Purpose and Questions in EDA

> "Doing data analysis requires quite a bit of thinking and we believe that when you've completed a good data analysis, you've spent more time thinking than doing." - Roger Peng
>
> "I wouldn't say that knowing your data is the most difficult thing in data science, but it is time-consuming. Therefore, it's easy to overlook this initial step and jump too soon into the water."

1. **Descriptive** - "seeks to summarize a characteristic of a set of data"
2. **Exploratory** - "analyze the data to see if there are patterns, trends, or relationships between variables" (hypothesis generating)
3. **Inferential** - "a restatement of this proposed hypothesis as a question and would be answered by analyzing a different set of data" (hypothesis testing)
4. **Predictive** - "determine the impact on one factor based on other factor in a population - to make a prediction"
5. **Causal** - "asks whether changing one factor will change another factor in a population - to establish a causal link"
6. **Mechanistic** - "establish *how* the change in one factor results in change in another factor in a population - to determine the exact mechanism"

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
#import math
import statsmodels.formula.api as smf
from statsmodels.graphics.gofplots import qqplot
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
housing_data = pd.read_csv('../data/housing_price.csv')
```

In [4]:
```
housing_data.shape
```

Out[4]:
```
(1460, 81)
```

Data Snooping Bias

- Before we decide on what algorithm to use on the data set, we have understand a lot about the data. If we look at whole dataset, we might stuble upon some seemingly interesting patterns in data that leads you to select a particular kind of Machine Learning Algorithm. This kind of approach **may fail** as the ML model may not be able generalize well on unseen datasets.
    - The concept of understanding whole data (with out keeping testset aside) is know as Data Snooping.
    - The concept of unable to generalize well on unseen datasets **due to Data Snooping** is know as Data Snooping Bias.
- To avoid this "Data Snooping Bias", we have to split dataset into TRAIN, TEST. Do exploratory data analysis, train ML algorithm on TRAIN set, test the model on unseen TEST dataset. (kind of simulating production data)

Stratified Sampling

In [5]:
```
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size = 0.2, random_state=42)
for train_index, test_index in split.split(housing_data, housing_data['Neighborhood']):
    housing_train_set = housing_data.loc[train_index]
    housing_test_set = housing_data.loc[test_index]
```

In [6]:
```
housing_train_set.shape
```

Out[6]:
```
(1168, 81)
```

Random Sampling - Coded

In [7]:
```
# 80% of housing_data.shape[0] = 1168
max_index = housing_data.shape[0] - 1
numbers = max_index*80//100
train_indexs = np.linspace(start=0, stop=max_index, num=numbers, dtype=int)
train_indexs
```

Out[7]:
```
array([   0,    1,    2, ..., 1456, 1457, 1459])
```

In [8]:
```
test_indexs = [x for x in range(max_index) if x not in train_indexs]
print(len(test_indexs))
```

```
293
```

In [9]:
```python
housing_train_set = housing_data.loc[train_indexs]
housing_test_set = housing_data.loc[test_indexs]
```

In [10]:
```python
housing_test_set.head()
```

Out[10]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	Mo:
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
9	10	190	RL	50.0	7420	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
14	15	20	RL	NaN	10920	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	GdWo	NaN	0	
19	20	20	RL	70.0	7560	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0	
24	25	20	RL	NaN	8246	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0	

5 rows × 81 columns

In [11]:
```python
from sklearn.model_selection import ShuffleSplit
shuffleSplit = ShuffleSplit(n_splits=1, test_size = 0.2, random_state=42)
for train_index, test_index in shuffleSplit.split(housing_data):
    housing_train_set = housing_data.loc[train_index]
    housing_test_set = housing_data.loc[test_index]
```

In [12]:
```python
housing_train_set.shape
```

Out[12]: (1168, 81)

```
In [13]: housing_train_set.columns
```

```
Out[13]: Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

```
In [14]: #housing_data[housing_data.GarageYrBlt.notnull()]
housing_train_set.rename(columns={'1stFlrSF':'FstFlrSF', '2ndFlrSF':'SecndFlrSF', '3SsnPorch':'ThreeSsnPorch'}, \
                         inplace=True)
```

As part of data analysis we refine the data - below are some common activities we do.

- **Missing :** Check for missing or incomplete data, impute/fillna with appropriate data
- **Quality :** Check for duplicates, accuracy, unusual data.
- **Parse :** Prase existing data and create new fearures. e.g. Extract year and month from date
- **Convert :** Free text to coded value (LabelEncoder, One-Hot-Encoding or LabelBinarizer)
- **Derive** Derive new feature out of existing featre/fearues e.g. gender from title Mr. Mrs.
- **Calculate** percentages, proportion
- **Remove** Remove redundant or not so useful data
- **Merge** Merge multiple columns e.g. first and surname for full name
- **Aggregate** e.g. rollup by year, cluster by area
- **Filter** e.g. exclude based on location
- **Sample** e.g. extract a representative data
- **Summary** Pandas describe function or stats like mean

Missing data : By “missing” data we simply mean null or “not present for whatever reason”. Lets see if we can find the missing data in our data set either because it exists and was not collected or it never existed

In [15]:
```
housing_train_set.count().head()
```

Out[15]:
```
Id             1168
MSSubClass     1168
MSZoning       1168
LotFrontage     951
LotArea        1168
dtype: int64
```

In [16]:
```
housing_train_set.describe()
```

Out[16]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	...	WoodDeck!
count	1168.000000	1168.000000	951.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1162.000000	1168.000000	...	1168.0000
mean	730.904966	56.849315	70.343849	10689.642123	6.121575	5.584760	1970.965753	1984.897260	103.771945	446.023973	...	95.9469
std	425.369088	42.531862	24.897021	10759.366198	1.367619	1.116062	30.675495	20.733955	173.032238	459.070977	...	129.6859
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	...	0.0000
25%	360.750000	20.000000	59.000000	7587.250000	5.000000	5.000000	1953.000000	1966.000000	0.000000	0.000000	...	0.0000
50%	732.500000	50.000000	70.000000	9600.000000	6.000000	5.000000	1972.000000	1994.000000	0.000000	384.500000	...	0.0000
75%	1101.750000	70.000000	80.000000	11700.000000	7.000000	6.000000	2001.000000	2004.000000	166.000000	721.000000	...	168.0000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1378.000000	5644.000000	...	857.0000

8 rows × 38 columns

In [17]:
```
housing_train_set[housing_train_set.EnclosedPorch > 0].head()
```

Out[17]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal
638	639	30	RL	67.0	8777	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0
799	800	50	RL	60.0	7200	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0
380	381	50	RL	50.0	5000	Pave	Pave	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0
1445	1446	85	RL	70.0	8400	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0
113	114	20	RL	NaN	21000	Pave	NaN	Reg	Bnk	AllPub	...	0	NaN	MnPrv	NaN	0

5 rows × 81 columns

In [18]:
```
housing_train_set[housing_train_set.EnclosedPorch > 0][['EnclosedPorch', 'SalePrice' ]].corr()
```

Out[18]:

	EnclosedPorch	SalePrice
EnclosedPorch	1.000000	0.285982
SalePrice	0.285982	1.000000

In [19]:
```
housing_train_set.count()[['PoolQC','EnclosedPorch']]
```

Out[19]:
```
PoolQC              6
EnclosedPorch    1168
dtype: int64
```

In [20]:
```
type(housing_train_set.isnull().any())
```

Out[20]:
```
pandas.core.series.Series
```

In [21]:
```
null_cols = housing_train_set.isnull().any()
for x in null_cols.index:
    if(null_cols[x]):
        print(x)
```

```
LotFrontage
Alley
MasVnrType
MasVnrArea
BsmtQual
BsmtCond
BsmtExposure
BsmtFinType1
BsmtFinType2
Electrical
FireplaceQu
GarageType
GarageYrBlt
GarageFinish
GarageQual
GarageCond
PoolQC
Fence
MiscFeature
```

In [22]:
```
def getMissingDataFeatures(df):
    ser_counts = df.count()
    data_size = df.shape[0]
    data_missing_features = []
    for idx in ser_counts.index:
        if(ser_counts[idx] < data_size):
            data_missing_features.append(idx)
    return data_missing_features
```

In [23]:
```
"""place all these features in excel and get discriptions,
   see if any of them are useful in predecting house price"""
print(getMissingDataFeatures(housing_train_set))
```

```
['LotFrontage', 'Alley', 'MasVnrType', 'MasVnrArea', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Ele
ctrical', 'FireplaceQu', 'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageQual', 'GarageCond', 'PoolQC', 'Fence', 'MiscFeatur
e']
```

How can we handle these missing value?

* Drop these rows / columns? Use .dropna(how='any')
* Fill with a dummy value? Use .fillna(value=dummy)
* Impute the cell with the most recent value? Use .fillna(method='ffill')
* Interpolate the amount in a linear fashion? Use .interpolate()
* model based imputation

In [24]:
```
# dropna(axis=1, how='all') - Drop the columns where all elements are nan
# dropna(axis=1, how='any') - Drop the columns where any of the elements is nan
housing_train_set.dropna(axis=1, how='all').head()
```

Out[24]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal
254	255	20	RL	70.0	8400	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0
1066	1067	60	RL	59.0	7837	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
638	639	30	RL	67.0	8777	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0
799	800	50	RL	60.0	7200	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0
380	381	50	RL	50.0	5000	Pave	Pave	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0

5 rows × 81 columns

In [25]:
```
housing_train_set.shape
```

Out[25]: (1168, 81)

In [26]:
```
housing_train_set[['LotFrontage', 'Alley', 'MasVnrType', 'MasVnrArea', 'BsmtQual', \
                   'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', \
                   'Electrical', 'FireplaceQu', 'GarageType', 'GarageYrBlt', \
                   'GarageFinish', 'GarageQual', 'GarageCond', 'PoolQC', 'Fence', \
                   'MiscFeature']].dtypes
```

Out[26]:
```
LotFrontage     float64
Alley            object
MasVnrType       object
MasVnrArea      float64
BsmtQual         object
BsmtCond         object
BsmtExposure     object
BsmtFinType1     object
BsmtFinType2     object
Electrical       object
FireplaceQu      object
GarageType       object
GarageYrBlt     float64
GarageFinish     object
GarageQual       object
GarageCond       object
PoolQC           object
Fence            object
MiscFeature      object
dtype: object
```

Handling missing values in a numeric feature

```
* Look at the distribution of the data.
    * We can fill all numeric features with mean value. This is not right, we have to do some analysis before imputing/fil
lna. Below are two of them.
    * If data is normally distributed we impute/fillna with mean.
    * If data is partially normal distributed, there are outliers. Then go with median/mode.
```

In [27]:
```
housing_train_set.mean()[['LotFrontage','MasVnrArea', 'GarageYrBlt']]
```

Out[27]:
```
LotFrontage      70.343849
MasVnrArea      103.771945
GarageYrBlt    1978.662138
dtype: float64
```

In [28]:
```
# fill mean for the numeric features.
housing_train_set.fillna(value=housing_train_set.mean()[['LotFrontage','MasVnrArea', 'GarageYrBlt']]).head()
```

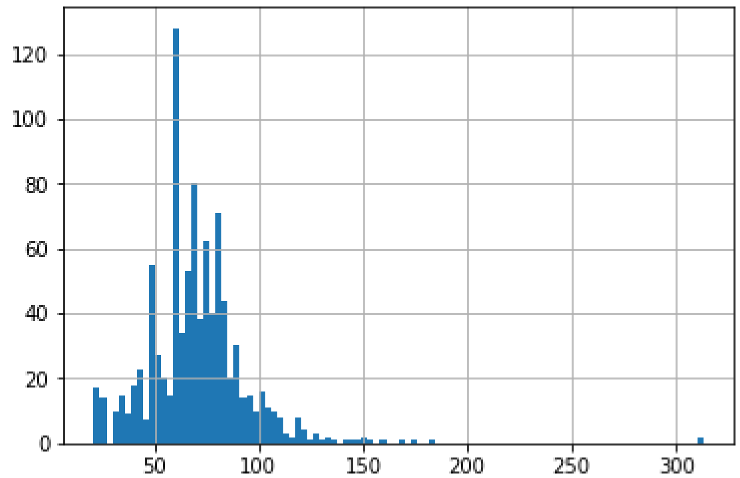
Out[28]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal
254	255	20	RL	70.0	8400	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0
1066	1067	60	RL	59.0	7837	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
638	639	30	RL	67.0	8777	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0
799	800	50	RL	60.0	7200	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0
380	381	50	RL	50.0	5000	Pave	Pave	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0

5 rows × 81 columns

In [29]:
```
housing_train_set['LotFrontage'].hist(bins=100)
```

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x2182305bdd8>



In [30]:
```
# Data is close to normal, around 350 items are close to 75, the mean is 70. Hence we can fillna with mean.
housing_train_set['LotFrontage'].mean()
```

Out[30]: 70.34384858044164

In [31]:
```python
housing_train_set.fillna(value=housing_train_set.mean()[['LotFrontage']], inplace=True).head()
```

Out[31]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal
254	255	20	RL	70.0	8400	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0
1066	1067	60	RL	59.0	7837	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
638	639	30	RL	67.0	8777	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0
799	800	50	RL	60.0	7200	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0
380	381	50	RL	50.0	5000	Pave	Pave	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0

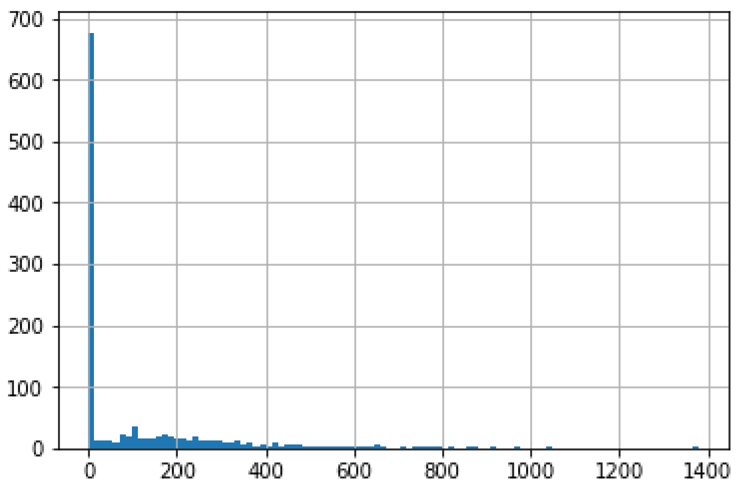
5 rows × 81 columns

In [32]:
```python
housing_train_set[housing_train_set.LotFrontage.isnull()].shape
```

Out[32]: (0, 81)

In [33]:
```python
housing_train_set['MasVnrArea'].hist(bins=100)
```

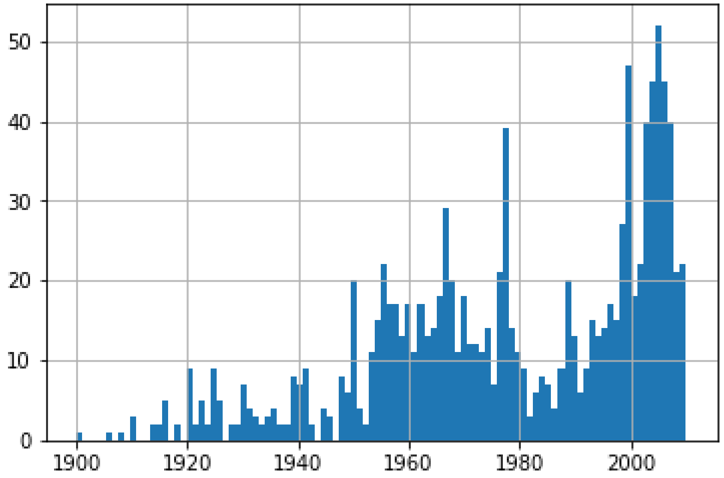
Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x21823648e10>



In [34]:
```python
"""most of the data is at zero, I think it is appropriate fillna with zero.
   Means there is no masonry(stone work) veneer area in square feet"""
housing_train_set['MasVnrArea'].fillna(value=0, inplace=True)
```

In [35]:
```
# GarageYrBlt - Year garage was built
housing_train_set['GarageYrBlt'].hist(bins=100)
```

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x218230811d0>



In [36]:
```
"""The distribution is not normal, hence I will try interpolation. pandas DataFrame uses index,
  feature as two variables.
We can change index if we think a feature is influencing the feature we have impute/fillna."""
interpol_dumb = housing_train_set['GarageYrBlt'].interpolate(method='nearest')
```

Finding a co-related feature with 'GarageYrBlt'

In [37]:
```
housing_train_set.corr().head()
```

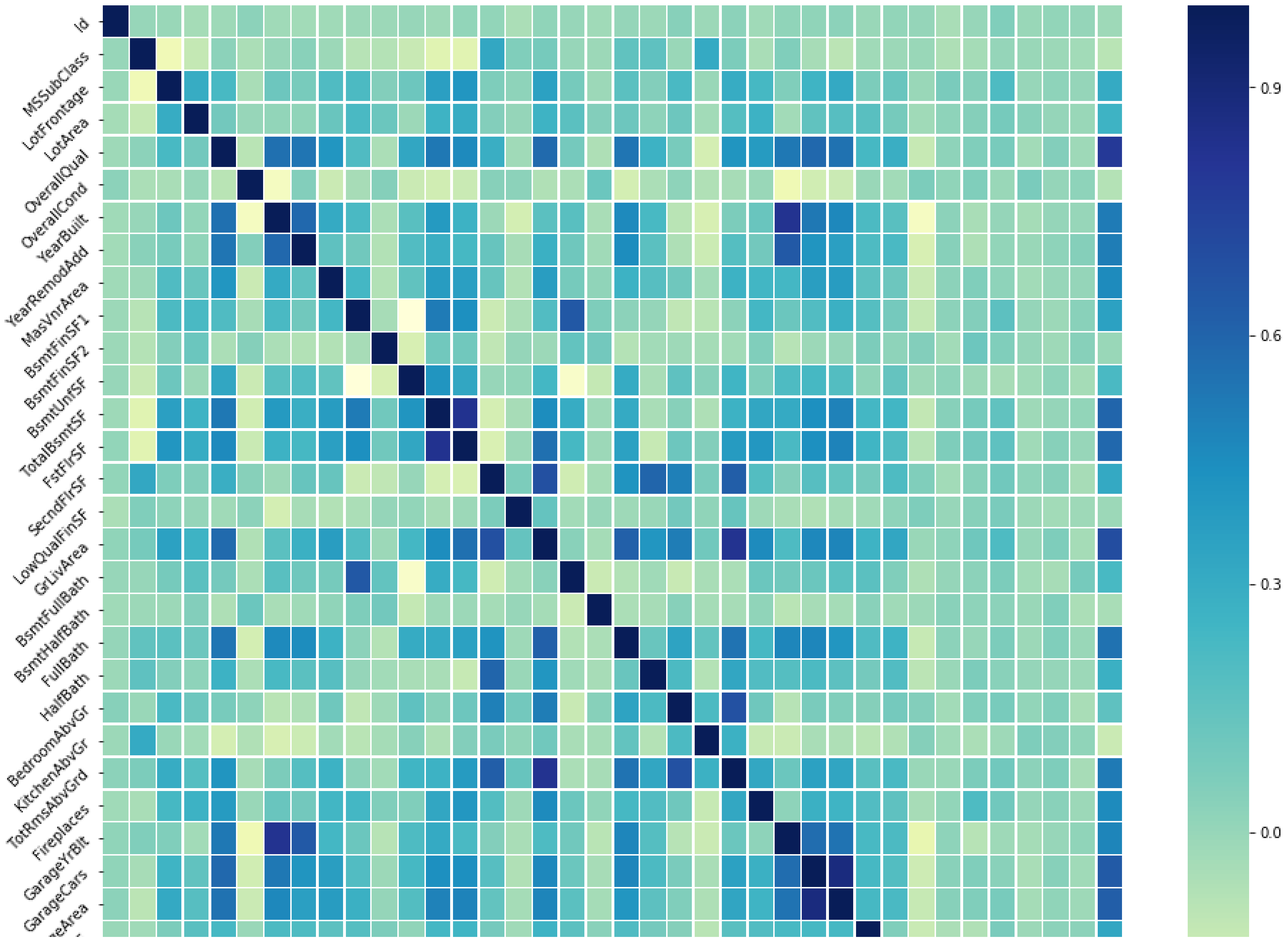
Out[37]:

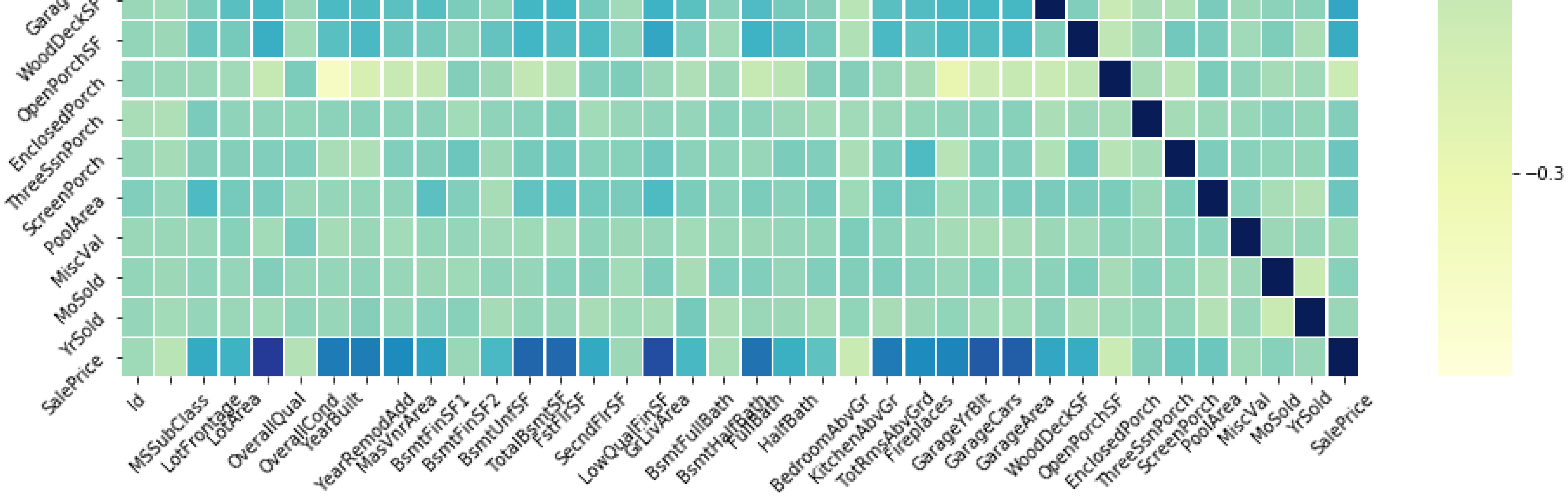
	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	...	WoodDeckSF
Id	1.000000	0.002216	-0.003732	-0.033411	-0.017880	0.029571	-0.022142	-0.032560	-0.024427	-0.011790	...	-0.005502
MSSubClass	0.002216	1.000000	-0.343054	-0.116501	0.029719	-0.052768	-0.001928	0.036081	-0.013443	-0.080944	...	-0.022712
LotFrontage	-0.003732	-0.343054	1.000000	0.299831	0.230090	-0.045974	0.117912	0.084473	0.202426	0.218827	...	0.067139
LotArea	-0.033411	-0.116501	0.299831	1.000000	0.102088	0.001625	0.013541	0.017216	0.126098	0.224270	...	0.177537
OverallQual	-0.017880	0.029719	0.230090	0.102088	1.000000	-0.087599	0.558124	0.538251	0.413083	0.204864	...	0.232991

5 rows × 38 columns

In [38]:

```
plt.figure(figsize=(16, 16))
sns.set_palette("PuBuGn_d")
g = sns.heatmap(housing_train_set.corr(), linewidths=.5, cmap="YlGnBu")
lst1 = g.set_yticklabels(g.get_yticklabels(), rotation = 45)
lst2 = g.set_xticklabels(g.get_xticklabels(), rotation = 45)
```





Alternative Way of finding a co-related feature with 'GarageYrBlt' (Optional)

In [39]:
```
housing_train_set.select_dtypes(['int64','float64']).columns
```

Out[39]:
```
Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1',
       'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'FstFlrSF', 'SecndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd',
       'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF',
       'OpenPorchSF', 'EnclosedPorch', 'ThreeSsnPorch', 'ScreenPorch',
       'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
```

In [40]:
```
housing_train_set.dtypes.head()
```

Out[40]:
```
Id               int64
MSSubClass       int64
MSZoning        object
LotFrontage    float64
LotArea          int64
dtype: object
```

In [41]:
```
def getNumericalNonNullFeatureNames(df, feature):
    ser_dtypes = df.dtypes
    #Feature names which have numeric data and have no nulls.
    num_col_idx = [x for x in ser_dtypes.index
                   if ser_dtypes[x] in ['int64','float64']
                                 and x not in [feature]
                                 and df[x].count() == len(df)]
    return num_col_idx
    #return housing_train_set.select_dtypes(['int64','float64']).notnull().any().index

# Above way is not appropriate as there will not be any correlation between Index and Feature. \
# We will try to a best correlated column for 'GarageYrBlt'. Then make it the index and then do interpolation.
def buildLinearModel_ExtractPvalues(df, feature):
    p_val_dict = dict()
    #Create a Data Frame whose rows have non-null values for feature.
    filtered_df = df[df[feature].notnull()]
    ser_dtypes = df.dtypes
    #Feature names which have numeric data and have no nulls.
    num_col_idx = getNumericalNonNullFeatureNames(df, feature)
    for col in num_col_idx:
        lm = smf.ols(formula=feature +'~' + col, data=filtered_df).fit()
        p_val_dict[col] = lm.pvalues[col]
    return p_val_dict
```

In [42]:
```
import scipy.stats as stats
num_col_idx = getNumericalNonNullFeatureNames(housing_train_set, 'GarageYrBlt')
df_no_nulls_in_GarageYrBlt = housing_train_set[housing_train_set['GarageYrBlt'].notnull()]
p_val_dict = dict()
for col in num_col_idx:
    p_val_dict[col] = stats.pearsonr(df_no_nulls_in_GarageYrBlt['GarageYrBlt'].values,\
                                     df_no_nulls_in_GarageYrBlt[col].values)
```

In [43]:
```
#print(type(p_val_dict.items()))
#p_val_dict = buildLinearModel_ExtractPvalues(housing_train_set, 'GarageYrBlt')
#print(sorted(p_val_dict.items(), key=lambda x: x[1]))
```

In [44]:
```
#for i, v in p_val_dict.items():
#    print(i,v)
```

In [45]:
```
sorted(p_val_dict.items(), key=lambda x: x[1][0], reverse=True)
```

Out[45]:
```
[('YearBuilt', (0.8195574824669863 4, 8.9298821973388776e-269)),
 ('YearRemodAdd', (0.6457929031433058 4, 2.713849725893654e-131)),
 ('GarageCars', (0.5696000039088112 7, 5.8575247586182823e-96)),
 ('GarageArea', (0.5478125153777975 2, 1.8052344354249254e-87)),
 ('OverallQual', (0.5269365563563007 7, 6.6669194383397063e-80)),
 ('SalePrice', (0.4803509120415035 4, 8.4164690364643652e-65)),
 ('FullBath', (0.4774155627329325 1, 6.3200236544603214e-64)),
 ('TotalBsmtSF', (0.3127353897775360 7, 1.7904606687109599e-26)),
 ('MasVnrArea', (0.2398144579102325 1, 6.6205486961810178e-16)),
 ('WoodDeckSF', (0.2268454444783164 8, 2.376845794253531e-14)),
 ('FstFlrSF', (0.22284546806202274, 6.8679605814749319e-14)),
 ('OpenPorchSF', (0.2167613230065945 1, 3.3186593085896137e-13)),
 ('BsmtUnfSF', (0.2088820150870825 4, 2.3828047575425634e-12)),
 ('GrLivArea', (0.2084559630173546 7, 2.6450096072745038e-12)),
 ('HalfBath', (0.1880055629062955, 3.0559712336546689e-10)),
 ('TotRmsAbvGrd', (0.1240728003382712 1, 3.5678837934327332e-05)),
 ('BsmtFinSF1', (0.1219843697191812 8, 4.8299645434296537e-05)),
 ('BsmtFullBath', (0.10616688031332469, 0.00041002174404686437)),
 ('MSSubClass', (0.05979566734139305, 0.046996995735141531)),
 ('LotFrontage', (0.05584269221863329 1, 0.063624440184977371)),
 ('SecndFlrSF', (0.05401933778720659 2, 0.072789219045095097)),
 ('Fireplaces', (0.02544044104751148 9, 0.39840358561688938)),
 ('ThreeSsnPorch', (0.020660195350301929, 0.49286356459739533)),
 ('YrSold', (0.014323828931309665, 0.63449058224277688)),
 ('Id', (0.014309910550617434, 0.63481985502113303)),
 ('MoSold', (-0.0010421666057988214, 0.97240801040972569)),
 ('PoolArea', (-0.017173009535622789, 0.56868138827937975)),
 ('LotArea', (-0.028553473297136173, 0.3432056445463163)),
 ('MiscVal', (-0.037059314651063206, 0.21855525851586496)),
 ('LowQualFinSF', (-0.045994559687516631, 0.1266826757132857)),
 ('ScreenPorch', (-0.081456702289641206, 0.0067699054249318264)),
 ('BedroomAbvGr', (-0.084272478138590276, 0.0050803806337824751)),
 ('BsmtFinSF2', (-0.084548833456039973, 0.0049370334901591246)),
 ('BsmtHalfBath', (-0.088031707632921924, 0.0034185693935061419)),
 ('KitchenAbvGr', (-0.13895890912894304, 3.5799470994920576e-06)),
 ('EnclosedPorch', (-0.2881046889666119, 1.5182271540993961e-22)),
 ('OverallCond', (-0.32856308416630775, 3.3449926666525943e-29))]
```

In [46]:
```
housing_train_set.index
```
Out[46]:
```
Int64Index([ 254, 1066,  638,  799,  380,  303,   86, 1385,  265,  793,
            ...
             330, 1238,  466,  121, 1044, 1095, 1130, 1294,  860, 1126],
           dtype='int64', length=1168)
```

In [47]:
```
housing_train_set.index = housing_train_set.YearBuilt
```

In [48]:
```
housing_train_set.head()
```
Out[48]:

YearBuilt	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	Misc
1957	255	20	RL	70.0	8400	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	
1993	1067	60	RL	59.0	7837	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	
1910	639	30	RL	67.0	8777	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	
1937	800	50	RL	60.0	7200	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	
1924	381	50	RL	50.0	5000	Pave	Pave	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	

5 rows × 81 columns

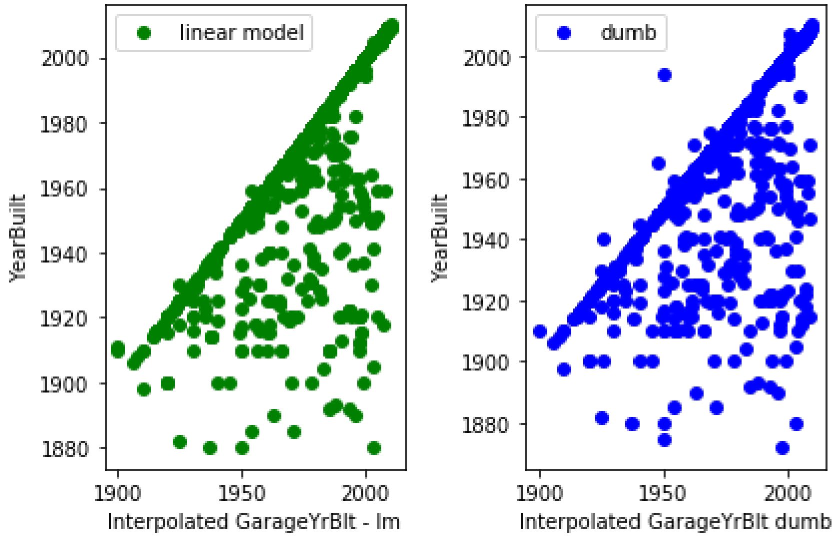
In [49]:
```
interpol_lm = housing_train_set['GarageYrBlt'].interpolate(method='nearest')
```

In [50]:
```
interpol_lm.head()
```
Out[50]:
```
YearBuilt
1957    1957.0
1993    1993.0
1910    1985.0
1937    1939.0
1924    1924.0
Name: GarageYrBlt, dtype: float64
```

In [51]:
```
plt.subplot(1,2,2)
# 1,2,1 - one row, tow columns, the first plot
plt.scatter(interpol_dumb, housing_train_set.YearBuilt, c='blue', label='dumb')
plt.xlabel('Interpolated GarageYrBlt dumb')
plt.ylabel('YearBuilt')
plt.legend()

plt.subplot(1,2,1)
# 1,2,2 - one row, tow columns, the second plot
plt.scatter(interpol_lm, housing_train_set.YearBuilt, c='green', label='linear model')
plt.xlabel('Interpolated GarageYrBlt - lm')
plt.ylabel('YearBuilt')
plt.legend()

plt.tight_layout()
```



In [52]:
```
"""np.seterr(divide='ignore', invalid='ignore')

f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)

graph_min = interpol_lm.index.min()
graph_max = interpol_lm.index.max()

ax1.hist(interpol_dumb, range=(graph_min, graph_max), bins=10, color='blue')
ax1.set_title('Interpolation lm/dumb')

ax2.hist(interpol_lm, range=(graph_min, graph_max), bins=10, color='green')

f.subplots_adjust(hspace=0.3)
plt.show()
"""
```

Out[52]:
```
"np.seterr(divide='ignore', invalid='ignore')\n\nf, (ax1, ax2) = plt.subplots(1, 2, sharey=True)\n\ngraph_min = interpol_lm.inde
x.min()\ngraph_max = interpol_lm.index.max()\n\nax1.hist(interpol_dumb, range=(graph_min, graph_max), bins=10, color='blue')\nax
1.set_title('Interpolation lm/dumb')\n\nax2.hist(interpol_lm, range=(graph_min, graph_max), bins=10, color='green')\n\nf.subplots
_adjust(hspace=0.3)\nplt.show()\n"
```

In [53]:
```
housing_train_set.reset_index(drop=True).head()
```

Out[53]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	Mo
0	255	20	RL	70.0	8400	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
1	1067	60	RL	59.0	7837	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
2	639	30	RL	67.0	8777	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0	
3	800	50	RL	60.0	7200	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0	
4	381	50	RL	50.0	5000	Pave	Pave	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	

5 rows × 81 columns

In [54]:
```
housing_train_set.reset_index(drop=True, inplace=True)
```

In [55]:
```
housing_train_set.index
```

Out[55]:
```
RangeIndex(start=0, stop=1168, step=1)
```

Handliing missing values in categorical features

- Ignore observations of missing values if we are dealing with large data sets and very few records have missing values. (unlikely approach)
- Ignore variable, if it is not significant.(unlikely approach)
- Treat missing data as just another category.
- **Replace with most frequent value.** (for now we will go with this apprach)
- Model based imputation: Build a model to predict missing values.

In [56]:
```
housing_train_set[housing_train_set['Alley'].isnull()].shape
```

Out[56]: (1094, 81)

In [57]:
```
def getNullPercentage(df, feature):
    null_count = len(df[df[feature].isnull()])
    percent_of_nulls = null_count*100/len(df)
    return null_count, percent_of_nulls

def isFeatureDropable(df, lst_featrues, threshold=75):
    sample_size = len(df)
    dict_drop_feat = dict()
    for feature in lst_featrues:
        null_count, percent_of_nulls = getNullPercentage(df, feature)
        print('Null count in {0} : {1}, Percent of Null: {2}'.format(feature, null_count, percent_of_nulls))
        if(percent_of_nulls > threshold):
            print('Drop --- {}'.format(feature))
            dict_drop_feat[feature] = True
        else:
            dict_drop_feat[feature] = False
    return dict_drop_feat

#print('Total Data size {0}'.format(sample_size))
#print('Missing data in Alley Feature {0}'.format(feature_missing_data_size))
#print('We could drop this feature as we have only {0}-{1} = {2} data points available'.format(sample_size, feature_missing_data_si
```

In [58]:
```
housing_train_set.dtypes.head()
```

Out[58]:
```
Id               int64
MSSubClass       int64
MSZoning        object
LotFrontage    float64
LotArea          int64
dtype: object
```

In [59]:
```
housing_train_set.select_dtypes(["object"]).columns
```

Out[59]:
```
Index(['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation',
       'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
       'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
       'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual',
       'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature',
       'SaleType', 'SaleCondition'],
      dtype='object')
```

In [60]:
```
def getCatFeaturesWithNulls(df):
    ser_dtypes = df.dtypes
    str_col_with_nulls = [x for x in ser_dtypes.index
                              if ser_dtypes[x] in ['object']
                                and len(df[df[x].notnull()]) < len(df)]
    return str_col_with_nulls

def getCatFeatures(df):
    ser_dtypes = housing_data.dtypes
    cat_features = [x for x in ser_dtypes.index
                              if ser_dtypes[x] in ['object']]
    return cat_features
```

In [61]:
```
dict_drop_feat = isFeatureDropable(housing_data, getCatFeaturesWithNulls(housing_train_set))
```

```
Null count in Alley : 1369, Percent of Null: 93.76712328767124
Drop --- Alley
Null count in MasVnrType : 8, Percent of Null: 0.547945205479452
Null count in BsmtQual : 37, Percent of Null: 2.5342465753424657
Null count in BsmtCond : 37, Percent of Null: 2.5342465753424657
Null count in BsmtExposure : 38, Percent of Null: 2.6027397260273974
Null count in BsmtFinType1 : 37, Percent of Null: 2.5342465753424657
Null count in BsmtFinType2 : 38, Percent of Null: 2.6027397260273974
Null count in Electrical : 1, Percent of Null: 0.0684931506849315
Null count in FireplaceQu : 690, Percent of Null: 47.26027397260274
Null count in GarageType : 81, Percent of Null: 5.5479452054794525
Null count in GarageFinish : 81, Percent of Null: 5.5479452054794525
Null count in GarageQual : 81, Percent of Null: 5.5479452054794525
Null count in GarageCond : 81, Percent of Null: 5.5479452054794525
Null count in PoolQC : 1453, Percent of Null: 99.52054794520548
Drop --- PoolQC
Null count in Fence : 1179, Percent of Null: 80.75342465753425
Drop --- Fence
Null count in MiscFeature : 1406, Percent of Null: 96.3013698630137
Drop --- MiscFeature
```

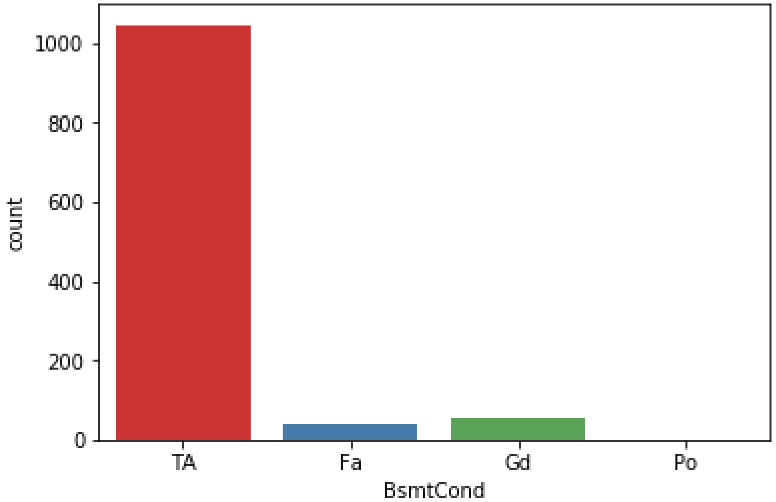
In [62]:
```
# Delete features with more nulls.
print(housing_train_set.shape)
for col, flag in dict_drop_feat.items():
    if flag:
        housing_train_set.drop(col, axis=1, inplace=True)
print(housing_train_set.shape)
```

```
(1168, 81)
(1168, 77)
```

In [63]:
```
#housing_data.drop('Alley', axis=1, inplace=True)
```

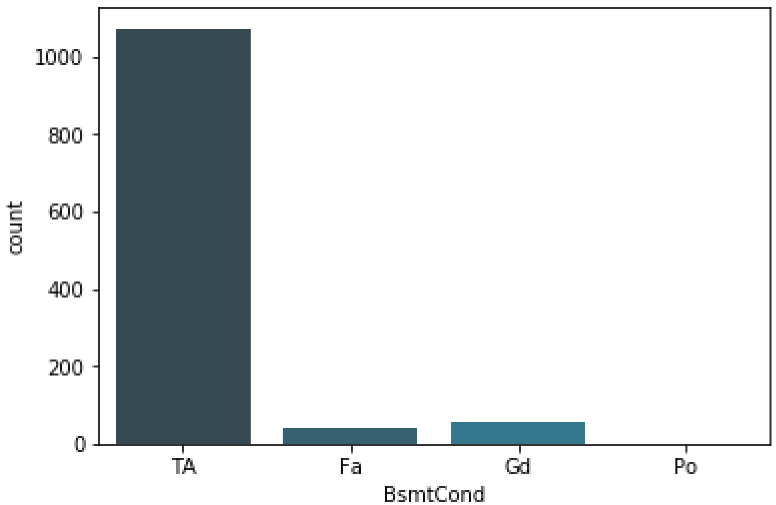
In [64]:
```
sns.countplot(x="BsmtCond", data=housing_train_set, palette="Set1")
```

Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x21823c325f8>



In [65]:
```
housing_train_set.BsmtCond.fillna('TA', inplace=True)
sns.countplot(x="BsmtCond", data=housing_train_set)
```

Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x21823b4f630>

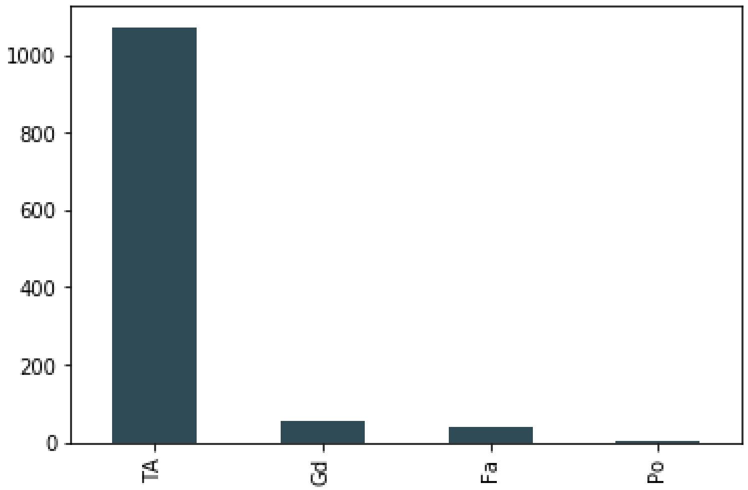


In [66]:
```
housing_train_set['BsmtCond'].value_counts()
```

Out[66]:
```
TA    1073
Gd      55
Fa      39
Po       1
Name: BsmtCond, dtype: int64
```

In [67]:
```
housing_train_set['BsmtCond'].value_counts().plot(kind='bar')
```

Out[67]: <matplotlib.axes._subplots.AxesSubplot at 0x21823cc4dd8>

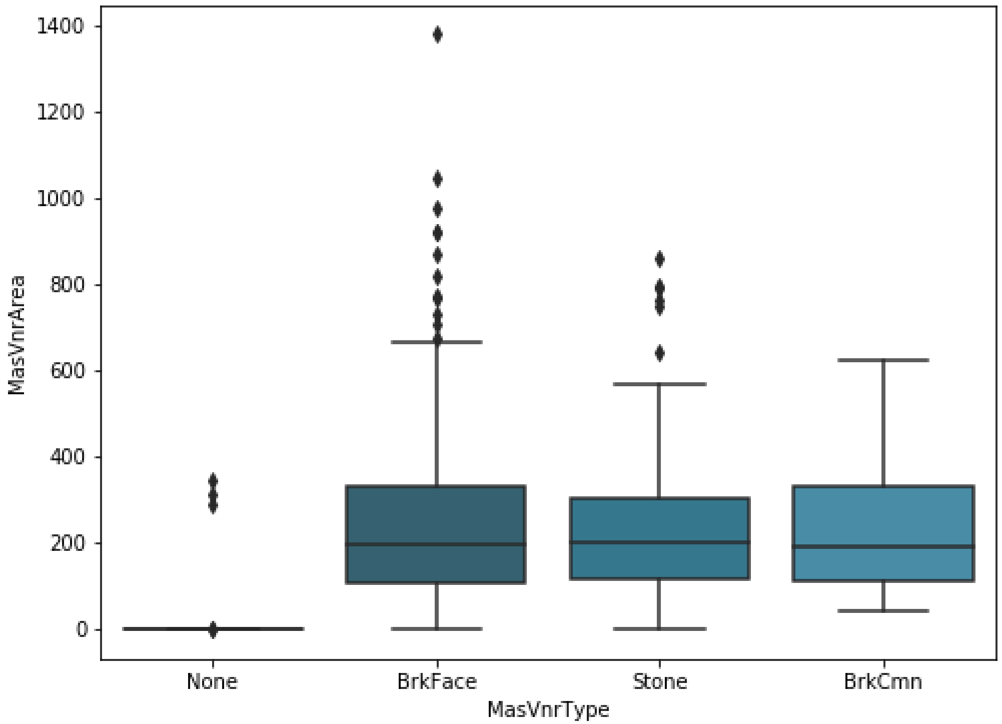


Check if there is any relation between above features and target variable (SalePrice). If the relation is strong enough, then keep the featrue and impute.

- Now let us see value counts of above features.

From below box wisker plot, value counts -> we can conclue that when the 'MasVnrType' = None, MAsVnrArea = 0. This is the pattern we figured out from above plot and aggriation. Hence we can build method (function) to impute data based this logic

In [68]:
```
var = 'MasVnrType'
data = pd.concat([housing_train_set['MasVnrArea'], housing_train_set[var]], axis=1)
f, ax = plt.subplots(figsize=(8, 6))
fig = sns.boxplot(x=var, y="MasVnrArea", data=data)
```



In [69]:
```
housing_train_set[(housing_train_set['MasVnrType'] == 'None') & \
                  (housing_train_set['MasVnrArea'] != 0)][['MasVnrType','MasVnrArea']]
```

Out[69]:

	MasVnrType	MasVnrArea
345	None	288.0
364	None	1.0
663	None	344.0
803	None	1.0
809	None	312.0

In [70]:
```
housing_train_set['MasVnrType'].value_counts()
```

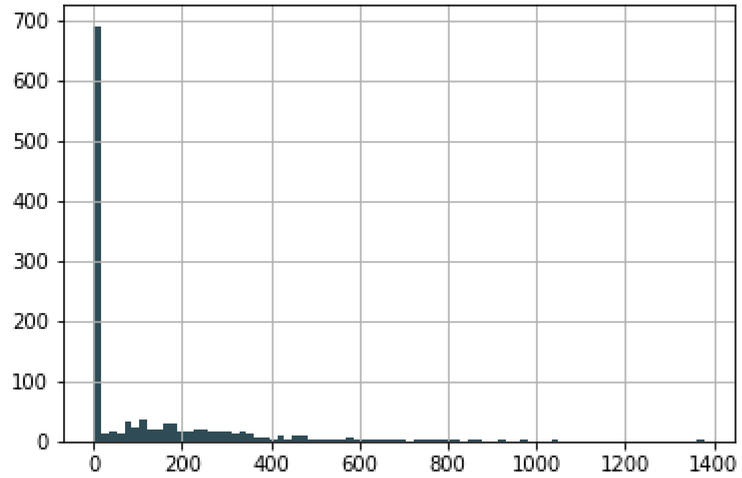
Out[70]:
```
None       677
BrkFace    366
Stone      106
BrkCmn      13
Name: MasVnrType, dtype: int64
```

In [71]:
```
housing_train_set.describe()['MasVnrArea']
```

Out[71]:
```
count    1168.000000
mean      103.238870
std       172.746354
min         0.000000
25%         0.000000
50%         0.000000
75%       166.000000
max      1378.000000
Name: MasVnrArea, dtype: float64
```

In [72]:
```
# have a look at MasVnrArea as Q1 - 0, Q2 - 0, Q3 - 164 and max - 1600
#housing_data['MasVnrArea'].value_counts()
# Masonry veneer area in square feet, more than 75% of the houses have zero of this.
# Need to check if this featrue has any effect on SalePrice ?
housing_train_set.MasVnrArea.hist(bins=80)
```

Out[72]: <matplotlib.axes._subplots.AxesSubplot at 0x21823be4898>



Understanding relationship between columns with different datatypes (Categorical to Categorical, Categorical to Quantitative, Quantitative to Quantitative. Before building any plot one need to identify below things.

- Identify the explanatory/independent (X-axis), response/dependent (Y-axis) variable for the analysis we are trying to do. Then find answers for below three questions.
- Q1) What is the type of the response variable (Categorical/Quantitative) ?
- Q2) If Categorical - How many categories are in this response variable ?. **To plot a bar graph to represent relation between two categorical variables, the represent variable must be collapsed to bi-variate (only two categories).**
- Q3) What is the type of explanatory variable (Categorical/Quantitative) ?

Categorical to Categorical relationship.

- Lets try to understand if there is any effect of "Severe Slope" (LandSlope feature) on SaleCondition
  - Exploratory/Independent Variable - LandSlope
  - Response/Dependent Variable - SaleCondition
- In this example we have to bring values in "SaleCondition" to two categories. **"Normal" - 1** vs other categories **"Partial", "Abnorml", "Family", "Alloca", "AdjLand" - 0**;

In [73]:
```
def mapSaleConditionToBinary(val):
    if(val == "Normal"):
        return 1
    else:
        return 0
```

In [74]:
```
housing_train_set["SaleCondition_binary"] = \
                housing_train_set.SaleCondition.apply(lambda x: mapSaleConditionToBinary(x))
```

In [75]:
```
housing_train_set["SaleCondition_binary"].value_counts()
```

Out[75]:
```
1    964
0    204
Name: SaleCondition_binary, dtype: int64
```

In [76]:
```
housing_train_set.SaleCondition.value_counts()
```

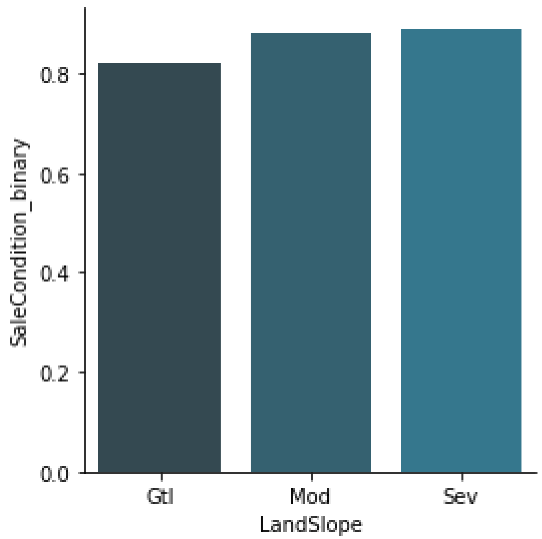
Out[76]:
```
Normal     964
Partial     98
Abnorml     77
Family      18
Alloca       7
AdjLand      4
Name: SaleCondition, dtype: int64
```

Categorical to Categorical bar chart explanation - bars explain the percentage of houses with SaleCondition - "Normal"

- **LandSlope - Glt bar :** around 82% of houses are with "Gentle slope"
- **LandSlope - Mod bar :** around 82% of houses are with "Moderate Slope"
- **LandSlope - Sev bar :** around 65% of houses are with "Severe Slope"

In [77]:
```
sns.factorplot(data=housing_train_set, x="LandSlope", y="SaleCondition_binary", kind="bar", ci=None)
```

Out[77]: <seaborn.axisgrid.FacetGrid at 0x21823dfcb38>



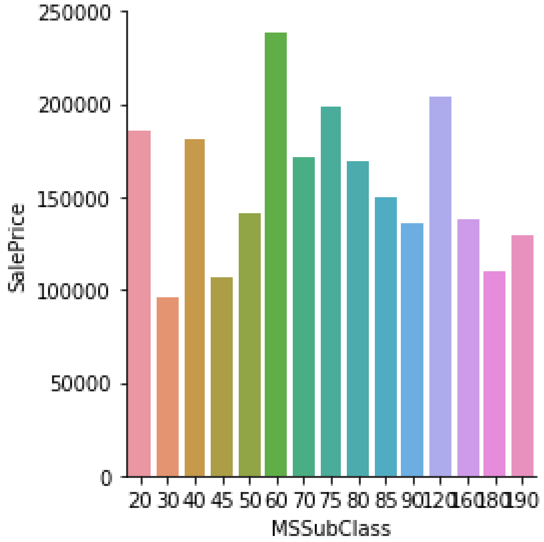
Categorical to Quantitative relationship

- Lets examin if **MSSubClass (type of dwelling)** has any effect on **SalePrice**.

- Explonatory/Independent Variable - MSSubClass
- Response/Dependent Variable - SalePrice

In [78]:
```
sns.factorplot(data=housing_train_set, x="MSSubClass", y="SalePrice", kind="bar", ci=None)
```

Out[78]: <seaborn.axisgrid.FacetGrid at 0x21823e270b8>



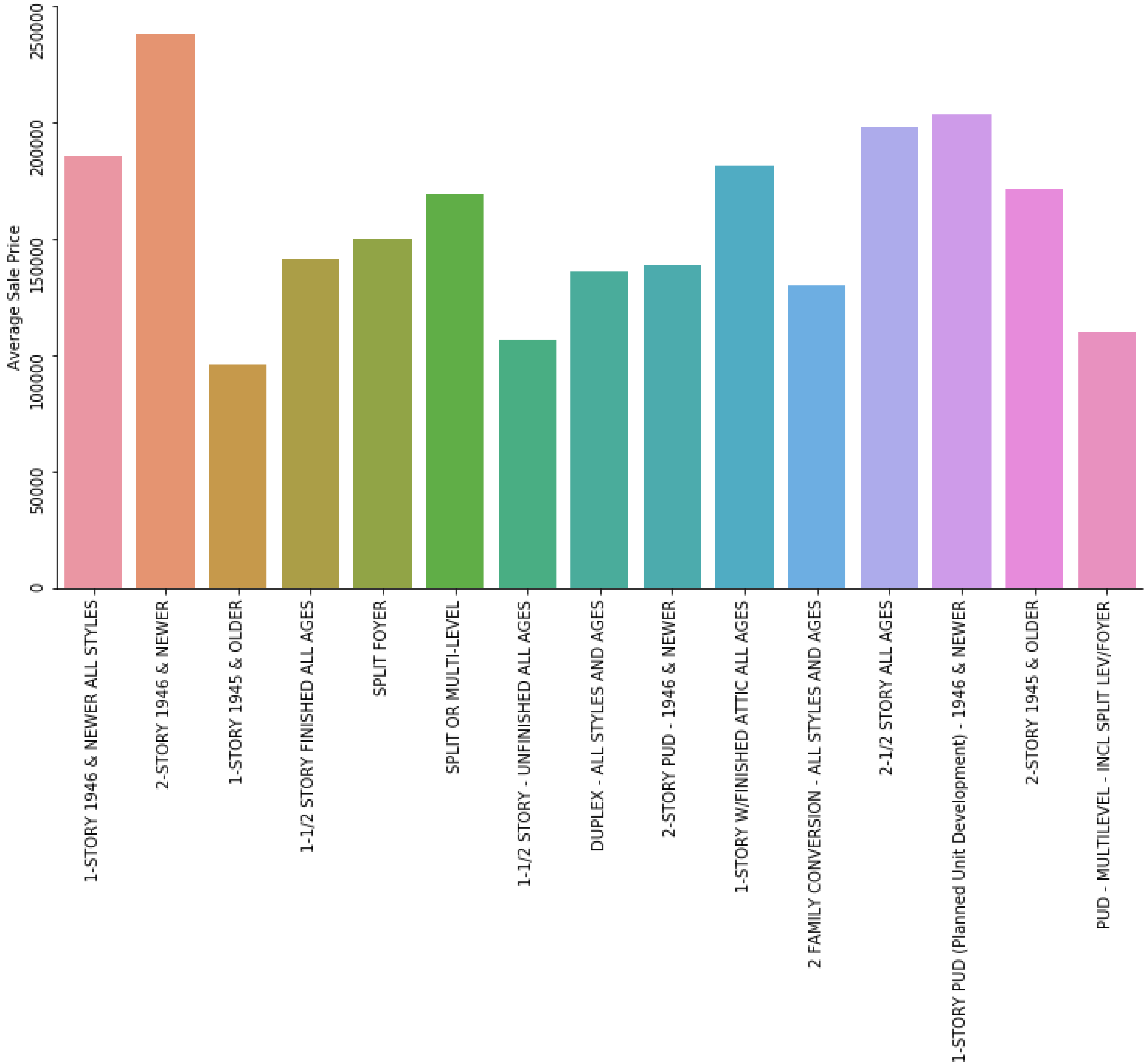
Categorical to Quantitative bar chart explanation : Y - axis scale will have "mean of the quantitative variable" as values

- **MSSubClass - 60** has the maximum average SalePrice.
- The numbers on X - axis are not so clear, lets map them the actual description.

In [79]:
```
mssub_class_dict = {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"}
housing_train_set["MSSubClass_mapped"] = housing_train_set["MSSubClass"].map(mssub_class_dict)
housing_train_set["MSSubClass_mapped"].value_counts()
```

Out[79]:
```
1-STORY 1946 & NEWER ALL STYLES                          434
2-STORY 1946 & NEWER                                     240
1-1/2 STORY FINISHED ALL AGES                            113
1-STORY PUD (Planned Unit Development) - 1946 & NEWER     64
2-STORY 1945 & OLDER                                      52
1-STORY 1945 & OLDER                                      50
2-STORY PUD - 1946 & NEWER                                49
SPLIT OR MULTI-LEVEL                                      45
DUPLEX - ALL STYLES AND AGES                              41
2 FAMILY CONVERSION - ALL STYLES AND AGES                 28
SPLIT FOYER                                               17
2-1/2 STORY ALL AGES                                      15
1-1/2 STORY - UNFINISHED ALL AGES                         10
PUD - MULTILEVEL - INCL SPLIT LEV/FOYER                    7
1-STORY W/FINISHED ATTIC ALL AGES                          3
Name: MSSubClass_mapped, dtype: int64
```

In [80]:
```
gr = sns.factorplot(data=housing_train_set, x="MSSubClass_mapped", y="SalePrice", \
                    kind="bar", ci=None,size=6, aspect=2)
plt.xlabel("House Type")
plt.ylabel("Average Sale Price")
l1 = gr.set_yticklabels( rotation = 90)
l2 = gr.set_xticklabels(rotation = 90)
```



Quantitative to Quantitative relationship can be understood using scatter plot. Lets understand a little more about correlation coefficient

- **Corelatoion coefficient explains strength of the linear relationship between two Quantitative Variables**. We may get a correlation coefficient **close to ZERO** when there is a **non linear relationship**. It is always suggestable to look at **scatter plot** along with correlation coefficient to understand strengh of relationship.

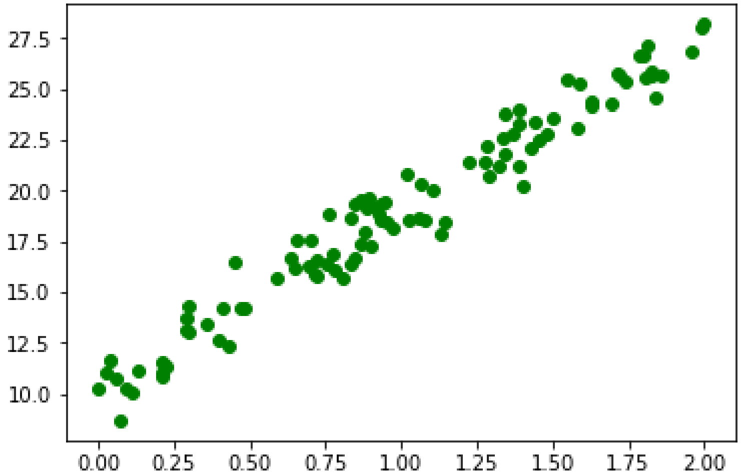
In [81]:
```
#import numpy as np

#X = 2 * np.random.rand(100, 1)
#y = 10 + 9 * X + np.random.randn(100, 1)
#dat = np.hstack((X,y))
#dff = pd.DataFrame(dat, columns=["input_feature", "output_feature"])
#dff.to_csv("../data/linear_set.csv", index=False)
```

In [82]:
```
linear = pd.read_csv("../data/linear_set.csv")
print("Correlation : ", linear.corr())
plt.scatter(linear.input_feature,linear.output_feature, c="green")
```

```
Correlation :                 input_feature  output_feature
input_feature        1.000000        0.978974
output_feature       0.978974        1.000000
```

Out[82]: <matplotlib.collections.PathCollection at 0x2182403a400>



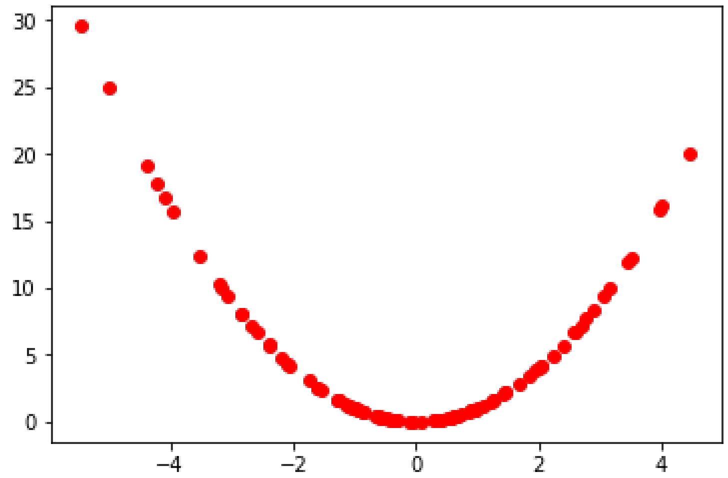
In [83]:
```
#import numpy as np

#X = 2 * np.random.randn(100, 1)
#y =  X**2
#dat = np.hstack((X,y))
#dff = pd.DataFrame(dat, columns=["input_feature", "output_feature"])
#dff.to_csv("../data/polynomial_set_new.csv", index=False)
```

In [84]:
```
ploy = pd.read_csv("../data/polynomial_set_new.csv")
print("Correlation : ", ploy.corr())
plt.scatter(ploy.input_feature,ploy.output_feature, c="red")
```

```
Correlation :                 input_feature  output_feature
input_feature        1.000000       -0.229165
output_feature      -0.229165        1.000000
```

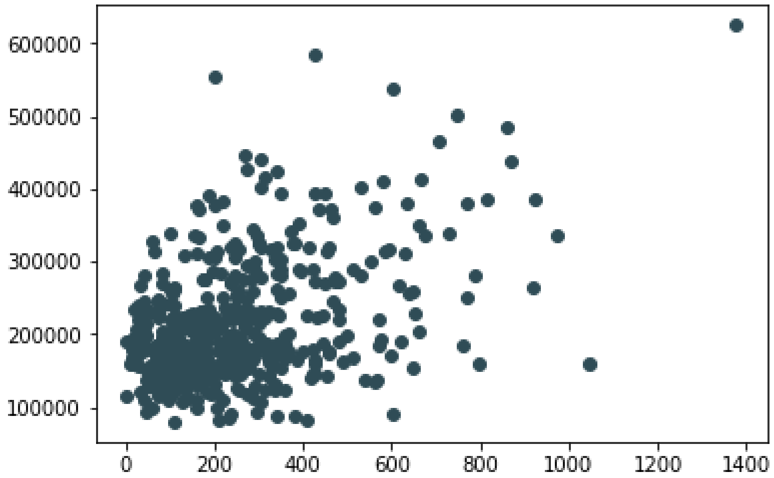
Out[84]:
```
<matplotlib.collections.PathCollection at 0x2182542cdd8>
```



In [85]:
```
df_nz_vnr_area = housing_train_set[housing_train_set.MasVnrArea != 0 & \
                                   housing_train_set.MasVnrArea.notnull()][['MasVnrArea','SalePrice']]
print('Shape of df_nz_vnr_area : ', df_nz_vnr_area.shape)
print('Correlation between MasVnrArea - SalePrice ')
print(df_nz_vnr_area.corr())
plt.scatter(df_nz_vnr_area.MasVnrArea,df_nz_vnr_area.SalePrice)
```

```
Shape of df_nz_vnr_area :  (488, 2)
Correlation between MasVnrArea - SalePrice
            MasVnrArea  SalePrice
MasVnrArea    1.000000   0.438302
SalePrice     0.438302   1.000000
```

Out[85]: <matplotlib.collections.PathCollection at 0x2182548bdd8>



- **Correlation between MasVnrArea - SalePrice is not significant and most of the data is zero or null**. Hence we can remove MasVnrArea, related feature MasVnrType.
- **Correlation coefficient is denoted with r, r^2 gives the perdictive power of variable to the other.**
- scipy.stats.pearsonr will give P-Value also.
- References:
  - [https://support.minitab.com/en-us/minitab-express/1/help-and-how-to/modeling-statistics/regression/supporting-topics/basics/a-comparison-of-the-pearson-and-spearman-correlation-methods/ (https://support.minitab.com/en-us/minitab-express/1/help-and-how-to/modeling-statistics/regression/supporting-topics/basics/a-comparison-of-the-pearson-and-spearman-correlation-methods/)](https://support.minitab.com/en-us/minitab-express/1/help-and-how-to/modeling-statistics/regression/supporting-topics/basics/a-comparison-of-the-pearson-and-spearman-correlation-methods/)
  - [https://statistics.laerd.com/statistical-guides/spearmans-rank-order-correlation-statistical-guide.php (https://statistics.laerd.com/statistical-guides/spearmans-rank-order-correlation-statistical-guide.php)](https://statistics.laerd.com/statistical-guides/spearmans-rank-order-correlation-statistical-guide.php)

In [86]:
```
housing_train_set.drop('MasVnrArea', axis=1, inplace=True)
housing_train_set.drop('MasVnrType', axis=1, inplace=True)
```

In [87]:
```
# I am going to drop rows with np.nan, I am doing this for training purposes.
# We have to handle all missing values in real-time
housing_train_set.dropna(axis=0, how='any', inplace=True)
```

In [88]:
```
lst_missing_features = getMissingDataFeatures(housing_train_set)
lst_missing_features
```

Out[88]: []

In [89]:
```
#im_show_mod.showImage('./bivariate_tools.png', 1000,400)
```

Bivariate Statistical Tools for Hypothesis Tests

		Response / Dependent Variable	
Explanatory / Independent Variable	Data Type	***Categorical***	***Quantitative***
	Categorical	Chi Square Test of Independence	Analysis of Variance (ANOVA)
	Quantitative	Chi Square Test of Independence (Derive 2 or more categories out of Qantitative variable)	Correlation

Relationship between Categorical and Quantitative Variable (ANOVA)

- ANOVA is statistical tool to examin differences in the **"mean of response variable"** for **"each category in explanatory variable"**

- **The null hypothesis H_0:** There is no relationship between explanatory and response variable. In other words, different categories in explanatory variable have no effect on mean of the corresponding response variable. OR ALL MEANS ARE SAME across different categories.
    - F-Statistic = $\frac{Variation\ Amoung\ Sample\ Means}{Variation\ With\ In\ Groups}$
    - If P-Value <= 0.05 we can reject null hypothesis. Hence can prove that *"each category has an effect on means of corresponding responsive variable"*.

ANOVA F-Test for a bivariate Categorical and Quantitative variable

- relationship between *CentralAir* and *SalePrice* before deciding on proper encoding.
- Explanatory Variable (Categorical) : "CentralAir" (Central air conditioning) - has got two categories "Y", "N".
- Response Variable (Quantitative) : "SalePrice"

In [90]:
```
import statsmodels.formula.api as smf

df1 = housing_train_set[["CentralAir", "SalePrice"]].dropna()
df1.head()
```

Out[90]:

	CentralAir	SalePrice
1	Y	178000
3	Y	175000
4	Y	127000
6	Y	174000
8	Y	175500

In [91]:
```
df1["CentralAir"].value_counts()
```

Out[91]:
```
Y    596
N     12
Name: CentralAir, dtype: int64
```

Below we can see a significant difference in means of SalePrice "with" and "with out" Central air conditioning.

In [92]:
```
df1.groupby("CentralAir").mean()
```
Out[92]:

	SalePrice
CentralAir	
N	152144.083333
Y	218819.771812

In [93]:
```
df1.groupby("CentralAir").std()
```
Out[93]:

	SalePrice
CentralAir	
N	52325.507269
Y	83282.559196

Lets see if the P-Value from ANOVA F-Test is also providing the same inference or not.

In [94]:
```
ols = smf.ols(formula='SalePrice ~ C(CentralAir)', data=df1).fit()
ols.summary()
```

Out[94]:

OLS Regression Results

Dep. Variable:	SalePrice	**R-squared:**	0.012
Model:	OLS	**Adj. R-squared:**	0.011
Method:	Least Squares	**F-statistic:**	7.623
Date:	Tue, 26 Jun 2018	**Prob (F-statistic):**	0.00594
Time:	08:00:27	**Log-Likelihood:**	-7747.0
No. Observations:	608	**AIC:**	1.550e+04
Df Residuals:	606	**BIC:**	1.551e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.521e+05	2.39e+04	6.363	0.000	1.05e+05	1.99e+05
C(CentralAir)[T.Y]	6.668e+04	2.41e+04	2.761	0.006	1.93e+04	1.14e+05

Omnibus:	218.145	**Durbin-Watson:**	1.947
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	811.568
Skew:	1.653	**Prob(JB):**	5.89e-177
Kurtosis:	7.594	**Cond. No.**	14.2

Prob (F-statistic): 0.00594, which is very close to zero and < 0.05. We can reject H_0 : hence there is some effect of "Central Air Condition" on "Sale Price".

ANOVA F-Test for a <u>Categorical variable with more than 2 categories and Quantitative variable</u>

- Need to analyze this between *Heating* and *SalePrice* before deciding on proper encoding.
- Explanatory Variable (Categorical) : "Heating" (Type of heating) - has got 6 categories "GasA", "GasW", "Grav", "Wall", "OthW", "Floor".
- Response Variable (Quantitative) : "SalePrice"

In [95]:
```python
df2 = housing_train_set[["Heating", "SalePrice"]].dropna()
```

In [96]:
```python
df2["Heating"].value_counts()
```

Out[96]:
```
GasA    598
GasW      8
OthW      1
Grav      1
Name: Heating, dtype: int64
```

Below are the means of SalePrice of different houses with different heating systems.

In [97]:
```python
means = df2.groupby("Heating").mean()
print(means)
```

```
             SalePrice
Heating
GasA     217722.297659
GasW     224234.875000
Grav     121000.000000
OthW     129500.000000
```

Lets calculate PAIR-WISE differences between mean Sale Price of houses with different Heating systems

In [98]:
```python
unseen = list(means.index)
for idx1 in means.index:
    unseen.remove(idx1)
    for idx2 in means.index :
        if(idx2 in unseen):
            dif = int(means.loc[idx1].values) - int(means.loc[idx2].values)
            if(abs(dif) < 15000):
                print("Difference Between \"{0}\" and \"{1}\" = \"{2}\" is LOW ".format(idx1, idx2, dif) )
            else:
                print("Difference Between \"{0}\" and \"{1}\" = \"{2}\" is HIGH ".format(idx1, idx2, dif) )
```

```
Difference Between "GasA" and "GasW" = "-6512" is LOW
Difference Between "GasA" and "Grav" = "96722" is HIGH
Difference Between "GasA" and "OthW" = "88222" is HIGH
Difference Between "GasW" and "Grav" = "103234" is HIGH
Difference Between "GasW" and "OthW" = "94734" is HIGH
Difference Between "Grav" and "OthW" = "-8500" is LOW
```

It is evident that difference between mean sale price of different kinds of Heatings are not same, 3 of them are LOW and rest are HIGH.

Lets see if the P-Value from ANOVA F-Test is also providing the same inference or not.

In [99]:
```
ols2 = smf.ols(formula='SalePrice ~ C(Heating)', data=df2).fit()
ols2.summary()
```

Out[99]:

OLS Regression Results

Dep. Variable:	SalePrice	**R-squared:**	0.004
Model:	OLS	**Adj. R-squared:**	-0.001
Method:	Least Squares	**F-statistic:**	0.8380
Date:	Tue, 26 Jun 2018	**Prob (F-statistic):**	0.473
Time:	08:00:28	**Log-Likelihood:**	-7749.5
No. Observations:	608	**AIC:**	1.551e+04
Df Residuals:	604	**BIC:**	1.552e+04
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	2.177e+05	3406.709	63.910	0.000	2.11e+05	2.24e+05
C(Heating)[T.GasW]	6512.5773	2.97e+04	0.220	0.826	-5.17e+04	6.47e+04
C(Heating)[T.Grav]	-9.672e+04	8.34e+04	-1.160	0.246	-2.6e+05	6.7e+04
C(Heating)[T.OthW]	-8.822e+04	8.34e+04	-1.058	0.290	-2.52e+05	7.55e+04

Omnibus:	217.122	**Durbin-Watson:**	1.937
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	803.286
Skew:	1.647	**Prob(JB):**	3.70e-175
Kurtosis:	7.568	**Cond. No.**	24.7

Prob (F-statistic): 0.473, which is greater than 0.05. We have to accept H_0 : . Hence there is no effect of different

"heating systems" on "mean sale price".

this is not true in all PAIR-WISE differences calculated above (3 - "Floor" and "Grav", "GasA" and "GasW", "Grav" and "Wall" have LOW effect on SalePrice). We may do a "Type I Error" or "Type II Error"

Type I Error : An in-correct decision is made to reject Null Hypothesis (rejection of a true null hypothesis).

Type II Error : An in-correct decision is made to accept Null Hypothesis failing to reject a false null hypothesis.

Family-wise P-Value : When we did pair-wise comparisions, P-Value will increase as it uses below formula

- $\alpha_{FW} = 1 - (1 - \alpha_{PC})^c$
- c = Number of pair-wise comparisions
- α = Normal Type 1 Error (0.05)

# Tests	Comparision (α)	Family- wise (α)
1	0.05	0.05
3	0.05	0.14
6	0.05	0.26
10	0.05	0.40
15	0.05	0.54

- With number of pairs increasing, P-Value to accept Null Hypothesis increases.

To avoid "Type I Error or Type II Error", we have to do PAIR-WISE ANOVA test for exploratory variables with more than 2 categories. POST HOC TEST for ANOVA is the solution.

POST HOC TEST for ANOVA.

When there are multiple levels (or categories) in explanatory category variable, we have perform POST HOC TEST for ANOVA. This test tells us about "which groups are different from others". In other words we have to perform pair-wise ANOVA test, while protecting against inflation of Type 1 Error. Below are some of the POST HOC TESTs.

- Sidak
- Holm T
- fisher's Latest Significant Difference
- Tukey's Honestly Significant Difference
- etc...

Lets do Tukey's Honestly Singnificant Difference test

In [100]:
```
import statsmodels.stats.multicomp as multi

mc1 = multi.MultiComparison(df2["SalePrice"],df2["Heating"])
res1 = mc1.tukeyhsd()
print(res1.summary())
```

```
    Multiple Comparison of Means - Tukey HSD,FWER=0.05
==========================================================
group1 group2   meandiff     lower        upper    reject
----------------------------------------------------------
 GasA   GasW   6512.5773  -69873.3466   82898.5013 False
 GasA   Grav  -96722.2977 -311522.8673  118078.272 False
 GasA   OthW  -88222.2977 -303022.8673  126578.272 False
 GasW   Grav  -103234.875 -330875.029   124405.279 False
 GasW   OthW   -94734.875 -322375.029   132905.279 False
 Grav   OthW     8500.0   -295020.2054 312020.2054 False
----------------------------------------------------------
```

Above table show the Pair-Wise staus to reject Null Hypothesis. We see no "True" in reject column, hence none of the Heating systems have an effect on Sale Price (Accept Null Hypothesis).

Chi-Square Test of Independence (Relationship between two Categorical Variables)

- **The Null Hypothesis H_0 : The categorical varibles are independent (They have no relationship)**
- This will measure how far Observed Values are from Expected values.
    - **Expected values : The counts/probabilities when Null Hypothesis is true**
    - **Observed values : The data in our dataset**
- **Chi Square test cannot handle more than two categories in Explanatory Variable**. If we have more than two categories in Explatory variable, we have to do POST HOC TEST.

Observed values.

In [101]:
```
df3 = housing_train_set[["BsmtQual", "OverallQual"]].dropna()
df3["BsmtQual"] = pd.Categorical(df3["BsmtQual"])
df3["OverallQual"] = pd.Categorical(df3["OverallQual"])
```

In [102]:
```
cross_tab = pd.crosstab(df3.BsmtQual, df3.OverallQual, margins=True)
```

In [103]:
```
cross_tab
```

Out[103]:

OverallQual	4	5	6	7	8	9	10	All
BsmtQual								
Ex	0	0	2	10	30	30	11	83
Fa	0	3	3	1	0	0	0	7
Gd	1	18	83	116	79	4	0	301
TA	9	70	94	36	6	0	2	217
All	10	91	182	163	115	34	13	608

How to calculate expected values ?

- If the probability of Event A, Event B are independent, P(A and B) = P(A).P(B). This is the situation when Null Hypothesis is True (No relationship between A and B). Hence we can use this formula.

P(BsmtQual=EX and OverallQual=9) = p(BsmtQual=EX) * P(OverallQual=1)

P(BsmtQual=EX) = 83/608
P(OverallQual=9) = 34/608

P(BsmtQual=EX and OverallQual=9) = 83 * 34/608*608

Expected value for all 608 records is = 608*83*34/608*608
= 83*34/608 = (row total) * (column total)/Total records

	1	2	3	4	5	6	7	8	9	10
EX	----	----	----	----	----	----	----	----	4.64	---
FA	----	----	----	----	----	----	----	----	----	---
GD	----	----	----	----	----	----	----	----	----	---
NA	----	----	----	----	----	----	----	----	----	---
TA	----	----	----	----	----	----	----	----	----	---

Null Hypothesis : BsmtQual has no effect on OverallQual of a house.

In [104]:
```
import scipy.stats as stats

c2table = stats.chi2_contingency(cross_tab)
print(c2table)
```

```
(424.26922975680679, 2.2392416721007541e-72, 28, array([[  1.36513158e+00,   1.24226974e+01,   2.48453947e+01,
          2.22516447e+01,   1.56990132e+01,   4.64144737e+00,
          1.77467105e+00,   8.30000000e+01],
       [  1.15131579e-01,   1.04769737e+00,   2.09539474e+00,
          1.87664474e+00,   1.32401316e+00,   3.91447368e-01,
          1.49671053e-01,   7.00000000e+00],
       [  4.95065789e+00,   4.50509868e+01,   9.01019737e+01,
          8.06957237e+01,   5.69325658e+01,   1.68322368e+01,
          6.43585526e+00,   3.01000000e+02],
       [  3.56907895e+00,   3.24786184e+01,   6.49572368e+01,
          5.81759868e+01,   4.10444079e+01,   1.21348684e+01,
          4.63980263e+00,   2.17000000e+02],
       [  1.00000000e+01,   9.10000000e+01,   1.82000000e+02,
          1.63000000e+02,   1.15000000e+02,   3.40000000e+01,
          1.30000000e+01,   6.08000000e+02]]))
```

P-Value (2.2392416721007541e-72) from above Chi Squre test is very small, hence we can reject Null Hypothesis.

as there are more than 2 categories in explanatory variable, there is a possibility of "Type I Error". To protect againest "Type I Error" we will be using Bonferroni Adjustment.

- Bonferroni Adjustment adjustment will adjust P-Value using $\frac{P}{C}$ formula, where C - is the number of comparisions we would like to look into.
- In the above example we have to calculate PAIR WISE Chi-Square tests. Here we have to compare effect of different BsmtQual categories. There are 5 different categories. We have to do 5c2, That is 10 combinations. **Hence Bonferroni Adjusted P-Value is $\frac{0.05}{10}$ = 0.005.**

Below is how we need to compute PAIR-WISE Chi-Squre tests. The P-Value to reject Null Hypothesis should be < 0.005 (0.05/10 - 10 pairs).

In [105]:
```
#temp1 = df3[df3.BsmtQual.isin(["TA", "Gd "])]
#pd.crosstab(temp1.BsmtQual, temp1.OverallQual, margins=True)
```

In [106]:
```
map1 = {"TA":"TA" ,"Gd":"Gd"}
temp2 = df3.copy()
temp2["BsmtQual"] = df3["BsmtQual"].map(map1) # temp2 will get only 2 rows with "TA" and "Gd"
pd.crosstab(temp2.BsmtQual, temp2.OverallQual, margins=True)
```

Out[106]:

OverallQual	4	5	6	7	8	9	10	All
BsmtQual								
Gd	1	18	83	116	79	4	0	301
TA	9	70	94	36	6	0	2	217
All	10	88	177	152	85	4	2	518

In [107]:
```
unq_cats = df3["BsmtQual"].unique()
seen_cat = []
accept_nh_list = []
reject_nh_list = []
for cat1 in unq_cats:
    seen_cat.append(cat1)
    for cat2 in unq_cats:
        if(cat2 not in seen_cat):
            print("Calculating Chi Squre Test for \"{0}\" - \"{1}\" PAIR".format(cat1, cat2))
            df3_copy = df3.copy()
            df3_copy["BsmtQual"] = df3["BsmtQual"].map({cat1:cat1, cat2:cat2})
            ct = pd.crosstab(df3_copy.BsmtQual, df3_copy.OverallQual, margins=True)
            c2table = stats.chi2_contingency(cross_tab)
            #print(c2table)
            pair = cat1, "-", cat2
            if(c2table[1] < 0.005):
                print("Reject Null Hypothesis for")
                reject_nh_list.append(pair)
            else:
                print("Accept Null Hypothesis")
                accept_nh_list.append(pair)
            print(100*"_")
```

```
Calculating Chi Squre Test for "Gd" - "TA" PAIR
Reject Null Hypothesis for
____________________________________________________________________________________________________
Calculating Chi Squre Test for "Gd" - "Ex" PAIR
Reject Null Hypothesis for
____________________________________________________________________________________________________
Calculating Chi Squre Test for "Gd" - "Fa" PAIR
Reject Null Hypothesis for
____________________________________________________________________________________________________
Calculating Chi Squre Test for "TA" - "Ex" PAIR
Reject Null Hypothesis for
____________________________________________________________________________________________________
Calculating Chi Squre Test for "TA" - "Fa" PAIR
Reject Null Hypothesis for
____________________________________________________________________________________________________
Calculating Chi Squre Test for "Ex" - "Fa" PAIR
Reject Null Hypothesis for
____________________________________________________________________________________________________
```

All pairs are saying reject Null Hypothesis, hence there is an effect of "BsmtQual" on "OverAllQual" attribute.

Data Filtering & Aggregate the Data

To aggregate, we typically use the “group by” function, which involves the following steps

```
- Splitting the data into groups based on some criteria
- Applying a function to each group independently
- Combining the results into a data structure
```

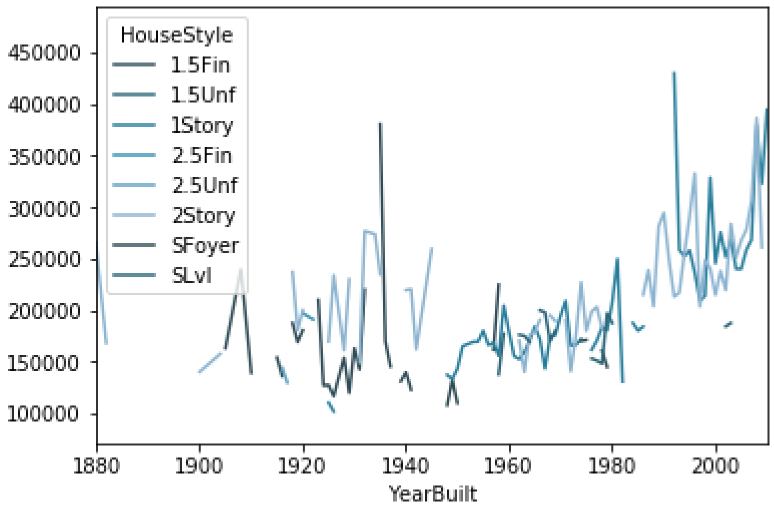
In [108]:
```
# Aggregate and analyze the trend of total sale price of different HouseStyles for YearBuilt
df_house_style = pd.pivot_table(housing_train_set, values='SalePrice', index=['YearBuilt'], columns=['HouseStyle'] )
df_house_style.head()
```

Out[108]:

HouseStyle	1.5Fin	1.5Unf	1Story	2.5Fin	2.5Unf	2Story	SFoyer	SLvl
YearBuilt								
1880	NaN	NaN	NaN	295000.0	NaN	265979.0	NaN	NaN
1882	NaN	NaN	NaN	NaN	NaN	168000.0	NaN	NaN
1892	NaN	NaN	NaN	475000.0	NaN	NaN	NaN	NaN
1893	NaN	NaN	NaN	NaN	325000.0	NaN	NaN	NaN
1900	160000.0	NaN	NaN	NaN	NaN	140000.0	NaN	NaN

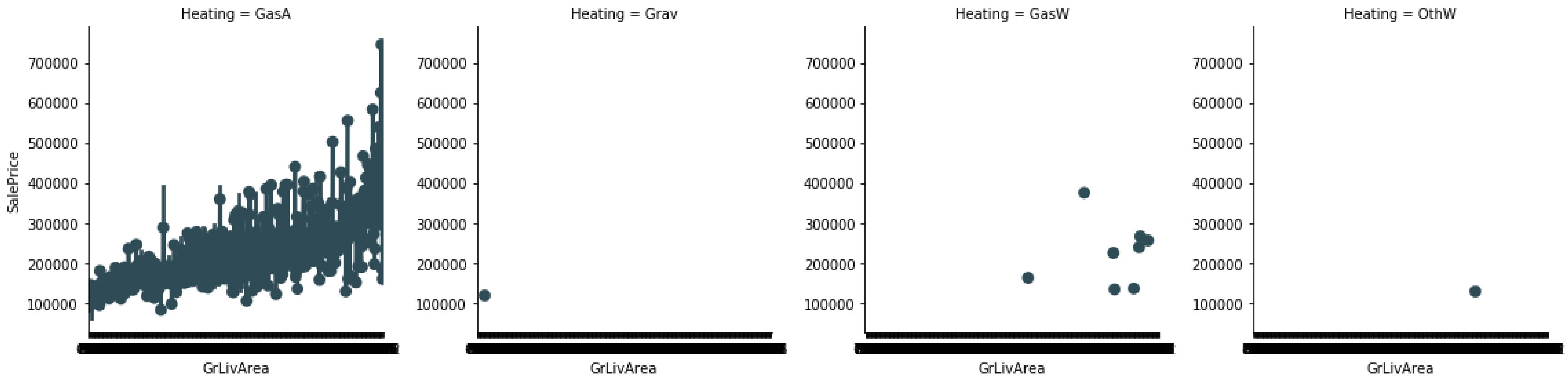
In [109]:
```
df_house_style.plot()
```

Out[109]: <matplotlib.axes._subplots.AxesSubplot at 0x218264dfa90>



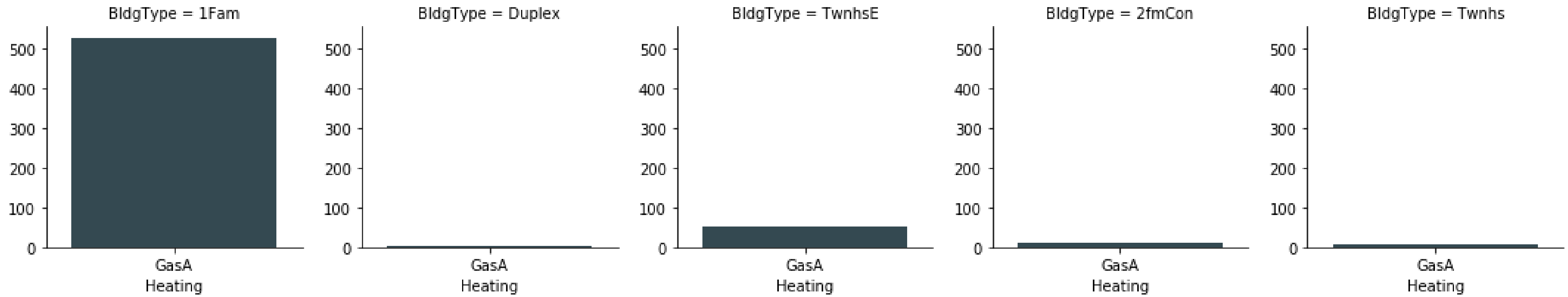
In [110]:
```
# Get scatter plot group by col
# taking around 3 mins to generate
sns.factorplot(data=housing_train_set, x="GrLivArea", y="SalePrice", col="Heating")
```

Out[110]: <seaborn.axisgrid.FacetGrid at 0x21825530da0>



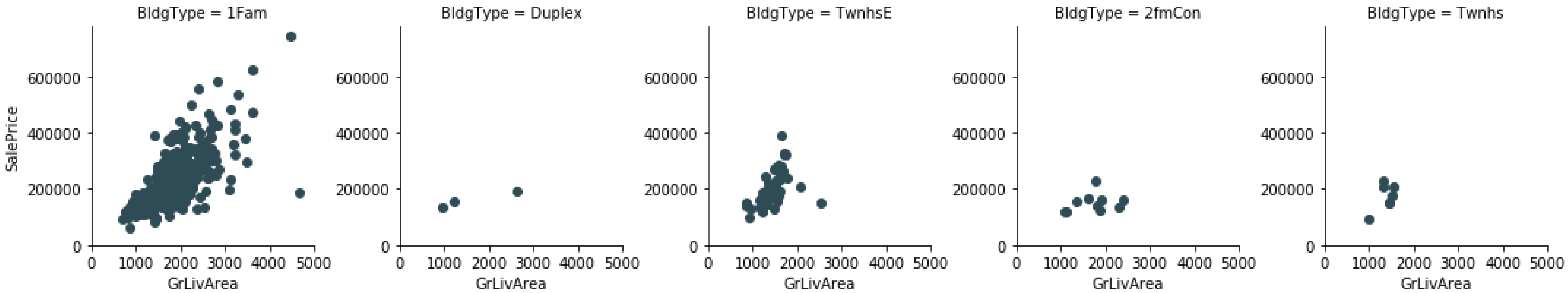
In [111]:
```
# Get bar chart of one categorical variable by other categorical variable.
g = sns.FacetGrid(data=housing_train_set, col="BldgType")
g.map(sns.countplot, "Heating")
```

Out[111]: <seaborn.axisgrid.FacetGrid at 0x218291e27f0>

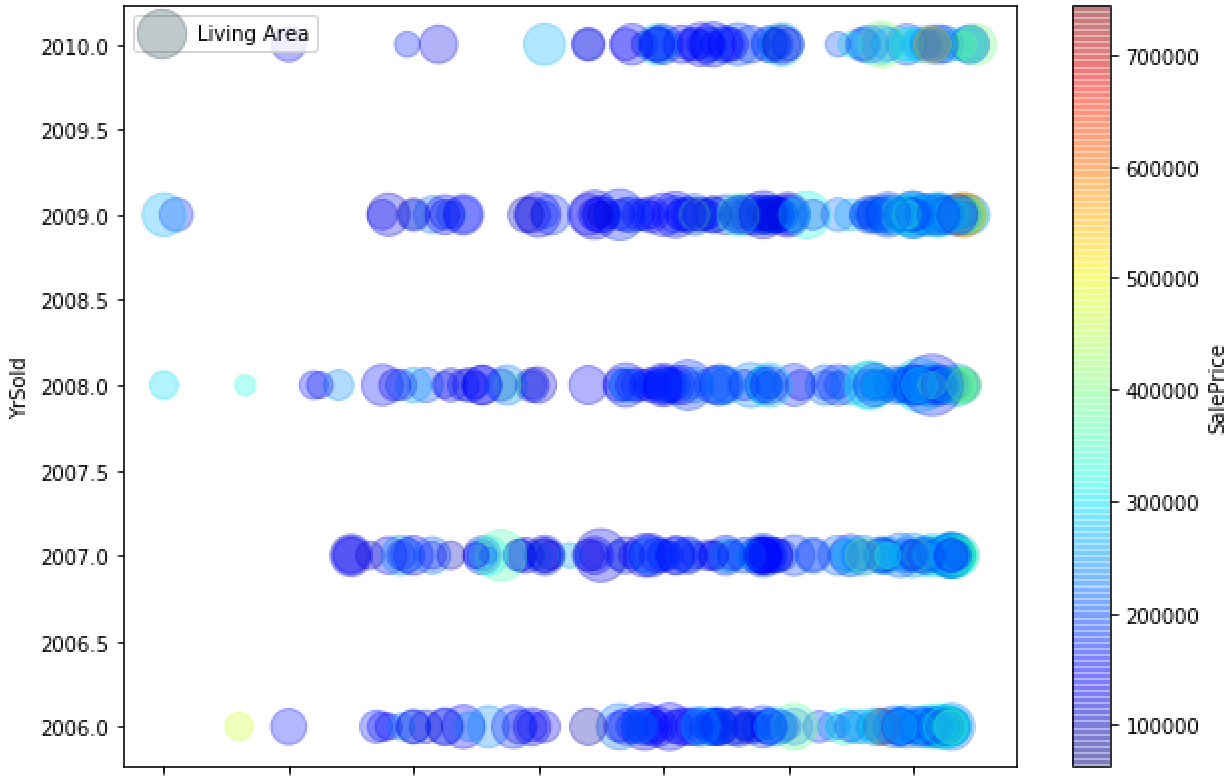


In [112]:
```
# Get bar chart of one categorical variable by other categorical variable.
g = sns.FacetGrid(data=housing_train_set, col="BldgType")
g.map(plt.scatter, "GrLivArea", "SalePrice")
g.set(xlim=(0, 5000))
g.set(ylim=(0, None))
```

Out[112]: <seaborn.axisgrid.FacetGrid at 0x2182b188588>



In [113]:
```
housing_train_set.plot(kind="scatter", x="YearBuilt", y="YrSold",
    s=housing_data['GrLivArea']/5, label="Living Area",
    c="SalePrice", cmap=plt.get_cmap("jet"),
    colorbar=True, alpha=0.3, figsize=(10,7)
)
plt.legend()
plt.show()
```



In [114]:
```
housing_train_set.YrSold.value_counts()
```

Out[114]:
```
2009    146
2007    146
2006    133
2008    118
2010     65
Name: YrSold, dtype: int64
```

Quality of Data

```
* For this dataset we can check
1) is YearBuilt <= YrSold ?
2) is YearBuilt <= GarageYrBlt  ?
```

In [115]:
```
print('Is YearBuilt <= YrSold : ', len(housing_train_set) == \
      len(housing_train_set[housing_train_set['YearBuilt'] <= housing_train_set['YrSold']]))
```

```
Is YearBuilt <= YrSold :  True
```

In [116]:
```
print('Is YearBuilt < GarageYrBlt : ', len(housing_train_set) == \
      len(housing_train_set[housing_train_set['YearBuilt'] <= housing_train_set['GarageYrBlt']]))
```

```
Is YearBuilt < GarageYrBlt :  False
```

In [117]:
```
# This records for which the Garage built before Building are not correct.
# This might have came because of the imputation method we chose.
housing_train_set[housing_train_set['GarageYrBlt'] < \
                  housing_train_set['YearBuilt']][['GarageYrBlt', 'YearBuilt']]
```

Out[117]:

	GarageYrBlt	YearBuilt
288	1954.0	1959
769	1900.0	1910
1117	2003.0	2005

In [118]:
```
housing_train_set[housing_train_set['GarageYrBlt'] < housing_train_set['YearBuilt']].index
```

Out[118]:
```
Int64Index([288, 769, 1117], dtype='int64')
```

In [119]:
```
# Droping inconsistent data from the data frame.
housing_train_set.drop(housing_train_set[housing_train_set['GarageYrBlt'] < \
                                         housing_train_set['YearBuilt']].index, inplace=True)
```

How to check Normal Distribution of data

Shapior-Wilk Test :

* Null Hypothesis : The Shapiro-Wilk test tests the null hypothesis that the data was drawn from a normal distribution.
* The test gives us 'Statistic' and the 'P-Value'. If P-Value is less than chosen alpha level (0.05) then we can reject null hypothesis.

Note: However, since the test is biased by sample size, the test may be statistically significant from a normal distribution in any large samples. Thus a Q–Q plot is required for verification in addition to the test.

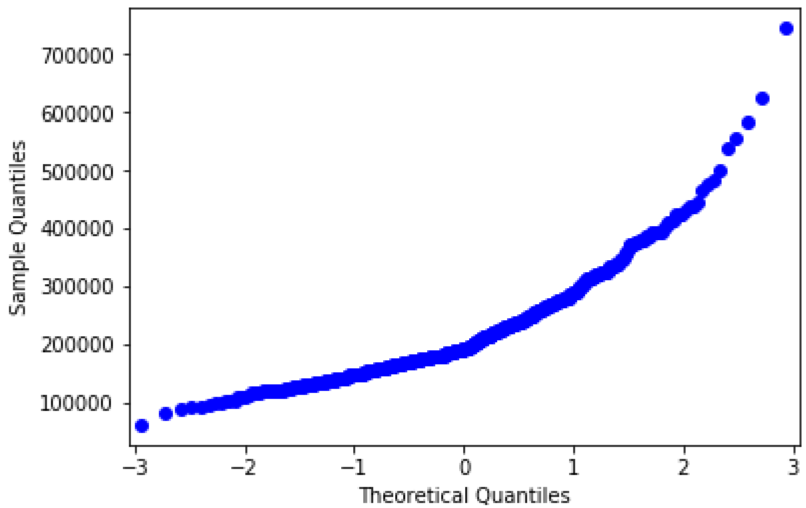
In [120]:
```
stats.shapiro(housing_train_set.SalePrice)
```

Out[120]:
```
(0.8833791017532349, 6.210418724672713e-21)
```

Shaport-Wilk Test says that SalePrice is **not from a normally distrbuted population** as p-values is near zero. But when we look at the Q-Q polt it shows that the data is not normally distributed (A little skewness).

In [121]:
```
plt.figure(figsize=(30, 30))
fig1 = qqplot(housing_train_set.SalePrice)
plt.show()
```

```
<matplotlib.figure.Figure at 0x2182b07fba8>
```



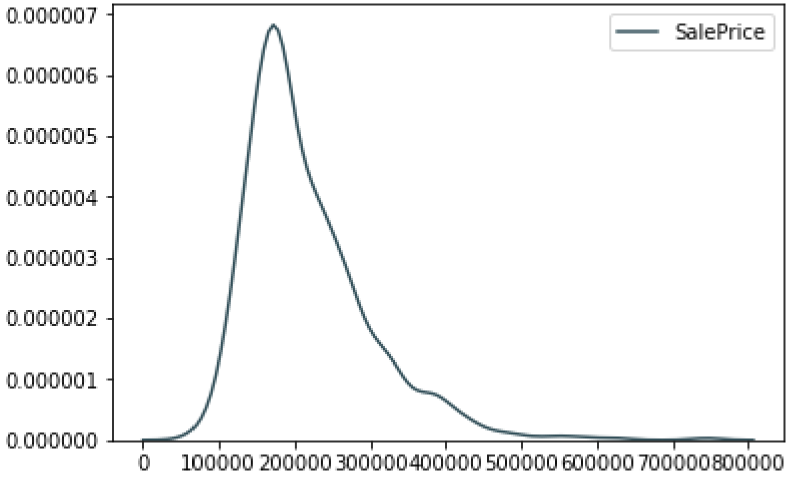
- Linear models rely upon a lot of assumptions. If assumptions are violated, the diagnostics obtained from the model cannot be relied.
- R-square vs adjusted R-squre : Biggest challenge is that adding any feature will increase the R-square. One way to counter this is to use adjusted R-square.
- Take a step back and think - why do we need to report those numbers? We want some estimate of generalization. Cross-validation score provides a general framework for reporting generalization. And this will hold good across all models. And thus, multiple models can be compared. This is the machine learning approach and is widely used in practice.

Try transform Skewed or non-normal distribution to normal by applying transformations.

    - These transformations may or may not transorm data, but there is nothing wrong in trying.

In [122]:
```
sns.kdeplot(housing_train_set.SalePrice)
```

Out[122]: <matplotlib.axes._subplots.AxesSubplot at 0x2182b6714e0>



In [123]:
```
stats.skew(housing_train_set.SalePrice)
```

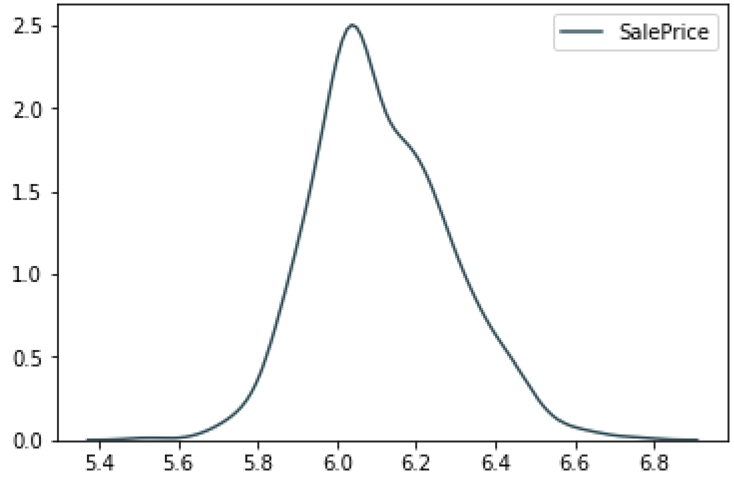
Out[123]: 1.6428793772578885

In [124]:
```
stats.skew(np.log(np.sqrt(housing_train_set.SalePrice)))
```

Out[124]: 0.3858487839761124

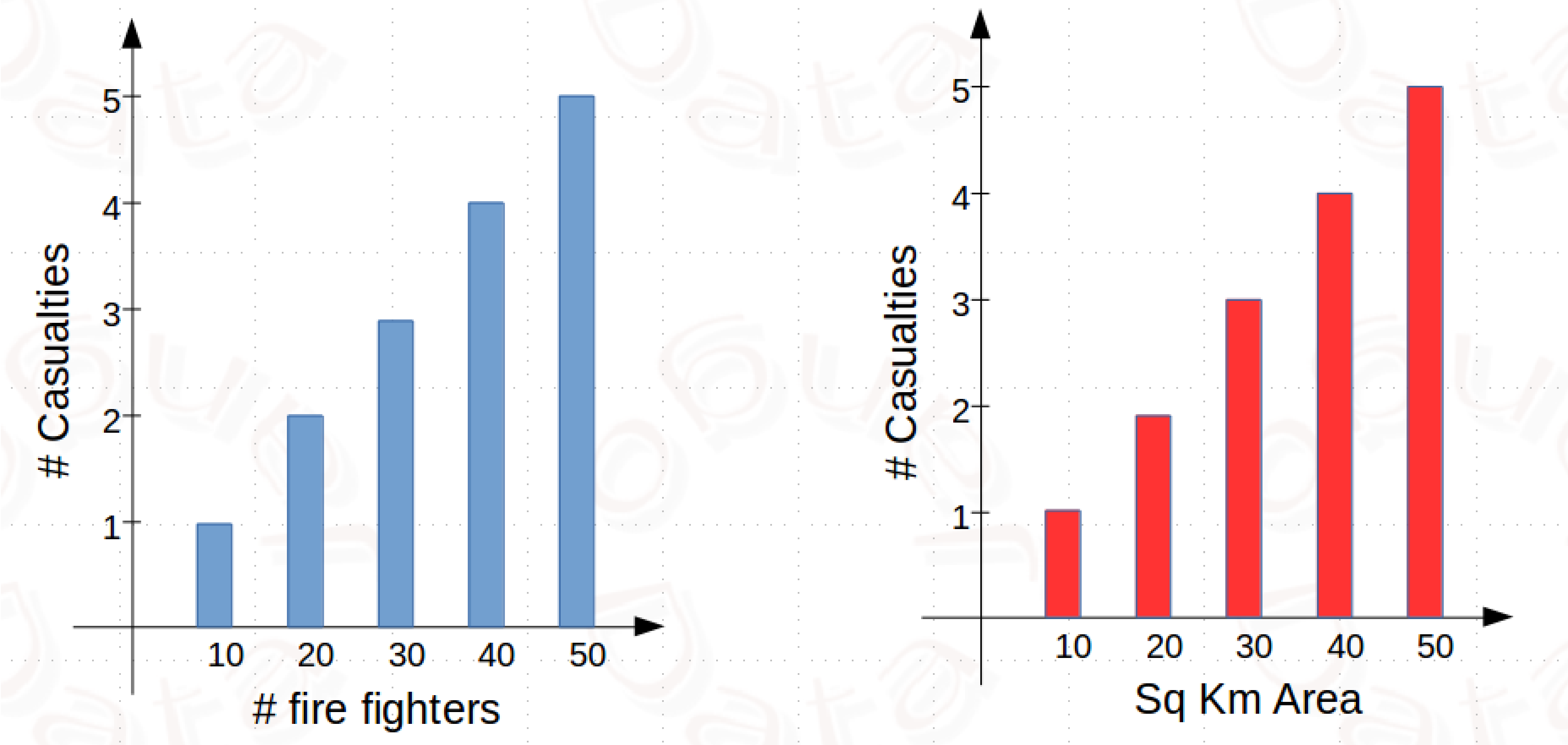
In [125]:
```
sns.kdeplot(np.log(np.sqrt(housing_train_set.SalePrice)))
```

Out[125]: <matplotlib.axes._subplots.AxesSubplot at 0x2182b61e7b8>



Association is not Causation (What are confounding variables ?)

- Most of the times we may missunderstand (linear or non-linear) relationship between two variable causing each other or may be able predict each other. This may not be correct all the time. Example is as below.
  - Suppose "Number of casualties", "Number of fire fighters" and "Sq KM Area" are three variables. "Number or casualties" is the response variable.
  - If we look first graph it looks at first graph it looks like sending more "number of fire fighters" is increasing number of causalties. But **the confounding feature "Sq KM Area" in association with "# fire fighters" might be the real cause behind number of causualties**.



- **Confounding Variables** are also know as "Control Variable", "Covarite", "Third Variable", "Lurking Variable".

Conclusion:

```
- Till now we have found few ways of exploring the data and gain insights.
- We have performed some data cleanup activity, before feeding the data to Machine Learning algorthm.
- We found some interesting correlations between featrues, especially with target variable.
- We found some tail heavy distributions and applied transformations to make the data normally distributed.
```