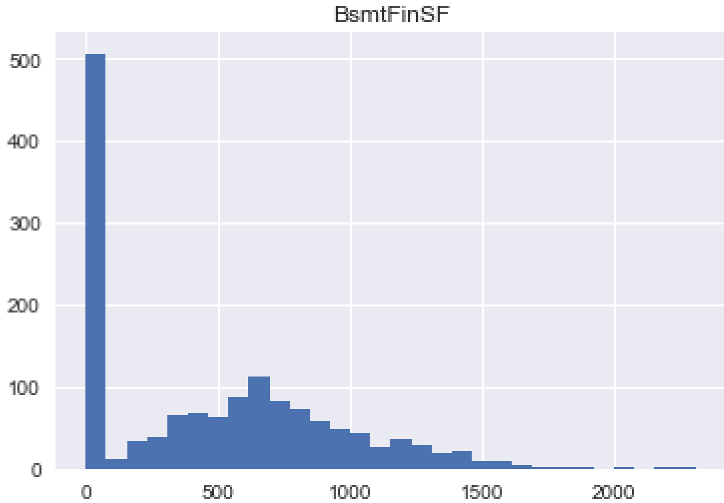
```
quantvar = ['LotFrontage','LotArea','MasVnrArea'
,'BsmtFinSF','BsmtUnfSF','TotalBsmtSF','1stFlrS
F','2ndFlrSF',
            'LowQualFinSF','GrLivArea','GarageAr
ea','WoodDeckSF','PorchSF','PoolArea'
           ,'MiscVal','SalePrice']
cont_plot=housetrain[quantvar]
for i in range(len(cont_plot.columns)):
        plt.hist(cont_plot.iloc[:,i].dropna(),bi
ns=30)
        plt.title('%s' % cont_plot.columns[i])
        plt.show()
```

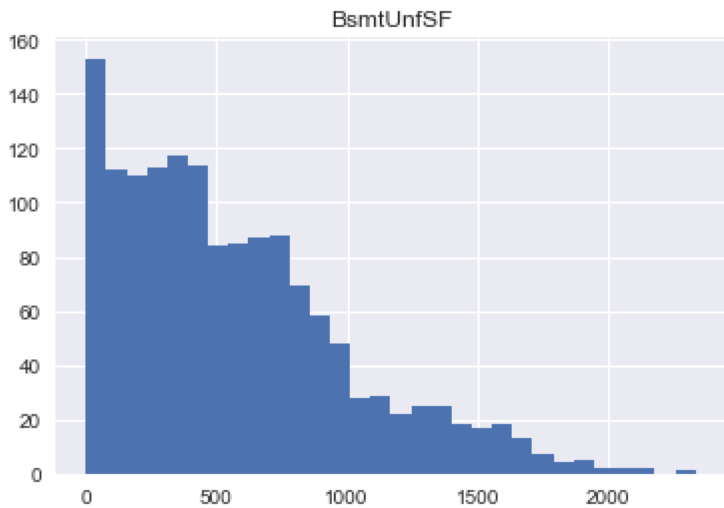


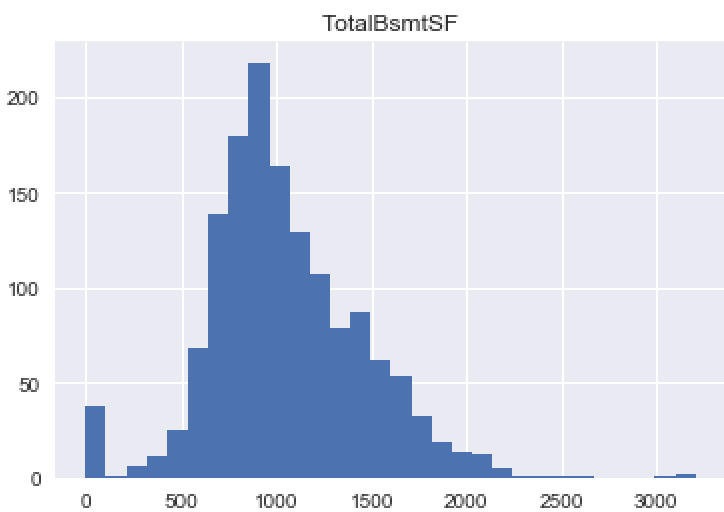


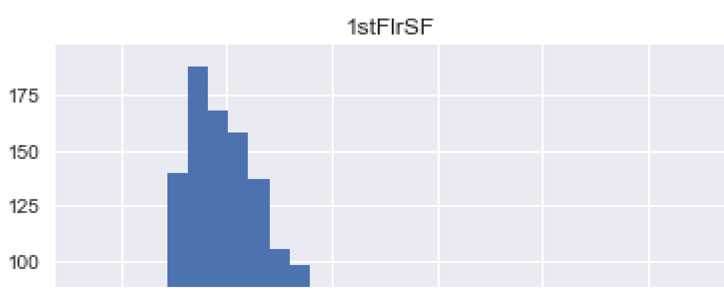


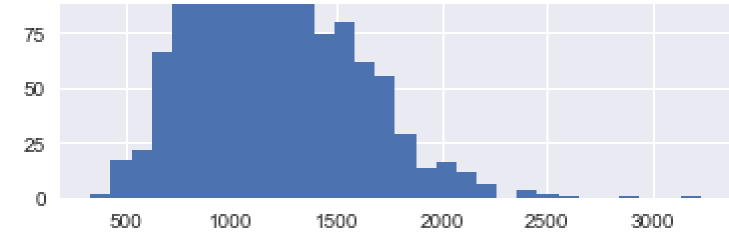

0
0
200
400
600
800
1000
1200
1400
1600


BsmtFinSF
500
400
300
200
100
0
0
500
1000
1500
2000

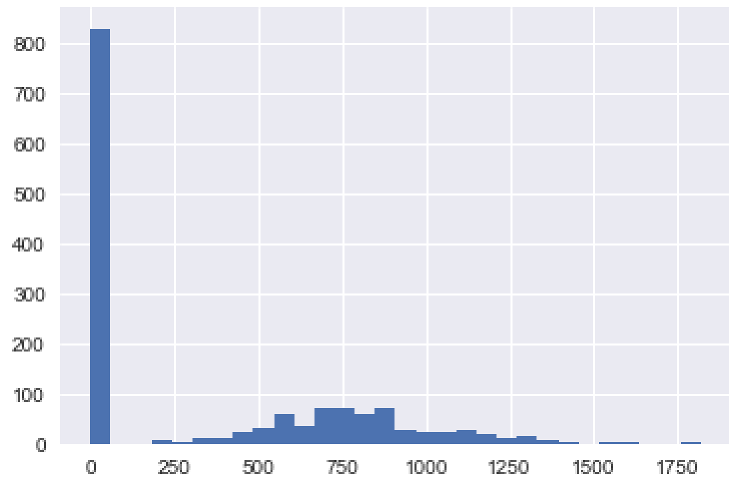

BsmtUnfSF
160
140
120
100
80
60
40
20
0
0
500
1000
1500
2000


TotalBsmtSF
200
150
100
50
0
0
500
1000
1500
2000
2500
3000

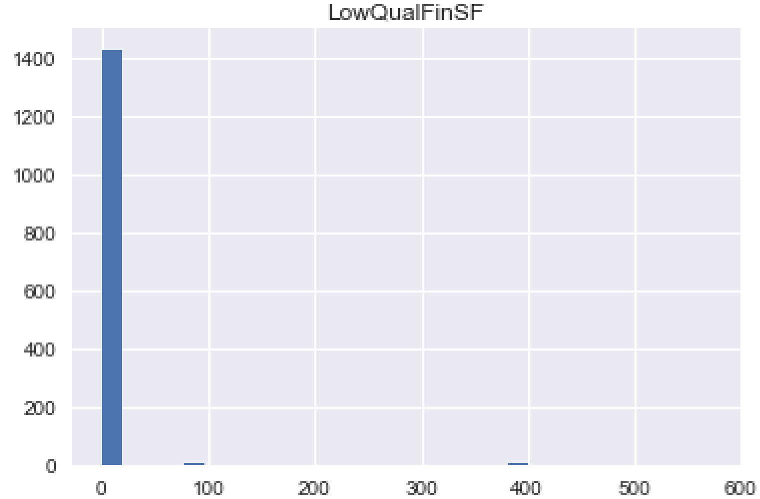

1stFlrSF
175
150
125
100



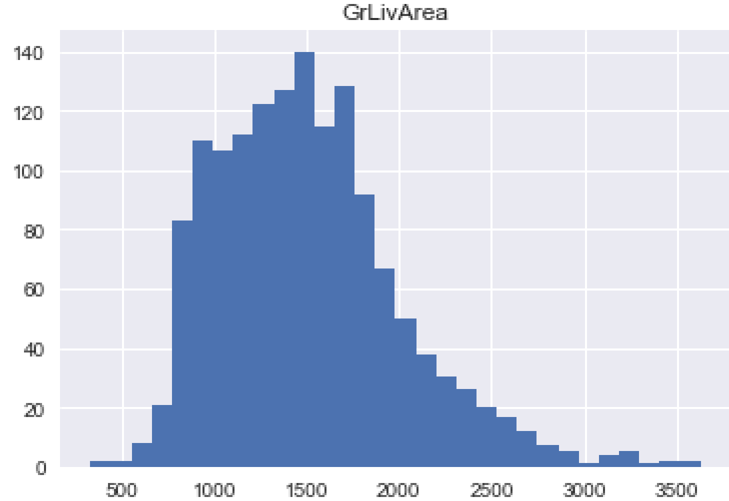
2ndFlrSF



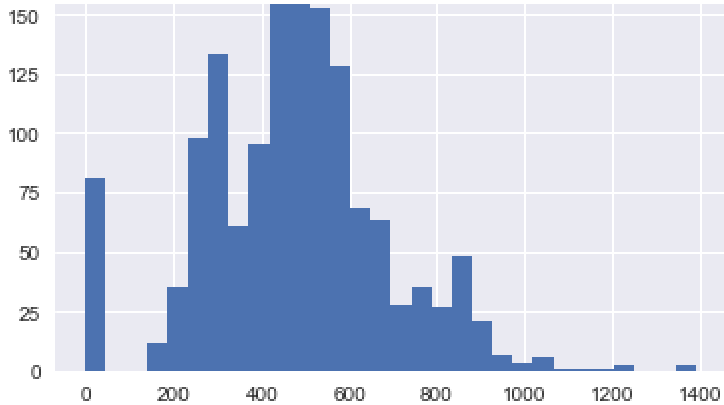
LowQualFinSF

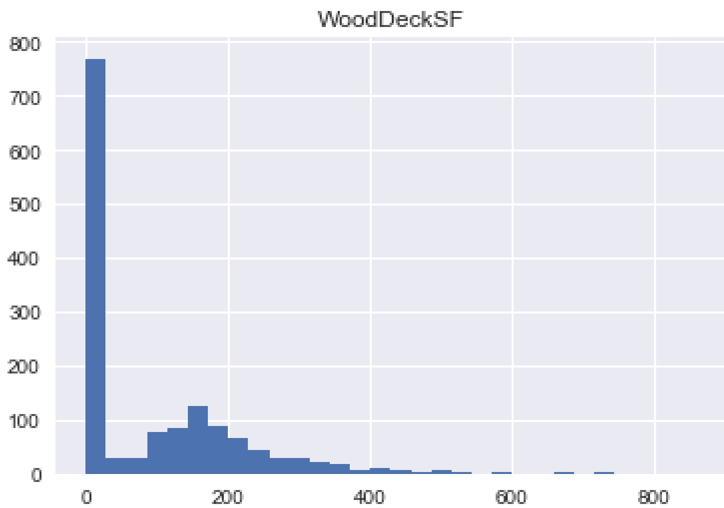


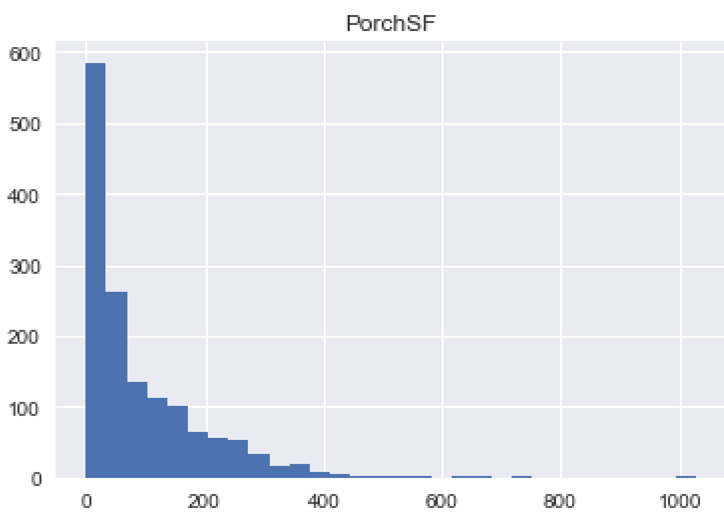
GrLivArea

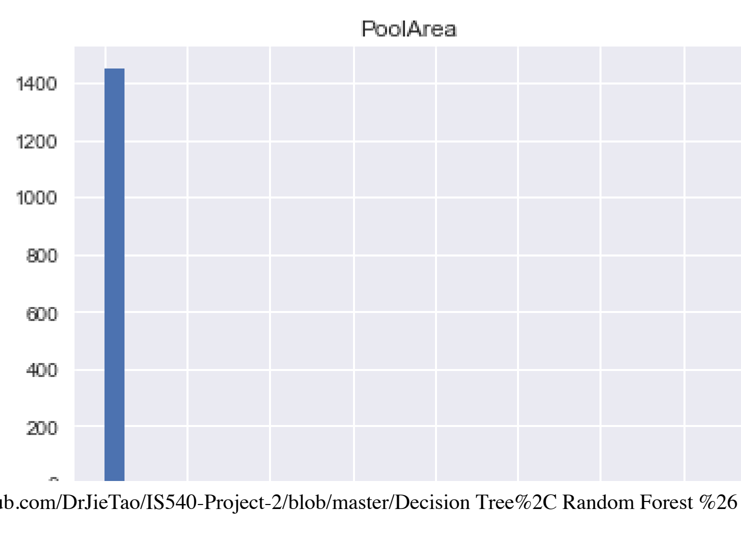


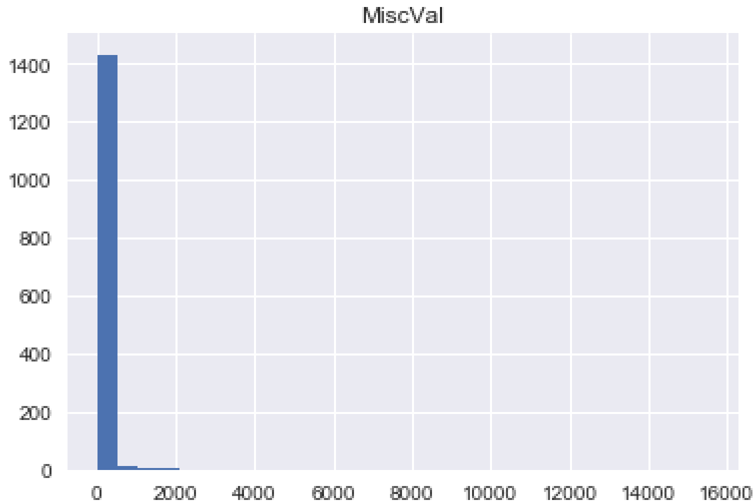
GarageArea


150
125
100
75
50
25
0
0
200
400
600
800
1000
1200
1400


WoodDeckSF
800
700
600
500
400
300
200
100
0
0
200
400
600
800


PorchSF
600
500
400
300
200
100
0
0
200
400
600
800
1000


PoolArea
1400
1200
1000
800
600
400
200



Examining the charts, we note several points of interest. We have several variables that skewed to the right. There do not appear to be any left skewed. This is more than likely caused by the presence of smaller outliers that were missed by our earlier scatter plot.

For dealing with skew, the following transformations perform well:

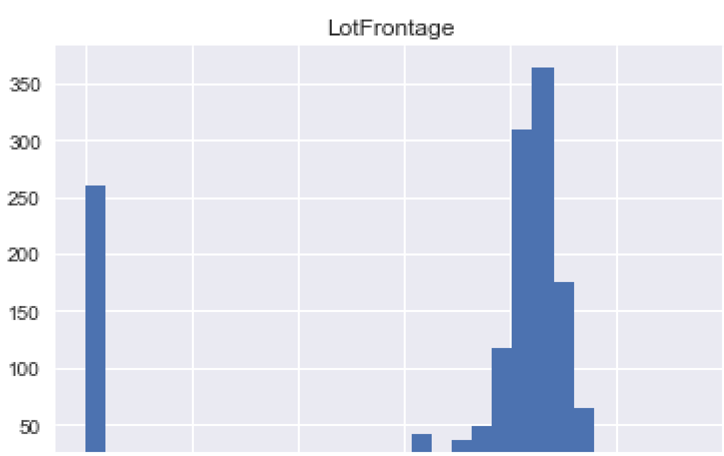
- The log transformation (sometimes computed log(x+A) where A is some constant. This is done to deal with negative or 0 values.
- The Square Root function
- Converting to a Fraction, i.e. 1/x
- The Powers transformation

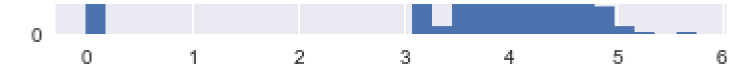
We can also use some combination thereof. For our base model, it was decided to keep things simple. As we try to improve our models, we will try alternate methods.

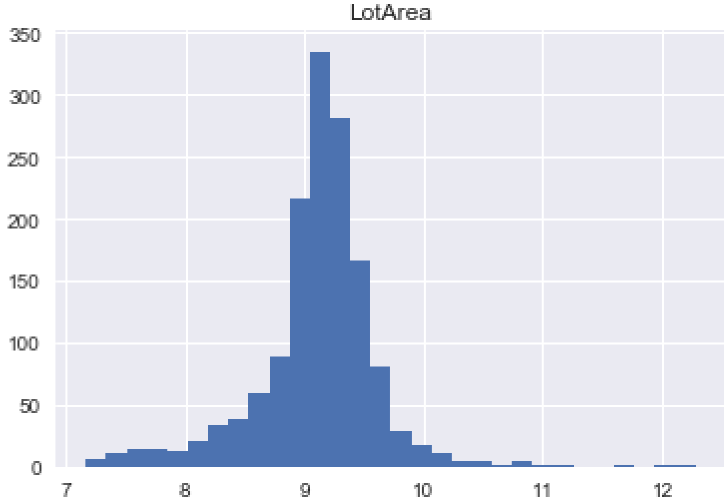
For right skewed data, the log transformation works well, and this was the selected transformation for our model for the severly right skewed data listed above. It is suspected that if the outliers right skewed data listed above. It is suspected that if the outliers were dealt with, the data would become more normally distributed. This was the decided approach with a view of returning to this as we seek to improve the model.

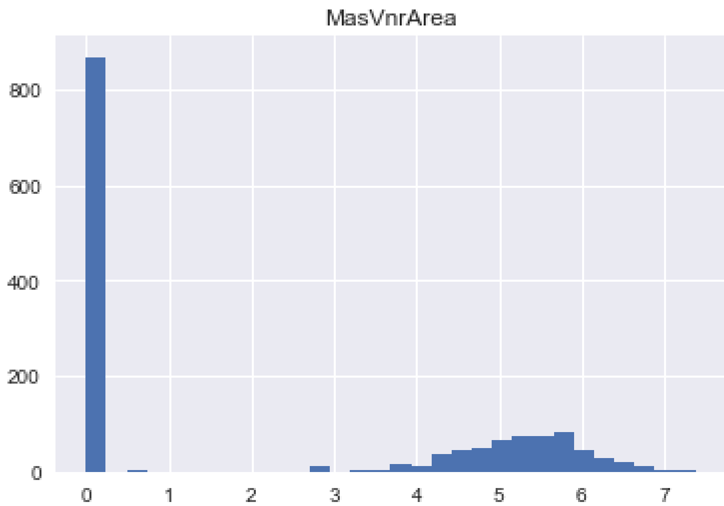
In [15]:

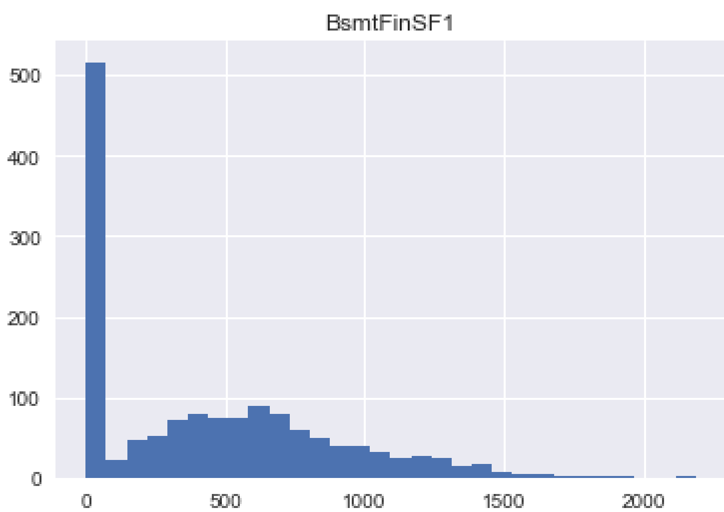
```
def log_transform(feature):
    housetrain[feature] = np.log1p(housetrain[fe
ature].values) # does a log transform on x+1

#log transforming variables
log_transform('GrLivArea')
log_transform('PorchSF')
log_transform('1stFlrSF')
log_transform('2ndFlrSF')
log_transform('BsmtUnfSF')
log_transform('BsmtFinSF')
log_transform('TotalBsmtSF')
log_transform('LotArea')
log_transform('LotFrontage')
log_transform('KitchenAbvGr')
log_transform('GarageArea')
log_transform('MasVnrArea')
log_transform('WoodDeckSF')
log_transform('PorchSF')
log_transform('SalePrice')
quantvar = ['LotFrontage','LotArea','MasVnrArea'
,'BsmtFinSF1','BsmtFinSF2','BsmtUnfSF','TotalBsm
tSF','1stFlrSF','2ndFlrSF',
            'LowQualFinSF','GrLivArea','GarageAr
ea','WoodDeckSF','PorchSF','PoolArea'
           ,'MiscVal','SalePrice']
cont_plot=housetrain[quantvar]
for i in range(len(cont_plot.columns)):
        plt.hist(cont_plot.iloc[:,i].dropna(),bi
ns=30)
        plt.title('%s' % cont_plot.columns[i])
        plt.show()
# f = pd.melt(housetrain, value_vars=quantitativ
e)
# g = sns.FacetGrid(f, col="variable",  col_wrap
=2, sharex=False, sharey=False)
# g = g.map(sns.distplot, "value")
```

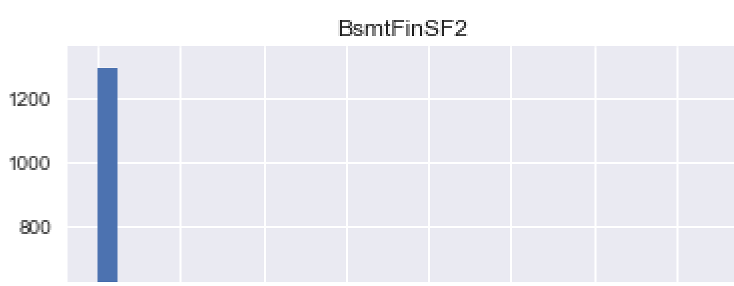


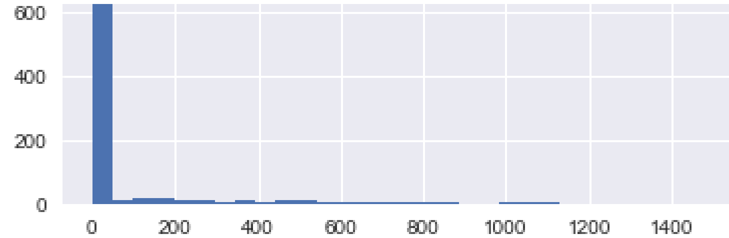

0
0
1
2
3
4
5
6


LotArea
350
300
250
200
150
100
50
0
7
8
9
10
11
12

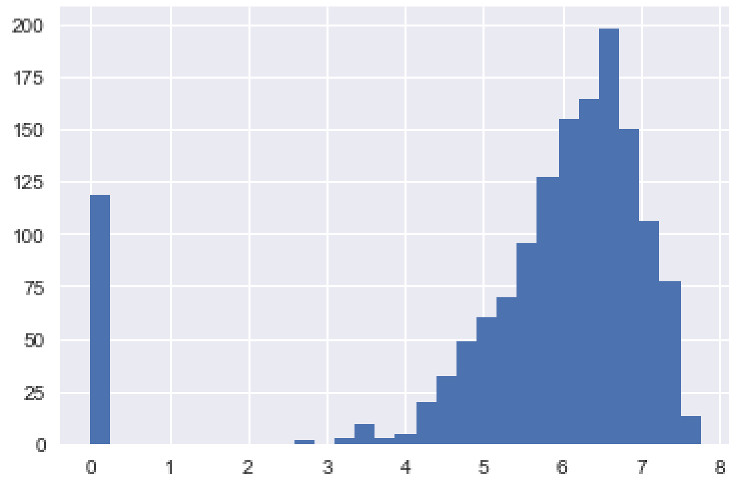

MasVnrArea
800
600
400
200
0
0
1
2
3
4
5
6
7


BsmtFinSF1
500
400
300
200
100
0
0
500
1000
1500
2000

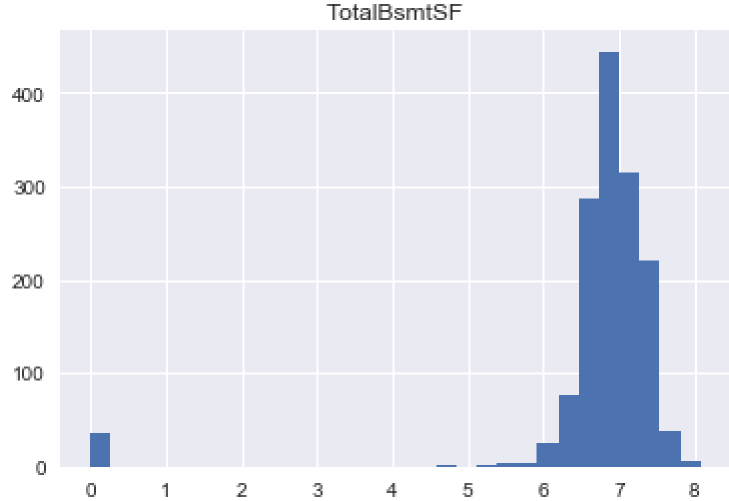

BsmtFinSF2
1200
1000
800



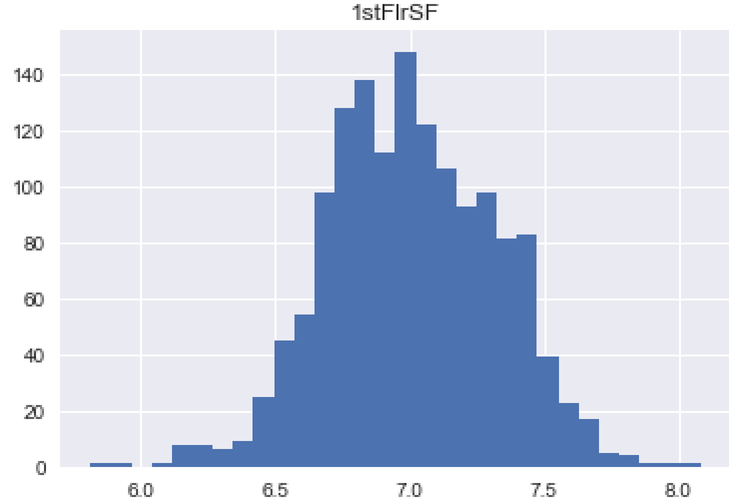
BsmtUnfSF



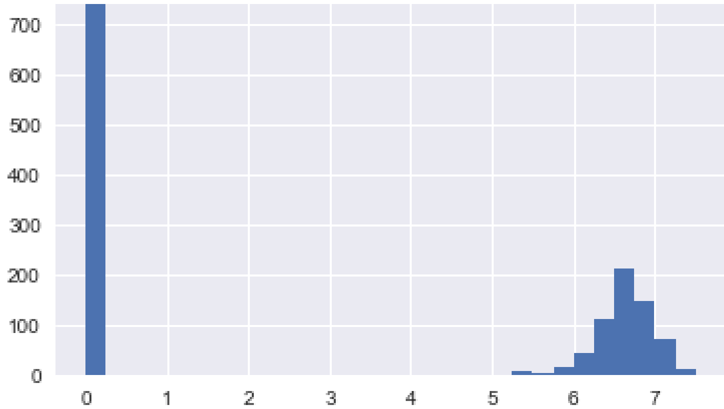
TotalBsmtSF

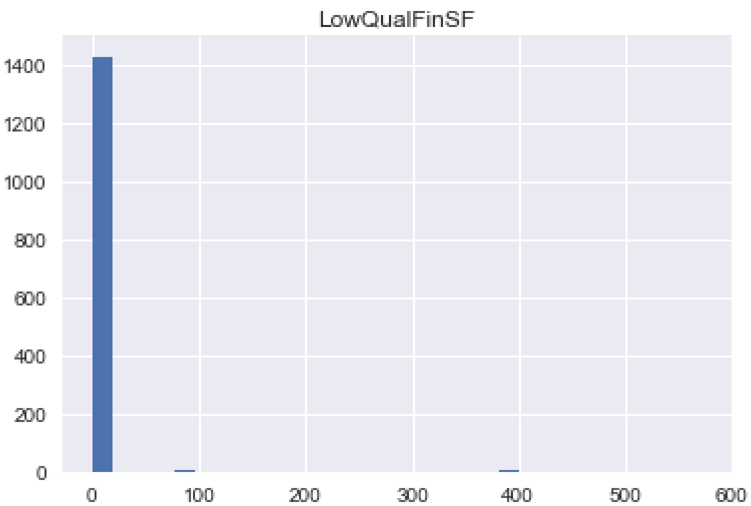


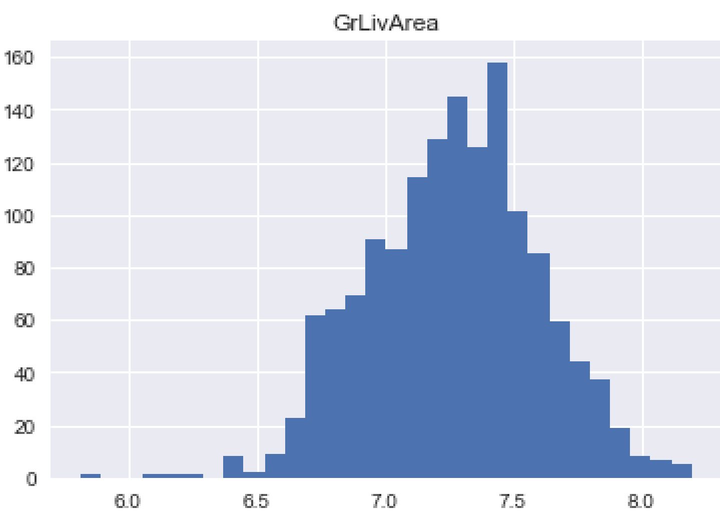
1stFlrSF

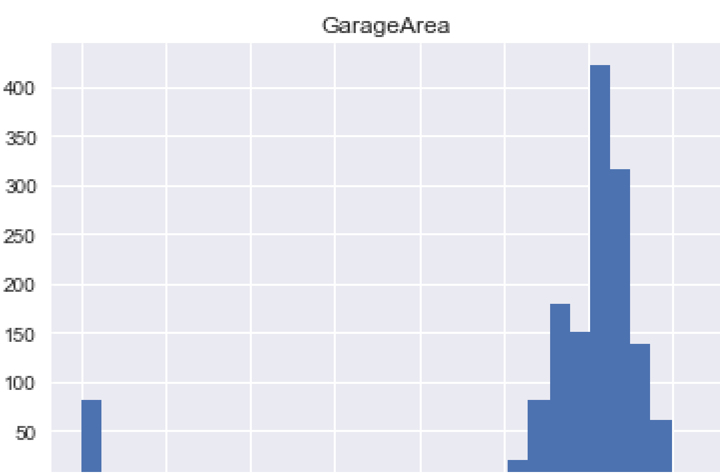


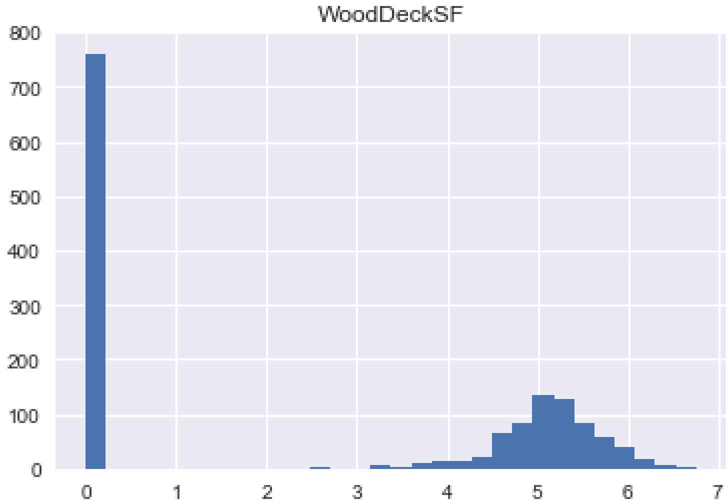
2ndFlrSF

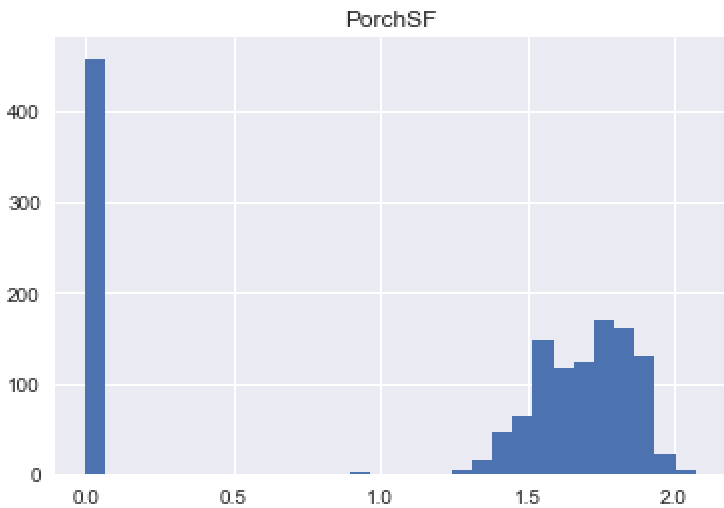

0
100
200
300
400
500
600
700
0 1 2 3 4 5 6 7

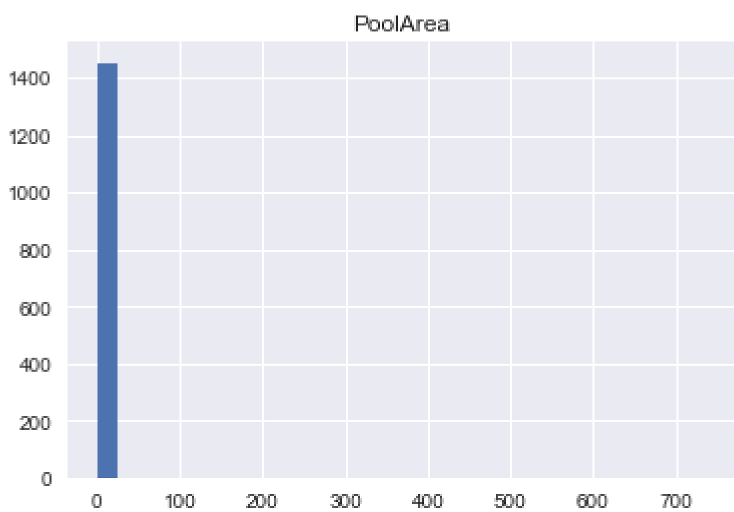

LowQualFinSF
0
200
400
600
800
1000
1200
1400
0 100 200 300 400 500 600

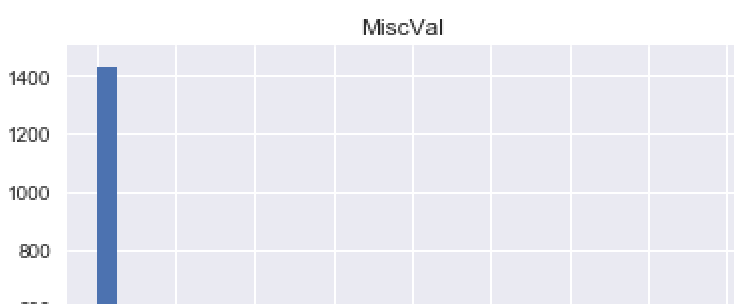

GrLivArea
0
20
40
60
80
100
120
140
160
6.0 6.5 7.0 7.5 8.0

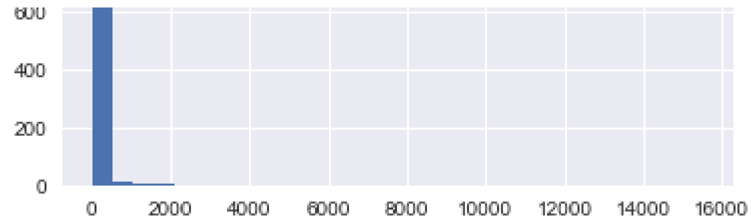

GarageArea
50
100
150
200
250
300
350
400


WoodDeckSF


PorchSF


PoolArea


MiscVal



As we can see, by using a log transformation we have normalised our data (excluding the zero values representing missing) and have tidied up the outliers.

Standardising

In [16]:

```
import scipy.stats as st
housetrain.Id = housetrain.Id.astype(str)
def standard(data,method):
    """Standarising data using various methods.

    Method 1 is MinMax scaling
    Method 2 is decimal
    Method 3 is Z score
    Version Control:
    Initial coding
    ------------------------------------------
    Date 4-Feb-18, Author: Conor Feeney, Desc: I
nitial Coding
    """
    if method == 1:
        X_std = (data - data.min(axis=0)) / (dat
a.max(axis=0) - data.min(axis=0))
        data = X_std * (1 - 0) + 0
        return data
    elif method==2:
        data = (data)/(10**len(str(int(max(data
)))))
        return data
    elif method ==3:
        data = (data - data.mean(axis=0))/data.s
```

```
td(axis=0)
        data=st.norm.cdf(data)
        return data
    elif method==4:
        return data

for i in range(len(housetrain.columns)):
    if housetrain.iloc[:,i].dtype !=object:
        housetrain.iloc[:,i]=standard(housetrain
.iloc[:,i],3)
housetrain.Id = housetrain.Id.astype(int)
housetrain.describe()
```

```
C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\scipy\stats\_distn_in
frastructure.py:875: RuntimeWarning: invalid val
ue encountered in greater
  return (self.a < x) & (x < self.b)
C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\scipy\stats\_distn_in
frastructure.py:875: RuntimeWarning: invalid val
ue encountered in less
  return (self.a < x) & (x < self.b)
C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\scipy\stats\_distn_in
frastructure.py:1731: RuntimeWarning: invalid va
lue encountered in greater_equal
  cond2 = (x >= self.b) & cond0
```

Out[16]:

	Id	LotFrontage	LotArea	LotShape
count	1456.000000	1456.000000	1456.000000	1456.000000
mean	729.967033	0.556345	0.513281	0.526844
std	421.722909	0.260035	0.246158	0.328427
min	1.000000	0.017423	0.000081	0.093519
25%	364.750000	0.573835	0.363158	0.093519
50%	730.500000	0.666017	0.537402	0.775596
75%	1094.250000	0.714023	0.686993	0.775596
max	1460.000000	0.919241	1.000000	0.775596

Correlation Analysis

Next, we needed to select the predictor variables with low pair-wise correlation values. In order to do this, we used Spearman's correlation test to determine the statistical dependence between the rankings of pairs of variables.

Spearman Test

The Spearman's rank-order correlation is the nonparametric

The Spearman's rank-order correlation is the nonparametric version of the Pearson product-moment correlation. Spearman's correlation coefficient, measures the strength and direction of association between two ranked variables. This test has some assumptions. You need two variables that are either ordinal, interval or ratio. Although you would normally hope to use a Pearson product-moment correlation on interval or ratio data, the Spearman correlation can be used when the assumptions of the Pearson correlation are markedly violated. However, Spearman's correlation determines the strength and direction of the monotonic relationship between your two variables rather than the strength and direction of the linear relationship between your two variables, which is what Pearson's correlation determines. A monotonic relationship is a relationship that does one of the following: (1) as the value of one variable increases, so does the value of the other variable; or (2) as the value of one variable increases, the other variable value decreases.

Below, you can observe the results of our Spearman correlation test.

In [17]:

```
corr=['LotFrontage','LotArea','LotShape',"BsmtFinType1","BsmtFinType2",'LandSlope','ExterQual','ExterCond','OverallQual','OverallCond','YearBuilt','YearRemodAdd','MasVnrArea','BsmtQual','BsmtCond','BsmtExposure','BsmtFinSF1','BsmtFinSF2','BsmtUnfSF','TotalBsmtSF','HeatingQC','1stFlrSF','2ndFlrSF','LowQualFinSF','GrLivArea','BsmtFullBath','BsmtHalfBath','FullBath','HalfBath','BedroomAbvGr','KitchenAbvGr','KitchenQual','TotRmsAbvGrd','Fireplaces','GarageYrBlt','GarageCars','GarageArea','GarageQual','GarageCond','WoodDeckSF','OpenPorchSF','EnclosedPorch','3SsnPorch','ScreenPorch','PoolArea','MiscVal','MoSold','YrSold','SalePrice']

housetrain[corr].corr(method='spearman').style.format("{:.2}").background_gradient(cmap=plt.get_cmap('coolwarm'), axis=1)
```

Out[17]:

	LotFrontage	LotArea	LotShape	BsmtFinTy
LotFrontage	1.0	0.34	0.13	0.04
LotArea	0.34	1.0	-0.33	0.051
LotShape	0.13	-0.33	1.0	-0.11
BsmtFinType1	0.04	0.051	-0.11	1.0
BsmtFinType2	0.00055	0.072	-0.069	0.11
LandSlope	0.036	-0.12	0.11	-0.047

ExterQual	0.12	0.13	-0.19	0.31
ExterCond	-0.054	0.033	-0.029	0.038
OverallQual	0.17	0.19	-0.17	0.26
OverallCond	-0.062	-0.031	0.011	-0.059
YearBuilt	0.13	0.099	-0.22	0.38
YearRemodAdd	0.13	0.072	-0.14	0.27
MasVnrArea	0.14	0.17	-0.094	0.25
BsmtQual	0.069	0.13	-0.23	0.42
BsmtCond	0.034	0.052	-0.11	0.25
BsmtExposure	0.078	0.2	-0.17	0.31
BsmtFinSF1	0.029	0.17	-0.13	0.69
BsmtFinSF2	-0.0024	0.074	-0.038	-0.0013
BsmtUnfSF	0.15	0.076	-0.00052	-0.28
TotalBsmtSF	0.26	0.36	-0.18	0.33
HeatingQC	0.075	0.053	-0.086	0.2
1stFlrSF	0.27	0.44	-0.17	0.21
2ndFlrSF	0.027	0.11	-0.05	-0.067
LowQualFinSF	0.0023	-0.02	0.037	-0.075
GrLivArea	0.23	0.44	-0.19	0.11
BsmtFullBath	0.0098	0.092	-0.074	0.53
BsmtHalfBath	-0.029	0.043	-0.034	0.038
FullBath	0.14	0.23	-0.17	0.15
HalfBath	0.021	0.14	-0.12	0.051
BedroomAbvGr	0.2	0.34	-0.075	-0.094
KitchenAbvGr	0.035	-0.022	0.095	-0.15
KitchenQual	0.099	0.13	-0.17	0.31
TotRmsAbvGrd	0.25	0.4	-0.12	-0.0086
Fireplaces	0.069	0.35	-0.21	0.12
GarageYrBlt	0.11	0.037	-0.17	0.32
GarageCars	0.24	0.34	-0.2	0.27
GarageArea	0.26	0.36	-0.19	0.28
GarageQual	0.052	0.16	-0.12	0.16
GarageCond	0.05	0.13	-0.12	0.15
WoodDeckSF	0.024	0.18	-0.17	0.19

OpenPorchSF	0.1	0.17	-0.13	0.17
EnclosedPorch	-0.039	-0.065	0.11	-0.16
3SsnPorch	0.026	0.063	-0.037	0.038
ScreenPorch	0.029	0.094	-0.053	0.02
PoolArea	0.034	0.063	-0.0043	0.002
MiscVal	-0.021	0.06	-0.023	-0.028
MoSold	0.037	0.0083	-0.037	-0.029
YrSold	-0.01	-0.026	0.036	0.041
SalePrice	0.24	0.45	-0.31	0.39

Looking at the above results, we see some expected correlations:

- **SalePrice** is positively correlated with YearBuilt, OverallQual, BsmtQual,TotalBsmtSF, GrLivArea, FullBath, KitchenQual, GarageCars, GarageArea
- **BsmtFinType1** and **BsmtFinType2** are positively correlated with BsmtFinSF1 and BsmtFinSF2 respectively
- **ExterQual** is positively correlated with OverallQual, YearBuilt, YearRemodAdd, BsmtQual, KitchenQual, GarageYrBlt and SalePrice
- **OverallQual** is positively correlated with YearBuilt, ExterQual and KitchenQual
- **YearBlt** is positively correlated with GarageCars, GarageYrBlt, BsmntQual, YearRemodAdd, SalePrice
- **YearRemodAdd** is positively correlated with YearBuilt, KitchenQual, GarageYrBlt, SalePrice
- **BsmtQual** is positively correlated with YearBuilt, GarageYrBlt
- **GarageYrBlt** variable is positively correlated with YearBuilt, YearRemodAdd, and BsmtQual, and GarageCars
- **1stFlrSF** variable is positively correlated with TotalBsmtSF
- **TotRmsAbvGrd** variable is positively correlated with GrLivArea and BedroomAbvGrd
- **GarageArea** variable is positively correlated with GarageCars
- **GarageCond** variable is positively correlated with GarageQual
- **BsmtFinSF1** is positively correlated with BsmtFullBath
- **TotalBsmtSF** is positively correlated with 1stFlrSF, SalePrice
- **2ndFlrSF** is positively correlated with GrLivArea, and HalfBath
- **BsmtFullBath** is positively correlated with BsmtFinSF1

- **BsmtFullBath** is positively correlated with BsmtFinSF1
- **FullBath** is positively correlated with GrLivArea and SalePrice
- **BedroomAbvGr** is positively correlated with TotalRmsAbvGrd
- **KitchenQual** is positively correlated with YearRemodAdd, OverallQual, and SalePrice
- **GarageCars** is positively correlated with YrBuilt, GarageYrBlt, GarageArea, and Sale Price
- **GarageArea** is positively correlated with Sale Price, Garage Cars

ANOVA

Next, for Categorical vs Continuous variables, we used the analysis of variance (ANOVA). ANOVA provides a statistical test of whether or not the means of several groups are equal.

To keep our ANOVA correlations simple, we chose to only analyze our target variable, SalesPrice, against each categorical variable. Below, you can observe the results of our ANOVA correlation tests.

In [18]:

```
for i in (['SalePrice']):
    grps = pd.unique(housetrain.Condition1.value
s)
    d_data = {grp:housetrain[i][housetrain.Condi
tion1 == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['Norm'], d_dat
a['Feedr'], d_data['Artery'], d_data['Rail'], d_
data['Pos'])
    print('Condition1 v Variable {} Result {}'.f
ormat(i,anova))
```

```
Condition1 v Variable SalePrice Result F_onewayR
esult(statistic=17.553926520035429, pvalue=4.445
7161059550624e-14)
```

In [19]:

```
for i in (['SalePrice']):
    grps = pd.unique(housetrain.Condition2.value
s)
    d_data = {grp:housetrain[i][housetrain.Condi
tion2 == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['Norm'], d_dat
a['Feedr'], d_data['Artery'], d_data['Rail'], d_
data['Pos'])
    print('Condition2 v Variable {} Result {}'.f
ormat(i,anova))
```

```
Condition2 v Variable SalePrice Result F_onewayR
```

```
Condition2 v Variable SalePrice Result F_onewayR
esult(statistic=4.0371811668233164, pvalue=0.002
9259071303826801)
```

In [20]:

```
for i in (['SalePrice']):
    grps = pd.unique(housetrain.Foundation.value
s)
    d_data = {grp:housetrain[i][housetrain.Found
ation == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['PConc'], d_da
ta['CBlock'], d_data['Other'])
    print('Foundation v Variable {} Result {}'.f
ormat(i,anova))
```

```
Foundation v Variable SalePrice Result F_onewayR
esult(statistic=352.99904433933301, pvalue=1.126
4280358270717e-125)
```

In [21]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.Heating.values)
    d_data = {grp:housetrain[i][housetrain.Heati
ng == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['Gas'],d_data[
'Grav'],d_data['Wall'],d_data['OthW'],d_data['Fl
oor'])
    print('Heating v Variable {} Result {}'.form
at(i,anova))
```

```
Heating v Variable SalePrice Result F_onewayResu
lt(statistic=7.9881940556220661, pvalue=2.279210
4552219803e-06)
```

In [22]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.HouseStyle.value
s)
    d_data = {grp:housetrain[i][housetrain.House
Style == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['1to2Story'],d
_data['2+Story'],d_data['SLvl'],d_data['SFoyer'
])
    print('HouseStyle v Variable {} Result {}'.f
ormat(i,anova))
```

```
HouseStyle v Variable SalePrice Result F_onewayR
esult(statistic=44.728621189377243, pvalue=1.204
2751040081352e-27)
```

In [23]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.LotConfig.values
)
    d_data = {grp:housetrain[i][housetrain.LotCo
nfig == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['Inside'],d_da
ta['Corner'],d_data['CulDSac'],d_data['Frontage'
])
    print('LotConfig v Variable {} Result {}'.fo
rmat(i,anova))
```

```
LotConfig v Variable SalePrice Result F_onewayRe
sult(statistic=11.725262866714207, pvalue=1.3635
166072715538e-07)
```

In [24]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.RoofMatl.values)
    d_data = {grp:housetrain[i][housetrain.RoofM
atl == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['CompShg'],d_d
ata['Other'])
    print('RoofMatl v Variable {} Result {}'.for
mat(i,anova))
```

```
RoofMatl v Variable SalePrice Result F_onewayRes
ult(statistic=9.1792867351649434, pvalue=0.00249
08688525218737)
```

In [25]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.RoofStyle.values
)
    d_data = {grp:housetrain[i][housetrain.RoofS
tyle == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['Gable'],d_dat
a['Hip'],d_data['Other'])
    print('RoofStyle v Variable {} Result {}'.fo
rmat(i,anova))
```

```
RoofStyle v Variable SalePrice Result F_onewayRe
sult(statistic=19.348951143684129, pvalue=5.0910
832062187951e-09)
```

In [26]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.SaleType.values)
    d_data = {grp:housetrain[i][housetrain.SaleT
ype == grp] for grp in grps}
```

```
        #run anova
    anova = stats.f_oneway(d_data['Warrenty Dee
d'],d_data['New'],d_data['COD'],d_data['Contrac
t'],d_data['Oth'])
    print('SaleType v Variable {} Result {}'.for
mat(i,anova))
```

```
SaleType v Variable SalePrice Result F_onewayRes
ult(statistic=44.017640252645272, pvalue=6.52812
3230893233e-35)
```

In [27]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.MSZoning.values)
    d_data = {grp:housetrain[i][housetrain.MSZon
ing == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['RL'],d_data[
'RM'],d_data['FV'],d_data['RH'],d_data['C (all)'
])
    print('MSZoning v Variable {} Result {}'.for
mat(i,anova))
```

```
MSZoning v Variable SalePrice Result F_onewayRes
ult(statistic=78.918253594278056, pvalue=1.24153
37633012105e-60)
```

In [28]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.Street.values)
    d_data = {grp:housetrain[i][housetrain.Stree
t == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['Pave'],d_data
['Grvl'])
    print('Street v Variable {} Result {}'.forma
t(i,anova))
```

```
Street v Variable SalePrice Result F_onewayResul
t(statistic=2.9744617288552457, pvalue=0.0848007
8079618926)
```

In [29]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.Alley.values)
    d_data = {grp:housetrain[i][housetrain.Alley
== grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['None'],d_data
['Grvl'],d_data['Pave'])
    print('Alley v Variable {} Result {}'.format
(i,anova))
```

```
Alley v Variable SalePrice Result F_onewayResult
(statistic=22.367575442858147, pvalue=2.70660527
66030589e-10)
```

In [30]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.Utilities.values
)
    d_data = {grp:housetrain[i][housetrain.Utili
ties == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['NoSeWa'],d_da
ta['AllPub'])
    print('Utilities v Variable {} Result {}'.fo
rmat(i,anova))
```

```
Utilities v Variable SalePrice Result F_onewayRe
sult(statistic=0.40953160873447858, pvalue=0.522
30815934730224)
```

In [31]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.Electrical.value
s)
    d_data = {grp:housetrain[i][housetrain.Elect
rical == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['SBrkr'],d_dat
a['FuseA'],d_data['FuseF'],d_data['FuseP'],d_dat
a['Mix'])
    print('Electrical v Variable {} Result {}'.f
ormat(i,anova))
```

```
Electrical v Variable SalePrice Result F_onewayR
esult(statistic=35.306011007320102, pvalue=3.562
8571007823335e-28)
```

In [32]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.BldgType.values)
    d_data = {grp:housetrain[i][housetrain.BldgT
ype == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['1Fam'],d_data
['TwnhsE'],d_data['Duplex'],d_data['Twnhs'],d_da
ta['2fmCon'])
    print('BldgType v Variable {} Result {}'.for
mat(i,anova))
```

```
BldgType v Variable SalePrice Result F_onewayRes
ult(statistic=18.203774908534502, pvalue=1.33235
66234678146e-14)
```

In [33]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.MasVnrType.value
s)
    d_data = {grp:housetrain[i][housetrain.MasVn
rType == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['BrkFace'],d_d
ata['Stone'],d_data['BrkCmn'])
    print('MasVnrType v Variable {} Result {}'.f
ormat(i,anova))
```

```
MasVnrType v Variable SalePrice Result F_onewayR
esult(statistic=35.967136674498157, pvalue=1.871
3951553943954e-15)
```

In [34]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.FireplaceQu.valu
es)
    d_data = {grp:housetrain[i][housetrain.Firep
laceQu == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['Gd'],d_data[
'TA'],d_data['Fa'],d_data['Ex'],d_data['Po'])
    print('FireplaceQu v Variable {} Result {}'.
format(i,anova))
```

```
FireplaceQu v Variable SalePrice Result F_oneway
Result(statistic=21.533072284020118, pvalue=7.58
69862969595068e-17)
```

In [39]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.GarageType.value
s)
    d_data = {grp:housetrain[i][housetrain.Garag
eType == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['Attchd'],d_da
ta['Detchd'],d_data['Other'])
    print('GarageType v Variable {} Result {}'.f
ormat(i,anova))
```

```
GarageType v Variable SalePrice Result F_onewayR
esult(statistic=238.85681699806867, pvalue=9.800
2008988291577e-90)
```

In [ ]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.GarageFinish.val
ues)
```

```
    d_data = {grp:housetrain[i][housetrain.Garag
eFinish == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['Unf'],d_data[
'RFn'],d_data['Fin'],d_data['No'])
    print('GarageFinish v Variable {} Result {}'
.format(i,anova))
```

In [ ]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.PoolQC.values)
    d_data = {grp:housetrain[i][housetrain.PoolQ
C == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['Fa'],d_data[
'Gd'],d_data['Ex'])
    print('PoolQC v Variable {} Result {}'.forma
t(i,anova))
```

In [36]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.MiscFeature.valu
es)
    d_data = {grp:housetrain[i][housetrain.MiscF
eature == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['Shed'],d_data
['Gar2'],d_data['Othr'],d_data['TenC'])
    print('MiscFeature v Variable {} Result {}'.
format(i,anova))
```

```
MiscFeature v Variable SalePrice Result F_oneway
Result(statistic=1.906895927396892, pvalue=0.140
46501633657071)
```

In [42]:

```
#Should there only be one group??
for i in (['SalePrice']):
    grps = pd.unique(housetrain.SaleCondition.va
lues)
    d_data = {grp:housetrain[i][housetrain.SaleC
ondition == grp] for grp in grps}
        #run anova
    anova = stats.f_oneway(d_data['Normal'],d_da
ta['Partial'],d_data['Abnorml'],d_data['Other'])
    print('SaleCondition v Variable {} Result {}
'.format(i,anova))
```

```
SaleCondition v Variable SalePrice Result F_onew
ayResult(statistic=65.681410331609442, pvalue=7.
9635777855902515e-40)
```

Our ANOVA analysis resulted in the following correlations:

- **SalePrice** is positively correlated with Utilities and PoolQC

Base Model

Multiple linear regression attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data. Every value of the independent variable x is associated with a value of the dependent variable y. In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models.Most commonly, the conditional mean of y given the value of X is assumed to be an affine function of X, where X is our predictor variables. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of y given X, rather than on the joint probability distribution of y and X, which is the domain of multivariate analysis.

Linear regression was the first type of regression analysis to be studied rigorously and to be used extensively in practical applications. This is because models which depend linearly on their unknown parameters are easier to fit than models which are non-linearly related to their parameters and because the statistical properties of the resulting estimators are easier to determine. Linear regression models are often fitted using the least squares approach, but they may also be fitted in other ways, such as by minimizing the "lack of fit" in some other norm (as with least absolute deviations regression), or by minimizing a penalized version of the least squares cost function as in ridge regression (L2-norm penalty) and lasso (L1-norm penalty). We will be looking at some of these alternate versions in further advanced models.

Assumptions

- Weak exogeneity
- Constant variance
- Linearity
- Lack of perfect multicollinearity
- Independence of errors

Below, we begin to build our model. An important thing to note is that dimensionaility can cause serious problems with MLR, meaning we will also need to incorporate some feature selection techniques in order to reduce the number of predictor variables. This should help reduce the possibility of overfitting, as well.

In [45]:

```
# Importing packages for linear regression
from sklearn.feature_selection import RFE
from sklearn import linear_model

from sklearn.metrics import mean_squared_error,
r2_score
from sklearn.feature_selection import RFE

# creating a copy of the data set
testtrain=pd.DataFrame.copy(housetrain)
testtrain=testtrain.drop('Utilities',axis=1)
testtrain=testtrain.drop('PoolQC',axis=1)
testtrain=testtrain.drop('PoolArea',axis=1)
# creating dummy variables for categorical varia
bles
group = testtrain.columns.to_series().groupby(te
sttrain.dtypes).groups # grouping columns by typ
e
groups={k.name: v for k, v in group.items()} #
 creating as dictionary

dummies = pd.get_dummies(testtrain[groups['objec
t'].values])
testrain = testtrain.join(dummies)

# dropping ID so its not included in the analysi
s
testrain=testrain.drop('Id',axis=1)

# Pulling our target variable into its own dataf
rame
y=pd.DataFrame.copy(testrain['SalePrice'])

# Dropping variables that had been recoded
testrain=testrain.drop('SalePrice',axis=1)
testrain=testrain.drop(testrain[groups['object']
.values],axis=1)
testrain=testrain.drop('GarageYrBlt',axis=1)
testrain=testrain.drop('OpenPorchSF',axis=1)
testrain=testrain.drop('EnclosedPorch',axis=1)
testrain=testrain.drop('3SsnPorch',axis=1)
testrain=testrain.drop('ScreenPorch',axis=1)
testrain=testrain.drop('BsmtFinSF1',axis=1)
testrain=testrain.drop('BsmtFinSF2',axis=1)

#Variables not present in testing set so wont ex
plain any variance

testrain=testrain.drop('Condition2_Rail',axis=1)
testrain=testrain.drop('Heating_Floor',axis=1)
testrain=testrain.drop('Heating_OthW',axis=1)
testrain=testrain.drop('Electrical_Mix',axis=1)
testrain=testrain.drop('MiscFeature_TenC',axis=1
)
```

You may have noticed that we have removed additional columns from our data set. These dummy variables were removed because they do not exist in the test data set, so they will provide no explained variance in the test set, hence we drop them so they will not be included in the model. The variables in question are highlighted in the code above using comments.

In [46]:

```
# imporing packages
from sklearn import linear_model
from sklearn.metrics import mean_squared_error,
r2_score

# creating linear regression object
regr = linear_model.LinearRegression()

X=pd.DataFrame.copy(testrain)
# splitting the data into testing and training
X_train, X_test, y_train, y_test = train_test_sp
lit(X, y, test_size=0.3)
# Train the model using the training sets

# fitting our model to the data
regr.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = regr.predict(X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % np.sqrt(mean_squared_error(y_test, y_pre
d)))
# Explained variance score: 1 is perfect predict
ion
print('Variance score: %.2f' % r2_score(y_test,
y_pred))
```

```
Coefficients: 
 [  1.28442344e-02   8.14639543e-02  -3.35677286
e-03  -3.84009175e-02
  -3.30758326e-03   8.02472984e-02   8.67011072e
-02   7.69720757e-02
   3.47932527e-02  -7.24602596e-03   1.88142133e
-02   1.66699455e-03
  -2.26420152e-02   4.92423742e-02   4.64117670e
-03   3.24534964e-02
  -2.51945956e-03  -6.43366301e-02   2.71279465e
-01   9.22643909e-03
   1.28225763e-02   6.20489227e-02   3.21412950e
-02  -9.73143961e-02
   3.15013188e-01   6.07551894e-03   8.80458204e
-03   7.30168981e-02
   4.61432968e-02  -1.39123988e-02  -1.48073434e
-01   2.02894438e-02
```

```
  -6.19407943e-03   1.22316615e-01   6.07612500e
-02   5.99804963e-02
   6.81479046e-02   5.62784121e-02   4.43563728e
-02  -7.38684842e-03
   1.35642507e-02  -1.33147523e-02   1.58416454e
-01   5.50659643e-03
  -2.92053483e-03   1.89322219e-02   1.38094077e
-02   3.00453329e-01
  -2.81263214e+09  -2.81263214e+09  -2.81263214e
+09   1.28858119e+09
  -2.81263214e+09  -2.81263214e+09  -2.81263214e
+09  -2.81263214e+09
  -2.81263214e+09  -2.81263214e+09  -2.81263214e
+09  -2.81263214e+09
  -2.81263214e+09  -2.81263214e+09   4.03386104e
+09  -2.23791330e+10
  -2.23791330e+10  -2.23791330e+10  -2.23791330e
+10  -2.23791330e+10
  -2.43087800e+09  -2.43087800e+09  -2.45224191e
+10  -2.45224191e+10
  -2.45224191e+10  -6.87052352e+10  -6.87052352e
+10  -6.87052352e+10
  -6.87052352e+10  -1.73890715e+10  -1.73890715e
+10  -1.73890715e+10
   2.51746494e+10   2.51746494e+10   2.51746494e
+10   2.51746494e+10
   2.51746494e+10   1.08950501e-02   5.74917287e
-02   1.07801445e-02
   5.07164962e-02   4.76688742e+09   6.65674097e
+08  -2.07960576e+09
   4.76688742e+09   4.76688742e+09  -5.99797860e
+09  -5.99797860e+09
  -5.99797860e+09  -5.99797860e+09   4.95543343e
+09   4.95543343e+09
   4.95543343e+09   1.11530684e+10   1.11530684e
+10   3.59012913e+10
   3.59012913e+10   3.59012913e+10   3.59012913e
+10   3.59012913e+10
   7.85293985e+09   7.85293985e+09   7.85293985e
+09   7.85293985e+09
   7.85293985e+09  -8.04639117e-03   7.73033451e
-03   2.98259385e-02
   7.33600354e+10   7.33600354e+10   7.33600354e
+10   2.32080915e-02
   7.39855880e-02   9.81554061e-02  -2.86524066e
-02  -8.47547222e-02
   5.59309128e-03  -4.81850505e-02  -7.95823356e
+09  -7.95823356e+09
  -7.95823356e+09  -7.95823356e+09  -7.95823356e
+09  -7.95823356e+09
   3.64628813e+09   3.64628813e+09   2.77852409e
+09   3.64628813e+09
   1.25697199e+10   1.34374839e+10   1.25697199e
+10   1.25697199e+10
   2.09255873e-01   2.09688252e-01   2.24185551e
-01   1.74071135e-01
   1.53518789e+10   1.53518789e+10   1.53518789e
+10   1.53518789e+10
```

```
+10   1.53518789e+10
   1.53518789e+10   2.66158963e+10   2.66158963e
+10   2.66158963e+10
   2.66158963e+10]
Mean squared error: 196187639.30
Variance score: -512348367814795328.00
```

As we can see, the model shows over 90% accuracy, but overfitting will clearly be an issue due to the 200+ variables used. When dealing with dimensionailty issues, it is important to incorporate feature selection. There are several options for feature selection. In this next section we will examine Recursive Feature Selection. Recursive feature elimination is based on the idea to repeatedly construct a model (for example a regression model) and choose either the best or worst performing feature (for example based on coefficients), setting the feature aside and then repeating the process with the rest of the features. This process is applied until all features in the dataset are exhausted. Features are then ranked according to when they were eliminated. As such, it is a greedy optimization for finding the best performing subset of features.

The stability of RFE depends heavily on the type of model that is used for feature ranking at each iteration. Some benefits of feature selection are:

- Reduces Overfitting: Less redundant data means less opportunity to make decisions based on noise.
- Improves Accuracy: Less misleading data means modeling accuracy improves.
- Reduces Training Time: Less data means that algorithms train faster.

In [47]:

```
from sklearn.feature_selection import RFE
# load data

# arbitrarily decide to keep 85 variables
rfe = RFE(regr, 85)
rfe.fit(testrain, y)
#print("Num Features: %d") % fit.n_features_
# print("Selected Features: {}".format(fit.n_fea
tures_))
# print("Selected Features: {}".format(fit.suppo
rt_))
# print("Selected Features: {}".format(fit.ranki
ng_))


X_train, X_test, y_train, y_test = train_test_sp
lit(testrain, y, test_size=0.3)

# Make predictions using the testing set
y_pred = rfe.predict(X_test)
```

```
# The coefficients

# The mean squared error
print("Mean squared error: %.5f"
      % np.sqrt(mean_squared_error(y_test, y_pre
d)))
# Explained variance score: 1 is perfect predict
ion
print('Variance score: %.5f' % r2_score(y_test,
y_pred))
```

```
Mean squared error: 0.14116
Variance score: 0.75440
```

We now have a RMSE of 0.15, and an R squared value of around 70%. The R squared value tells us the amount of variance our model explains. Using feature selection has substantianally reduced the error while simultaneously increasing the varaince our model explains. As this is our base model, the number of variables to keep was arbitrarily choosen. Later we will use a loop to try and find a more optimal solution.

An important thing to note here is that the model failed to include any continious variables in the model. In our opinion, this is a failing of the model that will need to be addressed, because variables like square footage and overall quality definitely impact the selling price of a house. In order to prove this, we decided to build a model using only numeric and ordinal data.

In [48]:

```
quantvar = ['LotFrontage','LotArea','MasVnrArea'
,'BsmtFinSF','BsmtUnfSF','TotalBsmtSF','1stFlrS
F','2ndFlrSF',
            'LowQualFinSF','GrLivArea','GarageAr
ea','WoodDeckSF','PorchSF'
           ,'MiscVal']
intvar = ['BsmtCond','BsmtExposure','Fence','Lot
Shape','CentralAir','LandContour','PavedDrive',
'BsmtQual','ExterCond',
          'ExterQual','BsmtFinType1','BsmtFinTyp
e2','Functional','GarageCond','GarageQual','Heat
ingQC','KitchenQual','LandSlope'
         ,'OverallQual','OverallCond']

rfe = RFE(regr, 25)
rfe.fit(testrain[quantvar+intvar], y)
#print("Num Features: %d") % fit.n_features_
# print("Selected Features: {}".format(fit.n_fea
tures_))
# print("Selected Features: {}".format(fit.suppo
rt_))
# print("Selected Features: {}".format(fit.ranki
ng_))
```

```
X_train, X_test, y_train, y_test = train_test_sp
lit(testrain[quantvar+intvar], y, test_size=0.4,
random_state=0)

# Make predictions using the testing set
y_pred = rfe.predict(X_test)

# The coefficients

# The mean squared error
print("Mean squared error: %.5f"
      % np.sqrt(mean_squared_error(y_test, y_pre
d)))
# Explained variance score: 1 is perfect predict
ion
print('Variance score: %.5f' % r2_score(y_test,
y_pred))
```

```
Mean squared error: 0.09255
Variance score: 0.89450
```

As we can see, we actually get better results than when we select 25 out of the 35 numeric variables. This shows us that we will need both categorical and continuous variables in our model as individually they are both strong so together they should augment each other. We have a RMSE of 0.09, and an R squared value of nearly 90%. This is better than the feature selection linear regression model we saw earlier.

It was decided to take a new approach in our model building. In order to improve our results we built a Lasso Regression model. Lasso stands for least absolute shrinkage and selection operator. This method performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces. Regularization is a process of introducing additional information in order to solve an ill-posed problem or to prevent overfitting. The goal of this learning problem is to find a function that fits or predicts the outcome that minimizes the expected error over all possible inputs. Lasso was originally formulated for least squares models and this simple case reveals a substantial amount about the behavior of the estimator, including its relationship to ridge regression and best subset selection and the connections between lasso coefficient estimates and so-called soft thresholding. It also reveals that (like standard linear regression) the coefficient estimates need not be unique if covariates are collinear.

In [49]:

```
X=pd.DataFrame.copy(testrain)

# creating list of choices for alpha
clf = linear_model.LassoCV(alphas=[0.0001, 0.000
```

```
3, 0.0006, 0.001, 0.003, 0.006, 0.01, 0.03, 0.06
, 0.1,
                          0.3, 0.6, 1])
# splitting data
X_train, X_test, y_train, y_test = train_test_sp
lit(X, y, test_size=0.3)

# fitting data
clf.fit(X_train, y_train)

#Highlighting the best alpha
alpha = clf.alpha_
print("Best alpha :", alpha)
```

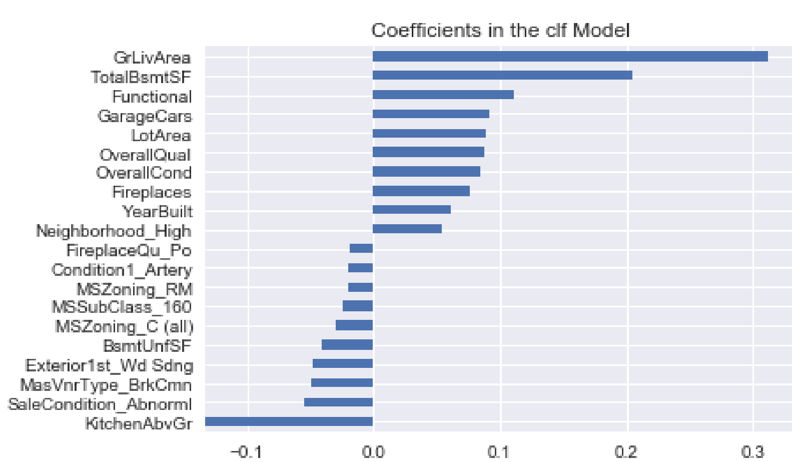
```
Best alpha : 0.0003
```

The alpha here refers to the constant that multiplies the L1 term. The L1 term is used in the regularisation, meaning it will influence the model significantly if not calculated correctly. Therefore having an appropriate alpha is important. Fortunately, in python the lasso regression package will pick the most optimol one from a provided list, as seen above.

In [50]:

```
# Predicting

y_test_las = clf.predict(X_test)
coefs = pd.Series(clf.coef_, index = X_train.col
umns)
print("clf picked " + str(sum(coefs != 0)) + " f
eatures and eliminated the other " +  \
      str(sum(coefs == 0)) + " features")
imp_coefs = pd.concat([coefs.sort_values().head(
10),
                     coefs.sort_values().tail(10
)])
imp_coefs.plot(kind = "barh")
plt.title("Coefficients in the clf Model")
plt.show()
```

```
clf picked 84 features and eliminated the other
69 features
```



Above we can see how many features were selected for our model. The graph then shows us the top ten postive coefficients and bottom ten negative coefficients. The first thing we noticed is that the lasso model picks a both numeric and categorical data. Below we see the results of the model.

In [51]:

```
from sklearn.linear_model import LassoCV
var=r2_score(y_test, y_test_las)

rmse=np.sqrt(mean_squared_error(y_test, y_test_l
as))

kfold = model_selection.KFold(n_splits=10)
modelCV = LassoCV()
scoring = 'neg_mean_absolute_error'


results = model_selection.cross_val_score(modelC
V, X_train, y_train, cv=kfold, scoring=scoring)

print('Var {}, RMSE {}'.format(var,rmse))
print(results)
```

```
Var 0.9134558607083294, RMSE 0.08233671932170174
[-0.06490223 -0.06555086 -0.06418607 -0.05736783
-0.05362963 -0.05552695
 -0.06925211 -0.05566504 -0.07701203 -0.0564437
9]
```

This is our best model so far. We have over 90% of our variance explained and also have the lowest RMSE which is around 0.08. Additionally, we used a k-fold cross validation in order to test for overfitting. We are looking for extreme value differences in the results. As we can see, all the values in our array are reasonably close together showing no obvious signs of overfitting.

Univariate Feature Selection

In [75]:

```
from sklearn.feature_selection import SelectKBes
t
from sklearn.feature_selection import chi2

X=pd.DataFrame.copy(testrain)

#X.shape

X_new = SelectKBest(chi2, k=2).fit_transform(X,
y)
#X_new.shape
```

```
--------------------------------------------------
--------------------------
```

```
ValueError                                Traceb
ack (most recent call last)
<ipython-input-75-1ff0ff011597> in <module>()
      6 #X.shape
      7
----> 8 X_new = SelectKBest(chi2, k=4).fit_trans
form(X, y)
      9 #X_new.shape

C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\sklearn\base.py in fi
t_transform(self, X, y, **fit_params)
    495         else:
    496             # fit method of arity 2 (sup
ervised transformation)
--> 497             return self.fit(X, y, **fit_
params).transform(X)
    498
    499

C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\sklearn\feature_selec
tion\univariate_selection.py in fit(self, X, y)
    328
    329         self._check_params(X, y)
--> 330         score_func_ret = self.score_func
(X, y)
    331         if isinstance(score_func_ret, (l
ist, tuple)):
    332             self.scores_, self.pvalues_
= score_func_ret

C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\sklearn\feature_selec
tion\univariate_selection.py in chi2(X, y)
    215         raise ValueError("Input X must b
e non-negative.")
    216
--> 217     Y = LabelBinarizer().fit_transform(y
)
    218     if Y.shape[1] == 1:
    219         Y = np.append(1 - Y, Y, axis=1)

C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\sklearn\base.py in fi
t_transform(self, X, y, **fit_params)
    492         if y is None:
    493             # fit method of arity 1 (uns
upervised transformation)
--> 494             return self.fit(X, **fit_par
ams).transform(X)
    495         else:
    496             # fit method of arity 2 (sup
ervised transformation)

C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\sklearn\preprocessing
\label.py in fit(self, y)
```

```
\label.py in fit(self, y)
    302
    303         self.sparse_input_ = sp.issparse
(y)
--> 304         self.classes_ = unique_labels(y)
    305         return self
    306

C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\sklearn\utils\multicl
ass.py in unique_labels(*ys)
     96     _unique_labels = _FN_UNIQUE_LABELS.g
et(label_type, None)
     97     if not _unique_labels:
---> 98         raise ValueError("Unknown label
 type: %s" % repr(ys))
     99
    100     ys_labels = set(chain.from_iterable(
_unique_labels(y) for y in ys))

ValueError: Unknown label type: (array([ 0.71564
88 ,  0.58699554,  0.7719741 , ...,  0.88290328,
        0.3454626 ,  0.38061031]),)
```

Tree-Based Feature Selection

In [76]:

```
from sklearn.ensemble import ExtraTreesClassifie
r
from sklearn.feature_selection import SelectFrom
Model

X=pd.DataFrame.copy(testrain)

X.shape

clf = ExtraTreesClassifier()
clf = clf.fit(X, y)
clf.feature_importances_

model = SelectFromModel(clf, prefit=True)
X_new = model.transform(X)
X_new.shape
```

```
------------------------------------------------
---------------------------
ValueError                                Traceb
ack (most recent call last)
<ipython-input-76-3d9a3a09347c> in <module>()
      7
      8 clf = ExtraTreesClassifier()
----> 9 clf = clf.fit(X, y)
     10 clf.feature_importances_
     11

C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\sklearn\ensemble\fore
```

```
st.py in fit(self, X, y, sample_weight)
    269         self.n_outputs_ = y.shape[1]
    270
--> 271         y, expanded_class_weight = self.
_validate_y_class_weight(y)
    272
    273         if getattr(y, "dtype", None) !=
 DOUBLE or not y.flags.contiguous:

C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\sklearn\ensemble\fore
st.py in _validate_y_class_weight(self, y)
    455
    456     def _validate_y_class_weight(self, y
):
--> 457         check_classification_targets(y)
    458
    459         y = np.copy(y)

C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\sklearn\utils\multicl
ass.py in check_classification_targets(y)
    170     if y_type not in ['binary', 'multicl
ass', 'multiclass-multioutput',
    171               'multilabel-indicator', 'mul
tilabel-sequences']:
--> 172         raise ValueError("Unknown label
 type: %r" % y_type)
    173
    174

ValueError: Unknown label type: 'continuous'
```

Recursive Feature Elimination

In [77]:

```
from sklearn.svm import SVC
from sklearn.feature_selection import RFE
import matplotlib.pyplot as plt

X=pd.DataFrame.copy(testrain)

# Create the RFE object and rank each pixel
svc = SVC(kernel="linear", C=1)
rfe = RFE(estimator=svc, n_features_to_select=1,
step=1)
rfe.fit(X, y)
ranking = rfe.ranking_.reshape(testrain.images[0
].shape)

# Plot pixel ranking
plt.matshow(ranking, cmap=plt.cm.Blues)
plt.colorbar()
plt.title("Ranking of pixels with RFE")
plt.show()
```

```
-----------------------------------------------
```

```
---------------------------
ValueError                                Traceb
ack (most recent call last)
<ipython-input-77-70f46b2fe5a1> in <module>()
      8 svc = SVC(kernel="linear", C=1)
      9 rfe = RFE(estimator=svc, n_features_to_s
elect=1, step=1)
---> 10 rfe.fit(X, y)
     11 ranking = rfe.ranking_.reshape(testrain.
images[0].shape)
     12

C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\sklearn\feature_selec
tion\rfe.py in fit(self, X, y)
    133             The target values.
    134         """
--> 135         return self._fit(X, y)
    136
    137     def _fit(self, X, y, step_score=None
):

C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\sklearn\feature_selec
tion\rfe.py in _fit(self, X, y, step_score)
    167                 print("Fitting estimator
with %d features." % np.sum(support_))
    168
--> 169             estimator.fit(X[:, features]
, y)
    170
    171             # Get coefs

C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\sklearn\svm\base.py i
n fit(self, X, y, sample_weight)
    150
    151         X, y = check_X_y(X, y, dtype=np.
float64, order='C', accept_sparse='csr')
--> 152         y = self._validate_targets(y)
    153
    154         sample_weight = np.asarray([]

C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\sklearn\svm\base.py i
n _validate_targets(self, y)
    518     def _validate_targets(self, y):
    519         y_ = column_or_1d(y, warn=True)
--> 520         check_classification_targets(y)
    521         cls, y = np.unique(y_, return_in
verse=True)
    522         self.class_weight_ = compute_cla
ss_weight(self.class_weight, cls, y_)

C:\Users\danielle.ezzo\AppData\Local\Continuum\A
naconda3\lib\site-packages\sklearn\utils\multicl
ass.py in check_classification_targets(y)
```

```
    170     if y_type not in ['binary', 'multicl
ass', 'multiclass-multioutput',
    171             'multilabel-indicator', 'mul
tilabel-sequences']:
--> 172         raise ValueError("Unknown label
 type: %r" % y_type)
    173
    174

ValueError: Unknown label type: 'continuous'
```

Decision Tree Regression

In [78]:

```
#Recursive Feature Elimination for Decision Tree
Regression

from sklearn.feature_selection import RFE
X=pd.DataFrame.copy(testrain)

from sklearn.model_selection import train_test_s
plit
X_train, X_test, y_train, y_test = train_test_sp
lit(X, y, test_size=0.2, random_state=0)

# create a base classifier used to evaluate a su
bset of attributes
model = DecisionTreeRegressor()
# create the RFE model and select 3 attributes
rfe = RFE(model, 90)
rfe = rfe.fit(X_train, y_train)
# summarize the selection of the attributes
print(rfe.support_)
print(rfe.ranking_)
```

```
[ True  True  True  True False  True  True  True
True  True  True  True
  True  True  True  True  True  True  True  True
True  True  True False
  True  True  True  True  True  True False  True
True  True  True  True
  True  True  True False  True  True False  True
True  True  True False
 False False False False  True False False False
True  True False False
 False False False False False False  True False
False False False  True
 False  True  True False  True  True  True  True
False False  True  True
 False False False False False  True False False
False False False False
 False False  True  True False  True False  True
True  True  True  True
  True  True  True  True  True False  True False
True  True  True False
 False False False False False  True False False
True  True False  True
```

```
  True  True  True  True  True False  True  True
False False False False
  True False  True False  True  True  True  True
False]
[ 1  1  1  1 14  1  1  1  1  1  1  1  1  1  1  1
1  1  1  1  1  1  1  4  1
  1  1  1  1  1 18  1  1  1  1  1  1  1  1 28  1
1 22  1  1  1  1 48 64 27
 26 30  1 35 40 42  1  1 49 45  5 57 58 54 11 62
1 13 44 17 19  1 60  1  1
 47  1  1  1  1 10  2  1  1 15 39 50 55 61  1  9
31 36 24  6 33 38 59  1  1
 51  1 29  1  1  1  1  1  1  1  1  1  1 41  1 12
1  1  1 21 20 25 32 34 46
  1 52 23  1  1 53  1  1  1  1  1  1  8  1  1 56
3 43 37  1 16  1  7  1  1
  1  1 63]
```

In [53]:

```
X=pd.DataFrame.copy(testrain)

from sklearn.model_selection import train_test_s
plit
X_train, X_test, y_train, y_test = train_test_sp
lit(X, y, test_size=0.2, random_state=0)

from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)
```

Out[53]:

```
DecisionTreeRegressor(criterion='mse', max_depth
=None, max_features=None,
           max_leaf_nodes=None, min_impurity_spl
it=1e-07,
           min_samples_leaf=1, min_samples_split
=2,
           min_weight_fraction_leaf=0.0, presort
=False, random_state=None,
           splitter='best')
```

In [85]:

```
print(regressor.feature_importances_)
```

```
[  3.67907123e-03   7.48475760e-03   8.55243249e
-04   2.65816090e-04
   3.96647335e-07   5.24634678e-01   4.85871324e
-03   1.85008632e-02
   8.83559075e-03   4.92374255e-03   6.18734655e
-04   8.59427886e-04
   1.48969493e-02   1.22879380e-04   2.79824353e
-03   1.86791269e-03
   2.22939256e-05   1.71782606e-03   4.86855958e
-02   1.37184247e-03
   1.14407196e-03   1.99417899e-02   2.65511730e
-03   0.00000000e+00
```

```
   1.27588086e-01   4.35891146e-04   3.41730918e
-04   2.50079073e-04
   4.19044054e-03   1.57042282e-03   1.24518749e
-06   3.42601203e-03
   3.40609787e-03   1.58663218e-03   9.01041747e
-05   6.24105066e-03
   5.26658803e-02   7.27365770e-04   6.07392940e
-04   2.67327617e-04
   3.34828592e-03   1.56746810e-04   0.00000000e
+00   2.38063838e-03
   1.49718610e-03   4.44532834e-02   5.10134976e
-03   0.00000000e+00
   1.36210456e-08   0.00000000e+00   0.00000000e
+00   0.00000000e+00
   3.16529716e-07   0.00000000e+00   0.00000000e
+00   0.00000000e+00
   8.29580288e-04   9.90119072e-06   4.83038188e
-06   7.40476427e-04
   9.28117024e-07   2.97049044e-06   0.00000000e
+00   0.00000000e+00
   3.75083369e-05   1.46384456e-04   3.81138143e
-05   0.00000000e+00
   0.00000000e+00   0.00000000e+00   1.15524938e
-05   3.45189207e-04
   0.00000000e+00   6.09769833e-04   0.00000000e
+00   1.15804128e-05
   2.85558014e-04   1.25379388e-02   2.07899981e
-02   1.14884817e-03
   0.00000000e+00   1.27582811e-04   6.52279623e
-04   3.80496000e-03
   1.30117547e-05   0.00000000e+00   0.00000000e
+00   3.29877608e-05
   0.00000000e+00   4.75009276e-05   0.00000000e
+00   1.63063827e-05
   5.30879733e-06   1.07201306e-07   5.80034909e
-05   1.08598931e-04
   0.00000000e+00   4.24946281e-07   1.31511463e
-04   4.63463456e-05
   5.57936169e-04   2.91145519e-05   3.99770957e
-04   2.74857673e-04
   2.38614582e-04   3.09042721e-05   2.07701346e
-04   5.69150361e-05
   1.48630798e-04   7.42900948e-04   5.03222108e
-05   1.23820617e-06
   1.61022015e-04   0.00000000e+00   2.32373480e
-04   2.19834600e-07
   1.65583748e-04   1.72143084e-03   3.71966403e
-06   0.00000000e+00
   0.00000000e+00   0.00000000e+00   8.87359849e
-05   0.00000000e+00
   0.00000000e+00   1.87126644e-05   0.00000000e
+00   5.46794717e-08
   1.31617620e-04   9.24028972e-03   0.00000000e
+00   1.03845726e-04
   1.56636676e-03   2.03515036e-03   4.82031963e
-06   6.15012707e-04
   1.98349493e-04   3.60676015e-07   4.64782898e
-04   8.71659103e-05
```

```
-04   8.71659105e-05
   0.00000000e+00   3.19539003e-07   4.99040190e
-05   2.33038622e-06
   3.38090683e-05   5.12210740e-04   5.89853003e
-05   0.00000000e+00
   1.03094744e-04   1.27769719e-04   1.86465151e
-03   3.91942258e-03
   7.77840297e-05]
```

In [55]:

```
y_pred = regressor.predict(X_test)
df=pd.DataFrame({'Actual':y_test, 'Predicted':y_
pred})
df
```

Out[55]:

	Actual	Predicted
511	0.690864	0.532047
963	0.819813	0.802977
231	0.987251	0.999039
688	0.984764	0.791537
34	0.901777	0.808407
302	0.700980	0.793248
527	0.993635	0.982641
925	0.550825	0.422696
647	0.429103	0.254024
54	0.266729	0.331551
1267	0.981026	0.874233
619	0.937030	0.954550
175	0.830605	0.822568
141	0.870189	0.650984
1059	0.759755	0.793248
333	0.709435	0.578818
361	0.364282	0.447548
541	0.843268	0.788499
1028	0.122627	0.201756
540	0.946493	0.943694
1273	0.562156	0.491393
1047	0.364282	0.370819
52	0.148126	0.377349

1452	0.364282	0.383869
408	0.905642	0.999039
1103	0.457631	0.403356
1092	0.308694	0.438670
31	0.392654	0.279546
1256	0.933344	0.980691
517	0.880078	0.897761
1019	0.735547	0.636316
909	0.545095	0.485516
1127	0.868121	0.578269
849	0.616077	0.545095
726	0.766804	0.822568
1340	0.222886	0.130052
1272	0.311952	0.175597
1100	0.005014	0.303813
445	0.250867	0.241451
855	0.247720	0.357739
667	0.648570	0.646144
...	...	...
1269	0.357739	0.447548
957	0.279546	0.132571
706	0.933883	0.892734
440	0.998824	0.988614
322	0.932801	0.822568
635	0.678991	0.232121
483	0.485578	0.473236
29	0.012535	0.164391
899	0.298940	0.279546
1252	0.266729	0.099372
959	0.429103	0.491698
501	0.782674	0.613493
431	0.032036	0.247720
124	0.584281	0.485516
198	0.117788	0.232121

789	0.618650	0.678991
1412	0.060433	0.072859
965	0.572756	0.542214
746	0.811317	0.819813
549	0.876210	0.777045
279	0.641255	0.780371
1337	0.001796	0.022139
1290	0.581555	0.376044
519	0.805457	0.946493
308	0.038295	0.210735
1222	0.351192	0.311952
643	0.409822	0.553674
317	0.889254	0.941951
1014	0.199975	0.235221
1072	0.065597	0.332860
481	0.979583	0.944267
61	0.103832	0.099372
491	0.285991	0.409822
436	0.181301	0.046416
92	0.482505	0.282766
1338	0.678991	0.740956
310	0.495354	0.605668
1430	0.641942	0.305439
40	0.460767	0.416269
1243	0.995271	0.905642
75	0.063848	0.046129

292 rows × 2 columns

In [57]:

```
print('Mean Absolute Error:', metrics.mean_absol
ute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_square
d_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metric
s.mean_squared_error(y_test, y_pred)))
print('Variance:', metrics.r2_score(y_test,y_pre
d))
```

```
Mean Absolute Error: 0.0995033911091
Mean Squared Error: 0.0178736615979
Root Mean Squared Error: 0.133692414137
Variance: 0.773872543422
```

Random Forest

In [81]:

```
from sklearn.ensemble import RandomForestRegress
or
from sklearn.datasets import make_regression

X=pd.DataFrame.copy(testrain)

from sklearn.model_selection import train_test_s
plit
X_train, X_test, y_train, y_test = train_test_sp
lit(X, y, test_size=0.2, random_state=0)


regr = RandomForestRegressor(max_depth=2, random
_state=0)

regr.fit(X_train, y_train)
```

Out[81]:

```
RandomForestRegressor(bootstrap=True, criterion
='mse', max_depth=2,
           max_features='auto', max_leaf_nodes=N
one,
           min_impurity_split=1e-07, min_samples
_leaf=1,
           min_samples_split=2, min_weight_fract
ion_leaf=0.0,
           n_estimators=10, n_jobs=1, oob_score=
False, random_state=0,
           verbose=0, warm_start=False)
```

In [82]:

```
y_pred = regr.predict(X_test)
df=pd.DataFrame({'Actual':y_test, 'Predicted':y_
pred})
df
```

Out[82]:

	Actual	Predicted
511	0.690864	0.649398
963	0.819813	0.870797
231	0.987251	0.870797
688	0.984764	0.649398

34	0.901777	0.695996
302	0.700980	0.870797
527	0.993635	0.870797
925	0.550825	0.298313
647	0.429103	0.276852
54	0.266729	0.339910
1267	0.981026	0.870797
619	0.937030	0.870797
175	0.830605	0.433141
141	0.870189	0.870797
1059	0.759755	0.454603
333	0.709435	0.649398
361	0.364282	0.391283
541	0.843268	0.870797
1028	0.122627	0.298052
540	0.946493	0.870797
1273	0.562156	0.276590
1047	0.364282	0.298313
52	0.148126	0.276852
1452	0.364282	0.276852
408	0.905642	0.870797
1103	0.457631	0.298313
1092	0.308694	0.454603
31	0.392654	0.234994
1256	0.933344	0.870797
517	0.880078	0.870797
1019	0.735547	0.695996
909	0.545095	0.454603
1127	0.868121	0.783401
849	0.616077	0.454603
726	0.766804	0.454603
1340	0.222886	0.298313
1272	0.311952	0.234994
1100	0.005014	0.256456

445	0.250867	0.433141
855	0.247720	0.234994
667	0.648570	0.454603
...	...	...
1269	0.357739	0.412745
957	0.279546	0.276852
706	0.933883	0.870797
440	0.998824	0.870797
322	0.932801	0.695996
635	0.678991	0.412745
483	0.485578	0.298313
29	0.012535	0.234994
899	0.298940	0.254326
1252	0.266729	0.256456
959	0.429103	0.649398
501	0.782674	0.783401
431	0.032036	0.234994
124	0.584281	0.454603
198	0.117788	0.391283
789	0.618650	0.454603
1412	0.060433	0.298313
965	0.572756	0.454603
746	0.811317	0.695996
549	0.876210	0.783401
279	0.641255	0.695996
1337	0.001796	0.234994
1290	0.581555	0.298313
519	0.805457	0.695996
308	0.038295	0.276852
1222	0.351192	0.412745
643	0.409822	0.454603
317	0.889254	0.870797
1014	0.199975	0.256456
1072	0.065597	0.391283

481	0.979583	0.870797
61	0.103832	0.234994
491	0.285991	0.412745
436	0.181301	0.276852
92	0.482505	0.298313
1338	0.678991	0.695996
310	0.495354	0.454603
1430	0.641942	0.454603
40	0.460767	0.339910
1243	0.995271	0.870797
75	0.063848	0.234994

292 rows × 2 columns

In [83]:

```
print(regr.feature_importances_)
```

```
[ 0.          0.          0.          0.
0.          0.78883094
  0.          0.          0.          0.
0.          0.          0.
  0.          0.          0.          0.
0.          0.          0.
  0.          0.          0.          0.
0.1079397   0.          0.
  0.          0.          0.          0.
0.          0.          0.
  0.          0.02525917  0.06440483  0.
0.          0.          0.
  0.          0.          0.          0.
0.          0.          0.
  0.          0.          0.          0.
0.          0.          0.
  0.          0.          0.          0.
0.          0.          0.
  0.          0.          0.          0.
0.          0.          0.
  0.          0.          0.          0.
0.          0.          0.
  0.          0.          0.01356536  0.
0.          0.          0.
  0.          0.          0.          0.
0.          0.          0.
  0.          0.          0.          0.
0.          0.          0.
  0.          0.          0.          0.
0.          0.          0.
  0.          0.          0.          0.
0.          0.          0.
```

```
  0.           0.           0.           0.
0.           0.           0.
  0.           0.           0.           0.
0.           0.           0.
  0.           0.           0.           0.
0.           0.           0.
  0.           0.           0.           0.
0.           0.           0.
  0.           0.           0.           0.
0.           0.           0.
  0.           0.           0.           0.
0.           0.           0.         ]
```

In [84]:

```
print('Mean Absolute Error:', metrics.mean_absol
ute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_square
d_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metric
s.mean_squared_error(y_test, y_pred)))
print('Variance:', metrics.r2_score(y_test,y_pre
d))
```

```
Mean Absolute Error: 0.12945015847
Mean Squared Error: 0.0252072710414
Root Mean Squared Error: 0.15876797864
Variance: 0.681091864885
```

SVM Regression

In [86]:

```
from sklearn.svm import SVR
import numpy as np

X=pd.DataFrame.copy(testrain)

from sklearn.model_selection import train_test_s
plit
X_train, X_test, y_train, y_test = train_test_sp
lit(X, y, test_size=0.2, random_state=0)


clf = SVR(C=1.0, epsilon=0.2)
clf.fit(X_train, y_train)
```

Out[86]:

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3,
epsilon=0.2, gamma='auto',
  kernel='rbf', max_iter=-1, shrinking=True, tol
=0.001, verbose=False)
```

In [88]:

```
y_pred = clf.predict(X_test)
df=pd.DataFrame({'Actual':y_test, 'Predicted':y_
pred})
```

```
df
```

Out[88]:

	Actual	Predicted
511	0.690864	0.676188
963	0.819813	0.693272
231	0.987251	0.846576
688	0.984764	0.760834
34	0.901777	0.774626
302	0.700980	0.729888
527	0.993635	0.899240
925	0.550825	0.392626
647	0.429103	0.400438
54	0.266729	0.378904
1267	0.981026	0.847157
619	0.937030	0.899657
175	0.830605	0.628381
141	0.870189	0.744331
1059	0.759755	0.579092
333	0.709435	0.709092
361	0.364282	0.339624
541	0.843268	0.728677
1028	0.122627	0.358555
540	0.946493	0.823904
1273	0.562156	0.511137
1047	0.364282	0.402324
52	0.148126	0.284283
1452	0.364282	0.335266
408	0.905642	0.782135
1103	0.457631	0.417490
1092	0.308694	0.508377
31	0.392654	0.334371
1256	0.933344	0.807742
517	0.880078	0.756649
1019	0.735547	0.625902

[illegible]	[illegible]	[illegible]
909	0.545095	0.589295
1127	0.868121	0.711847
849	0.616077	0.640126
726	0.766804	0.736151
1340	0.222886	0.226732
1272	0.311952	0.229888
1100	0.005014	0.087547
445	0.250867	0.380500
855	0.247720	0.337037
667	0.648570	0.572757
...	...	...
1269	0.357739	0.305688
957	0.279546	0.295152
706	0.933883	0.758471
440	0.998824	0.863220
322	0.932801	0.752000
635	0.678991	0.370980
483	0.485578	0.364116
29	0.012535	0.090647
899	0.298940	0.331289
1252	0.266729	0.245217
959	0.429103	0.602404
501	0.782674	0.681428
431	0.032036	0.235908
124	0.584281	0.461134
198	0.117788	0.247046
789	0.618650	0.670204
1412	0.060433	0.194659
965	0.572756	0.564094
746	0.811317	0.693104
549	0.876210	0.772577
279	0.641255	0.712881
1337	0.001796	0.094094
1290	0.581555	0.455957

1290	0.581555	0.455957
519	0.805457	0.738130
308	0.038295	0.209263
1222	0.351192	0.443100
643	0.409822	0.450727
317	0.889254	0.806381
1014	0.199975	0.297780
1072	0.065597	0.336554
481	0.979583	0.830578
61	0.103832	0.240030
491	0.285991	0.414129
436	0.181301	0.223820
92	0.482505	0.415754
1338	0.678991	0.719534
310	0.495354	0.540748
1430	0.641942	0.598558
40	0.460767	0.387206
1243	0.995271	0.859810
75	0.063848	0.218764

292 rows × 2 columns

In [89]:

```
print('Mean Absolute Error:', metrics.mean_absol
ute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_square
d_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metric
s.mean_squared_error(y_test, y_pred)))
print('Variance:', metrics.r2_score(y_test,y_pre
d))
```

```
Mean Absolute Error: 0.0959989286173
Mean Squared Error: 0.0132289158684
Root Mean Squared Error: 0.115017024255
Variance: 0.832635239164
```

Test Data Set

Data Understanding

We have looked at and preprocessed the data for the training set (above). Next we look at the test data set. The first thing we do is (above). Next we look at the test data set. The first thing we do is read in the data and get some summary statistics.

The test set contains 80 variables and 1459 rows of data. The missing varaible is the target variable SalePrice. All other variables are the same

In [63]:

```
# Reading in our test data set
housetest = pd.read_csv("test.csv",header=0,na_v
alues='None')
housetest.MSSubClass = housetest.MSSubClass.asty
pe(str)
# Data describe

print(housetest.describe())
```

```
                Id  LotFrontage       LotArea  O
verallQual  OverallCond  \
count  1459.000000  1232.000000   1459.000000  1
459.000000  1459.000000
mean   2190.000000    68.580357   9819.161069
6.078821     5.553804
std     421.321334    22.376841   4955.517327
1.436812     1.113740
min    1461.000000    21.000000   1470.000000
1.000000     1.000000
25%    1825.500000    58.000000   7391.000000
5.000000     5.000000
50%    2190.000000    67.000000   9399.000000
6.000000     5.000000
75%    2554.500000    80.000000  11517.500000
7.000000     6.000000
max    2919.000000   200.000000  56600.000000
10.000000     9.000000

         YearBuilt  YearRemodAdd   MasVnrArea
BsmtFinSF1   BsmtFinSF2  \
count  1459.000000   1459.000000  1444.000000  1
458.000000  1458.000000
mean   1971.357779   1983.662783   100.709141
439.203704    52.619342
std      30.390071     21.130467   177.625900
455.268042   176.753926
min    1879.000000   1950.000000     0.000000
0.000000     0.000000
25%    1953.000000   1963.000000     0.000000
0.000000     0.000000
50%    1973.000000   1992.000000     0.000000
350.500000     0.000000
75%    2001.000000   2004.000000   164.000000
753.500000     0.000000
max    2010.000000   2010.000000  1290.000000  4
010.000000  1526.000000

         BsmtUnfSF  TotalBsmtSF     1stFlrSF
2ndFlrSF  LowQualFinSF  \
```

```
count  1458.000000  1458.000000  1459.000000  14
59.000000   1459.000000
mean    554.294925  1046.117970  1156.534613   3
25.967786      3.543523
std     437.260486   442.898624   398.165820   4
20.610226     44.043251
min       0.000000     0.000000   407.000000
0.000000      0.000000
25%     219.250000   784.000000   873.500000
0.000000      0.000000
50%     460.000000   988.000000  1079.000000
0.000000      0.000000
75%     797.750000  1305.000000  1382.500000   6
76.000000      0.000000
max    2140.000000  5095.000000  5095.000000  18
62.000000   1064.000000

         GrLivArea  BsmtFullBath  BsmtHalfBath
FullBath     HalfBath  \
count  1459.000000   1457.000000   1457.000000
1459.000000  1459.000000
mean   1486.045922      0.434454      0.065202
1.570939     0.377656
std     485.566099      0.530648      0.252468
0.555190     0.503017
min     407.000000      0.000000      0.000000
0.000000     0.000000
25%    1117.500000      0.000000      0.000000
1.000000     0.000000
50%    1432.000000      0.000000      0.000000
2.000000     0.000000
75%    1721.000000      1.000000      0.000000
2.000000     1.000000
max    5095.000000      3.000000      2.000000
4.000000     2.000000

       BedroomAbvGr  KitchenAbvGr  TotRmsAbvGrd
Fireplaces  GarageYrBlt  \
count   1459.000000   1459.000000   1459.000000
1459.00000  1381.000000
mean       2.854010      1.042495      6.385195
0.58122  1977.721217
std        0.829788      0.208472      1.508895
0.64742    26.431175
min        0.000000      0.000000      3.000000
0.00000  1895.000000
25%        2.000000      1.000000      5.000000
0.00000  1959.000000
50%        3.000000      1.000000      6.000000
0.00000  1979.000000
75%        3.000000      1.000000      7.000000
1.00000  2002.000000
max        6.000000      2.000000     15.000000
4.00000  2207.000000

        GarageCars   GarageArea   WoodDeckSF  Op
enPorchSF  EnclosedPorch  \
count  1458.000000  1458.000000  1459.000000  14
```

```
count  1459.000000  1459.000000  1459.000000  14
59.000000    1459.000000
mean       1.766118   472.768861    93.174777
48.313914       24.243317
std        0.775945   217.048611   127.744882
68.883364       67.227765
min        0.000000     0.000000     0.000000
0.000000       0.000000
25%        1.000000   318.000000     0.000000
0.000000       0.000000
50%        2.000000   480.000000     0.000000
28.000000       0.000000
75%        2.000000   576.000000   168.000000
72.000000       0.000000
max        5.000000  1488.000000  1424.000000   7
42.000000    1012.000000

         3SsnPorch  ScreenPorch     PoolArea
MiscVal       MoSold  \
count  1459.000000  1459.000000  1459.000000   1
459.000000  1459.000000
mean       1.794380    17.064428     1.744345
58.167923     6.104181
std       20.207842    56.609763    30.491646
630.806978     2.722432
min        0.000000     0.000000     0.000000
0.000000     1.000000
25%        0.000000     0.000000     0.000000
0.000000     4.000000
50%        0.000000     0.000000     0.000000
0.000000     6.000000
75%        0.000000     0.000000     0.000000
0.000000     8.000000
max      360.000000   576.000000   800.000000  17
000.000000    12.000000

            YrSold
count  1459.000000
mean   2007.769705
std       1.301740
min    2006.000000
25%    2007.000000
50%    2008.000000
75%    2009.000000
max    2010.000000
```

- Some of our variables contain missing data. This is by and large due to the formatting of the data in its use of "NA" to show when a house doesn't contain a feature. Nonetheless it was decided to use it as missing initially to investigate if any variables contained a imbalances due to missing data. Additionally, some variables contain all records (1460) but have zero as the minimum. Based on our analysis, this is more than likely due to the fact the house doesn't have this feature. For example, if we look at TotBsmtSF, which is the total square feet of the basement, we see that it is missing no records but

the basement, we see that it is missing no records but has zero as a minimum. This more than likely means that the house does not have a basement.
- We notice on average, there is more unfinished basement space than finished basement space. This is similar to the training set
- There is on average substantionally less square feet space upstairs than downstairs in houses. This makes sense as some homes don't have a complete second floor, and most houses are not built as a perfect square but reduce size on the second floor for structural requirements. However the size of this difference could be explained if there was more one and one and a half story homes in the testing set.
- Some of our summary statistic variables are actually ordinal data so their summary statistics do not reveal much other than that they have no erroneous values (OverallQual, OverallCond, YearBuilt, YearRemodAdd, MasVnrArea, GarageYrBlt, MoSold, YrSold)

In this next section we will examine the missingness of our test data.

In [64]:

```
# Get numeric value to missing features
for i in range(len(housetest.columns)):
    j = housetest.columns[i]
    miss=((1459-housetest[str(j)].count())/1459)
*100
    print("The missingness of variable {}".forma
t(j))
    print("{0:.2f}%".format(miss))
```

```
The missingness of variable Id
0.00%
The missingness of variable MSSubClass
0.00%
The missingness of variable MSZoning
0.27%
The missingness of variable LotFrontage
15.56%
The missingness of variable LotArea
0.00%
The missingness of variable Street
0.00%
The missingness of variable Alley
92.67%
The missingness of variable LotShape
0.00%
The missingness of variable LandContour
0.00%
The missingness of variable Utilities
0.14%
The missingness of variable LotConfig
```

```
0.00%
The missingness of variable LandSlope
0.00%
The missingness of variable Neighborhood
0.00%
The missingness of variable Condition1
0.00%
The missingness of variable Condition2
0.00%
The missingness of variable BldgType
0.00%
The missingness of variable HouseStyle
0.00%
The missingness of variable OverallQual
0.00%
The missingness of variable OverallCond
0.00%
The missingness of variable YearBuilt
0.00%
The missingness of variable YearRemodAdd
0.00%
The missingness of variable RoofStyle
0.00%
The missingness of variable RoofMatl
0.00%
The missingness of variable Exterior1st
0.07%
The missingness of variable Exterior2nd
0.07%
The missingness of variable MasVnrType
61.27%
The missingness of variable MasVnrArea
1.03%
The missingness of variable ExterQual
0.00%
The missingness of variable ExterCond
0.00%
The missingness of variable Foundation
0.00%
The missingness of variable BsmtQual
3.02%
The missingness of variable BsmtCond
3.08%
The missingness of variable BsmtExposure
3.02%
The missingness of variable BsmtFinType1
2.88%
The missingness of variable BsmtFinSF1
0.07%
The missingness of variable BsmtFinType2
2.88%
The missingness of variable BsmtFinSF2
0.07%
The missingness of variable BsmtUnfSF
0.07%
The missingness of variable TotalBsmtSF
0.07%
The missingness of variable Heating
```

```
0.00%
The missingness of variable HeatingQC
0.00%
The missingness of variable CentralAir
0.00%
The missingness of variable Electrical
0.00%
The missingness of variable 1stFlrSF
0.00%
The missingness of variable 2ndFlrSF
0.00%
The missingness of variable LowQualFinSF
0.00%
The missingness of variable GrLivArea
0.00%
The missingness of variable BsmtFullBath
0.14%
The missingness of variable BsmtHalfBath
0.14%
The missingness of variable FullBath
0.00%
The missingness of variable HalfBath
0.00%
The missingness of variable BedroomAbvGr
0.00%
The missingness of variable KitchenAbvGr
0.00%
The missingness of variable KitchenQual
0.07%
The missingness of variable TotRmsAbvGrd
0.00%
The missingness of variable Functional
0.14%
The missingness of variable Fireplaces
0.00%
The missingness of variable FireplaceQu
50.03%
The missingness of variable GarageType
5.21%
The missingness of variable GarageYrBlt
5.35%
The missingness of variable GarageFinish
5.35%
The missingness of variable GarageCars
0.07%
The missingness of variable GarageArea
0.07%
The missingness of variable GarageQual
5.35%
The missingness of variable GarageCond
5.35%
The missingness of variable PavedDrive
0.00%
The missingness of variable WoodDeckSF
0.00%
The missingness of variable OpenPorchSF
0.00%
The missingness of variable EnclosedPorch
```

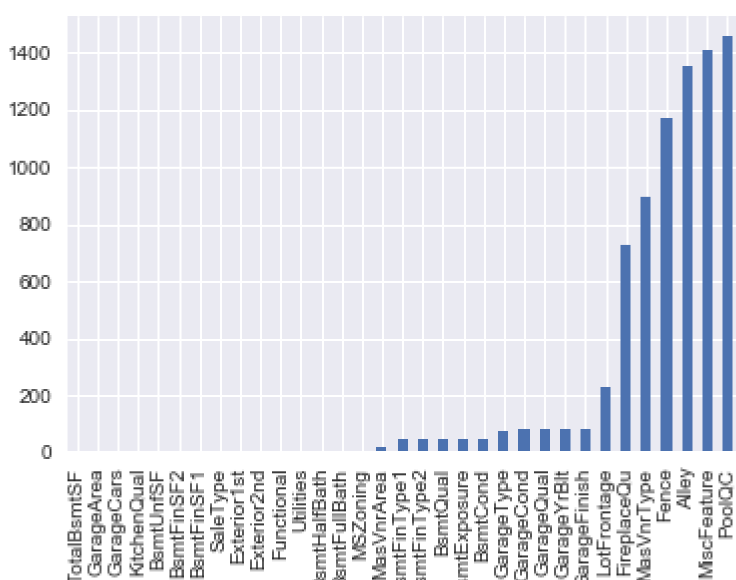
```
0.00%
The missingness of variable 3SsnPorch
0.00%
The missingness of variable ScreenPorch
0.00%
The missingness of variable PoolArea
0.00%
The missingness of variable PoolQC
99.79%
The missingness of variable Fence
80.12%
The missingness of variable MiscFeature
96.50%
The missingness of variable MiscVal
0.00%
The missingness of variable MoSold
0.00%
The missingness of variable YrSold
0.00%
The missingness of variable SaleType
0.07%
The missingness of variable SaleCondition
0.00%
```

In [65]:

```python
missing = housetest.isnull().sum()
missing = missing[missing > 0]
missing.sort_values(inplace=True)
missing.plot.bar()
```

Out[65]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe373
f60>
```



The first thing we notice is that we have a more variables with missing data in our test set than our training set. It is important to note, however, that all the variables with significant amounts of missing data are the same, and nearly to the exact same level, as missing data are the same, and nearly to the exact same level, as our training set.

Imputation

Just like our training set, we impute our missing data.

In [66]:

```
# Alley : data description says NA means "no all
ey access"
housetest.loc[:, "Alley"] = housetest.loc[:, "Al
ley"].fillna("None")
# BedroomAbvGr : NA most likely means 0
housetest.loc[:, "BedroomAbvGr"] = housetest.loc
[:, "BedroomAbvGr"].fillna(0)
# BsmtQual etc : data description says NA for ba
sement features is "no basement"
housetest.loc[:, "BsmtQual"] = housetest.loc[:,
"BsmtQual"].fillna("No")
housetest.loc[:, "BsmtCond"] = housetest.loc[:,
"BsmtCond"].fillna("No")
housetest.loc[:, "BsmtExposure"] = housetest.loc
[:, "BsmtExposure"].fillna("No")
housetest.loc[:, "BsmtFinType1"] = housetest.loc
[:, "BsmtFinType1"].fillna("No")
housetest.loc[:, "BsmtFinType2"] = housetest.loc
[:, "BsmtFinType2"].fillna("No")
housetest.loc[:, "BsmtFullBath"] = housetest.loc
[:, "BsmtFullBath"].fillna(0)
housetest.loc[:, "BsmtHalfBath"] = housetest.loc
[:, "BsmtHalfBath"].fillna(0)
housetest.loc[:, "BsmtUnfSF"] = housetest.loc[:,
"BsmtUnfSF"].fillna(0)
# CentralAir : NA most likely means No
housetest.loc[:, "CentralAir"] = housetest.loc
[:, "CentralAir"].fillna("N")
# Condition : NA most likely means Normal
housetest.loc[:, "Condition1"] = housetest.loc
[:, "Condition1"].fillna("Norm")
housetest.loc[:, "Condition2"] = housetest.loc
[:, "Condition2"].fillna("Norm")
# EnclosedPorch : NA most likely means no enclos
ed porch
housetest.loc[:, "EnclosedPorch"] = housetest.lo
c[:, "EnclosedPorch"].fillna(0)
# External stuff : NA most likely means average
housetest.loc[:, "ExterCond"] = housetest.loc[:,
"ExterCond"].fillna("TA")
housetest.loc[:, "ExterQual"] = housetest.loc[:,
"ExterQual"].fillna("TA")
# Fence : data description says NA means "no fen
ce"
housetest.loc[:, "Fence"] = housetest.loc[:, "Fe
nce"].fillna("No")
# FireplaceQu : data description says NA means
 "no fireplace"
```

```
housetest.loc[:, "FireplaceQu"] = housetest.loc
[:, "FireplaceQu"].fillna("No")
housetest.loc[:, "Fireplaces"] = housetest.loc
[:, "Fireplaces"].fillna(0)
# Functional : data description says NA means ty
pical
housetest.loc[:, "Functional"] = housetest.loc
[:, "Functional"].fillna("Typ")
# GarageType etc : data description says NA for
 garage features is "no garage"
housetest.loc[:, "GarageType"] = housetest.loc
[:, "GarageType"].fillna("No")
housetest.loc[:, "GarageFinish"] = housetest.loc
[:, "GarageFinish"].fillna("No")
housetest.loc[:, "GarageQual"] = housetest.loc
[:, "GarageQual"].fillna("No")
housetest.loc[:, "GarageCond"] = housetest.loc
[:, "GarageCond"].fillna("No")
housetest.loc[:, "GarageArea"] = housetest.loc
[:, "GarageArea"].fillna(0)
housetest.loc[:, "GarageCars"] = housetest.loc
[:, "GarageCars"].fillna(0)
# HalfBath : NA most likely means no half baths
 above grade
housetest.loc[:, "HalfBath"] = housetest.loc[:,
"HalfBath"].fillna(0)
# HeatingQC : NA most likely means typical
housetest.loc[:, "HeatingQC"] = housetest.loc[:,
"HeatingQC"].fillna("TA")
# KitchenAbvGr : NA most likely means 0
housetest.loc[:, "KitchenAbvGr"] = housetest.loc
[:, "KitchenAbvGr"].fillna(0)
# KitchenQual : NA most likely means typical
housetest.loc[:, "KitchenQual"] = housetest.loc
[:, "KitchenQual"].fillna("TA")
# LotFrontage : NA most likely means no lot fron
tage
housetest.loc[:, "LotFrontage"] = housetest.loc
[:, "LotFrontage"].fillna(0)
# LotShape : NA most likely means regular
housetest.loc[:, "LotShape"] = housetest.loc[:,
"LotShape"].fillna("Reg")
# MasVnrType : NA most likely means no veneer
#housetest.loc[:, "MasVnrType"] = housetest.loc
[:, "MasVnrType"].fillna("None")
housetest.loc[:, "MasVnrArea"] = housetest.loc
[:, "MasVnrArea"].fillna(0)
# MiscFeature : data description says NA means
 "no misc feature"
housetest.loc[:, "MiscFeature"] = housetest.loc
[:, "MiscFeature"].fillna("No")
housetest.loc[:, "MiscVal"] = housetest.loc[:,
"MiscVal"].fillna(0)
# OpenPorchSF : NA most likely means no open por
ch
housetest.loc[:, "OpenPorchSF"] = housetest.loc
[:, "OpenPorchSF"].fillna(0)
# PavedDrive : NA most likely means not paved
```

```
housetest.loc[:, "PavedDrive"] = housetest.loc
[:, "PavedDrive"].fillna("N")
# PoolQC : data description says NA means "no po
ol"
housetest.loc[:, "PoolQC"] = housetest.loc[:, "P
oolQC"].fillna("No")
housetest.loc[:, "PoolArea"] = housetest.loc[:,
"PoolArea"].fillna(0)
# SaleCondition : NA most likely means normal sa
le
housetest.loc[:, "SaleCondition"] = housetest.lo
c[:, "SaleCondition"].fillna("Normal")
# ScreenPorch : NA most likely means no screen p
orch
housetest.loc[:, "ScreenPorch"] = housetest.loc
[:, "ScreenPorch"].fillna(0)
# TotRmsAbvGrd : NA most likely means 0
housetest.loc[:, "TotRmsAbvGrd"] = housetest.loc
[:, "TotRmsAbvGrd"].fillna(0)
# Utilities : NA most likely means all public ut
ilities
housetest.loc[:, "Utilities"] = housetest.loc[:,
"Utilities"].fillna("AllPub")
# WoodDeckSF : NA most likely means no wood deck
housetest.loc[:, "WoodDeckSF"] = housetest.loc
[:, "WoodDeckSF"].fillna(0)
# Bsmt Square foot: missing probably means no ba
sement
housetest.loc[:, "BsmtFinSF1"] = housetest.loc
[:, "BsmtFinSF1"].fillna(0)
housetest.loc[:, "BsmtFinSF2"] = housetest.loc
[:, "BsmtFinSF2"].fillna(0)
housetest.loc[:, "TotalBsmtSF"] = housetest.loc
[:, "TotalBsmtSF"].fillna(0)
```

Recoding

We also need to carry out recoding on our test set. We need to ensure that the test set dose not contain any extra values not seen in the training or is missing any values not seen in the training.

In [67]:

```
# Graphing missing data
group = housetest.columns.to_series().groupby(ho
usetest.dtypes).groups # grouping columns by typ
e
groups={k.name: v for k, v in group.items()} #
 creating as dictionary

# Taking only the object type col names
objects=housetest[groups['object'].values]
#print(objects.head(5))
# Printing freqiency counts
for i in objects.columns:
```

```
        #print('{} \n' .format(objects[i]))
        print(objects[i].value_counts())
        print('\n')
```

```
20     543
60     276
50     143
120     95
30      70
70      68
160     65
80      60
90      57
190     31
85      28
180      7
75       7
45       6
40       2
150      1
Name: MSSubClass, dtype: int64


RL          1114
RM           242
FV            74
C (all)       15
RH            10
Name: MSZoning, dtype: int64


Pave    1453
Grvl       6
Name: Street, dtype: int64


None    1352
Grvl      70
Pave      37
Name: Alley, dtype: int64


Reg    934
IR1    484
IR2     35
IR3      6
Name: LotShape, dtype: int64


Lvl    1311
HLS      70
Bnk      54
Low      24
Name: LandContour, dtype: int64


AllPub    1459
Name: Utilities, dtype: int64
```

```
Inside     1081
Corner      248
CulDSac      82
FR2          38
FR3          10
Name: LotConfig, dtype: int64


Gtl    1396
Mod      60
Sev       3
Name: LandSlope, dtype: int64


NAmes      218
OldTown    126
CollgCr    117
Somerst     96
Edwards     94
NridgHt     89
Gilbert     86
Sawyer      77
SawyerW     66
Mitchel     65
NWAmes      58
IDOTRR      56
Crawfor     52
BrkSide     50
Timber      34
NoRidge     30
StoneBr     26
SWISU       23
MeadowV     20
ClearCr     16
NPkVill     14
BrDale      14
Veenker     13
Blmngtn     11
Blueste      8
Name: Neighborhood, dtype: int64


Norm      1251
Feedr       83
Artery      44
RRAn        24
PosN        20
RRAe        17
PosA        12
RRNn         4
RRNe         4
Name: Condition1, dtype: int64


Norm      1444
Feedr        7
```

```
PosA         3
Artery       3
PosN         2
Name: Condition2, dtype: int64


1Fam      1205
TwnhsE     113
Duplex      57
Twnhs       53
2fmCon      31
Name: BldgType, dtype: int64


1Story    745
2Story    427
1.5Fin    160
SLvl       63
SFoyer     46
2.5Unf     13
1.5Unf      5
Name: HouseStyle, dtype: int64


Gable      1169
Hip         265
Gambrel      11
Flat          7
Mansard       4
Shed          3
Name: RoofStyle, dtype: int64


CompShg    1442
Tar&Grv      12
WdShake       4
WdShngl       1
Name: RoofMatl, dtype: int64


VinylSd    510
MetalSd    230
HdBoard    220
Wd Sdng    205
Plywood    113
CemntBd     65
BrkFace     37
WdShing     30
AsbShng     24
Stucco      18
BrkComm      4
CBlock       1
AsphShn      1
Name: Exterior1st, dtype: int64


VinylSd    510
```

```
MetalSd    233
HdBoard    199
Wd Sdng    194
Plywood    128
CmentBd     66
Wd Shng     43
BrkFace     22
Stucco      21
AsbShng     18
Brk Cmn     15
ImStucc      5
CBlock       2
AsphShn      1
Stone        1
Name: Exterior2nd, dtype: int64


BrkFace    434
Stone      121
BrkCmn      10
Name: MasVnrType, dtype: int64


TA    892
Gd    491
Ex     55
Fa     21
Name: ExterQual, dtype: int64


TA    1256
Gd     153
Fa      39
Ex       9
Po       2
Name: ExterCond, dtype: int64


PConc     661
CBlock    601
BrkTil    165
Slab       25
Stone       5
Wood        2
Name: Foundation, dtype: int64


TA    634
Gd    591
Ex    137
Fa     53
No     44
Name: BsmtQual, dtype: int64


TA    1295
Fa      59
Gd      57
```

```
No      45
Po       3
Name: BsmtCond, dtype: int64


No    995
Av    197
Gd    142
Mn    125
Name: BsmtExposure, dtype: int64


GLQ    431
Unf    421
ALQ    209
Rec    155
BLQ    121
LwQ     80
No      42
Name: BsmtFinType1, dtype: int64


Unf    1237
Rec      51
No       42
LwQ      41
BLQ      35
ALQ      33
GLQ      20
Name: BsmtFinType2, dtype: int64


GasA    1446
GasW       9
Grav       2
Wall       2
Name: Heating, dtype: int64


Ex    752
TA    429
Gd    233
Fa     43
Po      2
Name: HeatingQC, dtype: int64


Y    1358
N     101
Name: CentralAir, dtype: int64


SBrkr    1337
FuseA      94
FuseF      23
FuseP       5
Name: Electrical, dtype: int64
```

```
TA    758
Gd    565
Ex    105
Fa     31
Name: KitchenQual, dtype: int64


Typ     1359
Min2      36
Min1      34
Mod       20
Maj1       5
Maj2       4
Sev        1
Name: Functional, dtype: int64


No    730
Gd    364
TA    279
Fa     41
Po     26
Ex     19
Name: FireplaceQu, dtype: int64


Attchd     853
Detchd     392
BuiltIn     98
No          76
2Types      17
Basment     17
CarPort      6
Name: GarageType, dtype: int64


Unf    625
RFn    389
Fin    367
No      78
Name: GarageFinish, dtype: int64


TA    1293
No      78
Fa      76
Gd      10
Po       2
Name: GarageQual, dtype: int64


TA    1328
No      78
Fa      39
Po       7
Gd       6
Ex       1
```

```
Ex       1
Name: GarageCond, dtype: int64


Y    1301
N     126
P      32
Name: PavedDrive, dtype: int64


No    1456
Ex       2
Gd       1
Name: PoolQC, dtype: int64


No       1169
MnPrv     172
GdPrv      59
GdWo       58
MnWw        1
Name: Fence, dtype: int64


No      1408
Shed      46
Gar2       3
Othr       2
Name: MiscFeature, dtype: int64


WD       1258
New       117
COD        44
ConLD      17
CWD         8
ConLI       4
Oth         4
ConLw       3
Con         3
Name: SaleType, dtype: int64


Normal     1204
Partial     120
Abnorml      89
Family       26
Alloca       12
AdjLand       8
Name: SaleCondition, dtype: int64
```

In [68]:

```
# reg or irreg
housetest['LotShape']=housetest['LotShape'].repl
ace(['IR1','IR2','IR3'],'IRReg')
```

```
#print(housetest['LotShape'].value_counts())

# flat or not flat
housetest['LandContour']=housetest['LandContour'
].replace(['Bnk','HLS','Low'],'NotFlat')
#print(housetest['LandContour'].value_counts())

# combined frontage
housetest['LotConfig']=housetest['LotConfig'].re
place(['FR2','FR3'],'Frontage')
#print(housetest['LotConfig'].value_counts())

# combined rail and pos
housetest['Condition1']=housetest['Condition1'].
replace(['RRNn','RRAn','RRNe','RRAe'],'Rail')
housetest['Condition1']=housetest['Condition1'].
replace(['PosN','PosA'],'Pos')
#print(housetest['Condition1'].value_counts())

# combined rail and pos
housetest['Condition2']=housetest['Condition2'].
replace(['RRNn','RRAn','RRNe','RRAe'],'Rail')
housetest['Condition2']=housetest['Condition2'].
replace(['PosN','PosA'],'Pos')
#print(housetest['Condition2'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetest['ExterQual']=housetest['ExterQual'].re
place(['Ex','Gd'],'Above Average')
housetest['ExterQual']=housetest['ExterQual'].re
place(['Fa','Po'],'Below Average')
#print(housetest['ExterQual'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetest['ExterCond']=housetest['ExterCond'].re
place(['Ex','Gd'],'Above Average')
housetest['ExterCond']=housetest['ExterCond'].re
place(['Fa','Po'],'Below Average')
#print(housetest['ExterCond'].value_counts())

housetest['HouseStyle']=housetest['HouseStyle'].
replace(['1Story','1.5Unf','1.5Fin'],'1to2Story'
)
housetest['HouseStyle']=housetest['HouseStyle'].
replace(['2Story','2.5Unf','2.5Fin'],'2+Story')
#print(housetest['HouseStyle'].value_counts())

housetest['RoofStyle']=housetest['RoofStyle'].re
place(['Flat','Gambrel','Mansard','Shed'],'Othe
r')
#print(housetest['RoofStyle'].value_counts())

housetest['RoofMatl']=housetest['RoofMatl'].repl
ace(['ClyTile','Membran','Metal','Roll','Tar&Gr
v','WdShake','WdShngl'],'Other')
#print(housetest['RoofMatl'].value_counts())
```

```
# Recoding to have less options and grouping sim
ilar
housetest['SaleType']=housetest['SaleType'].repl
ace(['WD','CWD','VWD'],'Warrenty Deed')
housetest['SaleType']=housetest['SaleType'].repl
ace(['Con','ConLw','ConLI','ConLD'],'Contract')
#print(housetest['SaleType'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetest['GarageCond']=housetest['GarageCond'].
replace(['Ex','Gd'],'Above Average')
housetest['GarageCond']=housetest['GarageCond'].
replace(['Fa','Po'],'Below Average')
#print(housetest['GarageCond'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetest['GarageQual']=housetest['GarageQual'].
replace(['Ex','Gd'],'Above Average')
housetest['GarageQual']=housetest['GarageQual'].
replace(['Fa','Po'],'Below Average')
#print(housetest['GarageQual'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetest['Functional']=housetest['Functional'].
replace(['Min1','Min2'],'Min')
housetest['Functional']=housetest['Functional'].
replace(['Maj1','Maj2','Sev','Sal'],'Maj')
#print(housetest['Functional'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetest['KitchenQual']=housetest['KitchenQual'
].replace(['Ex','Gd'],'Above Average')
housetest['KitchenQual']=housetest['KitchenQual'
].replace(['Fa','Po'],'Below Average')
#print(housetest['KitchenQual'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetest['HeatingQC']=housetest['HeatingQC'].re
place(['Ex','Gd'],'Above Average')
housetest['HeatingQC']=housetest['HeatingQC'].re
place(['Fa','Po'],'Below Average')
#print(housetest['HeatingQC'].value_counts())

# Merging Gas
housetest['Heating']=housetest['Heating'].replac
e(['GasA','GasW'],'Gas')
#print(housetest['Heating'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetest['BsmtFinType2']=housetest['BsmtFinType
2'].replace(['ALQ','Rec'],'Average')
```

```
housetest['BsmtFinType2']=housetest['BsmtFinType
2'].replace(['BLQ','LwQ'],'Below Average')
#print(housetest['BsmtFinType2'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetest['BsmtFinType1']=housetest['BsmtFinType
1'].replace(['ALQ','Rec'],'Average')
housetest['BsmtFinType1']=housetest['BsmtFinType
1'].replace(['BLQ','LwQ'],'Below Average')
#print(housetest['BsmtFinType1'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetest['BsmtCond']=housetest['BsmtCond'].repl
ace(['Ex','Gd'],'Above Average')
housetest['BsmtCond']=housetest['BsmtCond'].repl
ace(['Fa','Po'],'Below Average')
#print(housetest['BsmtCond'].value_counts())

# Recoding to have less options and grouping sim
ilar
housetest['BsmtQual']=housetest['BsmtQual'].repl
ace(['Ex','Gd'],'Above Average')
housetest['BsmtQual']=housetest['BsmtQual'].repl
ace(['Fa','Po'],'Below Average')
#print(housetest['BsmtQual'].value_counts())

# Foundation: One of the more standard options o
r other
housetest['Foundation']=housetest['Foundation'].
replace(['BrkTil','Slab','Stone','Wood'],'Other'
)
#print(housetest['Foundation'].value_counts())
group = housetest.columns.to_series().groupby(ho
usetest.dtypes).groups # grouping columns by typ
e
groups={k.name: v for k, v in group.items()} #
 creating as dictionary

# Taking only the object type col names
objects=housetest[groups['object'].values]
for i in objects.columns:
        #print('{} \n' .format(objects[i]))
        print(objects[i].value_counts())
        print('\n')
```

```
20     543
60     276
50     143
120     95
30      70
70      68
160     65
80      60
90      57
190     31
85      28
```

```
180      7
75       7
45       6
40       2
150      1
Name: MSSubClass, dtype: int64


RL          1114
RM           242
FV            74
C (all)       15
RH            10
Name: MSZoning, dtype: int64


Pave    1453
Grvl       6
Name: Street, dtype: int64


None    1352
Grvl      70
Pave      37
Name: Alley, dtype: int64


Reg      934
IRReg    525
Name: LotShape, dtype: int64


Lvl         1311
NotFlat      148
Name: LandContour, dtype: int64


AllPub    1459
Name: Utilities, dtype: int64


Inside      1081
Corner       248
CulDSac       82
Frontage      48
Name: LotConfig, dtype: int64


Gtl    1396
Mod      60
Sev       3
Name: LandSlope, dtype: int64


NAmes      218
OldTown    126
CollgCr    117
```

```
Somerst      96
Edwards      94
NridgHt      89
Gilbert      86
Sawyer       77
SawyerW      66
Mitchel      65
NWAmes       58
IDOTRR       56
Crawfor      52
BrkSide      50
Timber       34
NoRidge      30
StoneBr      26
SWISU        23
MeadowV      20
ClearCr      16
NPkVill      14
BrDale       14
Veenker      13
Blmngtn      11
Blueste       8
Name: Neighborhood, dtype: int64


Norm      1251
Feedr       83
Rail        49
Artery      44
Pos         32
Name: Condition1, dtype: int64


Norm      1444
Feedr        7
Pos          5
Artery       3
Name: Condition2, dtype: int64


1Fam      1205
TwnhsE     113
Duplex      57
Twnhs       53
2fmCon      31
Name: BldgType, dtype: int64


1to2Story    910
2+Story      440
SLvl          63
SFoyer        46
Name: HouseStyle, dtype: int64


Gable    1169
Hip       265
Other      25
```

```
Name: RoofStyle, dtype: int64


CompShg    1442
Other        17
Name: RoofMatl, dtype: int64


VinylSd    510
MetalSd    230
HdBoard    220
Wd Sdng    205
Plywood    113
CemntBd     65
BrkFace     37
WdShing     30
AsbShng     24
Stucco      18
BrkComm      4
CBlock       1
AsphShn      1
Name: Exterior1st, dtype: int64


VinylSd    510
MetalSd    233
HdBoard    199
Wd Sdng    194
Plywood    128
CmentBd     66
Wd Shng     43
BrkFace     22
Stucco      21
AsbShng     18
Brk Cmn     15
ImStucc      5
CBlock       2
AsphShn      1
Stone        1
Name: Exterior2nd, dtype: int64


BrkFace    434
Stone      121
BrkCmn      10
Name: MasVnrType, dtype: int64


TA                892
Above Average    546
Below Average     21
Name: ExterQual, dtype: int64


TA                1256
Above Average      162
Below Average       41
Name: ExterCond, dtype: int64
```

```
PConc      661
CBlock     601
Other      197
Name: Foundation, dtype: int64


Above Average    728
TA               634
Below Average     53
No                44
Name: BsmtQual, dtype: int64


TA               1295
Below Average      62
Above Average      57
No                 45
Name: BsmtCond, dtype: int64


No    995
Av    197
Gd    142
Mn    125
Name: BsmtExposure, dtype: int64


GLQ              431
Unf              421
Average          364
Below Average    201
No                42
Name: BsmtFinType1, dtype: int64


Unf              1237
Average            84
Below Average      76
No                 42
GLQ                20
Name: BsmtFinType2, dtype: int64


Gas     1455
Grav       2
Wall       2
Name: Heating, dtype: int64


Above Average    985
TA               429
Below Average     45
Name: HeatingQC, dtype: int64
```

```
Y     1358
N     101
Name: CentralAir, dtype: int64


SBrkr    1337
FuseA      94
FuseF      23
FuseP       5
Name: Electrical, dtype: int64


TA               758
Above Average    670
Below Average     31
Name: KitchenQual, dtype: int64


Typ    1359
Min      70
Mod      20
Maj      10
Name: Functional, dtype: int64


No    730
Gd    364
TA    279
Fa     41
Po     26
Ex     19
Name: FireplaceQu, dtype: int64


Attchd     853
Detchd     392
BuiltIn     98
No          76
2Types      17
Basment     17
CarPort      6
Name: GarageType, dtype: int64


Unf    625
RFn    389
Fin    367
No      78
Name: GarageFinish, dtype: int64


TA               1293
No                 78
Below Average      78
```