1.Problem Analysis

In the estate market, each house has a selling price based on its different features and conditions. Based on the given train dataset, test dataset and description of each features, this project aims to predict the provided house sale price on the market.

2.Get Data and Data cleaning

Import the pandas library and load train.csv file into Python. A quick review of the input data is as follows: there are 81 different features in column and 1460 observations samples in row:

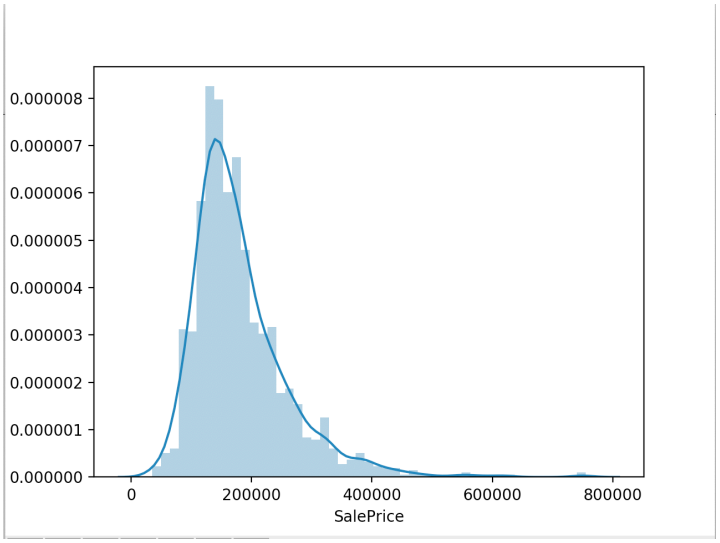
```
>>> df_train
      Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape LandContour Utilities  ... PoolArea PoolQC  Fence MiscFeature MiscVal MoSold YrSold  SaleType  SaleCondition  SalePric
e
0      1          60       RL         65.0     8450   Pave   NaN      Reg         Lvl    AllPub  ...        0    NaN    NaN         NaN       0      2   2008        WD         Normal     20850
0
1      2          20       RL         80.0     9600   Pave   NaN      Reg         Lvl    AllPub  ...        0    NaN    NaN         NaN       0      5   2007        WD         Normal     18150
0
2      3          60       RL         68.0    11250   Pave   NaN      IR1         Lvl    AllPub  ...        0    NaN    NaN         NaN       0      9   2008        WD         Normal     22350
0
3      4          70       RL         60.0     9550   Pave   NaN      IR1         Lvl    AllPub  ...        0    NaN    NaN         NaN       0      2   2006        WD        Abnorml     14000
0
4      5          60       RL         84.0    14260   Pave   NaN      IR1         Lvl    AllPub  ...        0    NaN    NaN         NaN       0     12   2008        WD         Normal     25000
0
...  ...         ...      ...          ...      ...    ...   ...      ...         ...       ...  ...      ...    ...    ...         ...     ...    ...    ...       ...            ...       ..
.
1455  1456          60       RL         62.0     7917   Pave   NaN      Reg         Lvl    AllPub  ...        0    NaN    NaN         NaN       0      8   2007        WD         Normal     17500
0
1456  1457          20       RL         85.0    13175   Pave   NaN      Reg         Lvl    AllPub  ...        0    NaN  MnPrv         NaN       0      2   2010        WD         Normal     21000
0
1457  1458          70       RL         66.0     9042   Pave   NaN      Reg         Lvl    AllPub  ...        0    NaN  GdPrv        Shed    2500      5   2010        WD         Normal     26650
0
1458  1459          20       RL         68.0     9717   Pave   NaN      Reg         Lvl    AllPub  ...        0    NaN    NaN         NaN       0      4   2010        WD         Normal     14212
5
1459  1460          20       RL         75.0     9937   Pave   NaN      Reg         Lvl    AllPub  ...        0    NaN    NaN         NaN       0      6   2008        WD         Normal     14750
0

[1460 rows x 81 columns]
```

All column headers in the train dataset can be observed as below:

```
>>> df_train=pd.read_csv('train.csv')
>>> print(df_train.columns)
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')
>>>
```

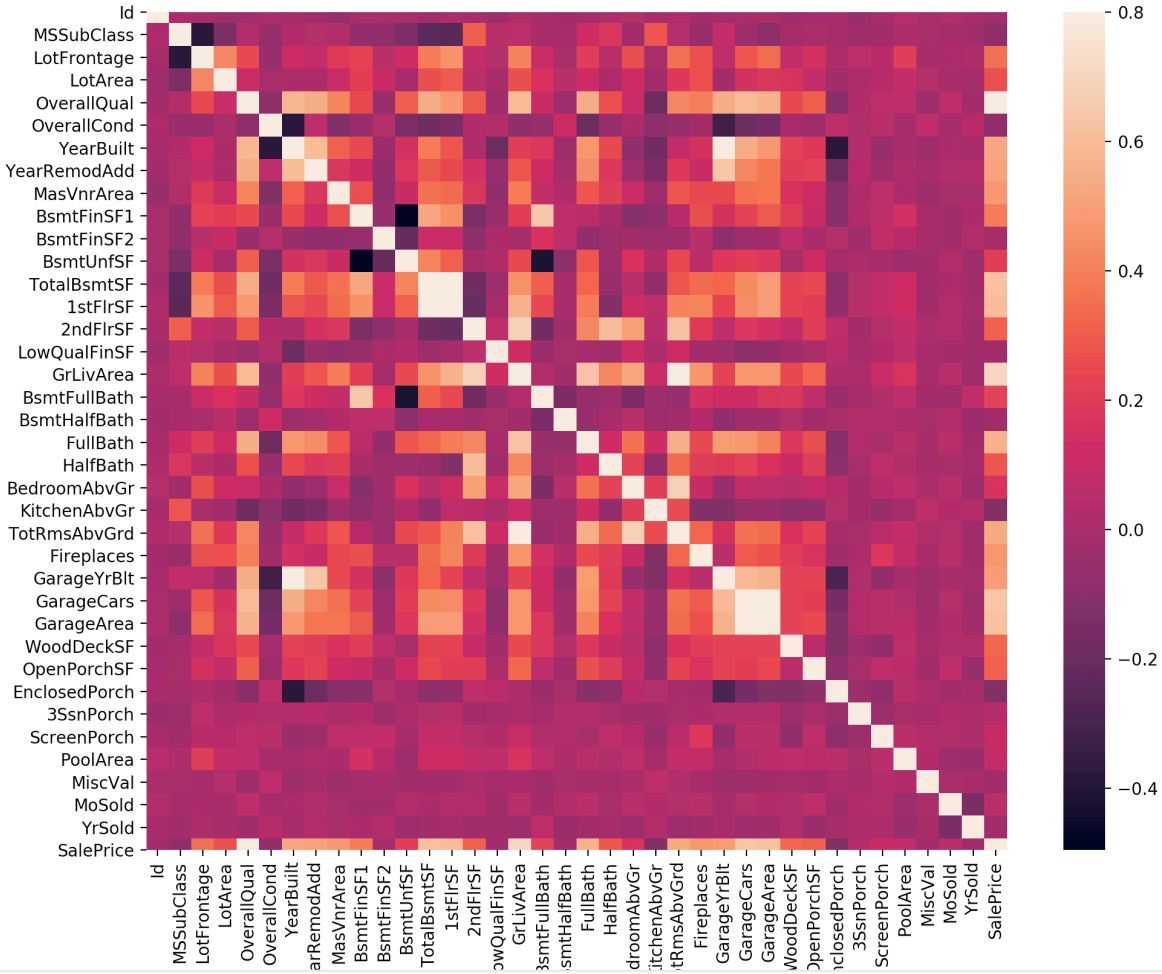
A descriptive statistic summary of SalePrice and distribution plot in histogram are as below:

```
>>> df_train['SalePrice'].describe()
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
>>>
```



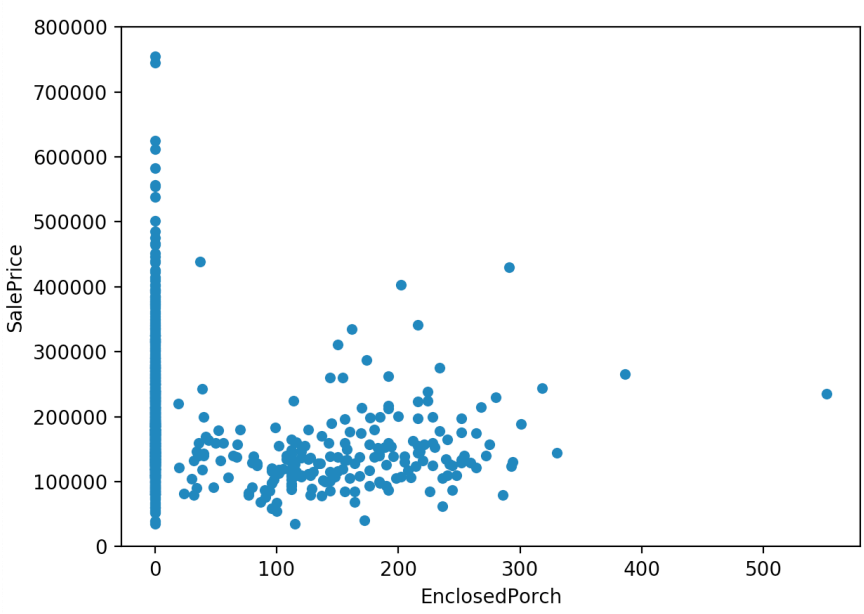
From the statistics and SalePrice distribution perspective, the mean and max value of SalePrice is about $180921 and $755000, and most of the SalePrice values are between $100000 to $400000.

Through correlation matrix, the correlation between each two variables can be observed:



Based on the above Heatmap figure, a low correlation can be observed between feature EnclosedPorch and SalePrice. A further scatter plot is applied as below which further shows that there is no clear linear correlation between this two feature. EnclosedPorch will be kept until further confirmation.

This strategy can be applied to check whether a feature is highly related with SalePrice or not.



Duplicated observation in sample dataset can be showed by using duplicated() function. The output will be Empty DataFrame when there is no duplicated rows in the dataset.

```
>>> df_train[df_train.duplicated()==True]
Empty DataFrame
Columns: [Id, MSSubClass, MSZoning, LotFrontage, LotArea, Street, Alley, LotShape, LandContour, Utilities, LotConfig, LandSlope, Neighborhood, Condition1, Condition2, BldgType, HouseStyle, Overa
llQual, OverallCond, YearBuilt, YearRemodAdd, RoofStyle, RoofMatl, Exterior1st, Exterior2nd, MasVnrType, MasVnrArea, ExterQual, ExterCond, Foundation, BsmtQual, BsmtCond, BsmtExposure, BsmtFinTy
pe1, 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, Paved
Drive, WoodDeckSF, OpenPorchSF, EnclosedPorch, 3SsnPorch, ScreenPorch, PoolArea, PoolQC, Fence, MiscFeature, MiscVal, MoSold, YrSold, SaleType, SaleCondition, SalePrice]
Index: []
>>>
```

There are mainly 3 data types in the train dataset, which are int64, object and float64.

The detailed info can be observed as below snapshots:

```
>>> res = df_train.dtypes
>>> print(res[res == np.dtype('int64')])
Id               int64
MSSubClass       int64
LotArea          int64
OverallQual      int64
OverallCond      int64
YearBuilt        int64
YearRemodAdd     int64
BsmtFinSF1       int64
BsmtFinSF2       int64
BsmtUnfSF        int64
TotalBsmtSF      int64
1stFlrSF         int64
2ndFlrSF         int64
LowQualFinSF     int64
GrLivArea        int64
BsmtFullBath     int64
BsmtHalfBath     int64
FullBath         int64
HalfBath         int64
BedroomAbvGr     int64
KitchenAbvGr     int64
TotRmsAbvGrd     int64
Fireplaces       int64
GarageCars       int64
GarageArea       int64
WoodDeckSF       int64
OpenPorchSF      int64
EnclosedPorch    int64
3SsnPorch        int64
ScreenPorch      int64
PoolArea         int64
MiscVal          int64
MoSold           int64
YrSold           int64
SalePrice        int64
dtype: object
>>>
```

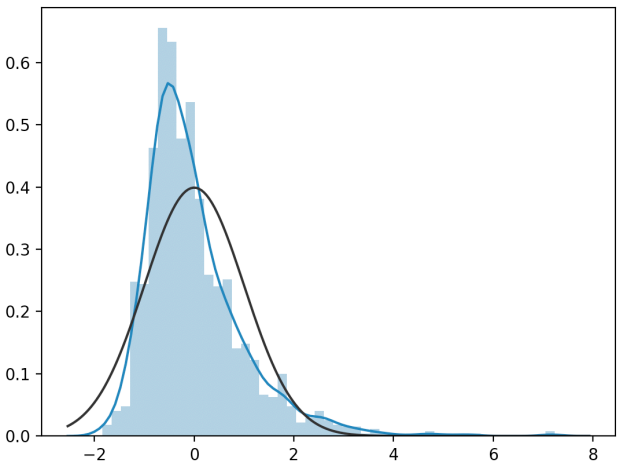
```
>>> print(res[res == np.dtype('float64')])
LotFrontage    float64
MasVnrArea     float64
GarageYrBlt    float64
dtype: object
>>>
```

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

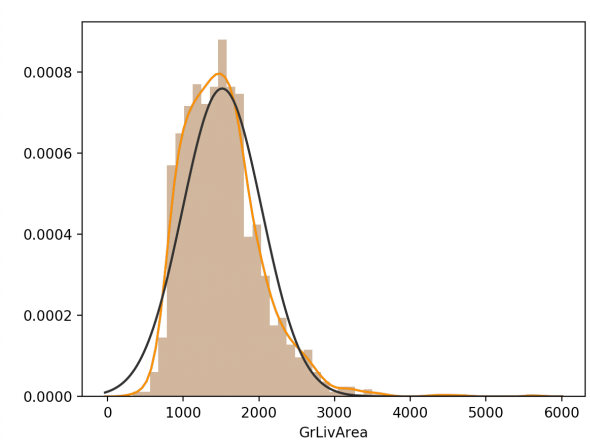
The unique values of each variable can also be observed through unique() function as below example:

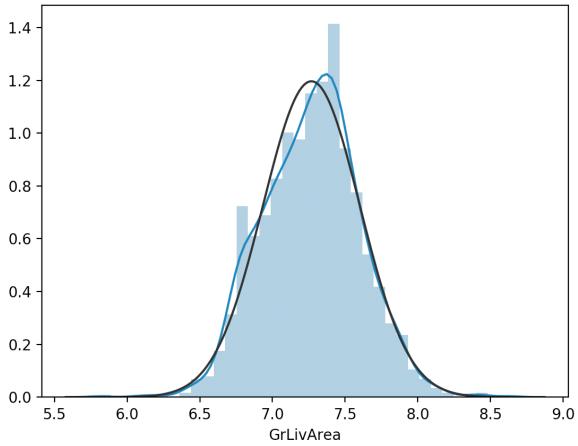
```
>>> print(df_train["LotConfig"].unique())
['Inside' 'FR2' 'Corner' 'CulDSac' 'FR3']
>>>
```

For numerical variable, feature scaling can be applied to fit and transform data into normal distribution .



Data transformation: the distribution of GrLivArea column can be observed with fitting normalisation format as the left figure. GrLivArea column can also be calculated through log() function and a new distribution picture can be plotted as the right figure.





About missing data processing:

Sum all null values of each column and calculate the percentage of missing values for each column, then the first 20 highest percentages of

missing values for columns can be found out and further processing can be applied based on different situations, for example, for any column whose missing percentage is grater than 15% will be dropped.

```
>>> total = df_train.isnull().sum().sort_values(ascending=False)
>>> total
PoolQC          1453
MiscFeature     1406
Alley           1369
Fence           1179
FireplaceQu      690
                ...
CentralAir         0
SaleCondition      0
Heating            0
TotalBsmtSF        0
Id                 0
Length: 81, dtype: int64
>>>
```

```
>>> missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
>>> missing_data.head(20)
              Total   Percent
PoolQC         1453  0.995205
MiscFeature    1406  0.963014
Alley          1369  0.937671
Fence          1179  0.807534
FireplaceQu     690  0.472603
LotFrontage     259  0.177397
GarageCond       81  0.055479
GarageType       81  0.055479
GarageYrBlt      81  0.055479
GarageFinish     81  0.055479
GarageQual       81  0.055479
BsmtExposure     38  0.026027
BsmtFinType2     38  0.026027
BsmtFinType1     37  0.025342
BsmtCond         37  0.025342
BsmtQual         37  0.025342
MasVnrArea        8  0.005479
MasVnrType        8  0.005479
Electrical        1  0.000685
Utilities         0  0.000000
>>>
```

```
>>> percent = (df_train.isnull().sum()/df_train.isnull().count()).sort_values(ascending=False)
>>> percent
PoolQC          0.995205
MiscFeature     0.963014
Alley           0.937671
Fence           0.807534
FireplaceQu     0.472603
                  ...
CentralAir      0.000000
SaleCondition   0.000000
Heating         0.000000
TotalBsmtSF     0.000000
Id              0.000000
Length: 81, dtype: float64
>>>
```

Missing data can also be imputed: The method is that set the values of a highly correlated column to replace the missing values.

There are 259 values are missing before the imputation, and after processing the missing value number is 0.

```
>>>
>>> print(df_train["LotFrontage"].isnull().sum())
259
>>> cond = df_train['LotFrontage'].isnull()
>>> df_train["LotFrontage"][cond]=df_train["SqrtLotArea"][cond]
__main__:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
>>> print(df_train["LotFrontage"].isnull().sum())
0
>>>
```

Another way is to mark the missing value of the column as “Missing”:

```
>>> mis=df_train['GarageType'].isnull()
>>>
>>> df_train["GarageType"][mis]="Missing"
>>> df_train["GarageType"].unique()
array(['Attchd', 'Detchd', 'BuiltIn', 'CarPort', 'Missing', 'Basment',
       '2Types'], dtype=object)
>>>
```

Drop the columns whose data value missing percentage is greater than 15%, drop the row whose “Electrical” is null. After dropping, the data frame is changed into 1459 rows * 76 columns from original 1460 rows * 81 columns.

```
>>> df_train = df_train.drop((missing_data[missing_data['Percent'] > 0.15]).index,1)
>>> df_train = df_train.drop(df_train.loc[df_train['Electrical'].isnull()].index)
>>> df_train
        Id  MSSubClass MSZoning  LotArea Street LotShape LandContour Utilities LotConfig  ... ScreenPorch PoolArea MiscVal MoSold YrSold SaleType  SaleCondition  SalePrice  SqrtLotArea
0        1          60       RL     8450   Pave      Reg         Lvl    AllPub    Inside  ...           0        0       0      2   2008       WD         Normal     208500    91.923882
1        2          20       RL     9600   Pave      Reg         Lvl    AllPub       FR2  ...           0        0       0      5   2007       WD         Normal     181500    97.979590
2        3          60       RL    11250   Pave      IR1         Lvl    AllPub    Inside  ...           0        0       0      9   2008       WD         Normal     223500   106.066017
3        4          70       RL     9550   Pave      IR1         Lvl    AllPub    Corner  ...           0        0       0      2   2006       WD        Abnorml     140000    97.724101
4        5          60       RL    14260   Pave      IR1         Lvl    AllPub       FR2  ...           0        0       0     12   2008       WD         Normal     250000   119.415242
...    ...         ...      ...      ...    ...      ...         ...       ...       ...  ...         ...      ...     ...    ...    ...      ...            ...        ...          ...
1455  1456          60       RL     7917   Pave      Reg         Lvl    AllPub    Inside  ...           0        0       0      8   2007       WD         Normal     175000    88.977525
1456  1457          20       RL    13175   Pave      Reg         Lvl    AllPub    Inside  ...           0        0       0      2   2010       WD         Normal     210000   114.782403
1457  1458          70       RL     9042   Pave      Reg         Lvl    AllPub    Inside  ...           0        0    2500      5   2010       WD         Normal     266500    95.089432
1458  1459          20       RL     9717   Pave      Reg         Lvl    AllPub    Inside  ...           0        0       0      4   2010       WD         Normal     142125    98.574845
1459  1460          20       RL     9937   Pave      Reg         Lvl    AllPub    Inside  ...           0        0       0      6   2008       WD         Normal     147500    99.684502

[1459 rows x 76 columns]
```

Check correlation between two variables through corr() function: LotFrontage and LotArea

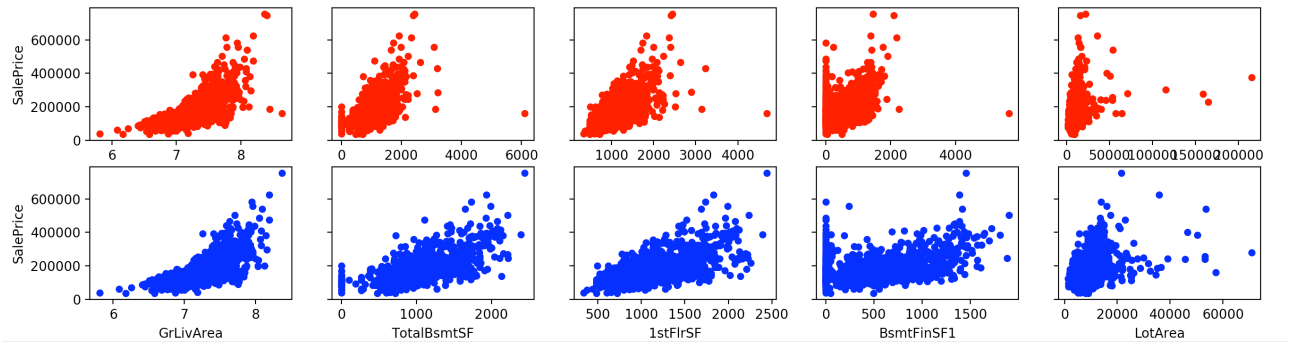
```
>>> df_train['LotFrontage'].corr(df_train['LotArea'])
0.42609501877180833
>>>
```

Another sqrt() function can be further applied to reduce the numeric effect.

```
>>> df_train['SqrtLotArea']=np.sqrt(df_train['LotArea'])
>>> df_train['LotFrontage'].corr(df_train['SqrtLotArea'])
0.6020022167939361
>>>
```

For outliers processing:

There are some outliers for several columns, these outliers can be removed directly to reduce the effects of outliers in final prediction processing. The train data is changed into 1444 * 76 from 1459 * 76.

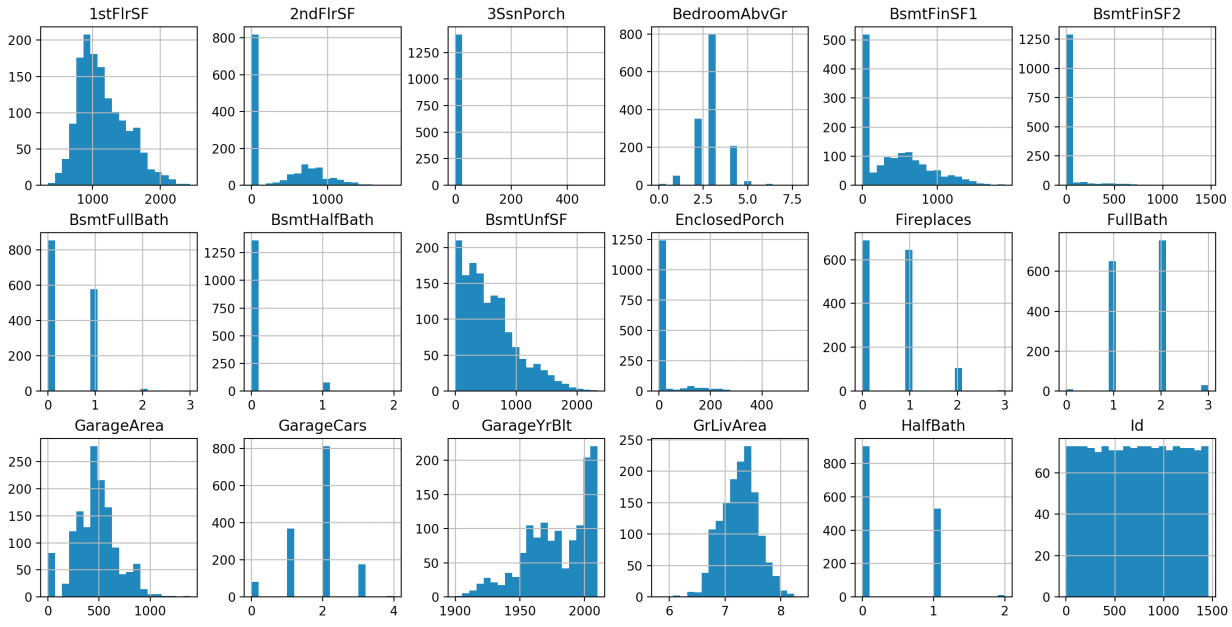


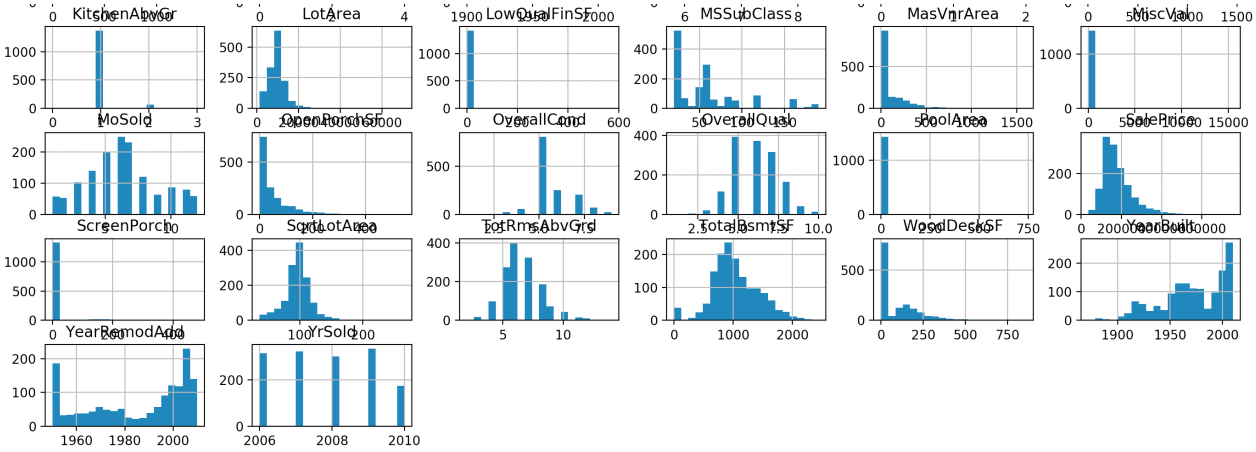
3. Feature Selection and Engineering

After data cleaning process, now there are 1444 entries(observation samples) and 76 columns in train dataset before feature selection and engineering processing.

```
>>> print(df_train.info())
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1444 entries, 0 to 1459
Data columns (total 76 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1444 non-null   int64
 1   MSSubClass     1444 non-null   int64
 2   MSZoning       1444 non-null   object
 3   LotArea        1444 non-null   int64
 4   Street         1444 non-null   object
 5   LotShape       1444 non-null   object
 6   LandContour    1444 non-null   object
 7   Utilities      1444 non-null   object
 8   LotConfig      1444 non-null   object
 9   LandSlope      1444 non-null   object
 10  Neighborhood   1444 non-null   object
 11  Condition1     1444 non-null   object
 12  Condition2     1444 non-null   object
```

The histogram for each feature can be plotted as below showing:





Based on above the histogram figures, feature “3SsnPorch” is suspicious because almost all observe values of sample are zero. Therefore, statistical summary information can be further checked as below snapshots, and at least 75% of the sample values are zero, which means this feature may be not quite useful during the modelling and prediction.

```
>>> df_train['3SsnPorch'].describe()
count    1444.000000
mean        3.447368
std        29.477207
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max       508.000000
Name: 3SsnPorch, dtype: float64
>>>
```

And another feature “BedroomAbvGr” is not normal distributed. By checking it’s unique values and then count the total number of each category, the conclusion can be got that most sample values are between [2, 4].

```
>>> np.unique(df_train['BedroomAbvGr'].values)
array([0, 1, 2, 3, 4, 5, 6, 8])
>>> df_train.groupby('BedroomAbvGr').count()['Id']
BedroomAbvGr
0      6
1     50
2    353
3    798
4    208
5     21
6      7
8      1
Name: Id, dtype: int64
>>>
```

Basement full bath is basement half bath are all refer to bathroom, therefore, these two feature can be merge together as one.

```
>>> df_train.groupby('BsmtFullBath').count()['Id']
BsmtFullBath
0    854
1    577
2     12
3      1
Name: Id, dtype: int64
>>> df_train.groupby('BsmtHalfBath').count()['Id']
df_train['Bathroom']=df_train['BsmtFullBath']+ df_train[BsmtHalfBath
0    1363
1      79
2       2
Name: Id, dtype: int64
>>> df_train['Bathroom']=df_train['BsmtFullBath']+ df_train['BsmtHalfBath']
>>> df_train.groupby('Bathroom').count()['Id']
Bathroom
0    782
1    638
2     23
3      1
Name: Id, dtype: int64
>>>
```

There are totally 4 basement area related features: TotalBsmtSF, BsmtfinSF2, BsmtFinSF1 and BsmtUnfSF, they are all literally referring to the basement area square feet. The first few sample values can be observed as below snapshot, we can keep TotalbsmtSF only since it's the sum value of other 3 features.

```
>>> df_train[['TotalBsmtSF', 'BsmtFinSF2', 'BsmtFinSF1', 'BsmtUnfSF']].head()
   TotalBsmtSF  BsmtFinSF2  BsmtFinSF1  BsmtUnfSF
0          856           0         706        150
1         1262           0         978        284
2          920           0         486        434
3          756           0         216        540
4         1145           0         655        490
>>>
```

There are 3 porch related features as below snapshots showing, they can be merged together as one or keep only one of them since some of them have more than 75% values as zero. Here we can merge them together.

```
>>> df_train[['TotalBsmtSF', 'BsmtFinSF2', 'BsmtFinSF1', 'BsmtUnfSF']].head()
   TotalBsmtSF  BsmtFinSF2  BsmtFinSF1  BsmtUnfSF
0          856           0         706        150
1         1262           0         978        284
2          920           0         486        434
3          756           0         216        540
4         1145           0         655        490
>>> df_train[['OpenPorchSF', 'EnclosedPorch', 'ScreenPorch']].head()
   OpenPorchSF  EnclosedPorch  ScreenPorch
0           61              0            0
1            0              0            0
2           42              0            0
3           35            272            0
4           84              0            0
>>> df_train[['OpenPorchSF', 'EnclosedPorch', 'ScreenPorch']].describe()
       OpenPorchSF  EnclosedPorch  ScreenPorch
count  1444.000000    1444.000000  1444.000000
mean     46.160665      21.736842    15.110111
std      65.464028      60.699492    55.896012
min       0.000000       0.000000     0.000000
25%       0.000000       0.000000     0.000000
50%      24.000000       0.000000     0.000000
75%      66.500000       0.000000     0.000000
max     547.000000     552.000000   480.000000
>>> df_train['PorchSF'] = df_train['OpenPorchSF'] + df_train['EnclosedPorch'] + df_train['ScreenPorch']
>>> df_train['PorchSF'].describe()
count    1444.000000
mean       83.007618
std       101.755468
min         0.000000
25%         0.000000
50%        48.000000
75%       128.000000
max      1027.000000
Name: PorchSF, dtype: float64
>>>
```

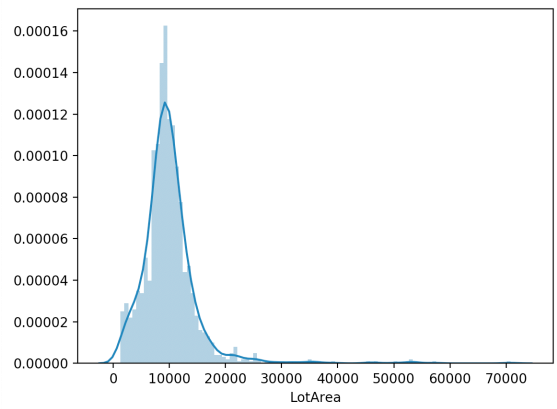
Garage area and cars must be correlated, here we keep only one of them. Similarly, garage build year is highly correlated with house build year, therefor, only house built year is kept. And we also drop the YearRemodAdd since it's highly correlated with year built.

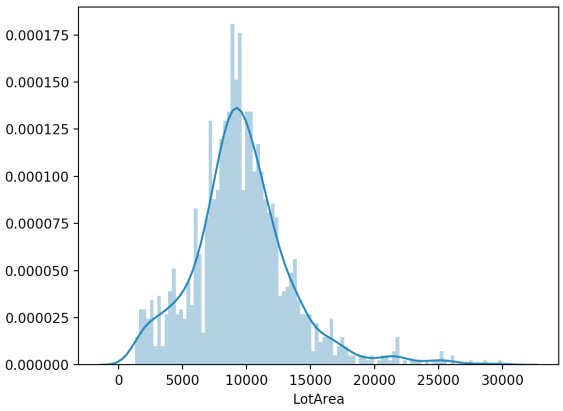
```
>>> df_train.corr()['GarageArea']['GarageCars']
0.8871173298962094
>>> df_train = df_train.drop(['GarageCars'],axis=1)
>>>
>>> df_train.corr()['GarageYrBlt']['YearBuilt']
0.8244781251158749
>>> df_train = df_train.drop(['GarageYrBlt'],axis=1)
>>>
```

Kitchens above ground is dropped since about more than 75% values are 1, which has no much values during the prediction.

```
>>> df_train['KitchenAbvGr'].describe()
count    1444.000000
mean        1.046399
std         0.220079
min         0.000000
25%         1.000000
50%         1.000000
75%         1.000000
max         3.000000
Name: KitchenAbvGr, dtype: float64
>>>
```

For Lot area, a small proportion houses which has very big lot area is going to be filtered, like the lot area is greater than 30000. Below snapshots are origin data distribution vs. filtered data distribution.





After all above processing, now the train dataset has below columns. We can check the correlation matrix between SalePrice and all other features and order by correlation value. The correlation values can only be calculated between numeric variables.

```
>>> print(df_train.columns)
Index(['Id', 'MSSubClass', 'MSZoning', 'LotArea', 'Street', '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', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MoSold', 'YrSold', 'SaleType', 'SaleCondition', 'SalePrice',
       'SqrtLotArea', 'Bathroom', 'PorchSF'],
      dtype='object')
```

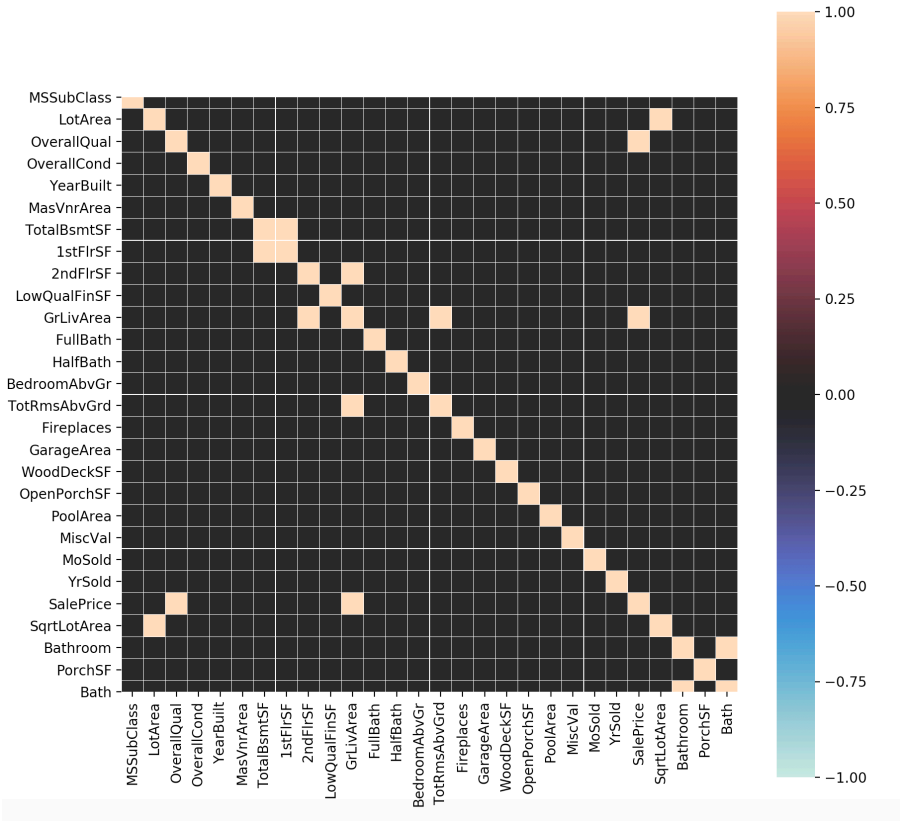
```
>>> df_train.corr()['SalePrice'].sort_values()
EnclosedPorch   -0.141146
KitchenAbvGr    -0.140459
OverallCond     -0.073748
MSSubClass      -0.072995
YrSold          -0.035372
BsmtFinSF2      -0.034692
Id              -0.027073
LowQualFinSF    -0.022872
MiscVal         -0.021022
BsmtHalfBath    -0.013022
PoolArea         0.035956
3SsnPorch        0.052916
MoSold           0.054692
ScreenPorch      0.107701
BedroomAbvGr     0.178096
Bathroom         0.187217
PorchSF          0.189630
BsmtFullBath     0.201022
BsmtUnfSF        0.234582
HalfBath         0.291308
WoodDeckSF       0.314317
2ndFlrSF         0.322433
OpenPorchSF      0.335280
BsmtFinSF1       0.358962
SqrtLotArea      0.396077
LotArea          0.397370
Fireplaces       0.460317
MasVnrArea       0.475367
GarageYrBlt      0.511604
TotRmsAbvGrd     0.528965
YearRemodAdd     0.530042
YearBuilt        0.543587
FullBath         0.569324
1stFlrSF         0.612981
TotalBsmtSF      0.631038
GarageArea       0.645251
GarageCars       0.652021
GrLivArea        0.720624
OverallQual      0.804666
SalePrice        1.000000
Name: SalePrice, dtype: float64
```

Since the final result SalePrice is a linear regression, a new data frame is defined which only have the numeric variables including SalePrice. And some variable are dropped as mentioned during above process.

```
>>> num_attrs = df_train.select_dtypes([np.int64, np.float64]).columns.values
>>> df_train_num= df_train[num_attrs]
>>> df_train_num['Bath']= df_train_num['BsmtFullBath'] + df_train_num['BsmtHalfBath']
>>> df_train_num=df_train_num.drop(['Id','3SsnPorch','BsmtFinSF2','BsmtFinSF1','BsmtUnfSF','EnclosedPorch','ScreenPorch','GarageCars',
...                                 'GarageYrBlt','KitchenAbvGr','YearRemodAdd','BsmtFullBath', 'BsmtHalfBath'],axis=1)
>>>
>>> print(df_train_num.columns)
Index(['MSSubClass', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt',
       'MasVnrArea', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'TotRmsAbvGrd',
       'Fireplaces', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'PoolArea',
       'MiscVal', 'MoSold', 'YrSold', 'SalePrice', 'SqrtLotArea', 'Bathroom',
       'PorchSF', 'Bath'],
      dtype='object')
>>>
```

```
>>> df_train_num.corr()['SalePrice'].sort_values()
OverallCond    -0.073748
MSSubClass     -0.072995
YrSold         -0.035372
LowQualFinSF   -0.022872
MiscVal        -0.021022
PoolArea        0.035956
MoSold          0.054692
BedroomAbvGr    0.178096
Bath            0.187217
Bathroom        0.187217
PorchSF         0.189630
HalfBath        0.291308
WoodDeckSF      0.314317
2ndFlrSF        0.322433
OpenPorchSF     0.335280
SqrtLotArea     0.396077
LotArea         0.397370
Fireplaces      0.460317
MasVnrArea      0.475367
TotRmsAbvGrd    0.528965
YearBuilt       0.543587
FullBath        0.569324
1stFlrSF        0.612981
TotalBsmtSF     0.631038
GarageArea      0.645251
GrLivArea       0.720624
OverallQual     0.804666
SalePrice       1.000000
Name: SalePrice, dtype: float64
>>>
```

To make the correlation matrix more clear, a lambda function is applied to define the correlation value as 1 if its absolute value is greater than 0.7 and 0 otherwise. From this figure, it's shown that TotalBsmeSF is highly correlated with 1sfFlrSF, so do GrLiveArea and 2ndFlrSF and LotArea and SqrtLotArea. Therefore, 'TotRmsAbvGrd' , '1stFlrSF' and 'LotArea'are dropped.



After more dropping processing, below features are remained. Now we only choose features with correlation values are greater than 0.5.

```
>>> df_train_num=df_train_num.drop(['LotArea'],axis=1)
>>> print(df_train_num.columns)
Index(['MSSubClass', 'OverallQual', 'OverallCond', 'YearBuilt', 'MasVnrArea',
       'TotalBsmtSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'Fireplaces', 'GarageArea', 'WoodDeckSF',
       'OpenPorchSF', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'SalePrice',
       'SqrtLotArea', 'Bathroom', 'PorchSF', 'Bath'],
      dtype='object')
>>> df_train_num.corr()['SalePrice'].sort_values()
OverallCond    -0.073748
MSSubClass     -0.072995
YrSold         -0.035372
LowQualFinSF   -0.022872
MiscVal        -0.021022
PoolArea        0.035956
MoSold          0.054692
BedroomAbvGr    0.178096
Bathroom        0.187217
Bath            0.187217
PorchSF         0.189630
HalfBath        0.291308
WoodDeckSF      0.314317
2ndFlrSF        0.322433
OpenPorchSF     0.335280
SqrtLotArea     0.396077
Fireplaces      0.460317
MasVnrArea      0.475367
YearBuilt       0.543587
FullBath        0.569324
TotalBsmtSF     0.631038
GarageArea      0.645251
GrLivArea       0.720624
OverallQual     0.804666
SalePrice       1.000000
```

```
>>> df_train_num=df_train_num[df_train_num.columns[df_train_num.corr()['SalePrice']>0.5]]
>>> df_train_num.columns
Index(['OverallQual', 'YearBuilt', 'TotalBsmtSF', 'GrLivArea', 'FullBath',
       'GarageArea', 'SalePrice'],
      dtype='object')
>>>
```

Therefore, the final features selected for next modelling and prediction are: 'OverallQual', 'YearBuilt', 'TotalBsmtSF', 'GrLivArea', ‘FullBath', ‘GarageArea’.

4. Model Building

Linear Regression model is applied since the result of prediction is the house SalePrice which is continuous number.

Linear_model library can be used from sklearn. Split the input variables and output variable ‘SalePrice’. Then train the model.

```
>>> from sklearn import linear_model
>>> reg= linear_model.LinearRegression()
>>> df_train_num_x=df_train_num.drop('SalePrice',axis=1)
>>> df_train_num_y=df_train_num['SalePrice']
>>> reg.fit(df_train_num_x, df_train_num_y)
LinearRegression()
>>>
```

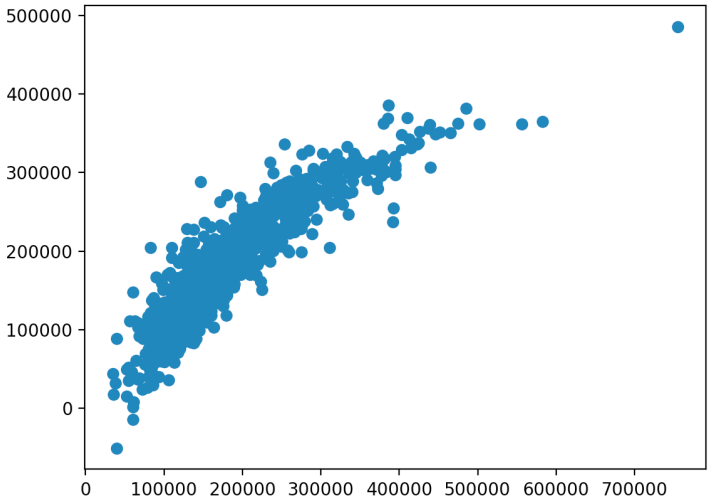
The related coefficient for each features are as follows:

```
>>> print('Coefficients: \n', reg.coef_)
Coefficients:
 [18477.95062528   397.39088293    36.54463317    60.50191137
 -8653.1813531     45.34474976]
>>> print(df_train_num_x.columns)
Index(['OverallQual', 'YearBuilt', 'TotalBsmtSF', 'GrLivArea', 'FullBath',
       'GarageArea'],
      dtype='object')
>>>
```

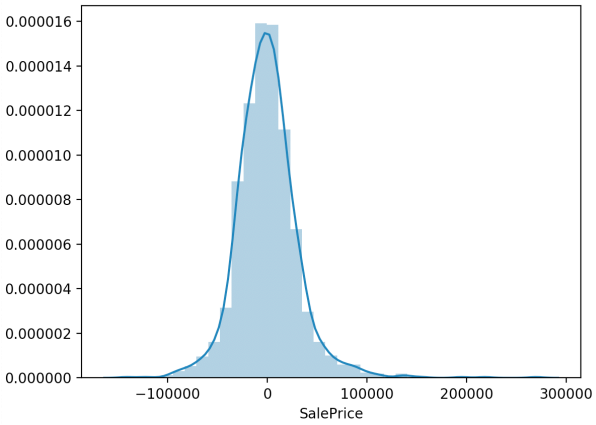
Get the prediction based on the training dataset and then we can get the error between prediction and real house SalePrice, and as we all know for the error, the fewer the better.

```
>>> preds = reg.predict(df_train_num_x)
>>> from sklearn import metrics
>>> print('MAE:', metrics.mean_absolute_error(df_train_num_y, preds))
MAE: 22283.120663727463
>>> print('MSE:', metrics.mean_squared_error(df_train_num_y, preds))
MSE: 994337925.797842
>>> print('RMSE:', np.sqrt(metrics.mean_squared_error(df_train_num_y, preds)))
RMSE: 31533.124263190955
>>>
```

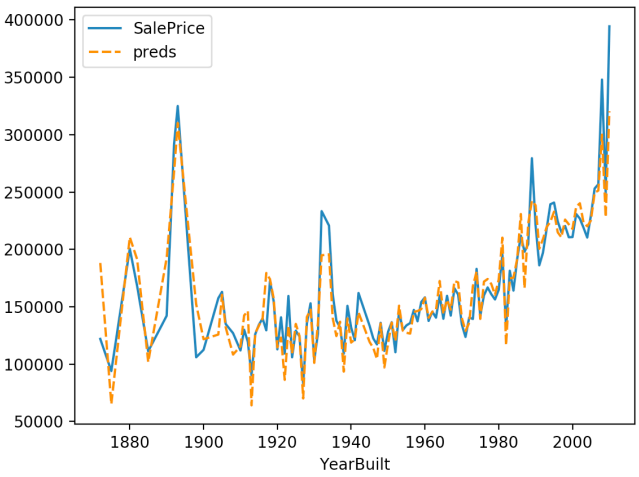
A scatter figure can be plotted to show the reaction between real SalePrice(x axis) and predictions(y axis). From this scatter we can see that the prediction value is very close to the actual house SalePrice when it's between $100000 and $400000, and when the actual house price is greater than $400000, the predicted SalePrice is much lower than the actual SalePrice.



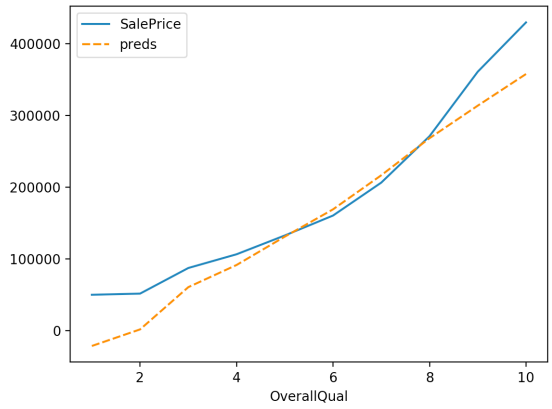
The difference between the predicted and the actual SalePrice can also be plotted as below snapshots. Obviously, about 99% SalePrice difference is mainly between -$100000 and $100000.

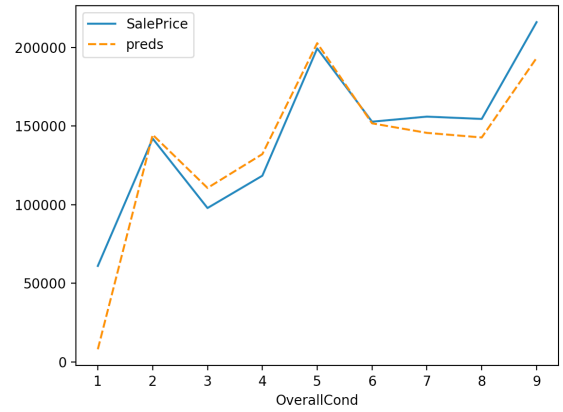


The line plot between year built and predicted and actual SalePrice can also be plotted as snapshot showing: mostly the predicted SalePrice is aligned with the actual house SalePrice.

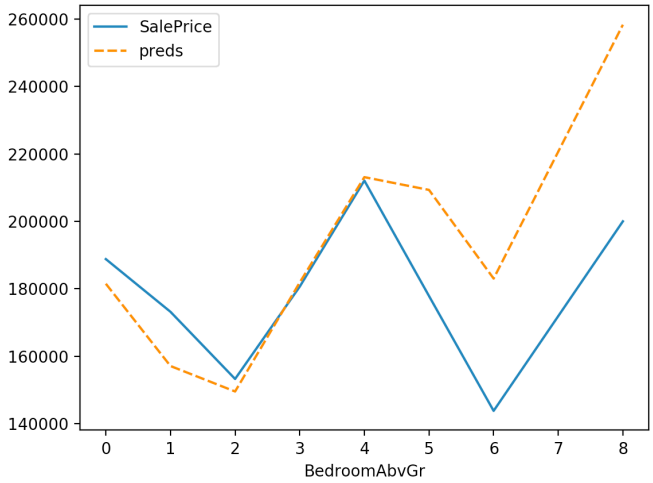


Similarly line plot between the overall quality and predicted and actual SalePrice and the line plot between the overall condition and predicted and actual SalePrice:





The line plot between the bedrooms and predicted and actual SalePrice:



The predicted and actual SalePrice has no much difference when the bedroom number is between 2 to 4, he predicted SalePrice is lower than actual ones when the bedroom number is less than 2, and the predicted SalePrice is much higher than the actual price when the bedroom number are are than 4.

Lasso model can be used to replace the linear model with no other changes applied in data processing part. And the house SalePrice prediction becomes different. The public score might be refer to the error rate, and the Lasso version predicted result is much better than the linear regression prediction.

Submission and Description	Public Score
LassoCV_submission.csv just now by LJing Lasso version submission	0.22413
test_submit.csv a minute ago by LJing add submission details	0.72799