NB Week 1 EDA

May 23, 2019

1 House Prices: Advanced Regression Techniques

1.1 Introduction:

This project and the data can be found in https://www.kaggle.com/c/house-prices-advanced-regression-techniques.

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

With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, this competition challenges you to predict the final price of each home.

We will start first with EDA to check the dataset, available rows, the distribution of the sale price (target).

1.2 EDA

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import sys
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
%matplotlib inline
```

In [2]:
```
# Read the data
data = pd.read_csv('../train.csv', index_col=0)
data.head()
```

Out[2]:

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

```
   LandContour Utilities LotConfig    ...     PoolArea PoolQC Fence  \
Id                                    ...
1          Lvl    AllPub    Inside    ...            0    NaN   NaN
2          Lvl    AllPub       FR2    ...            0    NaN   NaN
3          Lvl    AllPub    Inside    ...            0    NaN   NaN
4          Lvl    AllPub    Corner    ...            0    NaN   NaN
5          Lvl    AllPub       FR2    ...            0    NaN   NaN

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

[5 rows x 80 columns]
```

In [3]:
```
# Read the description of the file
with open('../data_description.txt',  'r') as fi:
    print(fi.read())
```

```
MSSubClass: Identifies the type of dwelling involved in the sale.

        20        1-STORY 1946 & NEWER ALL STYLES
        30        1-STORY 1945 & OLDER
        40        1-STORY W/FINISHED ATTIC ALL AGES
        45        1-1/2 STORY - UNFINISHED ALL AGES
        50        1-1/2 STORY FINISHED ALL AGES
        60        2-STORY 1946 & NEWER
        70        2-STORY 1945 & OLDER
        75        2-1/2 STORY ALL AGES
        80        SPLIT OR MULTI-LEVEL
        85        SPLIT FOYER
        90        DUPLEX - ALL STYLES AND AGES
       120        1-STORY PUD (Planned Unit Development) - 1946 & NEWER
       150        1-1/2 STORY PUD - ALL AGES
       160        2-STORY PUD - 1946 & NEWER
       180        PUD - MULTILEVEL - INCL SPLIT LEV/FOYER
       190        2 FAMILY CONVERSION - ALL STYLES AND AGES

MSZoning: Identifies the general zoning classification of the sale.

       A        Agriculture
       C        Commercial
       FV        Floating Village Residential
       I        Industrial
       RH        Residential High Density
```

```
       RL        Residential Low Density
       RP        Residential Low Density Park
       RM        Residential Medium Density

LotFrontage: Linear feet of street connected to property

LotArea: Lot size in square feet

Street: Type of road access to property

       Grvl        Gravel
       Pave        Paved

Alley: Type of alley access to property

       Grvl        Gravel
       Pave        Paved
       NA         No alley access

LotShape: General shape of property

       Reg        Regular
       IR1        Slightly irregular
       IR2        Moderately Irregular
       IR3        Irregular

LandContour: Flatness of the property

       Lvl        Near Flat/Level
       Bnk        Banked - Quick and significant rise from street grade to building
       HLS        Hillside - Significant slope from side to side
       Low        Depression

Utilities: Type of utilities available

       AllPub        All public Utilities (E,G,W,& S)
       NoSewr        Electricity, Gas, and Water (Septic Tank)
       NoSeWa        Electricity and Gas Only
       ELO        Electricity only

LotConfig: Lot configuration

       Inside        Inside lot
       Corner        Corner lot
       CulDSac        Cul-de-sac
       FR2        Frontage on 2 sides of property
       FR3        Frontage on 3 sides of property
```

LandSlope: Slope of property

Gtl	Gentle slope
Mod	Moderate Slope
Sev	Severe Slope

Neighborhood: Physical locations within Ames city limits

Blmngtn	Bloomington Heights
Blueste	Bluestem
BrDale	Briardale
BrkSide	Brookside
ClearCr	Clear Creek
CollgCr	College Creek
Crawfor	Crawford
Edwards	Edwards
Gilbert	Gilbert
IDOTRR	Iowa DOT and Rail Road
MeadowV	Meadow Village
Mitchel	Mitchell
Names	North Ames
NoRidge	Northridge
NPkVill	Northpark Villa
NridgHt	Northridge Heights
NWAmes	Northwest Ames
OldTown	Old Town
SWISU	South & West of Iowa State University
Sawyer	Sawyer
SawyerW	Sawyer West
Somerst	Somerset
StoneBr	Stone Brook
Timber	Timberland
Veenker	Veenker

Condition1: Proximity to various conditions

Artery	Adjacent to arterial street
Feedr	Adjacent to feeder street
Norm	Normal
RRNn	Within 200' of North-South Railroad
RRAn	Adjacent to North-South Railroad
PosN	Near positive off-site feature--park, greenbelt, etc.
PosA	Adjacent to postive off-site feature
RRNe	Within 200' of East-West Railroad
RRAe	Adjacent to East-West Railroad

Condition2: Proximity to various conditions (if more than one is present)

```
       Artery        Adjacent to arterial street
       Feedr         Adjacent to feeder street
       Norm         Normal
       RRNn         Within 200' of North-South Railroad
       RRAn         Adjacent to North-South Railroad
       PosN         Near positive off-site feature--park, greenbelt, etc.
       PosA         Adjacent to postive off-site feature
       RRNe         Within 200' of East-West Railroad
       RRAe         Adjacent to East-West Railroad

BldgType: Type of dwelling

       1Fam         Single-family Detached
       2FmCon         Two-family Conversion; originally built as one-family dwelling
       Duplx         Duplex
       TwnhsE         Townhouse End Unit
       TwnhsI         Townhouse Inside Unit

HouseStyle: Style of dwelling

       1Story         One story
       1.5Fin         One and one-half story: 2nd level finished
       1.5Unf         One and one-half story: 2nd level unfinished
       2Story         Two story
       2.5Fin         Two and one-half story: 2nd level finished
       2.5Unf         Two and one-half story: 2nd level unfinished
       SFoyer         Split Foyer
       SLvl         Split Level

OverallQual: Rates the overall material and finish of the house

       10         Very Excellent
       9         Excellent
       8         Very Good
       7         Good
       6         Above Average
       5         Average
       4         Below Average
       3         Fair
       2         Poor
       1         Very Poor

OverallCond: Rates the overall condition of the house

       10         Very Excellent
       9         Excellent
       8         Very Good
       7         Good
```

```
       6        Above Average
       5        Average
       4        Below Average
       3        Fair
       2        Poor
       1        Very Poor

YearBuilt: Original construction date

YearRemodAdd: Remodel date (same as construction date if no remodeling or additions)

RoofStyle: Type of roof

       Flat        Flat
       Gable        Gable
       Gambrel        Gabrel (Barn)
       Hip        Hip
       Mansard        Mansard
       Shed        Shed

RoofMatl: Roof material

       ClyTile        Clay or Tile
       CompShg        Standard (Composite) Shingle
       Membran        Membrane
       Metal        Metal
       Roll        Roll
       Tar&Grv        Gravel & Tar
       WdShake        Wood Shakes
       WdShngl        Wood Shingles

Exterior1st: Exterior covering on house

       AsbShng        Asbestos Shingles
       AsphShn        Asphalt Shingles
       BrkComm        Brick Common
       BrkFace        Brick Face
       CBlock        Cinder Block
       CemntBd        Cement Board
       HdBoard        Hard Board
       ImStucc        Imitation Stucco
       MetalSd        Metal Siding
       Other        Other
       Plywood        Plywood
       PreCast        PreCast
       Stone        Stone
       Stucco        Stucco
       VinylSd        Vinyl Siding
```

```
       Wd Sdng        Wood Siding
       WdShing        Wood Shingles

Exterior2nd: Exterior covering on house (if more than one material)

       AsbShng        Asbestos Shingles
       AsphShn        Asphalt Shingles
       BrkComm        Brick Common
       BrkFace        Brick Face
       CBlock        Cinder Block
       CemntBd        Cement Board
       HdBoard        Hard Board
       ImStucc        Imitation Stucco
       MetalSd        Metal Siding
       Other        Other
       Plywood        Plywood
       PreCast        PreCast
       Stone        Stone
       Stucco        Stucco
       VinylSd        Vinyl Siding
       Wd Sdng        Wood Siding
       WdShing        Wood Shingles

MasVnrType: Masonry veneer type

       BrkCmn        Brick Common
       BrkFace        Brick Face
       CBlock        Cinder Block
       None        None
       Stone        Stone

MasVnrArea: Masonry veneer area in square feet

ExterQual: Evaluates the quality of the material on the exterior

       Ex        Excellent
       Gd        Good
       TA        Average/Typical
       Fa        Fair
       Po        Poor

ExterCond: Evaluates the present condition of the material on the exterior

       Ex        Excellent
       Gd        Good
       TA        Average/Typical
       Fa        Fair
       Po        Poor
```

Foundation: Type of foundation

BrkTil	Brick & Tile
CBlock	Cinder Block
PConc	Poured Contrete
Slab	Slab
Stone	Stone
Wood	Wood

BsmtQual: Evaluates the height of the basement

Ex	Excellent (100+ inches)
Gd	Good (90-99 inches)
TA	Typical (80-89 inches)
Fa	Fair (70-79 inches)
Po	Poor (<70 inches
NA	No Basement

BsmtCond: Evaluates the general condition of the basement

Ex	Excellent
Gd	Good
TA	Typical - slight dampness allowed
Fa	Fair - dampness or some cracking or settling
Po	Poor - Severe cracking, settling, or wetness
NA	No Basement

BsmtExposure: Refers to walkout or garden level walls

Gd	Good Exposure
Av	Average Exposure (split levels or foyers typically score average or above)
Mn	Mimimum Exposure
No	No Exposure
NA	No Basement

BsmtFinType1: Rating of basement finished area

GLQ	Good Living Quarters
ALQ	Average Living Quarters
BLQ	Below Average Living Quarters
Rec	Average Rec Room
LwQ	Low Quality
Unf	Unfinshed
NA	No Basement

BsmtFinSF1: Type 1 finished square feet

BsmtFinType2: Rating of basement finished area (if multiple types)

GLQ	Good Living Quarters
ALQ	Average Living Quarters
BLQ	Below Average Living Quarters
Rec	Average Rec Room
LwQ	Low Quality
Unf	Unfinshed
NA	No Basement

BsmtFinSF2: Type 2 finished square feet

BsmtUnfSF: Unfinished square feet of basement area

TotalBsmtSF: Total square feet of basement area

Heating: Type of heating

Floor	Floor Furnace
GasA	Gas forced warm air furnace
GasW	Gas hot water or steam heat
Grav	Gravity furnace
OthW	Hot water or steam heat other than gas
Wall	Wall furnace

HeatingQC: Heating quality and condition

Ex	Excellent
Gd	Good
TA	Average/Typical
Fa	Fair
Po	Poor

CentralAir: Central air conditioning

N	No
Y	Yes

Electrical: Electrical system

SBrkr	Standard Circuit Breakers & Romex
FuseA	Fuse Box over 60 AMP and all Romex wiring (Average)
FuseF	60 AMP Fuse Box and mostly Romex wiring (Fair)
FuseP	60 AMP Fuse Box and mostly knob & tube wiring (poor)
Mix	Mixed

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

FullBath: Full bathrooms above grade

HalfBath: Half baths above grade

Bedroom: Bedrooms above grade (does NOT include basement bedrooms)

Kitchen: Kitchens above grade

KitchenQual: Kitchen quality

Ex	Excellent
Gd	Good
TA	Typical/Average
Fa	Fair
Po	Poor

TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)

Functional: Home functionality (Assume typical unless deductions are warranted)

Typ	Typical Functionality
Min1	Minor Deductions 1
Min2	Minor Deductions 2
Mod	Moderate Deductions
Maj1	Major Deductions 1
Maj2	Major Deductions 2
Sev	Severely Damaged
Sal	Salvage only

Fireplaces: Number of fireplaces

FireplaceQu: Fireplace quality

Ex	Excellent - Exceptional Masonry Fireplace
Gd	Good - Masonry Fireplace in main level
TA	Average - Prefabricated Fireplace in main living area or Masonry Fireplace in
Fa	Fair - Prefabricated Fireplace in basement
Po	Poor - Ben Franklin Stove

```
       NA        No Fireplace

GarageType: Garage location

       2Types        More than one type of garage
       Attchd        Attached to home
       Basment        Basement Garage
       BuiltIn        Built-In (Garage part of house - typically has room above garage)
       CarPort        Car Port
       Detchd        Detached from home
       NA        No Garage

GarageYrBlt: Year garage was built

GarageFinish: Interior finish of the garage

       Fin        Finished
       RFn        Rough Finished
       Unf        Unfinished
       NA        No Garage

GarageCars: Size of garage in car capacity

GarageArea: Size of garage in square feet

GarageQual: Garage quality

       Ex        Excellent
       Gd        Good
       TA        Typical/Average
       Fa        Fair
       Po        Poor
       NA        No Garage

GarageCond: Garage condition

       Ex        Excellent
       Gd        Good
       TA        Typical/Average
       Fa        Fair
       Po        Poor
       NA        No Garage

PavedDrive: Paved driveway

       Y        Paved
       P        Partial Pavement
       N        Dirt/Gravel
```

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

Ex	Excellent
Gd	Good
TA	Average/Typical
Fa	Fair
NA	No Pool

Fence: Fence quality

GdPrv	Good Privacy
MnPrv	Minimum Privacy
GdWo	Good Wood
MnWw	Minimum Wood/Wire
NA	No Fence

MiscFeature: Miscellaneous feature not covered in other categories

Elev	Elevator
Gar2	2nd Garage (if not described in garage section)
Othr	Other
Shed	Shed (over 100 SF)
TenC	Tennis Court
NA	None

MiscVal: $Value of miscellaneous feature

MoSold: Month Sold (MM)

YrSold: Year Sold (YYYY)

SaleType: Type of sale

WD	Warranty Deed - Conventional
CWD	Warranty Deed - Cash

```
       VWD        Warranty Deed - VA Loan
       New        Home just constructed and sold
       COD        Court Officer Deed/Estate
       Con        Contract 15% Down payment regular terms
       ConLw       Contract Low Down payment and low interest
       ConLI       Contract Low Interest
       ConLD       Contract Low Down
       Oth        Other

SaleCondition: Condition of sale

       Normal        Normal Sale
       Abnorml        Abnormal Sale -  trade, foreclosure, short sale
       AdjLand        Adjoining Land Purchase
       Alloca        Allocation - two linked properties with separate deeds, typically condo w
       Family        Sale between family members
       Partial        Home was not completed when last assessed (associated with New Homes)
```

```
In [4]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 1 to 1460
Data columns (total 80 columns):
MSSubClass       1460 non-null int64
MSZoning         1460 non-null object
LotFrontage      1201 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null object
Alley            91 non-null object
LotShape         1460 non-null object
LandContour      1460 non-null object
Utilities        1460 non-null object
LotConfig        1460 non-null object
LandSlope        1460 non-null object
Neighborhood     1460 non-null object
Condition1       1460 non-null object
Condition2       1460 non-null object
BldgType         1460 non-null object
HouseStyle       1460 non-null object
OverallQual      1460 non-null int64
OverallCond      1460 non-null int64
YearBuilt        1460 non-null int64
YearRemodAdd     1460 non-null int64
RoofStyle        1460 non-null object
RoofMatl         1460 non-null object
Exterior1st      1460 non-null object
```

```
Exterior2nd      1460 non-null object
MasVnrType       1452 non-null object
MasVnrArea       1452 non-null float64
ExterQual        1460 non-null object
ExterCond        1460 non-null object
Foundation       1460 non-null object
BsmtQual         1423 non-null object
BsmtCond         1423 non-null object
BsmtExposure     1422 non-null object
BsmtFinType1     1423 non-null object
BsmtFinSF1       1460 non-null int64
BsmtFinType2     1422 non-null object
BsmtFinSF2       1460 non-null int64
BsmtUnfSF        1460 non-null int64
TotalBsmtSF      1460 non-null int64
Heating          1460 non-null object
HeatingQC        1460 non-null object
CentralAir       1460 non-null object
Electrical       1459 non-null object
1stFlrSF         1460 non-null int64
2ndFlrSF         1460 non-null int64
LowQualFinSF     1460 non-null int64
GrLivArea        1460 non-null int64
BsmtFullBath     1460 non-null int64
BsmtHalfBath     1460 non-null int64
FullBath         1460 non-null int64
HalfBath         1460 non-null int64
BedroomAbvGr     1460 non-null int64
KitchenAbvGr     1460 non-null int64
KitchenQual      1460 non-null object
TotRmsAbvGrd     1460 non-null int64
Functional       1460 non-null object
Fireplaces       1460 non-null int64
FireplaceQu      770 non-null object
GarageType       1379 non-null object
GarageYrBlt      1379 non-null float64
GarageFinish     1379 non-null object
GarageCars       1460 non-null int64
GarageArea       1460 non-null int64
GarageQual       1379 non-null object
GarageCond       1379 non-null object
PavedDrive       1460 non-null object
WoodDeckSF       1460 non-null int64
OpenPorchSF      1460 non-null int64
EnclosedPorch    1460 non-null int64
3SsnPorch        1460 non-null int64
ScreenPorch      1460 non-null int64
PoolArea         1460 non-null int64
```

```
PoolQC           7 non-null object
Fence            281 non-null object
MiscFeature      54 non-null object
MiscVal          1460 non-null int64
MoSold           1460 non-null int64
YrSold           1460 non-null int64
SaleType         1460 non-null object
SaleCondition    1460 non-null object
SalePrice        1460 non-null int64
dtypes: float64(3), int64(34), object(43)
memory usage: 923.9+ KB
```

1.2.1 Notes on the feature columns:

The following columns have NA, but NA here indicate something: * Alley column => NA means "No alley access". * BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, BsmtFinSF1 columns => NA means "No Basement" * FireplaceQu column => NA means "No fireplace" * GarageType, GarageCond, GarageFinish columns => NA means "No Garage" * PoolQC column => No Pool * Fence column => No fence * MiscFeature column => None

So be carful with dropping NA values. Next cell, I will fillna as if I didn't pandas will ignore NA values.

```
In [5]: data.Alley = data.Alley.fillna(value = 'NoAlley')
        data.BsmtCond = data.BsmtCond.fillna(value = 'NoBsmt')
        data.BsmtQual = data.BsmtQual.fillna(value = 'NoBsmt')
        data.BsmtExposure = data.BsmtExposure.fillna(value= 'NoBsmt')
        data.BsmtFinType1 = data.BsmtFinType1.fillna(value= 'NoBsmt')
        data.BsmtFinType2 = data.BsmtFinType2.fillna(value= 'NoBsmt')
        data.LotFrontage = data.LotFrontage.fillna(value = 0)
        data.FireplaceQu = data.FireplaceQu.fillna(value = 'Nofireplace')
        data.GarageType = data.GarageType.fillna(value = 'NoGarage')
        data.GarageCond = data.GarageCond.fillna(value = 'NoGarage')
        data.GarageFinish = data.GarageFinish.fillna(value = 'NoGarage')
        data.GarageYrBlt = data.GarageYrBlt.fillna(value = 0)
        data.GarageQual = data.GarageQual.fillna(value = 'NoGarage')

        data.PoolQC = data.PoolQC.fillna(value = 'NoPool')
        data.Fence = data.Fence.fillna(value = 'NoFence')
        data.MiscFeature = data.MiscFeature.fillna(value = 'NoMisc')
        data.MasVnrType = data.MasVnrType.fillna(value = 'noMas')
        data.MasVnrArea = data.MasVnrArea.fillna(value = 'noMas')

        data.info()
```

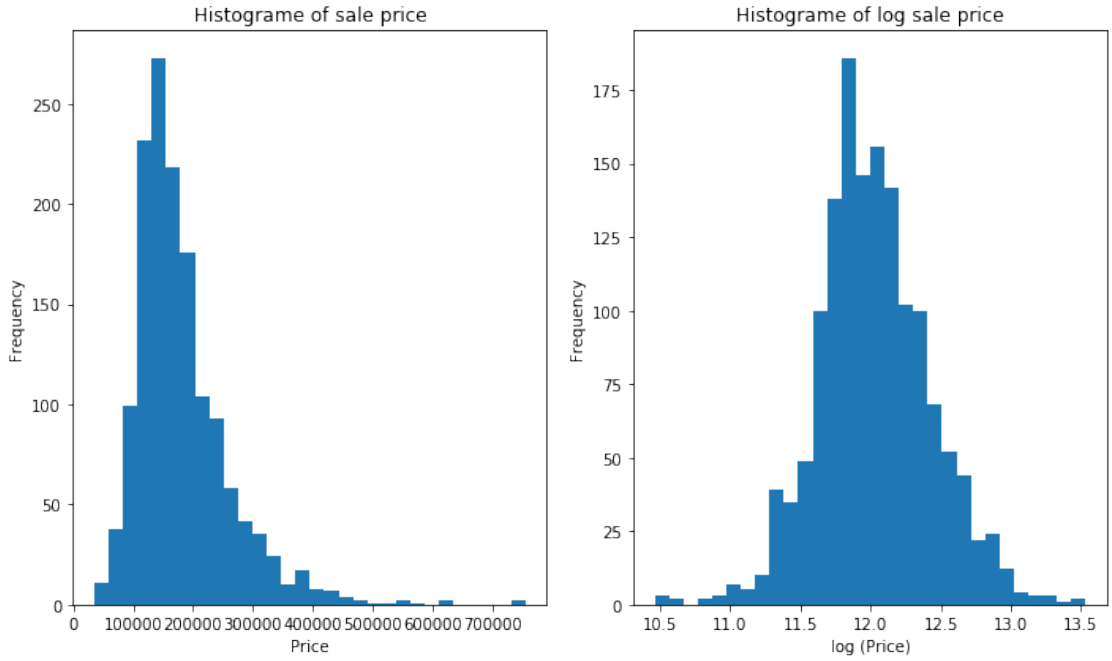
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 1 to 1460
```

```
Data columns (total 80 columns):
MSSubClass       1460 non-null int64
MSZoning         1460 non-null object
LotFrontage      1460 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null object
Alley            1460 non-null object
LotShape         1460 non-null object
LandContour      1460 non-null object
Utilities        1460 non-null object
LotConfig        1460 non-null object
LandSlope        1460 non-null object
Neighborhood     1460 non-null object
Condition1       1460 non-null object
Condition2       1460 non-null object
BldgType         1460 non-null object
HouseStyle       1460 non-null object
OverallQual      1460 non-null int64
OverallCond      1460 non-null int64
YearBuilt        1460 non-null int64
YearRemodAdd     1460 non-null int64
RoofStyle        1460 non-null object
RoofMatl         1460 non-null object
Exterior1st      1460 non-null object
Exterior2nd      1460 non-null object
MasVnrType       1460 non-null object
MasVnrArea       1460 non-null object
ExterQual        1460 non-null object
ExterCond        1460 non-null object
Foundation       1460 non-null object
BsmtQual         1460 non-null object
BsmtCond         1460 non-null object
BsmtExposure     1460 non-null object
BsmtFinType1     1460 non-null object
BsmtFinSF1       1460 non-null int64
BsmtFinType2     1460 non-null object
BsmtFinSF2       1460 non-null int64
BsmtUnfSF        1460 non-null int64
TotalBsmtSF      1460 non-null int64
Heating          1460 non-null object
HeatingQC        1460 non-null object
CentralAir       1460 non-null object
Electrical       1459 non-null object
1stFlrSF         1460 non-null int64
2ndFlrSF         1460 non-null int64
LowQualFinSF     1460 non-null int64
GrLivArea        1460 non-null int64
BsmtFullBath     1460 non-null int64
```

```
BsmtHalfBath     1460 non-null int64
FullBath         1460 non-null int64
HalfBath         1460 non-null int64
BedroomAbvGr     1460 non-null int64
KitchenAbvGr     1460 non-null int64
KitchenQual      1460 non-null object
TotRmsAbvGrd     1460 non-null int64
Functional       1460 non-null object
Fireplaces       1460 non-null int64
FireplaceQu      1460 non-null object
GarageType       1460 non-null object
GarageYrBlt      1460 non-null float64
GarageFinish     1460 non-null object
GarageCars       1460 non-null int64
GarageArea       1460 non-null int64
GarageQual       1460 non-null object
GarageCond       1460 non-null object
PavedDrive       1460 non-null object
WoodDeckSF       1460 non-null int64
OpenPorchSF      1460 non-null int64
EnclosedPorch    1460 non-null int64
3SsnPorch        1460 non-null int64
ScreenPorch      1460 non-null int64
PoolArea         1460 non-null int64
PoolQC           1460 non-null object
Fence            1460 non-null object
MiscFeature      1460 non-null object
MiscVal          1460 non-null int64
MoSold           1460 non-null int64
YrSold           1460 non-null int64
SaleType         1460 non-null object
SaleCondition    1460 non-null object
SalePrice        1460 non-null int64
dtypes: float64(2), int64(34), object(44)
memory usage: 923.9+ KB
```

```
In [9]: fig = plt.figure(figsize=(10,6))
        plt.subplot(121)
        plt.hist(data.SalePrice, bins=30)
        plt.xlabel('Price')
        plt.ylabel('Frequency')
        plt.title('Histograme of sale price');
        plt.subplot(122)
        plt.hist(np.log(data.SalePrice), bins=30)
        plt.xlabel('log (Price)')
        plt.ylabel('Frequency')
        plt.title('Histograme of log sale price')
```

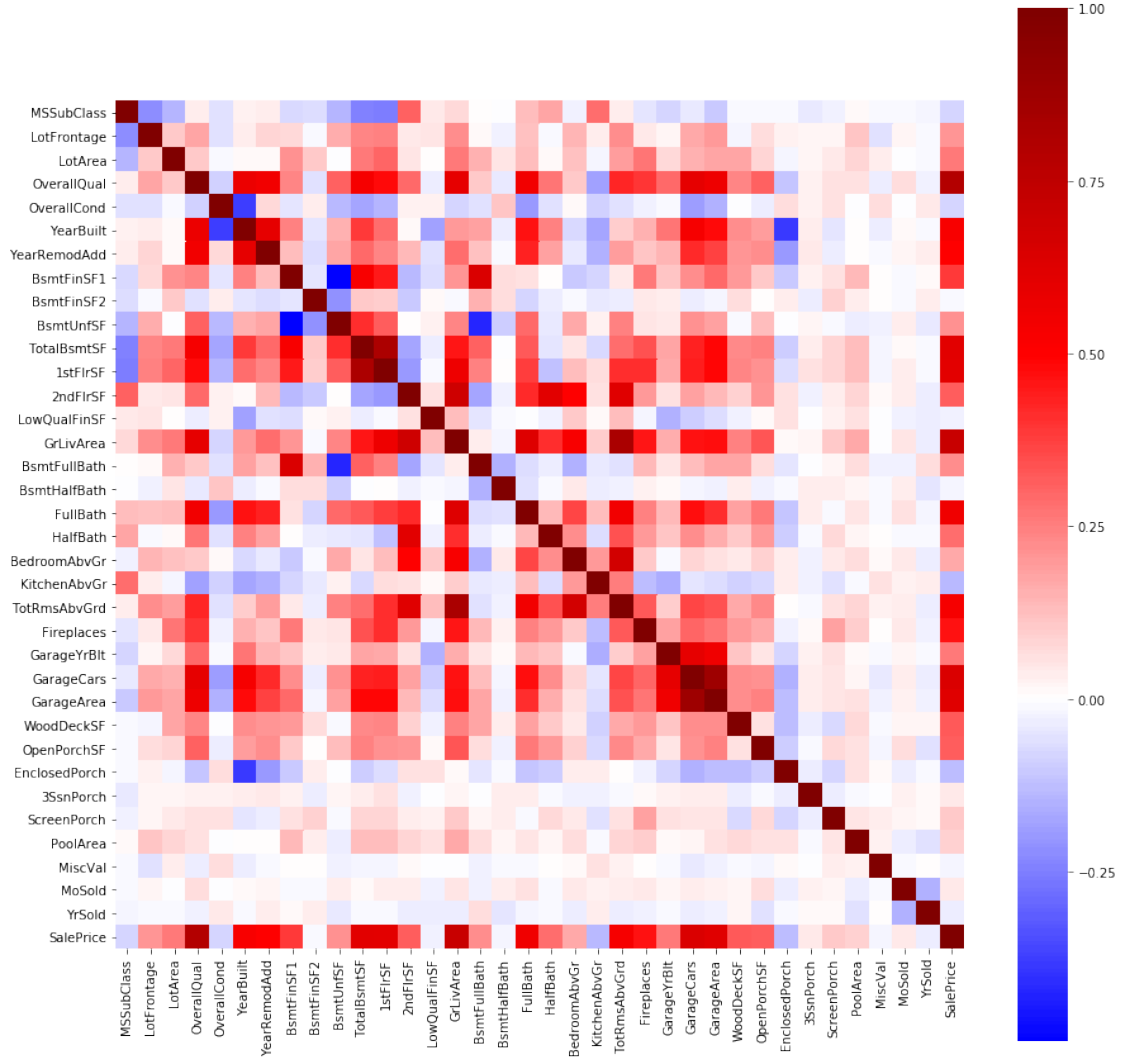
```
plt.tight_layout()
```



The plot of log(sale price) looks normal without any outliers.

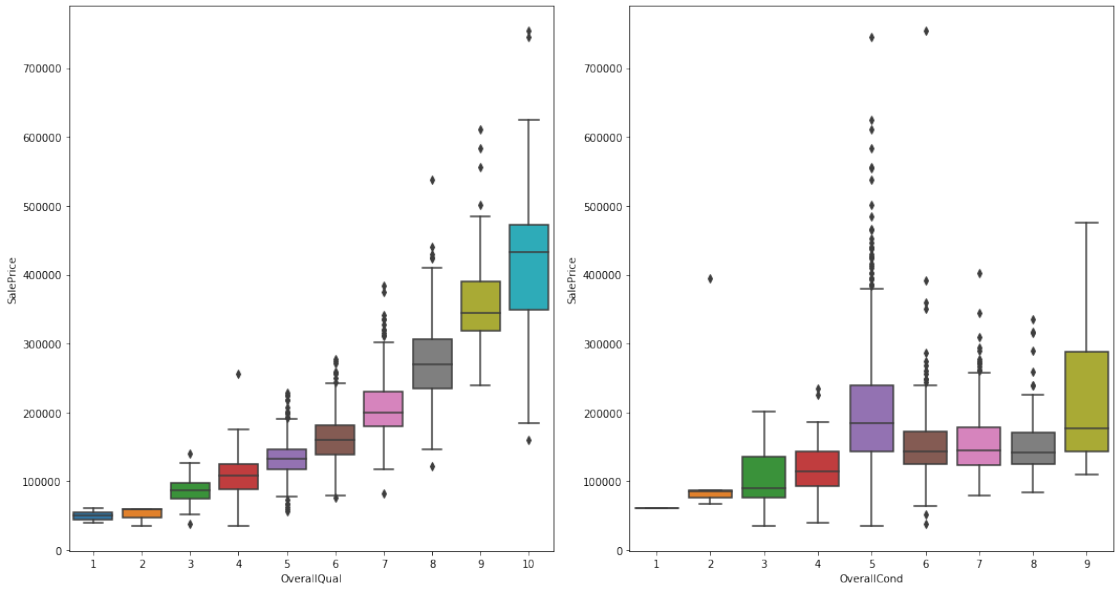
1.3 EDA for Numerical Columns:

```
In [7]: # heatmap of the Sale price, with the numerical columns
        corr = data.corr()
        fig, ax = plt.subplots(figsize=(15,15))
        sns.heatmap(corr, square=True, ax=ax, cmap='seismic', center= 0.0)
        plt.xticks(fontsize=10);
        plt.yticks(fontsize=10);
```



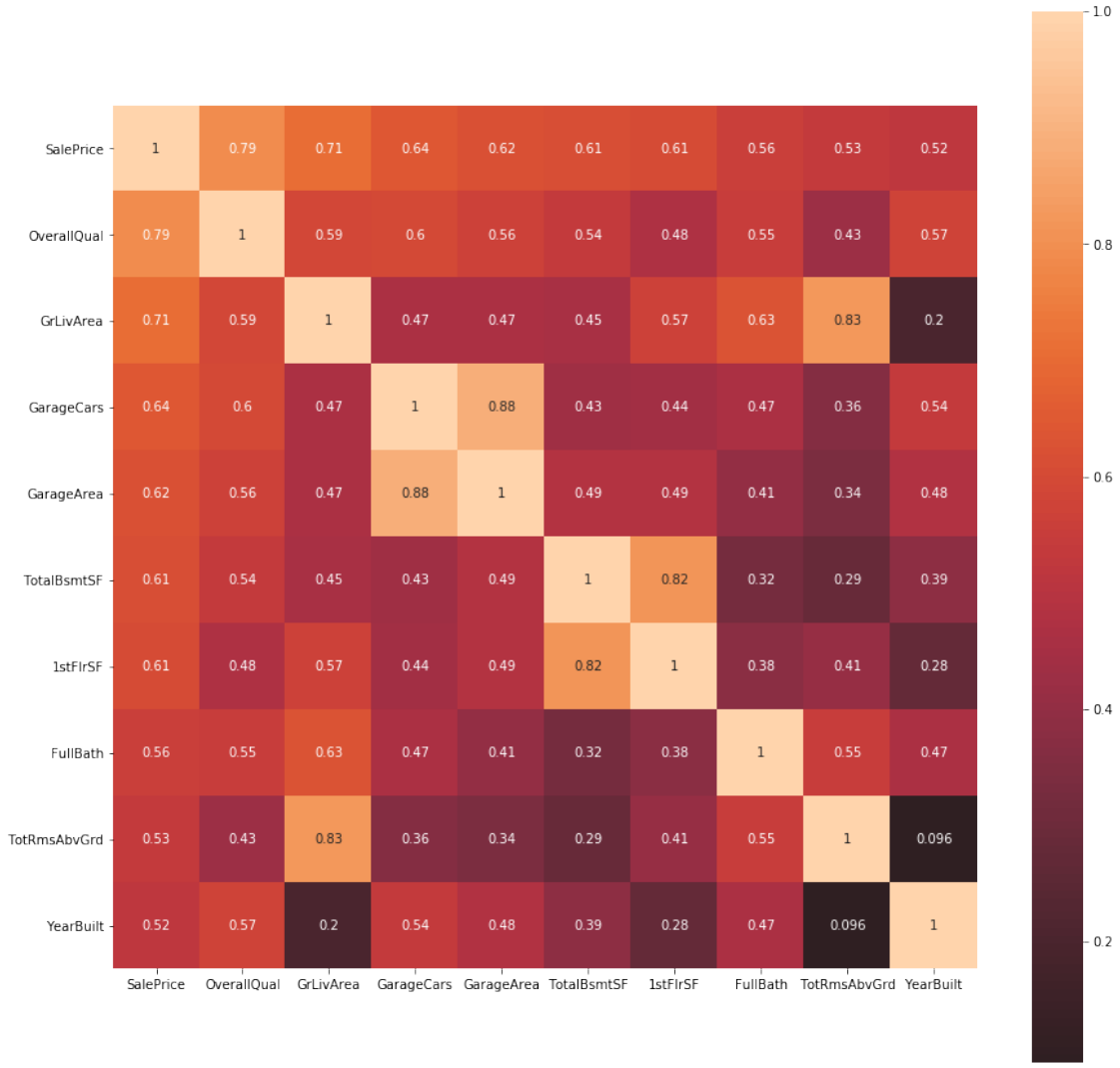
- From the above figure, there are some features which have high corrolation with the "Sale price" column, most of them with positive corrolation.
- It is interesting to find that OverallQual has high corrolation with the Sale price, on the otherhand Overallcond has a small corrolation factor with sale price.

```
In [8]: # Check OverallQual and OverallCond columns
        fig = plt.figure(figsize=(15,8))
        ax = fig.add_subplot(121)
        g = sns.catplot(x="OverallQual", y="SalePrice", kind="box", data=data, ax = ax)
        plt.close(g.fig)
        ax = fig.add_subplot(122)
        g = sns.catplot(x="OverallCond", y="SalePrice", kind="box", data=data, ax = ax)
        plt.close(g.fig)

        plt.tight_layout()
```



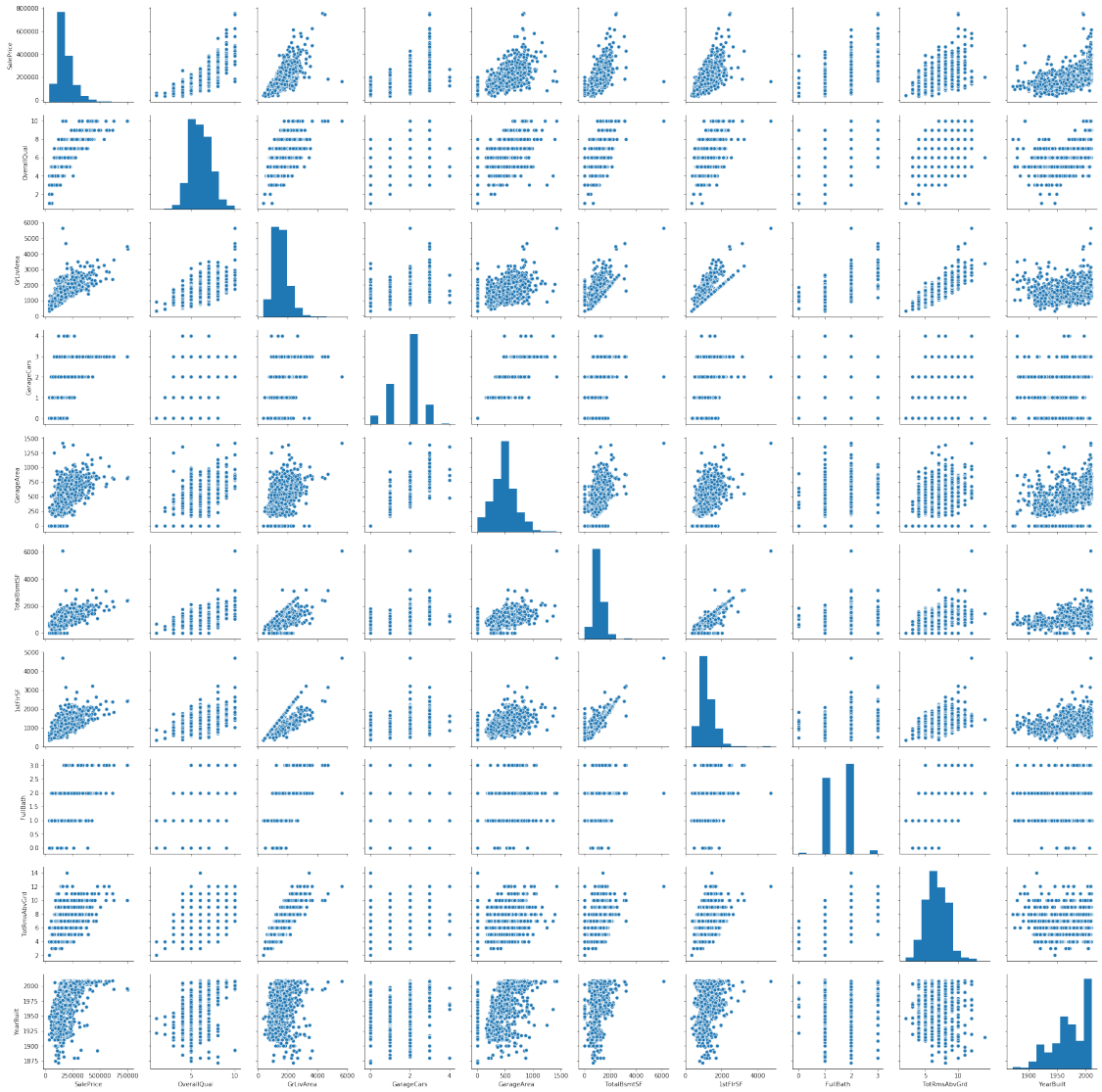
- It make sense now why OverallQual gives high corrolation with saleprice than OverallCond

```
In [9]: corr = data.corr()
        cols = corr.nlargest(10, 'SalePrice')['SalePrice'].index    # Take the max 10
        corr = data[cols].corr()
        fig, ax = plt.subplots(figsize=(15,15))
        sns.heatmap(corr, square=True, ax=ax, center= 0.0, xticklabels=cols, yticklabels=cols,
        plt.xticks(fontsize=10);
        plt.yticks(fontsize=10);
```



- Numerical columns with high corrolation with Sale price are: ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea','TotalBsmtSF', '1stFlrSF', 'FullBath', 'TotRmsAbvGrd', 'YearBuilt']

```
In [10]: sns.pairplot(data[cols]);
```

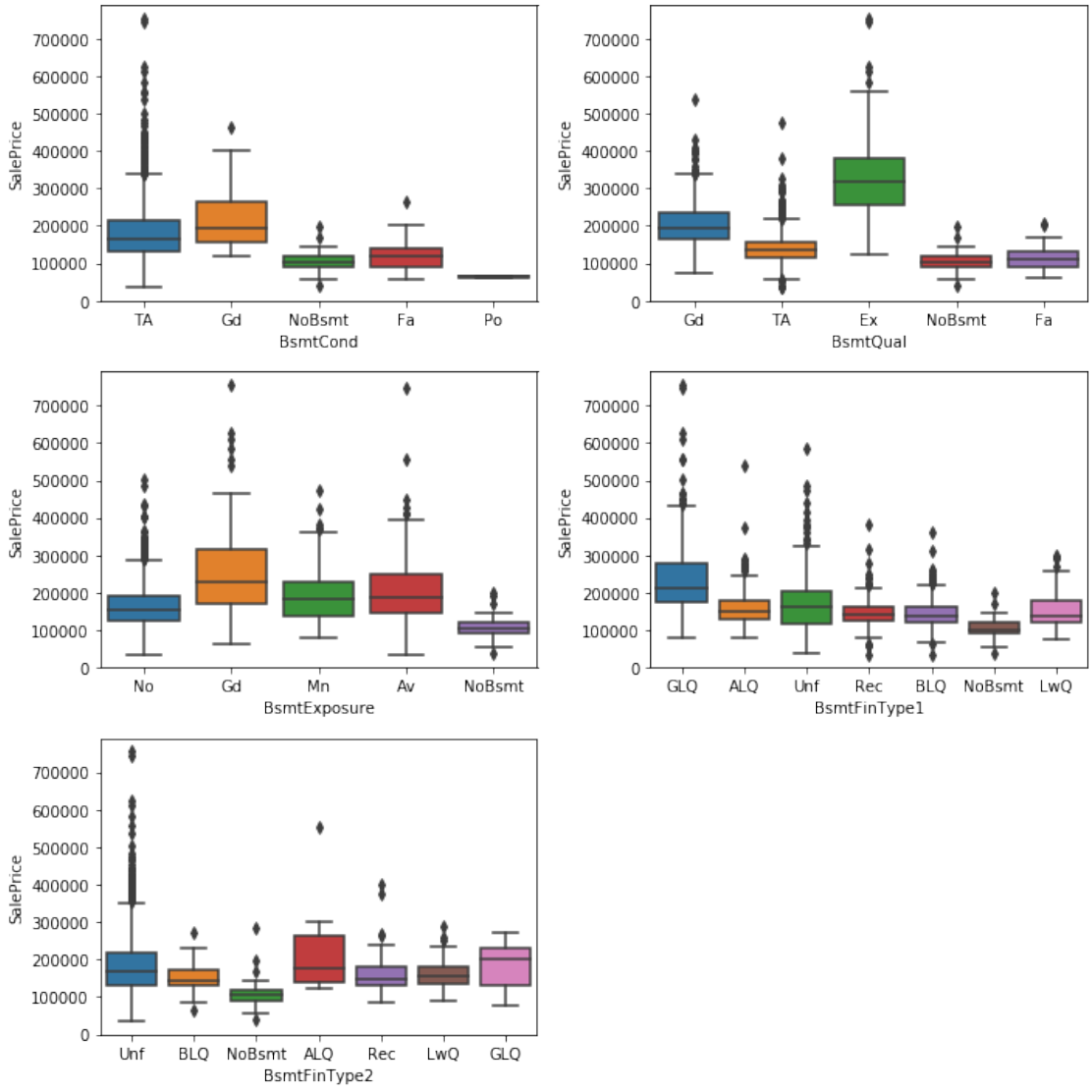


The following columns are integer and they give different corrolation with sale price: * GarageCars: Size of garage in car capacity, * FullBath: Full bathrooms above grade, and * TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)

1.4 EDA for Categorical columns

```
In [11]: # Columns related to Basement
         fig = plt.figure(figsize=(10,10))
         ax = fig.add_subplot(321)
         g = sns.catplot(x="BsmtCond", y="SalePrice", kind="box", data=data, ax=ax)
         plt.close(g.fig)
         ax = fig.add_subplot(322)
         g = sns.catplot(x="BsmtQual", y="SalePrice", kind="box", data=data, ax=ax)
         plt.close(g.fig)
```

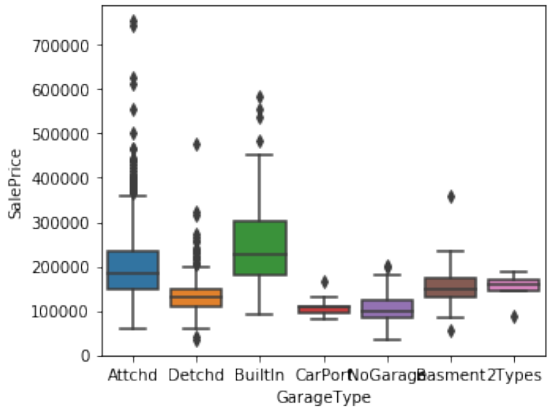
```
ax = fig.add_subplot(323)
g = sns.catplot(x="BsmtExposure", y="SalePrice", kind="box", data=data, ax=ax)
plt.close(g.fig)
ax = fig.add_subplot(324)
g = sns.catplot(x="BsmtFinType1", y="SalePrice", kind="box", data=data, ax=ax)
plt.close(g.fig)
ax = fig.add_subplot(325)
g = sns.catplot(x="BsmtFinType2", y="SalePrice", kind="box", data=data, ax=ax)
plt.close(g.fig)
plt.tight_layout()
```

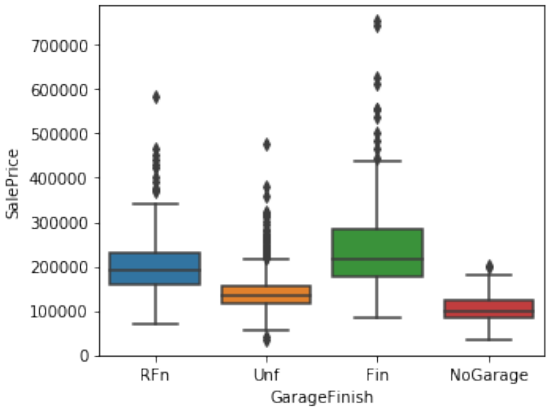


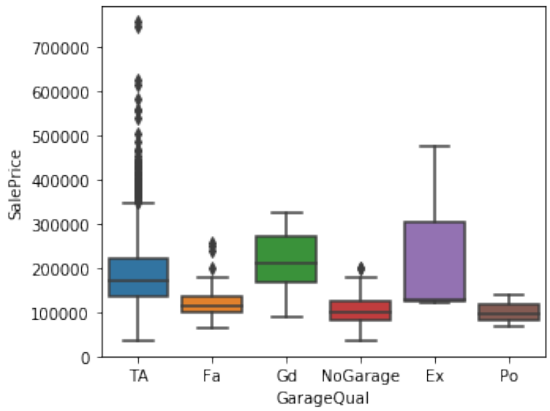
In [12]:
```
# I will use only BsmtCond and BsmtQual and drop the rest
# It is better to use lable encoder for these columns than one-hot code:
```

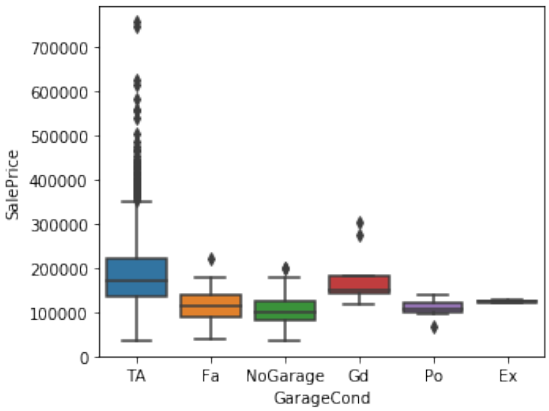
```
data.BsmtCond = data.BsmtCond.map({'Ex':5 ,'Gd':4 , 'TA':3 ,'Fa':2 ,'Po':1 , 'NoBsmt'
data.BsmtQual = data.BsmtQual.map({'Ex':5 ,'Gd':4 , 'TA':3 ,'Fa':2 ,'Po':1 , 'NoBsmt'
data.BsmtExposure = data.BsmtExposure.map({'Gd':4, 'Av':3, 'Mn':2, 'No':1, 'NoBsmt':0
data.BsmtFinType1 = data.BsmtFinType1.map({'GLQ':6,'ALQ':5,'BLQ':4,'Rec':3,'LwQ':2,'U
data.BsmtFinType2 = data.BsmtFinType2.map({'GLQ':6,'ALQ':5,'BLQ':4,'Rec':3,'LwQ':2,'U
```

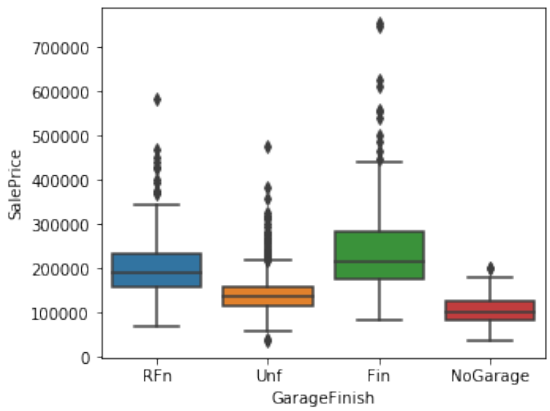
In [13]:
```
# Columns related to Garage
fig = plt.figure(figsize=(10,15))
ax = fig.add_subplot(421)
g = sns.catplot(x="GarageType", y="SalePrice", kind="box", data=data, ax=ax)
plt.close(g.fig)
ax = fig.add_subplot(422)
g = sns.catplot(x="GarageFinish", y="SalePrice", kind="box", data=data, ax=ax)
plt.close(g.fig)
ax = fig.add_subplot(423)
g = sns.catplot(x="GarageQual", y="SalePrice", kind="box", data=data, ax=ax)
plt.close(g.fig)
ax = fig.add_subplot(424)
g = sns.catplot(x="GarageCond", y="SalePrice", kind="box", data=data, ax=ax)
plt.close(g.fig)
ax = fig.add_subplot(425)
g = sns.catplot(x="GarageFinish", y="SalePrice", kind="box", data=data, ax=ax)
plt.close(g.fig)
ax = fig.add_subplot(426)
g = sns.catplot(x="GarageQual", y="SalePrice", kind="box", data=data, ax=ax)
plt.close(g.fig)
ax = fig.add_subplot(427)
g = sns.catplot(x="GarageCond", y="SalePrice", kind="box", data=data, ax=ax)
plt.close(g.fig)
ax = fig.add_subplot(428)
g = sns.catplot(x="PavedDrive", y="SalePrice", kind="box", data=data, ax=ax)
plt.close(g.fig)

plt.tight_layout()
```

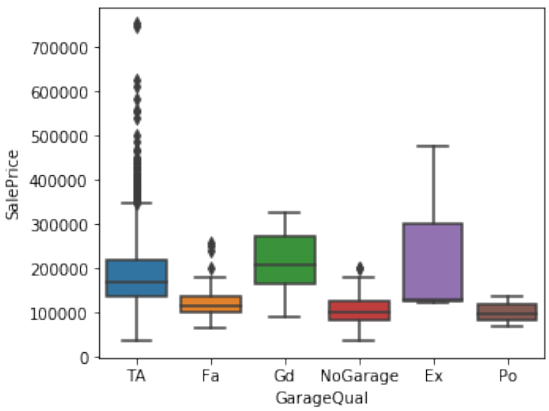

SalePrice
700000
600000
500000
400000
300000
200000
100000
0
Attchd
Detchd
BuiltIn
CarPort
NoGarage
Basment
2Types
GarageType

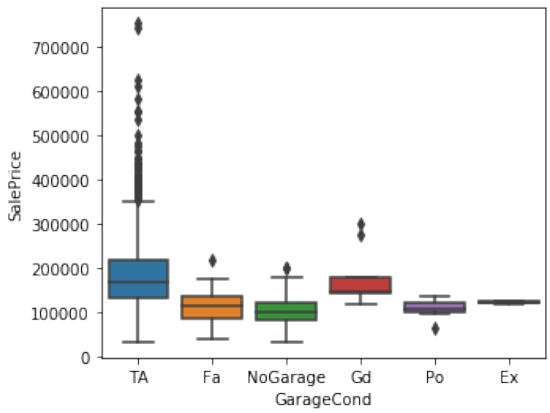

SalePrice
700000
600000
500000
400000
300000
200000
100000
0
RFn
Unf
Fin
NoGarage
GarageFinish

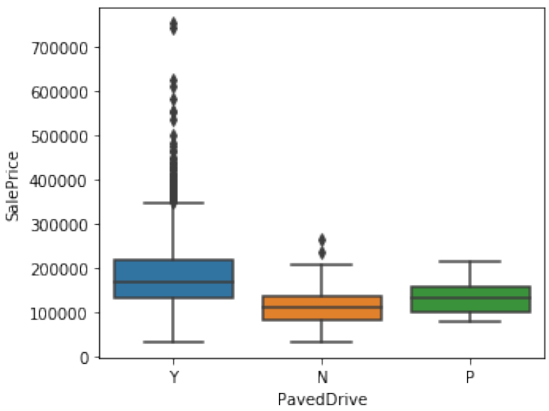

SalePrice
700000
600000
500000
400000
300000
200000
100000
0
TA
Fa
Gd
NoGarage
Ex
Po
GarageQual


SalePrice
700000
600000
500000
400000
300000
200000
100000
0
TA
Fa
NoGarage
Gd
Po
Ex
GarageCond


SalePrice
700000
600000
500000
400000
300000
200000
100000
0
RFn
Unf
Fin
NoGarage
GarageFinish

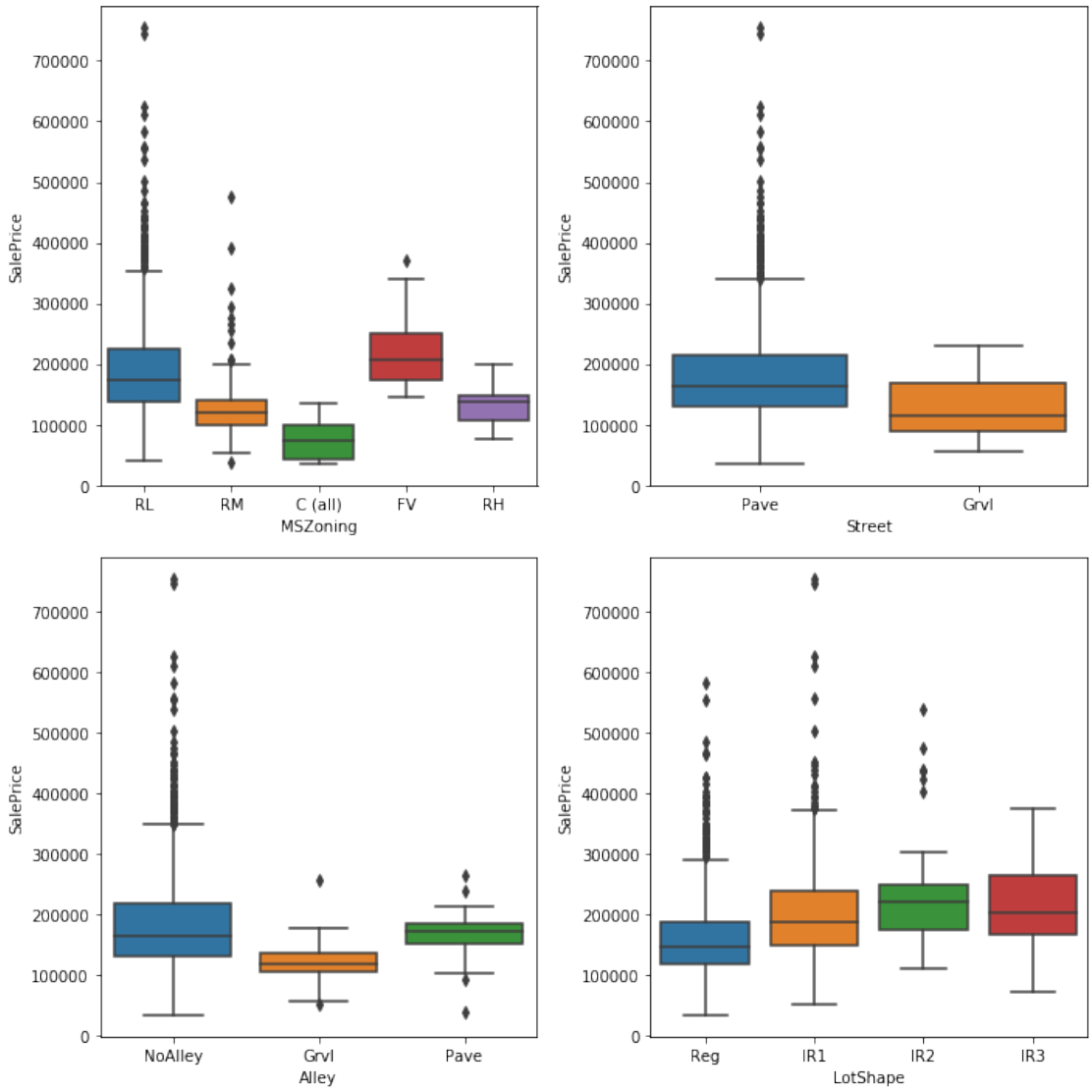

SalePrice
700000
600000
500000
400000
300000
200000
100000
0
TA
Fa
Gd
NoGarage
Ex
Po
GarageQual


SalePrice
700000
600000
500000
400000
300000
200000
100000
0
TA
Fa
NoGarage
Gd
Po
Ex
GarageCond

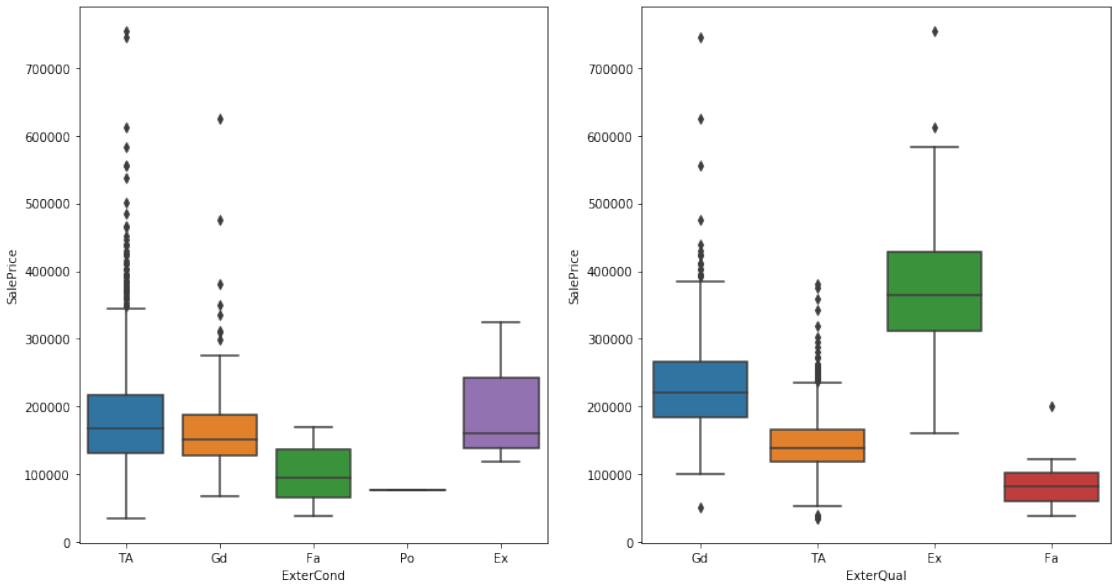

SalePrice
700000
600000
500000
400000
300000
200000
100000
0
Y
N
P
PavedDrive

In [14]:
```
data.GarageType = data.GarageType.map({'2Types':4 , 'Attchd': 5, 'Basment':3 ,'BuiltIn
                                       'CarPort' :1, 'Detchd':2 , 'NoGarage': 0})

data.GarageCond = data.GarageCond.map({'NoGarage':0, 'Po':1, 'Fa':2, 'TA':3, 'Gd':4,
data.GarageQual = data.GarageQual.map({'NoGarage':0, 'Po':1, 'Fa':2, 'TA':3, 'Gd':4,
data.GarageFinish = data.GarageFinish.map({'Fin':3, 'RFn':2, 'Unf':1, 'NoGarage':0})
data.PavedDrive = data.PavedDrive.map({'Y':2,'P':1, 'N':0 })
```

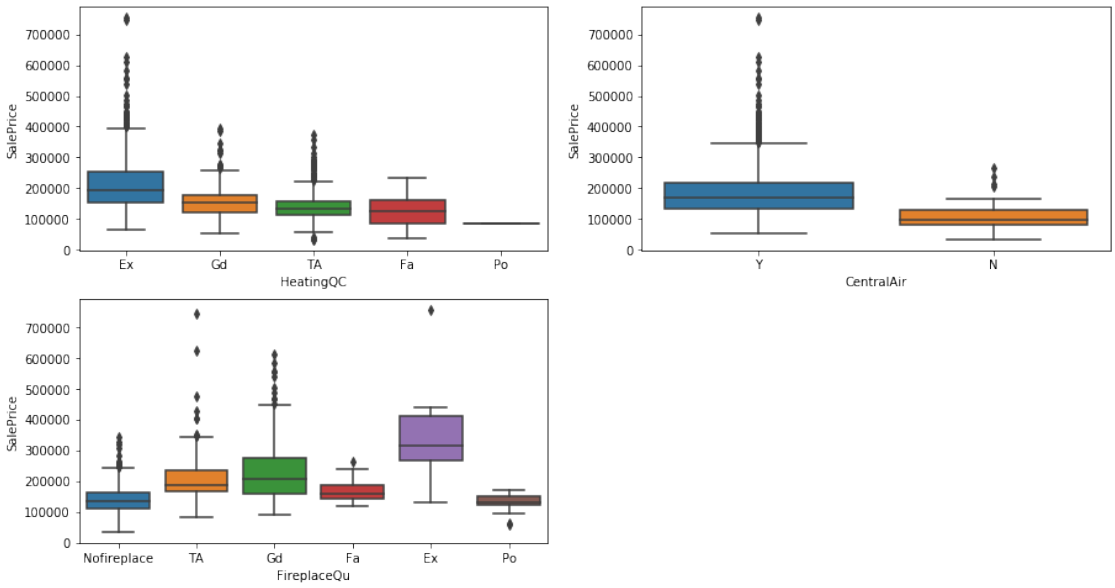
In [15]:
```
# Columns related to surrounding condition
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(221)
g = sns.catplot(x="MSZoning", y="SalePrice", kind="box", data=data, ax=ax)
plt.close(g.fig)
ax = fig.add_subplot(222)
g = sns.catplot(x="Street", y="SalePrice", kind="box", data=data, ax=ax)
plt.close(g.fig)
ax = fig.add_subplot(223)
g = sns.catplot(x="Alley", y="SalePrice", kind="box", data=data, ax=ax)
plt.close(g.fig)
ax = fig.add_subplot(224)
g = sns.catplot(x="LotShape", y="SalePrice", kind="box", data=data, ax=ax)
plt.close(g.fig)

plt.tight_layout()
```



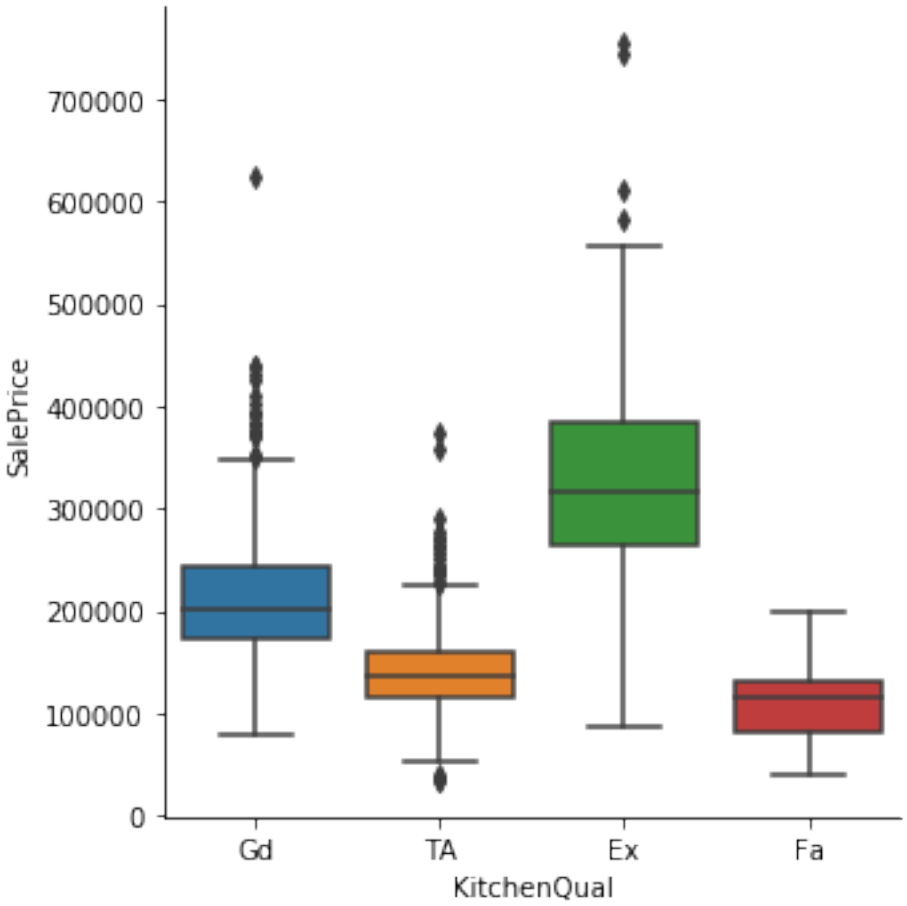

```
In [16]: # Columns related to surrounding condition
         fig = plt.figure(figsize=(15,8))
         ax = fig.add_subplot(121)
         g = sns.catplot(x="ExterCond", y="SalePrice", kind="box", data=data, ax=ax)
         plt.close(g.fig)
         ax = fig.add_subplot(122)
         g = sns.catplot(x="ExterQual", y="SalePrice", kind="box", data=data, ax=ax)
         plt.close(g.fig)
         data.ExterCond = data.ExterCond.map({"Ex":4,'Gd':3,'TA':2,'Fa':1,'Po':0})
         data.ExterQual = data.ExterQual.map({"Ex":4,'Gd':3,'TA':2,'Fa':1,'Po':0})
```



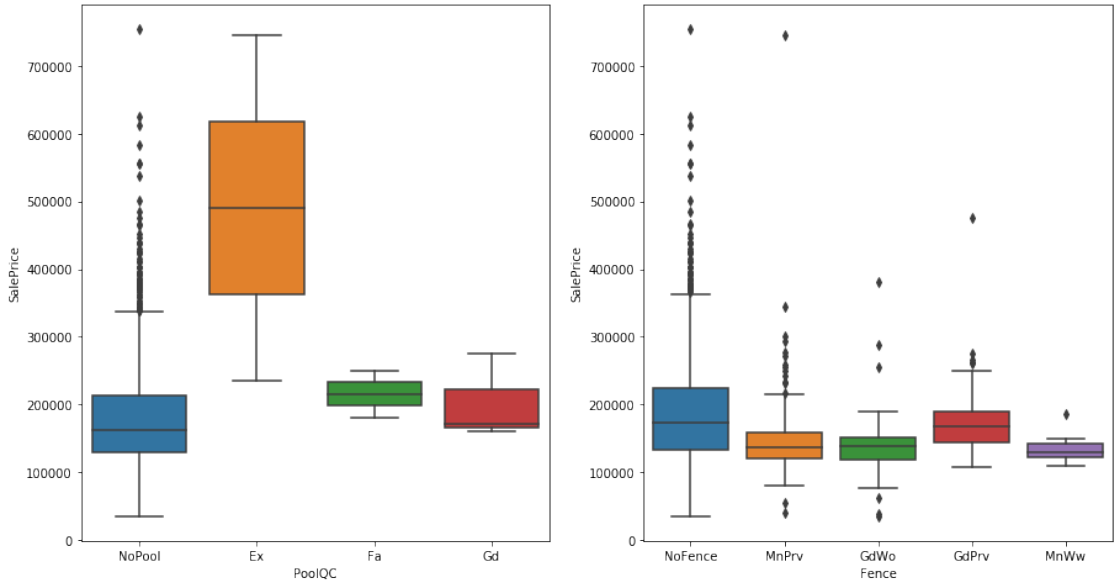

```
In [17]: fig = plt.figure(figsize=(15,8))
         ax = fig.add_subplot(221)
         g = sns.catplot(x="HeatingQC", y="SalePrice", kind="box", data=data, ax=ax)
         plt.close(g.fig)
         ax = fig.add_subplot(222)
         g = sns.catplot(x="CentralAir", y="SalePrice", kind="box", data=data, ax=ax)
         plt.close(g.fig)
         ax = fig.add_subplot(223)
         g = sns.catplot(x="FireplaceQu", y="SalePrice", kind="box", data=data, ax=ax)
         plt.close(g.fig)
         data.CentralAir = data.CentralAir.map({'Y':1, 'N':0})
         data.HeatingQC = data.HeatingQC.map({"Ex":4,'Gd':3,'TA':2,'Fa':1,'Po':0})
         data.FireplaceQu = data.FireplaceQu.map({"Ex":5,'Gd':4,'TA':3,'Fa':2,'Po':1, 'Nofirep
```




```
In [18]: sns.catplot(x="KitchenQual", y="SalePrice", kind="box", data=data)
         data.KitchenQual = data.KitchenQual.map({"Ex":4,'Gd':3,'TA':2,'Fa':1,'Po':0})
```




```
In [19]: fig = plt.figure(figsize=(15,8))
         ax = fig.add_subplot(121)
         g = sns.catplot(x="PoolQC", y="SalePrice", kind="box", data=data, ax=ax)
         plt.close(g.fig)
         ax = fig.add_subplot(122)
         g = sns.catplot(x="Fence", y="SalePrice", kind="box", data=data, ax=ax)
         plt.close(g.fig)
         data.PoolQC = data.PoolQC.map({"Ex":4,'Gd':3,'TA':2,'Fa':1,'NoPool':0})
         data.Fence = data.Fence.map({'GdPrv':4 , 'MnPrv':3 , 'GdWo':2 , 'MnWw':1 , 'NoFence':
```



1.5 Converting categorical columns

- The task no is to convert the rest of the categorical columns into one-hot code.

```
In [20]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 1 to 1460
Data columns (total 80 columns):
MSSubClass       1460 non-null int64
MSZoning         1460 non-null object
LotFrontage      1460 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null object
Alley            1460 non-null object
LotShape         1460 non-null object
LandContour      1460 non-null object
Utilities        1460 non-null object
LotConfig        1460 non-null object
LandSlope        1460 non-null object
Neighborhood     1460 non-null object
Condition1       1460 non-null object
Condition2       1460 non-null object
BldgType         1460 non-null object
HouseStyle       1460 non-null object
OverallQual      1460 non-null int64
OverallCond      1460 non-null int64
YearBuilt        1460 non-null int64
```

```
YearRemodAdd     1460 non-null int64
RoofStyle        1460 non-null object
RoofMatl         1460 non-null object
Exterior1st      1460 non-null object
Exterior2nd      1460 non-null object
MasVnrType       1460 non-null object
MasVnrArea       1460 non-null object
ExterQual        1460 non-null int64
ExterCond        1460 non-null int64
Foundation       1460 non-null object
BsmtQual         1460 non-null int64
BsmtCond         1460 non-null int64
BsmtExposure     1460 non-null int64
BsmtFinType1     1460 non-null int64
BsmtFinSF1       1460 non-null int64
BsmtFinType2     1460 non-null int64
BsmtFinSF2       1460 non-null int64
BsmtUnfSF        1460 non-null int64
TotalBsmtSF      1460 non-null int64
Heating          1460 non-null object
HeatingQC        1460 non-null int64
CentralAir       1460 non-null int64
Electrical       1459 non-null object
1stFlrSF         1460 non-null int64
2ndFlrSF         1460 non-null int64
LowQualFinSF     1460 non-null int64
GrLivArea        1460 non-null int64
BsmtFullBath     1460 non-null int64
BsmtHalfBath     1460 non-null int64
FullBath         1460 non-null int64
HalfBath         1460 non-null int64
BedroomAbvGr     1460 non-null int64
KitchenAbvGr     1460 non-null int64
KitchenQual      1460 non-null int64
TotRmsAbvGrd     1460 non-null int64
Functional       1460 non-null object
Fireplaces       1460 non-null int64
FireplaceQu      1460 non-null int64
GarageType       1460 non-null int64
GarageYrBlt      1460 non-null float64
GarageFinish     1460 non-null int64
GarageCars       1460 non-null int64
GarageArea       1460 non-null int64
GarageQual       1460 non-null int64
GarageCond       1460 non-null int64
PavedDrive       1460 non-null int64
WoodDeckSF       1460 non-null int64
OpenPorchSF      1460 non-null int64
```

```
EnclosedPorch    1460 non-null int64
3SsnPorch        1460 non-null int64
ScreenPorch      1460 non-null int64
PoolArea         1460 non-null int64
PoolQC           1460 non-null int64
Fence            1460 non-null int64
MiscFeature      1460 non-null object
MiscVal          1460 non-null int64
MoSold           1460 non-null int64
YrSold           1460 non-null int64
SaleType         1460 non-null object
SaleCondition    1460 non-null object
SalePrice        1460 non-null int64
dtypes: float64(2), int64(52), object(26)
memory usage: 963.9+ KB
```

```
In [21]: data.Electrical.unique()  # One missing value in this column
```

```
Out[21]: array(['SBrkr', 'FuseF', 'FuseA', 'FuseP', 'Mix', nan], dtype=object)
```

We have only one missing observation, I will drop it.

```
In [22]: data.dropna(inplace=True)
```

```
In [27]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1459 entries, 1 to 1460
Data columns (total 80 columns):
MSSubClass       1459 non-null int64
MSZoning         1459 non-null object
LotFrontage      1459 non-null float64
LotArea          1459 non-null int64
Street           1459 non-null object
Alley            1459 non-null object
LotShape         1459 non-null object
LandContour      1459 non-null object
Utilities        1459 non-null object
LotConfig        1459 non-null object
LandSlope        1459 non-null object
Neighborhood     1459 non-null object
Condition1       1459 non-null object
Condition2       1459 non-null object
BldgType         1459 non-null object
HouseStyle       1459 non-null object
OverallQual      1459 non-null int64
OverallCond      1459 non-null int64
YearBuilt        1459 non-null int64
```

```
YearRemodAdd     1459 non-null int64
RoofStyle        1459 non-null object
RoofMatl         1459 non-null object
Exterior1st      1459 non-null object
Exterior2nd      1459 non-null object
MasVnrType       1459 non-null object
MasVnrArea       1459 non-null object
ExterQual        1459 non-null int64
ExterCond        1459 non-null int64
Foundation       1459 non-null object
BsmtQual         1459 non-null int64
BsmtCond         1459 non-null int64
BsmtExposure     1459 non-null int64
BsmtFinType1     1459 non-null int64
BsmtFinSF1       1459 non-null int64
BsmtFinType2     1459 non-null int64
BsmtFinSF2       1459 non-null int64
BsmtUnfSF        1459 non-null int64
TotalBsmtSF      1459 non-null int64
Heating          1459 non-null object
HeatingQC        1459 non-null int64
CentralAir       1459 non-null int64
Electrical       1459 non-null object
1stFlrSF         1459 non-null int64
2ndFlrSF         1459 non-null int64
LowQualFinSF     1459 non-null int64
GrLivArea        1459 non-null int64
BsmtFullBath     1459 non-null int64
BsmtHalfBath     1459 non-null int64
FullBath         1459 non-null int64
HalfBath         1459 non-null int64
BedroomAbvGr     1459 non-null int64
KitchenAbvGr     1459 non-null int64
KitchenQual      1459 non-null int64
TotRmsAbvGrd     1459 non-null int64
Functional       1459 non-null object
Fireplaces       1459 non-null int64
FireplaceQu      1459 non-null int64
GarageType       1459 non-null int64
GarageYrBlt      1459 non-null float64
GarageFinish     1459 non-null int64
GarageCars       1459 non-null int64
GarageArea       1459 non-null int64
GarageQual       1459 non-null int64
GarageCond       1459 non-null int64
PavedDrive       1459 non-null int64
WoodDeckSF       1459 non-null int64
OpenPorchSF      1459 non-null int64
```

```
EnclosedPorch    1459 non-null int64
3SsnPorch        1459 non-null int64
ScreenPorch      1459 non-null int64
PoolArea         1459 non-null int64
PoolQC           1459 non-null int64
Fence            1459 non-null int64
MiscFeature      1459 non-null object
MiscVal          1459 non-null int64
MoSold           1459 non-null int64
YrSold           1459 non-null int64
SaleType         1459 non-null object
SaleCondition    1459 non-null object
SalePrice        1459 non-null int64
dtypes: float64(2), int64(52), object(26)
memory usage: 923.3+ KB
```

```
In [23]: n_data = pd.get_dummies(data, drop_first= True)
```

```
In [24]: n_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1459 entries, 1 to 1460
Columns: 533 entries, MSSubClass to SaleCondition_Partial
dtypes: float64(2), int64(52), uint8(479)
memory usage: 1.3 MB
```

```
In [25]: n_data.head()
```

Out[25]:

Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	\
1	60	65.0	8450	7	5	2003	
2	20	80.0	9600	6	8	1976	
3	60	68.0	11250	7	5	2001	
4	70	60.0	9550	7	5	1915	
5	60	84.0	14260	8	5	2000	

Id	YearRemodAdd	ExterQual	ExterCond	BsmtQual	...	\
1	2003	3	2	4	...	
2	1976	2	2	4	...	
3	2002	3	2	4	...	
4	1970	2	2	3	...	
5	2000	3	2	4	...	

Id	SaleType_ConLI	SaleType_ConLw	SaleType_New	SaleType_Oth	SaleType_WD	\
1	0	0	0	0	1	

```
2                0              0              0              0           1
3                0              0              0              0           1
4                0              0              0              0           1
5                0              0              0              0           1

    SaleCondition_AdjLand  SaleCondition_Alloca  SaleCondition_Family  \
Id
1                       0                     0                     0
2                       0                     0                     0
3                       0                     0                     0
4                       0                     0                     0
5                       0                     0                     0

    SaleCondition_Normal  SaleCondition_Partial
Id
1                      1                     0
2                      1                     0
3                      1                     0
4                      0                     0
5                      1                     0

[5 rows x 533 columns]
```

In [28]:
```
n_data.to_csv('../clean_data.csv')
```

In [29]:
```
data.to_csv('../semi_clean_data.csv')
```

[20]
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from scipy import stats
from math import ceil

%matplotlib inline
```

1. Load data

[52]
```
df = pd.read_csv('../train.csv', index_col=0)
df.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	I
Id							
1	60	RL	65.0	8450	Pave	NaN	I
2	20	RL	80.0	9600	Pave	NaN	I
3	60	RL	68.0	11250	Pave	NaN	I
4	70	RL	60.0	9550	Pave	NaN	I
5	60	RL	84.0	14260	Pave	NaN	I

5 rows × 80 columns

[53]
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 1 to 1460
Data columns (total 80 columns):
MSSubClass       1460 non-null int64
MSZoning         1460 non-null object
LotFrontage      1201 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null object
Alley            91 non-null object
LotShape         1460 non-null object
LandContour      1460 non-null object
Utilities        1460 non-null object
LotConfig        1460 non-null object
LandSlope        1460 non-null object
Neighborhood     1460 non-null object
Condition1       1460 non-null object
```

```
Condition2       1460 non-null object
BldgType         1460 non-null object
HouseStyle       1460 non-null object
OverallQual      1460 non-null int64
OverallCond      1460 non-null int64
YearBuilt        1460 non-null int64
YearRemodAdd     1460 non-null int64
RoofStyle        1460 non-null object
RoofMatl         1460 non-null object
Exterior1st      1460 non-null object
Exterior2nd      1460 non-null object
MasVnrType       1452 non-null object
MasVnrArea       1452 non-null float64
ExterQual        1460 non-null object
ExterCond        1460 non-null object
Foundation       1460 non-null object
```

2. Clean Data

2.1 Columns with NaN Values

[54]
```
cols_with_na = df.isnull().sum()
cols_with_na = cols_with_na[cols_with_na>0]
print(cols_with_na.sort_values(ascending=False))
```

```
PoolQC          1453
MiscFeature     1406
Alley           1369
Fence           1179
FireplaceQu      690
LotFrontage      259
GarageYrBlt       81
GarageType        81
GarageFinish      81
GarageQual        81
GarageCond        81
BsmtFinType2      38
BsmtExposure      38
BsmtFinType1      37
BsmtCond          37
BsmtQual          37
MasVnrArea         8
MasVnrType         8
Electrical         1
dtype: int64
```

2.2 Meaningful NaN Values

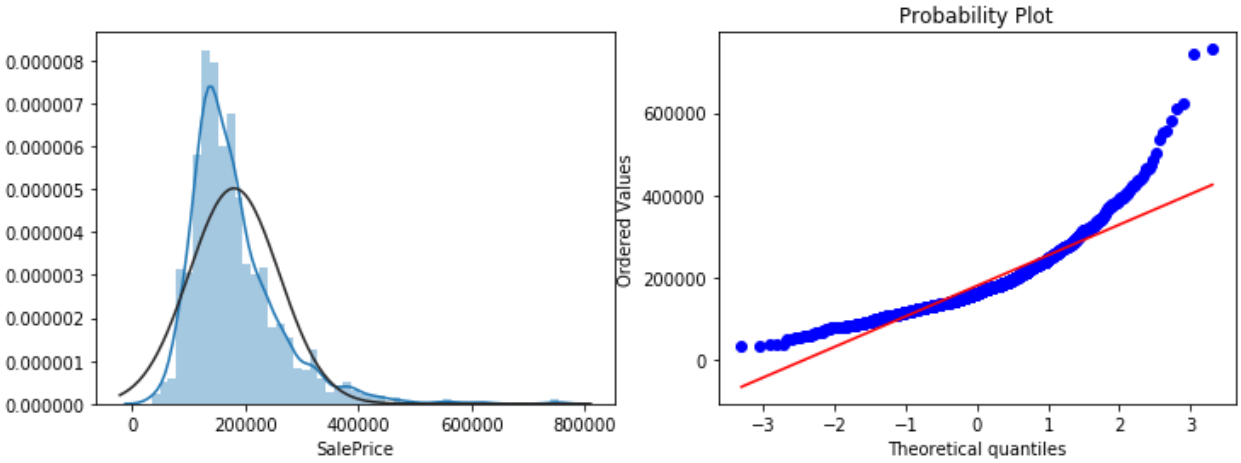
[83]
```
df.Alley = df.Alley.fillna(value = 'NoAlley')
df.BsmtCond = df.BsmtCond.fillna(value = 'NoBsmt')
df.BsmtQual = df.BsmtQual.fillna(value = 'NoBsmt')
df.BsmtExposure = df.BsmtExposure.fillna(value= 'NoBsmt')
df.BsmtFinType1 = df.BsmtFinType1.fillna(value= 'NoBsmt')
df.BsmtFinType2 = df.BsmtFinType2.fillna(value= 'NoBsmt')
df.LotFrontage = df.LotFrontage.fillna(value = 0)
df.FireplaceQu = df.FireplaceQu.fillna(value = 'Nofireplace')
df.GarageType = df.GarageType.fillna(value = 'NoGarage')
df.GarageCond = df.GarageCond.fillna(value = 'NoGarage')
df.GarageFinish = df.GarageFinish.fillna(value = 'NoGarage')
df.GarageYrBlt = df.GarageYrBlt.fillna(value = 0)
df.GarageQual = df.GarageQual.fillna(value = 'NoGarage')

df.PoolQC = df.PoolQC.fillna(value = 'NoPool')
df.Fence = df.Fence.fillna(value = 'NoFence')
df.MiscFeature = df.MiscFeature.fillna(value = 'NoMisc')
df.MasVnrType = df.MasVnrType.fillna(value = 'noMas')
df.MasVnrArea = df.MasVnrArea.fillna(value = 'noMas')

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 1 to 1460
Data columns (total 80 columns):
MSSubClass       1460 non-null category
MSZoning         1460 non-null category
LotFrontage      1460 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null category
Alley            1460 non-null category
LotShape         1460 non-null category
LandContour      1460 non-null category
Utilities        1460 non-null category
LotConfig        1460 non-null category
LandSlope        1460 non-null category
Neighborhood     1460 non-null category
Condition1       1460 non-null category
Condition2       1460 non-null category
BldgType         1460 non-null category
HouseStyle       1460 non-null category
OverallQual      1460 non-null int64
OverallCond      1460 non-null int64
YearBuilt        1460 non-null int64
YearRemodAdd     1460 non-null int64
RoofStyle        1460 non-null category
RoofMatl         1460 non-null category
Exterior1st      1460 non-null category
Exterior2nd      1460 non-null category
MasVnrType       1460 non-null category
MasVnrArea       1460 non-null category
ExterQual        1460 non-null int64
ExterCond        1460 non-null int64
Foundation       1460 non-null category
```

2.3 Distribution of SalePrice

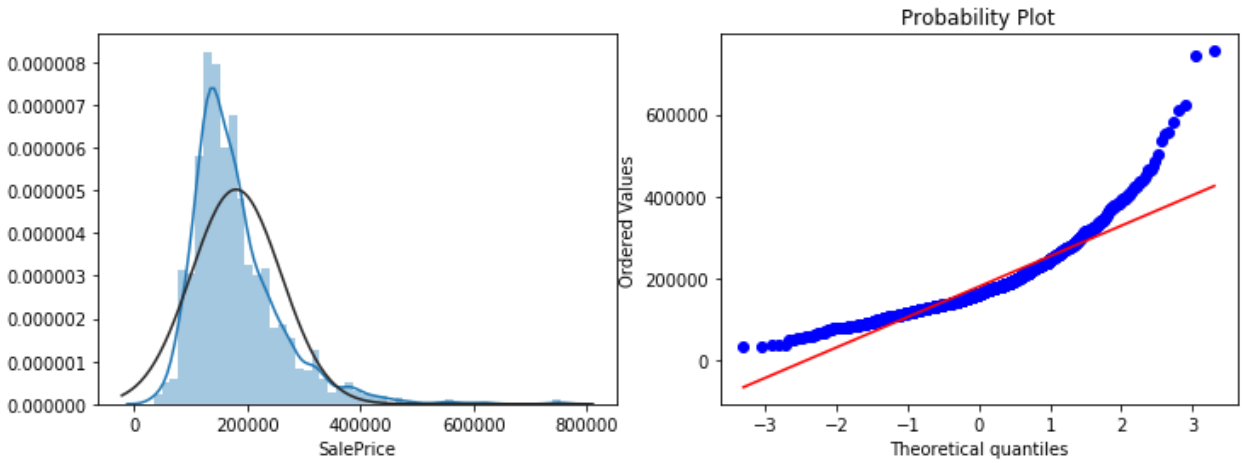
[56]
```
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
_ = sns.distplot(df.SalePrice.dropna() , fit=stats.norm);
plt.subplot(1,2,2)
_=stats.probplot(df.SalePrice.dropna(), plot=plt)
```

```
/Users/changyaohua/anaconda3/lib/python3.7/site-
packages/scipy/stats/stats.py:1713: FutureWarning: Using a non-tuple
sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]`
instead of `arr[seq]`. In the future this will be interpreted as an array
index, `arr[np.array(seq)]`, which will result either in an error or a
different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



2.4 Log Transform SalePrice

[57]
```
#Log Transform SalePrice to improve normality
sp = df.SalePrice
df.SalePrice = np.log(sp)

plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
_ = sns.distplot(sp.dropna() , fit=stats.norm);
plt.subplot(1,2,2)
_=stats.probplot(sp.dropna(), plot=plt)
```



3. Exploratory Data Analysis (EDA)

Different types of features will need to be treated differently when digging deeper in to the data. Here I identify three types of features:

- Numeric-discrete: Numeric features with less than 13 unique values, such as month of the year, or the numeric scales created above.
- Numeric-continuous: Numeric features with 13 or more unique values, such as areas, the year a property was built etc.
- Categorical: The remaining non-numeric features.

[58]
```
# extract names of numeric columns
dtypes = df.dtypes
cols_numeric = dtypes[dtypes != object].index.tolist()

# MSubClass should be treated as categorical
cols_numeric.remove('MSSubClass')

# choose any numeric column with less than 13 values to be
# "discrete". 13 chosen to include months of the year.
# other columns "continuous"
col_nunique = dict()

for col in cols_numeric:
    col_nunique[col] = df[col].nunique()

col_nunique = pd.Series(col_nunique)

cols_discrete = col_nunique[col_nunique<13].index.tolist()
cols_continuous = col_nunique[col_nunique>=13].index.tolist()

print(len(cols_numeric),'numeric columns, of which',
      len(cols_continuous),'are continuous and',
      len(cols_discrete),'are discrete.')
```

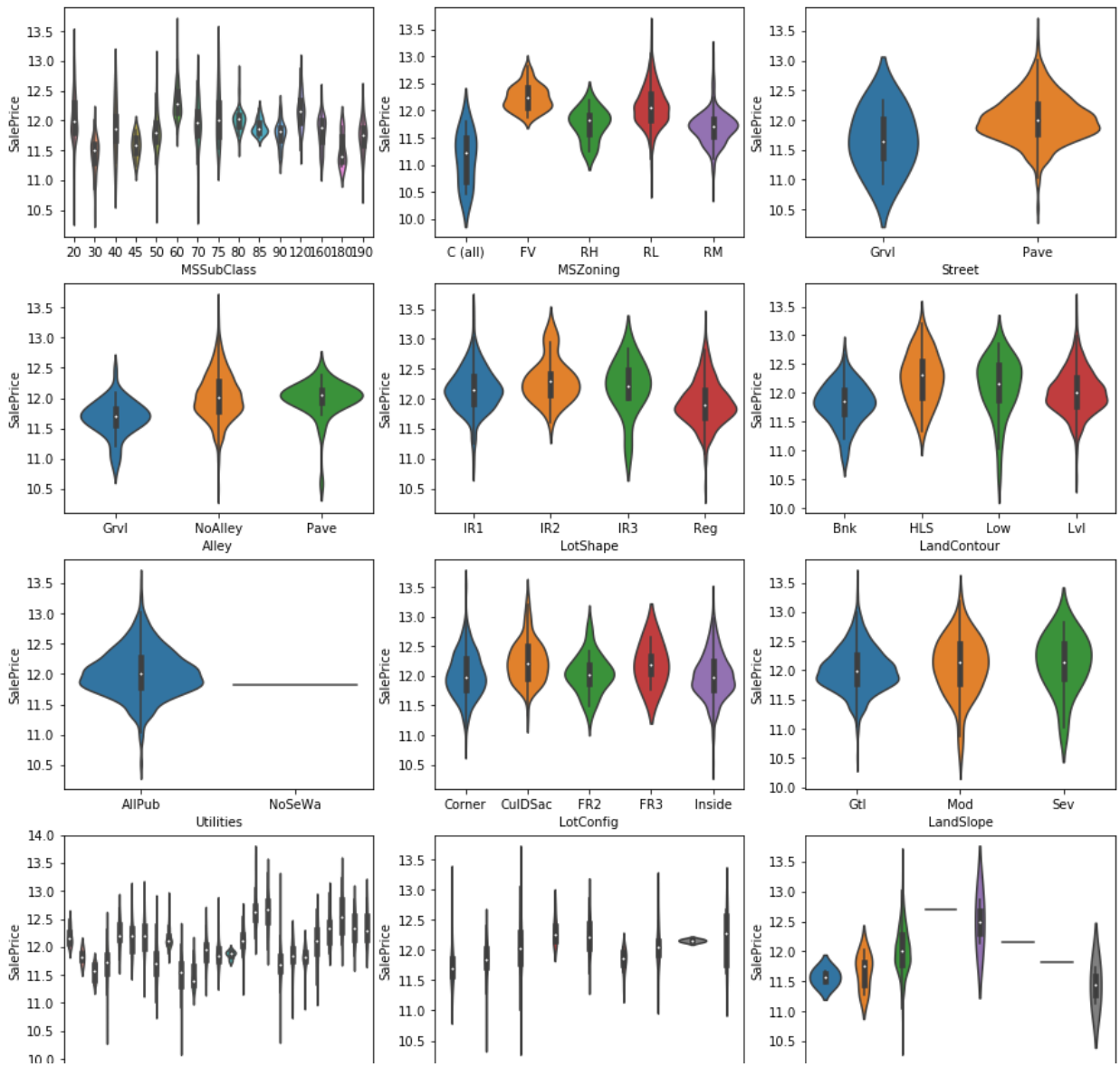
```
35 numeric columns, of which 21 are continuous and 14 are discrete.
```

[59]
```
# extract names of categorical columns
cols_categ = dtypes[~dtypes.index.isin(cols_numeric)].index.tolist()

for col in cols_categ:
    df[col] = df[col].astype('category')

print(len(cols_categ),'categorical columns.')
```

```
45 categorical columns.
```

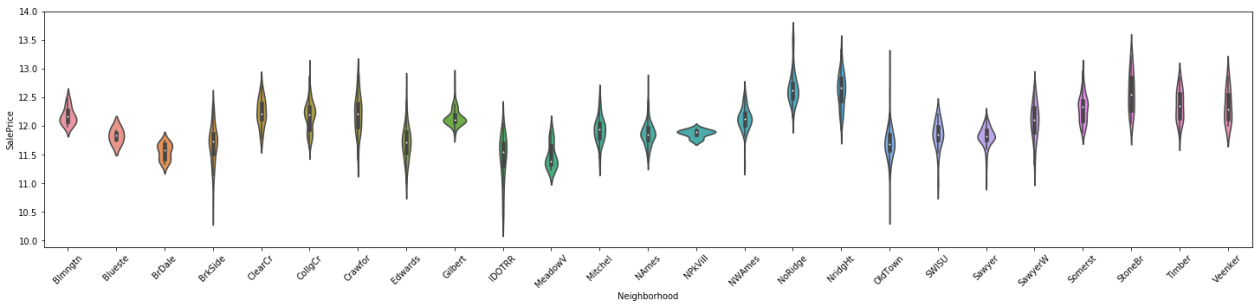
3.1 Distribution of SalePrice in Categorical Variables

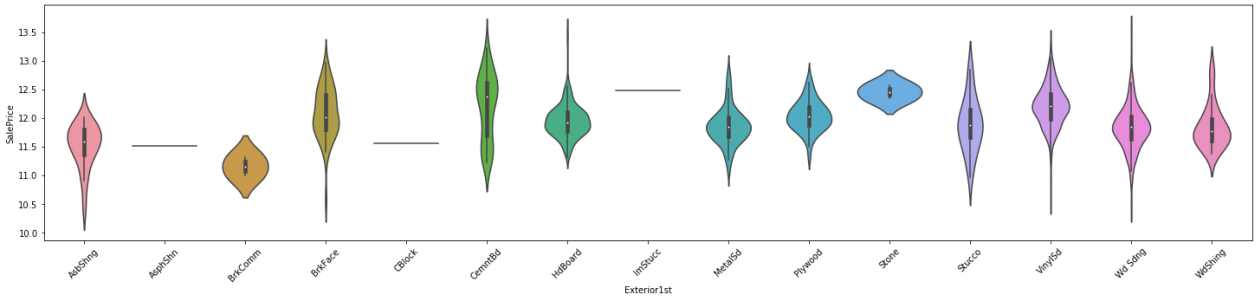
[60]
```
# plot categorical variables
fcols = 3
frows = ceil(len(cols_categ)/fcols)
plt.figure(figsize=(15,4*frows))

for i,col in enumerate(cols_categ):
    plt.subplot(frows,fcols,i+1)
    _ = sns.violinplot(df[col],df['SalePrice'])
```



[61]

```
#Neighbourhood
plt.figure(figsize=(25,5))
sns.violinplot(x='Neighborhood',y='SalePrice',data=df)
plt.xticks(rotation=45);

#Exterior1st
plt.figure(figsize=(25,5))
sns.violinplot(x='Exterior1st',y='SalePrice',data=df)
plt.xticks(rotation=45);
```



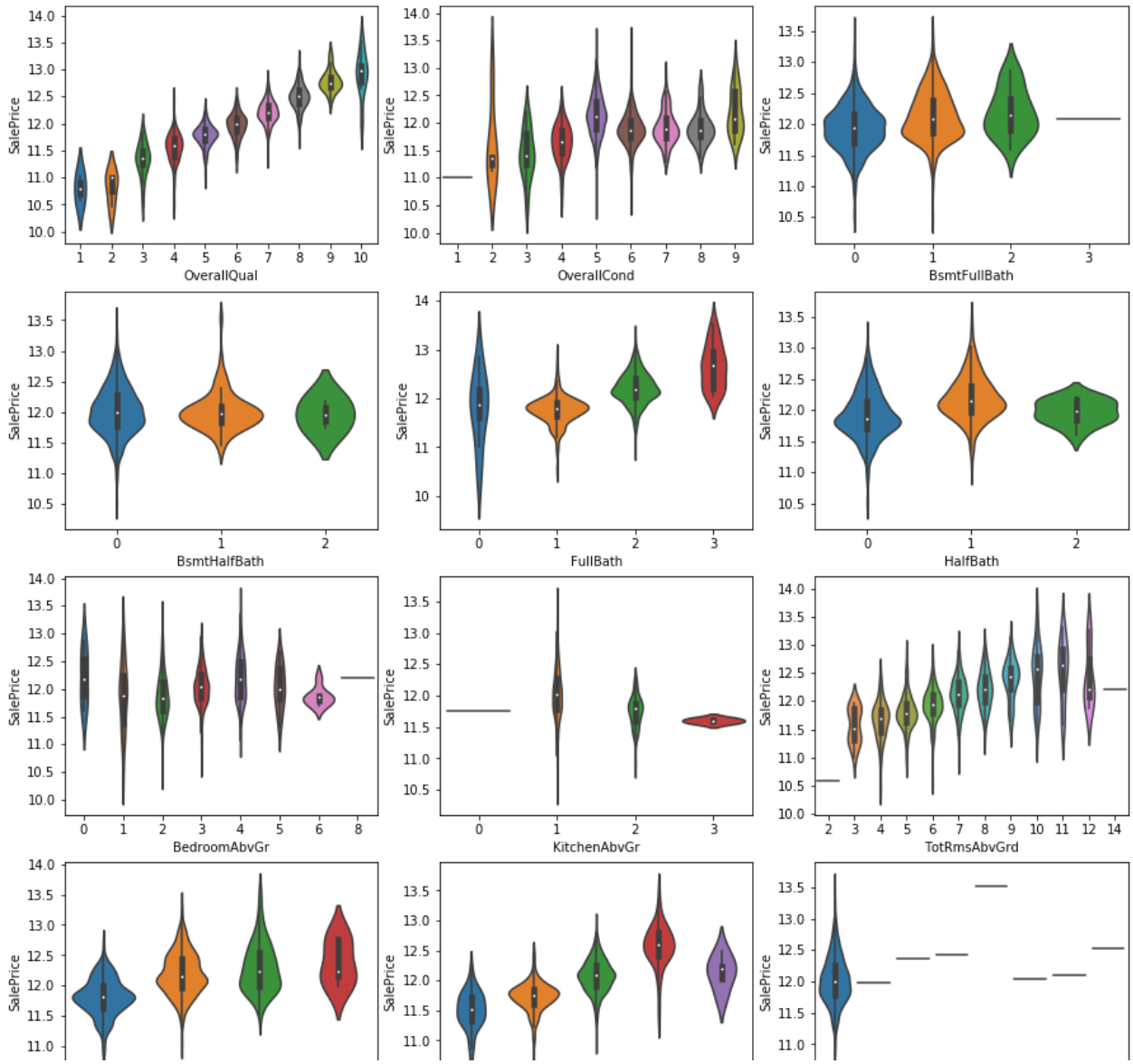


[62]
```
df.BsmtCond = df.BsmtCond.map({'Ex':5 ,'Gd':4 , 'TA':3 ,'Fa':2 ,'Po':1 ,
df.BsmtQual = df.BsmtQual.map({'Ex':5 ,'Gd':4 , 'TA':3 ,'Fa':2 ,'Po':1 ,
df.BsmtExposure = df.BsmtExposure.map({'Gd':4, 'Av':3, 'Mn':2, 'No':1, 'N
df.BsmtFinType1 = df.BsmtFinType1.map({'GLQ':6,'ALQ':5,'BLQ':4,'Rec':3,'L
df.BsmtFinType2 = df.BsmtFinType2.map({'GLQ':6,'ALQ':5,'BLQ':4,'Rec':3,'L

df.GarageType = df.GarageType.map({'2Types':4 , 'Attchd': 5, 'Basment':3
                                   'CarPort' :1, 'Detchd':2 , 'NoGara

df.GarageCond = df.GarageCond.map({'NoGarage':0, 'Po':1, 'Fa':2, 'TA':3,
df.GarageQual = df.GarageQual.map({'NoGarage':0, 'Po':1, 'Fa':2, 'TA':3,
df.GarageFinish = df.GarageFinish.map({'Fin':3, 'RFn':2, 'Unf':1, 'NoGara
df.PavedDrive = df.PavedDrive.map({'Y':2,'P':1, 'N':0 })
```

[63]
```
df.ExterCond = df.ExterCond.map({"Ex":4,'Gd':3,'TA':2,'Fa':1,'Po':0})
df.ExterQual = df.ExterQual.map({"Ex":4,'Gd':3,'TA':2,'Fa':1,'Po':0})

df.CentralAir = df.CentralAir.map({'Y':1, 'N':0})
df.HeatingQC = df.HeatingQC.map({"Ex":4,'Gd':3,'TA':2,'Fa':1,'Po':0})
df.FireplaceQu = df.FireplaceQu.map({"Ex":5,'Gd':4,'TA':3,'Fa':2,'Po':1,

df.KitchenQual = df.KitchenQual.map({"Ex":4,'Gd':3,'TA':2,'Fa':1,'Po':0})

df.PoolQC = df.PoolQC.map({"Ex":4,'Gd':3,'TA':2,'Fa':1,'NoPool':0})
df.Fence = df.Fence.map({'GdPrv':4 , 'MnPrv':3 , 'GdWo':2 , 'MnWw':1 , 'N
```

3.2 Distribution of SalePrice in Discrete Numeric Features

[64]
```
fcols = 3
frows = ceil(len(cols_discrete)/fcols)
plt.figure(figsize=(15,4*frows))

for i,col in enumerate(cols_discrete):
    plt.subplot(frows,fcols,i+1)
    sns.violinplot(df[col],df['SalePrice'])
```

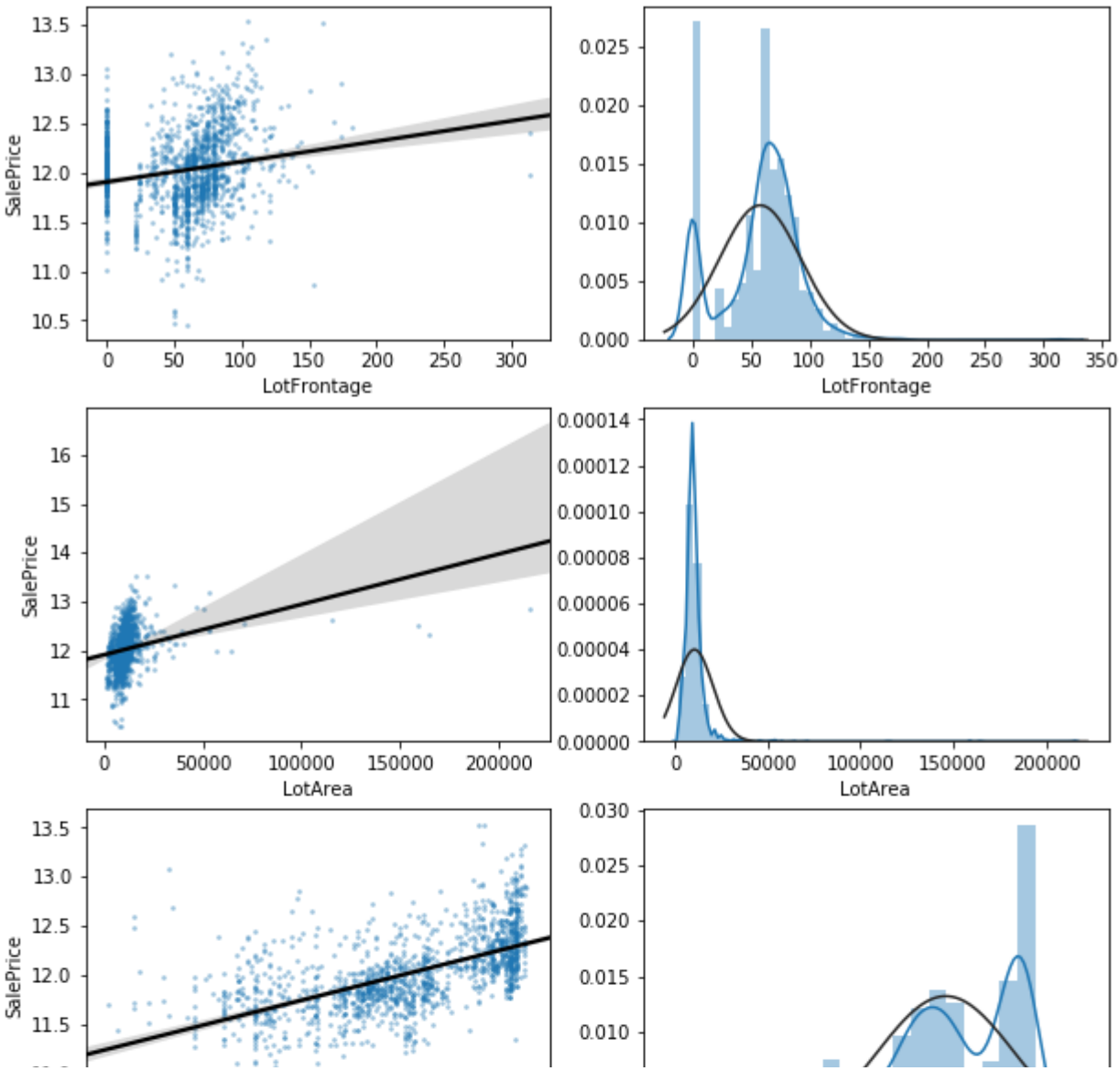


3.3 Distribution of Continuous Variables and Effect on SalePrice

[65]

```
fcols = 2
frows = len(cols_continuous)
plt.figure(figsize=(5*fcols,4*frows))

i=0
for col in cols_continuous:
    i+=1
    ax=plt.subplot(frows,fcols,i)
    sns.regplot(x=col, y='SalePrice', data=df, ax=ax,
                scatter_kws={'marker':'.','s':3,'alpha':0.3},
                line_kws={'color':'k'});
    plt.xlabel(col)
    plt.ylabel('SalePrice')

    i+=1
    ax=plt.subplot(frows,fcols,i)
    sns.distplot(df[col].dropna() , fit=stats.norm)
```

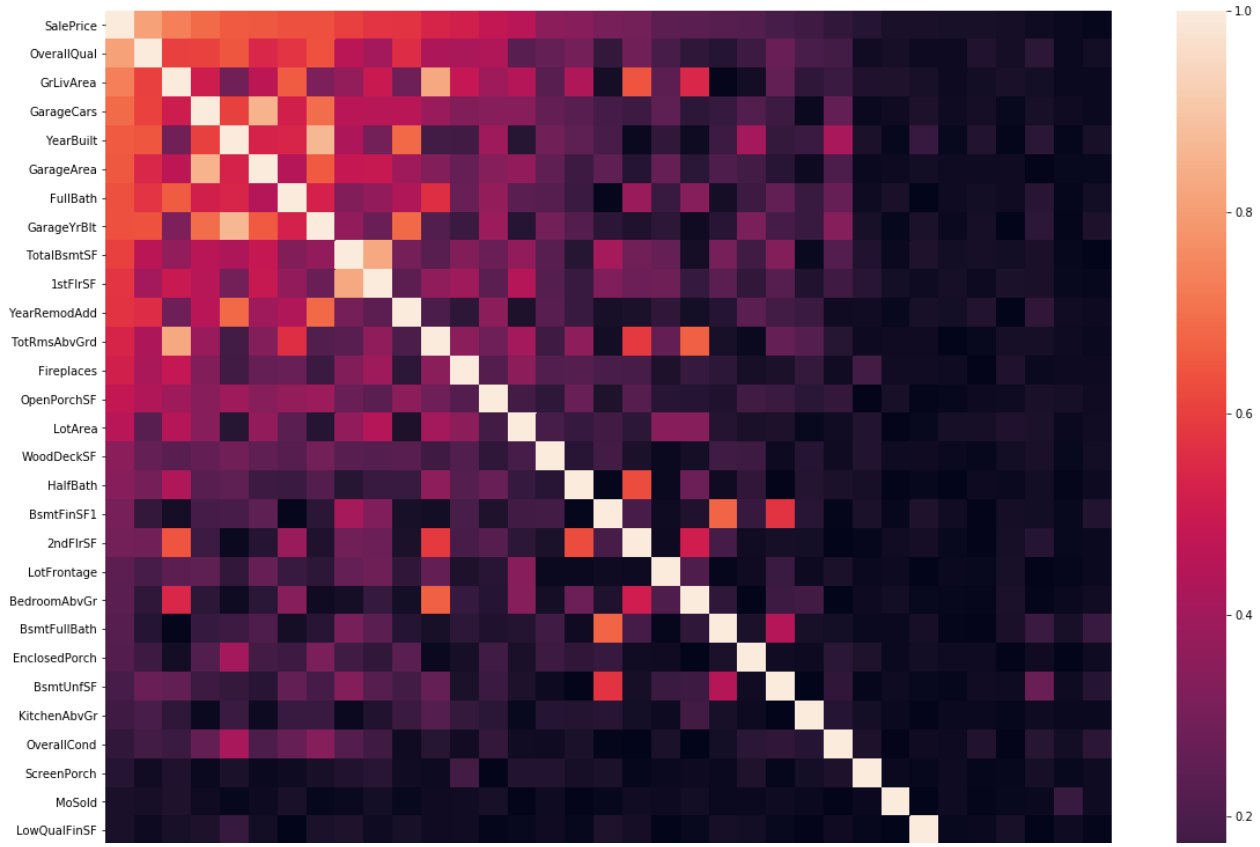
```
            plt.xlabel(col)
```



3.4 Correlation Between Numeric Features

[66]
```
# correlation between numeric variables
df_corr = df[cols_numeric].corr(method='spearman').abs()

# order columns and rows by correlation with SalePrice
df_corr = df_corr.sort_values('SalePrice',axis=0,ascending=False).sort_va

ax=plt.figure(figsize=(20,16)).gca()
sns.heatmap(df_corr,ax=ax,square=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a24f63e48>
```



[67]
```
df.info()
```

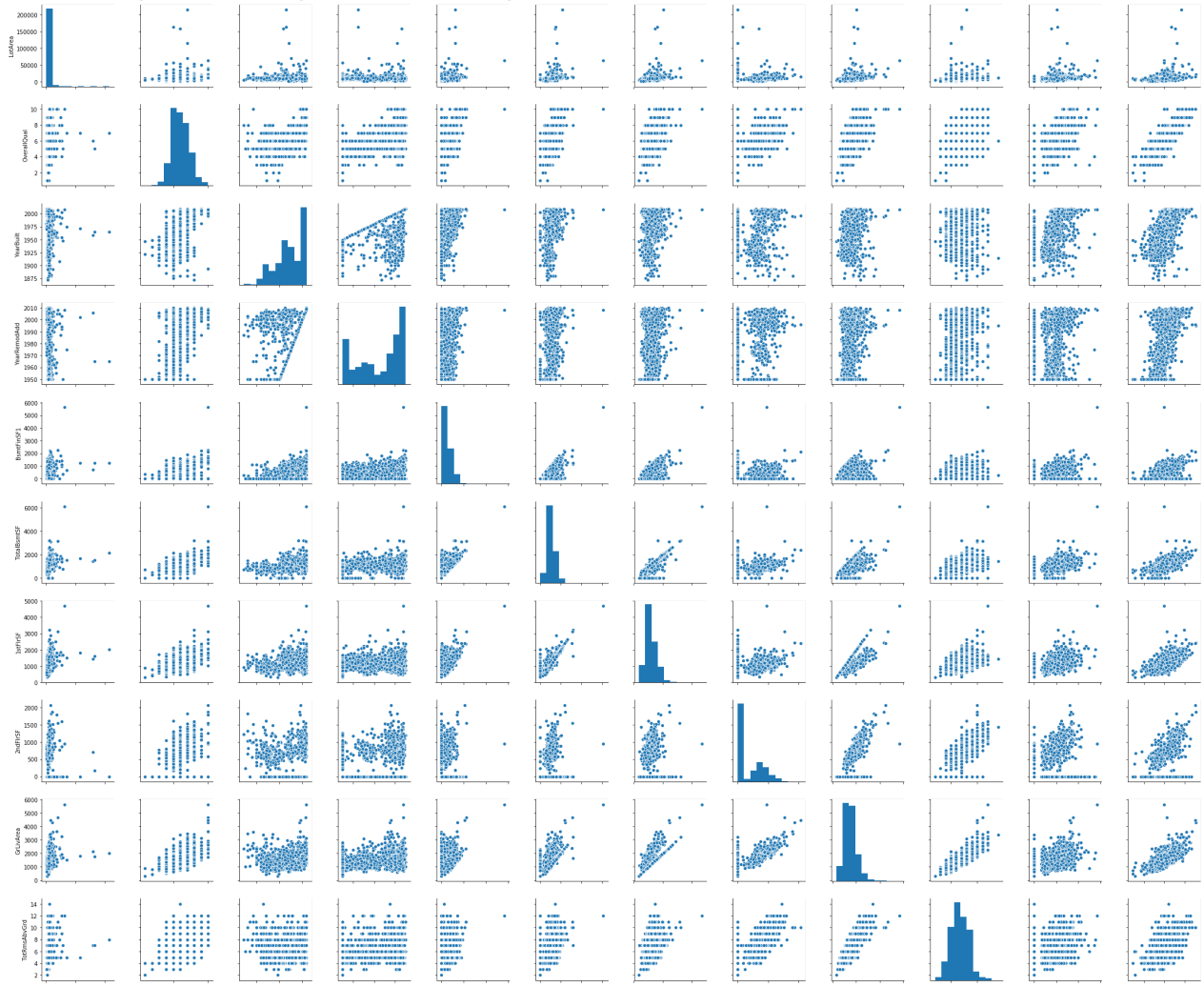
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 1 to 1460
Data columns (total 80 columns):
MSSubClass       1460 non-null category
MSZoning         1460 non-null category
LotFrontage      1460 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null category
Alley            1460 non-null category
LotShape         1460 non-null category
LandContour      1460 non-null category
Utilities        1460 non-null category
LotConfig        1460 non-null category
LandSlope        1460 non-null category
Neighborhood     1460 non-null category
Condition1       1460 non-null category
Condition2       1460 non-null category
BldgType         1460 non-null category
HouseStyle       1460 non-null category
OverallQual      1460 non-null int64
OverallCond      1460 non-null int64
YearBuilt        1460 non-null int64
YearRemodAdd     1460 non-null int64
RoofStyle        1460 non-null category
RoofMatl         1460 non-null category
Exterior1st      1460 non-null category
Exterior2nd      1460 non-null category
MasVnrType       1460 non-null category
MasVnrArea       1460 non-null category
ExterQual        1460 non-null int64
ExterCond        1460 non-null int64
```

```
Foundation       1460 non-null category
```

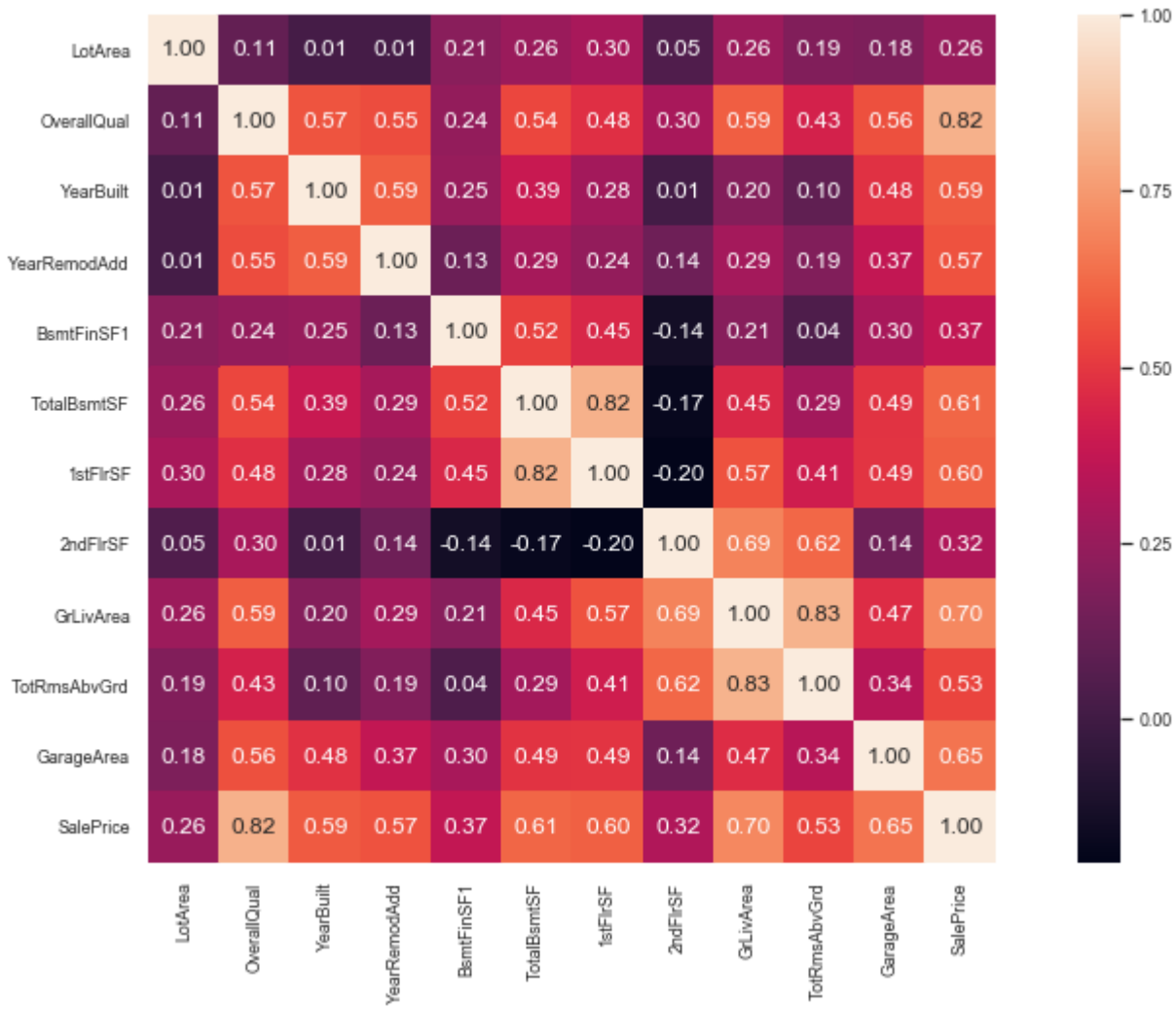
[68]
```
# cols = ['LotArea', 'OverallQual', 'YearBuilt', 'YearRemodAdd', 'MasVnrA
cols = ['LotArea', 'OverallQual', 'YearBuilt', 'YearRemodAdd', 'BsmtFinSF
```

[69]
```
sns.pairplot(df[cols], size=2.5)
plt.tight_layout()
plt.show()
```

```
/Users/changyaohua/anaconda3/lib/python3.7/site-
packages/seaborn/axisgrid.py:2065: UserWarning: The `size` parameter has
been renamed to `height`; pleaes update your code.
  warnings.warn(msg, UserWarning)
```



[70]
```
fig = plt.figure(figsize=(15,8))
cm = np.corrcoef(df[cols].values.T)
sns.set(font_scale=0.8)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot
plt.show()
```



[71]
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 1 to 1460
Data columns (total 80 columns):
MSSubClass       1460 non-null category
MSZoning         1460 non-null category
LotFrontage      1460 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null category
Alley            1460 non-null category
LotShape         1460 non-null category
LandContour      1460 non-null category
Utilities        1460 non-null category
LotConfig        1460 non-null category
LandSlope        1460 non-null category
Neighborhood     1460 non-null category
Condition1       1460 non-null category
Condition2       1460 non-null category
BldgType         1460 non-null category
HouseStyle       1460 non-null category
OverallQual      1460 non-null int64
OverallCond      1460 non-null int64
YearBuilt        1460 non-null int64
YearRemodAdd     1460 non-null int64
RoofStyle        1460 non-null category
```

```
RoofMatl         1460 non-null category
Exterior1st      1460 non-null category
Exterior2nd      1460 non-null category
MasVnrType       1460 non-null category
MasVnrArea       1460 non-null category
ExterQual        1460 non-null int64
ExterCond        1460 non-null int64
Foundation       1460 non-null category
```

3.5 Identify and Remove Outliers

[78]
```
from sklearn.metrics import make_scorer
from sklearn.linear_model import Ridge
```

[75]
```
# metric for evaluation
def rmse(y_true, y_pred):
    diff = y_pred - y_true
    sum_sq = sum(diff**2)
    n = len(y_pred)

    return np.sqrt(sum_sq/n)

# scorer to be used in sklearn model fitting
rmse_scorer = make_scorer(rmse, greater_is_better=False)
```

[76]
```
# function to detect outliers based on the predictions of a model
def find_outliers(model, X, y, sigma=3):

    # predict y values using model
    try:
        y_pred = pd.Series(model.predict(X), index=y.index)
    # if predicting fails, try fitting the model first
    except:
        model.fit(X,y)
        y_pred = pd.Series(model.predict(X), index=y.index)

    # calculate residuals between the model prediction and true y values
    resid = y - y_pred
    mean_resid = resid.mean()
    std_resid = resid.std()

    # calculate z statistic, define outliers to be where |z|>sigma
    z = (resid - mean_resid)/std_resid
    outliers = z[abs(z)>sigma].index

    # print and plot the results
    print('R2=',model.score(X,y))
    print('rmse=',rmse(y, y_pred))
    print('---------------------------------------')
```

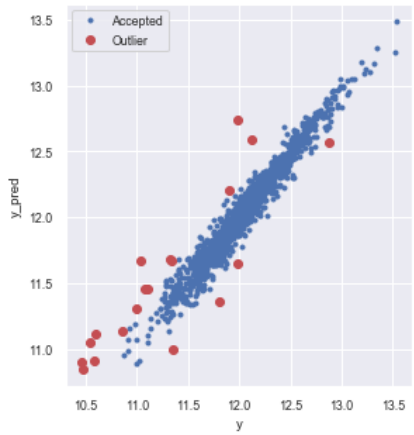
```
        print('mean of residuals:',mean_resid)
        print('std of residuals:',std_resid)
        print('---------------------------------------')

        print(len(outliers),'outliers:')
        print(outliers.tolist())

        plt.figure(figsize=(15,5))
        ax_131 = plt.subplot(1,3,1)
        plt.plot(y,y_pred,'.')
        plt.plot(y.loc[outliers],y_pred.loc[outliers],'ro')
        plt.legend(['Accepted','Outlier'])
        plt.xlabel('y')
        plt.ylabel('y_pred');

        ax_132=plt.subplot(1,3,2)
        plt.plot(y,y-y_pred,'.')
        plt.plot(y.loc[outliers],y.loc[outliers]-y_pred.loc[outliers],'ro')
        plt.legend(['Accepted','Outlier'])
        plt.xlabel('y')
        plt.ylabel('y - y_pred');

        ax_133=plt.subplot(1,3,3)
        z.plot.hist(bins=50,ax=ax_133)
        z.loc[outliers].plot.hist(color='r',bins=50,ax=ax_133)
        plt.legend(['Accepted','Outlier'])
        plt.xlabel('z')

        plt.savefig('outliers.png')

        return outliers
```

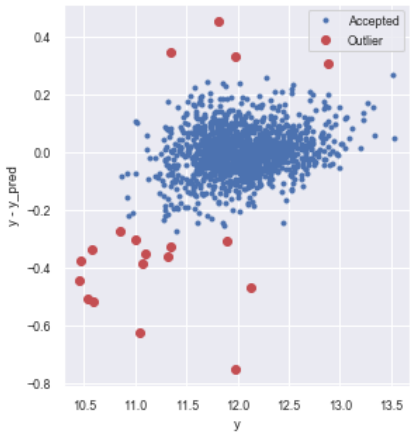
[84]
```
d_df = pd.get_dummies(df, drop_first= True)
d_df.info()
```

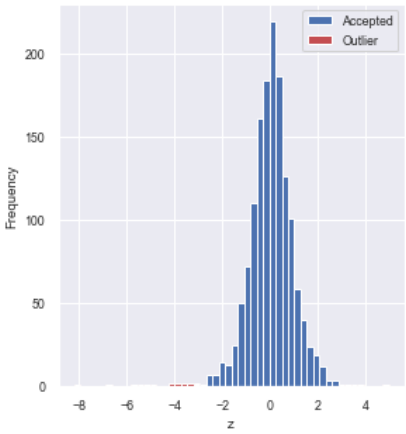
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 1 to 1460
Columns: 546 entries, LotFrontage to SaleCondition_Partial
dtypes: float64(3), int64(50), uint8(493)
memory usage: 1.3 MB
```

[85]
```
y = d_df.SalePrice
X = d_df.drop('SalePrice',axis=1)

# find and remove outliers using a Ridge model
outliers = find_outliers(Ridge(), X, y)

# permanently remove these outliers from the data
df_model = df.drop(outliers)
```

```
R2= 0.947325794892571
rmse= 0.09164624386294766
----------------------------------------
mean of residuals: -7.884104328297002e-16
std of residuals: 0.09167764569489989
----------------------------------------
19 outliers:
[31, 89, 411, 463, 496, 524, 534, 633, 682, 689, 711, 875, 917, 969, 971,
1299, 1325, 1433, 1454]
```







[86]
```
d_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 1 to 1460
Columns: 546 entries, LotFrontage to SaleCondition_Partial
dtypes: float64(3), int64(50), uint8(493)
memory usage: 1.3 MB
```

[87]
```
d_df.to_csv('./clean_data.csv')
```

[ ]

Week 2 Feature selection

May 23, 2019

0.1 Feature selection

In this section, we will use different techniques for feature selection such as random forest, RFECV, etc. We will try working on subset of features and also we'll try reducing the dimension of the dataaset usinn PCA and feed the results into one of the regression algorithms.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.feature_selection import RFECV, mutual_info_regression
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_absolute_error, mean_squared_error
import seaborn as sns
from heapq import nlargest
%matplotlib inline
```

In [2]:
```
data = pd.read_csv('../clean_data.csv', index_col=0)
data.head()
```

Out[2]:

Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	\
1	60	65.0	8450	7	5	2003	
2	20	80.0	9600	6	8	1976	
3	60	68.0	11250	7	5	2001	
4	70	60.0	9550	7	5	1915	
5	60	84.0	14260	8	5	2000	

Id	YearRemodAdd	ExterQual	ExterCond	BsmtQual	...	\
1	2003	3	2	4	...	
2	1976	2	2	4	...	

```
3           2002          3          2          4          ...
4           1970          2          2          3          ...
5           2000          3          2          4          ...

    SaleType_ConLI  SaleType_ConLw  SaleType_New  SaleType_Oth  SaleType_WD  \
Id
1                0               0             0             0            1
2                0               0             0             0            1
3                0               0             0             0            1
4                0               0             0             0            1
5                0               0             0             0            1

    SaleCondition_AdjLand  SaleCondition_Alloca  SaleCondition_Family  \
Id
1                       0                     0                     0
2                       0                     0                     0
3                       0                     0                     0
4                       0                     0                     0
5                       0                     0                     0

    SaleCondition_Normal  SaleCondition_Partial
Id
1                      1                     0
2                      1                     0
3                      1                     0
4                      0                     0
5                      1                     0

[5 rows x 533 columns]
```

```
In [3]: data.shape
```

```
Out[3]: (1459, 533)
```

0.1.1 As shown in EDA, the log(sale price) makes the sale price normally distributed without any outliers.

All the models will be fit on log(Sale Price)

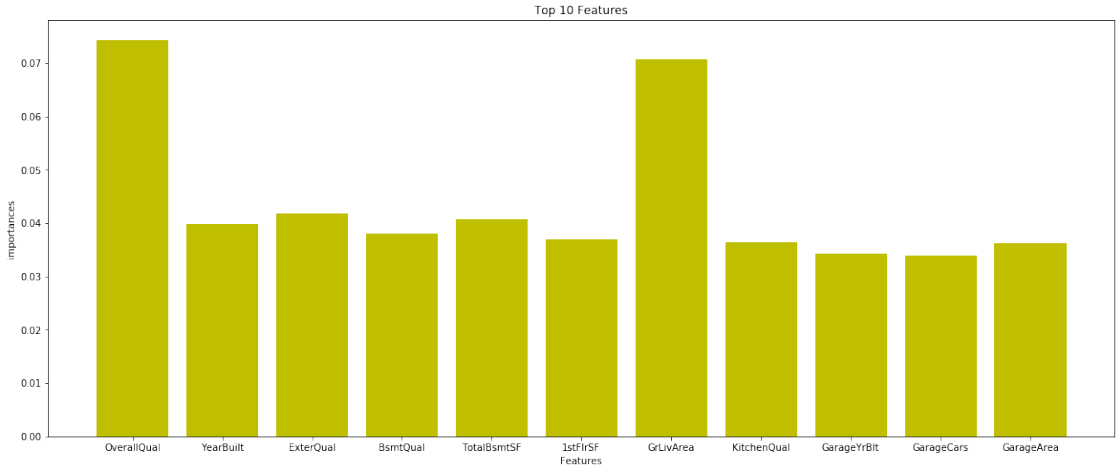
```
In [4]: y = np.log(data['SalePrice'])
        x = data.drop(labels = 'SalePrice', axis=1).astype("float64")
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state
```

```
In [5]: RF = RandomForestRegressor(n_estimators=1000, max_features=30, random_state=42)
        cv_score2 = cross_val_score(RF, x_train, y_train, cv=5, scoring='neg_mean_squared_erro
        print('Cross validation scores for GBR model:', np.sqrt(-cv_score2).mean())
```

```
Cross validation scores for GBR model: 0.15129308478316808
```

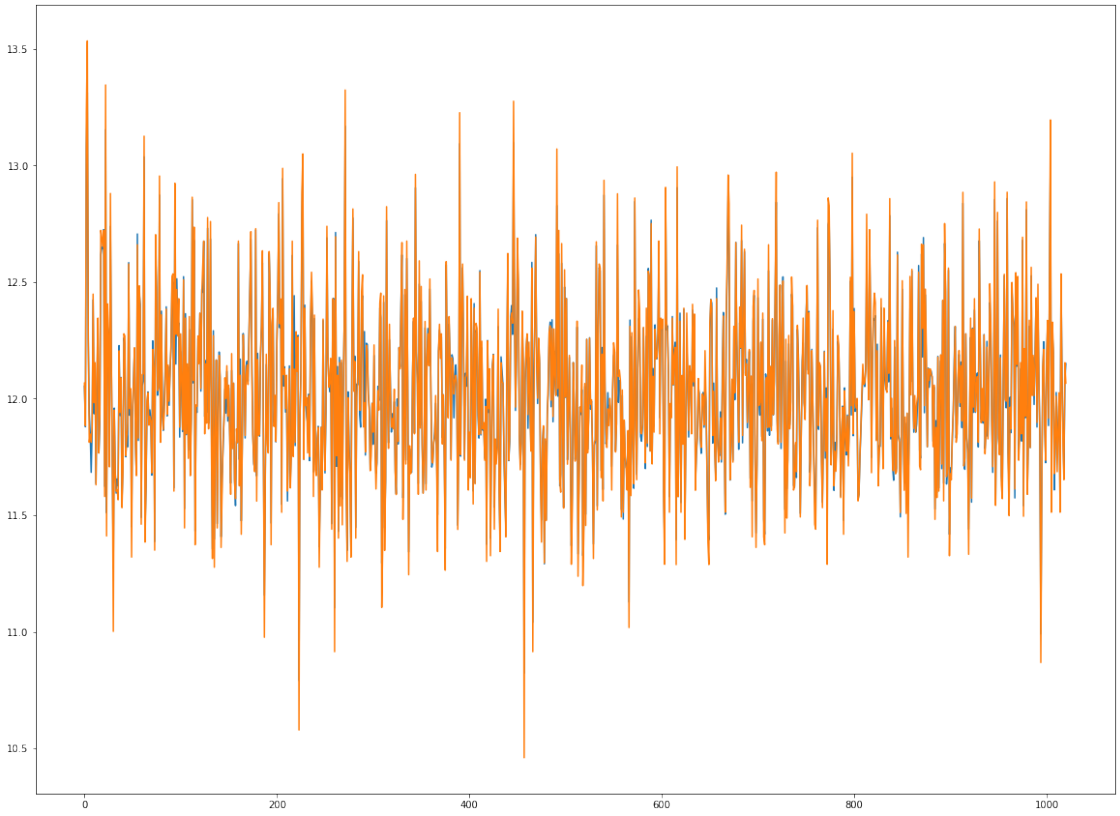
In [6]:
```
RF.fit(x_train, y_train)
y_pred_train = RF.predict(x_train)
```

In [7]:
```
imp_cols = x.columns[RF.feature_importances_ > 0.0328]
vals = RF.feature_importances_[RF.feature_importances_ > 0.0328]

plt.figure(figsize=(20,8))
plt.title("Top 10 Features")
plt.bar(imp_cols,vals,
       color="y",  align="center")
plt.xlabel('Features')
plt.ylabel('importances')
plt.show()
```

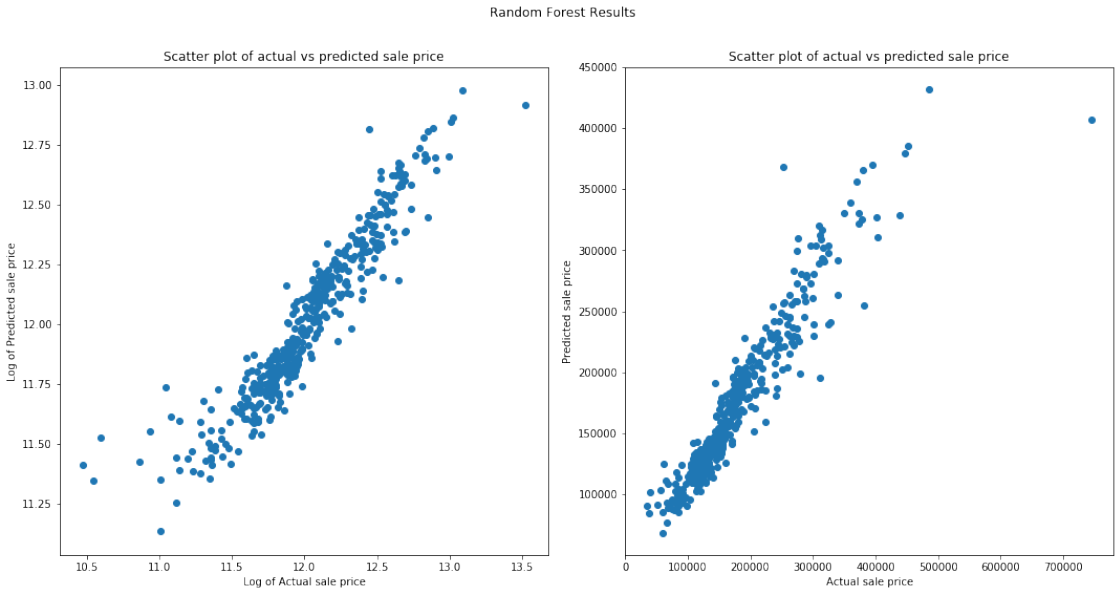


In [8]:
```
plt.subplots(figsize=(20,15))
plt.plot(y_pred_train)
plt.plot(y_train.values)
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x29236002a90>]
```



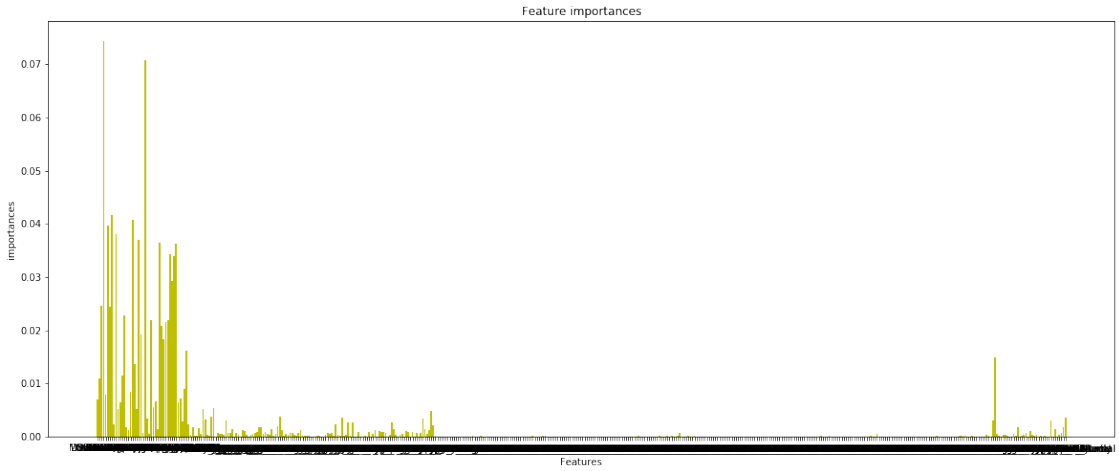

```
In [14]: y_pred = RF.predict(x_test)
         fig = plt.figure(figsize=(15,8))
         fig.suptitle('Random Forest Results')
         plt.subplot(121)
         plt.scatter(y_test, y_pred)
         plt.xlabel('Log of Actual sale price')
         plt.ylabel('Log of Predicted sale price')
         plt.title('Scatter plot of actual vs predicted sale price')
         plt.subplot(122)
         plt.scatter(np.exp(y_test.values), np.exp(y_pred))
         plt.xlabel('Actual sale price')
         plt.ylabel('Predicted sale price')
         plt.title('Scatter plot of actual vs predicted sale price')
         plt.tight_layout()
         fig.subplots_adjust(top=0.88)
```




```
In [15]: ## Calculate error with the actual values of sale price
         y_pred = RF.predict(x_test)

         MSEscore = np.sqrt(mean_squared_error(np.exp(y_pred), np.exp(y_test)))
         print('Score RMSE = {}'.format(MSEscore))

         MAEscore = np.sqrt(mean_absolute_error(np.exp(y_pred), np.exp(y_test)))
         print('Score RMAE = {}'.format(MAEscore))
```

```
Score RMSE = 29473.80466390691
Score RMAE = 130.2935307010514
```

```
In [16]: imp_cols = x.columns[RF.feature_importances_>0.0]
         vals = RF.feature_importances_[RF.feature_importances_>0.0]

         plt.figure(figsize=(20,8))
         plt.title("Feature importances")
         plt.bar(imp_cols,vals,
                 color="y",  align="center")
         plt.xlabel('Features')
         plt.ylabel('importances')
         plt.show()
```



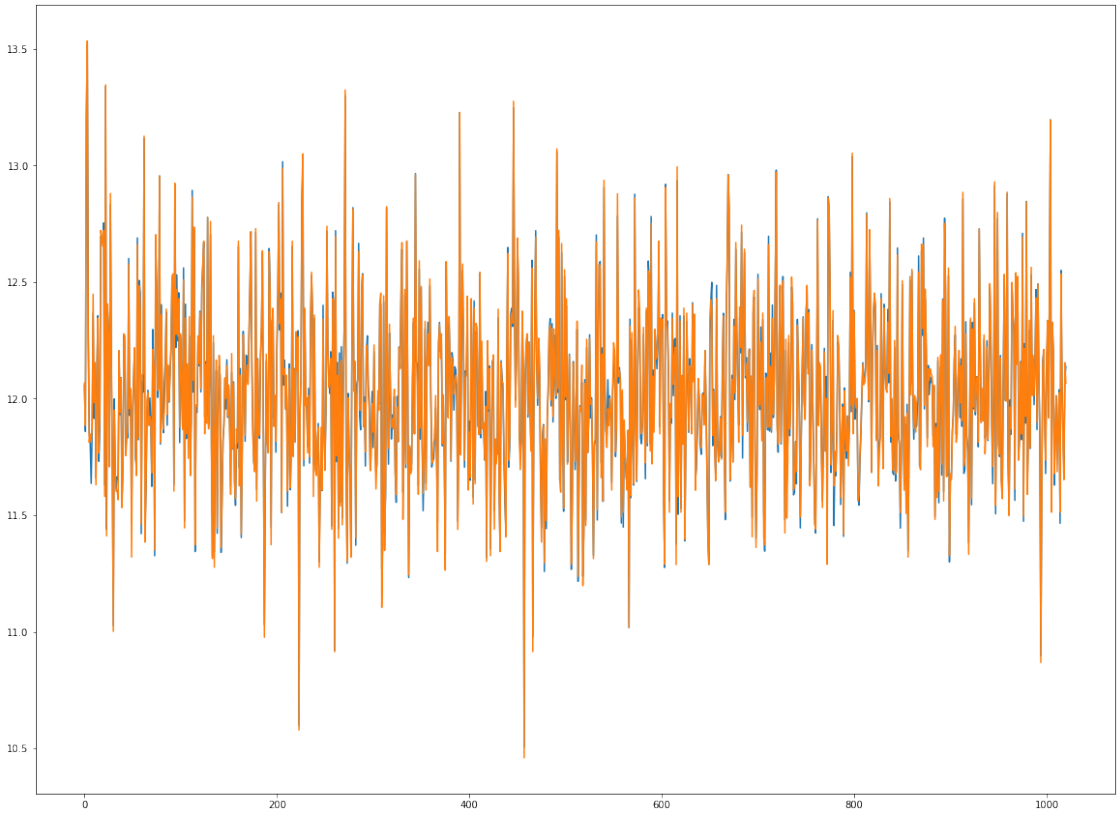
0.2 Feature Selection using GradientBoostingRegressor

```
In [17]: GBR = GradientBoostingRegressor(learning_rate= 0.1, n_estimators=1500, max_depth=2)
         cv_score = cross_val_score(GBR, x_train, y_train, cv=5, scoring='neg_mean_squared_erro
         print('Cross validation scores for GBR model:', np.sqrt(-cv_score).mean())
```

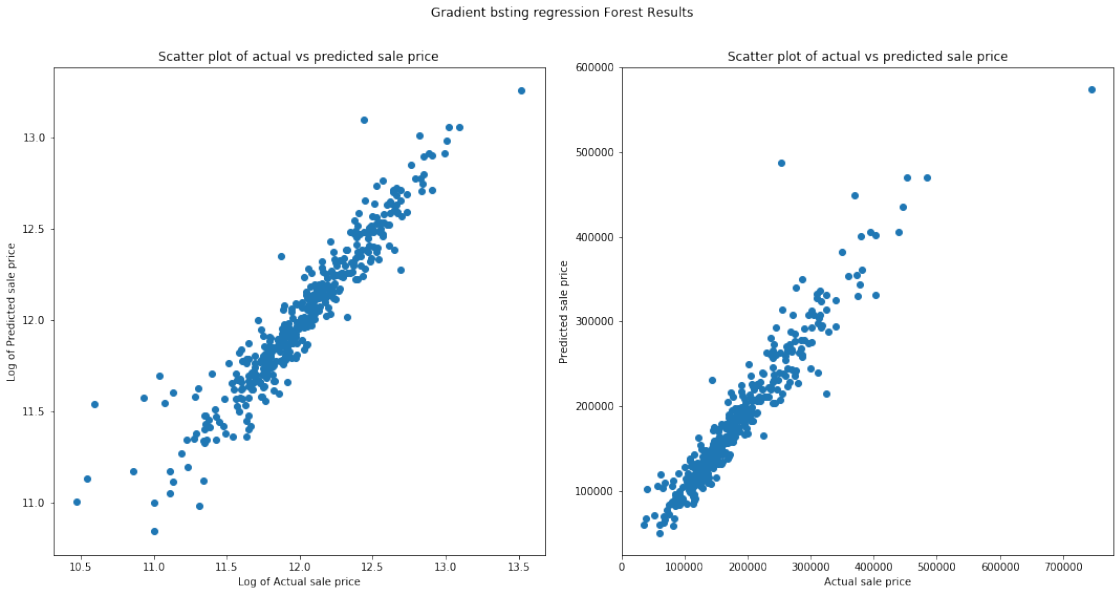
```
Cross validation scores for GBR model: 0.13754658935554992
```

```
In [18]: GBR.fit(x_train, y_train)
         y_pred_train = GBR.predict(x_train)
         plt.subplots(figsize=(20,15))
         plt.plot(y_pred_train)
         plt.plot(y_train.values)
```

```
Out[18]: [<matplotlib.lines.Line2D at 0x2923d193320>]
```



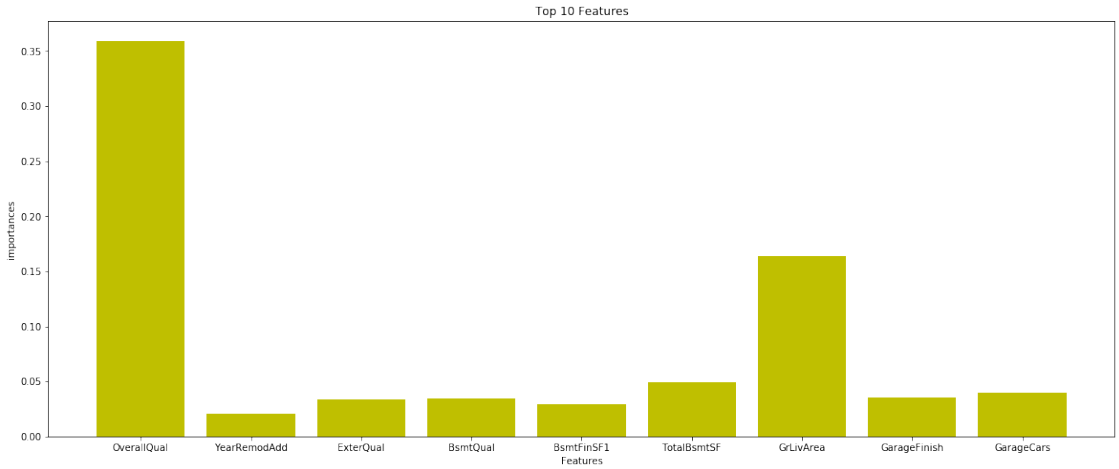

```
In [20]: y_pred = GBR.predict(x_test)
         fig = plt.figure(figsize=(15,8))
         fig.suptitle('Gradient bsting regression Forest Results')
         plt.subplot(121)
         plt.scatter(y_test, y_pred)
         plt.xlabel('Log of Actual sale price')
         plt.ylabel('Log of Predicted sale price')
         plt.title('Scatter plot of actual vs predicted sale price')
         plt.subplot(122)
         plt.scatter(np.exp(y_test.values), np.exp(y_pred))
         plt.xlabel('Actual sale price')
         plt.ylabel('Predicted sale price')
         plt.title('Scatter plot of actual vs predicted sale price')
         plt.tight_layout()
         fig.subplots_adjust(top=0.88)
```



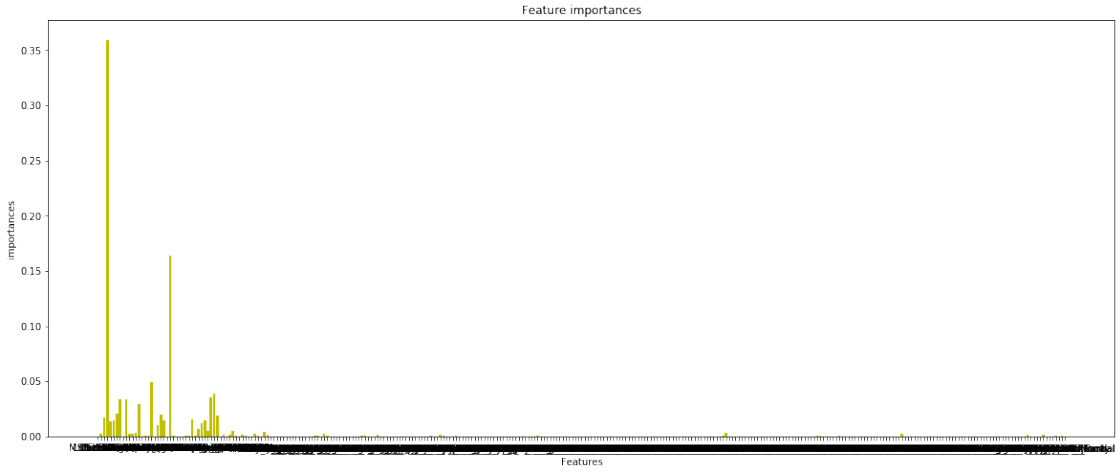

```
In [65]: ## Calculate error with the actual values of sale price
         y_pred = GBR.predict(x_test)

         MSEscore = np.sqrt(mean_squared_error(np.exp(y_pred), np.exp(y_test)))
         print('Score RMSE = {}'.format(MSEscore))

         MAEscore = np.sqrt(mean_absolute_error(np.exp(y_pred), np.exp(y_test)))
         print('Score RMAE = {}'.format(MAEscore))
```

```
Score RMSE = 24850.068530060966
Score RMAE = 123.20512665853853
```

```
In [33]: imp_cols = x.columns[GBR.feature_importances_ > 0.02]
         vals = GBR.feature_importances_[GBR.feature_importances_ > 0.02]

         plt.figure(figsize=(20,8))
         plt.title("Top 10 Features")
         plt.bar(imp_cols,vals,
                 color="y",  align="center")
         plt.xlabel('Features')
         plt.ylabel('importances')
         plt.show()
```




```
In [34]: imp_cols = x.columns[GBR.feature_importances_ > 0.0]
         vals = GBR.feature_importances_[GBR.feature_importances_ > 0.0]

         plt.figure(figsize=(20,8))
         plt.title("Feature importances")
         plt.bar(imp_cols,vals,
                 color="y",  align="center")
         plt.xlabel('Features')
         plt.ylabel('importances')
         plt.show()
```



Gradient boosting regressor filter out most features. Now, I will work only with a small number of features. GBR will be used to select the features.

```
In [35]: (GBR.feature_importances_>0.002).sum()
```

Week 3 SVR

May 23, 2019

0.1 Apply different Regression algorithms

0.1.1 SVM

```
In [24]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         from sklearn.preprocessing import StandardScaler
         from sklearn.svm import SVR
         from sklearn.pipeline import Pipeline
         from sklearn.model_selection import GridSearchCV
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import mean_absolute_error, r2_score
         import seaborn as sns
         %matplotlib inline
```

```
In [25]: data = pd.read_csv('reduced_var_data.csv', index_col = 0)
         y = data['SalePrice']
         x = data.drop(labels = 'SalePrice', axis=1)
```

- We will try first working on the actual value of the sale price, and then the log of sale price

```
In [3]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state
        ss = StandardScaler()
        ss.fit(x_train)
        x_train = ss.transform(x_train)
        x_test = ss.transform(x_test)
```

```
In [4]: pipe = Pipeline(steps= [('ss', StandardScaler()), ('clf', SVR(gamma='scale'))])
```

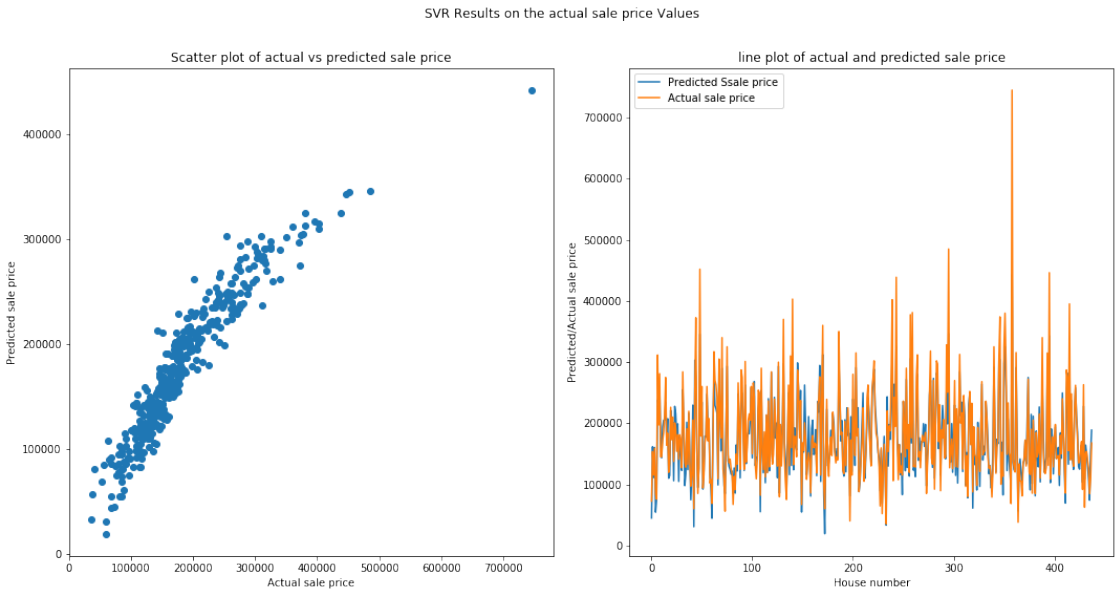
```
In [5]: param_grid = {
            'clf__C':[0.1, 0.5, 1.0, 1.5, 10,100, 150],
            'clf__kernel': ['linear', 'rbf', 'sigmoid', 'poly']
        }

        search = GridSearchCV(pipe, param_grid, cv=5, iid=False, scoring='neg_mean_absolute_er
                              return_train_score=False)
        search.fit(x, y)          # Here I am using the whole training data
        print("Best parameter (CV score=%0.3f):" % search.best_score_)
        print(search.best_params_)
```

```
Best parameter (CV score=-19736.160):
{'clf__C': 150, 'clf__kernel': 'linear'}
```

```
In [6]: best_svr = SVR(kernel='linear', gamma ='scale', C = 150)
        best_svr.fit(x_train, y_train)
        y_pred = best_svr.predict(x_test)
        print(mean_absolute_error(y_test, y_pred))
        print(r2_score(y_test, y_pred))
```

```
18799.026333339327
0.8607851431235201
```

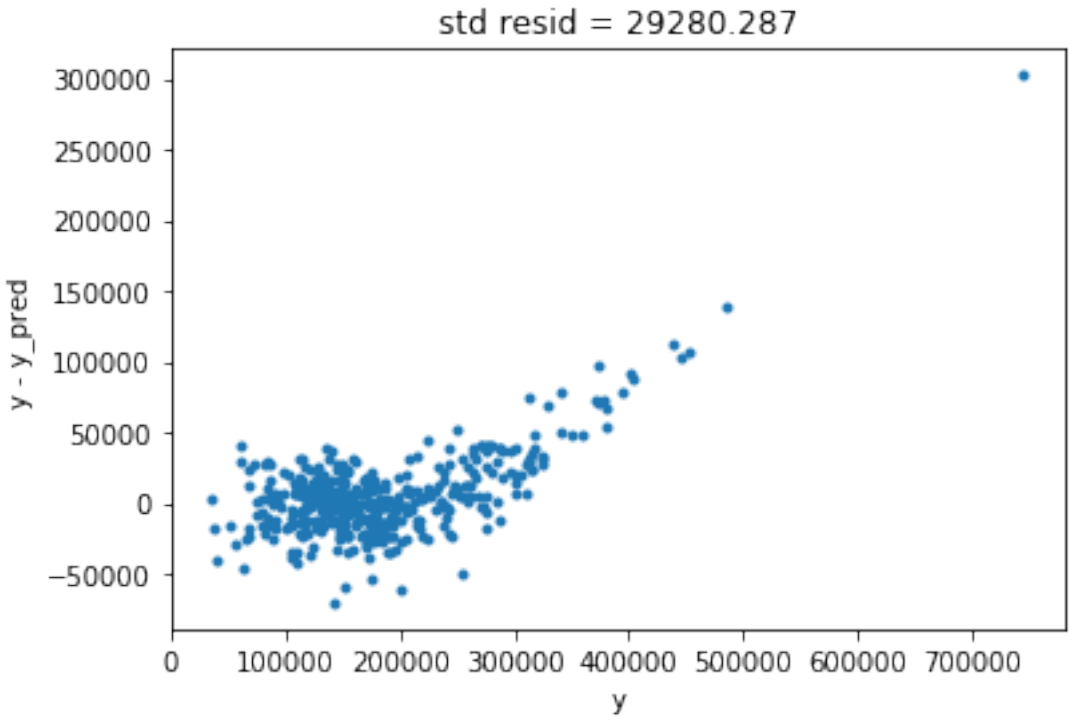
```
In [10]: y_pred = best_svr.predict(x_test)
         fig = plt.figure(figsize=(15,8))
         fig.suptitle('SVR Results on the actual sale price Values')
         plt.subplot(121)
         plt.scatter(y_test.values, y_pred)
         plt.xlabel('Actual sale price')
         plt.ylabel('Predicted sale price')
         plt.title('Scatter plot of actual vs predicted sale price')
         plt.subplot(122)
         plt.plot((y_pred), label='Predicted Ssale price')
         plt.plot((y_test.values), label='Actual sale price')
         plt.xlabel('House number')
         plt.ylabel('Predicted/Actual sale price')
         plt.title('line plot of actual and predicted sale price')
         plt.legend()
         plt.tight_layout()
         fig.subplots_adjust(top=0.88)
```



```
In [12]: print("Corrolation between true and predicted value using SVR on the actual sale price
               format(np.corrcoef(y_test,y_pred)[0][1]))
```

```
Corrolation between true and predicted value using SVR on the actual sale price is 0.937710614
```

```
In [13]: resid = y_test - y_pred
         mean_resid = resid.mean()
         std_resid = resid.std()
         plt.plot(y_test,y_test-y_pred,'.')
         plt.xlabel('y')
         plt.ylabel('y - y_pred');
         plt.title('std resid = {:.3f}'.format(std_resid));
```



As shown in the figures above, There are some outliers which increase the value of the RMSE and the std of the residual.

0.1.2 Try working on log(Sale price)

```
In [14]: x_train, x_test, y_train, y_test = train_test_split(x, np.log(y), test_size = 0.3, ra
         ss = StandardScaler()
         ss.fit(x_train)
```

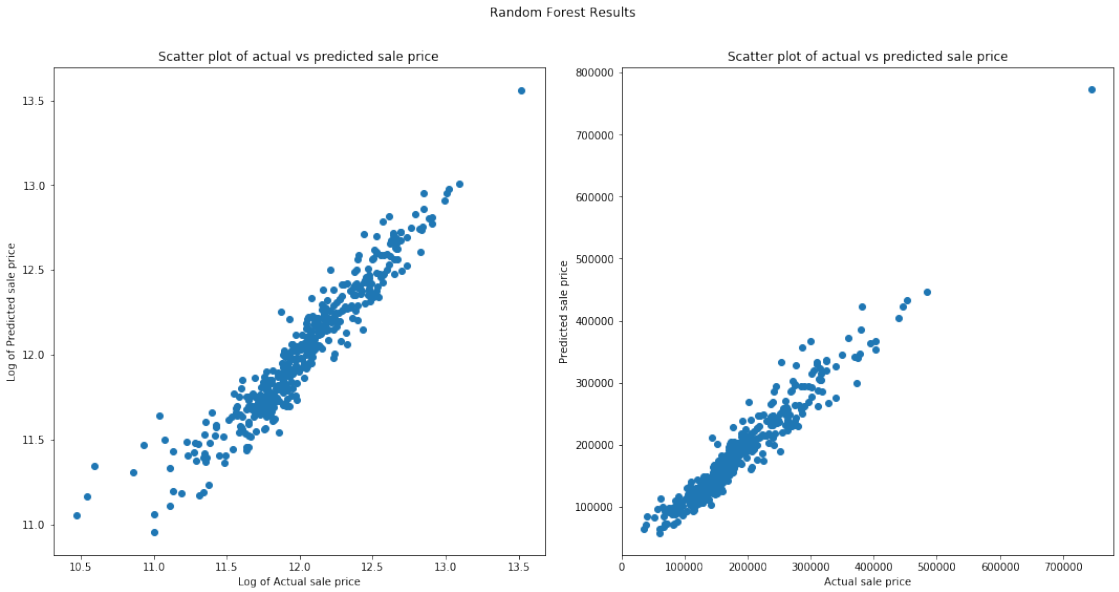
```
        x_train = ss.transform(x_train)
        x_test = ss.transform(x_test)
        search.fit(x, np.log(y))          # Here I am using the whole training data
        print("Best parameter (CV score=%0.3f):" % search.best_score_)
        print(search.best_params_)
```

```
Best parameter (CV score=-0.092):
{'clf__C': 0.1, 'clf__kernel': 'linear'}
```

```
In [19]: best_svr = SVR(kernel='linear', gamma ='scale', C = 0.1)
         best_svr.fit(x_train, y_train)
         y_pred = best_svr.predict(x_test)
         print(mean_absolute_error(np.exp(y_test), np.exp(y_pred)))
         print(r2_score(np.exp(y_test), np.exp(y_pred)))
```

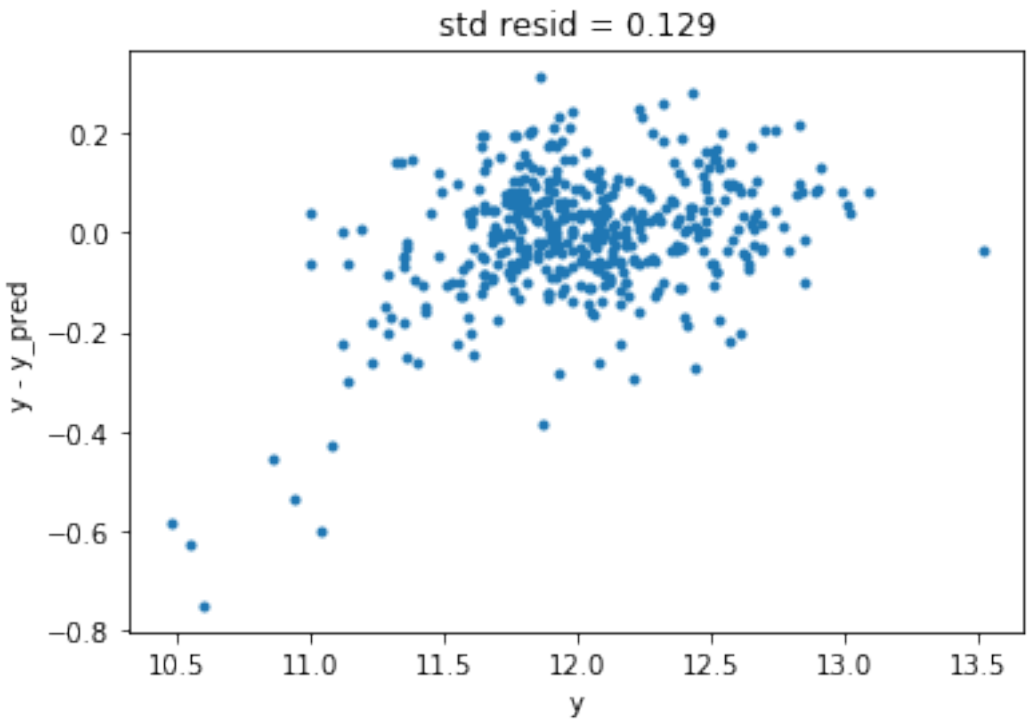
```
15236.845896887477
0.93274653988136
```

```
In [17]: y_pred = best_svr.predict(x_test)
         fig = plt.figure(figsize=(15,8))
         fig.suptitle('Random Forest Results')
         plt.subplot(121)
         plt.scatter(y_test, y_pred)
         plt.xlabel('Log of Actual sale price')
         plt.ylabel('Log of Predicted sale price')
         plt.title('Scatter plot of actual vs predicted sale price')
         plt.subplot(122)
         plt.scatter(np.exp(y_test.values), np.exp(y_pred))
         plt.xlabel('Actual sale price')
         plt.ylabel('Predicted sale price')
         plt.title('Scatter plot of actual vs predicted sale price')
         plt.tight_layout()
         fig.subplots_adjust(top=0.88)
```



SVM algorithm with log of sale price gives a very good results

```
In [21]: resid = y_test - y_pred
         mean_resid = resid.mean()
         std_resid = resid.std()
         plt.plot(y_test,y_test-y_pred,'.')
         plt.xlabel('y')
         plt.ylabel('y - y_pred');
         plt.title('std resid = {:.3f}'.format(std_resid));
```




```
In [23]: print("Corrolation between true and predicted value using SVR on the log of sale price
                format(np.corrcoef(np.exp(y_test),np.exp(y_pred))[0][1]))
```

```
Corrolation between true and predicted value using SVR on the log of sale price is 0.965901765
```

Working on the actual value of the sale price is better than working on log od the sale price

Week 04 NN

May 24, 2019

0.0.1 Neural Networks

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import seaborn as sns
%matplotlib inline
```

In [3]:
```
data = pd.read_csv('reduced_var_data.csv', index_col = 0)
y = data['SalePrice']
x = data.drop(labels = 'SalePrice', axis=1)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state

ss = StandardScaler()
ss.fit(x_train)
x_train = ss.transform(x_train)
x_test = ss.transform(x_test)
```

As usual, I will start ** Training on the actual Sale price **

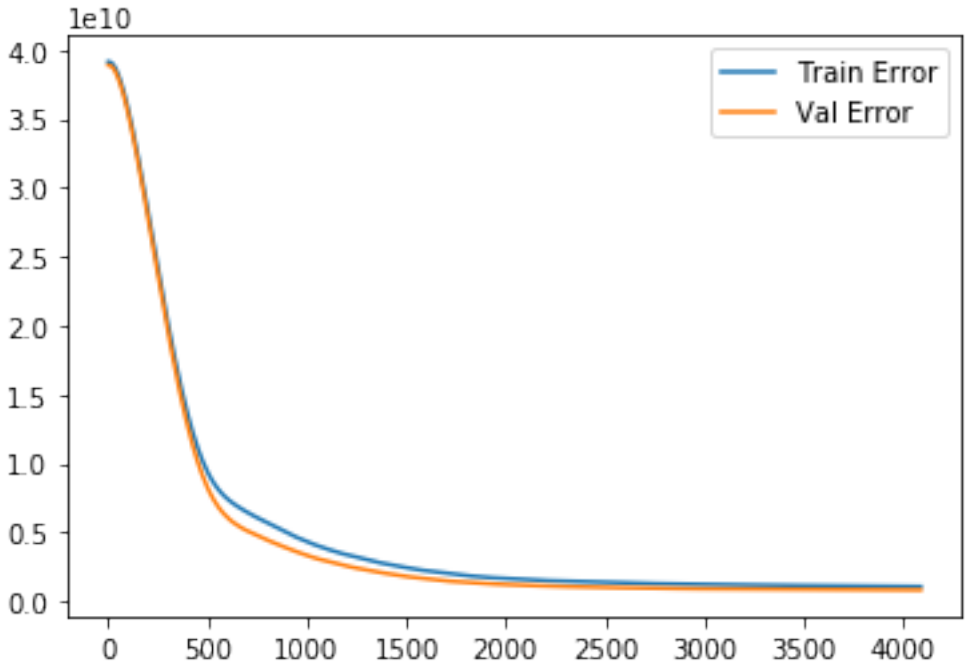
In [4]:
```
seed = 42
np.random.seed(seed)

n_feat = x.shape[1]
mdl = Sequential()
mdl.add(Dense(units=256, input_dim = n_feat, activation = 'relu'))
mdl.add(Dense(units=1, activation='linear'))
```

```
        mdl.compile(loss='mean_squared_error', optimizer='adam', metrics= ['mse', 'mae'])
        monitor = EarlyStopping(monitor= 'val_loss', min_delta=1e-3,
                                patience = 10, verbose=1, mode = 'auto')
        history = mdl.fit(x_train, y_train, validation_data = (x_test, y_test),
                          callbacks=[monitor],batch_size= 64,epochs=5000,verbose=0)

        y_pred = mdl.predict(x_test)
        MSEscore = (mean_squared_error(y_pred, y_test))
        print('Score MSE = {}'.format(MSEscore))

        MAEscore = (mean_absolute_error(y_pred, y_test))
        print('Score MAE = {}'.format(MAEscore))
        print('R2_score = {}'.format(r2_score(y_test, y_pred)))
        mdl.summary()
```

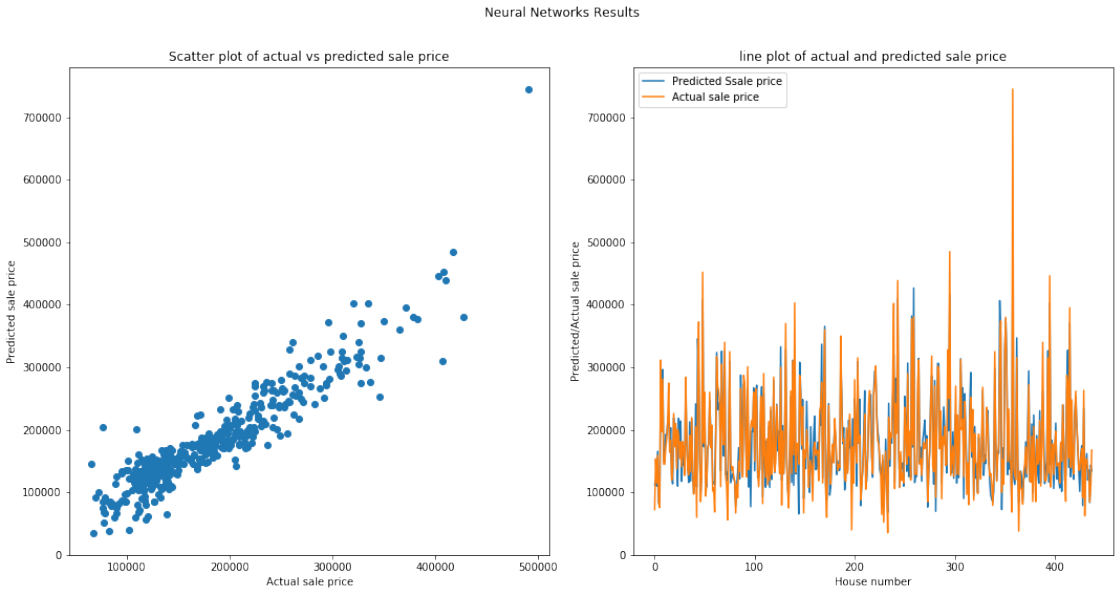
```
Epoch 04090: early stopping
Score MSE = 830356207.3640757
Score MAE = 19768.238566638127
R2_score = 0.8682731094023859
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               8192
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 8,449
Trainable params: 8,449
Non-trainable params: 0
_________________________________________________________________
```

```
In [5]: plt.figure()
        plt.plot(history.epoch, history.history['mean_squared_error'], label = 'Train Error')
        plt.plot(history.epoch, history.history['val_mean_squared_error'], label = 'Val Error')
        plt.legend()
```

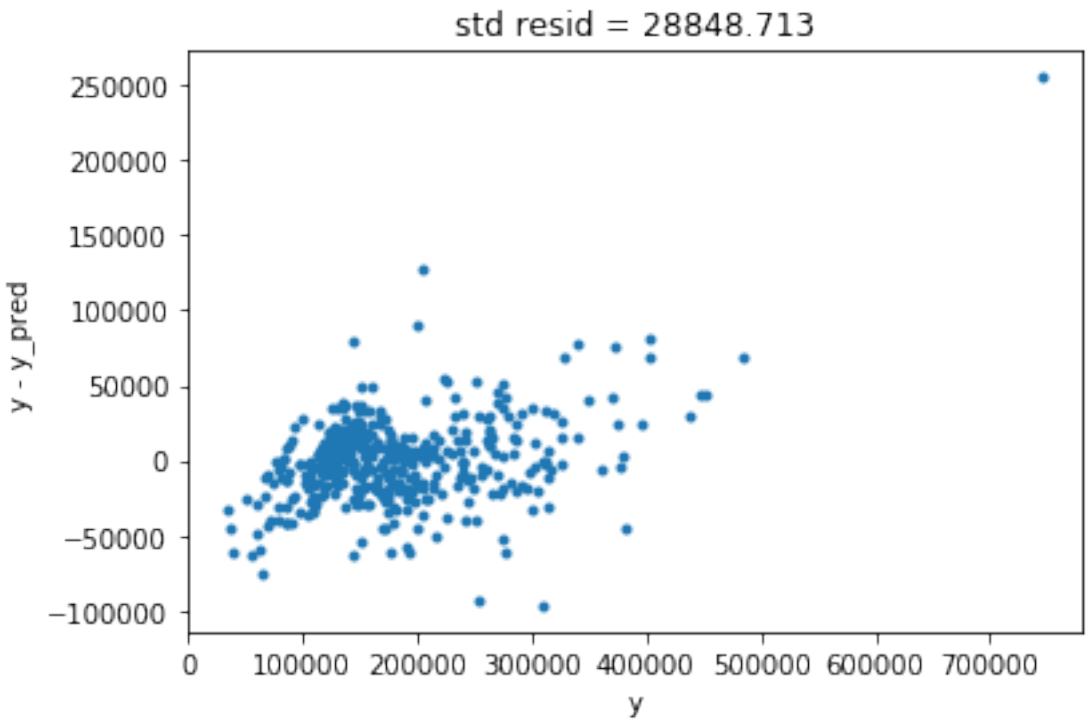
```
Out[5]: <matplotlib.legend.Legend at 0x212201894e0>
```



In [6]:
```
y_pred = mdl.predict(x_test)
fig = plt.figure(figsize=(15,8))
fig.suptitle('Neural Networks Results')
plt.subplot(121)
plt.scatter((y_pred), (y_test))
plt.xlabel('Actual sale price')
plt.ylabel('Predicted sale price')
plt.title('Scatter plot of actual vs predicted sale price')
plt.subplot(122)
plt.plot((y_pred), label='Predicted Ssale price')
plt.plot((y_test.values), label='Actual sale price')
plt.xlabel('House number')
plt.ylabel('Predicted/Actual sale price')
plt.title('line plot of actual and predicted sale price')
plt.legend()
plt.tight_layout()
fig.subplots_adjust(top=0.88)
```




```
In [7]: y_test.shape, y_pred.shape

Out[7]: ((438,), (438, 1))

In [8]: resid = y_test - y_pred.squeeze()
        mean_resid = resid.mean()
        std_resid = resid.std()
        plt.plot(y_test,resid,'.')
        plt.xlabel('y')
        plt.ylabel('y - y_pred');
        plt.title('std resid = {:.3f}'.format(std_resid));
```




```
In [9]: print("Corrolation between true and predicted value using NN on the actual sale price i
              format(np.corrcoef(y_test,y_pred.squeeze())[0][1]))
```

```
Corrolation between true and predicted value using NN on the actual sale price is 0.9318429100
```

0.1 Train on log of the sale price

```
In [22]: x_train, x_test, y_train, y_test = train_test_split(x, np.log(y), test_size = 0.3, ra
```

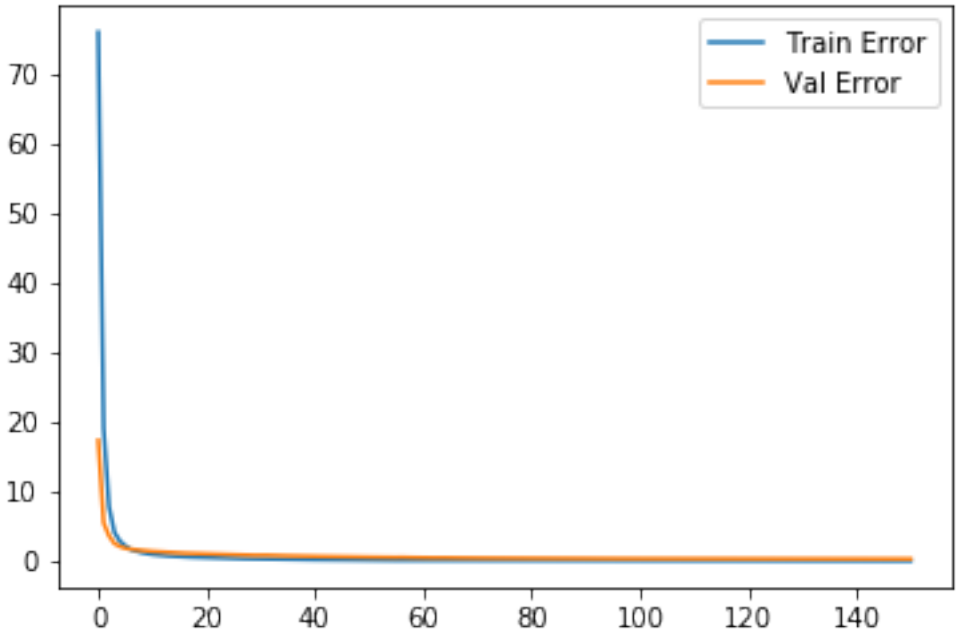
```
ss = StandardScaler()
ss.fit(x_train)
x_train = ss.transform(x_train)
x_test = ss.transform(x_test)


seed = 42
np.random.seed(seed)

n_feat = x.shape[1]
mdl = Sequential()
mdl.add(Dense(units=256, input_dim = n_feat, activation = 'relu'))
mdl.add(Dense(units=256, input_dim = n_feat, activation = 'relu'))
mdl.add(Dense(units=1, activation='linear'))
```

```
        mdl.compile(loss='mean_squared_error', optimizer='adam', metrics= ['mse', 'mae'])
        monitor = EarlyStopping(monitor= 'val_loss', min_delta=1e-3,
                                patience = 10, verbose=1, mode = 'auto')
        history = mdl.fit(x_train, y_train, validation_data = (x_test, y_test),
                          callbacks=[monitor],batch_size= 64,epochs=5000,verbose=0)

        y_pred = mdl.predict(x_test)
        MSEscore = (mean_squared_error(np.exp(y_pred), np.exp(y_test)))
        print('Score MSE = {}'.format(MSEscore))

        MAEscore = (mean_absolute_error(np.exp(y_pred), np.exp(y_test)))
        print('Score MAE = {}'.format(MAEscore))

        print('R2_score = {}'.format(r2_score(np.exp(y_pred), np.exp(y_test))))
        mdl.summary()
```

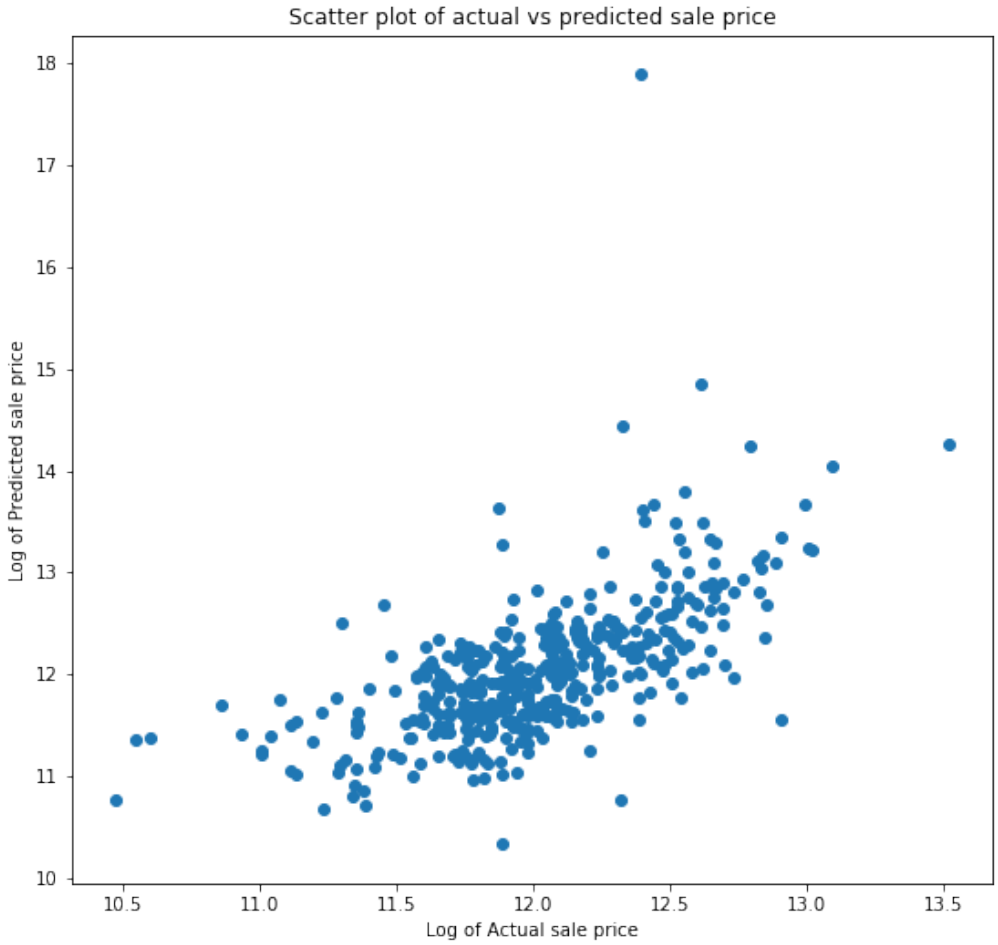
```
Epoch 00151: early stopping
Score MSE = 7890035460598.446
Score MAE = 211136.2905429509
R2_score = 0.00039785770222766637
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 256)               8192
_________________________________________________________________
dense_9 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 257
=================================================================
Total params: 74,241
Trainable params: 74,241
Non-trainable params: 0
_________________________________________________________________
```

```
In [23]: plt.figure()
         plt.plot(history.epoch, history.history['mean_squared_error'], label = 'Train Error')
         plt.plot(history.epoch, history.history['val_mean_squared_error'], label = 'Val Error
         plt.legend()
```

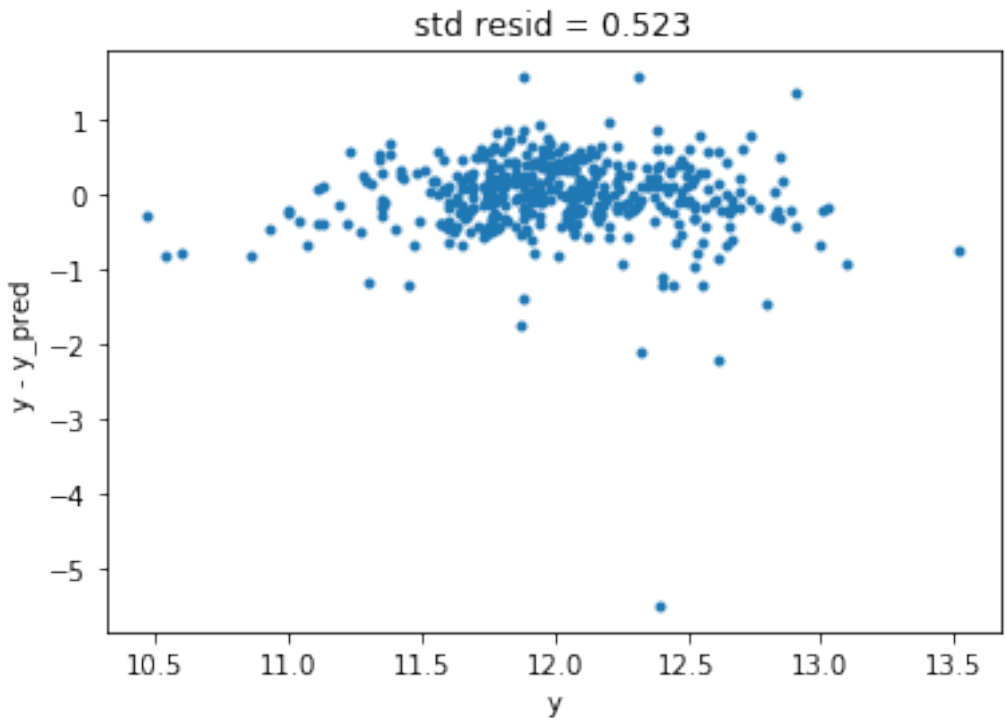
```
Out[23]: <matplotlib.legend.Legend at 0x2122377b940>
```



In [31]:
```
y_pred = mdl.predict(x_test)
fig = plt.figure(figsize=(15,8))
fig.suptitle('Neural Networks Results')
plt.subplot(121)
plt.scatter(y_test, y_pred)
plt.xlabel('Log of Actual sale price')
plt.ylabel('Log of Predicted sale price')
plt.title('Scatter plot of actual vs predicted sale price')


plt.tight_layout()
fig.subplots_adjust(top=0.88)
```




```
In [25]: y_test.shape, y_pred.shape

Out[25]: ((438,), (438, 1))

In [26]: resid = y_test - y_pred.squeeze()
         mean_resid = resid.mean()
         std_resid = resid.std()
         plt.plot(y_test,resid,'.')
         plt.xlabel('y')
         plt.ylabel('y - y_pred');
         plt.title('std resid = {:.3f}'.format(std_resid));
```




```
In [27]: print("Corrolation between true and predicted value using NN on the log of sale price
                format(np.corrcoef(y_test,y_pred.squeeze())[0][1]))
```

```
Corrolation between true and predicted value using NN on the log of sale price is 0.64048579316
```

Working on the Actual sale price value gives better results than working on the log of sale price

** Neural Neworks gives a bad results compared to SVM. **

```
Out[35]: 31

In [36]: x.loc[:,GBR.feature_importances_>0.002].columns

Out[36]: Index(['LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt',
                'YearRemodAdd', 'ExterQual', 'BsmtQual', 'BsmtCond', 'BsmtExposure',
                'BsmtFinType1', 'BsmtFinSF1', 'TotalBsmtSF', 'CentralAir', '1stFlrSF',
                '2ndFlrSF', 'GrLivArea', 'KitchenQual', 'Fireplaces', 'FireplaceQu',
                'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea',
                'OpenPorchSF', 'MoSold', 'MSZoning_RL', 'Neighborhood_Crawfor',
                'MasVnrArea_209.0', 'MasVnrArea_428.0'],
               dtype='object')

In [21]: (RF.feature_importances_>0.01).sum()

Out[21]: 26

In [23]: x.loc[:,RF.feature_importances_>0.01].columns

Out[23]: Index(['LotFrontage', 'LotArea', 'OverallQual', 'YearBuilt', 'YearRemodAdd',
                'ExterQual', 'BsmtQual', 'BsmtFinType1', 'BsmtFinSF1', 'TotalBsmtSF',
                'HeatingQC', '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'FullBath',
                'KitchenQual', 'TotRmsAbvGrd', 'Fireplaces', 'FireplaceQu',
                'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea',
                'OpenPorchSF', 'Foundation_PConc'],
               dtype='object')
```

Most of the columns selected by GBR are the same as selected by Random forest

```
In [24]: reduced_var_data = x.loc[:,GBR.feature_importances_>0.002].copy()

         reduced_var_data['SalePrice'] = data['SalePrice']
         reduced_var_data.columns

         reduced_var_data.to_csv('reduced_var_data.csv')
```

model_ridge

May 24, 2019

Fit in Ridge with clean data and see its performance. Use grid search to improve performance. I also tried ensemble Lasso and Ridge here. Wenyu/remove outliers and log chang y to improve performance.ipynb

I try to remove some outliers in X for better performance. Also use log transfer y. Fit in Ridge and Lasso with clean data. Use grid search to improve performance.

In [1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

%matplotlib inline
```

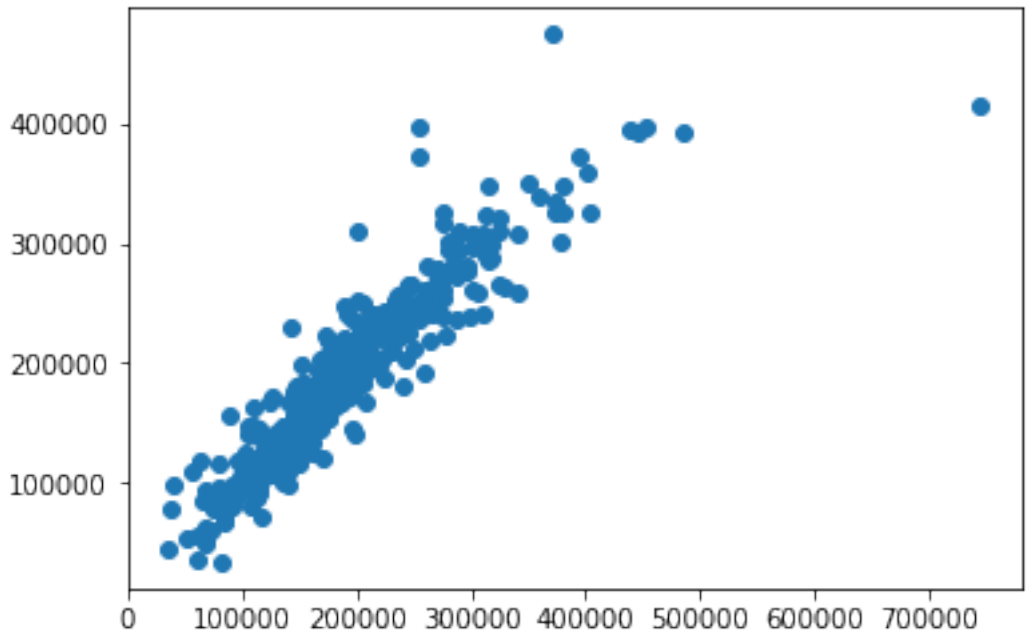
In [2]:
```
clean_data = pd.read_csv('clean_data.csv')
y=clean_data['SalePrice']
clean_data=clean_data.drop(['SalePrice'],axis=1) #drop the y in clean_data
X = clean_data.iloc[:,1:] # drop the id column
```

In [59]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [60]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
# scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_sc = scaler.fit_transform(X_train)
X_test_sc = scaler.transform(X_test)
```

In [61]:
```
rg = Ridge()
rg.fit(X_train_sc,y_train)
y_test_pred = rg.predict(X_test_sc)
y_train_pred = rg.predict(X_train_sc)
from sklearn.metrics import mean_squared_error
from math import sqrt
mse = mean_squared_error(y_test, y_test_pred)
sqrt(mse) #ridge's score is not so good
```

```
Out[61]: 32846.437585314496

In [62]: #use gridsearch on ridge
         from sklearn.model_selection import GridSearchCV
         parameters = { 'alpha':[0.1,1, 10,100,1000,10000]}
         rg = Ridge(random_state=42)
         clf = GridSearchCV(rg, parameters, cv=5)
         clf.fit(X_train_sc,y_train)
         y_test_pred = clf.predict(X_test_sc)
         y_train_pred = clf.predict(X_train_sc)

In [63]: mse = mean_squared_error(y_test, y_test_pred)
         sqrt(mse)

Out[63]: 30723.944334034182

In [64]: clf.best_estimator_

Out[64]: Ridge(alpha=100, copy_X=True, fit_intercept=True, max_iter=None,
            normalize=False, random_state=42, solver='auto', tol=0.001)

In [98]: plt.scatter(y_test, y_test_pred)

Out[98]: <matplotlib.collections.PathCollection at 0x10e8ad048>
```



```
In [74]: #ensemble lasso and ridge
         from sklearn.model_selection import cross_validate
         from sklearn.metrics import mean_squared_error
         from sklearn.linear_model import Lasso
         rg = Ridge(random_state=42,alpha=100)
         ls = Lasso(random_state=42,alpha=1000)
         regresers= [rg,ls]

         for regreser in regresers:

             scores = cross_validate(regreser,X_train_sc,y_train,
                                     scoring='neg_mean_squared_error',cv=5,return_train_score=
             print(regreser)
             for k,v in scores.items():
                 print(k+" : %0.2f +/- %0.2f" % (v.mean(), v.std()))

Ridge(alpha=100, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=42, solver='auto', tol=0.001)
fit_time : 0.02 +/- 0.00
score_time : 0.00 +/- 0.00
test_score : -1354644906.00 +/- 713770211.47
train_score : -219848276.68 +/- 6110185.84
Lasso(alpha=1000, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=42,
   selection='cyclic', tol=0.0001, warm_start=False)
fit_time : 0.02 +/- 0.01
score_time : 0.00 +/- 0.00
test_score : -1269186256.26 +/- 894450161.36
train_score : -355650029.71 +/- 4793514.48


In [76]: from sklearn.ensemble import BaggingRegressor
         bagged_config = dict(n_estimators=20, max_samples=.6, max_features=.7, random_state=4
         bagging_rg = BaggingRegressor(Ridge(alpha=100), **bagged_config)
         bagging_ls = BaggingRegressor(Lasso(alpha=1000), **bagged_config)

In [77]: regresers= [bagging_rg,bagging_ls]

         for regreser in regresers:

             scores = cross_validate(regreser,X_train_sc,y_train,
                                     scoring='neg_mean_squared_error',cv=5,return_train_score=
             print(regreser)
             for k,v in scores.items():
                 print(k+" : %0.2f +/- %0.2f" % (v.mean(), v.std()))

BaggingRegressor(base_estimator=Ridge(alpha=100, copy_X=True, fit_intercept=True, max_iter=None
   normalize=False, random_state=None, solver='auto', tol=0.001),
         bootstrap=True, bootstrap_features=False, max_features=0.7,
```

```
         max_samples=0.6, n_estimators=20, n_jobs=None, oob_score=False,
         random_state=42, verbose=0, warm_start=False)
fit_time : 0.16 +/- 0.01
score_time : 0.00 +/- 0.00
test_score : -1332284719.05 +/- 700794400.09
train_score : -668039547.95 +/- 71215202.36
BaggingRegressor(base_estimator=Lasso(alpha=1000, copy_X=True, fit_intercept=True, max_iter=100
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False),
         bootstrap=True, bootstrap_features=False, max_features=0.7,
         max_samples=0.6, n_estimators=20, n_jobs=None, oob_score=False,
         random_state=42, verbose=0, warm_start=False)
fit_time : 0.29 +/- 0.04
score_time : 0.00 +/- 0.00
test_score : -1373306492.90 +/- 865519408.52
train_score : -701705217.53 +/- 84580216.40
```

```
In [97]: from math import sqrt
         test_score = -scores['test_score'].mean()
         sqrt(test_score)

Out[97]: 37058.150154907366
```

model_Lasso

May 24, 2019

Fit in Lasso with clean data and see its performance. Use grid search to improve performance.

In [3]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

%matplotlib inline
```

In [4]:
```
clean_data = pd.read_csv('clean_data.csv')
clean_data.head()
```

Out[4]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	\
0	1	60	65.0	8450	7	5	2003	
1	2	20	80.0	9600	6	8	1976	
2	3	60	68.0	11250	7	5	2001	
3	4	70	60.0	9550	7	5	1915	
4	5	60	84.0	14260	8	5	2000	

	YearRemodAdd	ExterQual	ExterCond	...	SaleType_ConLI	\
0	2003	3	2	...	0	
1	1976	2	2	...	0	
2	2002	3	2	...	0	
3	1970	2	2	...	0	
4	2000	3	2	...	0	

	SaleType_ConLw	SaleType_New	SaleType_Oth	SaleType_WD	\
0	0	0	0	1	
1	0	0	0	1	
2	0	0	0	1	
3	0	0	0	1	
4	0	0	0	1	

	SaleCondition_AdjLand	SaleCondition_Alloca	SaleCondition_Family	\
0	0	0	0	
1	0	0	0	
2	0	0	0	
3	0	0	0	

```
4                     0                     0                     0

   SaleCondition_Normal  SaleCondition_Partial
0                     1                     0
1                     1                     0
2                     1                     0
3                     0                     0
4                     1                     0

[5 rows x 534 columns]
```

In [5]:
```
y=clean_data['SalePrice']
clean_data=clean_data.drop(['SalePrice'],axis=1) #drop the y in clean_data
X = clean_data.iloc[:,1:] # drop the id column
X.shape
```

Out[5]: (1459, 532)

In [9]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [6]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
# scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_sc = scaler.fit_transform(X_train)
X_test_sc = scaler.transform(X_test)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/preprocessing/data.py:625: DataC
  return self.partial_fit(X, y)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/base.py:462: DataConversionWarni
  return self.fit(X, **fit_params).transform(X)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:8: DataConversionW
```

In [7]:
```
clf = Lasso(random_state=42)
```

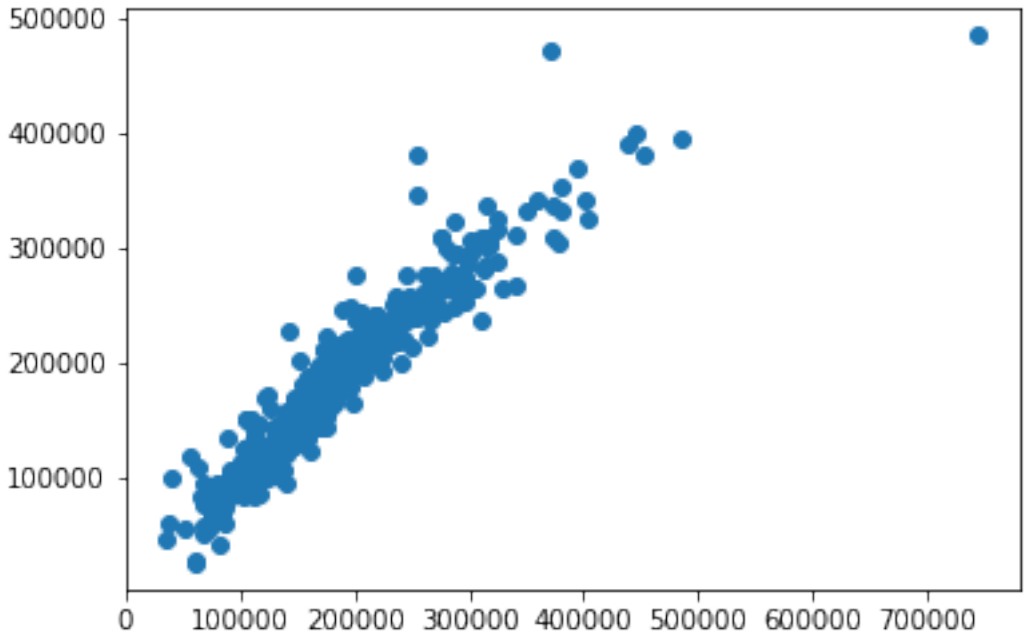
In [8]:
```
clf.fit(X_train_sc,y_train)
y_test_pred = clf.predict(X_test_sc)
y_train_pred = clf.predict(X_train_sc)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.
  ConvergenceWarning)
```

In [10]:
```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_test_pred)
from math import sqrt
sqrt(mse)
```

```
Out[10]: 33069.93524702181

In [55]: mse = mean_squared_error(y_train, y_train_pred)
         mse

Out[55]: 200068353.65906683

In [56]: # grid search to get the best hyperparameter.
         from sklearn.model_selection import GridSearchCV
         parameters = { 'max_iter':[1000,2000,5000],'alpha':[1, 10,100,1000,10000]}
         ls = Lasso(random_state=42)
         clf = GridSearchCV(ls, parameters, cv=5)
         clf.fit(X_train_sc,y_train)
         y_test_pred = clf.predict(X_test_sc)
         y_train_pred = clf.predict(X_train_sc)

In [2]: clf.best_params_


        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-2-9772bec5683a> in <module>()
    ----> 1 clf.best_params_


        NameError: name 'clf' is not defined


In [1]: clf.coef_


        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-1-eaad8d6955f1> in <module>()
    ----> 1 clf.coef_


        NameError: name 'clf' is not defined


In [57]: from math import sqrt
         mse_test = mean_squared_error(y_test, y_test_pred)
         sqrt(mse_test)

Out[57]: 26802.87448312158
```

```
In [58]: mse_train = mean_squared_error(y_train, y_train_pred)
         sqrt(mse_train)

Out[58]: 19794.68865118897

In [59]: plt.scatter(y_test, y_test_pred)

Out[59]: <matplotlib.collections.PathCollection at 0x10f1a3828>
```



change_y

May 24, 2019

LotArea means Lot size in square feet of a house. This should be a important feature since the larger the house, the higher the price should be naively. So I come up with an approach to change the question to be predict house price per square feet. Then, my y becomes house price/ LotArea and my X need to remove the LotArea feature. I trained Lasso on this new X and y. The performance is not improved so I didn't continue this method.

In [1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

%matplotlib inline
```

In [3]:
```
raw_data = pd.read_csv('house-prices-advanced-regression-techniques/train.csv')
raw_data.head()
```

Out[3]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	\
0	1	60	RL	65.0	8450	Pave	NaN	Reg	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	

	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	\
0	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
2	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
3	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
4	Lvl	AllPub	...	0	NaN	NaN	NaN	0	

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

[5 rows x 81 columns]

```
In [4]: raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
Id               1460 non-null int64
MSSubClass       1460 non-null int64
MSZoning         1460 non-null object
LotFrontage      1201 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null object
Alley            91 non-null object
LotShape         1460 non-null object
LandContour      1460 non-null object
Utilities        1460 non-null object
LotConfig        1460 non-null object
LandSlope        1460 non-null object
Neighborhood     1460 non-null object
Condition1       1460 non-null object
Condition2       1460 non-null object
BldgType         1460 non-null object
HouseStyle       1460 non-null object
OverallQual      1460 non-null int64
OverallCond      1460 non-null int64
YearBuilt        1460 non-null int64
YearRemodAdd     1460 non-null int64
RoofStyle        1460 non-null object
RoofMatl         1460 non-null object
Exterior1st      1460 non-null object
Exterior2nd      1460 non-null object
MasVnrType       1452 non-null object
MasVnrArea       1452 non-null float64
ExterQual        1460 non-null object
ExterCond        1460 non-null object
Foundation       1460 non-null object
BsmtQual         1423 non-null object
BsmtCond         1423 non-null object
BsmtExposure     1422 non-null object
BsmtFinType1     1423 non-null object
BsmtFinSF1       1460 non-null int64
BsmtFinType2     1422 non-null object
BsmtFinSF2       1460 non-null int64
BsmtUnfSF        1460 non-null int64
TotalBsmtSF      1460 non-null int64
Heating          1460 non-null object
HeatingQC        1460 non-null object
CentralAir       1460 non-null object
Electrical       1459 non-null object
```

```
1stFlrSF         1460 non-null int64
2ndFlrSF         1460 non-null int64
LowQualFinSF     1460 non-null int64
GrLivArea        1460 non-null int64
BsmtFullBath     1460 non-null int64
BsmtHalfBath     1460 non-null int64
FullBath         1460 non-null int64
HalfBath         1460 non-null int64
BedroomAbvGr     1460 non-null int64
KitchenAbvGr     1460 non-null int64
KitchenQual      1460 non-null object
TotRmsAbvGrd     1460 non-null int64
Functional       1460 non-null object
Fireplaces       1460 non-null int64
FireplaceQu      770 non-null object
GarageType       1379 non-null object
GarageYrBlt      1379 non-null float64
GarageFinish     1379 non-null object
GarageCars       1460 non-null int64
GarageArea       1460 non-null int64
GarageQual       1379 non-null object
GarageCond       1379 non-null object
PavedDrive       1460 non-null object
WoodDeckSF       1460 non-null int64
OpenPorchSF      1460 non-null int64
EnclosedPorch    1460 non-null int64
3SsnPorch        1460 non-null int64
ScreenPorch      1460 non-null int64
PoolArea         1460 non-null int64
PoolQC           7 non-null object
Fence            281 non-null object
MiscFeature      54 non-null object
MiscVal          1460 non-null int64
MoSold           1460 non-null int64
YrSold           1460 non-null int64
SaleType         1460 non-null object
SaleCondition    1460 non-null object
SalePrice        1460 non-null int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

```
In [5]: selected = raw_data.loc[:,['MSSubClass','LotFrontage','LotArea','OverallQual','Overall
```

```
In [8]: selected.head()
```

```
Out[8]:    MSSubClass  LotFrontage  LotArea  OverallQual  OverallCond  YearBuilt  \
        0          60         65.0     8450            7            5       2003
```

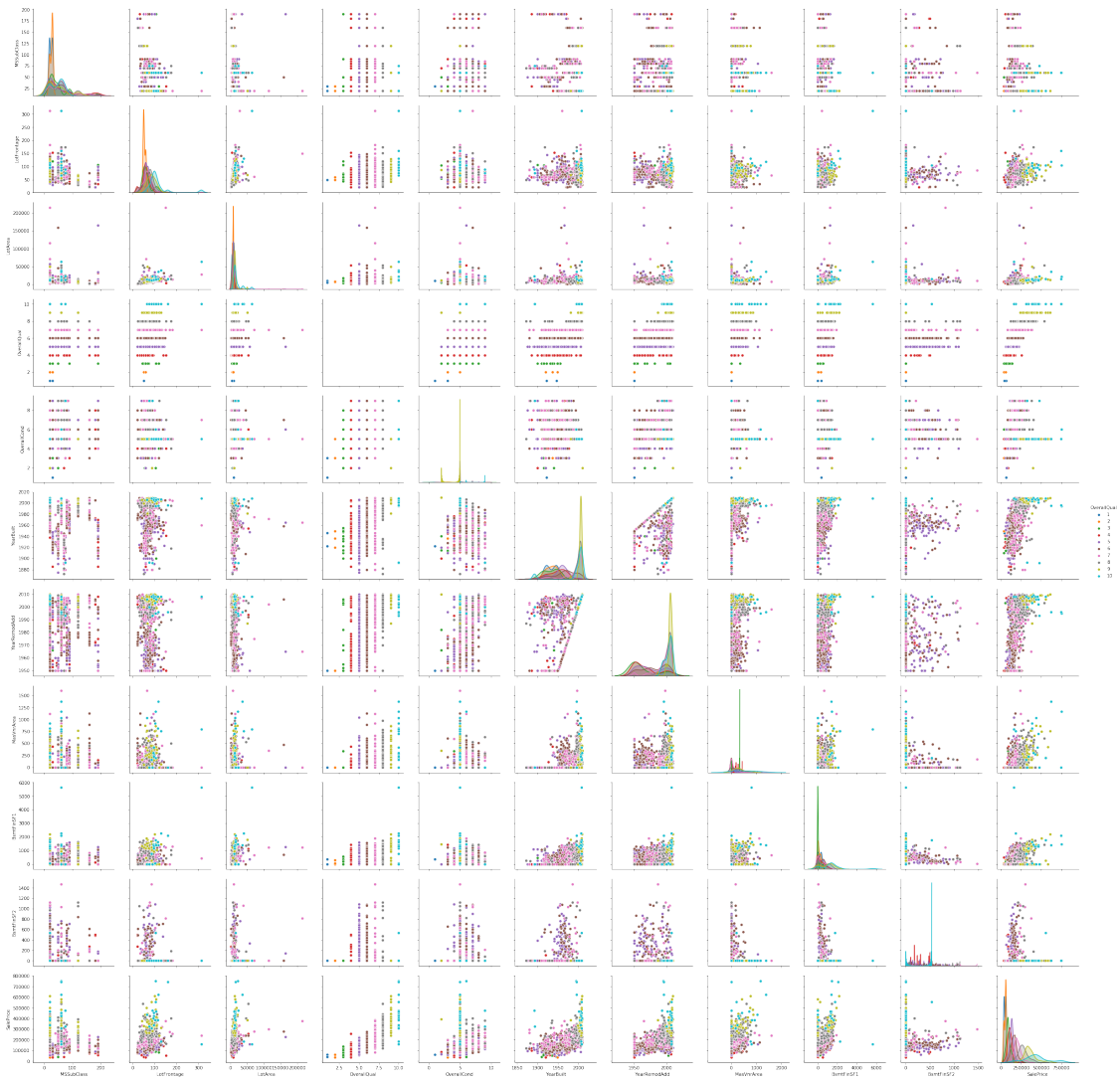
1	20	80.0	9600	6	8	1976
2	60	68.0	11250	7	5	2001
3	70	60.0	9550	7	5	1915
4	60	84.0	14260	8	5	2000

	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	SalePrice
0	2003	196.0	706	0	208500
1	1976	0.0	978	0	181500
2	2002	162.0	486	0	223500
3	1970	0.0	216	0	140000
4	2000	350.0	655	0	250000

In [10]:
```
sns.pairplot(selected,height=3,hue = 'OverallQual')
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/statsmodels/nonparametric/kde.py:448: Ru
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/statsmodels/nonparametric/kde.py:448: Ru
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/numpy/core/_methods.py:135: RuntimeWarni
  keepdims=keepdims)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/numpy/core/_methods.py:127: RuntimeWarni
  ret = ret.dtype.type(ret / rcount)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/statsmodels/nonparametric/kde.py:488: Ru
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/statsmodels/nonparametric/kdetools.py:34
  FAC1 = 2*(np.pi*bw/RANGE)**2
```

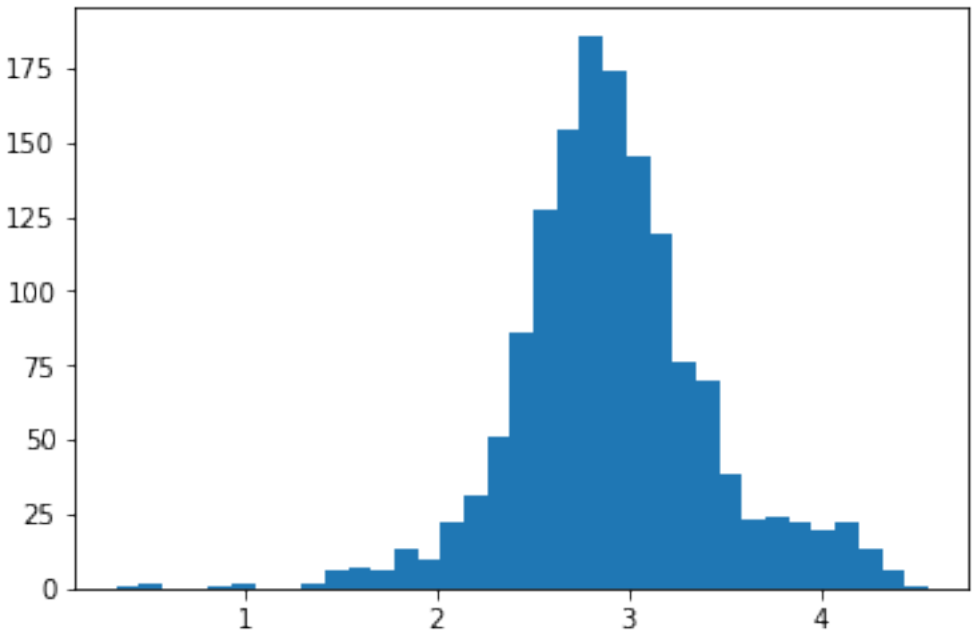
Out[10]: <seaborn.axisgrid.PairGrid at 0x1a26902668>



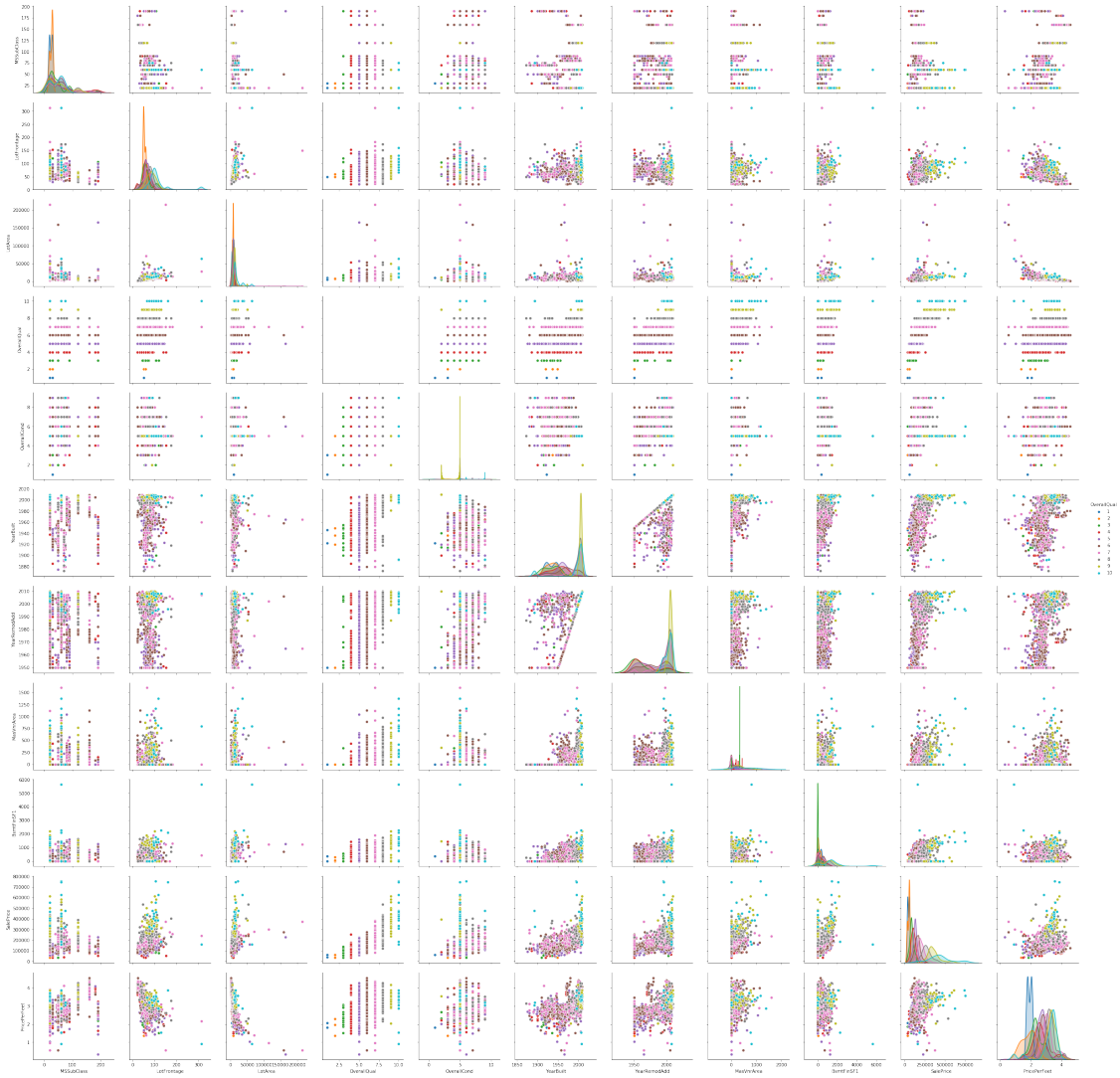
0.1 change the question to predict the price of each square feet

```
In [37]: raw_data['PricePerFeet'] = np.log(raw_data.SalePrice/raw_data.LotArea)

In [39]: plt.hist(raw_data.PricePerFeet,bins = 35)

Out[39]: (array([  1.,   2.,   0.,   0.,   1.,   2.,   0.,   0.,   2.,   6.,   7.,
                   6.,  13.,  10.,  22.,  31.,  51.,  86., 127., 154., 186., 174.,
                 145., 119.,  76.,  70.,  38.,  23.,  24.,  22.,  20.,  22.,  13.,
                   6.,   1.]),
          array([0.3296209 , 0.45042937, 0.57123783, 0.6920463 , 0.81285477,
                 0.93366324, 1.05447171, 1.17528018, 1.29608865, 1.41689712,
                 1.53770558, 1.65851405, 1.77932252, 1.90013099, 2.02093946,
                 2.14174793, 2.2625564 , 2.38336486, 2.50417333, 2.6249818 ,
```

```
        2.74579027, 2.86659874, 2.98740721, 3.10821568, 3.22902415,
        3.34983261, 3.47064108, 3.59144955, 3.71225802, 3.83306649,
        3.95387496, 4.07468343, 4.1954919 , 4.31630036, 4.43710883,
        4.5579173 ]),
 <a list of 35 Patch objects>)
```




```
In [40]: selected = raw_data.loc[:,['MSSubClass','LotFrontage','LotArea','OverallQual','Overall

In [42]: sns.pairplot(selected,height=3,hue = 'OverallQual')

/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/statsmodels/nonparametric/kde.py:448: Ru
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/statsmodels/nonparametric/kde.py:448: Ru
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/numpy/core/_methods.py:135: RuntimeWarni
  keepdims=keepdims)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/numpy/core/_methods.py:127: RuntimeWarni
  ret = ret.dtype.type(ret / rcount)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/statsmodels/nonparametric/kde.py:488: Ru
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/statsmodels/nonparametric/kdetools.py:34
  FAC1 = 2*(np.pi*bw/RANGE)**2


Out[42]: <seaborn.axisgrid.PairGrid at 0x1a312fceb8>
```

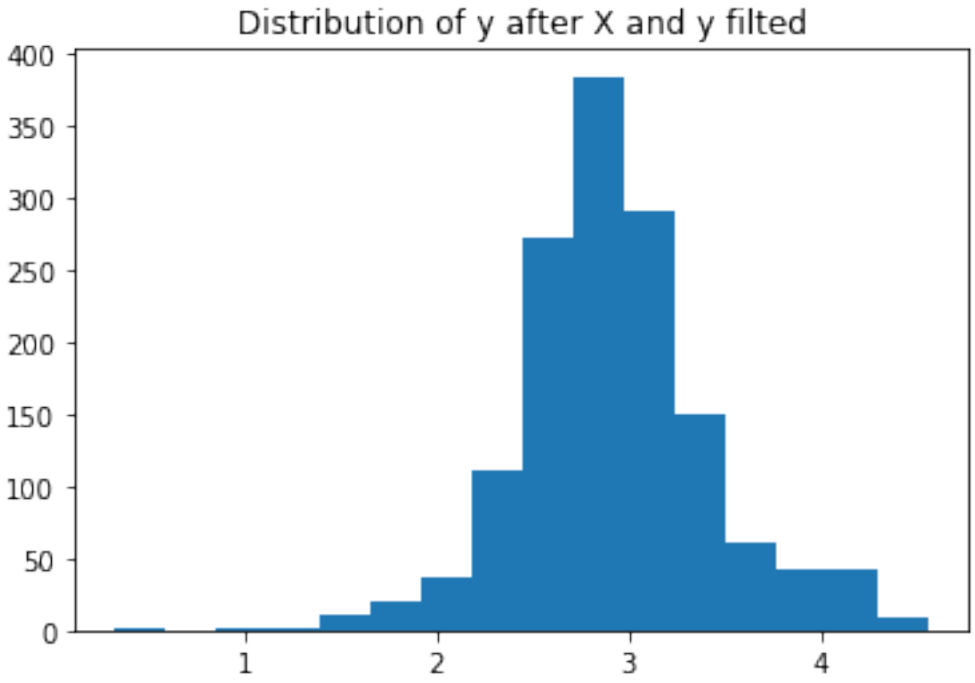


0.2 Redo Lasso with cleaned data and change y

```
In [43]: clean_data = pd.read_csv('clean_data.csv')
         y=clean_data['SalePrice']
         clean_data=clean_data.drop(['SalePrice'],axis=1) #drop the y in clean_data
         X = clean_data.iloc[:,1:] # drop the id column
         y_loged = np.log(y/X.LotArea) # log transform y
```

```
In [47]: # remove X outliers.
         X_filted = X[X['LotFrontage'] < 129]
         y_filted = y_loged[X['LotFrontage'] < 129]
         X_filted = X_filted.drop(['LotArea'],axis=1) #drop the LotArea in X because it's cons
         plt.hist(y_filted,bins=16)
         plt.title('Distribution of y after X and y filted')
```

Out[47]: Text(0.5,1,'Distribution of y after X and y filted')



0.3 fit into lasso

```
In [52]: from sklearn.model_selection import train_test_split
         from sklearn.linear_model import Lasso
         from sklearn.preprocessing import StandardScaler
         from sklearn.metrics import mean_squared_error
         from math import sqrt
         X_train, X_test, y_train, y_test = train_test_split(X_filted, y_filted, test_size=0.3
         scaler = StandardScaler()
         X_train_sc = scaler.fit_transform(X_train)
         X_test_sc = scaler.transform(X_test)
         clf = Lasso(random_state=42)
         clf.fit(X_train_sc,y_train)
         y_test_pred = clf.predict(X_test_sc)
         y_train_pred = clf.predict(X_train_sc)

         mse = mean_squared_error(y_test, y_test_pred)
         sqrt(mse)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/preprocessing/data.py:625: DataC
  return self.partial_fit(X, y)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/base.py:462: DataConversionWarni
  return self.fit(X, **fit_params).transform(X)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:9: DataConversionWa
  if __name__ == '__main__':
```

Out[52]: 0.5030524700097734

In [58]:
```python
from sklearn.model_selection import GridSearchCV
parameters = { 'max_iter':[3,5,10,50,100],'alpha':[0.001,0.01,0.1,1, 10,100,1000]}
ls = Lasso(random_state=42)
clf = GridSearchCV(ls, parameters, cv=5)
clf.fit(X_train_sc,y_train)
y_test_pred = clf.predict(X_test_sc)
y_train_pred = clf.predict(X_train_sc)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
```

In [59]:
```
clf.best_params_
```

Out[59]:
```
{'alpha': 0.01, 'max_iter': 5}
```

In [60]:
```
mse_test = mean_squared_error(y_test, y_test_pred)
sqrt(mse_test)
```

Out[60]:
```
0.27444189034191757
```

[1]
```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split, KFold, cross_val_sc
from sklearn.pipeline import make_pipeline

from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_absolute_error

import matplotlib.pyplot as plt
```

[2]
```
df = pd.read_csv("../clean_data.csv", index_col=0)
```

[3]
```
df.head()
```

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	Ye
Id						
1	60	65.0	8450	7	5	20
2	20	80.0	9600	6	8	19
3	60	68.0	11250	7	5	20
4	70	60.0	9550	7	5	19
5	60	84.0	14260	8	5	20

5 rows × 533 columns

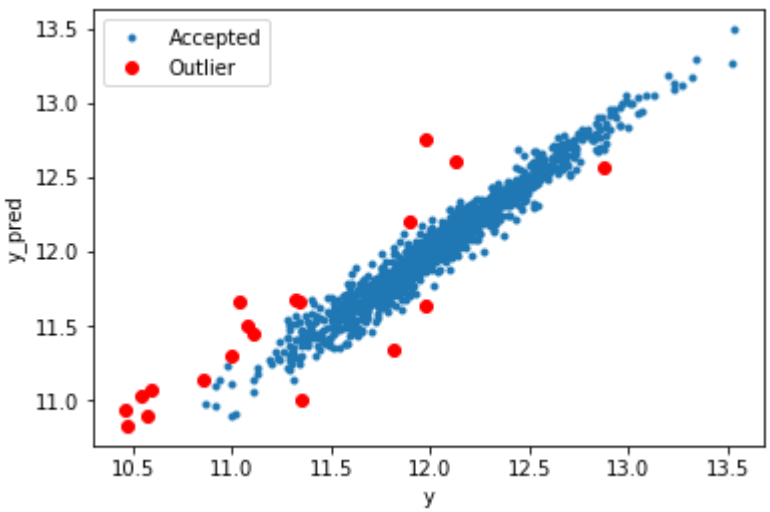
Identify and Remove Outliers

[13]
```
# function to detect outliers based on the predictions of a model
def find_outliers(model, df, sigma=3):

    X = df.drop("SalePrice", axis=1)
    y = np.log(df['SalePrice'])

    model.fit(X,y)
    y_pred = pd.Series(model.predict(X), index=y.index)

    resid = y - y_pred
```

```
    mean_resid = resid.mean()
    std_resid = resid.std()

    z = (resid - mean_resid)/std_resid
    outliers = z[abs(z)>sigma].index

    print('R2=',model.score(X,y))
    print('rmse=',mean_absolute_error(y, y_pred))
    print('----------------------------------------')

    print('mean of residuals:',mean_resid)
    print('std of residuals:',std_resid)
    print('----------------------------------------')

    print(len(outliers),'outliers:')
    print(outliers.tolist())

    plt.plot(y,y_pred,'.')
    plt.plot(y.loc[outliers],y_pred.loc[outliers],'ro')
    plt.legend(['Accepted','Outlier'])
    plt.xlabel('y')
    plt.ylabel('y_pred');

    return outliers
```

[14]
```
outliers = find_outliers(Ridge(), df)

df_model = df.drop(outliers)
```

```
R2= 0.9457976002390555
rmse= 0.06523816293351563
----------------------------------------
mean of residuals: -1.5645089366890485e-15
std of residuals: 0.09302990503748261
----------------------------------------
19 outliers:
[31, 89, 411, 463, 496, 524, 534, 633, 682, 689, 711, 875, 917, 969, 971,
1299, 1325, 1433, 1454]
```



[16]
```
X = df_model.drop("SalePrice", axis=1)
y = np.log(df_model['SalePrice'])

model = Ridge()
model.fit(X,y)
y_pred = pd.Series(model.predict(X), index=y.index)

print('Correlation:', np.corrcoef(y,y_pred)[0][1])
print('R2=',model.score(X,y))
print('rmse=',mean_absolute_error(y, y_pred))
```

```
Correlation: 0.9814716471008289
R2= 0.9632687458251801
rmse= 0.055460424577453954
```

Fit and Optimise Models

Linear Regression:

- sklearn.linear_model.Ridge
- sklearn.linear_model.Lasso
- sklearn.linear_model.ElasticNet

Support Vector Machines:

- sklearn.svm.LinearSVR
- sklearn.svm.SVR

Nearest Neighbours:

- sklearn.neighbors.KNearestNeighborsRegressor

Tree Based:

- sklearn.ensemble.RandomForestRegressor
- sklearn.ensemble.GradientBoostingRegressor
- xgboost.XGBRegressor

[9]
```
def train_model(model, param_grid=[], X=[], y=[], splits=5, repeats=5):

    rkfold = RepeatedKFold(n_splits=splits, n_repeats=repeats)

    if len(param_grid)>0:
        gsearch = GridSearchCV(model, param_grid, cv=rkfold,
                               scoring='neg_mean_absolute_error', n_jobs
                               verbose=1, return_train_score=True)
        gsearch.fit(X,y)

        model = gsearch.best_estimator_
        best_idx = gsearch.best_index_

        grid_results = pd.DataFrame(gsearch.cv_results_)
```

```
            cv_mean = abs(grid_results.loc[best_idx,'mean_test_score'])
            cv_std = grid_results.loc[best_idx,'std_test_score']

        else:
            grid_results = []
            cv_results = cross_val_score(model, X, y, scoring='neg_mean_absol
            cv_mean = abs(np.mean(cv_results))
            cv_std = np.std(cv_results)

        y_pred = model.predict(X)

        print('----------------------')
        print(model)
        print('----------------------')
        print('Correlation:', np.corrcoef(y,y_pred)[0][1])
        print('R2=',model.score(X,y))
        print('rmse=',mean_absolute_error(y, y_pred))
        print('cross_val: mean=',cv_mean,', std=',cv_std)



        return model, grid_results
```

ElasticNet

[18]
```
en_int = ElasticNet()

param_grid = {'alpha': np.arange(1e-4,1e-3,1e-4),
              'l1_ratio': np.arange(0.1,1.0,0.1),
              'max_iter':[100000]}

# param_grid = {'l1_ratio': np.arange(0.1,1.0,0.1),
#               'max_iter':[100000]}

en_int, grid_results = train_model(en_int, X=X, y=y, param_grid=param_gri
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
----------------------
ElasticNet(alpha=0.0004, copy_X=True, fit_intercept=True, l1_ratio=0.9,
      max_iter=100000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
----------------------
Correlation: 0.9736281907947597
R2= 0.9479347232669106
rmse= 0.0669953327081602
cross_val: mean= 0.07380454712979753 , std= 0.0031932122104661904
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    6.5s
[Parallel(n_jobs=8)]: Done 184 tasks      | elapsed:   16.6s
[Parallel(n_jobs=8)]: Done 405 out of 405 | elapsed:   23.3s finished
```

Random Forest Regression

[17]
```
rfr = RandomForestRegressor()

param_grid = {'n_estimators':[100,150,200],
              'max_features':[25,50,75],
              'min_samples_split':[2,4,6]}

rfr, grid_results = train_model(rfr, X=X, y=y,param_grid=param_grid,
                                                splits=5, repeats=1)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
----------------------
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=75, max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=150, n_jobs=None, oob_score=False,
           random_state=None, verbose=0, warm_start=False)
----------------------
Correlation: 0.9949262251431482
R2= 0.9870952256382878
rmse= 0.031135445053697107
cross_val: mean= 0.08481487732403423 , std= 0.0028683554643976787
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    5.5s
[Parallel(n_jobs=8)]: Done 135 out of 135 | elapsed:   20.1s finished
```

[ ]

[1]
```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split, KFold, cross_val_sc
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.pipeline import make_pipeline

from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import ElasticNet

from sklearn.metrics import mean_squared_error
from sklearn.metrics import make_scorer

import matplotlib.pyplot as plt
```

[2]
```
df = pd.read_csv("./clean_data.csv", index_col=0)
```

[3]
```
df.head()
```

	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	Year
Id						
1	65.0	8450	7	5	2003	2003
2	80.0	9600	6	8	1976	1976
3	68.0	11250	7	5	2001	2002
4	60.0	9550	7	5	1915	1970
5	84.0	14260	8	5	2000	2000

5 rows × 546 columns

[4]
```
X = df.drop("SalePrice", axis=1)
y = np.log(df['SalePrice'])
```

Fit and Optimise Models

Linear Regression:

- sklearn.linear_model.Ridge
- sklearn.linear_model.Lasso
- sklearn.linear_model.ElasticNet

Support Vector Machines:

- sklearn.svm.LinearSVR
- sklearn.svm.SVR

Nearest Neighbours:

- sklearn.neighbors.KNearestNeighborsRegressor

Tree Based:

- sklearn.ensemble.RandomForestRegressor
- sklearn.ensemble.GradientBoostingRegressor
- xgboost.XGBRegressor

[5]
```
def rmse(y_true, y_pred):
    diff = y_pred - y_true
    sum_sq = sum(diff**2)
    n = len(y_pred)

    return np.sqrt(sum_sq/n)

rmse_scorer = make_scorer(rmse, greater_is_better=False)
```

[6]
```
def train_model(model, param_grid=[], X=[], y=[],
                splits=5, repeats=5):

    rkfold = RepeatedKFold(n_splits=splits, n_repeats=repeats)

    if len(param_grid)>0:
        gsearch = GridSearchCV(model, param_grid, cv=rkfold,
                               scoring=rmse_scorer, n_jobs = 8,
                               verbose=1, return_train_score=True)

        gsearch.fit(X,y)

        model = gsearch.best_estimator_
        best_idx = gsearch.best_index_

        grid_results = pd.DataFrame(gsearch.cv_results_)
        cv_mean = abs(grid_results.loc[best_idx,'mean_test_score'])
        cv_std = grid_results.loc[best_idx,'std_test_score']

    else:
        grid_results = []
        cv_results = cross_val_score(model, X, y, scoring=rmse_scorer, cv
        cv_mean = abs(np.mean(cv_results))
        cv_std = np.std(cv_results)

    cv_score = pd.Series({'mean':cv_mean,'std':cv_std})

    y_pred = model.predict(X)

    print('----------------------')
```

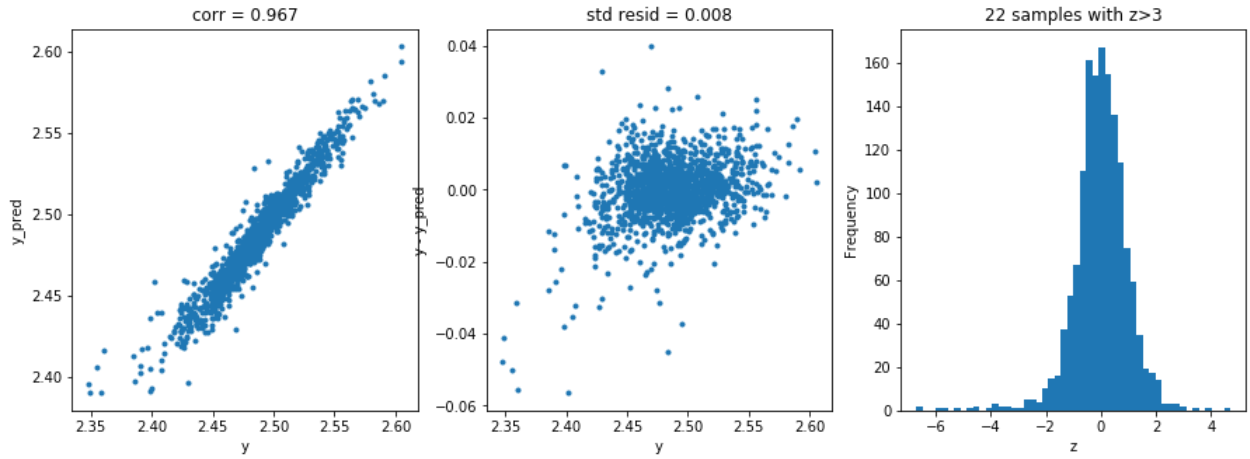
```
    print(model)
    print('----------------------')
    print('score=',model.score(X,y))
    print('rmse=',rmse(y, y_pred))
    print('cross_val: mean=',cv_mean,', std=',cv_std)

    y_pred = pd.Series(y_pred,index=y.index)
    resid = y - y_pred
    mean_resid = resid.mean()
    std_resid = resid.std()
    z = (resid - mean_resid)/std_resid
    n_outliers = sum(abs(z)>3)

    plt.figure(figsize=(15,5))
    ax_131 = plt.subplot(1,3,1)
    plt.plot(y,y_pred,'.')
    plt.xlabel('y')
    plt.ylabel('y_pred');
    plt.title('corr = {:.3f}'.format(np.corrcoef(y,y_pred)[0][1]))
    ax_132=plt.subplot(1,3,2)
    plt.plot(y,y-y_pred,'.')
    plt.xlabel('y')
    plt.ylabel('y - y_pred');
    plt.title('std resid = {:.3f}'.format(std_resid))

    ax_133=plt.subplot(1,3,3)
    z.plot.hist(bins=50,ax=ax_133)
    plt.xlabel('z')
    plt.title('{:.0f} samples with z>3'.format(n_outliers))

    return model, cv_score, grid_results
```

ElasticNet

[7]
```
en_int = ElasticNet()

param_grid = {'alpha': np.arange(1e-4,1e-3,1e-4),
              'l1_ratio': np.arange(0.1,1.0,0.1),
              'max_iter':[100000]}
# param_grid = {'l1_ratio': np.arange(0.1,1.0,0.1),
#               'max_iter':[100000]}


en_int, cv_score, grid_results = train_model(en_int, X=X, y=y, param_grid
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
----------------------
ElasticNet(alpha=0.0001, copy_X=True, fit_intercept=True, l1_ratio=0.2,
      max_iter=100000, normalize=False, positive=False, precompute=False,
```

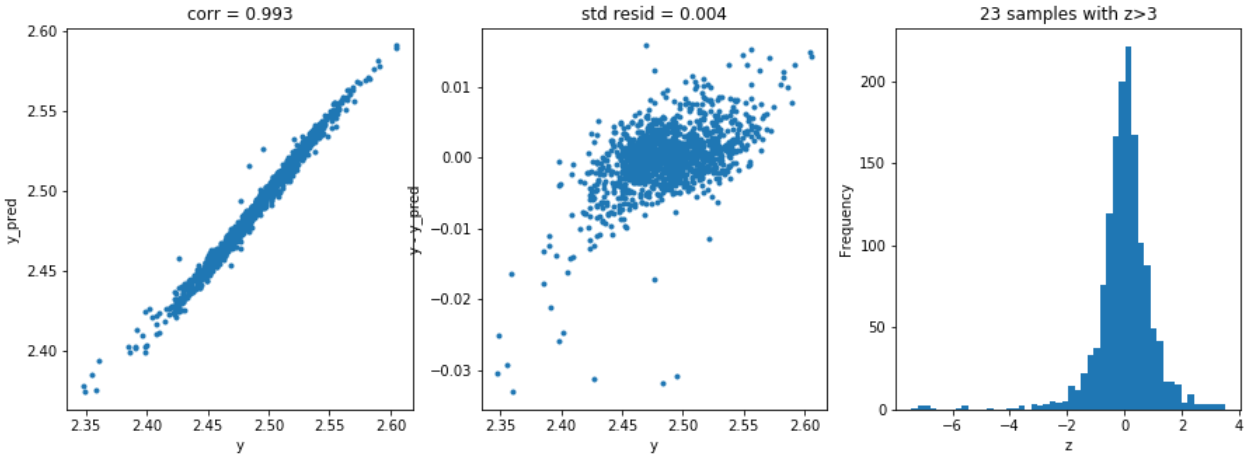
```
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
----------------------
score= 0.9353337276090429
rmse= 0.008443259909998552
cross_val: mean= 0.011679146088891477 , std= 0.0022280288068281012
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    3.7s
[Parallel(n_jobs=8)]: Done 184 tasks      | elapsed:    7.1s
[Parallel(n_jobs=8)]: Done 390 out of 405 | elapsed:   10.1s remaining:
0.4s
[Parallel(n_jobs=8)]: Done 405 out of 405 | elapsed:   10.2s finished
```



Random Forest Regression

[8]
```
rfr = RandomForestRegressor()

param_grid = {'n_estimators':[100,150,200],
              'max_features':[25,50,75],
              'min_samples_split':[2,4,6]}

rfr, cv_score, grid_results = train_model(rfr, X=X, y=y,param_grid=param_
                                          splits=5, repeats=1)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
----------------------
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=75, max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=100, n_jobs=None, oob_score=False,
           random_state=None, verbose=0, warm_start=False)
----------------------
score= 0.981874803806623
rmse= 0.004470052284003907
cross_val: mean= 0.01170475735716495 , std= 0.0005079339374733876
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    3.9s
[Parallel(n_jobs=8)]: Done 135 out of 135 | elapsed:   19.5s finished
```


corr = 0.993
y_pred
y
std resid = 0.004
y - y_pred
y
23 samples with z>3
Frequency
z

[ ]

Remove Outliers

May 24, 2019

0.0.1 Remove Ourliers

In this section, We will remove outliers using different approaches. First, we will start by usig RANSAC algorithm which is able to define the inliers and outliers.

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import RANSACRegressor    # Robust method for regression and
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

%matplotlib inline
```

In [3]:
```
data = pd.read_csv('reduced_var_data.csv', index_col = 0)
y = data['SalePrice']
x = data.drop(labels = 'SalePrice', axis=1)
print(data.shape)
data.head()
```

```
(1459, 32)
```

Out[3]:

Id	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	\
1	65.0	8450.0	7.0	5.0	2003.0	2003.0	
2	80.0	9600.0	6.0	8.0	1976.0	1976.0	
3	68.0	11250.0	7.0	5.0	2001.0	2002.0	
4	60.0	9550.0	7.0	5.0	1915.0	1970.0	

```
5          84.0  14260.0          8.0          5.0     2000.0        2000.0

    ExterQual  BsmtQual  BsmtCond  BsmtExposure    ...      GarageFinish  \
Id                                                 ...
1         3.0       4.0       3.0           1.0    ...               2.0
2         2.0       4.0       3.0           4.0    ...               2.0
3         3.0       4.0       3.0           2.0    ...               2.0
4         2.0       3.0       4.0           1.0    ...               1.0
5         3.0       4.0       3.0           3.0    ...               2.0

    GarageCars  GarageArea  OpenPorchSF  MoSold  MSZoning_RL  \
Id
1          2.0       548.0         61.0     2.0          1.0
2          2.0       460.0          0.0     5.0          1.0
3          2.0       608.0         42.0     9.0          1.0
4          3.0       642.0         35.0     2.0          1.0
5          3.0       836.0         84.0    12.0          1.0

    Neighborhood_Crawfor  MasVnrArea_209.0  MasVnrArea_428.0  SalePrice
Id
1                    0.0               0.0               0.0     208500
2                    0.0               0.0               0.0     181500
3                    0.0               0.0               0.0     223500
4                    1.0               0.0               0.0     140000
5                    0.0               0.0               0.0     250000

[5 rows x 32 columns]
```

0.1 Find Outliers using RANSACRegressor

In [4]:
```
# Define outliers in the data set
ransac = RANSACRegressor(LinearRegression(),
                         max_trials=1000,
                         min_samples=1000,
                         loss='absolute_loss',
                         random_state=42)


ransac.fit(x, y)
```

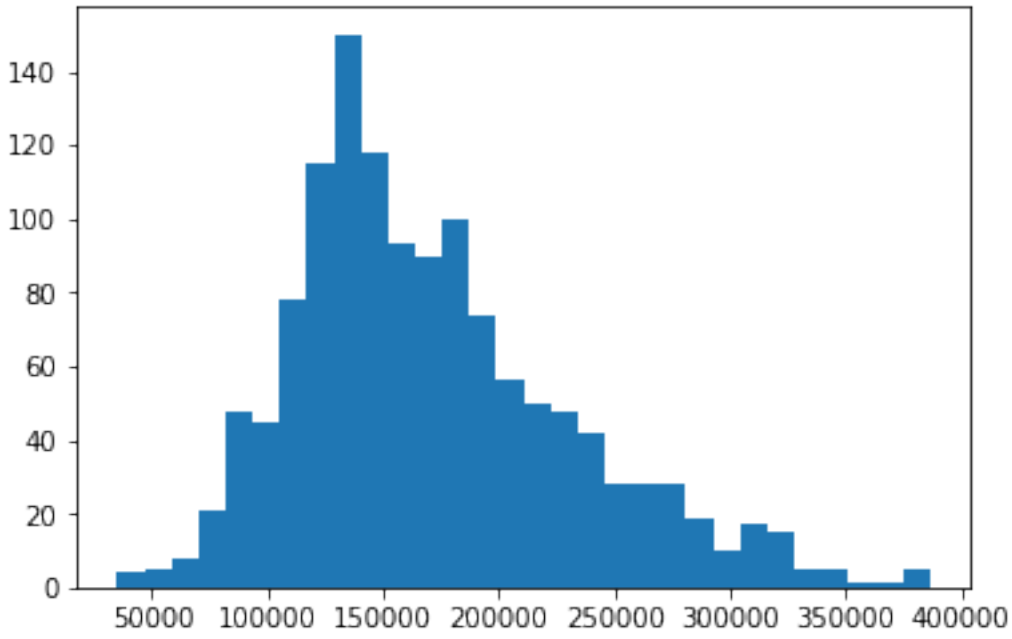
Out[4]:
```
RANSACRegressor(base_estimator=LinearRegression(copy_X=True, fit_intercept=True, n_jobs
         normalize=False),
        is_data_valid=None, is_model_valid=None, loss='absolute_loss',
        max_skips=inf, max_trials=1000, min_samples=1000, random_state=42,
        residual_threshold=None, stop_n_inliers=inf, stop_probability=0.99,
        stop_score=inf)
```

In [5]:
```
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)
```

```
In [6]: outlier_mask.sum(), inlier_mask.sum()

Out[6]: (152, 1307)
```

Now we removed the outliers from the dataset. The original data contains 1459 examples and the data without any outliers have 1307 examples. In other words, we removed 152 examples.

```
In [7]: plt.hist(y[inlier_mask], bins = 30);
```



0.2 SVR on data without outliers

```
In [8]: x_train, x_test, y_train, y_test = train_test_split(x[inlier_mask], y[inlier_mask], tes

In [9]: pipe = Pipeline(steps= [('ss', StandardScaler()), ('clf', SVR(gamma='scale'))])

        param_grid = {
            'clf__C':[0.1, 0.5, 1.0, 1.5, 10,100, 150, 1000],
            'clf__kernel': ['linear', 'rbf', 'sigmoid', 'poly']
        }

        search = GridSearchCV(pipe, param_grid, cv=5, iid=False, scoring='neg_mean_absolute_er
                              return_train_score=False)
        search.fit(x, y)          # Here I am using the whole training data
        print("Best parameter (CV score=%0.3f):" % search.best_score_)
        print(search.best_params_)
```

```
Best parameter (CV score=-19636.219):
{'clf__C': 1000, 'clf__kernel': 'linear'}
```

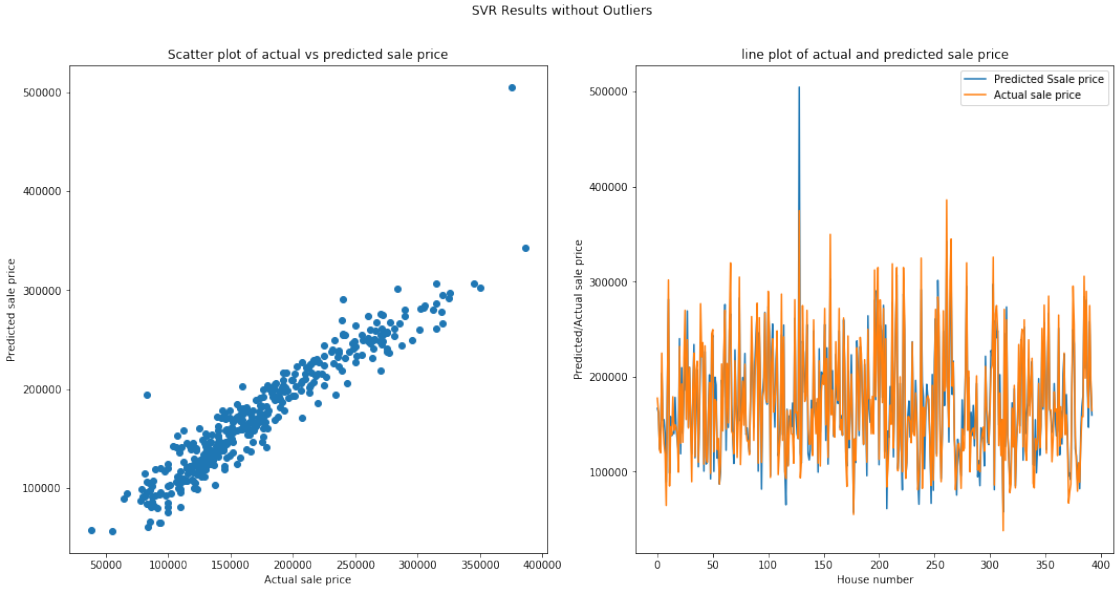
```
In [10]: ss = StandardScaler()
         ss.fit(x_train)
         x_train = ss.transform(x_train)
         x_test = ss.transform(x_test)

         best_svr = SVR(kernel='linear', gamma ='scale', C = 1000)
         best_svr.fit(x_train, y_train)
         y_pred = best_svr.predict(x_test)
         print(mean_absolute_error(y_test, y_pred))
         print('R2_score = {}'.format(r2_score(y_test, y_pred)))
```

```
14062.252899828172
R2_score = 0.9027249959650371
```

```
In [11]: print(best_svr.score(x_test, y_test))
```

```
0.9027249959650371
```

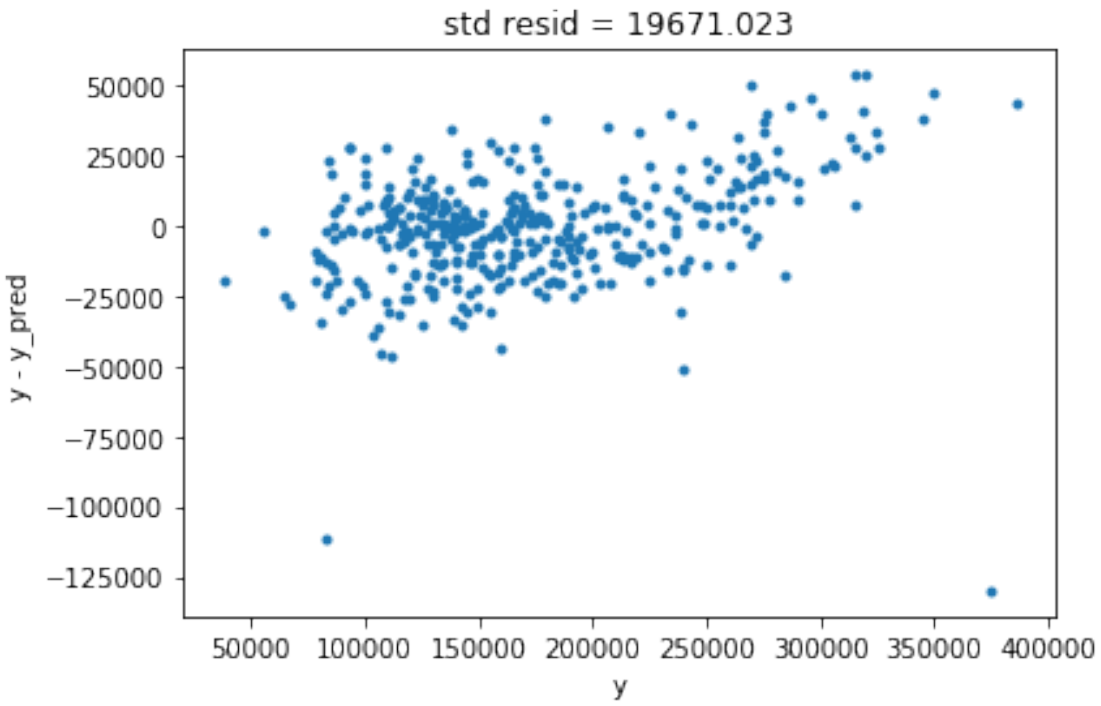
```
In [12]: y_pred = best_svr.predict(x_test)
         fig = plt.figure(figsize=(15,8))
         fig.suptitle('SVR Results without Outliers')
         plt.subplot(121)
         plt.scatter(y_test.values, y_pred)
         plt.xlabel('Actual sale price')
         plt.ylabel('Predicted sale price')
         plt.title('Scatter plot of actual vs predicted sale price')
         plt.subplot(122)
         plt.plot((y_pred), label='Predicted Ssale price')
         plt.plot((y_test.values), label='Actual sale price')
         plt.xlabel('House number')
         plt.ylabel('Predicted/Actual sale price')
         plt.title('line plot of actual and predicted sale price')
         plt.legend()
         plt.tight_layout()
         fig.subplots_adjust(top=0.88)
```



In [13]:
```
print("Corrolation between true and predicted value using SVR on the actual sale price
      format(np.corrcoef(y_test,y_pred)[0][1]))
```

```
Corrolation between true and predicted value using SVR on the actual sale price is 0.9502505465
```

In [14]:
```
resid = y_test - y_pred
mean_resid = resid.mean()
std_resid = resid.std()
plt.plot(y_test,y_test-y_pred,'.')
plt.xlabel('y')
plt.ylabel('y - y_pred');
plt.title('std resid = {:.3f}'.format(std_resid));
```



0.3 Neural Networks

```
In [15]: x_train, x_test, y_train, y_test = train_test_split(x[inlier_mask], y[inlier_mask], te

         ss = StandardScaler()
         ss.fit(x_train)
         x_train = ss.transform(x_train)
         x_test = ss.transform(x_test)

In [16]: seed = 42
         np.random.seed(seed)

         n_feat = x.shape[1]
         mdl = Sequential()
         mdl.add(Dense(units=256, input_dim = n_feat, activation = 'relu'))
         mdl.add(Dense(units=1, activation='linear'))

         mdl.compile(loss='mean_squared_error', optimizer='adam', metrics= ['mse', 'mae'])
         monitor = EarlyStopping(monitor= 'val_loss', min_delta=1e-3,
                                 patience = 10, verbose=1, mode = 'auto')
         history = mdl.fit(x_train, y_train, validation_data = (x_test, y_test),
                           callbacks=[monitor],batch_size= 64,epochs=5000,verbose=0)

         y_pred = mdl.predict(x_test)
```

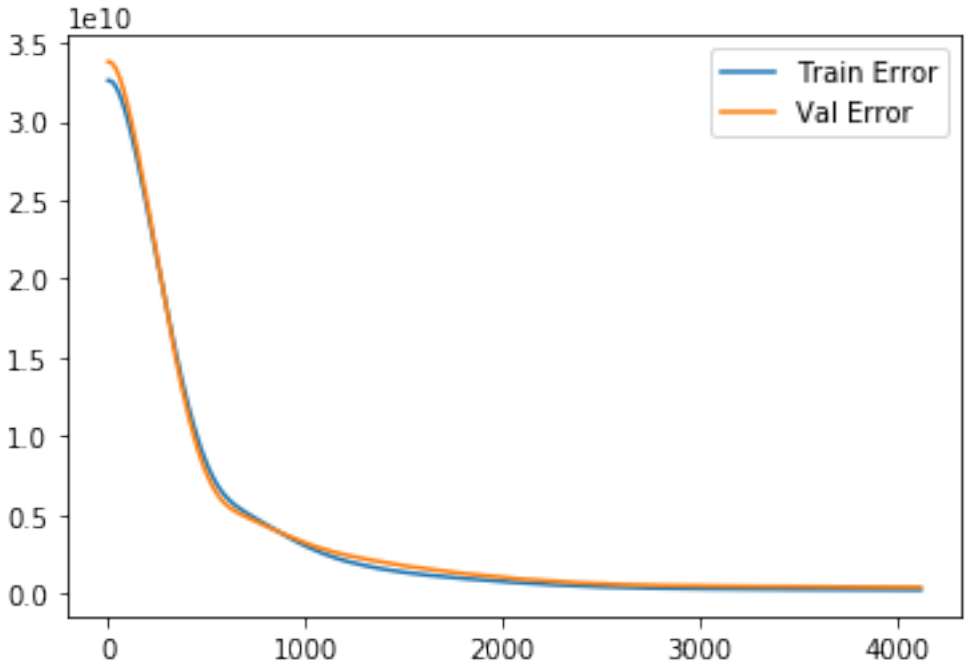
```
        MSEscore = (mean_squared_error(y_pred, y_test))
        print('Score MSE = {}'.format(MSEscore))

        MAEscore = (mean_absolute_error(y_pred, y_test))
        print('Score MAE = {}'.format(MAEscore))
        print('R2_score = {}'.format(r2_score(y_test, y_pred)))
        mdl.summary()
```

```
Epoch 04124: early stopping
Score MSE = 380631885.78712326
Score MAE = 13701.82104802799
R2_score = 0.90407417909799
```

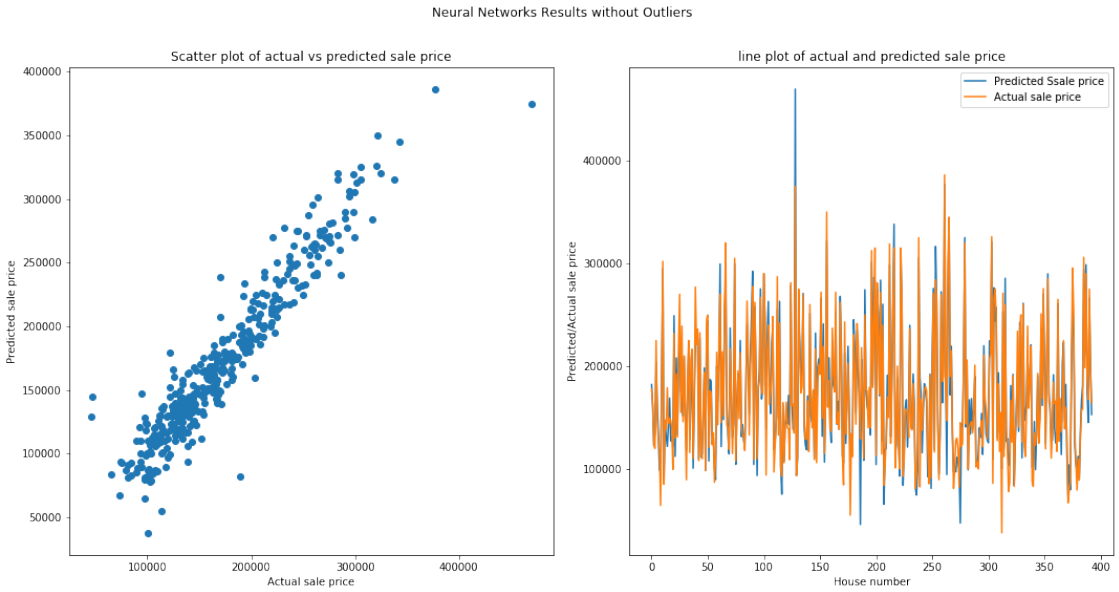
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               8192
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 8,449
Trainable params: 8,449
Non-trainable params: 0
_________________________________________________________________
```

In [17]:
```
plt.figure()
plt.plot(history.epoch, history.history['mean_squared_error'], label = 'Train Error')
plt.plot(history.epoch, history.history['val_mean_squared_error'], label = 'Val Error
plt.legend()
```

Out[17]: <matplotlib.legend.Legend at 0x2256aa6ab00>



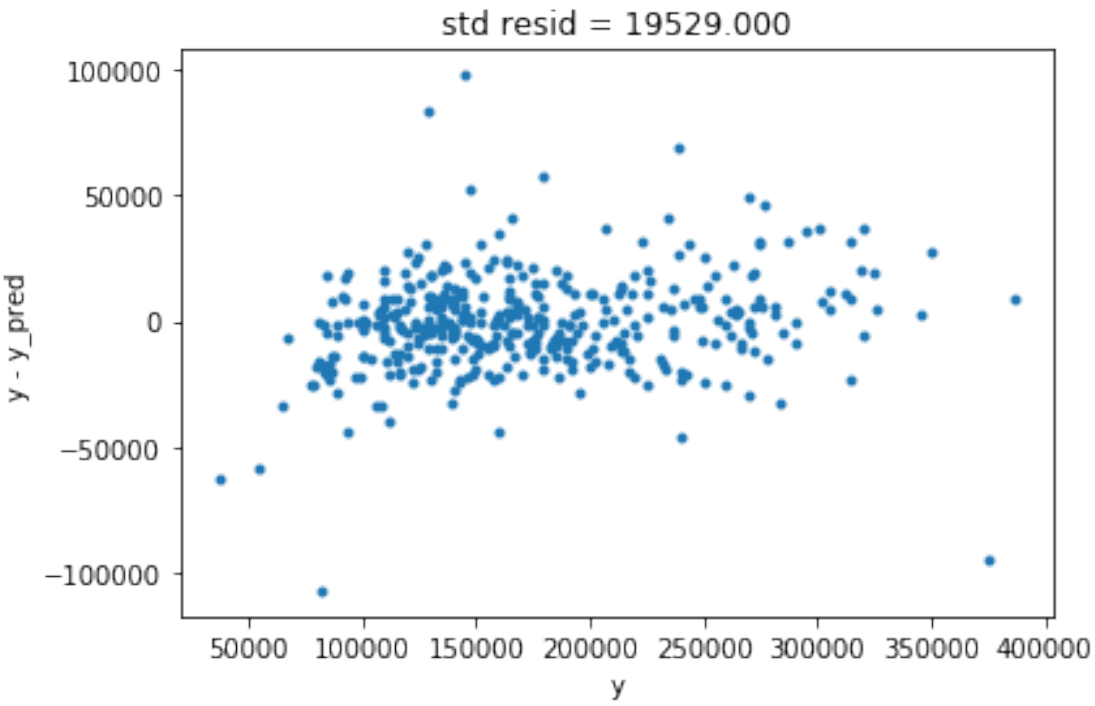

```
In [18]: y_pred = mdl.predict(x_test)
         fig = plt.figure(figsize=(15,8))
         fig.suptitle('Neural Networks Results without Outliers')
         plt.subplot(121)
         plt.scatter((y_pred), (y_test))
         plt.xlabel('Actual sale price')
         plt.ylabel('Predicted sale price')
         plt.title('Scatter plot of actual vs predicted sale price')
         plt.subplot(122)
         plt.subplots_adjust(bottom=0.25)
         plt.plot((y_pred), label='Predicted Ssale price')
         plt.plot((y_test.values), label='Actual sale price')
         plt.xlabel('House number')
         plt.ylabel('Predicted/Actual sale price')
         plt.title('line plot of actual and predicted sale price')
         plt.legend()
         plt.tight_layout()
         fig.subplots_adjust(top=0.88)
```



In [19]:
```
print("Corrolation between true and predicted value using NN on the actual sale price
      format(np.corrcoef(y_test,y_pred.squeeze())[0][1]))
```

```
Corrolation between true and predicted value using NN on the actual sale price is 0.9516045835
```

In [20]:
```
resid = y_test - y_pred.squeeze()
mean_resid = resid.mean()
std_resid = resid.std()
plt.plot(y_test,resid,'.')
plt.xlabel('y')
plt.ylabel('y - y_pred');
plt.title('std resid = {:.3f}'.format(std_resid));
```


std resid = 19529.000
100000
50000
0
−50000
−100000
y - y_pred
50000
100000
150000
200000
250000
300000
350000
400000
y

remove outliers and log chang y to improve performance

May 24, 2019

I try to remove some outliers in X for better performance. Also use log transfer y. Fit in Ridge and Lasso with clean data. Use grid search to improve performance.

In [1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

%matplotlib inline
```

In [2]:
```
clean_data = pd.read_csv('clean_data.csv')
clean_data.head()
```

Out[2]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	\
0	1	60	65.0	8450	7	5	2003	
1	2	20	80.0	9600	6	8	1976	
2	3	60	68.0	11250	7	5	2001	
3	4	70	60.0	9550	7	5	1915	
4	5	60	84.0	14260	8	5	2000	

	YearRemodAdd	ExterQual	ExterCond	...	SaleType_ConLI	\
0	2003	3	2	...	0	
1	1976	2	2	...	0	
2	2002	3	2	...	0	
3	1970	2	2	...	0	
4	2000	3	2	...	0	

	SaleType_ConLw	SaleType_New	SaleType_Oth	SaleType_WD	\
0	0	0	0	1	
1	0	0	0	1	
2	0	0	0	1	
3	0	0	0	1	
4	0	0	0	1	

	SaleCondition_AdjLand	SaleCondition_Alloca	SaleCondition_Family	\
0	0	0	0	
1	0	0	0	
2	0	0	0	

```
3                       0                       0                       0
4                       0                       0                       0

   SaleCondition_Normal  SaleCondition_Partial
0                     1                      0
1                     1                      0
2                     1                      0
3                     0                      0
4                     1                      0

[5 rows x 534 columns]
```

In [3]:
```
clean_data.columns
```

Out[3]:
```
Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'ExterQual', 'ExterCond',
       ...
       'SaleType_ConLI', 'SaleType_ConLw', 'SaleType_New', 'SaleType_Oth',
       'SaleType_WD', 'SaleCondition_AdjLand', 'SaleCondition_Alloca',
       'SaleCondition_Family', 'SaleCondition_Normal',
       'SaleCondition_Partial'],
      dtype='object', length=534)
```

In [4]:
```
y=clean_data['SalePrice']
clean_data=clean_data.drop(['SalePrice'],axis=1) #drop the y in clean_data
X = clean_data.iloc[:,1:] # drop the id column
X.shape
```

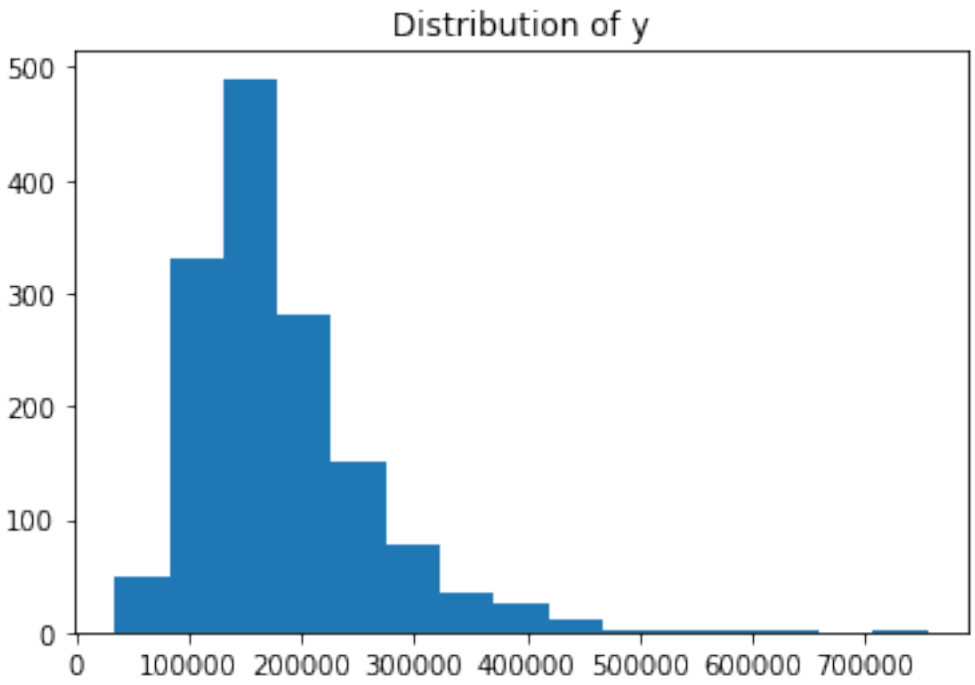
Out[4]:
```
(1459, 532)
```

In [5]:
```
y.describe()
```

Out[5]:
```
count      1459.000000
mean     180930.394791
std       79468.964025
min       34900.000000
25%      129950.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

In [6]:
```
plt.hist(y,bins=15)
plt.title('Distribution of y')
# Basically the first and last 9 bins have little samples,
#it's hard to predict these small groups of data,so drop it first.
```

Out[6]:
```
Text(0.5,1,'Distribution of y')
```



In [7]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
# scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_sc = scaler.fit_transform(X_train)
X_test_sc = scaler.transform(X_test)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/preprocessing/data.py:625: DataC
  return self.partial_fit(X, y)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/base.py:462: DataConversionWarni
  return self.fit(X, **fit_params).transform(X)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:8: DataConversionW
```

In [8]:
```
clf = Lasso(random_state=42)
clf.fit(X_train_sc,y_train)
y_test_pred = clf.predict(X_test_sc)
y_train_pred = clf.predict(X_train_sc)
from sklearn.metrics import mean_squared_error
from math import sqrt
mse = mean_squared_error(y_test, y_test_pred)
sqrt(mse)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
```

Out[8]: 33069.93524702181

```
In [9]: # grid search to get the best hyperparameter.
        from sklearn.model_selection import GridSearchCV
        parameters = { 'max_iter':[1000,2000,5000],'alpha':[1, 10,100,1000,10000]}
        ls = Lasso(random_state=42)
        clf = GridSearchCV(ls, parameters, cv=5)
        clf.fit(X_train_sc,y_train)
        y_test_pred = clf.predict(X_test_sc)
        y_train_pred = clf.predict(X_train_sc)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
```

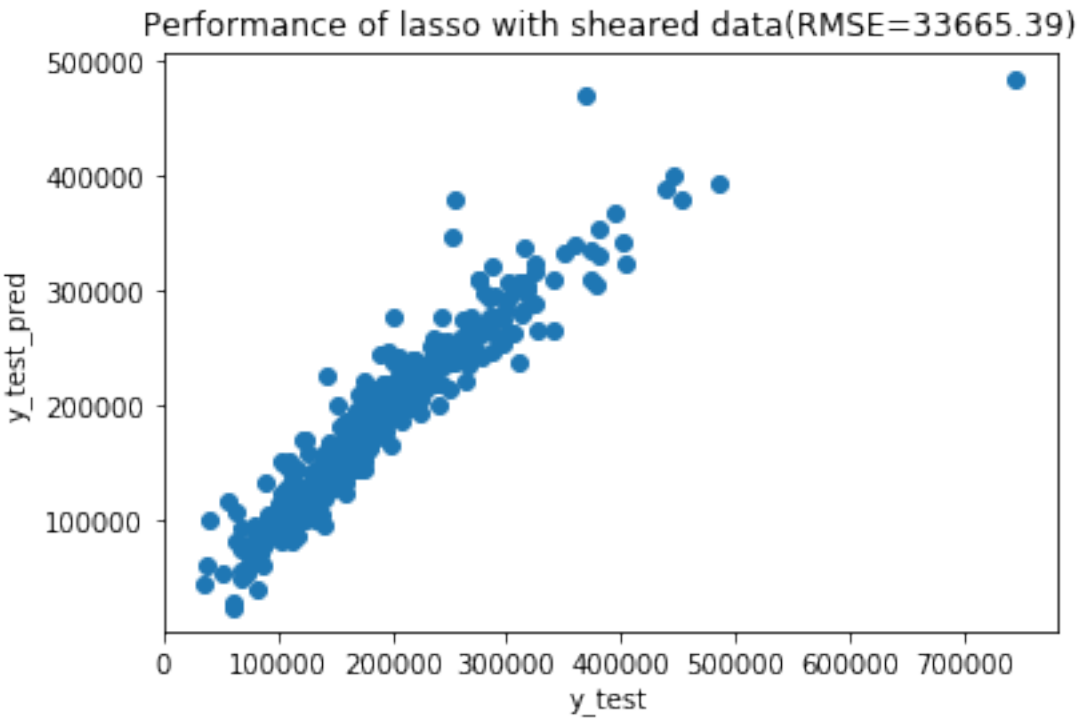
```
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_search.py:841: D
  DeprecationWarning)
```

```
In [10]: mse_test = mean_squared_error(y_test, y_test_pred)
         sqrt(mse_test)
```

```
Out[10]: 26802.87448312158
```

```
In [11]: plt.scatter(y_test, y_test_pred)
         plt.title('Performance of lasso with sheared data(RMSE=33665.39)')
         plt.xlabel('y_test')
         plt.ylabel('y_test_pred')
```

```
Out[11]: Text(0,0.5,'y_test_pred')
```




```
In [12]: clf.best_score_
         # R2
```

```
Out[12]: 0.8037315767569586
```

0.1 Remove outliers in X

```
In [13]: y_true_comb = np.hstack([y_train,y_test])
         len(y_true_comb)
```

```
Out[13]: 1459
```

```
In [14]: X_true_comb = np.vstack([X_train_sc,X_test_sc])
         X_true_comb = pd.DataFrame(X_true_comb,columns=X.columns)
```

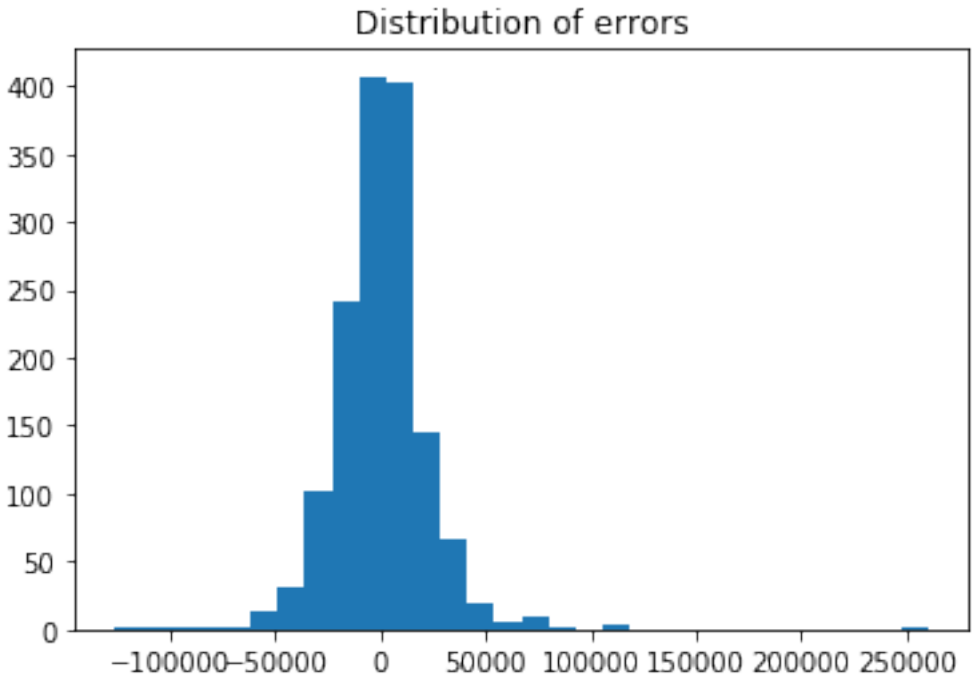
```
In [15]: y_pred = clf.predict(X_true_comb)
```

```
In [16]: error =y_true_comb-y_pred
         error
```

```
Out[16]: array([-14028.48554686,  18869.90785598,  31935.99323894, ...,
                 20869.21771686, -35026.38340765, -29035.94844561])
```

```
In [17]: plt.hist(y_true_comb-y_pred,bins = 30)
         plt.title('Distribution of errors')
```

```
Out[17]: Text(0.5,1,'Distribution of errors')
```




```
In [18]: strange_X = X_true_comb[error<-60000]
         strange_X
```

Out[18]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	\
700	0.052093	-1.673021	0.148034	-0.090550	1.328141	0.176179	
1064	-0.881004	1.196449	0.191906	1.376930	-0.494482	1.197285	
1094	-0.881004	0.259479	-0.175261	-0.824290	-0.494482	-0.647293	
1152	-0.881004	0.903646	0.075925	1.376930	-0.494482	1.098468	
1167	-0.881004	1.723495	0.265211	1.376930	-0.494482	1.131407	
1173	-0.881004	-1.673021	2.028113	-0.090550	-2.317105	-0.482598	
1218	0.285367	-0.209006	-0.194760	-1.558029	-1.405794	-1.701337	
1299	-0.881004	0.230199	1.331579	-0.824290	2.239452	-0.120270	

	YearRemodAdd	ExterQual	ExterCond	BsmtQual	...	\
700	0.492087	-0.692465	2.661743	0.577925	...	
1064	1.165509	2.756348	-0.217113	1.689151	...	
1094	-1.576280	-2.416871	-0.217113	-0.533302	...	
1152	1.021204	1.031941	-0.217113	1.689151	...	
1167	1.021204	1.031941	-0.217113	0.577925	...	
1173	-0.469944	1.031941	-0.217113	-0.533302	...	
1218	-1.672484	-0.692465	-3.095969	-0.533302	...	
1299	0.876900	-0.692465	-0.217113	-0.533302	...	

	SaleType_ConLI	SaleType_ConLw	SaleType_New	SaleType_Oth	SaleType_WD	\
700	-0.031311	-0.044302	-0.309020	-0.031311	0.393672	
1064	-0.031311	-0.044302	3.236033	-0.031311	-2.540188	
1094	-0.031311	-0.044302	-0.309020	-0.031311	0.393672	
1152	-0.031311	-0.044302	3.236033	-0.031311	-2.540188	
1167	-0.031311	-0.044302	-0.309020	-0.031311	0.393672	
1173	-0.031311	-0.044302	-0.309020	-0.031311	0.393672	
1218	-0.031311	-0.044302	-0.309020	-0.031311	0.393672	
1299	-0.031311	-0.044302	-0.309020	-0.031311	0.393672	

	SaleCondition_AdjLand	SaleCondition_Alloca	SaleCondition_Family	\
700	-0.062715	-0.076885	-0.126176	
1064	-0.062715	-0.076885	-0.126176	
1094	-0.062715	13.006409	-0.126176	
1152	-0.062715	-0.076885	-0.126176	
1167	-0.062715	-0.076885	-0.126176	
1173	-0.062715	13.006409	-0.126176	
1218	-0.062715	-0.076885	-0.126176	
1299	-0.062715	-0.076885	-0.126176	

	SaleCondition_Normal	SaleCondition_Partial
700	0.470417	-0.310918
1064	-2.125775	3.216278
1094	-2.125775	-0.310918
1152	-2.125775	3.216278
1167	0.470417	-0.310918
1173	-2.125775	-0.310918
1218	0.470417	-0.310918

```
1299             -2.125775               3.216278

[8 rows x 532 columns]
```

```
In [19]: strange_X = X_true_comb[error>80000]
         strange_X
```

Out[19]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	\
390	-0.881004	1.401411	0.442724	2.84441	-0.494482	1.197285	
446	0.052093	1.459972	0.301082	2.11067	-0.494482	1.197285	
491	0.402004	0.962207	1.134291	2.84441	3.150764	-2.623626	
554	-0.881004	0.083797	-0.232562	1.37693	0.416829	1.164346	
1316	0.052093	1.372131	0.266775	2.11067	-0.494482	1.197285	
1379	0.052093	3.011828	0.460384	2.84441	-0.494482	0.802018	

	YearRemodAdd	ExterQual	ExterCond	BsmtQual	...	\
390	1.117407	2.756348	-0.217113	1.689151	...	
446	1.165509	2.756348	-0.217113	1.689151	...	
491	0.395884	1.031941	2.661743	-0.533302	...	
554	1.069306	1.031941	-0.217113	0.577925	...	
1316	1.165509	2.756348	-0.217113	1.689151	...	
1379	0.540189	1.031941	-0.217113	1.689151	...	

	SaleType_ConLI	SaleType_ConLw	SaleType_New	SaleType_Oth	SaleType_WD	\
390	-0.031311	-0.044302	-0.309020	-0.031311	0.393672	
446	-0.031311	-0.044302	3.236033	-0.031311	-2.540188	
491	-0.031311	-0.044302	-0.309020	-0.031311	0.393672	
554	-0.031311	-0.044302	3.236033	-0.031311	-2.540188	
1316	-0.031311	-0.044302	3.236033	-0.031311	-2.540188	
1379	-0.031311	-0.044302	-0.309020	-0.031311	0.393672	

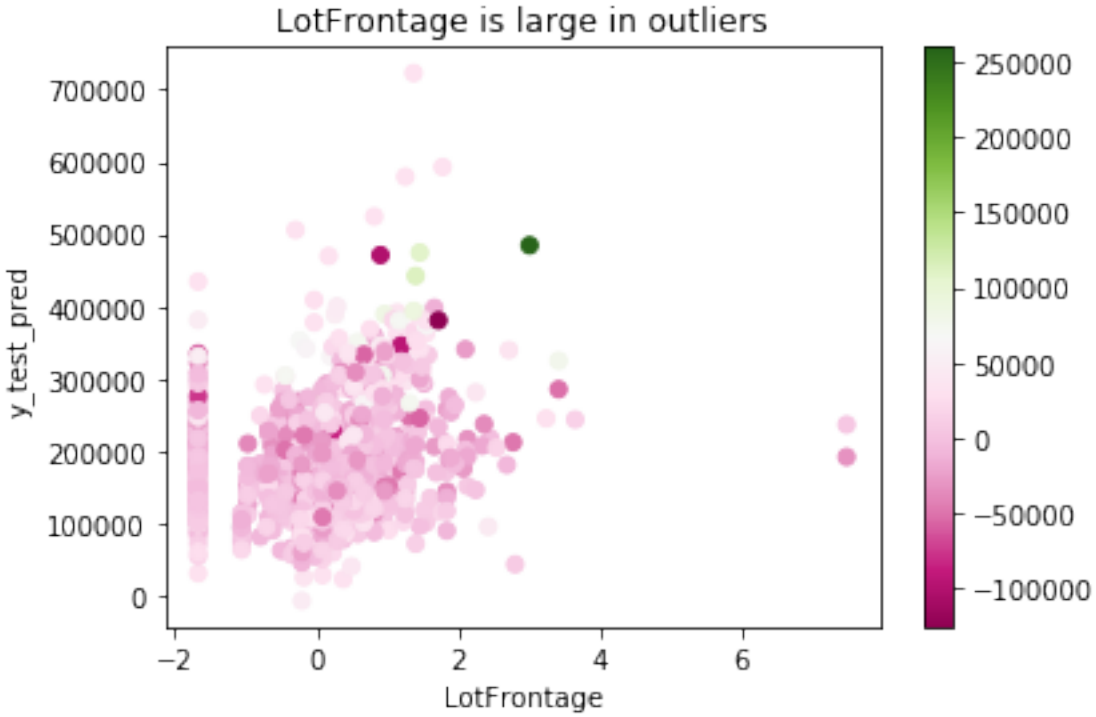
	SaleCondition_AdjLand	SaleCondition_Alloca	SaleCondition_Family	\
390	-0.062715	-0.076885	-0.126176	
446	-0.062715	-0.076885	-0.126176	
491	-0.062715	-0.076885	-0.126176	
554	-0.062715	-0.076885	-0.126176	
1316	-0.062715	-0.076885	-0.126176	
1379	-0.062715	-0.076885	-0.126176	

	SaleCondition_Normal	SaleCondition_Partial
390	0.470417	-0.310918
446	-2.125775	3.216278
491	0.470417	-0.310918
554	-2.125775	3.216278
1316	-2.125775	3.216278
1379	-2.125775	-0.310918

```
[6 rows x 532 columns]
```

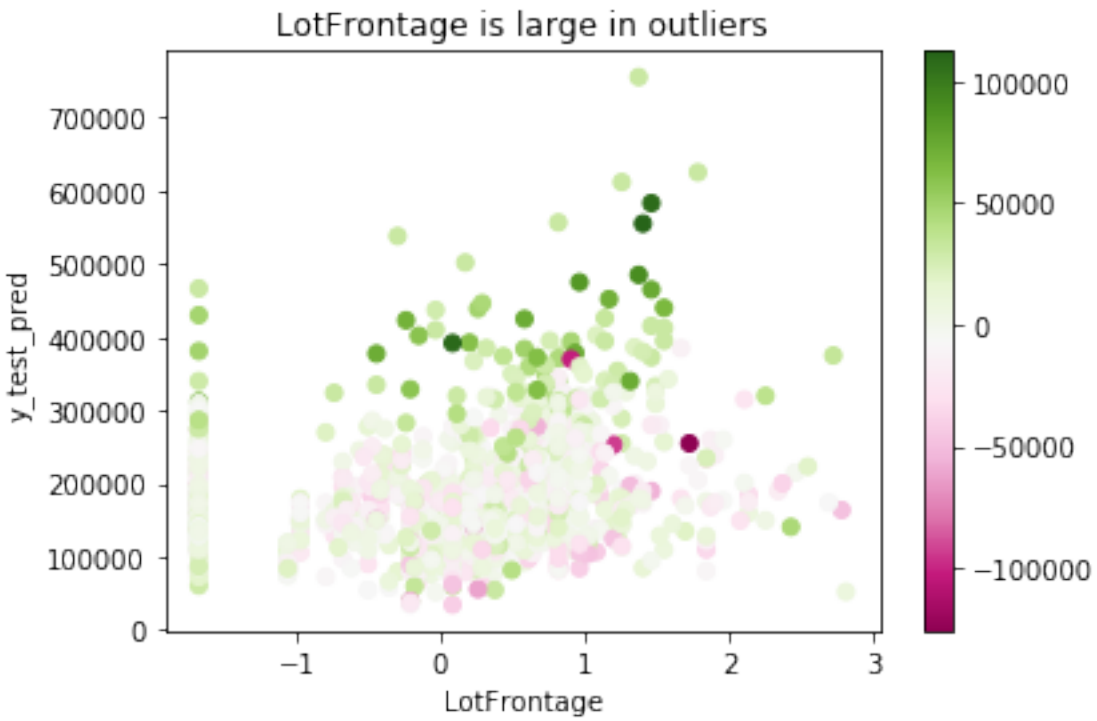
In [20]:
```
# identify outliers of X according to its feature
cm = plt.cm.get_cmap('PiYG')
sc = plt.scatter(X_true_comb.iloc[:,1],y_pred , c=error, cmap=cm) # LotFrontage:
                                                                  #Linear feet of s

plt.colorbar(sc)
plt.title('LotFrontage is large in outliers')
plt.xlabel('LotFrontage')
plt.ylabel('y_test_pred')
```

Out[20]: Text(0,0.5,'y_test_pred')



In [21]:
```
# remove according to LotFrontage.
X_filted = X_true_comb[X_true_comb['LotFrontage'] < 3]
y_filted = y_true_comb[X_true_comb['LotFrontage'] < 3]
error_filted = error[X_true_comb['LotFrontage'] < 3]
cm = plt.cm.get_cmap('PiYG')
sc = plt.scatter(X_filted.iloc[:,1],y_filted , c=error_filted, cmap=cm) # LotFrontage
                                                                        #Linear feet of s

plt.colorbar(sc)
plt.title('LotFrontage is large in outliers')
plt.xlabel('LotFrontage')
plt.ylabel('y_test_pred')
```

Out[21]: Text(0,0.5,'y_test_pred')




```
In [22]: X_train, X_test, y_train, y_test = train_test_split(X_filted, y_filted, test_size=0.3
         clf = Lasso(random_state=42)
         clf.fit(X_train,y_train)
         y_test_pred = clf.predict(X_test)
         y_train_pred = clf.predict(X_train)
         from sklearn.metrics import mean_squared_error
         mse = mean_squared_error(y_test, y_test_pred)
         sqrt(mse)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
```

```
Out[22]: 35317.15079537594
```

```
In [23]: parameters = { 'max_iter':[1000,2000,5000],'alpha':[1, 10,100,1000,10000]}
         ls = Lasso(random_state=42)
         clf = GridSearchCV(ls, parameters, cv=5)
         clf.fit(X_train,y_train)
         y_test_pred = clf.predict(X_test)
         y_train_pred = clf.predict(X_train)
         mse = mean_squared_error(y_test, y_test_pred)
         sqrt(mse)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
```

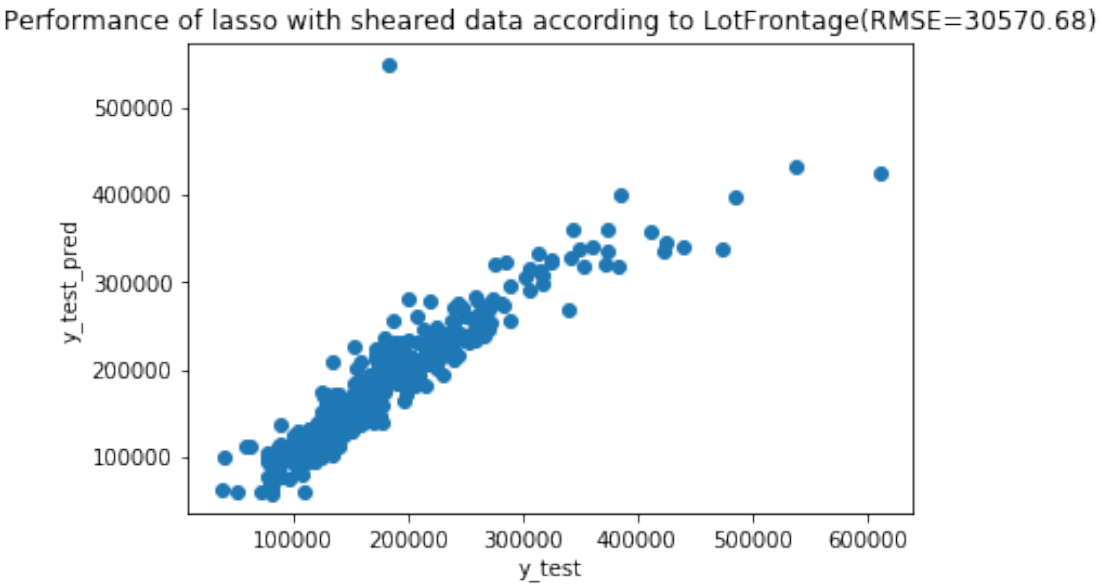
Out[23]: 30570.689624112565

```
In [24]: y.shape[0]-y_filted.shape[0] # 7 data points removed
```

Out[24]: 7

```
In [25]: plt.scatter(y_test, y_test_pred)
         plt.title('Performance of lasso with sheared data according to LotFrontage(RMSE=30570
         plt.xlabel('y_test')
         plt.ylabel('y_test_pred')
```

Out[25]: Text(0,0.5,'y_test_pred')

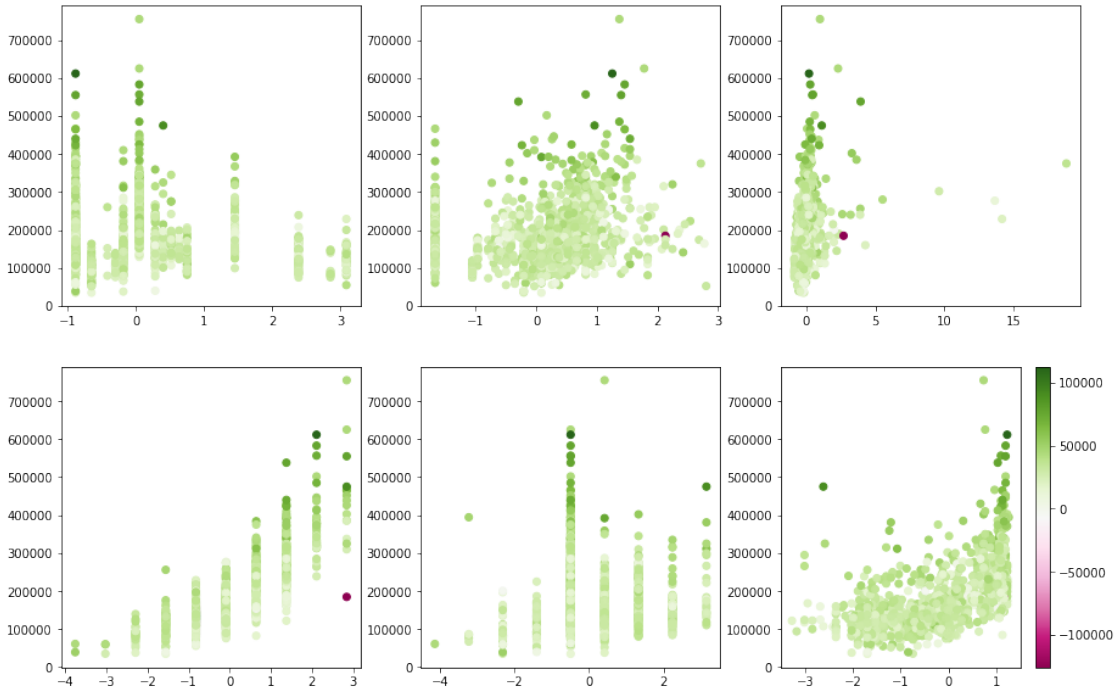


In [26]:
```
# combine data for some EDA and find more outliers
y_filted = np.hstack([y_train,y_test])
X_true_comb = np.vstack([X_train,X_test])
X_filted = pd.DataFrame(X_true_comb,columns=X.columns)
y_pred = clf.predict(X_filted)
error_filted =y_filted-y_pred
error_filted
```

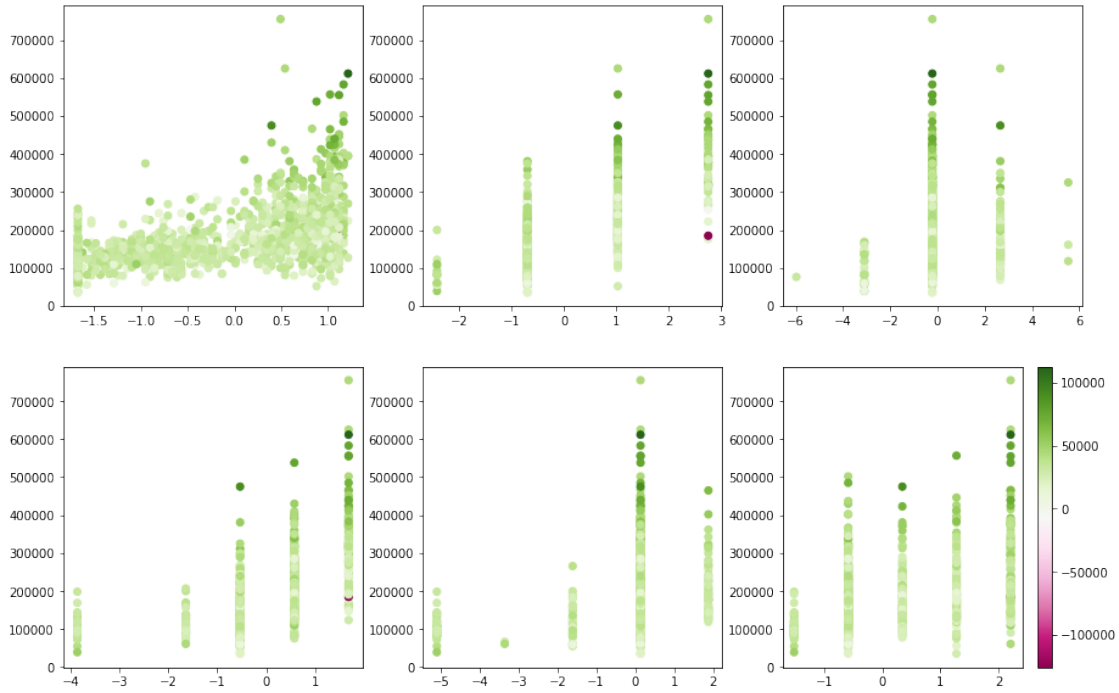
Out[26]:
```
array([ 5.21606840e+03, -9.47065676e+03,  5.37606815e+03, ...,
       -1.64250998e+02, -1.62202384e+01,  1.63888821e+04])
```

In [27]:
```
i = 0
fig, axes = plt.subplots(2, 3, figsize=(15,10))
axes[0, 0].scatter(X_filted.iloc[:,i+0],y_filted  , c=error_filted, cmap=cm)
axes[0, 1].scatter(X_filted.iloc[:,i+1],y_filted  , c=error_filted, cmap=cm)
axes[0, 2].scatter(X_filted.iloc[:,i+2],y_filted , c=error_filted, cmap=cm)
axes[1, 0].scatter(X_filted.iloc[:,i+3],y_filted  , c=error_filted, cmap=cm)
axes[1, 1].scatter(X_filted.iloc[:,i+4],y_filted  , c=error_filted, cmap=cm)
axes[1, 2].scatter(X_filted.iloc[:,i+5],y_filted  , c=error_filted, cmap=cm)
plt.colorbar(sc)
```

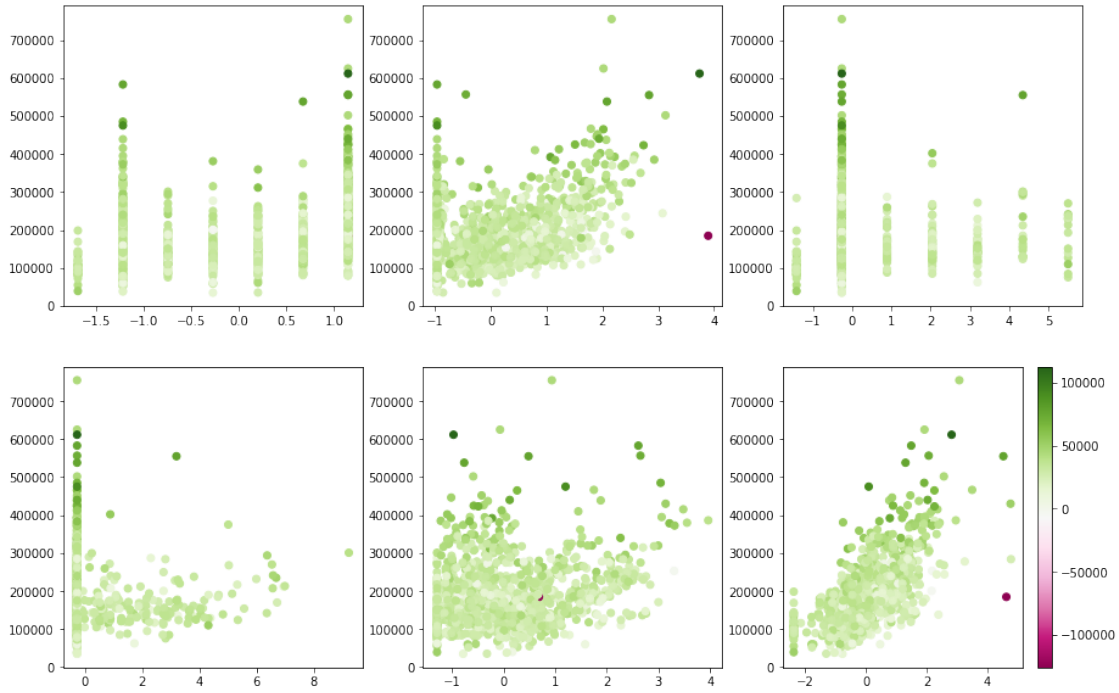
Out[27]:
```
<matplotlib.colorbar.Colorbar at 0x1a257a4b00>
```



```
In [28]: i = 6
         fig, axes = plt.subplots(2, 3, figsize=(15,10))
         axes[0, 0].scatter(X_filted.iloc[:,i+0],y_filted  , c=error_filted, cmap=cm)
         axes[0, 1].scatter(X_filted.iloc[:,i+1],y_filted  , c=error_filted, cmap=cm)
         axes[0, 2].scatter(X_filted.iloc[:,i+2],y_filted , c=error_filted, cmap=cm)
         axes[1, 0].scatter(X_filted.iloc[:,i+3],y_filted  , c=error_filted, cmap=cm)
         axes[1, 1].scatter(X_filted.iloc[:,i+4],y_filted  , c=error_filted, cmap=cm)
         axes[1, 2].scatter(X_filted.iloc[:,i+5],y_filted  , c=error_filted, cmap=cm)
         plt.colorbar(sc)

Out[28]: <matplotlib.colorbar.Colorbar at 0x1a13b234e0>
```




```
In [29]: i = 12
         fig, axes = plt.subplots(2, 3, figsize=(15,10))
         axes[0, 0].scatter(X_filted.iloc[:,i+0],y_filted  , c=error_filted, cmap=cm)
         axes[0, 1].scatter(X_filted.iloc[:,i+1],y_filted  , c=error_filted, cmap=cm)
         axes[0, 2].scatter(X_filted.iloc[:,i+2],y_filted , c=error_filted, cmap=cm)
         axes[1, 0].scatter(X_filted.iloc[:,i+3],y_filted  , c=error_filted, cmap=cm)
         axes[1, 1].scatter(X_filted.iloc[:,i+4],y_filted  , c=error_filted, cmap=cm)
         axes[1, 2].scatter(X_filted.iloc[:,i+5],y_filted  , c=error_filted, cmap=cm)
         plt.colorbar(sc)

Out[29]: <matplotlib.colorbar.Colorbar at 0x1a13df9198>
```



```
In [30]: # find some outliers in feature BsmtFinSF1
         import heapq
         print(heapq.nlargest(3, X_filted.iloc[:,13]))
```

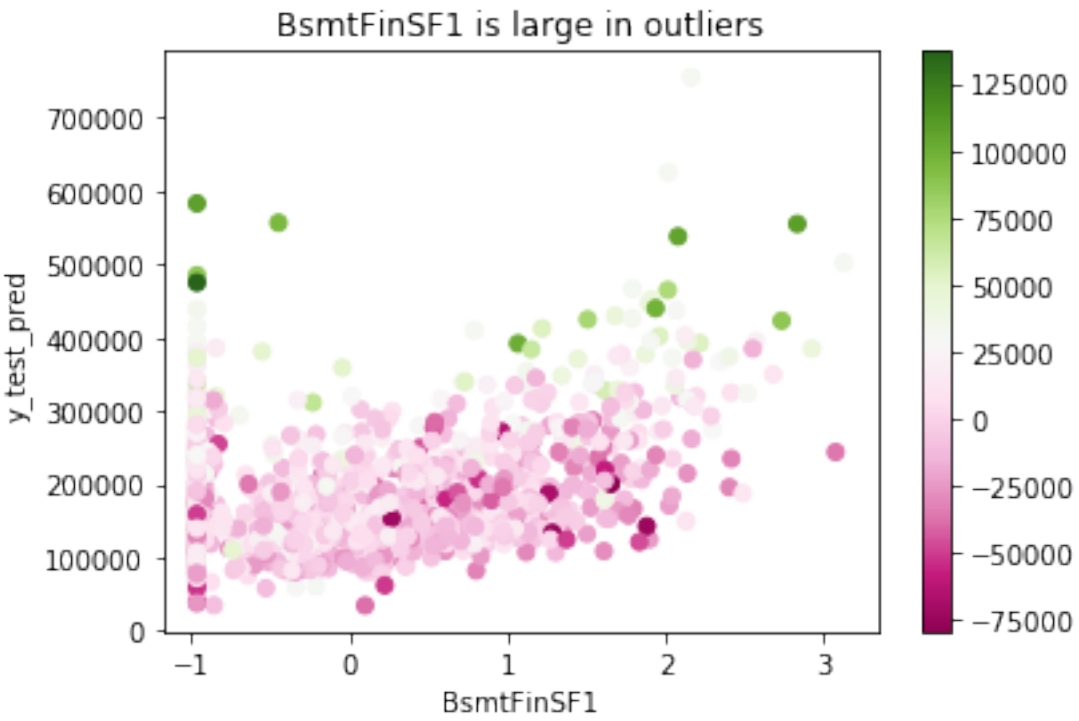
```
[3.8974066763514945, 3.7424466786490984, 3.131215576600758]
```

```
In [31]: X_filted.columns[13]
```

```
Out[31]: 'BsmtFinSF1'
```

```
In [33]: # remove according to BsmtFinSF1.
         X_filted2 = X_filted[X_filted['BsmtFinSF1'] < 3.6]
         y_filted2 = y_filted[X_filted['BsmtFinSF1'] < 3.6]
         error_filted2 = error_filted[X_filted['BsmtFinSF1'] < 3.6]
         cm = plt.cm.get_cmap('PiYG')
         sc = plt.scatter(X_filted2.iloc[:,13],y_filted2 , c=error_filted2, cmap=cm) # LotFron
                                                                                     #Linear feet of s

         plt.colorbar(sc)
         plt.title('BsmtFinSF1 is large in outliers')
         plt.xlabel('BsmtFinSF1')
         plt.ylabel('y_test_pred')
```

```
Out[33]: Text(0,0.5,'y_test_pred')
```



In [45]:
```
X_train, X_test, y_train, y_test = train_test_split(X_filted2, y_filted2, test_size=0

parameters = { 'max_iter':[1000,2000,5000],'alpha':[1, 10,100,1000,10000]}

ls = Lasso(random_state=42)

clf = GridSearchCV(ls, parameters, cv=5)

clf.fit(X_train,y_train)

y_test_pred = clf.predict(X_test)

y_train_pred = clf.predict(X_train)

mse = mean_squared_error(y_test, y_test_pred)

sqrt(mse)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
```

```
Out[45]: 25183.600856565372
```

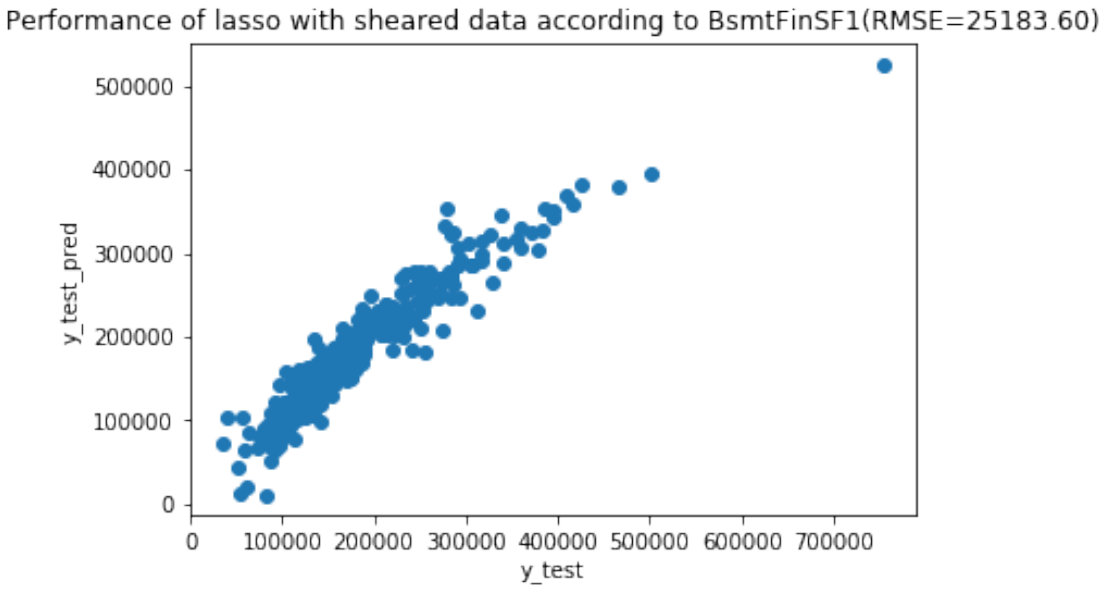
0.2 After remove the outliers in X , the root mean squared error droped to 25183.60

```
In [46]: y_filted.shape[0]-y_filted2.shape[0] # 2 data points removed
```

```
Out[46]: 2
```

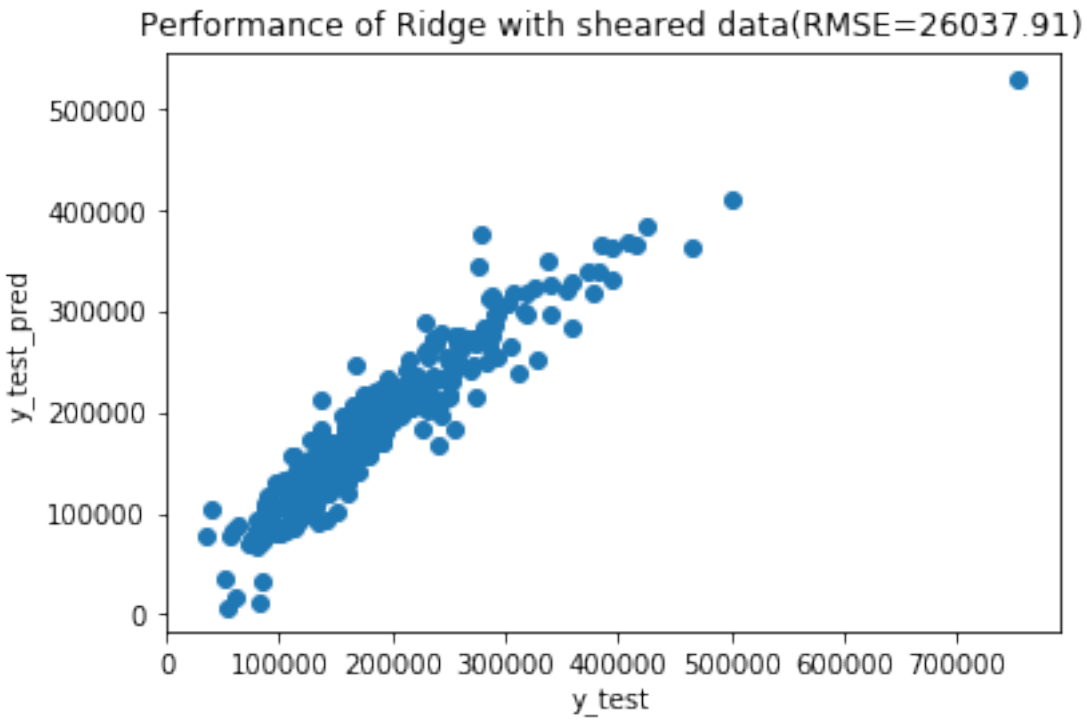
```
In [47]: plt.scatter(y_test, y_test_pred)
         plt.title('Performance of lasso with sheared data according to BsmtFinSF1(RMSE=25183.6
         plt.xlabel('y_test')
         plt.ylabel('y_test_pred')
```

```
Out[47]: Text(0,0.5,'y_test_pred')
```



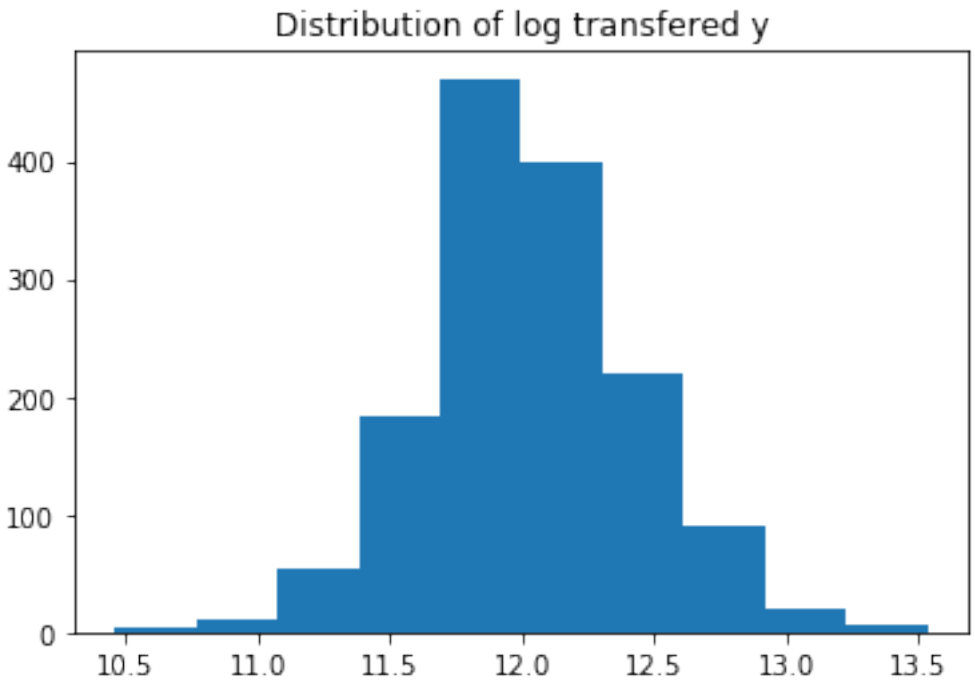
0.3 fit into Ridge

```
In [48]: parameters = { 'max_iter':[1000,2000,5000],'alpha':[1, 10,100,1000,10000]}
         from sklearn.linear_model import Ridge
         ls = Ridge(random_state=42)
         clf = GridSearchCV(ls, parameters, cv=5)
         clf.fit(X_train,y_train)
         y_test_pred = clf.predict(X_test)
         y_train_pred = clf.predict(X_train)

In [49]: mse_test = mean_squared_error(y_test, y_test_pred)
         sqrt(mse_test)

Out[49]: 26037.908039527552

In [50]: clf.best_score_

Out[50]: 0.8699814937005034

In [51]: plt.scatter(y_test, y_test_pred)
         plt.title('Performance of Ridge with sheared data(RMSE=26037.91)')
         plt.xlabel('y_test')
         plt.ylabel('y_test_pred')

Out[51]: Text(0,0.5,'y_test_pred')
```



0.4 log transfered y

```
In [52]: # log transfered y
         y_log = np.log(y)
         plt.hist(y_log)
         plt.title('Distribution of log transfered y')

Out[52]: Text(0.5,1,'Distribution of log transfered y')
```



0.5 fit into lasso

```
In [58]: X_train, X_test, y_train, y_test = train_test_split(X, y_log, test_size=0.3, random_st

         clf = Lasso(random_state=42)
         clf.fit(X_train,y_train)
         y_test_pred = clf.predict(X_test)
         y_train_pred = clf.predict(X_train)

         mse = mean_squared_error(y_test, y_test_pred)
         sqrt(mse)

Out[58]: 0.18036476431273618

In [63]: parameters = { 'max_iter':[100,500,1000],'alpha':[0.001,0.01,0.1,1, 10,100,1000]}
         ls = Lasso(random_state=42)
         clf = GridSearchCV(ls, parameters, cv=5)
         clf.fit(X_train,y_train)
         y_test_pred = clf.predict(X_test)
         y_train_pred = clf.predict(X_train)

/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.p
  ConvergenceWarning)
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_search.py:841: I
  DeprecationWarning)
```

```
In [64]: mse_test = mean_squared_error(y_test, y_test_pred)
         sqrt(mse_test)
```

```
Out[64]: 0.18036476431273618
```

```
In [65]: clf.best_params_
```

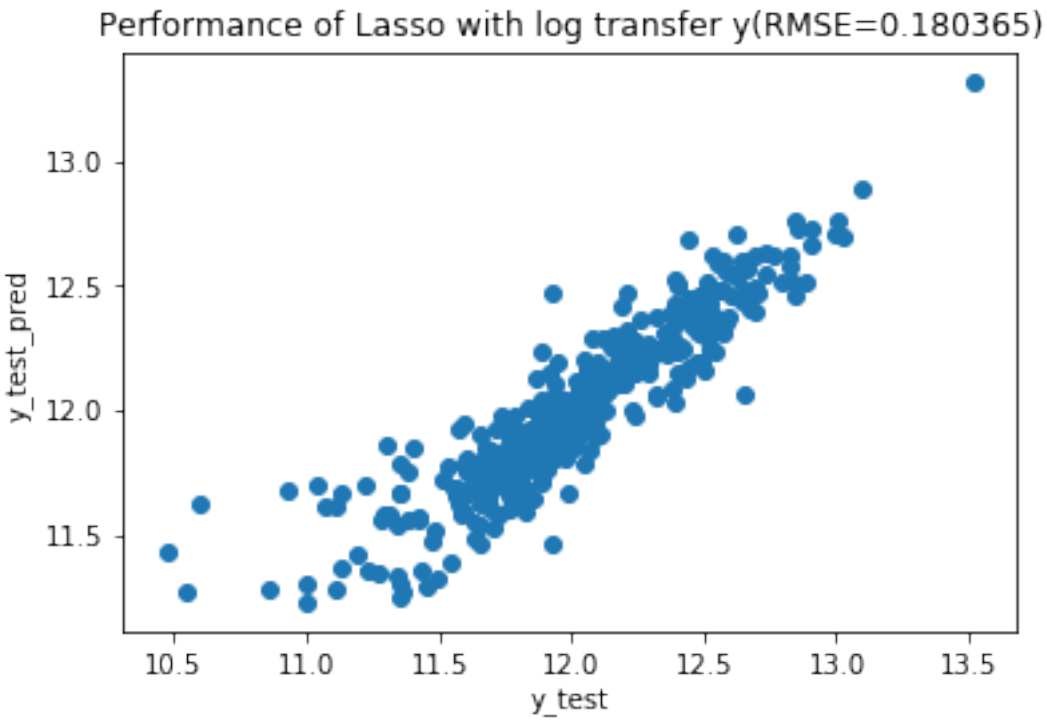
```
Out[65]: {'alpha': 1, 'max_iter': 500}
```

```
In [66]: clf.best_score_
```

```
Out[66]: 0.6492490027698465
```

```
In [68]: plt.scatter(y_test, y_test_pred)
         plt.title('Performance of Lasso with log transfer y(RMSE=0.180365)')
         plt.xlabel('y_test')
         plt.ylabel('y_test_pred')
```

```
Out[68]: Text(0,0.5,'y_test_pred')
```

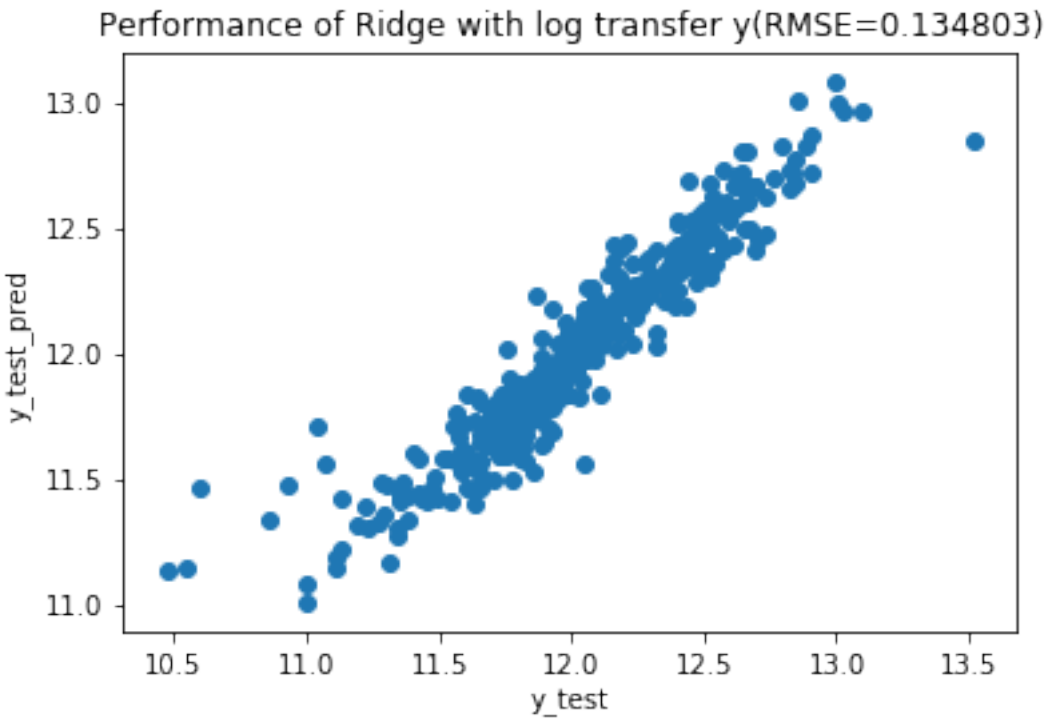


0.6 fit into Ridge

```
In [69]: parameters = { 'max_iter':[10,50,100,500],'alpha':[0.001,0.01,0.1,1, 10,100,1000]}
         rg = Ridge(random_state=42)
         clf = GridSearchCV(rg, parameters, cv=5)
         clf.fit(X_train,y_train)
         y_test_pred = clf.predict(X_test)
         y_train_pred = clf.predict(X_train)
```

```
/Users/fanwenyu/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_search.py:841: D
  DeprecationWarning)
```

```
In [70]: plt.scatter(y_test, y_test_pred)
         plt.title('Performance of Ridge with log transfer y(RMSE=0.134803)')
         plt.xlabel('y_test')
         plt.ylabel('y_test_pred')

Out[70]: Text(0,0.5,'y_test_pred')
```




```
In [71]: mse_test = mean_squared_error(y_test, y_test_pred)
         sqrt(mse_test)

Out[71]: 0.1375474209000953

In [72]: clf.best_score_

Out[72]: 0.6234102765983629
```

Conclusion

May 24, 2019

1 House Prices: Advanced Regression Techniques

1.1 Abstract

Predict sales prices with advanced regression techniques and practice feature engineering.

1.2 Introduction

This is our team's final project for Applied Machine Learning course. This team is consisted of Wenyu Fan, Tamer Ibranhim and Yaohua Chang.

1.3 Background

This dataset was constructed by Dean De Cock for use in Data Science education. It is viewed as a modern alternative to the Boston Housing dataset.

This dataset iseasy to understand so that we can put most time on studying and applying machine learning techniques rather than trying to understand it.

In addition, The community concerning this dataset is large so that we can study lots of brilliant ideas from there.

1.4 Data

Dataset from Kaggele:
https://www.kaggle.com/c/house-prices-advanced-regression-techniques

With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, this project challenges us to predict the final price of each home.

1.5 Method

1.5.1 Data Preprocessing

- Dealing with missing values
  - Filling LotFrontage NaN values using a linear regression model.
- Dealing with zeros
  Adding dummy variables for features with a lot of zeros to improve model fits.
- Encoding categorical variables

- Creating new features
  - Creation of features for different basement finish types.
- transform target variable into normal distribution
- Remove outliers

1.5.2 Exploratory Data Analysis

- Distribution of SalePrice for all the features
- Correlation between the features and SalePrice (or each other)

1.5.3 Feature Selection

As we can see there are many features in this dataset.Random forest can be used to do feature selection to remove least important features.

1.5.4 Fit and Optimise Models

To avoid repeating code I create a function to perform the fitting process for each model type I try. I use k-fold cross-validation to reduce the chances of overfitting the training data - specifically 5-fold cross-validation repeated 5 times.

The parameters of each model are optimised using a grid search, and the function returns the best model found and some stats on the model performance.

The models we tried are: * Linear Regression: * sklearn.linear_model.Ridge * sklearn.linear_model.Lasso * sklearn.linear_model.ElasticNet

- Support Vector Machines:
  - sklearn.svm.SVR
- Tree Based:
  - sklearn.ensemble.RandomForestRegressor
- Neural Networks:
  - keras

1.6 Evaluation

1.6.1 Compare Model with actual sale price

Model	R square (R2)	Mean absolute error (MAS)
Lasso	0.883223	19150.15
Ridge	0.95590900	11291.2240
Elastic Net	0.91388614	15122.4778
SVM	0.860785143	18799.0263
Neural Networks	0.868273	19768.23857
Random Forest	0.98622979	5580.97214

1.6.2 Compare Model with log of the sale price

Model	R square (R2)	Mean absolute error (MAS)
Lasso	0.902168	0.124385
Ridge	0.96326875	0.05546042
Elastic Net	0.9473058	0.07037367
SVM	0.932746	0.091442938
Random Forest	0.9877892	0.03351962

1.7 Conclusion

- We applied sveral algorithms in this project including Lasso, Ridge, Elastic Net, SVM, Neural Networks, Random Forest.
- After removing outliers which identified by Ridge, the MAS has slightly been decreased.
- Using log of the sale price as target value to train models, the R2 has slightly been increased.
- Random Forest perfroms best among all models, then linear regression models, then SVM.
- Neural Network gives a very bad results, we should do feature engineering use some advanced tools to imporove the NN performance.