ID: A1822

Submission Date: 30/05/2019

Kaggle House Price Assignment

Abstract:

The “House Prices: Advanced Regression Techniques” on Kaggle is a regression problem. It is a “playground” problem with a sizable dataset. Proper data preprocessing, mining and feature engineering is crucial for getting good predictions on this data. A shallow fully connected NN (via TensorFlow low level API) was the selected model for this assignment. As a starting point and base guideline Dataquest’s “Getting Started with Kaggle: House Prices Competition” and the Kaggle notebook of user “surya635” and "humananalog" was used. Majority of feature engineering techniques were discovered by reading various blogs/articles. [1][5][8]

1. Data investigation and pre-processing:

1.1 Feature info

In [45]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

from IPython.display import HTML
import base64

# function that takes in a dataframe and creates a text link to
# download it (will only work for files < 2MB or so)
def create_download_link(df, title = "Download CSV file", filename = "data.csv"):
    csv = df.to_csv()
    b64 = base64.b64encode(csv.encode())
    payload = b64.decode()
    html = '<a download="{filename}" href="data:text/csv;base64,{payload}" target="_blank">{title}</a>'
    html = html.format(payload=payload,title=title,filename=filename)
    return HTML(html)

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
import math
from scipy import stats
from scipy.stats import skew  # for some statistics
from scipy.special import boxcox1p
from scipy.stats import boxcox_normmax
from sklearn.metrics import r2_score



%matplotlib inline
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os
print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.

# Importing of train and test data from Kaggle folder
# Load the data.
train_df = pd.read_csv("../input/train.csv")
train2 = pd.read_csv("../input/train.csv")
test_df = pd.read_csv("../input/test.csv")
test2 = pd.read_csv("../input/test.csv")



# Attribute info
train_df.shape
def rstr(df, pred=None):
    obs = df.shape[0]
    types = df.dtypes
    counts = df.apply(lambda x: x.count())
    uniques = df.apply(lambda x: [x.unique()])
    nulls = df.apply(lambda x: x.isnull().sum())
```

```
    distincts = df.apply(lambda x: x.unique().shape[0])
    missing_ratio = (df.isnull().sum()/ obs) * 100
    skewness = df.skew()
    kurtosis = df.kurt()
    print('Data shape:', df.shape)

    if pred is None:
        cols = ['types', 'counts', 'distincts', 'nulls', 'missing ratio', 'uniques', 's
kewness', 'kurtosis']
        str = pd.concat([types, counts, distincts, nulls, missing_ratio, uniques, skewn
ess, kurtosis], axis = 1)

    else:
        corr = df.corr()[pred]
        str = pd.concat([types, counts, distincts, nulls, missing_ratio, uniques, skewn
ess, kurtosis, corr], axis = 1, sort=False)
        corr_col = 'corr '  + pred
        cols = ['types', 'counts', 'distincts', 'nulls', 'missing_ratio', 'uniques', 's
kewness', 'kurtosis', corr_col ]

    str.columns = cols
    dtypes = str.types.value_counts()
    print('_____________________________\nData types:\n',str.types.value_counts())
    print('_____________________________')
    return str

details = rstr(train_df, 'SalePrice')
display(details.sort_values(by='corr SalePrice', ascending=False))


#plot of missing value attributes
plt.figure(figsize=(12, 6))
sns.heatmap(train_df.isnull())
plt.show()
```

```
['test.csv', 'train.csv', 'sample_submission.csv', 'data_description.txt']
Data shape: (1460, 81)
_____________________________
Data types:
 object     43
int64      35
float64     3
Name: types, dtype: int64
_____________________________
```

	types	counts	distincts	nulls	missing_ratio	uniques	skewness	kur
SalePrice	int64	1460	663	0	0.000000	[[208500, 181500, 223500, 140000, 250000, 1430...	1.882876	6.53
OverallQual	int64	1460	10	0	0.000000	[[7, 6, 8, 5, 9, 4, 10, 3, 1, 2]]	0.216944	0.09
GrLivArea	int64	1460	861	0	0.000000	[[1710, 1262, 1786, 1717, 2198, 1362, 1694, 20...	1.366560	4.89
GarageCars	int64	1460	5	0	0.000000	[[2, 3, 1, 0, 4]]	-0.342549	0.22
GarageArea	int64	1460	441	0	0.000000	[[548, 460, 608, 642, 836, 480, 636, 484, 468,...	0.179981	0.91
TotalBsmtSF	int64	1460	721	0	0.000000	[[856, 1262, 920, 756, 1145, 796, 1686, 1107, ...	1.524255	13.25
1stFlrSF	int64	1460	753	0	0.000000	[[856, 1262, 920, 961, 1145, 796, 1694, 1107, ...	1.376757	5.74
FullBath	int64	1460	4	0	0.000000	[[2, 1, 3, 0]]	0.036562	-0.85
TotRmsAbvGrd	int64	1460	12	0	0.000000	[[8, 6, 7, 9, 5, 11, 4, 10, 12, 3, 2, 14]]	0.676341	0.88
YearBuilt	int64	1460	112	0	0.000000	[[2003, 1976, 2001, 1915, 2000, 1993, 2004, 19...	-0.613461	-0.43
YearRemodAdd	int64	1460	61	0	0.000000	[[2003, 1976, 2002, 1970, 2000, 1995, 2005, 19...	-0.503562	-1.27
GarageYrBlt	float64	1379	98	81	5.547945	[[2003.0, 1976.0, 2001.0, 1998.0, 2000.0, 1993...	-0.649415	-0.41
MasVnrArea	float64	1452	328	8	0.547945	[[196.0, 0.0, 162.0, 350.0, 186.0, 240.0, 286....	2.669084	10.08

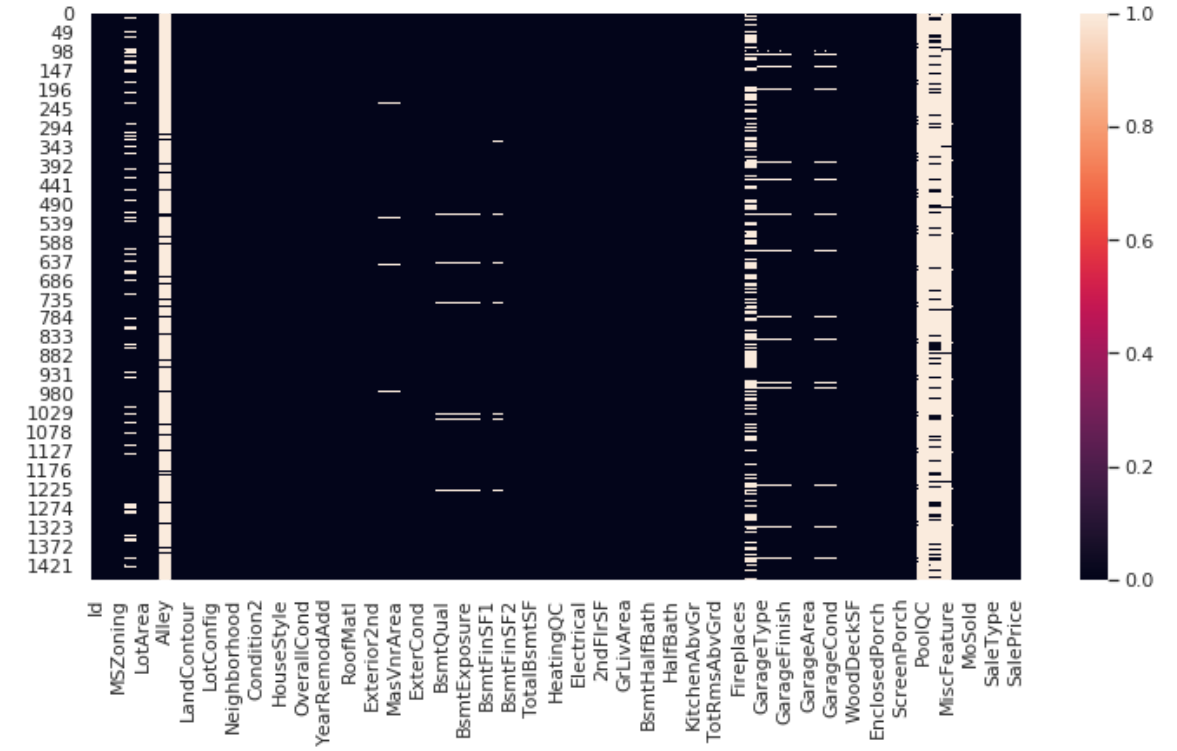
	types	counts	distincts	nulls	missing_ratio	uniques	skewness	kur
Fireplaces	int64	1460	4	0	0.000000	[[0, 1, 2, 3]]	0.649565	-0.21
BsmtFinSF1	int64	1460	637	0	0.000000	[[706, 978, 486, 216, 655, 732, 1369, 859, 0, ...	1.685503	11.11
LotFrontage	float64	1201	111	259	17.739726	[[65.0, 80.0, 68.0, 60.0, 84.0, 85.0, 75.0, na...	2.163569	17.45
WoodDeckSF	int64	1460	274	0	0.000000	[[0, 298, 192, 40, 255, 235, 90, 147, 140, 160...	1.541376	2.99
2ndFlrSF	int64	1460	417	0	0.000000	[[854, 0, 866, 756, 1053, 566, 983, 752, 1142,...	0.813030	-0.55
OpenPorchSF	int64	1460	202	0	0.000000	[[61, 0, 42, 35, 84, 30, 57, 204, 4, 21, 33, 2...	2.364342	8.49
HalfBath	int64	1460	3	0	0.000000	[[1, 0, 2]]	0.675897	-1.07
LotArea	int64	1460	1073	0	0.000000	[[8450, 9600, 11250, 9550, 14260, 14115, 10084...	12.207688	203.24
BsmtFullBath	int64	1460	4	0	0.000000	[[1, 0, 2, 3]]	0.596067	-0.83
BsmtUnfSF	int64	1460	780	0	0.000000	[[150, 284, 434, 540, 490, 64, 317, 216, 952, ...	0.920268	0.47
BedroomAbvGr	int64	1460	8	0	0.000000	[[3, 4, 1, 2, 0, 5, 6, 8]]	0.211790	2.23
ScreenPorch	int64	1460	76	0	0.000000	[[0, 176, 198, 291, 252, 99, 184, 168, 130, 14...	4.122214	18.43
PoolArea	int64	1460	8	0	0.000000	[[0, 512, 648, 576, 555, 480, 519, 738]]	14.828374	223.26
MoSold	int64	1460	12	0	0.000000	[[2, 5, 9, 12, 10, 8, 11, 4, 1, 7, 3, 6]]	0.212053	-0.40
3SsnPorch	int64	1460	20	0	0.000000	[[0, 320, 407, 130, 180, 168, 140, 508, 238, 2...	10.304342	123.66

	types	counts	distincts	nulls	missing_ratio	uniques	skewness	kur
BsmtFinSF2	int64	1460	144	0	0.000000	[[0, 32, 668, 486, 93, 491, 506, 712, 362, 41,...	4.255261	20.1
BsmtHalfBath	int64	1460	3	0	0.000000	[[0, 1, 2]]	4.103403	16.3
...	...	...	...	...	...	...	...	
RoofStyle	object	1460	6	0	0.000000	[[Gable, Hip, Gambrel, Mansard, Flat, Shed]]	NaN	
RoofMatl	object	1460	8	0	0.000000	[[CompShg, WdShngl, Metal, WdShake, Membran, T...	NaN	
Exterior1st	object	1460	15	0	0.000000	[[VinylSd, MetalSd, Wd Sdng, HdBoard, BrkFace,...	NaN	
Exterior2nd	object	1460	16	0	0.000000	[[VinylSd, MetalSd, Wd Shng, HdBoard, Plywood,...	NaN	
MasVnrType	object	1452	5	8	0.547945	[[BrkFace, None, Stone, BrkCmn, nan]]	NaN	
ExterQual	object	1460	4	0	0.000000	[[Gd, TA, Ex, Fa]]	NaN	
ExterCond	object	1460	5	0	0.000000	[[TA, Gd, Fa, Po, Ex]]	NaN	
Foundation	object	1460	6	0	0.000000	[[PConc, CBlock, BrkTil, Wood, Slab, Stone]]	NaN	
BsmtQual	object	1423	5	37	2.534247	[[Gd, TA, Ex, nan, Fa]]	NaN	
BsmtCond	object	1423	5	37	2.534247	[[TA, Gd, nan, Fa, Po]]	NaN	
BsmtExposure	object	1422	5	38	2.602740	[[No, Gd, Mn, Av, nan]]	NaN	
BsmtFinType1	object	1423	7	37	2.534247	[[GLQ, ALQ, Unf, Rec, BLQ, nan, LwQ]]	NaN	

	types	counts	distincts	nulls	missing_ratio	uniques	skewness	kur
BsmtFinType2	object	1422	7	38	2.602740	[[Unf, BLQ, nan, ALQ, Rec, LwQ, GLQ]]	NaN	
Heating	object	1460	6	0	0.000000	[[GasA, GasW, Grav, Wall, OthW, Floor]]	NaN	
HeatingQC	object	1460	5	0	0.000000	[[Ex, Gd, TA, Fa, Po]]	NaN	
CentralAir	object	1460	2	0	0.000000	[[Y, N]]	NaN	
Electrical	object	1459	6	1	0.068493	[[SBrkr, FuseF, FuseA, FuseP, Mix, nan]]	NaN	
KitchenQual	object	1460	4	0	0.000000	[[Gd, TA, Ex, Fa]]	NaN	
Functional	object	1460	7	0	0.000000	[[Typ, Min1, Maj1, Min2, Mod, Maj2, Sev]]	NaN	
FireplaceQu	object	770	6	690	47.260274	[[nan, TA, Gd, Fa, Ex, Po]]	NaN	
GarageType	object	1379	7	81	5.547945	[[Attchd, Detchd, BuiltIn, CarPort, nan, Basme...	NaN	
GarageFinish	object	1379	4	81	5.547945	[[RFn, Unf, Fin, nan]]	NaN	
GarageQual	object	1379	6	81	5.547945	[[TA, Fa, Gd, nan, Ex, Po]]	NaN	
GarageCond	object	1379	6	81	5.547945	[[TA, Fa, nan, Gd, Po, Ex]]	NaN	
PavedDrive	object	1460	3	0	0.000000	[[Y, N, P]]	NaN	
PoolQC	object	7	4	1453	99.520548	[[nan, Ex, Fa, Gd]]	NaN	
Fence	object	281	5	1179	80.753425	[[nan, MnPrv, GdWo, GdPrv, MnWw]]	NaN	
MiscFeature	object	54	5	1406	96.301370	[[nan, Shed, Gar2, Othr, TenC]]	NaN	

	types	counts	distincts	nulls	missing_ratio	uniques	skewness	kui
SaleType	object	1460	9	0	0.000000	[[WD, New, COD, ConLD, ConLI, CWD, ConLw, Con,...	NaN	
SaleCondition	object	1460	6	0	0.000000	[[Normal, Abnorml, Partial, AdjLand, Alloca, F...	NaN	

81 rows × 9 columns



Cell Report:

1. Our training DataFrame was broken down attribute wise to know the nature of the data, which was used for preprocessing.
2. From the detailed table: Types indicates the attribute data type, Counts indiacates number of time the attribute is present, Distincts is the number of unique values of a feature, Uniques lists some of the Distincts, Skewness is asymmetry in a feature's distrubution and Kurtosis is the peakiness of the distribution. Saleprice correlation with each of the other attributes was shown in decending order.
3. Number of Missing Values were graphically represented using Seaborn.
4. Though some features like 'PoolQC', 'MiscFeature', 'Alley' and 'Fence' had a lot of missing values, they were not dropped because those missing values indicate a meaning and can be categorized(One Hot Encoded) or Label Encoded.

Preprocessing & Feature Engineering

1.2 Target variable

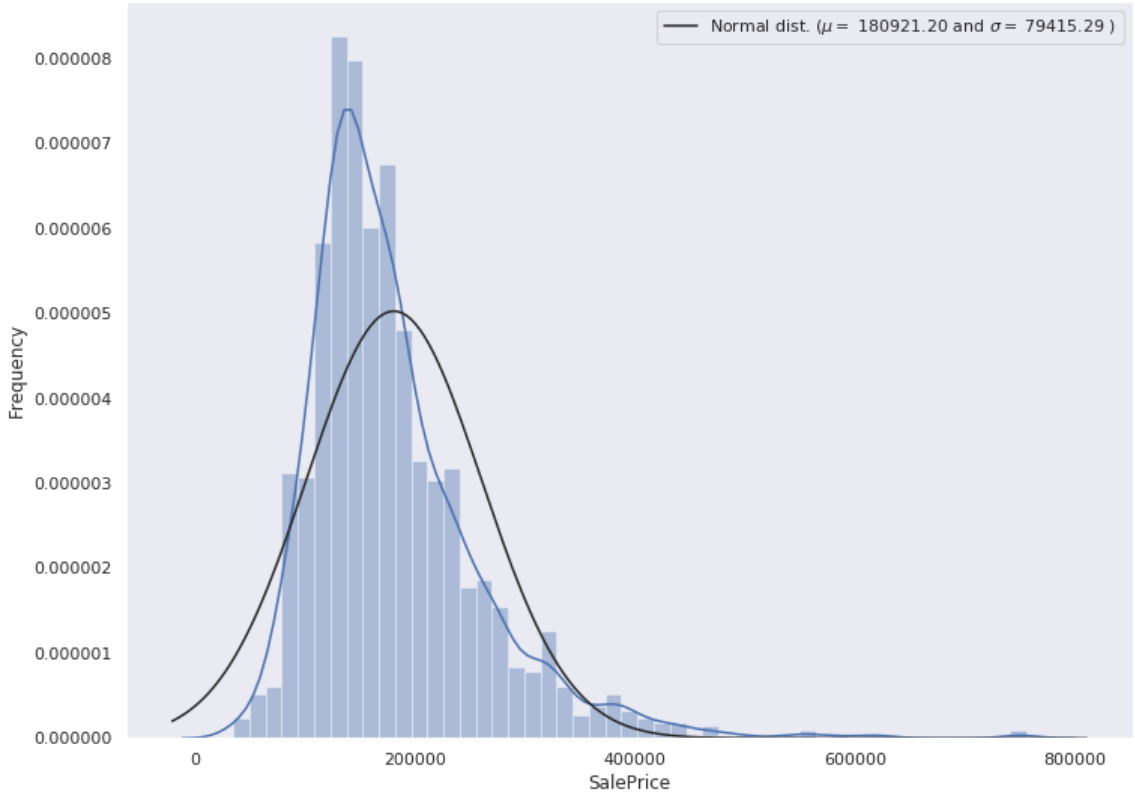
In [46]:

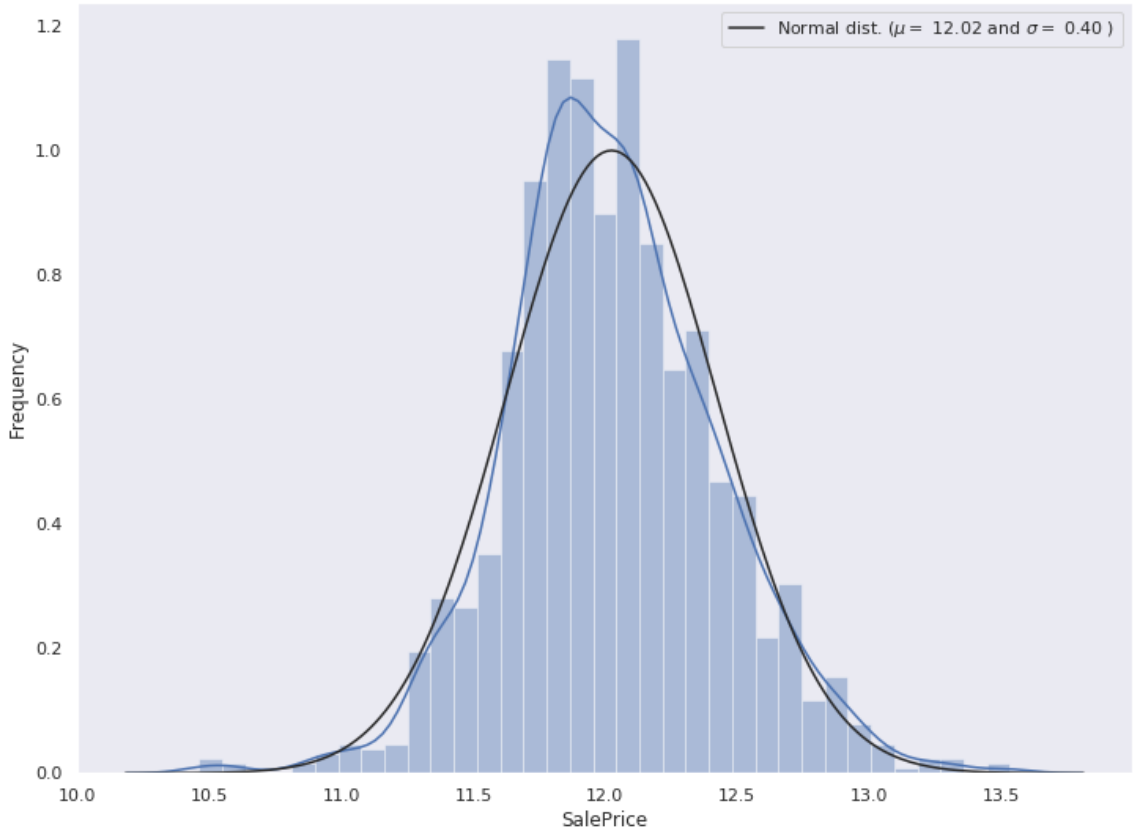
```
plt.subplots(figsize=(12,9))
sns.distplot(train_df['SalePrice'], fit=stats.norm)

# Get the fitted parameters used by the function

(mu, sigma) = stats.norm.fit(train_df['SalePrice'])

# plot with the distribution

plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)], lo
c='best')
plt.ylabel('Frequency')

# fix distribution
train_df['SalePrice'] = np.log1p(train_df['SalePrice'])
train2['SalePrice'] = np.log1p(train2['SalePrice'])
plt.subplots(figsize=(12,9))
sns.distplot(train_df['SalePrice'], fit=stats.norm)

# Get the fitted parameters used by the function

(mu, sigma) = stats.norm.fit(train_df['SalePrice'])

# plot with the distribution

plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)], lo
c='best')
plt.ylabel('Frequency')
```

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWa
rning: Using a non-tuple sequence for multidimensional indexing is depreca
ted; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result
either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[46]:

```
Text(0, 0.5, 'Frequency')
```





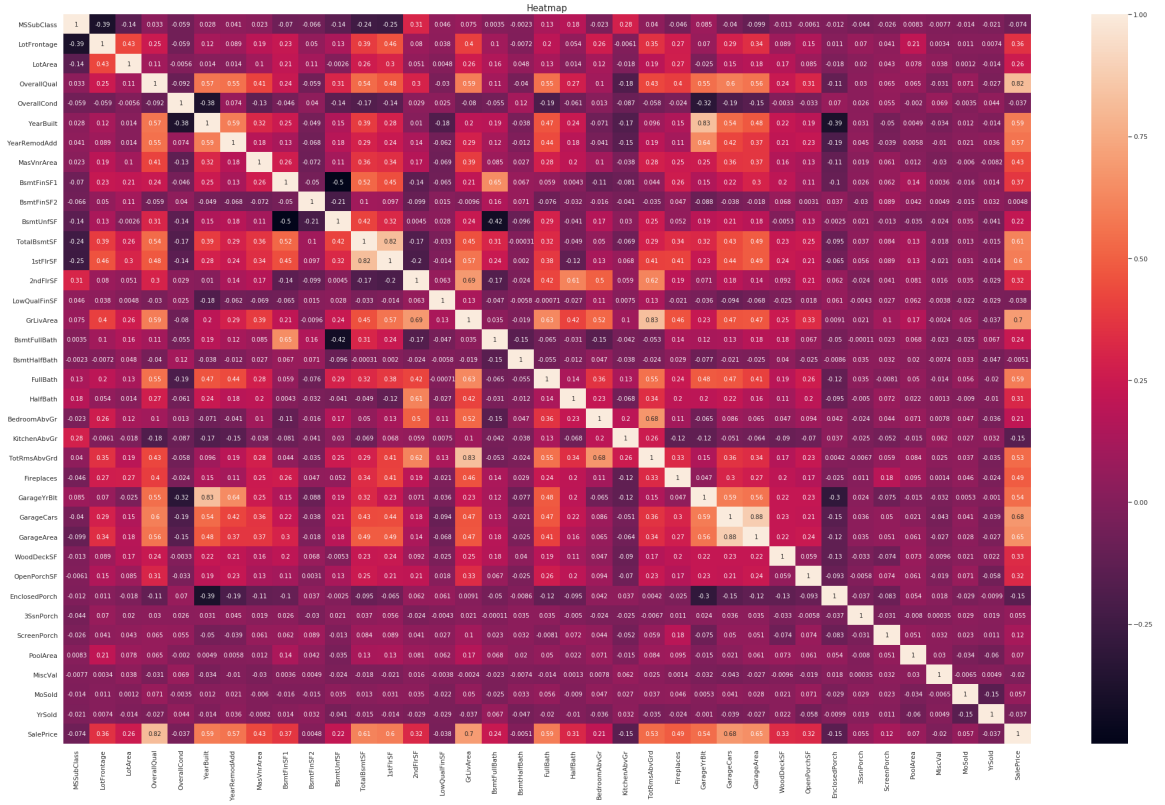
Cell Report:

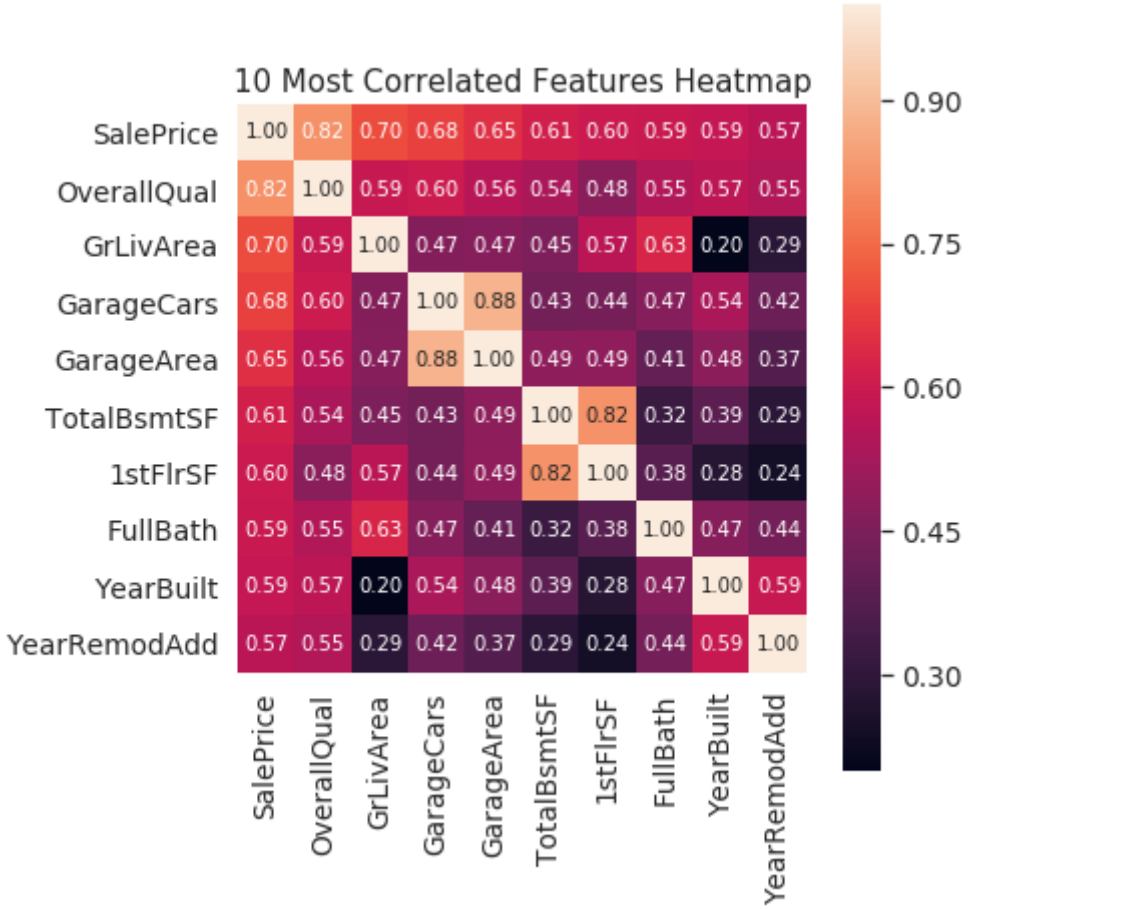
1. Since the **distribution** of the target variable was **skewed** to the left, log function was used to make the distribution more **normal**. Unskewed bell curve will be easier for regression model to approximate.

1.3 Data Correlation

In [47]:

```
#Separate variable into new dataframe from original dataframe which has only numerical
 values
#there is 38 numerical attribute from 81 attributes
train_corr = train_df.select_dtypes(include=[np.number])
#Delete Id because that is not need for corralation plot
del train_corr['Id']
#Coralation plot
corr = train_corr.corr()
plt.subplots(figsize=(30,19))
sns.heatmap(corr,annot=True)
plt.title('Heatmap', fontsize=15)
plt.tight_layout()
plt.show()

#saleprice correlation matrix
k = 10 #number of variables for heatmap
cols = corr.nlargest(k, 'SalePrice')['SalePrice'].index
cm = np.corrcoef(train_df[cols].values.T)
sns.set(font_scale=1.25)
plt.subplots(figsize=(7,7))
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size':
10}, yticklabels=cols.values, xticklabels=cols.values)
plt.title('10 Most Correlated Features Heatmap', fontsize=15)
plt.tight_layout()
plt.show()
```


Heatmap
MSSubClass
LotFrontage
LotArea
OverallQual
OverallCond
YearBuilt
YearRemodAdd
MasVnrArea
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
LowQualFinSF
GrLivArea
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageYrBlt
GarageCars
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
ScreenPorch
PoolArea
MiscVal
MoSold
YrSold
SalePrice
1.00
0.75
0.50
0.25
0.00
−0.25


10 Most Correlated Features Heatmap
SalePrice 1.00 0.82 0.70 0.68 0.65 0.61 0.60 0.59 0.59 0.57
OverallQual 0.82 1.00 0.59 0.60 0.56 0.54 0.48 0.55 0.57 0.55
GrLivArea 0.70 0.59 1.00 0.47 0.47 0.45 0.57 0.63 0.20 0.29
GarageCars 0.68 0.60 0.47 1.00 0.88 0.43 0.44 0.47 0.54 0.42
GarageArea 0.65 0.56 0.47 0.88 1.00 0.49 0.49 0.41 0.48 0.37
TotalBsmtSF 0.61 0.54 0.45 0.43 0.49 1.00 0.82 0.32 0.39 0.29
1stFlrSF 0.60 0.48 0.57 0.44 0.49 0.82 1.00 0.38 0.28 0.24
FullBath 0.59 0.55 0.63 0.47 0.41 0.32 0.38 1.00 0.47 0.44
YearBuilt 0.59 0.57 0.20 0.54 0.48 0.39 0.28 0.47 1.00 0.59
YearRemodAdd 0.57 0.55 0.29 0.42 0.37 0.29 0.24 0.44 0.59 1.00
SalePrice
OverallQual
GrLivArea
GarageCars
GarageArea
TotalBsmtSF
1stFlrSF
FullBath
YearBuilt
YearRemodAdd
0.90
0.75
0.60
0.45
0.30

Cell Report:

1. Heatmap was made with the Correlation Matrix of the numerical features.
2. There are some red colored squares that emmerge from the heatmap. The first one refers to the 'TotalBsmtSF' and '1stFlrSF' variables, and the second one refers to the 'GarageX' variables. Both cases show how significant the correlation is between these variables. Actually, this correlation is so strong that it can indicate a situation of multicollinearity.
3. As for SalePrice correlations, GrLivArea', 'TotalBsmtSF', and 'OverallQual' are very much lit up in the heatmap.
4. From the 10 Most Correlated Features Heatmap we can observe that 'OverallQual', 'GrLivArea' and 'TotalBsmtSF' are strongly correlated with 'SalePrice'. 'GarageCars' and 'GarageArea' are also some of the most strongly correlated variables. 'TotalBsmtSF' and '1stFloor' have same correlation as do 'TotRmsAbvGrd' and 'GrLivArea'. 'YearBuilt' is slightly correlated

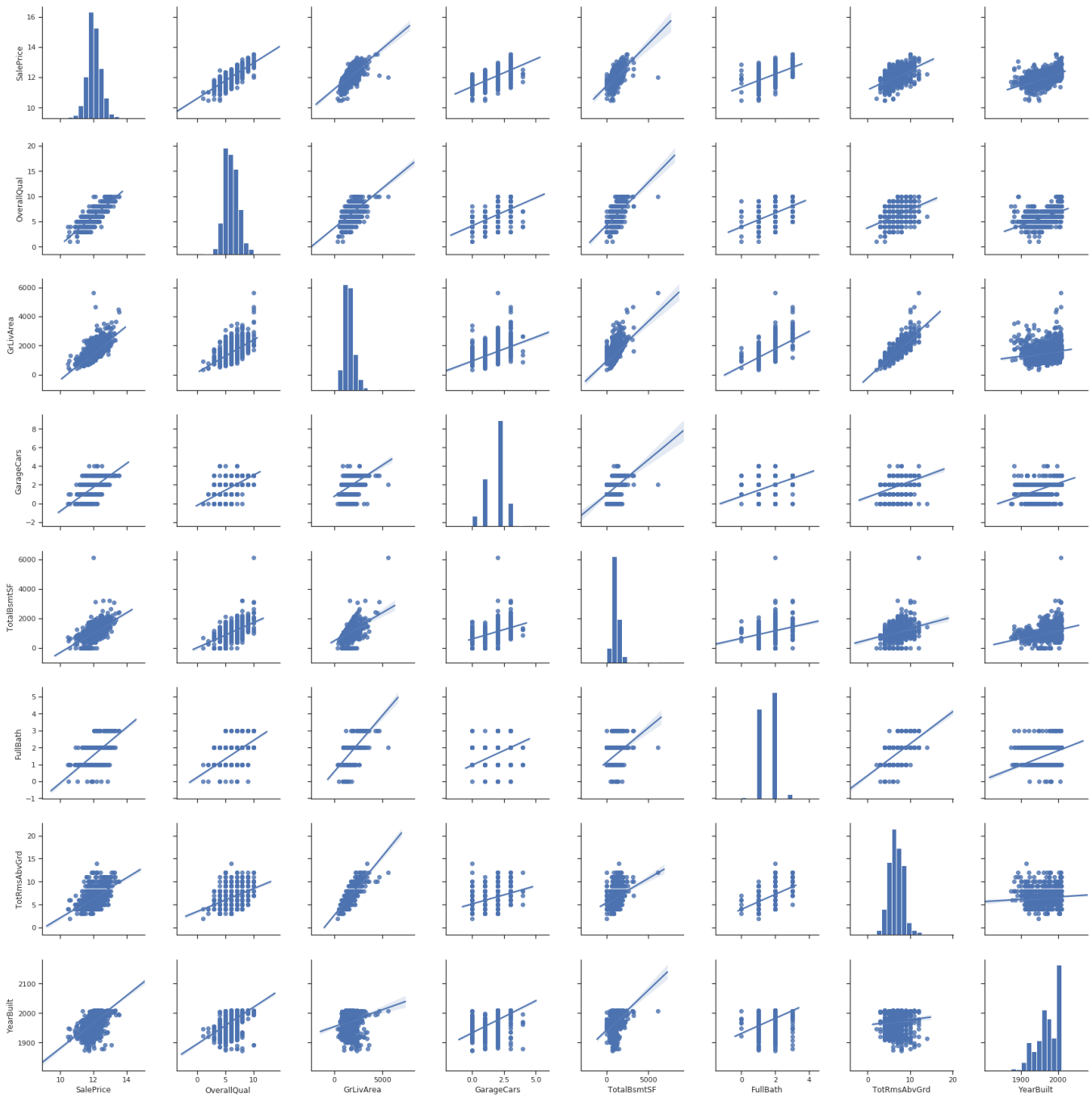
1.4 Visualization

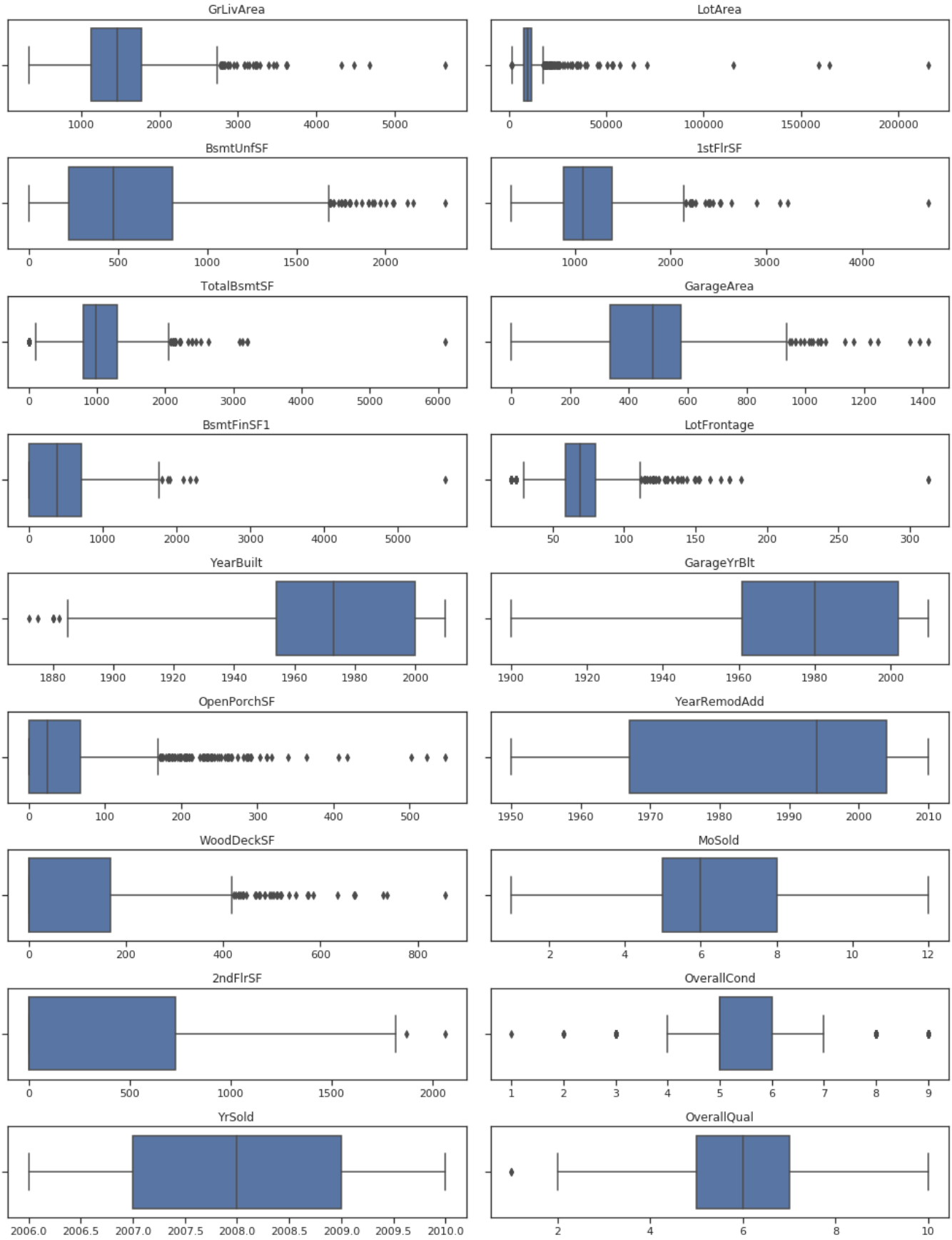
In [ ]:

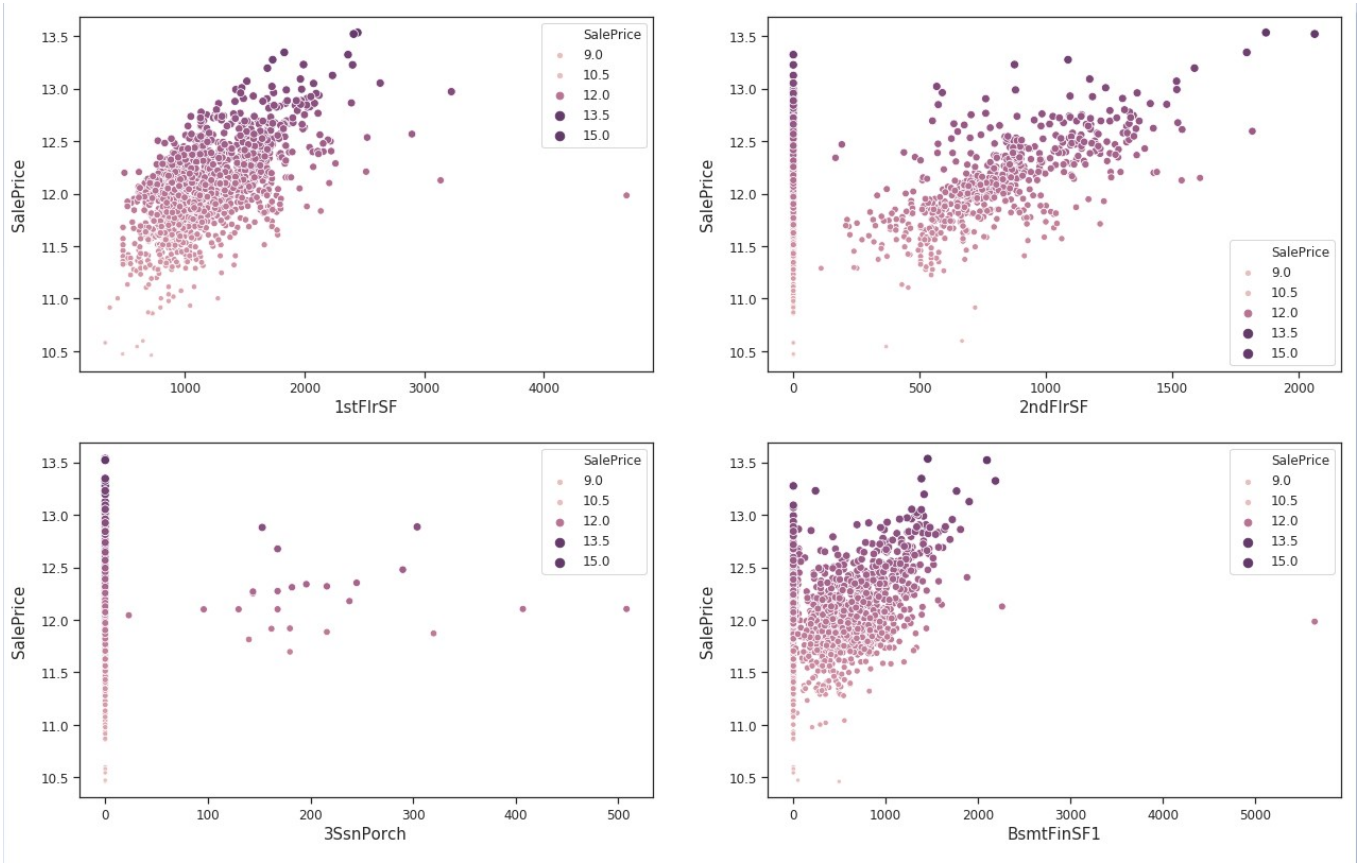
```
col = ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'FullBath'
, 'TotRmsAbvGrd', 'YearBuilt']
sns.set(style='ticks')
print('\nPairplot')
print('\nBoxplots')
print('\nNum Scatterplot')
print('\nNum Swarmplot')
sns.pairplot(train_df[col], height=3, kind='reg')
plt.tight_layout()
plt.show();



selected = ['GrLivArea',
 'LotArea',
 'BsmtUnfSF',
 '1stFlrSF',
 'TotalBsmtSF',
 'GarageArea',
 'BsmtFinSF1',
 'LotFrontage',
 'YearBuilt',
 'Neighborhood',
 'GarageYrBlt',
 'OpenPorchSF',
 'YearRemodAdd',
 'WoodDeckSF',
 'MoSold',
 '2ndFlrSF',
 'OverallCond',
 'Exterior1st',
 'YrSold',
 'OverallQual']

trainV = train2[selected].copy()
trainV['is_train'] = 1
trainV['SalePrice'] = train2['SalePrice'].values
trainV['Id'] = train2['Id'].values

testV = test2[selected].copy()
testV['is_train'] = 0
testV['SalePrice'] = 1  #dummy value
testV['Id'] = test2['Id'].values

full = pd.concat([trainV, testV])

not_features = ['Id', 'SalePrice', 'is_train']
features = [c for c in trainV.columns if c not in not_features]

cols = trainV[features].select_dtypes([np.float64, np.int64]).columns
n_rows = math.ceil(len(cols)/2)
print('\nBoxplots')
fig, ax = plt.subplots(n_rows, 2, figsize=(14, n_rows*2))
ax = ax.flatten()
for i,c in enumerate(cols):
    sns.boxplot(x=trainV[c], ax=ax[i])
    ax[i].set_title(c)
    ax[i].set_xlabel("")
plt.tight_layout()
```

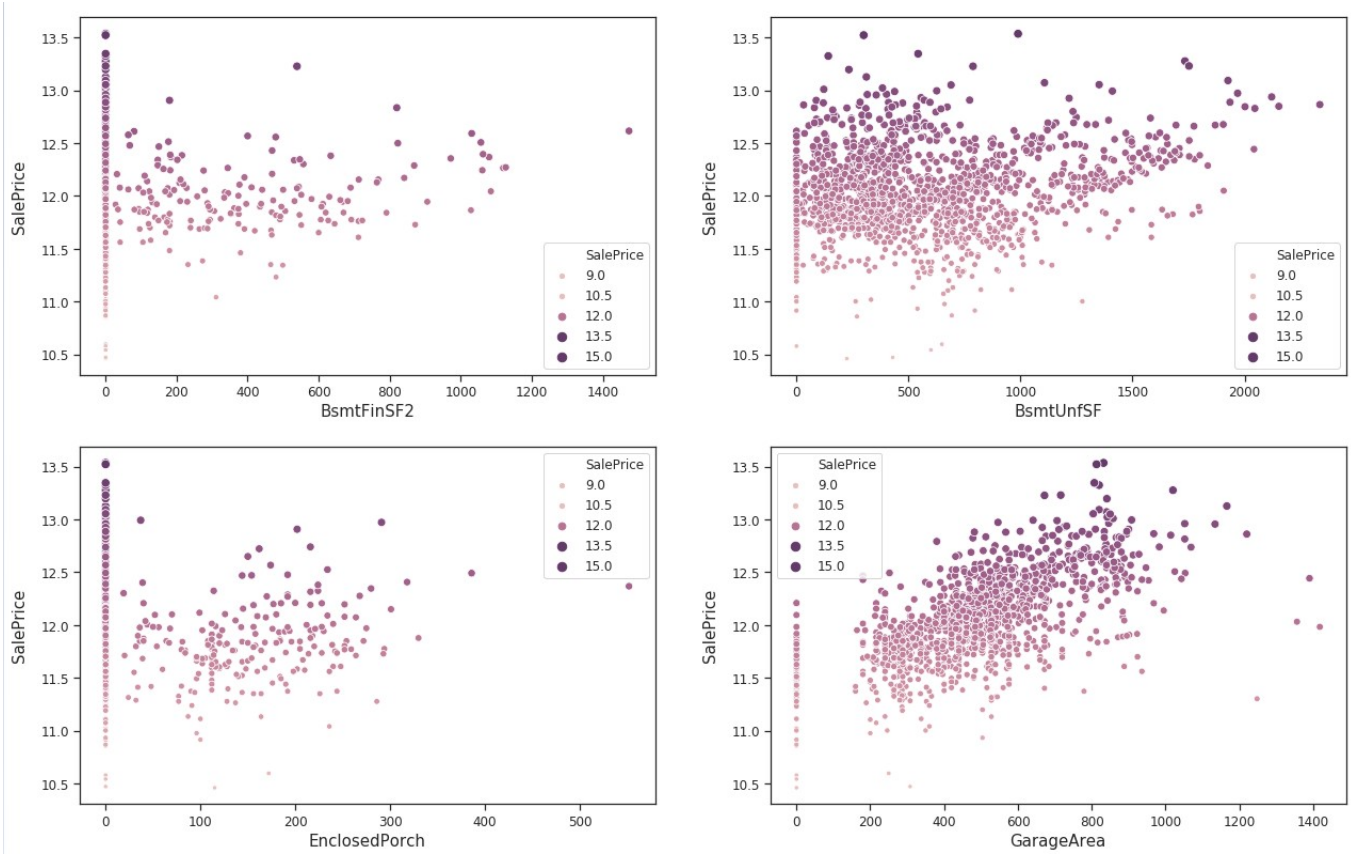
```
plt.show();


num_features = ['1stFlrSF', '2ndFlrSF', '3SsnPorch', 'BsmtFinSF1', 'BsmtFinSF2',
                'BsmtUnfSF', 'EnclosedPorch', 'GarageArea', 'GarageYrBlt', 'GrLivArea',
                'LotArea', 'LotFrontage', 'LowQualFinSF', 'MasVnrArea', 'MiscVal',
                'OpenPorchSF', 'PoolArea', 'ScreenPorch', 'TotalBsmtSF', 'WoodDeckSF',
                'YearBuilt', 'YearRemodAdd']

fig, axs = plt.subplots(ncols=2, nrows=11, figsize=(12, 80))
plt.subplots_adjust(right=1.5)
cmap = sns.cubehelix_palette(dark=0.3, light=0.8, as_cmap=True)

for i, feature in enumerate(num_features, 1):
    plt.subplot(11, 2, i)
    sns.scatterplot(x=feature, y='SalePrice', hue='SalePrice', size='SalePrice', palett
e=cmap, data=train_df)

    plt.xlabel('{}'.format(feature), size=15)
    plt.ylabel('SalePrice', size=15, labelpad=12.5)

    for j in range(2):
        plt.tick_params(axis='x', labelsize=12)
        plt.tick_params(axis='y', labelsize=12)

    plt.legend(loc='best', prop={'size': 12})

plt.show()




cat_features = ['Alley', 'BedroomAbvGr', 'BldgType', 'BsmtCond', 'BsmtExposure',
                'BsmtFinType1', 'BsmtFinType2', 'BsmtFullBath', 'BsmtHalfBath', 'BsmtQu
al',
                'CentralAir', 'Condition1', 'Condition2', 'Electrical', 'ExterCond',
                'ExterQual', 'Exterior1st', 'Exterior2nd', 'Fence', 'FireplaceQu',
                'Fireplaces', 'Foundation', 'FullBath', 'Functional', 'GarageCars',
                'GarageCond', 'GarageFinish', 'GarageQual', 'GarageType', 'HalfBath',
                'Heating', 'HeatingQC', 'KitchenAbvGr', 'KitchenQual', 'LandContour',
                'LandSlope', 'LotConfig', 'LotShape', 'MSSubClass', 'MSZoning',
                'MasVnrType', 'MiscFeature', 'MoSold', 'Neighborhood', 'OverallCond',
                'OverallQual', 'PavedDrive', 'PoolQC', 'RoofMatl', 'RoofStyle',
                'SaleCondition', 'SaleType', 'Street', 'TotRmsAbvGrd', 'Utilities', 'Yr
Sold']

fig, axs = plt.subplots(ncols=2, nrows=28, figsize=(18, 120))
plt.subplots_adjust(right=1.5, top=1.5)

for i, feature in enumerate(cat_features, 1):
    plt.subplot(28, 2, i)
    sns.swarmplot(x=feature, y='SalePrice', data=train_df, palette='Set3')

    plt.xlabel('{}'.format(feature), size=25)
    plt.ylabel('SalePrice', size=25, labelpad=15)

    for j in range(2):
        if train_df[feature].value_counts().shape[0] > 10:
            plt.tick_params(axis='x', labelsize=7)
        else:
            plt.tick_params(axis='x', labelsize=20)
```

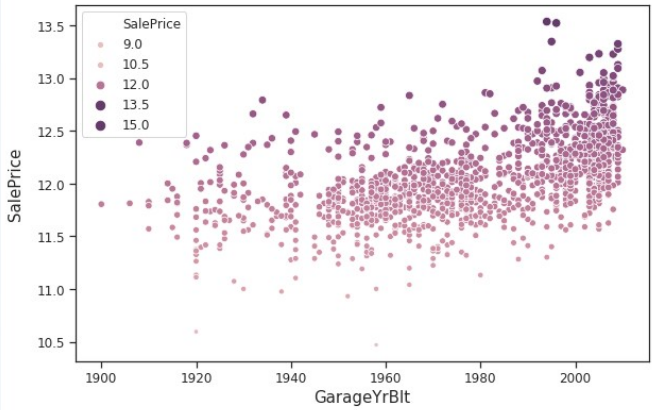
```
        plt.tick_params(axis='y', labelsize=20)

plt.show()
```

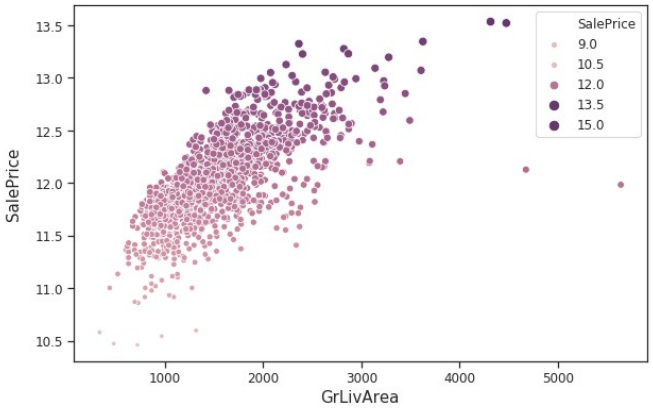


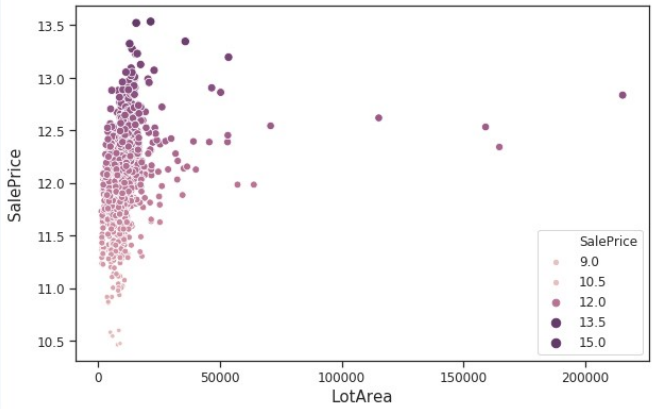

GrLivArea
1000 2000 3000 4000 5000
LotArea
0 50000 100000 150000 200000
BsmtUnfSF
0 500 1000 1500 2000
1stFlrSF
1000 2000 3000 4000
TotalBsmtSF
0 1000 2000 3000 4000 5000 6000
GarageArea
0 200 400 600 800 1000 1200 1400
BsmtFinSF1
0 1000 2000 3000 4000 5000
LotFrontage
50 100 150 200 250 300
YearBuilt
1880 1900 1920 1940 1960 1980 2000
GarageYrBlt
1900 1920 1940 1960 1980 2000
OpenPorchSF
0 100 200 300 400 500
YearRemodAdd
1950 1960 1970 1980 1990 2000 2010
WoodDeckSF
0 200 400 600 800
MoSold
2 4 6 8 10 12
2ndFlrSF
0 500 1000 1500 2000
OverallCond
1 2 3 4 5 6 7 8 9
YrSold
2006.0 2006.5 2007.0 2007.5 2008.0 2008.5 2009.0 2009.5 2010.0
OverallQual
2 4 6 8 10

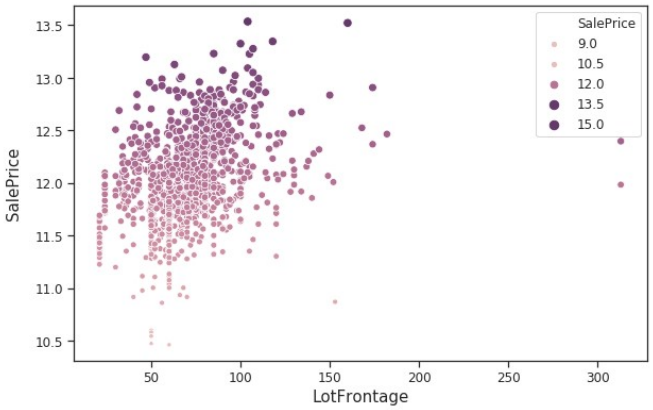

SalePrice
1stFlrSF
2ndFlrSF
3SsnPorch
BsmtFinSF1
9.0
10.5
12.0
13.5
15.0

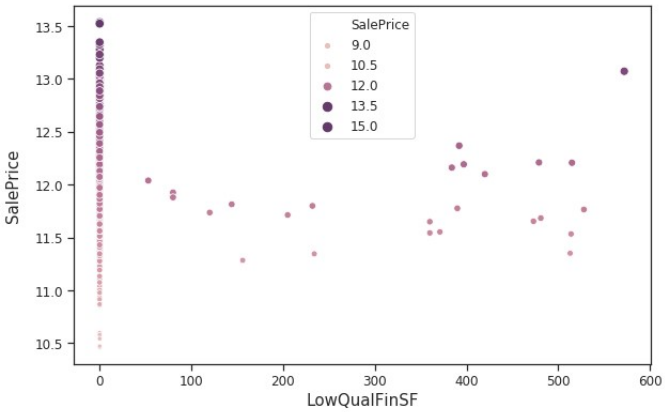

SalePrice
BsmtFinSF2
BsmtUnfSF
EnclosedPorch
GarageArea
9.0
10.5
12.0
13.5
15.0

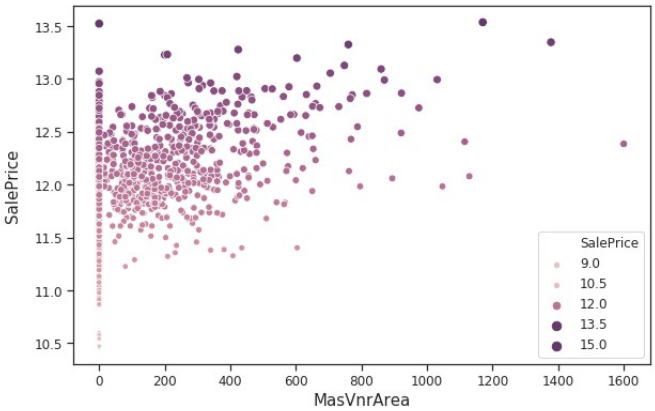

SalePrice
9.0
10.5
12.0
13.5
15.0
SalePrice
GarageYrBlt

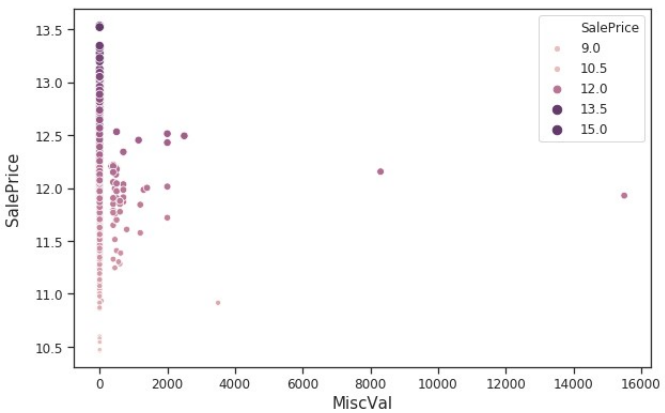

SalePrice
9.0
10.5
12.0
13.5
15.0
SalePrice
GrLivArea

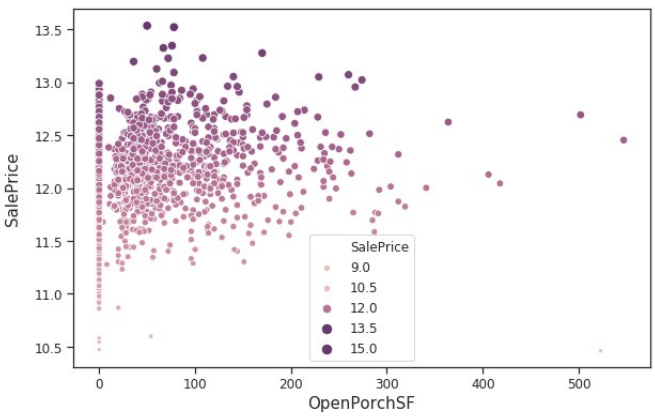

SalePrice
9.0
10.5
12.0
13.5
15.0
SalePrice
LotArea

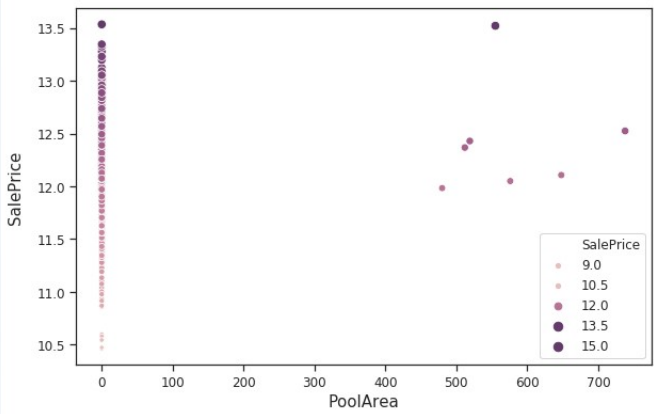

SalePrice
9.0
10.5
12.0
13.5
15.0
SalePrice
LotFrontage

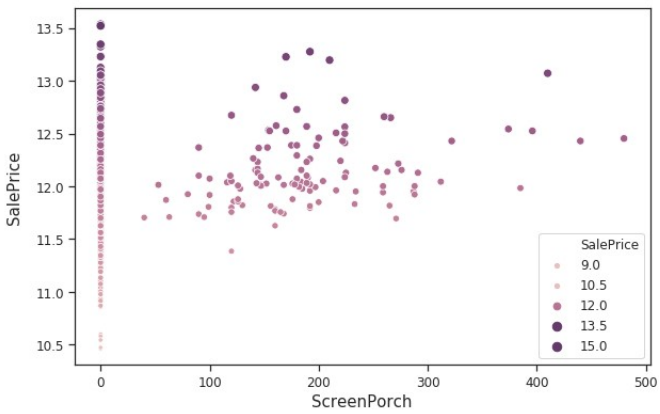

SalePrice
9.0
10.5
12.0
13.5
15.0
SalePrice
LowQualFinSF

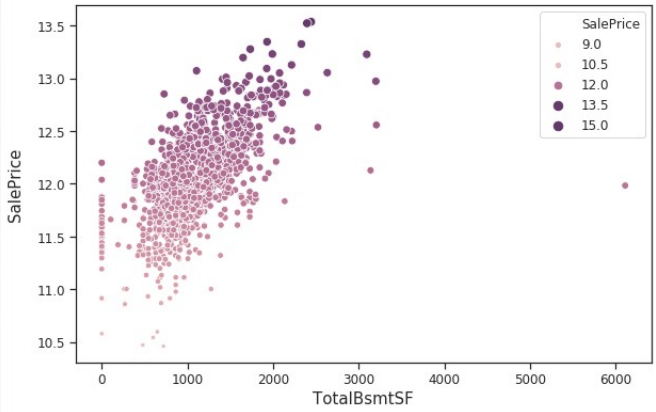

SalePrice
9.0
10.5
12.0
13.5
15.0
SalePrice
MasVnrArea

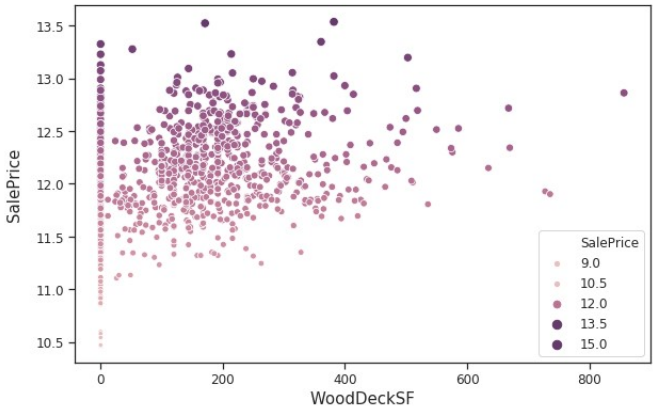

SalePrice
9.0
10.5
12.0
13.5
15.0
SalePrice
MiscVal

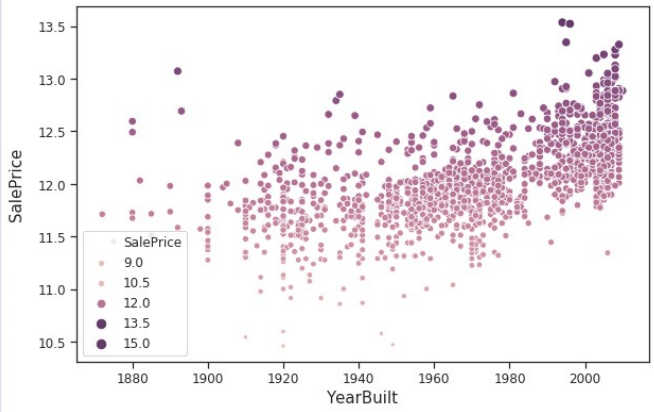

SalePrice
9.0
10.5
12.0
13.5
15.0
SalePrice
OpenPorchSF

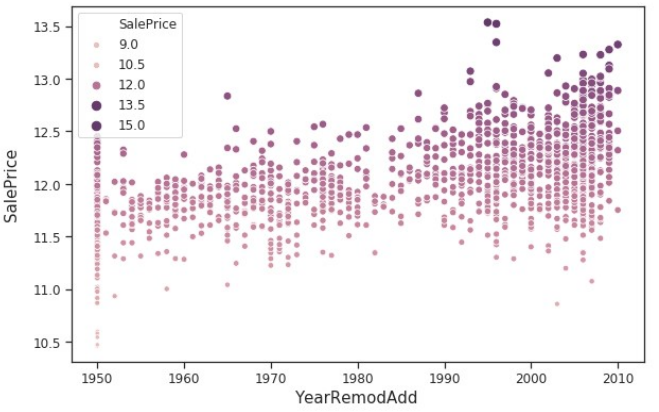

SalePrice
PoolArea
SalePrice
9.0
10.5
12.0
13.5
15.0

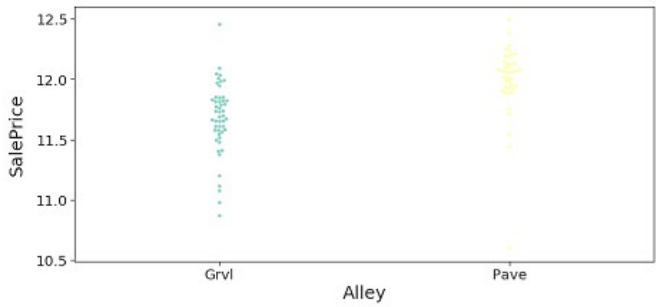

SalePrice
ScreenPorch
SalePrice
9.0
10.5
12.0
13.5
15.0

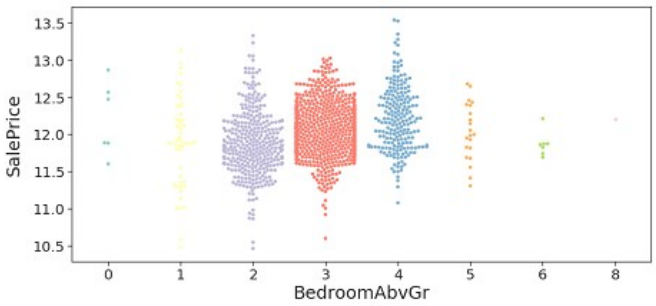

SalePrice
TotalBsmtSF
SalePrice
9.0
10.5
12.0
13.5
15.0

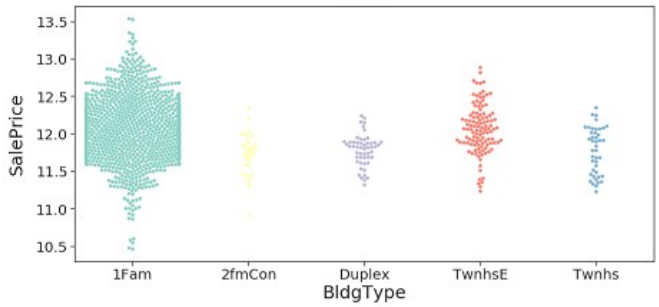

SalePrice
WoodDeckSF
SalePrice
9.0
10.5
12.0
13.5
15.0

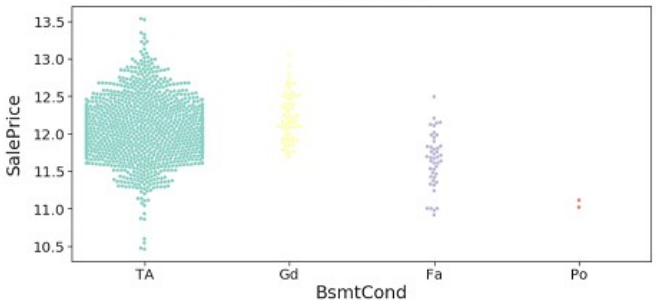

SalePrice
YearBuilt
SalePrice
9.0
10.5
12.0
13.5
15.0

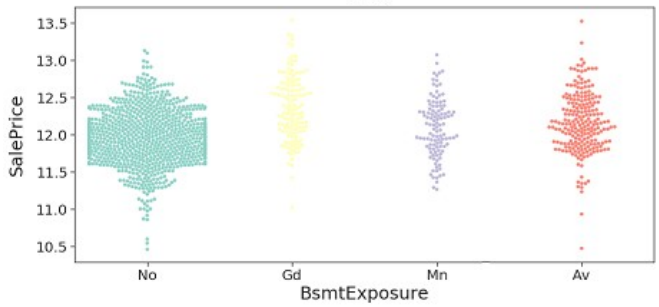

SalePrice
YearRemodAdd
SalePrice
9.0
10.5
12.0
13.5
15.0

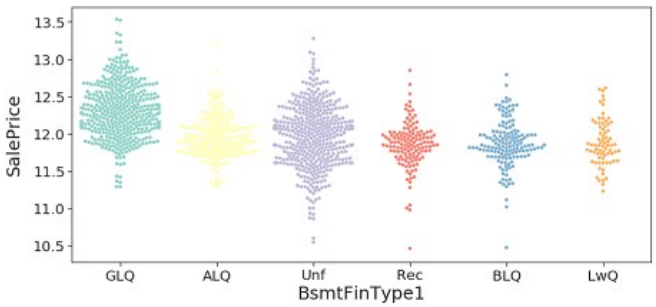

SalePrice
12.5
12.0
11.5
11.0
10.5
Grvl
Pave
Alley

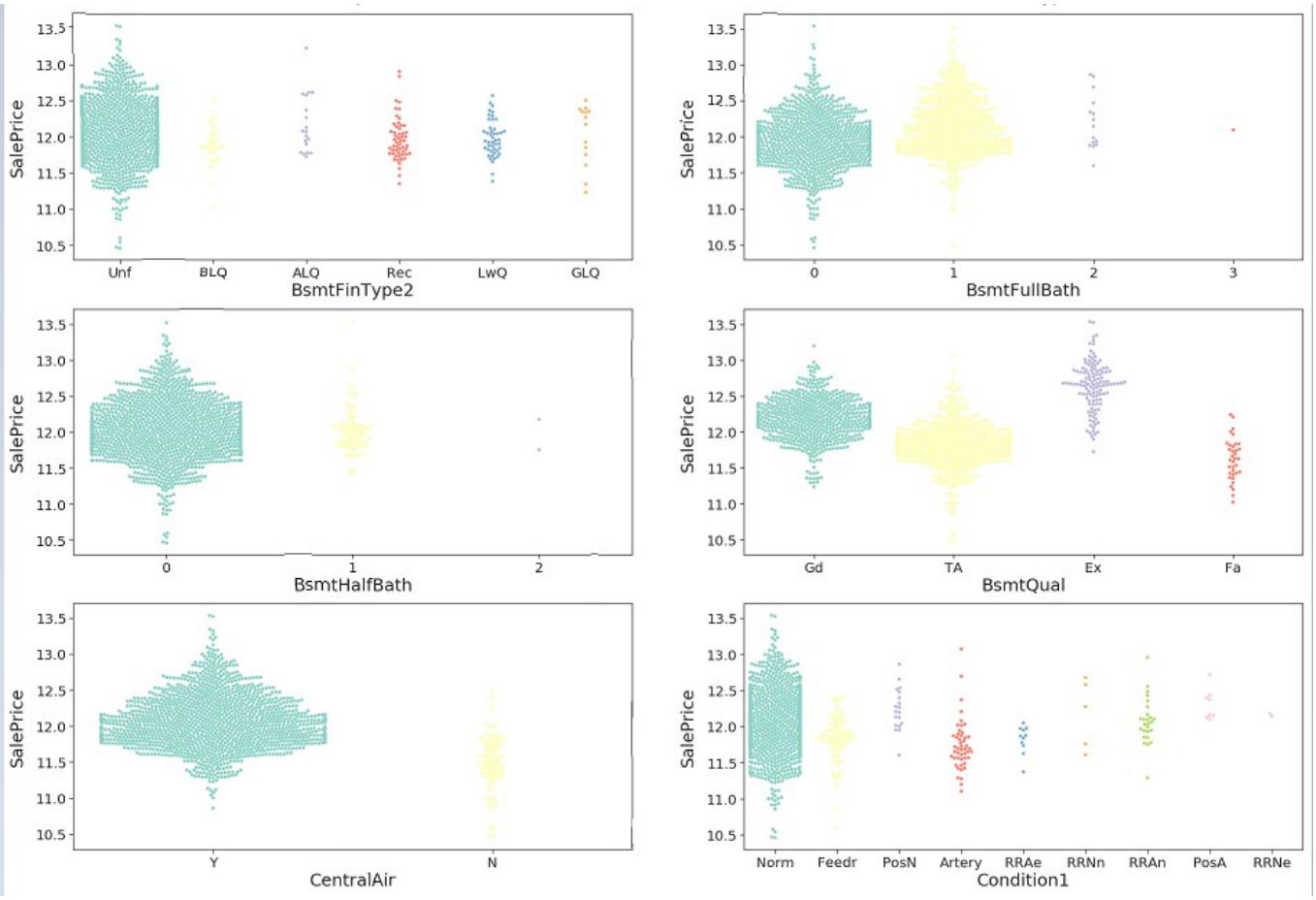

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
0
1
2
3
4
5
6
8
BedroomAbvGr

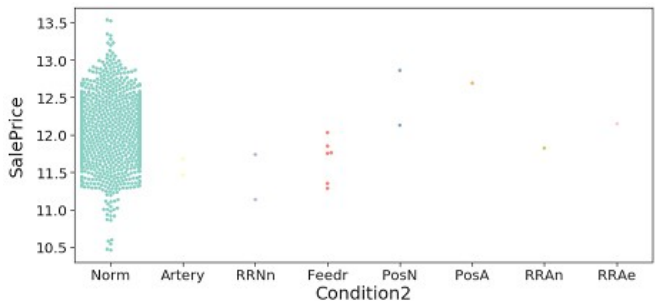

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
1Fam
2fmCon
Duplex
TwnhsE
Twnhs
BldgType

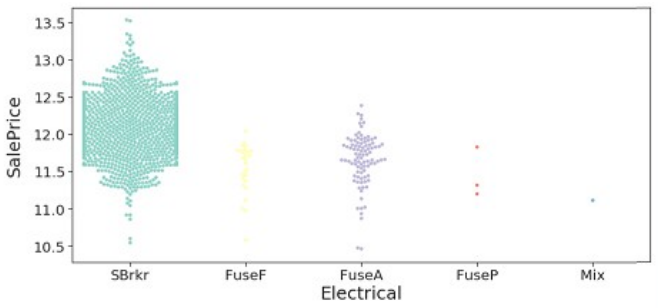

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
TA
Gd
Fa
Po
BsmtCond

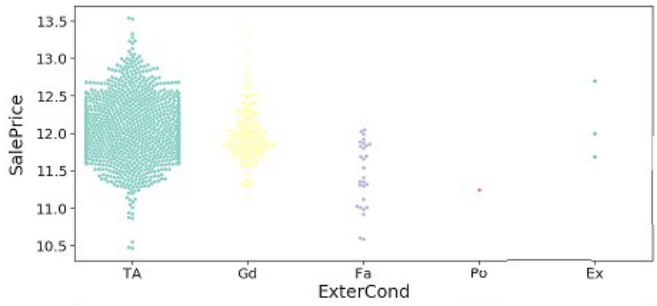

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
No
Gd
Mn
Av
BsmtExposure

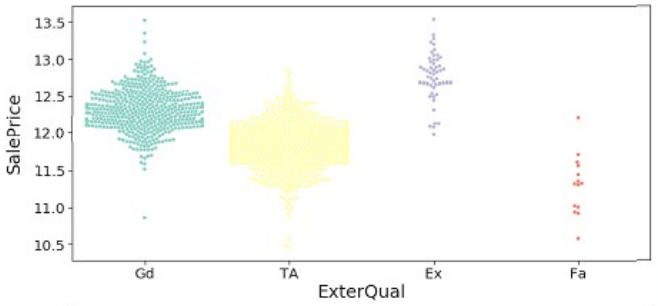

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
GLQ
ALQ
Unf
Rec
BLQ
LwQ
BsmtFinType1

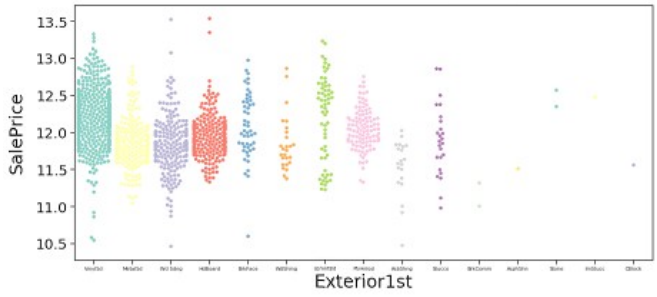

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Unf
BLQ
ALQ
Rec
LwQ
GLQ
BsmtFinType2
0
1
2
3
BsmtFullBath
BsmtHalfBath
Gd
TA
Ex
Fa
BsmtQual
Y
N
CentralAir
Norm
Feedr
PosN
Artery
RRAe
RRNn
RRAn
PosA
RRNe
Condition1

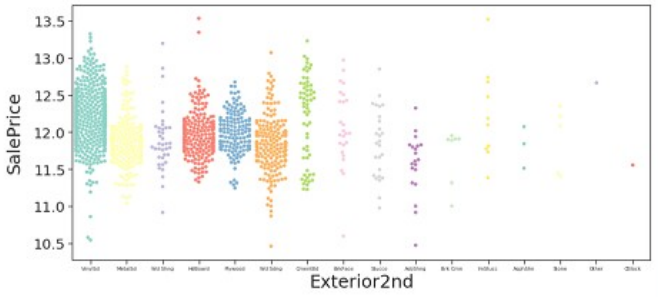

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Norm
Artery
RRNn
Feedr
PosN
PosA
RRAn
RRAe
Condition2

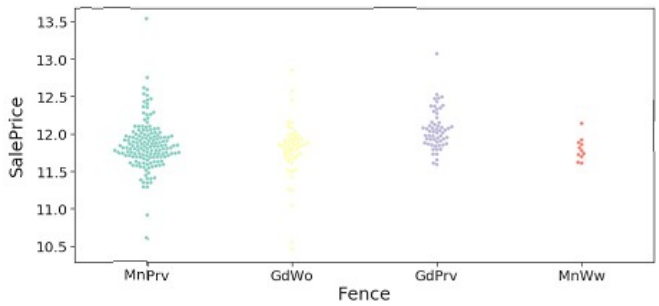

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
SBrkr
FuseF
FuseA
FuseP
Mix
Electrical

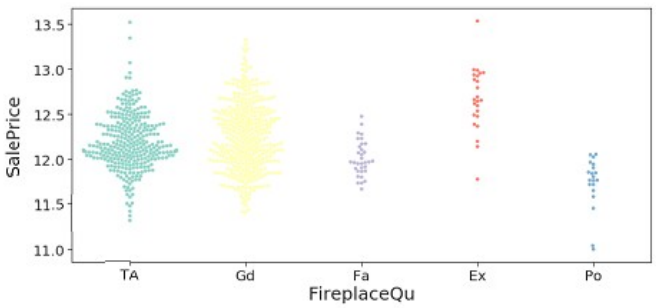

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
TA
Gd
Fa
Po
Ex
ExterCond

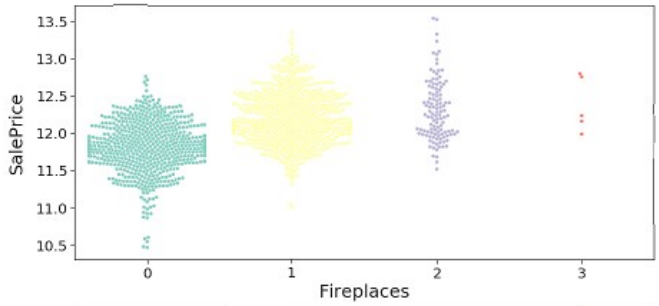

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Gd
TA
Ex
Fa
ExterQual

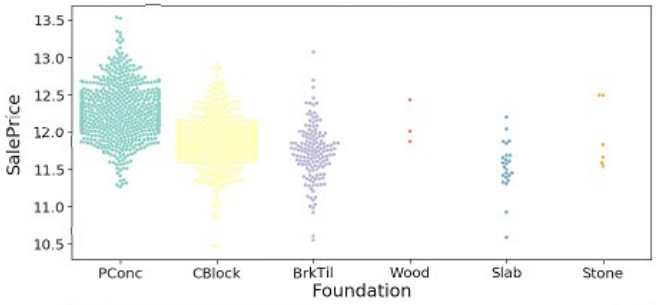

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Exterior1st

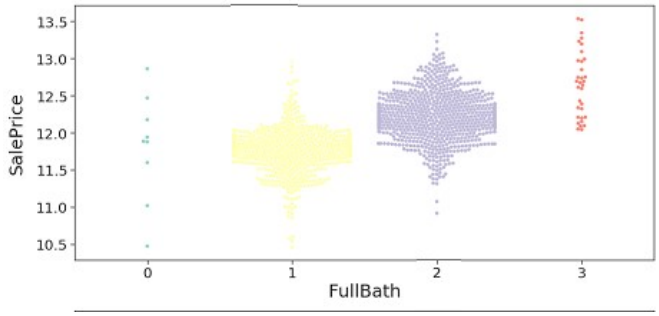

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Exterior2nd

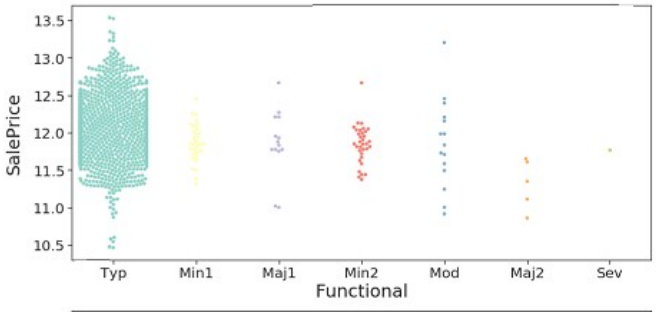

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
MnPrv
GdWo
GdPrv
MnWw
Fence

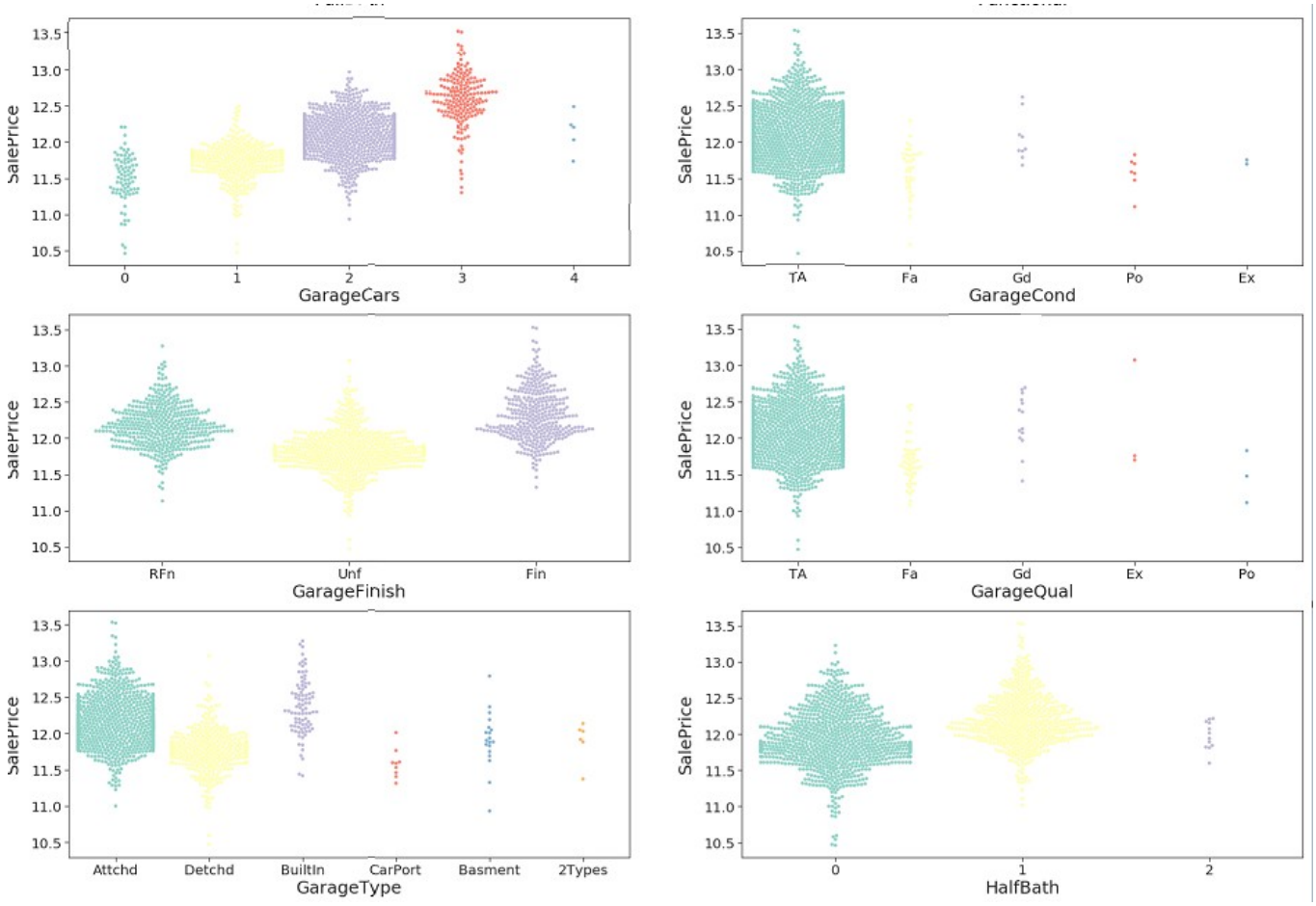

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
TA
Gd
Fa
Ex
Po
FireplaceQu

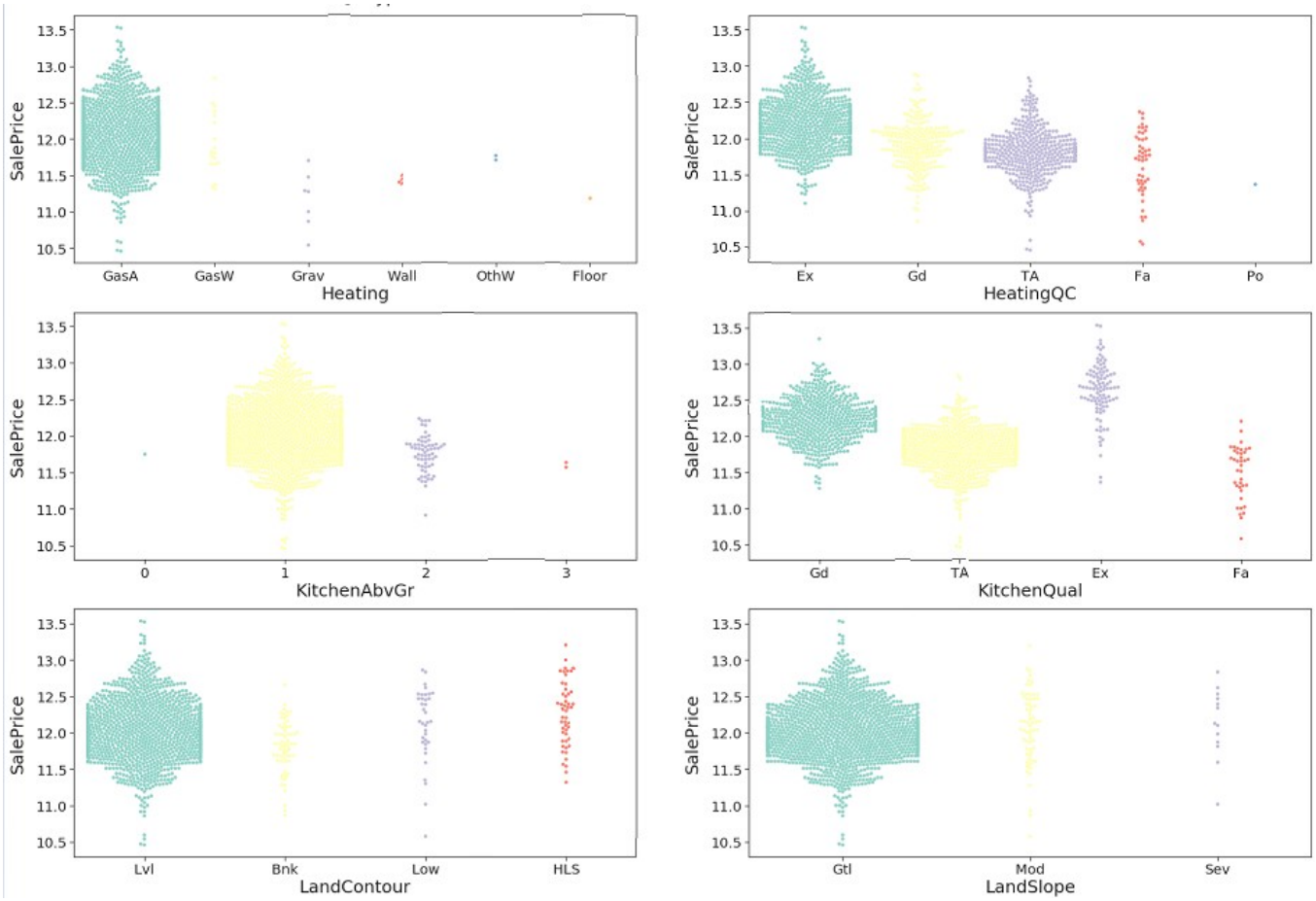

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
0
1
2
3
Fireplaces

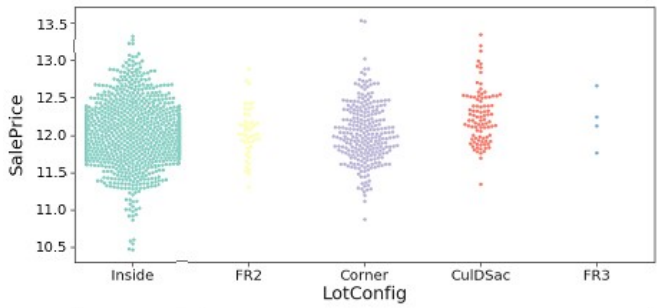

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
PConc
CBlock
BrkTil
Wood
Slab
Stone
Foundation

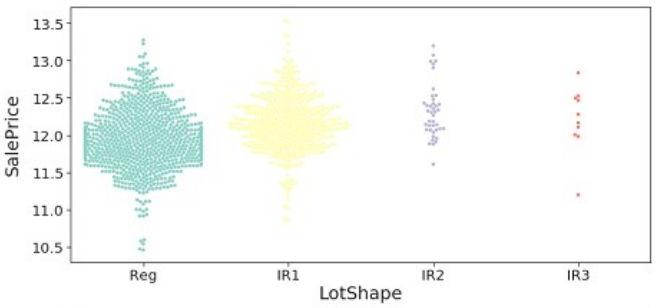

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
0
1
2
3
FullBath

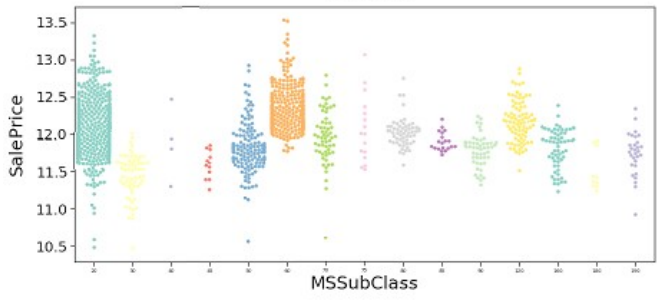

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Typ
Min1
Maj1
Min2
Mod
Maj2
Sev
Functional

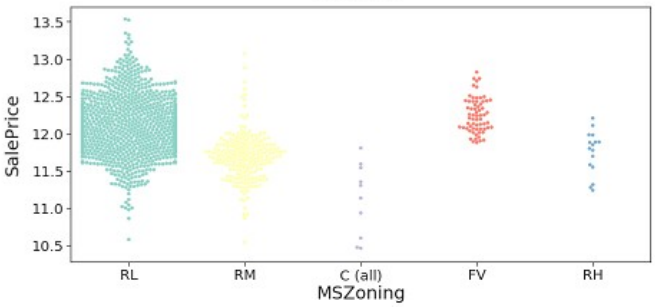

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
0
1
2
3
4
GarageCars
TA
Fa
Gd
Po
Ex
GarageCond
RFn
Unf
Fin
GarageFinish
TA
Fa
Gd
Ex
Po
GarageQual
Attchd
Detchd
BuiltIn
CarPort
Basment
2Types
GarageType
0
1
2
HalfBath

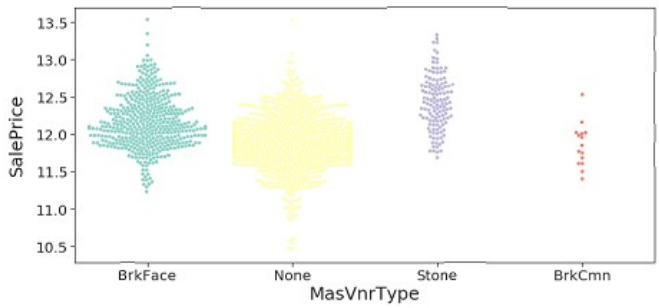

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
GasA
GasW
Grav
Wall
OthW
Floor
Heating
Ex
Gd
TA
Fa
Po
HeatingQC
0
1
2
3
KitchenAbvGr
Gd
TA
Ex
Fa
KitchenQual
Lvl
Bnk
Low
HLS
LandContour
Gtl
Mod
Sev
LandSlope

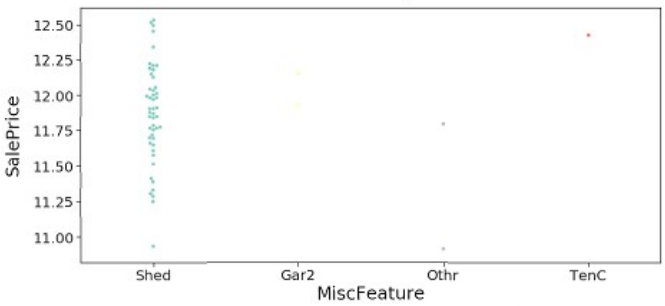

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Inside
FR2
Corner
CulDSac
FR3
LotConfig

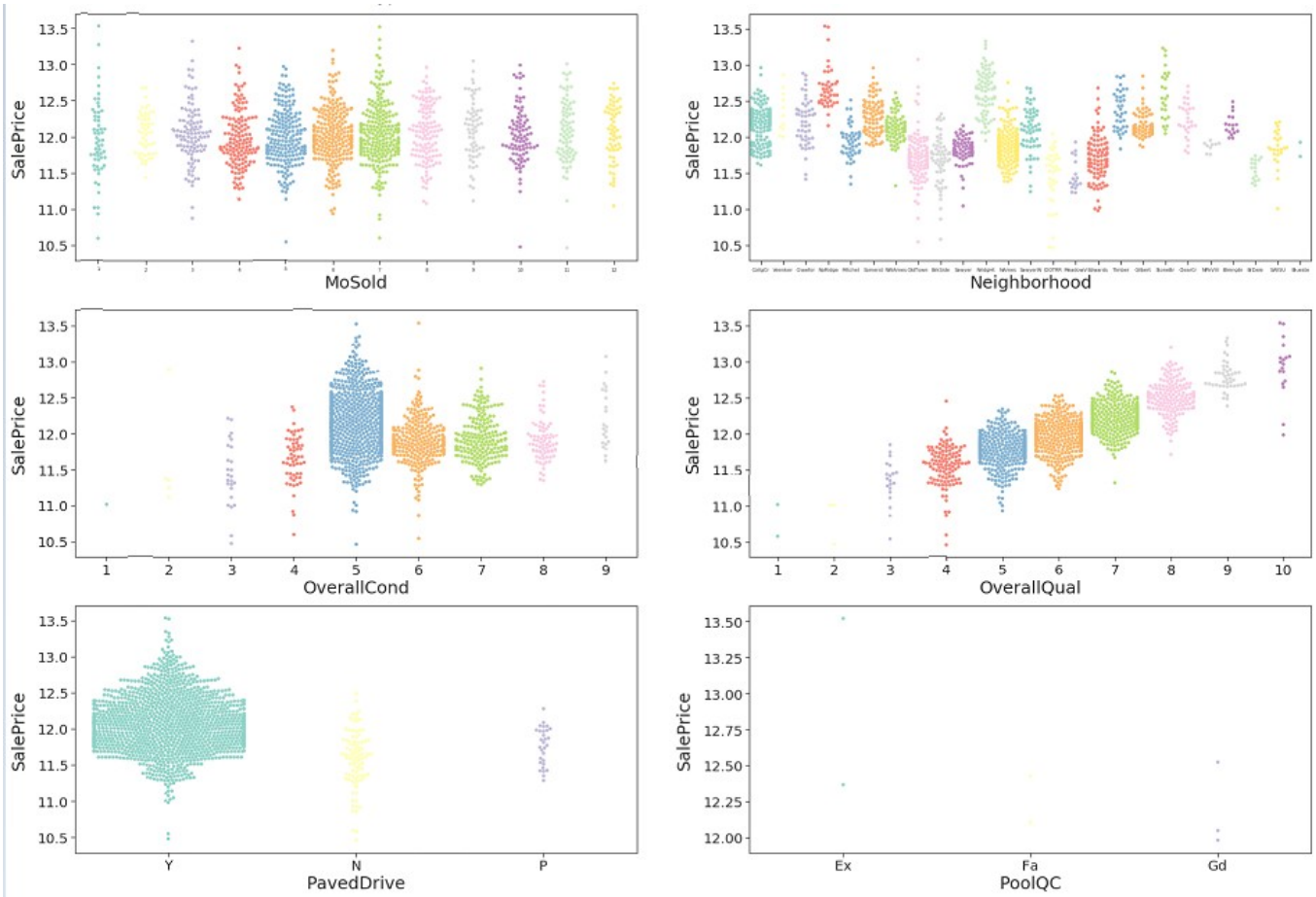

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Reg
IR1
IR2
IR3
LotShape

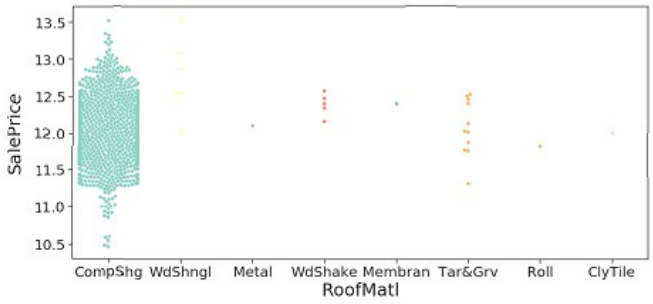

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
MSSubClass

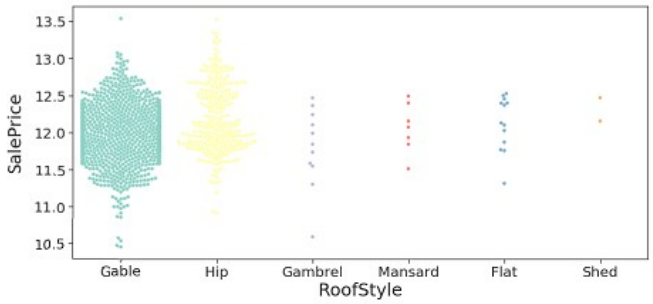

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
RL
RM
C (all)
FV
RH
MSZoning

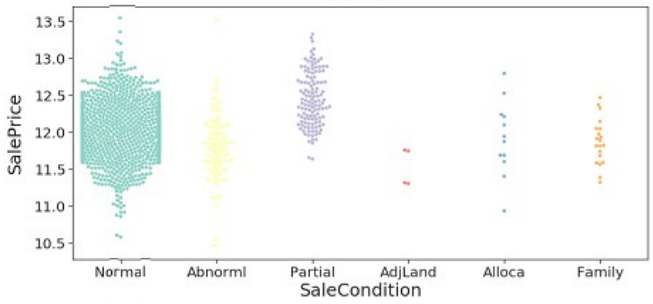

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
BrkFace
None
Stone
BrkCmn
MasVnrType

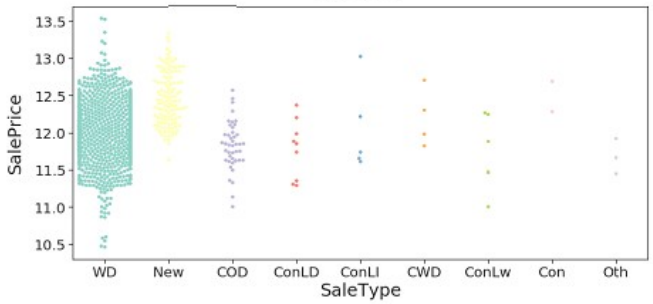

SalePrice
12.50
12.25
12.00
11.75
11.50
11.25
11.00
Shed
Gar2
Othr
TenC
MiscFeature

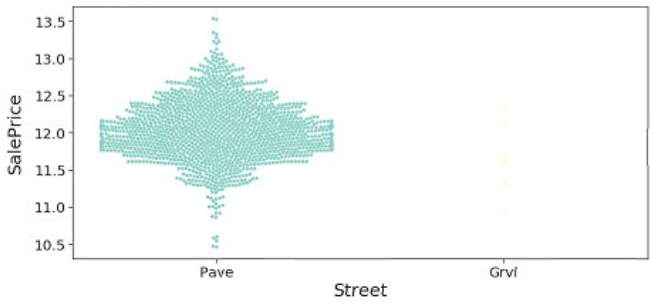

SalePrice
MoSold
Neighborhood
OverallCond
OverallQual
PavedDrive
Y
N
P
PoolQC
Ex
Fa
Gd

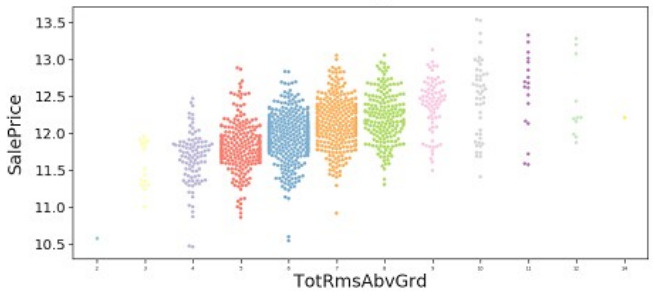

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
CompShg
WdShngl
Metal
WdShake
Membran
Tar&Grv
Roll
ClyTile
RoofMatl

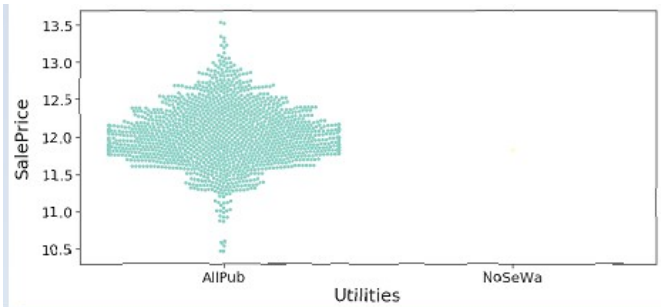

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Gable
Hip
Gambrel
Mansard
Flat
Shed
RoofStyle

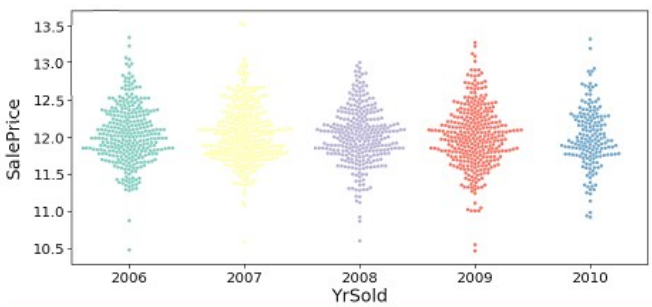

SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Normal
Abnorml
Partial
AdjLand
Alloca
Family
SaleCondition


SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
WD
New
COD
ConLD
ConLI
CWD
ConLw
Con
Oth
SaleType


SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Pave
Grvl
Street


SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
TotRmsAbvGrd


SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
AllPub
NoSeWa
Utilities


SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
2006
2007
2008
2009
2010
YrSold

Cell Report:

1. Seaborn Pairplot was used to show the most effective Features are correlated with the Target Variable.
2. As for the 'TotalBsmtSF' and 'GrLiveArea' Pairplot, we can see the dots drawing a linear line, which almost acts like a border. This makes sense as basement areas are usualy equal or lower than gound living areas of houses
3. 'SalePrice' and 'YearBuilt' Pairplot dot cloud shows some what of an exponential curve, which shows that house prices are increasing faster in recent years
4. Some outliers can be seen from the Pairplot and Boxplots, for example there are a few houses with more than 4000 sq ft living area, these were treated later on
5. From the Numerical Scatterplot: Data points of '2ndFlrSF', '3SsnPorch', 'BsmtFinSF1', 'BsmtFinSF2', 'EnclosedPorch', 'LowQualFinSF', 'MasVnrArea', 'MiscVal', 'OpenPorchSF', 'PoolArea', 'ScreenPorch' and 'WoodDeckSF' features are heavily stacked at 0. Which shows these are sparse attributes and rare for most houses. Sparse features may not be reliable when they are used as continuous features, because they are going to introduce bias to the regression function. 'GarageYrBlt', 'YearBuilt' and 'YearRemodAdd' are ordinal features, but a linear relationship can be seen from their plots.
6. From the Categorical Swarmplot: Categorical features are not strongly correlated with 'SalePrice'(some however have very distinct 'SalePrice' maximums, minimums). There are only 2 categorical features that have significant correlation with 'SalePrice', and they are 'OverallQual' and 'TotRmsAbvGrd'.

2. Data Preprocessing

2.1 Data Anomaly Removal

In [48]:

```
# outlier revoval
train_df.drop(train_df[train_df["GrLivArea"] > 4000].index, inplace=True)
train2.drop(train2[train2["GrLivArea"] > 4000].index, inplace=True)
# The test example with ID 666 has GarageArea, GarageCars, and GarageType
# but none of the other fields, so use the mode and median to fill them in.
test_df.loc[666, "GarageQual"] = "TA"
test_df.loc[666, "GarageCond"] = "TA"
test_df.loc[666, "GarageFinish"] = "Unf"
test_df.loc[666, "GarageYrBlt"] = "1980"

# The test example 1116 only has GarageType but no other information. We'll
# assume it does not have a garage.
test_df.loc[1116, "GarageType"] = np.nan

# For imputing missing values: fill in missing LotFrontage values by the median
# LotFrontage of the neighborhood.
lot_frontage_by_neighborhood = train_df["LotFrontage"].groupby(train_df["Neighborhood"
])


display(test_df.loc[test_df.GarageYrBlt==2207, ['GarageYrBlt', 'YearBuilt']])
test_df.loc[test_df.GarageYrBlt==2207.0, 'GarageYrBlt'] = 2007.0

display(test_df.loc[test_df.GarageYrBlt==2207, ['GarageYrBlt', 'YearBuilt']])
```

	GarageYrBlt	YearBuilt
1132	2207	2006

	GarageYrBlt	YearBuilt

Cell Report:

1. # There are a few houses with more than 4000 sq ft living area that are outliers, so those were dropped from the training data. (There is also one in the test set but those cannot be dropped as Kaggle Score function needs all test records to be present.)
2. The test example with ID 666 has GarageArea, GarageCars, and GarageType but none of the other fields, so use the mode and median to fill them in.
3. Reading blog posts about this dataset showed that the test set had an anomaly (GarageYrBlt was 2207 for one record, row no.1132).

2.2 Feature Engineering

In [49]:

```
# Used to convert categorical features into ordinal numbers.
# (There's probably an easier way to do this, but it works.)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

def factorize(df, factor_df, column, fill_na=None):
    factor_df[column] = df[column]
    if fill_na is not None:
        factor_df[column].fillna(fill_na, inplace=True)
    le.fit(factor_df[column].unique())
    factor_df[column] = le.transform(factor_df[column])
    return factor_df

# Combine all the (numerical) features into one big DataFrame. We don't add
# the one-hot encoded variables here yet, that happens later on.
def munge(df):
    all_df = pd.DataFrame(index = df.index)

    all_df["LotFrontage"] = df["LotFrontage"]
    for key, group in lot_frontage_by_neighborhood:
        idx = (df["Neighborhood"] == key) & (df["LotFrontage"].isnull())
        all_df.loc[idx, "LotFrontage"] = group.median()

    all_df["LotArea"] = df["LotArea"]

    all_df["MasVnrArea"] = df["MasVnrArea"]
    all_df["MasVnrArea"].fillna(0, inplace=True)

    all_df["BsmtFinSF1"] = df["BsmtFinSF1"]
    all_df["BsmtFinSF1"].fillna(0, inplace=True)

    all_df["BsmtFinSF2"] = df["BsmtFinSF2"]
    all_df["BsmtFinSF2"].fillna(0, inplace=True)

    all_df["BsmtUnfSF"] = df["BsmtUnfSF"]
    all_df["BsmtUnfSF"].fillna(0, inplace=True)

    all_df["TotalBsmtSF"] = df["TotalBsmtSF"]
    all_df["TotalBsmtSF"].fillna(0, inplace=True)

    all_df["1stFlrSF"] = df["1stFlrSF"]
    all_df["2ndFlrSF"] = df["2ndFlrSF"]
    all_df["GrLivArea"] = df["GrLivArea"]

    all_df["GarageArea"] = df["GarageArea"]
    all_df["GarageArea"].fillna(0, inplace=True)

    all_df["WoodDeckSF"] = df["WoodDeckSF"]
    all_df["OpenPorchSF"] = df["OpenPorchSF"]
    all_df["EnclosedPorch"] = df["EnclosedPorch"]
    all_df["3SsnPorch"] = df["3SsnPorch"]
    all_df["ScreenPorch"] = df["ScreenPorch"]

    all_df["BsmtFullBath"] = df["BsmtFullBath"]
    all_df["BsmtFullBath"].fillna(0, inplace=True)

    all_df["BsmtHalfBath"] = df["BsmtHalfBath"]
    all_df["BsmtHalfBath"].fillna(0, inplace=True)
```

```
    all_df["FullBath"] = df["FullBath"]
    all_df["HalfBath"] = df["HalfBath"]
    all_df["BedroomAbvGr"] = df["BedroomAbvGr"]
    all_df["KitchenAbvGr"] = df["KitchenAbvGr"]
    all_df["TotRmsAbvGrd"] = df["TotRmsAbvGrd"]
    all_df["Fireplaces"] = df["Fireplaces"]

    all_df["GarageCars"] = df["GarageCars"]
    all_df["GarageCars"].fillna(0, inplace=True)

    all_df["CentralAir"] = (df["CentralAir"] == "Y") * 1.0
    all_df["OverallQual"] = df["OverallQual"]
    all_df["OverallCond"] = df["OverallCond"]

    # Quality measurements are stored as text but we can convert them to
    # numbers where a higher number means higher quality.

    qual_dict = {None: 0, "Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5}
    all_df["ExterQual"] = df["ExterQual"].map(qual_dict).astype(int)
    all_df["ExterCond"] = df["ExterCond"].map(qual_dict).astype(int)
    all_df["BsmtQual"] = df["BsmtQual"].map(qual_dict).astype(int)
    all_df["BsmtCond"] = df["BsmtCond"].map(qual_dict).astype(int)
    all_df["HeatingQC"] = df["HeatingQC"].map(qual_dict).astype(int)
    all_df["KitchenQual"] = df["KitchenQual"].map(qual_dict).astype(int)
    all_df["FireplaceQu"] = df["FireplaceQu"].map(qual_dict).astype(int)
    all_df["GarageQual"] = df["GarageQual"].map(qual_dict).astype(int)
    all_df["GarageCond"] = df["GarageCond"].map(qual_dict).astype(int)

    all_df["BsmtExposure"] = df["BsmtExposure"].map(
        {None: 0, "No": 1, "Mn": 2, "Av": 3, "Gd": 4}).astype(int)

    bsmt_fin_dict = {None: 0, "Unf": 1, "LwQ": 2, "Rec": 3, "BLQ": 4, "ALQ": 5, "GLQ":
6}
    all_df["BsmtFinType1"] = df["BsmtFinType1"].map(bsmt_fin_dict).astype(int)
    all_df["BsmtFinType2"] = df["BsmtFinType2"].map(bsmt_fin_dict).astype(int)

    all_df["Functional"] = df["Functional"].map(
        {None: 0, "Sal": 1, "Sev": 2, "Maj2": 3, "Maj1": 4,
         "Mod": 5, "Min2": 6, "Min1": 7, "Typ": 8}).astype(int)

    all_df["GarageFinish"] = df["GarageFinish"].map(
        {None: 0, "Unf": 1, "RFn": 2, "Fin": 3}).astype(int)

    all_df["Fence"] = df["Fence"].map(
        {None: 0, "MnWw": 1, "GdWo": 2, "MnPrv": 3, "GdPrv": 4}).astype(int)

    all_df["YearBuilt"] = df["YearBuilt"]
    all_df["YearRemodAdd"] = df["YearRemodAdd"]

    all_df["GarageYrBlt"] = df["GarageYrBlt"]
    all_df["GarageYrBlt"].fillna(0.0, inplace=True)

    all_df["MoSold"] = df["MoSold"]
    all_df["YrSold"] = df["YrSold"]

    all_df["LowQualFinSF"] = df["LowQualFinSF"]
    all_df["MiscVal"] = df["MiscVal"]

    all_df["PoolQC"] = df["PoolQC"].map(qual_dict).astype(int)

    all_df["PoolArea"] = df["PoolArea"]
```

```
all_df["PoolArea"].fillna(0, inplace=True)

# Add categorical features as numbers too. It seems to help a bit.
all_df = factorize(df, all_df, "MSSubClass")
all_df = factorize(df, all_df, "MSZoning", "RL")
all_df = factorize(df, all_df, "LotConfig")
all_df = factorize(df, all_df, "Neighborhood")
all_df = factorize(df, all_df, "Condition1")
all_df = factorize(df, all_df, "BldgType")
all_df = factorize(df, all_df, "HouseStyle")
all_df = factorize(df, all_df, "RoofStyle")
all_df = factorize(df, all_df, "Exterior1st", "Other")
all_df = factorize(df, all_df, "Exterior2nd", "Other")
all_df = factorize(df, all_df, "MasVnrType", "None")
all_df = factorize(df, all_df, "Foundation")
all_df = factorize(df, all_df, "SaleType", "Oth")
all_df = factorize(df, all_df, "SaleCondition")

# IR2 and IR3 don't appear that often, so just make a distinction
# between regular and irregular.
all_df["IsRegularLotShape"] = (df["LotShape"] == "Reg") * 1

# Most properties are level; bin the other possibilities together
# as "not level".
all_df["IsLandLevel"] = (df["LandContour"] == "Lvl") * 1

# Most land slopes are gentle; treat the others as "not gentle".
all_df["IsLandSlopeGentle"] = (df["LandSlope"] == "Gtl") * 1

# Most properties use standard circuit breakers.
all_df["IsElectricalSBrkr"] = (df["Electrical"] == "SBrkr") * 1

# About 2/3rd have an attached garage.
all_df["IsGarageDetached"] = (df["GarageType"] == "Detchd") * 1

# Most have a paved drive. Treat dirt/gravel and partial pavement
# as "not paved".
all_df["IsPavedDrive"] = (df["PavedDrive"] == "Y") * 1

# The only interesting "misc. feature" is the presence of a shed.
all_df["HasShed"] = (df["MiscFeature"] == "Shed") * 1.

# If YearRemodAdd != YearBuilt, then a remodeling took place at some point.
all_df["Remodeled"] = (all_df["YearRemodAdd"] != all_df["YearBuilt"]) * 1

# Did a remodeling happen in the year the house was sold?
all_df["RecentRemodel"] = (all_df["YearRemodAdd"] == all_df["YrSold"]) * 1

# Was this house sold in the year it was built?
all_df["VeryNewHouse"] = (all_df["YearBuilt"] == all_df["YrSold"]) * 1

all_df["Has2ndFloor"] = (all_df["2ndFlrSF"] == 0) * 1
all_df["HasMasVnr"] = (all_df["MasVnrArea"] == 0) * 1
all_df["HasWoodDeck"] = (all_df["WoodDeckSF"] == 0) * 1
all_df["HasOpenPorch"] = (all_df["OpenPorchSF"] == 0) * 1
all_df["HasEnclosedPorch"] = (all_df["EnclosedPorch"] == 0) * 1
all_df["Has3SsnPorch"] = (all_df["3SsnPorch"] == 0) * 1
all_df["HasScreenPorch"] = (all_df["ScreenPorch"] == 0) * 1
```

```
    # Months with the largest number of deals may be significant.
    all_df["HighSeason"] = df["MoSold"].replace(
        {1: 0, 2: 0, 3: 0, 4: 1, 5: 1, 6: 1, 7: 1, 8: 0, 9: 0, 10: 0, 11: 0, 12: 0})

    all_df["NewerDwelling"] = df["MSSubClass"].replace(
        {20: 1, 30: 0, 40: 0, 45: 0,50: 0, 60: 1, 70: 0, 75: 0, 80: 0, 85: 0,
         90: 0, 120: 1, 150: 0, 160: 0, 180: 0, 190: 0})

    all_df.loc[df.Neighborhood == 'NridgHt', "Neighborhood_Good"] = 1
    all_df.loc[df.Neighborhood == 'Crawfor', "Neighborhood_Good"] = 1
    all_df.loc[df.Neighborhood == 'StoneBr', "Neighborhood_Good"] = 1
    all_df.loc[df.Neighborhood == 'Somerst', "Neighborhood_Good"] = 1
    all_df.loc[df.Neighborhood == 'NoRidge', "Neighborhood_Good"] = 1
    all_df["Neighborhood_Good"].fillna(0, inplace=True)

    all_df["SaleCondition_PriceDown"] = df.SaleCondition.replace(
        {'Abnorml': 1, 'Alloca': 1, 'AdjLand': 1, 'Family': 1, 'Normal': 0, 'Partial':
0})

    # House completed before sale or not
    all_df["BoughtOffPlan"] = df.SaleCondition.replace(
        {"Abnorml" : 0, "Alloca" : 0, "AdjLand" : 0, "Family" : 0, "Normal" : 0, "Parti
al" : 1})

    all_df["BadHeating"] = df.HeatingQC.replace(
        {'Ex': 0, 'Gd': 0, 'TA': 0, 'Fa': 1, 'Po': 1})

    area_cols = ['LotFrontage', 'LotArea', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'B
smtUnfSF',
                 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'GarageArea', 'Woo
dDeckSF',
                 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'LowQualFi
nSF', 'PoolArea' ]
    all_df["TotalArea"] = all_df[area_cols].sum(axis=1)

    all_df["TotalArea1st2nd"] = all_df["1stFlrSF"] + all_df["2ndFlrSF"]

    all_df["Age"] = 2010 - all_df["YearBuilt"]
    all_df["TimeSinceSold"] = 2010 - all_df["YrSold"]

    all_df["SeasonSold"] = all_df["MoSold"].map({12:0, 1:0, 2:0, 3:1, 4:1, 5:1,
                                                 6:2, 7:2, 8:2, 9:3, 10:3, 11:3}).asty
pe(int)

    all_df["YearsSinceRemodel"] = all_df["YrSold"] - all_df["YearRemodAdd"]

    # Simplifications of existing features into bad/average/good.
    all_df["SimplOverallQual"] = all_df.OverallQual.replace(
        {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2, 6 : 2, 7 : 3, 8 : 3, 9 : 3, 10 : 3})
    all_df["SimplOverallCond"] = all_df.OverallCond.replace(
        {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2, 6 : 2, 7 : 3, 8 : 3, 9 : 3, 10 : 3})
    all_df["SimplPoolQC"] = all_df.PoolQC.replace(
        {1 : 1, 2 : 1, 3 : 2, 4 : 2})
    all_df["SimplGarageCond"] = all_df.GarageCond.replace(
        {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
    all_df["SimplGarageQual"] = all_df.GarageQual.replace(
        {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
    all_df["SimplFireplaceQu"] = all_df.FireplaceQu.replace(
        {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
#     all_df["SimplFireplaceQu"] = all_df.FireplaceQu.replace(
#         {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
```

```
    all_df["SimplFunctional"] = all_df.Functional.replace(
        {1 : 1, 2 : 1, 3 : 2, 4 : 2, 5 : 3, 6 : 3, 7 : 3, 8 : 4})
    all_df["SimplKitchenQual"] = all_df.KitchenQual.replace(
        {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
    all_df["SimplHeatingQC"] = all_df.HeatingQC.replace(
        {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
    all_df["SimplBsmtFinType1"] = all_df.BsmtFinType1.replace(
        {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2, 6 : 2})
    all_df["SimplBsmtFinType2"] = all_df.BsmtFinType2.replace(
        {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2, 6 : 2})
    all_df["SimplBsmtCond"] = all_df.BsmtCond.replace(
        {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
    all_df["SimplBsmtQual"] = all_df.BsmtQual.replace(
        {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
    all_df["SimplExterCond"] = all_df.ExterCond.replace(
        {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
    all_df["SimplExterQual"] = all_df.ExterQual.replace(
        {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})

    # Bin by neighborhood (a little arbitrarily). Values were computed by:
    # train_df["SalePrice"].groupby(train_df["Neighborhood"]).median().sort_values()
    neighborhood_map = {
        "MeadowV" : 0,  #  88000
        "IDOTRR" : 1,   # 103000
        "BrDale" : 1,   # 106000
        "OldTown" : 1,  # 119000
        "Edwards" : 1,  # 119500
        "BrkSide" : 1,  # 124300
        "Sawyer" : 1,   # 135000
        "Blueste" : 1,  # 137500
        "SWISU" : 2,    # 139500
        "NAmes" : 2,    # 140000
        "NPkVill" : 2,  # 146000
        "Mitchel" : 2,  # 153500
        "SawyerW" : 2,  # 179900
        "Gilbert" : 2,  # 181000
        "NWAmes" : 2,   # 182900
        "Blmngtn" : 2,  # 191000
        "CollgCr" : 2,  # 197200
        "ClearCr" : 3,  # 200250
        "Crawfor" : 3,  # 200624
        "Veenker" : 3,  # 218000
        "Somerst" : 3,  # 225500
        "Timber" : 3,   # 228475
        "StoneBr" : 4,  # 278000
        "NoRidge" : 4,  # 290000
        "NridgHt" : 4,  # 315000
    }

    all_df["NeighborhoodBin"] = df["Neighborhood"].map(neighborhood_map)
    return all_df


train_df_munged = munge(train_df)
test_df_munged = munge(test_df)

print(train_df_munged.shape)
print(test_df_munged.shape)


# Copy NeighborhoodBin into a temporary DataFrame because we want to use the
```

```
# unscaled version later on (to one-hot encode it).
neighborhood_bin_train = pd.DataFrame(index = train_df.index)
neighborhood_bin_train["NeighborhoodBin"] = train_df_munged["NeighborhoodBin"]
neighborhood_bin_test = pd.DataFrame(index = test_df.index)
neighborhood_bin_test["NeighborhoodBin"] = test_df_munged["NeighborhoodBin"]
```

```
(1456, 111)
(1459, 111)
```

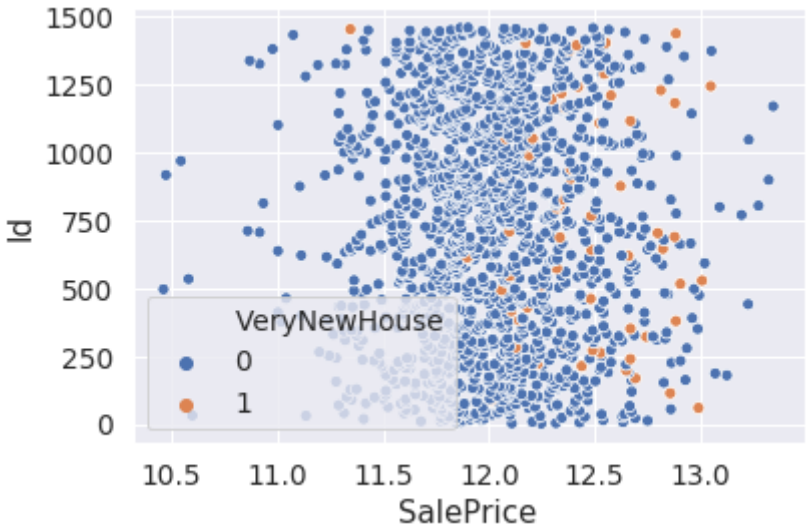
Cell Report:

1. The *factorize* function was used to convert some Categorical features as numbers
2. The *munge* function was used to Combine all the (numerical) features into one big DataFrame.
3. Categorical features were converted to Ordinal, arrganged by quality, where possible in order to extract learnable patterns for a model, with higher number representing higher quality (This significaly inproved the Kaggle score across the board/with all models).
4. As for 'IsRegularLotShape', IR2 and IR3 don't appear that often, so a distinction was made between regular and irregular. Most properties are level; the other possibilities were binned together as "not level". Most land slopes are gentle; the others were binned as "not gentle". 'IsElectricalSBrkr', 'IsGarageDetached', 'IsPavedDrive' and 'HasShed' were result of similar binning.
5. 'Has2ndFloor', 'HasMasVnr', 'HasWoodDeck', 'HasOpenPorch', 'HasEnclosedPorch', 'Has3SsnPorch', and 'HasScreenPorch' bool type featured were added.
6. If 'YearRemodAdd' was not equal to 'YearBuilt', then a remodeling took place at some point, so a feature 'Remodeled' was added, similarly the 'RecentRemodel' was added for houses that were remodelled in the same year of 'YearBuilt'.
7. 'HighSeason' attribute was added as months with the largest number of deals may be significant.
8. 'NewerDwelling' attribute was added by binning 'MSSubClass' as building class correspond with newer houses.
9. 'TotalArea' attribute was added. It is the sum of all *area* attributes. 'TotalArea1st2nd' was calculated in a similar way.
10. 'Age', 'TimeSinceSold', 'YearsSinceRemodel' were all calculated and added as features.
11. 'SeasonSold' attribute was derived from mapping 'MoSold' into seasons.
12. 'Neighborhood' attribute was binned by grouping the locations by median 'SalePrice'. This turned hepled 'Neighborhood' turn into Ordinal, therefore helping a model capture better patterns.
13. Missing values of numerical attributes were filled with 0.
14. Simplifications of existing features into Ordinal bad/average/good were made, this improved the Kaggle score a tiny bit.

2.2.1 Feature Engineering Vizualization(part 1)

In [50]:

```
train_NeighbourhoodBin = pd.concat([train_df_munged,train_df],axis=1)

train_NeighbourhoodBin = train_NeighbourhoodBin.loc[:,~train_NeighbourhoodBin.columns.duplicated()]


ax= sns.scatterplot(x="SalePrice", y="Id", hue="NeighborhoodBin",  data=train_NeighbourhoodBin)
plt.show()
a1= sns.scatterplot(x="SalePrice", y="Id", hue="VeryNewHouse",  data=train_NeighbourhoodBin)
plt.show()
```



Cell Report:

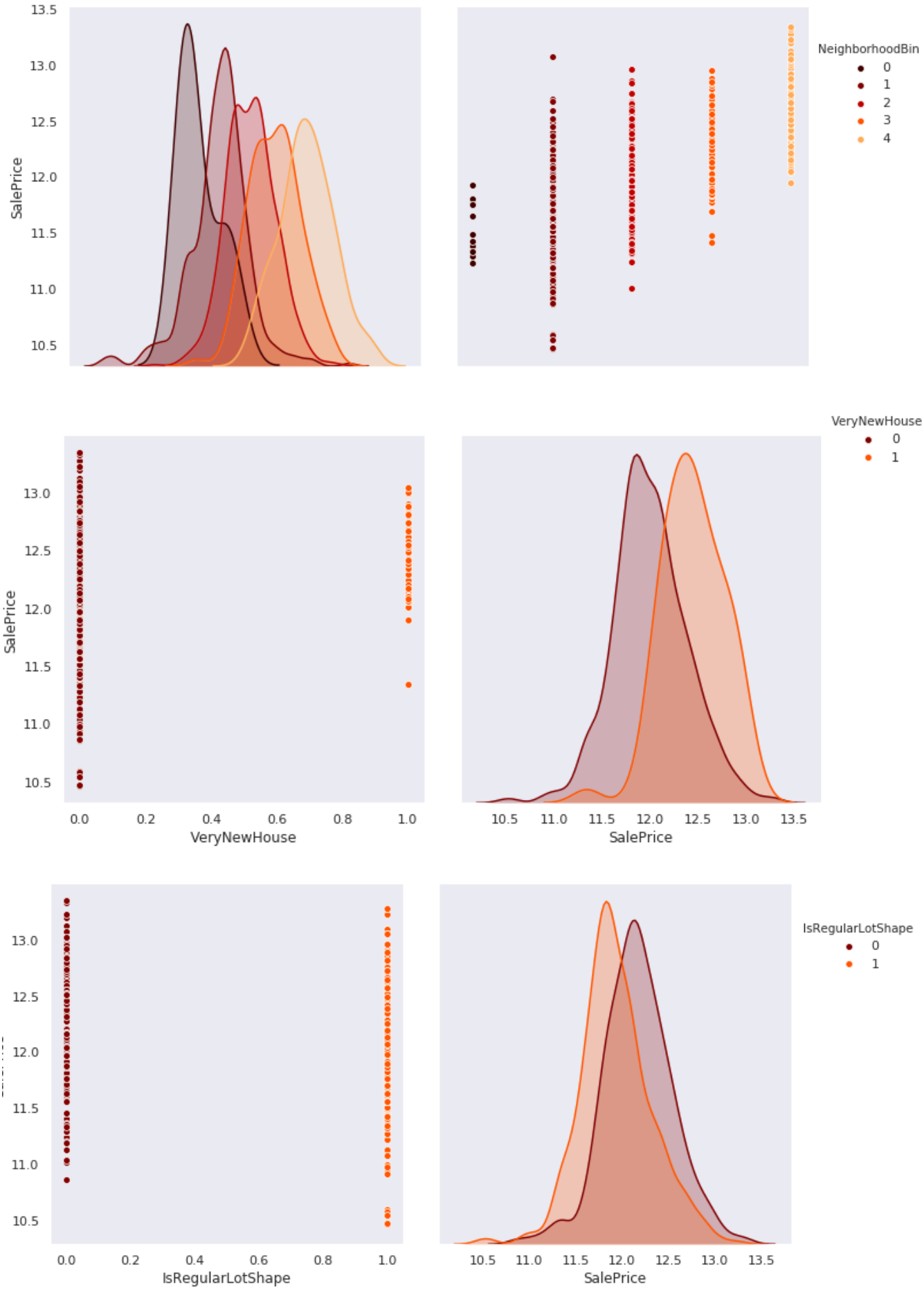
1. The Scatterplot was used to illustrate that both 'NeighborhoodBin' and 'VeryNewHouse' show correlation to 'SalePrice', NeighborhoodBin' more so.

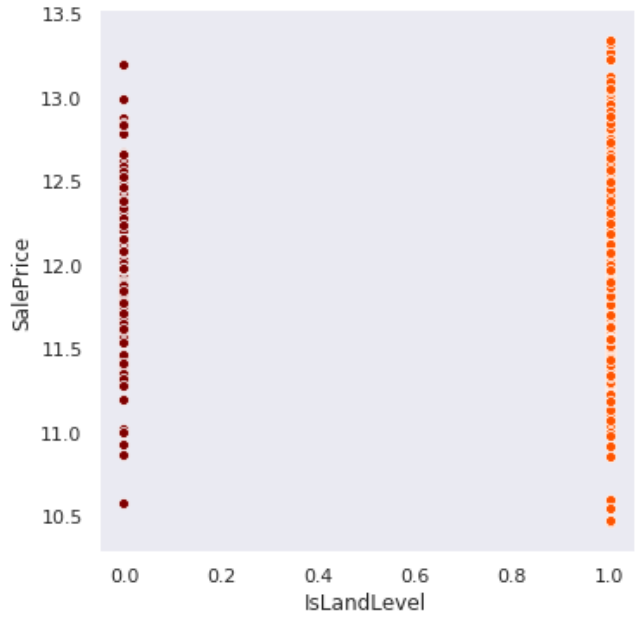
2.2.2 Feature Engineering Vizualization(part 2)

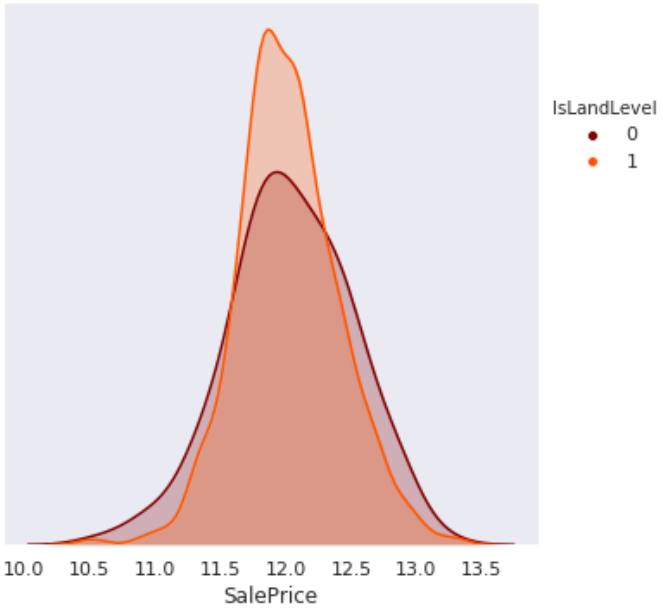
In [ ]:

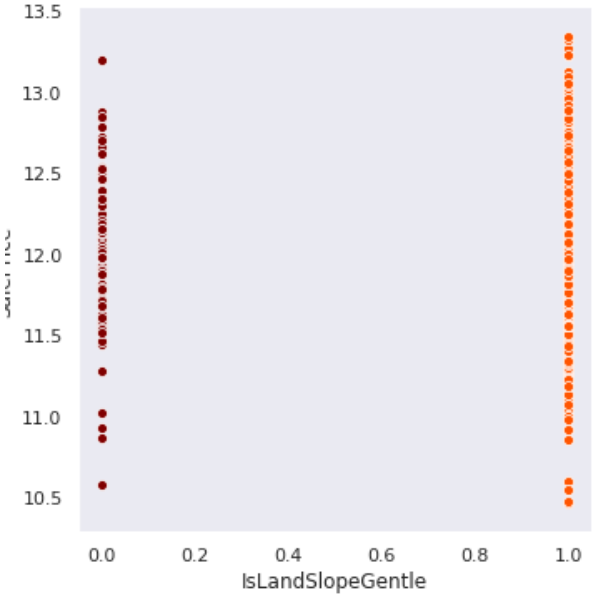
```
col = [ 'SalePrice','NeighborhoodBin']
sns.set_style('dark')

sns.pairplot(train_NeighbourhoodBin[col], hue='NeighborhoodBin', palette='gist_heat', d
ropna=True, size=5)

col = [ 'VeryNewHouse','SalePrice']
sns.set_style('dark')

sns.pairplot(train_NeighbourhoodBin[col], hue='VeryNewHouse', palette='gist_heat', drop
na=True, size=5)

col = [ 'IsRegularLotShape','SalePrice']
sns.set_style('dark')

sns.pairplot(train_NeighbourhoodBin[col], hue='IsRegularLotShape', palette='gist_heat',
dropna=True, height=5)

col = [ 'IsLandLevel','SalePrice']
sns.set_style('dark')

sns.pairplot(train_NeighbourhoodBin[col], hue='IsLandLevel', palette='gist_heat', dropn
a=True, height=5)

col = [ 'IsLandSlopeGentle','SalePrice']
sns.set_style('dark')

sns.pairplot(train_NeighbourhoodBin[col], hue='IsLandSlopeGentle', palette='gist_heat',
dropna=True, height=5)

col = [ 'IsGarageDetached','SalePrice']
sns.set_style('dark')

sns.pairplot(train_NeighbourhoodBin[col], hue='IsGarageDetached', palette='gist_heat',
dropna=True, height=5)

col = [ 'Remodeled','SalePrice']
sns.set_style('dark')

sns.pairplot(train_NeighbourhoodBin[col], hue='Remodeled', palette='gist_heat', dropna=
True, height=3)

col = [ 'RecentRemodel','SalePrice']
sns.set_style('dark')

sns.pairplot(train_NeighbourhoodBin[col], hue='RecentRemodel', palette='gist_heat', dro
pna=True, height=3)

col = [ 'HighSeason','SalePrice']
sns.set_style('dark')

sns.pairplot(train_NeighbourhoodBin[col], hue='HighSeason', palette='gist_heat', dropna
=True, height=3)

col = [ 'BoughtOffPlan','SalePrice']
sns.set_style('dark')

sns.pairplot(train_NeighbourhoodBin[col], hue='BoughtOffPlan', palette='gist_heat', dro
pna=True, height=3)
```

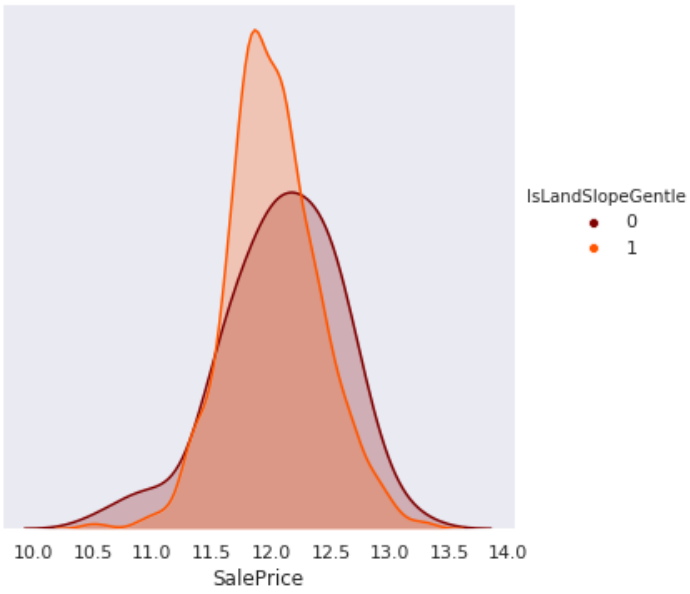
```
col = [ 'BadHeating','SalePrice']
sns.set_style('dark')

sns.pairplot(train_NeighbourhoodBin[col], hue='BadHeating', palette='gist_heat', dropna
=True, height=3)


col = [ 'SeasonSold','SalePrice']
sns.set_style('dark')

sns.pairplot(train_NeighbourhoodBin[col], hue='SeasonSold', palette='gist_heat', dropna
=True, height=3)

col = [ 'YearsSinceRemodel','SalePrice']
sns.set_style('dark')

sns.pairplot(train_NeighbourhoodBin[col], hue='YearsSinceRemodel', palette='gist_heat',
dropna=True, height=3)
```

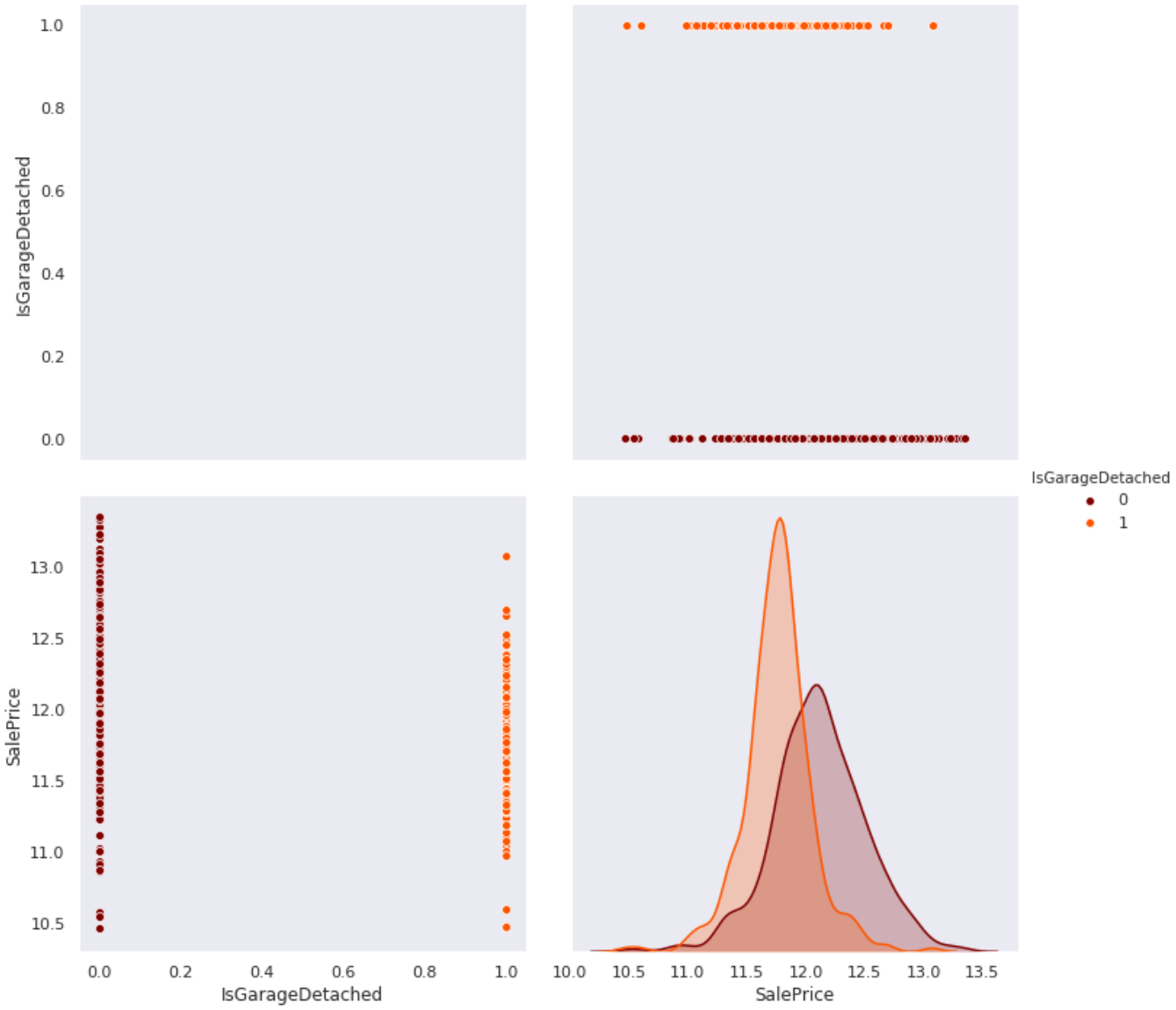

SalePrice
NeighborhoodBin
0
1
2
3
4
VeryNewHouse
0
1
VeryNewHouse
SalePrice
IsRegularLotShape
0
1
IsRegularLotShape
SalePrice

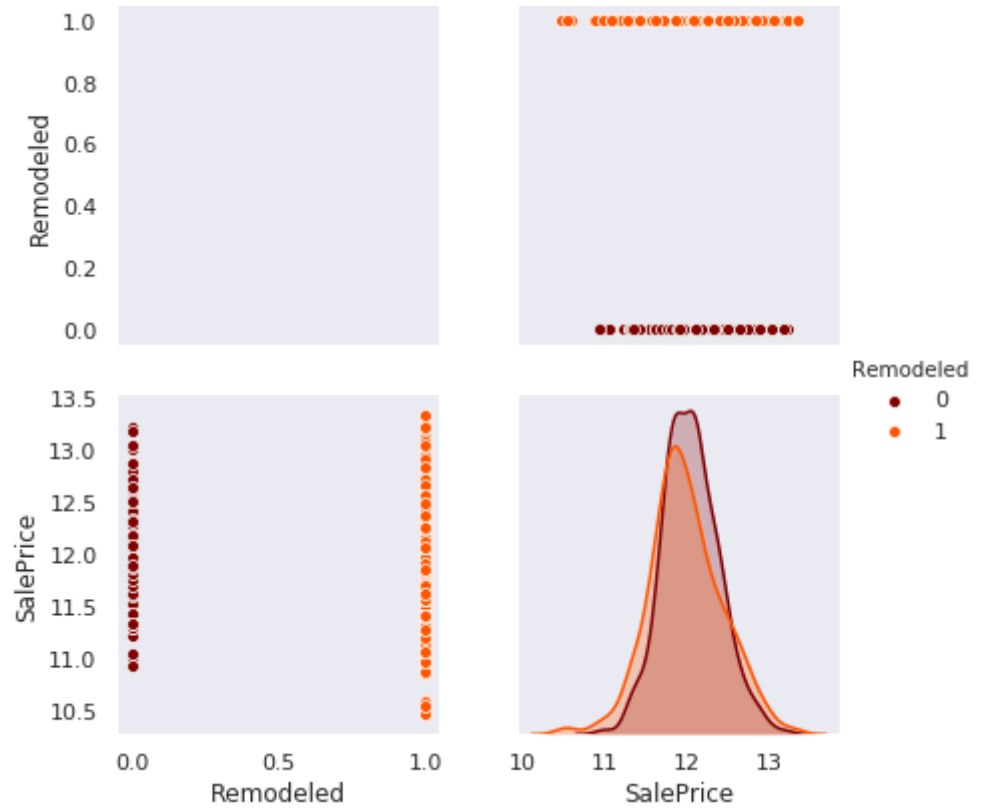

SalePrice
IsLandLevel

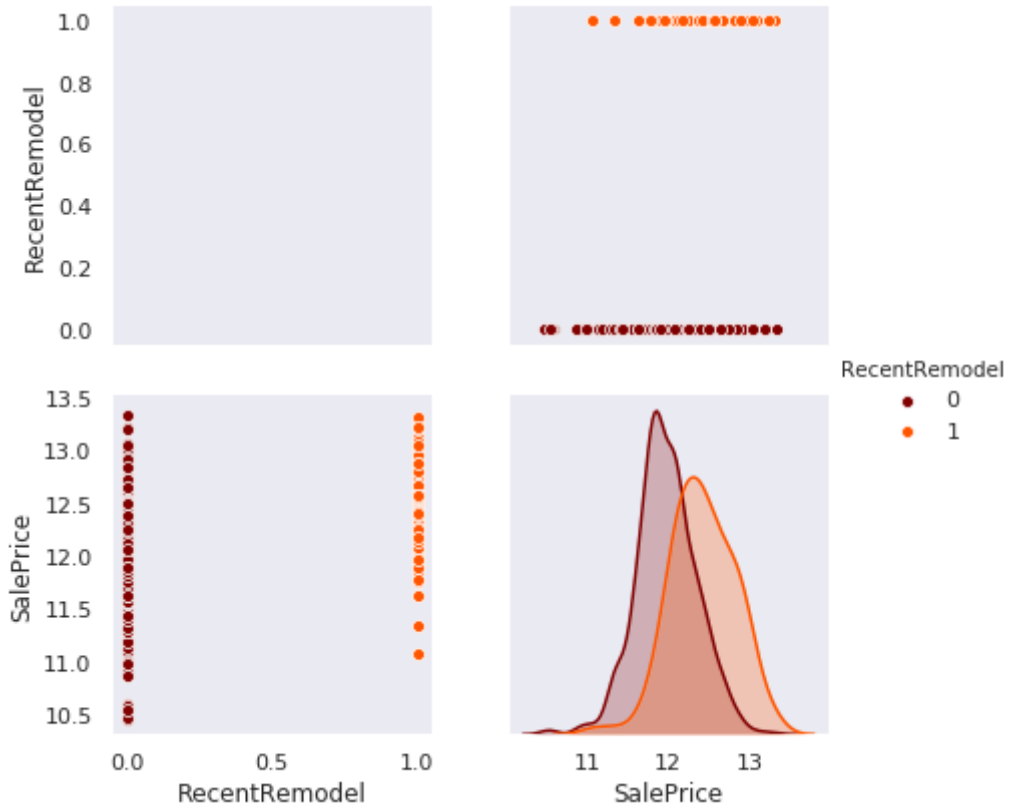

IsLandLevel
0
1
SalePrice

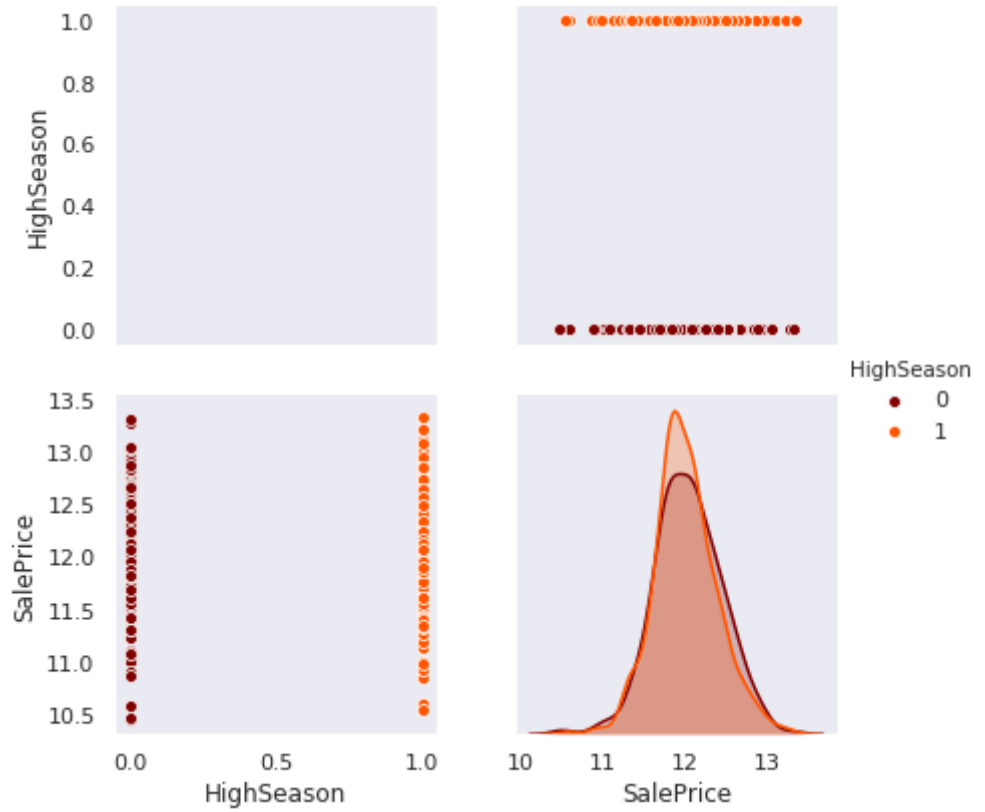

IsLandSlopeGentle

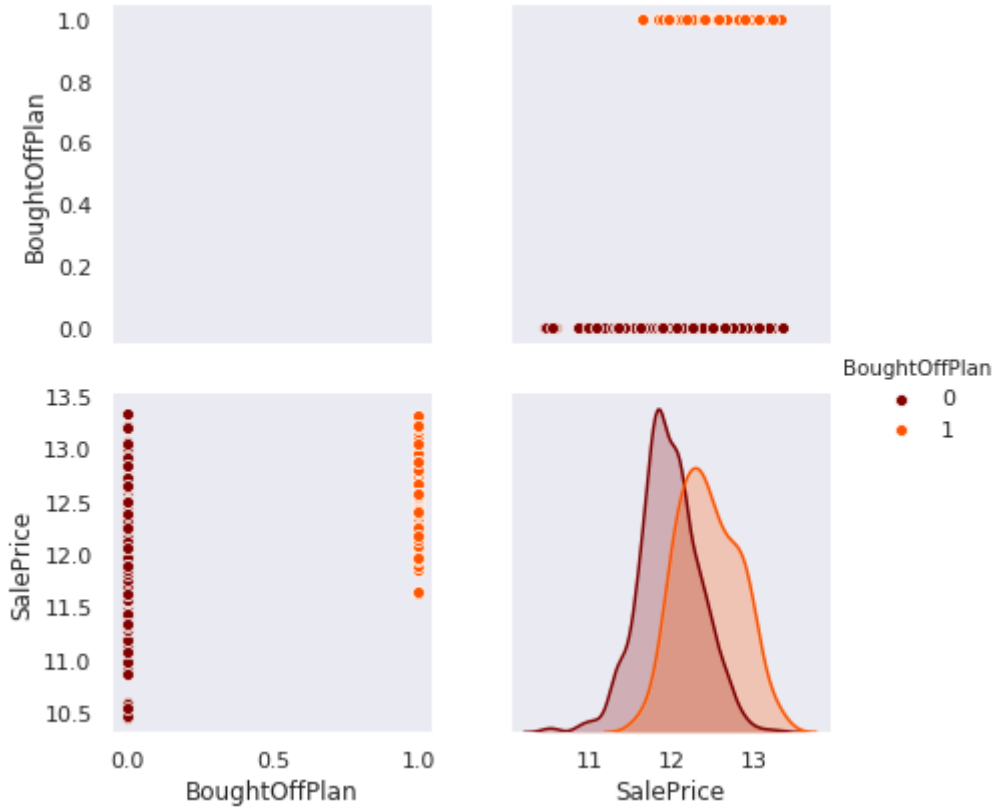

IsLandSlopeGentle
0
1
SalePrice

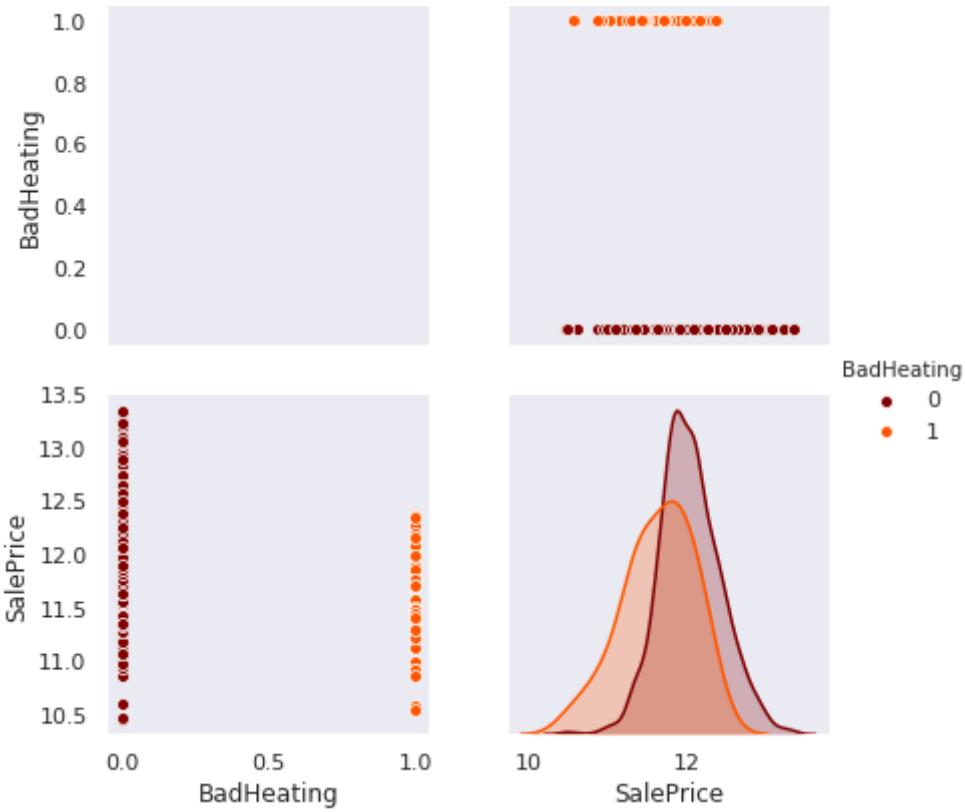

IsGarageDetached
SalePrice
IsGarageDetached
SalePrice
IsGarageDetached
0
1

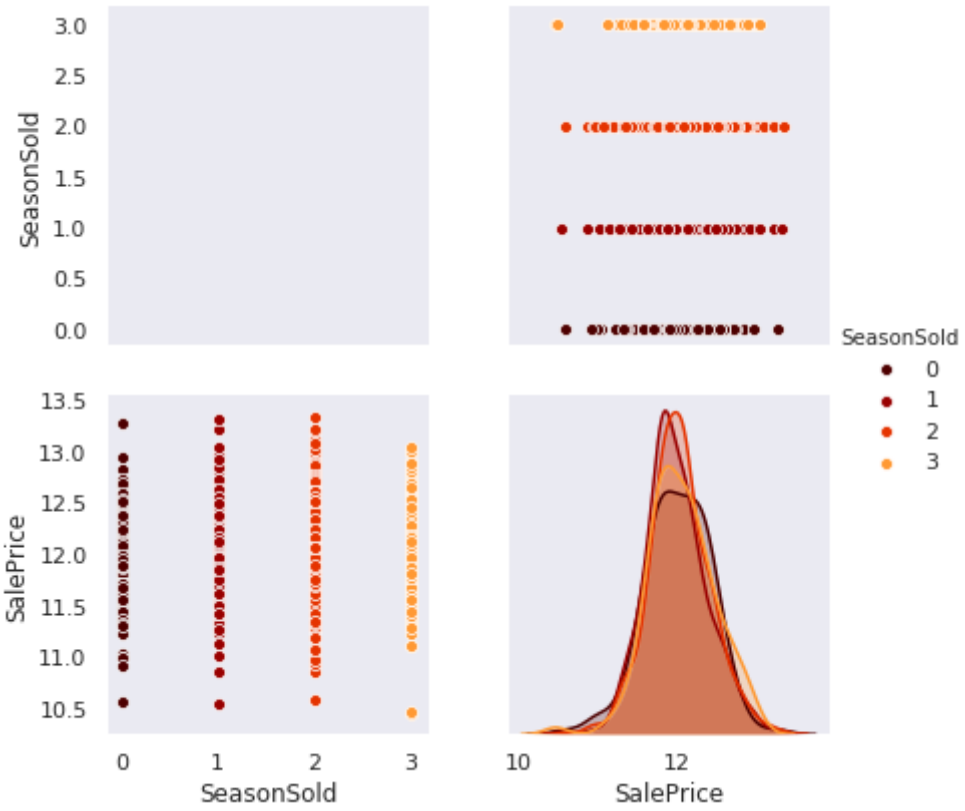

Remodeled
SalePrice
Remodeled
SalePrice
Remodeled
0
1

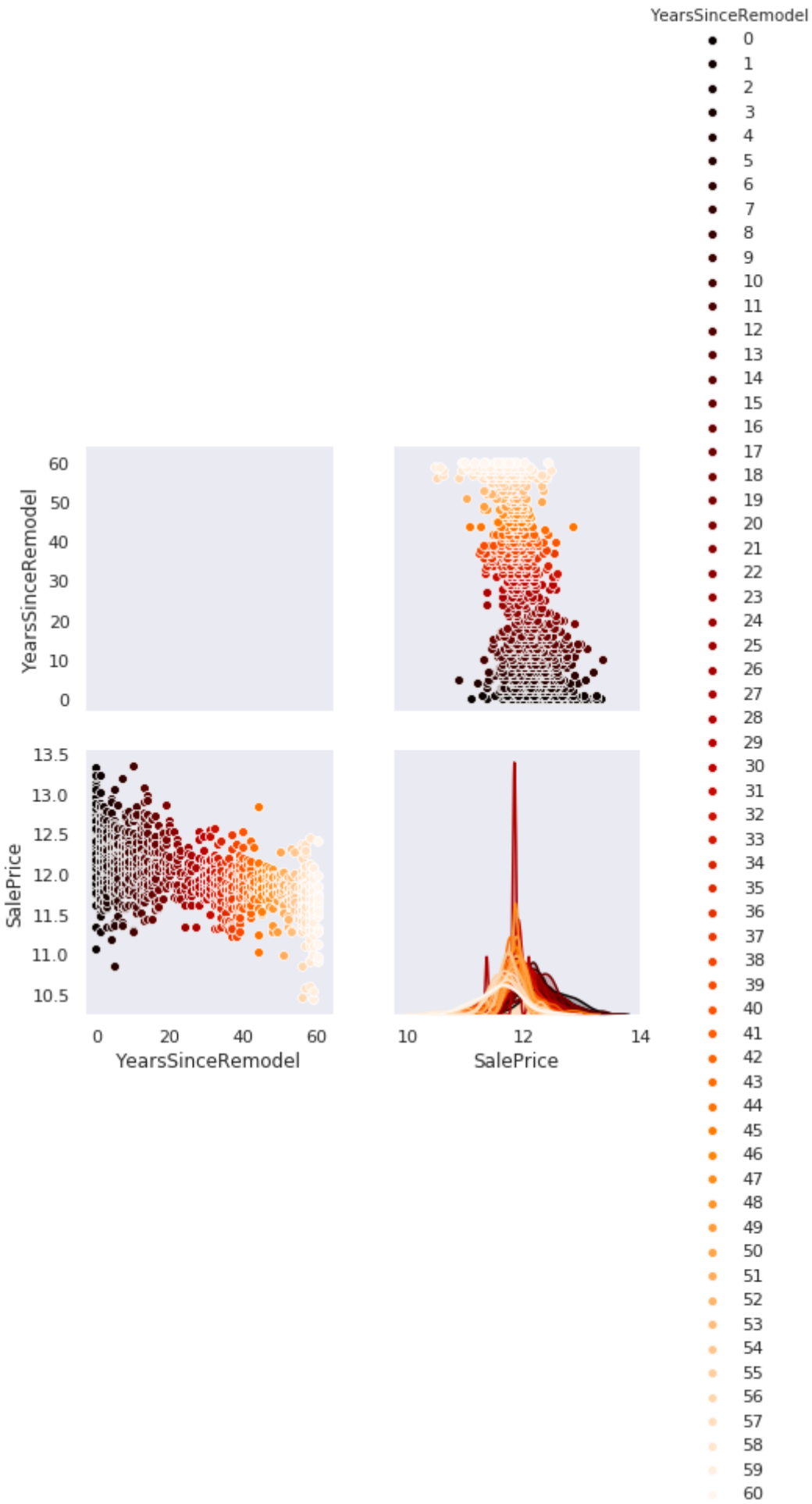

RecentRemodel
SalePrice
RecentRemodel
SalePrice
RecentRemodel
0
1


HighSeason
1.0
0.8
0.6
0.4
0.2
0.0
SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
0.0
0.5
1.0
HighSeason
10
11
12
13
SalePrice
HighSeason
0
1


BoughtOffPlan
1.0
0.8
0.6
0.4
0.2
0.0
SalePrice
13.5
13.0
12.5
12.0
11.5
11.0
10.5
0.0
0.5
1.0
BoughtOffPlan
11
12
13
SalePrice
BoughtOffPlan
0
1


BadHeating
SalePrice
BadHeating
SalePrice
BadHeating
0
1


SeasonSold
SalePrice
SeasonSold
SalePrice
SeasonSold
0
1
2
3


YearsSinceRemodel
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
YearsSinceRemodel
SalePrice
SalePrice
YearsSinceRemodel

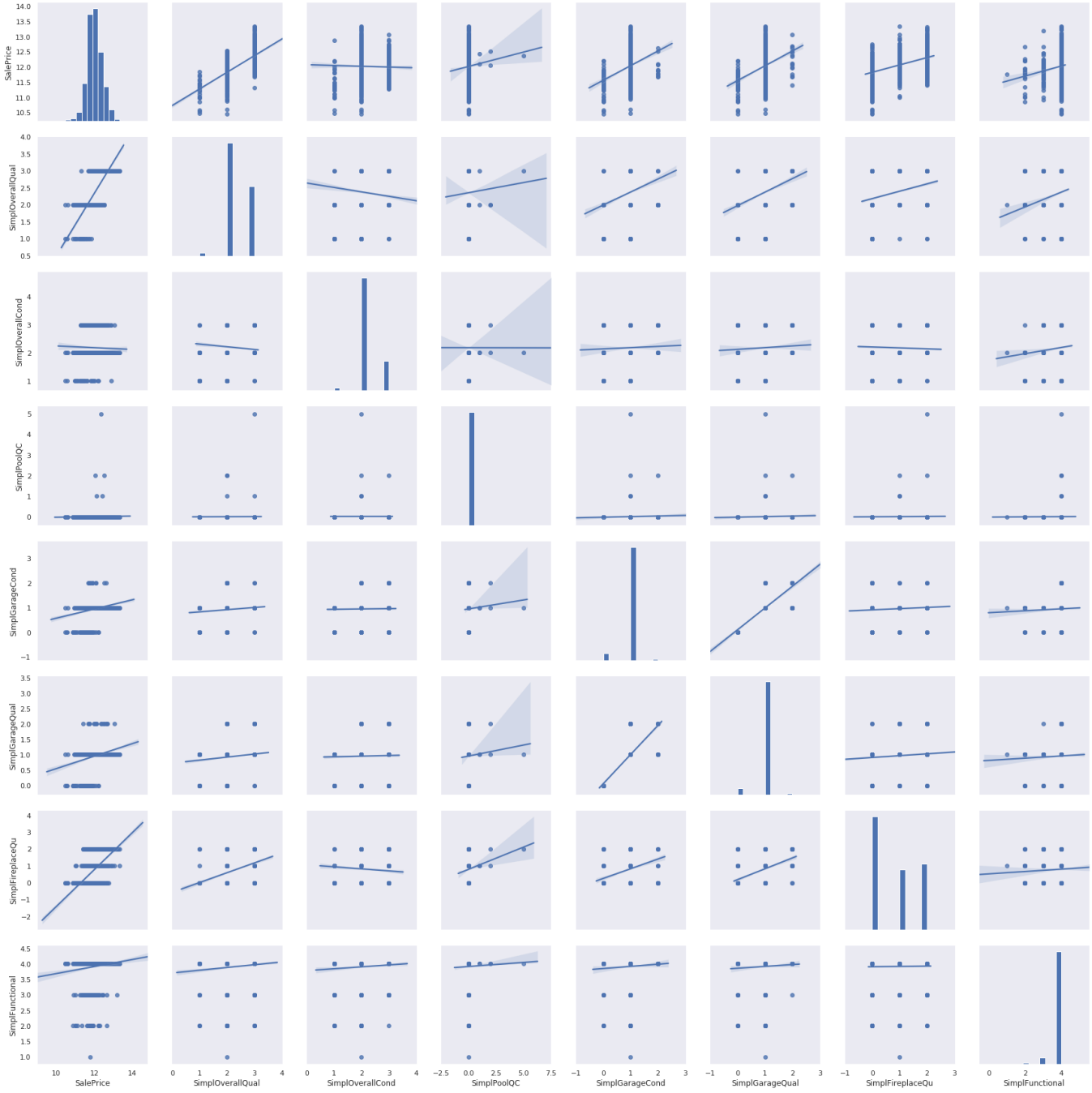
Cell Report:

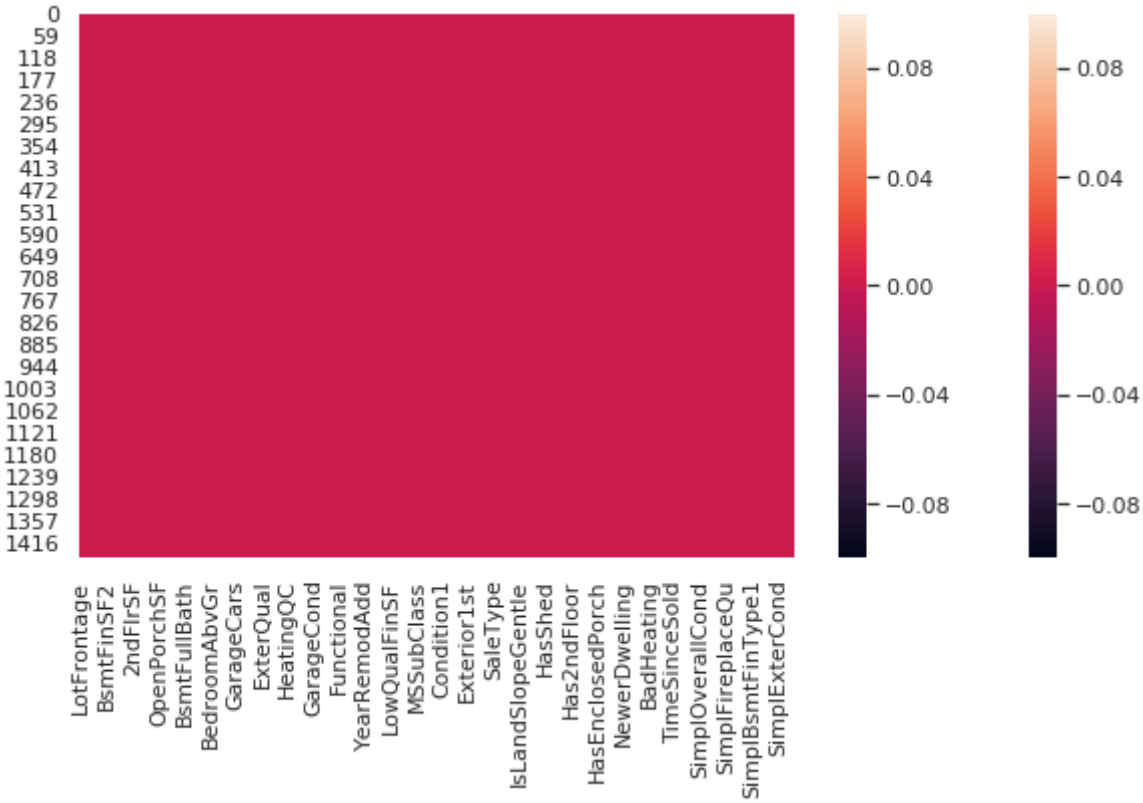
1. Seaborn Pairplot was used to verify effictiveness of some the engineered features.

2.2.3 Feature Engineering Vizualization(part 3)

In [ ]:

```
col = ['SalePrice', 'SimplOverallQual', 'SimplOverallCond', 'SimplPoolQC', 'SimplGarage
Cond', 'SimplGarageQual', 'SimplFireplaceQu', 'SimplFunctional']
sns.set(style='ticks')
sns.set_style('dark')

sns.pairplot(train_NeighbourhoodBin[col], height=3, kind='reg')

#Checking there is any null value or not
plt.figure(figsize=(10, 5))
sns.heatmap(train_df_munged.isnull())
sns.heatmap(test_df_munged.isnull())
```


SalePrice
SimplOverallQual
SimplOverallCond
SimplPoolQC
SimplGarageCond
SimplGarageQual
SimplFireplaceQu
SimplFunctional



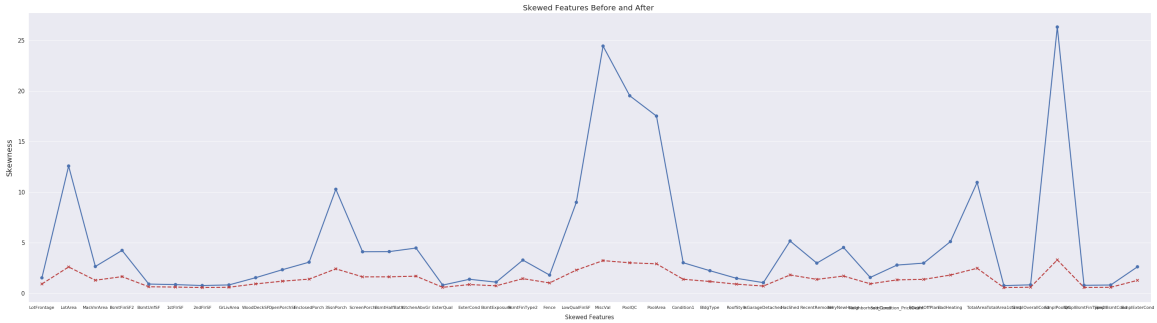
Cell Report:

1. Seaborn Pairplot was used to verify effictiveness of the *Simplified* features.
2. Final Null checks were done to make sure there were no *missing values* left.

2.3 Skew Fix and Data Normalization

In [51]:

```
numeric_features = train_df_munged.dtypes[train_df_munged.dtypes != "object"].index

# Transform the skewed numeric features by taking log(feature + 1).
# This will make the features more normal.
from scipy.stats import skew

skewed = train_df_munged[numeric_features].apply(lambda x: skew(x.dropna().astype(float
)))
skewed = skewed[skewed > 0.75]
skewedNParray= skewed
skewed = skewed.index


train_df_munged[skewed] = np.log1p(train_df_munged[skewed])
test_df_munged[skewed] = np.log1p(test_df_munged[skewed])

###
df_skew = pd.DataFrame(index=skewed, columns=['Skew', 'Skew after boxcox1p'])
df_skew['Skew'] = skewedNParray.values
df_skew['Skew after boxcox1p'] = np.log1p(skewedNParray.values)
###

# figure before/after
fig = plt.figure(figsize=(53, 14))

sns.pointplot(x=df_skew.index, y='Skew', data=df_skew, markers=['o'], linestyles=['-'])
sns.pointplot(x=df_skew.index, y='Skew after boxcox1p', data=df_skew, markers=['x'], li
nestyles=['--'], color='#bb3f3f')

plt.xlabel('Skewed Features', size=15, labelpad=12.5)
plt.ylabel('Skewness', size=20, labelpad=12.5)
plt.tick_params(axis='x', labelsize=11)
plt.tick_params(axis='y', labelsize=15)

plt.title('Skewed Features Before and After ', size=20)

plt.show()


plt.show()
# Additional processing: scale the data.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(train_df_munged[numeric_features])

scaled = scaler.transform(train_df_munged[numeric_features])
for i, col in enumerate(numeric_features):
    train_df_munged[col] = scaled[:, i]

scaled = scaler.transform(test_df_munged[numeric_features])
for i, col in enumerate(numeric_features):
    test_df_munged[col] = scaled[:, i]
```




```
/opt/conda/lib/python3.6/site-packages/sklearn/preprocessing/data.py:645:
DataConversionWarning: Data with input dtype int64, float64 were all conve
rted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:44: DataConve
rsionWarning: Data with input dtype int64, float64 were all converted to f
loat64 by StandardScaler.
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:48: DataConve
rsionWarning: Data with input dtype int64, float64, object were all conver
ted to float64 by StandardScaler.
```

Cell Report:

1. Skewed featured were treated with np.log1p function.
2. Numerical features were Normalized or work better with Deep Neural Networks and **convergence** is not hindered. [7]

2.4 One-hot Encoding

In [52]:

```
# Convert categorical features using one-hot encoding.
def onehot(onehot_df, df, column_name, fill_na, drop_name):
    onehot_df[column_name] = df[column_name]
    if fill_na is not None:
        onehot_df[column_name].fillna(fill_na, inplace=True)

    dummies = pd.get_dummies(onehot_df[column_name], prefix="_" + column_name)

    # Dropping one of the columns actually made the results slightly worse.
    # if drop_name is not None:
    #     dummies.drop(["_" + column_name + "_" + drop_name], axis=1, inplace=True)

    onehot_df = onehot_df.join(dummies)
    onehot_df = onehot_df.drop([column_name], axis=1)
    return onehot_df

def munge_onehot(df):
    onehot_df = pd.DataFrame(index = df.index)

    onehot_df = onehot(onehot_df, df, "MSSubClass", None, "40")
    onehot_df = onehot(onehot_df, df, "MSZoning", "RL", "RH")
    onehot_df = onehot(onehot_df, df, "LotConfig", None, "FR3")
    onehot_df = onehot(onehot_df, df, "Neighborhood", None, "OldTown")
    onehot_df = onehot(onehot_df, df, "Condition1", None, "RRNe")
    onehot_df = onehot(onehot_df, df, "BldgType", None, "2fmCon")
    onehot_df = onehot(onehot_df, df, "HouseStyle", None, "1.5Unf")
    onehot_df = onehot(onehot_df, df, "RoofStyle", None, "Shed")
    onehot_df = onehot(onehot_df, df, "Exterior1st", "VinylSd", "CBlock")
    onehot_df = onehot(onehot_df, df, "Exterior2nd", "VinylSd", "CBlock")
    onehot_df = onehot(onehot_df, df, "Foundation", None, "Wood")
    onehot_df = onehot(onehot_df, df, "SaleType", "WD", "Oth")
    onehot_df = onehot(onehot_df, df, "SaleCondition", "Normal", "AdjLand")

    # Fill in missing MasVnrType for rows that do have a MasVnrArea.
    temp_df = df[["MasVnrType", "MasVnrArea"]].copy()
    idx = (df["MasVnrArea"] != 0) & ((df["MasVnrType"] == "None") | (df["MasVnrType"].i
snull()))
    temp_df.loc[idx, "MasVnrType"] = "BrkFace"
    onehot_df = onehot(onehot_df, temp_df, "MasVnrType", "None", "BrkCmn")

    # Also add the booleans from calc_df as dummy variables.
    onehot_df = onehot(onehot_df, df, "LotShape", None, "IR3")
    onehot_df = onehot(onehot_df, df, "LandContour", None, "Low")
    onehot_df = onehot(onehot_df, df, "LandSlope", None, "Sev")
    onehot_df = onehot(onehot_df, df, "Electrical", "SBrkr", "FuseP")
    onehot_df = onehot(onehot_df, df, "GarageType", "None", "CarPort")
    onehot_df = onehot(onehot_df, df, "PavedDrive", None, "P")
    onehot_df = onehot(onehot_df, df, "MiscFeature", "None", "Othr")

    # Features we can probably ignore (but want to include anyway to see
    # if they make any positive difference).
    # Definitely ignoring Utilities: all records are "AllPub", except for
    # one "NoSeWa" in the train set and 2 NA in the test set.
    onehot_df = onehot(onehot_df, df, "Street", None, "Grvl")
    onehot_df = onehot(onehot_df, df, "Alley", "None", "Grvl")
    onehot_df = onehot(onehot_df, df, "Condition2", None, "PosA")
    onehot_df = onehot(onehot_df, df, "RoofMatl", None, "WdShake")
    onehot_df = onehot(onehot_df, df, "Heating", None, "Wall")
```

```
    # have these as numerical variables
    onehot_df = onehot(onehot_df, df, "ExterQual", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "ExterCond", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "BsmtQual", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "BsmtCond", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "HeatingQC", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "KitchenQual", "TA", "Ex")
    onehot_df = onehot(onehot_df, df, "FireplaceQu", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "GarageQual", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "GarageCond", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "PoolQC", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "BsmtExposure", "None", "Gd")
    onehot_df = onehot(onehot_df, df, "BsmtFinType1", "None", "GLQ")
    onehot_df = onehot(onehot_df, df, "BsmtFinType2", "None", "GLQ")
    onehot_df = onehot(onehot_df, df, "Functional", "Typ", "Typ")
    onehot_df = onehot(onehot_df, df, "GarageFinish", "None", "Fin")
    onehot_df = onehot(onehot_df, df, "Fence", "None", "MnPrv")
    onehot_df = onehot(onehot_df, df, "MoSold", None, None)

    # Divide up the years between 1871 and 2010 in slices of 20 years.
    year_map = pd.concat(pd.Series("YearBin" + str(i+1), index=range(1871+i*20,1891+i*2
0)) for i in range(0, 7))

    yearbin_df = pd.DataFrame(index = df.index)
    yearbin_df["GarageYrBltBin"] = df.GarageYrBlt.map(year_map)
    yearbin_df["GarageYrBltBin"].fillna("NoGarage", inplace=True)

    yearbin_df["YearBuiltBin"] = df.YearBuilt.map(year_map)
    yearbin_df["YearRemodAddBin"] = df.YearRemodAdd.map(year_map)

    onehot_df = onehot(onehot_df, yearbin_df, "GarageYrBltBin", None, None)
    onehot_df = onehot(onehot_df, yearbin_df, "YearBuiltBin", None, None)
    onehot_df = onehot(onehot_df, yearbin_df, "YearRemodAddBin", None, None)

    return onehot_df

# Add the one-hot encoded categorical features.
onehot_df = munge_onehot(train_df)
onehot_df = onehot(onehot_df, neighborhood_bin_train, "NeighborhoodBin", None, None)
train_df_munged = train_df_munged.join(onehot_df)
```

Cell Report:

1. One-hot Encoding was done rather than Label Encoding as it gives better accuracy with Deep Neural Networks [9].
2. The years were divided between 1871 and 2010 in slices of 20 years.
3. Utilities was left out as it was not useful: all records are "AllPub", except for one "NoSeWa" in the train set and 2 NA in the test set.

Feature Dropping (Exclusive Cell)

In [53]:

```
# These onehot columns are missing in the test data, so drop them from the
# training data or we might overfit on them.
drop_cols = [
                "_Exterior1st_ImStucc", "_Exterior1st_Stone",
                "_Exterior2nd_Other","_HouseStyle_2.5Fin",

                "_RoofMatl_Membran", "_RoofMatl_Metal", "_RoofMatl_Roll",
                "_Condition2_RRAe", "_Condition2_RRAn", "_Condition2_RRNn",
                "_Heating_Floor", "_Heating_OthW",

                "_Electrical_Mix",
                "_MiscFeature_TenC",
                "_GarageQual_Ex", "_PoolQC_Fa"
            ]
train_df_munged.drop(drop_cols, axis=1, inplace=True)

onehot_df = munge_onehot(test_df)
onehot_df = onehot(onehot_df, neighborhood_bin_test, "NeighborhoodBin", None, None)
test_df_munged = test_df_munged.join(onehot_df)

# This column is missing in the training data. There is only one example with
# this value in the test set. So just drop it.
test_df_munged.drop(["_MSSubClass_150"], axis=1, inplace=True)

# Drop these columns. They are either not very helpful or they cause overfitting.
drop_cols = [
    "_Condition2_PosN",    # only two are not zero
    "_MSZoning_C (all)",
    "_MSSubClass_160",
]
train_df_munged.drop(drop_cols, axis=1, inplace=True)
test_df_munged.drop(drop_cols, axis=1, inplace=True)
```

Note: These onehot columns are missing in the test data, so drop them from the training data or we might overfit on them.

Additioncal Data Prep (Exclusive Cell)

In [54]:

```
label_df = pd.DataFrame(index = train_df_munged.index, columns=["SalePrice"])
label_df["SalePrice"] = train_df["SalePrice"]
print(list(train_df_munged))
print("Training set size:", train_df_munged.shape)
print("Test set size:", test_df_munged.shape)
```

['LotFrontage', 'LotArea', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'Bsmt
UnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'GarageArea',
'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch',
'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'K
itchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'CentralAir', 'O
verallQual', 'OverallCond', 'ExterQual', 'ExterCond', 'BsmtQual', 'BsmtCon
d', 'HeatingQC', 'KitchenQual', 'FireplaceQu', 'GarageQual', 'GarageCond',
'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Functional', 'GarageFinis
h', 'Fence', 'YearBuilt', 'YearRemodAdd', 'GarageYrBlt', 'MoSold', 'YrSol
d', 'LowQualFinSF', 'MiscVal', 'PoolQC', 'PoolArea', 'MSSubClass', 'MSZoni
ng', 'LotConfig', 'Neighborhood', 'Condition1', 'BldgType', 'HouseStyle',
'RoofStyle', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'Foundation', 'Sa
leType', 'SaleCondition', 'IsRegularLotShape', 'IsLandLevel', 'IsLandSlope
Gentle', 'IsElectricalSBrkr', 'IsGarageDetached', 'IsPavedDrive', 'HasShe
d', 'Remodeled', 'RecentRemodel', 'VeryNewHouse', 'Has2ndFloor', 'HasMasVn
r', 'HasWoodDeck', 'HasOpenPorch', 'HasEnclosedPorch', 'Has3SsnPorch', 'Ha
sScreenPorch', 'HighSeason', 'NewerDwelling', 'Neighborhood_Good', 'SaleCo
ndition_PriceDown', 'BoughtOffPlan', 'BadHeating', 'TotalArea', 'TotalArea
1st2nd', 'Age', 'TimeSinceSold', 'SeasonSold', 'YearsSinceRemodel', 'Simpl
OverallQual', 'SimplOverallCond', 'SimplPoolQC', 'SimplGarageCond', 'Simpl
GarageQual', 'SimplFireplaceQu', 'SimplFunctional', 'SimplKitchenQual', 'S
implHeatingQC', 'SimplBsmtFinType1', 'SimplBsmtFinType2', 'SimplBsmtCond',
'SimplBsmtQual', 'SimplExterCond', 'SimplExterQual', 'NeighborhoodBin', '_
MSSubClass_20', '_MSSubClass_30', '_MSSubClass_40', '_MSSubClass_45', '_MS
SubClass_50', '_MSSubClass_60', '_MSSubClass_70', '_MSSubClass_75', '_MSSu
bClass_80', '_MSSubClass_85', '_MSSubClass_90', '_MSSubClass_120', '_MSSub
Class_180', '_MSSubClass_190', '_MSZoning_FV', '_MSZoning_RH', '_MSZoning_
RL', '_MSZoning_RM', '_LotConfig_Corner', '_LotConfig_CulDSac', '_LotConfi
g_FR2', '_LotConfig_FR3', '_LotConfig_Inside', '_Neighborhood_Blmngtn', '_
Neighborhood_Blueste', '_Neighborhood_BrDale', '_Neighborhood_BrkSide', '_
Neighborhood_ClearCr', '_Neighborhood_CollgCr', '_Neighborhood_Crawfor',
'_Neighborhood_Edwards', '_Neighborhood_Gilbert', '_Neighborhood_IDOTRR',
'_Neighborhood_MeadowV', '_Neighborhood_Mitchel', '_Neighborhood_NAmes',
'_Neighborhood_NPkVill', '_Neighborhood_NWAmes', '_Neighborhood_NoRidge',
'_Neighborhood_NridgHt', '_Neighborhood_OldTown', '_Neighborhood_SWISU',
'_Neighborhood_Sawyer', '_Neighborhood_SawyerW', '_Neighborhood_Somerst',
'_Neighborhood_StoneBr', '_Neighborhood_Timber', '_Neighborhood_Veenker',
'_Condition1_Artery', '_Condition1_Feedr', '_Condition1_Norm', '_Condition
1_PosA', '_Condition1_PosN', '_Condition1_RRAe', '_Condition1_RRAn', '_Con
dition1_RRNe', '_Condition1_RRNn', '_BldgType_1Fam', '_BldgType_2fmCon',
'_BldgType_Duplex', '_BldgType_Twnhs', '_BldgType_TwnhsE', '_HouseStyle_1.
5Fin', '_HouseStyle_1.5Unf', '_HouseStyle_1Story', '_HouseStyle_2.5Unf',
'_HouseStyle_2Story', '_HouseStyle_SFoyer', '_HouseStyle_SLvl', '_RoofStyl
e_Flat', '_RoofStyle_Gable', '_RoofStyle_Gambrel', '_RoofStyle_Hip', '_Roo
fStyle_Mansard', '_RoofStyle_Shed', '_Exterior1st_AsbShng', '_Exterior1st_
AsphShn', '_Exterior1st_BrkComm', '_Exterior1st_BrkFace', '_Exterior1st_CB
lock', '_Exterior1st_CemntBd', '_Exterior1st_HdBoard', '_Exterior1st_Metal
Sd', '_Exterior1st_Plywood', '_Exterior1st_Stucco', '_Exterior1st_VinylS
d', '_Exterior1st_Wd Sdng', '_Exterior1st_WdShing', '_Exterior2nd_AsbShn
g', '_Exterior2nd_AsphShn', '_Exterior2nd_Brk Cmn', '_Exterior2nd_BrkFac
e', '_Exterior2nd_CBlock', '_Exterior2nd_CmentBd', '_Exterior2nd_HdBoard',
'_Exterior2nd_ImStucc', '_Exterior2nd_MetalSd', '_Exterior2nd_Plywood', '_
Exterior2nd_Stone', '_Exterior2nd_Stucco', '_Exterior2nd_VinylSd', '_Exter
ior2nd_Wd Sdng', '_Exterior2nd_Wd Shng', '_Foundation_BrkTil', '_Foundatio
n_CBlock', '_Foundation_PConc', '_Foundation_Slab', '_Foundation_Stone',
'_Foundation_Wood', '_SaleType_COD', '_SaleType_CWD', '_SaleType_Con', '_S
aleType_ConLD', '_SaleType_ConLI', '_SaleType_ConLw', '_SaleType_New', '_S
aleType_Oth', '_SaleType_WD', '_SaleCondition_Abnorml', '_SaleCondition_Ad
jLand', '_SaleCondition_Alloca', '_SaleCondition_Family', '_SaleCondition_
Normal', '_SaleCondition_Partial', '_MasVnrType_BrkCmn', '_MasVnrType_BrkF
ace', '_MasVnrType_None', '_MasVnrType_Stone', '_LotShape_IR1', '_LotShape

```
_IR2', '_LotShape_IR3', '_LotShape_Reg', '_LandContour_Bnk', '_LandContour
_HLS', '_LandContour_Low', '_LandContour_Lvl', '_LandSlope_Gtl', '_LandSlo
pe_Mod', '_LandSlope_Sev', '_Electrical_FuseA', '_Electrical_FuseF', '_Ele
ctrical_FuseP', '_Electrical_SBrkr', '_GarageType_2Types', '_GarageType_At
tchd', '_GarageType_Basment', '_GarageType_BuiltIn', '_GarageType_CarPor
t', '_GarageType_Detchd', '_GarageType_None', '_PavedDrive_N', '_PavedDriv
e_P', '_PavedDrive_Y', '_MiscFeature_Gar2', '_MiscFeature_None', '_MiscFea
ture_Othr', '_MiscFeature_Shed', '_Street_Grvl', '_Street_Pave', '_Alley_G
rvl', '_Alley_None', '_Alley_Pave', '_Condition2_Artery', '_Condition2_Fee
dr', '_Condition2_Norm', '_Condition2_PosA', '_RoofMatl_CompShg', '_RoofMa
tl_Tar&Grv', '_RoofMatl_WdShake', '_RoofMatl_WdShngl', '_Heating_GasA', '_
Heating_GasW', '_Heating_Grav', '_Heating_Wall', '_ExterQual_Ex', '_ExterQ
ual_Fa', '_ExterQual_Gd', '_ExterQual_TA', '_ExterCond_Ex', '_ExterCond_F
a', '_ExterCond_Gd', '_ExterCond_Po', '_ExterCond_TA', '_BsmtQual_Ex', '_B
smtQual_Fa', '_BsmtQual_Gd', '_BsmtQual_None', '_BsmtQual_TA', '_BsmtCond_
Fa', '_BsmtCond_Gd', '_BsmtCond_None', '_BsmtCond_Po', '_BsmtCond_TA', '_H
eatingQC_Ex', '_HeatingQC_Fa', '_HeatingQC_Gd', '_HeatingQC_Po', '_Heating
QC_TA', '_KitchenQual_Ex', '_KitchenQual_Fa', '_KitchenQual_Gd', '_Kitchen
Qual_TA', '_FireplaceQu_Ex', '_FireplaceQu_Fa', '_FireplaceQu_Gd', '_Firep
laceQu_None', '_FireplaceQu_Po', '_FireplaceQu_TA', '_GarageQual_Fa', '_Ga
rageQual_Gd', '_GarageQual_None', '_GarageQual_Po', '_GarageQual_TA', '_Ga
rageCond_Ex', '_GarageCond_Fa', '_GarageCond_Gd', '_GarageCond_None', '_Ga
rageCond_Po', '_GarageCond_TA', '_PoolQC_Ex', '_PoolQC_Gd', '_PoolQC_Non
e', '_BsmtExposure_Av', '_BsmtExposure_Gd', '_BsmtExposure_Mn', '_BsmtExpo
sure_No', '_BsmtExposure_None', '_BsmtFinType1_ALQ', '_BsmtFinType1_BLQ',
'_BsmtFinType1_GLQ', '_BsmtFinType1_LwQ', '_BsmtFinType1_None', '_BsmtFinT
ype1_Rec', '_BsmtFinType1_Unf', '_BsmtFinType2_ALQ', '_BsmtFinType2_BLQ',
'_BsmtFinType2_GLQ', '_BsmtFinType2_LwQ', '_BsmtFinType2_None', '_BsmtFinT
ype2_Rec', '_BsmtFinType2_Unf', '_Functional_Maj1', '_Functional_Maj2', '_
Functional_Min1', '_Functional_Min2', '_Functional_Mod', '_Functional_Se
v', '_Functional_Typ', '_GarageFinish_Fin', '_GarageFinish_None', '_Garage
Finish_RFn', '_GarageFinish_Unf', '_Fence_GdPrv', '_Fence_GdWo', '_Fence_M
nPrv', '_Fence_MnWw', '_Fence_None', '_MoSold_1', '_MoSold_2', '_MoSold_
3', '_MoSold_4', '_MoSold_5', '_MoSold_6', '_MoSold_7', '_MoSold_8', '_MoS
old_9', '_MoSold_10', '_MoSold_11', '_MoSold_12', '_GarageYrBltBin_NoGarag
e', '_GarageYrBltBin_YearBin2', '_GarageYrBltBin_YearBin3', '_GarageYrBltB
in_YearBin4', '_GarageYrBltBin_YearBin5', '_GarageYrBltBin_YearBin6', '_Ga
rageYrBltBin_YearBin7', '_YearBuiltBin_YearBin1', '_YearBuiltBin_YearBin
2', '_YearBuiltBin_YearBin3', '_YearBuiltBin_YearBin4', '_YearBuiltBin_Yea
rBin5', '_YearBuiltBin_YearBin6', '_YearBuiltBin_YearBin7', '_YearRemodAdd
Bin_YearBin4', '_YearRemodAddBin_YearBin5', '_YearRemodAddBin_YearBin6',
'_YearRemodAddBin_YearBin7', '_NeighborhoodBin_0', '_NeighborhoodBin_1',
'_NeighborhoodBin_2', '_NeighborhoodBin_3', '_NeighborhoodBin_4']
Training set size: (1456, 403)
Test set size: (1459, 403)
```

In [ ]:

```
```

Linear Regression for comparison (Exclusive Cell)

- Kaggle score was around 0.22(approx.)

In [55]:

```
train2=munge(train2)
colsLinReg = ('FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond',
        'ExterQual', 'ExterCond','HeatingQC', 'PoolQC', 'KitchenQual', 'BsmtFinType1',
        'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish',
         'CentralAir', 'MSSubClass', 'OverallCond',
        'YrSold', 'MoSold', 'MSZoning',  'LotConfig', 'Neighborhood',
        'Condition1',  'BldgType', 'HouseStyle', 'RoofStyle', 'Exterior1st',
        'Exterior2nd', 'MasVnrType', 'MasVnrArea', 'Foundation',
        'SaleType', 'SaleCondition')

from sklearn.preprocessing import LabelEncoder
for c in colsLinReg:
    lbl = LabelEncoder()
    lbl.fit(list(train2[c].values))
    train2[c] = lbl.transform(list(train2[c].values))


#Take their values in X and y
X = train2.values
y = label_df.values

# Split data into train and test formate
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7
)

# The error metric: RMSE on the log of the sale prices.
from sklearn.metrics import mean_squared_error
def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))


#Train the model
from sklearn import linear_model
model = linear_model.LinearRegression()

#Fit the model
model.fit(X_train, y_train)

#Prediction
print("Predict value " + str(model.predict([X_test[142]])))
print("Real value " + str(y_test[142]))

#Score/Accuracy
print("Accuracy --> ", model.score(X_test, y_test)*100)

foo=model.predict(X_test)
#Score/Accuracy
print("RMSE --> ", rmse(y_test, foo))
```

```
Predict value [[12.06554692]]
Real value [12.06681633]
Accuracy -->  88.08908165564657
RMSE -->  0.1371500049418401
```

Batch Normalization and shuffled Batch Generator Functions(Exclusive Cell)

In [56]:

```
#change directory
os.chdir("/kaggle/working/")
import tensorflow as tf

def bn_layer(x, scope, is_training, epsilon=0.001, decay=0.99, reuse=None):
    """
    Performs a batch normalization layer

    Args:
        x: input tensor
        scope: scope name
        is_training: python boolean value
        epsilon: the variance epsilon - a small float number to avoid dividing by 0
        decay: the moving average decay

    Returns:
        The ops of a batch normalization layer
    """
    with tf.variable_scope(scope, reuse=reuse):
        shape = x.get_shape().as_list()
        # gamma: a trainable scale factor
        gamma = tf.get_variable("gamma", shape[-1], initializer=tf.constant_initializer
(1.0), trainable=True)
        # beta: a trainable shift value
        beta = tf.get_variable("beta", shape[-1], initializer=tf.constant_initializer(
0.0), trainable=True)
        moving_avg = tf.get_variable("moving_avg", shape[-1], initializer=tf.constant_i
nitializer(0.0), trainable=False)
        moving_var = tf.get_variable("moving_var", shape[-1], initializer=tf.constant_i
nitializer(1.0), trainable=False)
        if is_training:
            # tf.nn.moments == Calculate the mean and the variance of the tensor x
            avg, var = tf.nn.moments(x, np.arange(len(shape)-1), keep_dims=True)
            avg=tf.reshape(avg, [avg.shape.as_list()[-1]])
            var=tf.reshape(var, [var.shape.as_list()[-1]])
            #update_moving_avg = moving_averages.assign_moving_average(moving_avg, avg,
decay)
            update_moving_avg=tf.assign(moving_avg, moving_avg*decay+avg*(1-decay))
            #update_moving_var = moving_averages.assign_moving_average(moving_var, var,
decay)
            update_moving_var=tf.assign(moving_var, moving_var*decay+var*(1-decay))
            control_inputs = [update_moving_avg, update_moving_var]
        else:
            avg = moving_avg
            var = moving_var
            control_inputs = []
        with tf.control_dependencies(control_inputs):
            output = tf.nn.batch_normalization(x, avg, var, offset=beta, scale=gamma, v
ariance_epsilon=epsilon)

    return output


def bn_layer_top(x, scope, is_training, epsilon=0.001, decay=0.99):
    """
    Returns a batch normalization layer that automatically switch between train and tes
t phases based on the
    tensor is_training
```

```
    Args:
        x: input tensor
        scope: scope name
        is_training: boolean tensor or variable
        epsilon: epsilon parameter - see batch_norm_layer
        decay: epsilon parameter - see batch_norm_layer

    Returns:
        The correct batch normalization layer based on the value of is_training
    """
    #assert isinstance(is_training, (ops.Tensor, variables.Variable)) and is_training.d
type == tf.bool

    return tf.cond(
        is_training,
        lambda: bn_layer(x=x, scope=scope, epsilon=epsilon, decay=decay, is_training=Tr
ue, reuse=None),
        lambda: bn_layer(x=x, scope=scope, epsilon=epsilon, decay=decay, is_training=Fa
lse, reuse=True),
    )
```

Note: It is advantegeous to use batch normalization as our TensorFlow NN model uses mini fixed length randomized batches iterated over a fixed number of epochs. Batch normalization helps keep the data standardised over epochs. Code is from Zhongyu Kuang [10] [11].

3. Decision of Algorithm and Model

In [57]:

```
#Lets split the data into train and test set
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(train_df_munged, label_df, test_size=
0.30, random_state=2)

ntrain = X_train.shape[0]
nval=X_val.shape[0]

y_trainRanked= y_train.values
y_valRanked= y_val.values
y_trainRanked=np.reshape(y_trainRanked, (ntrain, 1))
y_valRanked=np.reshape(y_valRanked, (nval, 1))

X_testRanked= test_df_munged.values
test1X=X_testRanked

X_trainRanked= X_train.values
X_valRanked= X_val.values

def next_batch(num, data, labels):
    '''
    Return a total of `num` random samples and labels.
    '''
    idx = np.arange(0 , len(data))
    np.random.shuffle(idx)
    idx = idx[:num]
    data_shuffle = [data[ i] for i in idx]
    labels_shuffle = [labels[ i] for i in idx]

    return np.asarray(data_shuffle), np.asarray(labels_shuffle)

yShape=y_trainRanked.shape
XShape=X_trainRanked.shape
print(XShape)
print(y_trainRanked.shape)
print(y_trainRanked[2])
```

```
(1019, 403)
(1019, 1)
[12.20607765]
```

Cell Report:

1. Train dataset was spit 70-30 in order to do Cross Validation.
2. The *next_batch* function returns a randomly shuffled batch. This was needed as Mini-batch Training was used across all the DNNs.

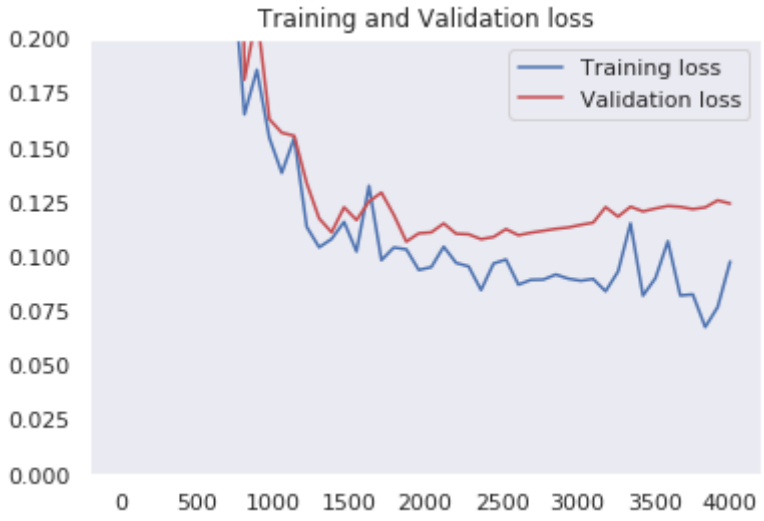
3.1 DNN 1

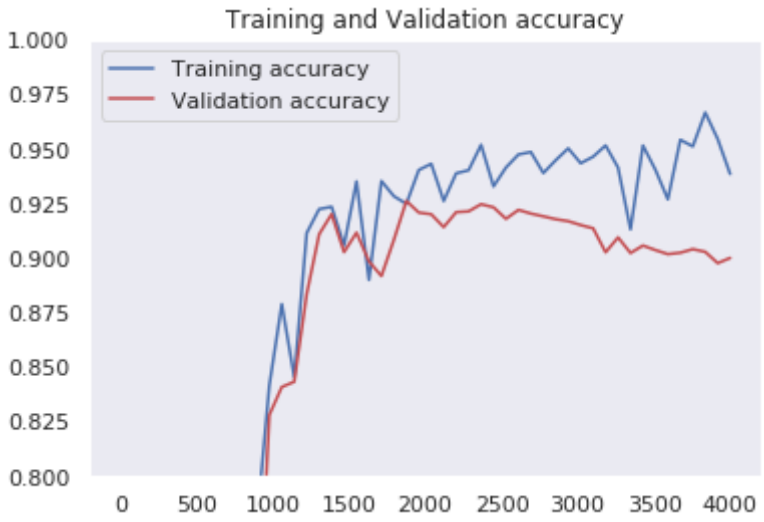
In [ ]:

```
lossArr =  np.array([])
accArr= np.array([])
val_lossArr =  np.array([])
val_accArr= np.array([])

epochArr=  np.array([])
pred_500X300_kp90 = pd.DataFrame()
pred_500X300_kp90['Id']= train_df['Id']



# Python optimisation variables
learning_rate = 0.001
epochs = 4000
batch_size = 80

#Reset TF graph
tf.reset_default_graph()


# placeholders for a tensor that will be always fed.
X = tf.placeholder(dtype=tf.float32, shape = [None, 403])
Y = tf.placeholder(dtype=tf.float32, shape = [None, 1])
keep_prob = tf.placeholder(tf.float32)
is_training = tf.placeholder(tf.bool)

# Training Data
n_samples = y_trainRanked.shape[0]

# # Set model weights

# # the weights connecting the input to the hidden layer


W1 = tf.Variable(tf.ones([403, 500]), name='W1',dtype=tf.float32)
b1 = tf.Variable(tf.zeros([500]), name='b1',dtype=tf.float32)

# and the weights connecting the hidden layer to the output layer
W2 = tf.Variable(tf.ones([500, 300]), name='W2',dtype=tf.float32)
b2 = tf.Variable(tf.zeros([300]), name='b2',dtype=tf.float32)

# and the weights connecting the hidden layer to the output layer
W3 = tf.Variable(tf.ones([300, 1]), name='W3',dtype=tf.float32)
b3 = tf.Variable(tf.zeros([1]), name='b3',dtype=tf.float32)



# calculate the output of the hidden layer
hidden_out = tf.add(tf.matmul(X, W1), b1)
hidden_out = bn_layer_top(hidden_out,'hidden_out_bn', is_training=is_training)
hidden_out = tf.nn.relu(hidden_out)
drop_out = tf.nn.dropout(hidden_out, keep_prob)  # DROP-OUT here

# calculate the output of the hidden layer 2
hidden_out2 = tf.add(tf.matmul(drop_out, W2), b2)
hidden_out2 = bn_layer_top(hidden_out2,'hidden_out_bn2', is_training=is_training)
hidden_out2 = tf.nn.relu(hidden_out2)
drop_out2 = tf.nn.dropout(hidden_out2, keep_prob)  # DROP-OUT here
```

```
#Prediction

pred = tf.add(tf.matmul(drop_out2, W3), b3)




 # Define a loss function
deltas = tf.abs(pred - Y)
loss = tf.reduce_sum(deltas)/batch_size




#  Note, minimize() knows to modify W and b because Variable objects are
 trainable=True by default
optimiser = tf.train.AdamOptimizer(learning_rate).minimize(loss)




# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()

# Start training

# start the session
with tf.Session() as sess:
    # initialise the variables
    sess.run(init)
    total_batch = int(len(X_trainRanked) / batch_size)
    val_total_batch = int(len(X_valRanked) / batch_size)

    for epoch in range(epochs):
        avg_cost = 0
        avg_acc = 0
        avg_val_cost = 0
        avg_val_acc = 0
        for i in range(total_batch):
            batch_x, batch_y = next_batch(num=batch_size,data=X_trainRanked,labels=y_tr
ainRanked)
            _, c = sess.run([optimiser, loss],
                         feed_dict={X: batch_x, Y: batch_y, keep_prob : 0.90, is_traini
ng: True},)
            train_pred=sess.run(pred, feed_dict={X: batch_x,keep_prob : 1.0, is_trainin
g: False})
            c= rmse(batch_y,train_pred)
            a= r2_score(batch_y, train_pred,multioutput='variance_weighted')
            avg_cost += c / total_batch
            avg_acc += a / total_batch


        val_pred=sess.run(pred, feed_dict={X: X_valRanked,keep_prob : 1.0, is_training:
False})
        val_c= rmse(y_valRanked,val_pred)
        val_a= r2_score(y_valRanked, val_pred,multioutput='variance_weighted')
        avg_val_cost = val_c
        avg_val_acc = val_a
        print("Epoch:", (epoch + 1), "cost =", "{:.3f}".format(avg_cost))
```

```
        lossArr = np.append (lossArr, [avg_cost])
        accArr= np.append (accArr, [avg_acc])
        val_lossArr = np.append (val_lossArr, [avg_val_cost])
        val_accArr= np.append (val_accArr, [avg_val_acc])
        epochArr= np.append (epochArr, [epoch])



    #Predict for Evaluation
    pred_500X300_kp90['SalePrice'] = sess.run(pred, feed_dict={X: train_df_munged,keep_
prob : 1.0, is_training: False})



def lcurves(ylim_loss_start=0.000,ylim_loss_end=0.200,ylim_acc_start=0.8000,ylim_acc_en
d=1.0000):
    from scipy.interpolate import make_interp_spline, BSpline

    # Train and validation loss
    xnew = np.linspace(epochArr.min(),epochArr.max(),50) #300 represents number of poin
ts to make between T.min and T.max
    spl = make_interp_spline(epochArr, lossArr, k=3) #BSpline object
    val_spl = make_interp_spline(epochArr, val_lossArr, k=3) #BSpline object
    power_smooth = spl(xnew)
    val_power_smooth = val_spl(xnew)
    plt.ylim(ylim_loss_start,ylim_loss_end)
    plt.plot(xnew,power_smooth, 'b', label='Training loss')
    plt.plot(xnew,val_power_smooth, 'r', label='Validation loss')
    plt.title('Training and Validation loss')
    plt.legend()
    plt.show()
    # Train and validation acc
    spl = make_interp_spline(epochArr, accArr, k=3) #BSpline object
    val_spl = make_interp_spline(epochArr, val_accArr, k=3) #BSpline object
    power_smooth = spl(xnew)
    val_power_smooth = val_spl(xnew)
    plt.ylim(ylim_acc_start,ylim_acc_end)
    plt.plot(xnew,power_smooth, 'b', label='Training accuracy')
    plt.plot(xnew,val_power_smooth, 'r', label='Validation accuracy')
    plt.title('Training and Validation accuracy')
    plt.legend()
    plt.show()

lcurves()
```





Cell Report:

1. This NN consists of Input layer->500 Neurons->300 Neurons->Single Output Neuron.
2. *Keep_prob* of Dropout is set to 0.90
3. A Learning Curve is also generated that will be discussed further on.

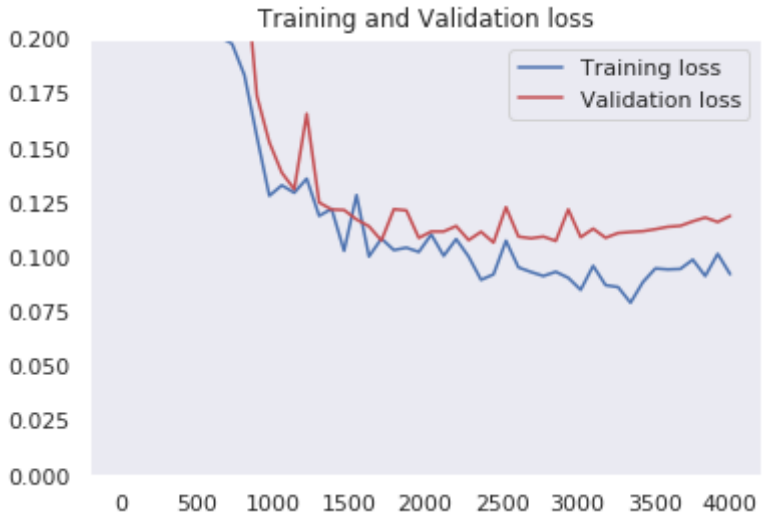
3.2 DNN 2

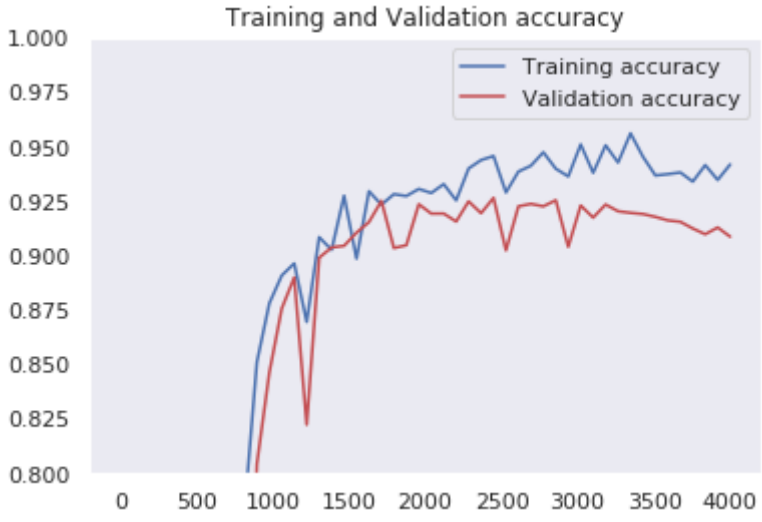
In [ ]:

```
lossArr =  np.array([])
accArr= np.array([])
val_lossArr =  np.array([])
val_accArr= np.array([])

epochArr=  np.array([])
pred_200X200_kp90 = pd.DataFrame()
pred_200X200_kp90['Id']= train_df['Id']



# Python optimisation variables
learning_rate = 0.001
epochs = 4000
batch_size = 80

#Reset TF graph
tf.reset_default_graph()


# placeholders for a tensor that will be always fed.
X = tf.placeholder(dtype=tf.float32, shape = [None, 403])
Y = tf.placeholder(dtype=tf.float32, shape = [None, 1])
keep_prob = tf.placeholder(tf.float32)
is_training = tf.placeholder(tf.bool)

# Training Data
n_samples = y_trainRanked.shape[0]

# # Set model weights

# # the weights connecting the input to the hidden layer


W1 = tf.Variable(tf.ones([403, 200]), name='W1',dtype=tf.float32)
b1 = tf.Variable(tf.zeros([200]), name='b1',dtype=tf.float32)

# and the weights connecting the hidden layer to the output layer
W2 = tf.Variable(tf.ones([200, 200]), name='W2',dtype=tf.float32)
b2 = tf.Variable(tf.zeros([200]), name='b2',dtype=tf.float32)

# and the weights connecting the hidden layer to the output layer
W3 = tf.Variable(tf.ones([200, 1]), name='W3',dtype=tf.float32)
b3 = tf.Variable(tf.zeros([1]), name='b3',dtype=tf.float32)



# calculate the output of the hidden layer
hidden_out = tf.add(tf.matmul(X, W1), b1)
hidden_out = bn_layer_top(hidden_out,'hidden_out_bn', is_training=is_training)
hidden_out = tf.nn.relu(hidden_out)
drop_out = tf.nn.dropout(hidden_out, keep_prob)  # DROP-OUT here

# calculate the output of the hidden layer 2
hidden_out2 = tf.add(tf.matmul(drop_out, W2), b2)
hidden_out2 = bn_layer_top(hidden_out2,'hidden_out_bn2', is_training=is_training)
hidden_out2 = tf.nn.relu(hidden_out2)
drop_out2 = tf.nn.dropout(hidden_out2, keep_prob)  # DROP-OUT here
```

```
#Prediction

pred = tf.add(tf.matmul(drop_out2, W3), b3)




 # Define a loss function
deltas = tf.abs(pred - Y)
loss = tf.reduce_sum(deltas)/batch_size




#  Note, minimize() knows to modify W and b because Variable objects are
 trainable=True by default
optimiser = tf.train.AdamOptimizer(learning_rate).minimize(loss)




# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()

# Start training

# start the session
with tf.Session() as sess:
    # initialise the variables
    sess.run(init)
    total_batch = int(len(X_trainRanked) / batch_size)
    val_total_batch = int(len(X_valRanked) / batch_size)

    for epoch in range(epochs):
        avg_cost = 0
        avg_acc = 0
        avg_val_cost = 0
        avg_val_acc = 0
        for i in range(total_batch):
            batch_x, batch_y = next_batch(num=batch_size,data=X_trainRanked,labels=y_tr
ainRanked)
            _, c = sess.run([optimiser, loss],
                         feed_dict={X: batch_x, Y: batch_y, keep_prob : 0.90, is_traini
ng: True},)
            train_pred=sess.run(pred, feed_dict={X: batch_x,keep_prob : 1.0, is_trainin
g: False})
            c= rmse(batch_y,train_pred)
            a= r2_score(batch_y, train_pred, multioutput='variance_weighted')
            avg_cost += c / total_batch
            avg_acc += a / total_batch


        val_pred=sess.run(pred, feed_dict={X: X_valRanked,keep_prob : 1.0, is_training:
False})
        val_c= rmse(y_valRanked,val_pred)
        val_a= r2_score(y_valRanked, val_pred, multioutput='variance_weighted')
        avg_val_cost = val_c
        avg_val_acc = val_a
        print("Epoch:", (epoch + 1), "cost =", "{:.3f}".format(avg_cost))
```

```
        lossArr = np.append (lossArr, [avg_cost])
        accArr= np.append (accArr, [avg_acc])
        val_lossArr = np.append (val_lossArr, [avg_val_cost])
        val_accArr= np.append (val_accArr, [avg_val_acc])
        epochArr= np.append (epochArr, [epoch])



    #Predict for Evaluation
    pred_200X200_kp90['SalePrice'] = sess.run(pred, feed_dict={X: train_df_munged,keep_
prob : 1.0, is_training: False})




lcurves()
```





Cell Report:

1. This NN consists of Input layer->200 Neurons->200 Neurons->Single Output Neuron.
2. *Keep_prob* of Dropout is set to 0.90
3. A Learning Curve is also generated that will be discussed further on.

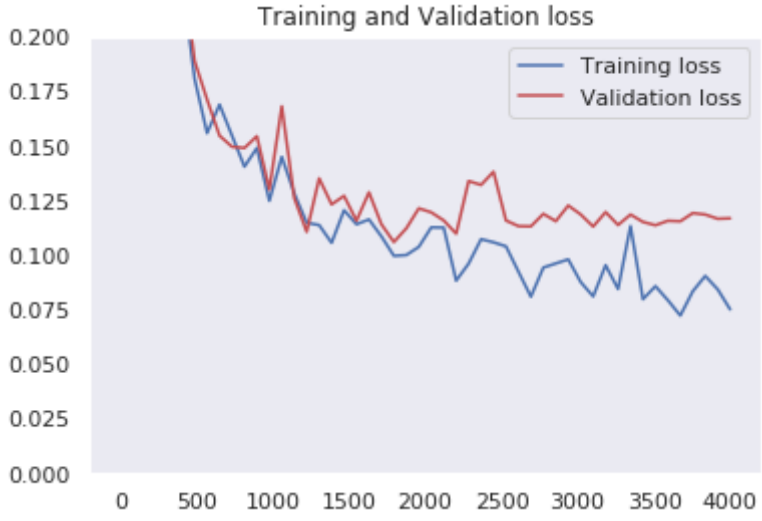
3.3 DNN 3

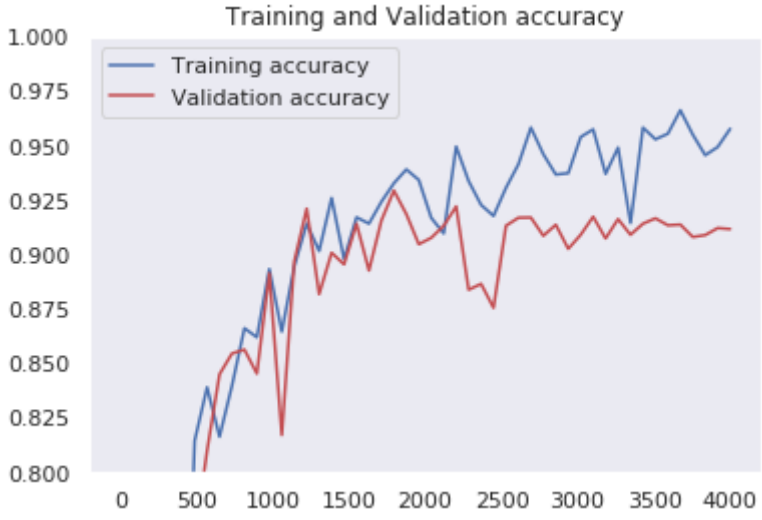
In [ ]:

```
lossArr =  np.array([])
accArr= np.array([])
val_lossArr =  np.array([])
val_accArr= np.array([])

epochArr=  np.array([])
pred_50X50_kp90 = pd.DataFrame()
pred_50X50_kp90['Id']= train_df['Id']



# Python optimisation variables
learning_rate = 0.001
epochs = 4000
batch_size = 80

#Reset TF graph
tf.reset_default_graph()


# placeholders for a tensor that will be always fed.
X = tf.placeholder(dtype=tf.float32, shape = [None, 403])
Y = tf.placeholder(dtype=tf.float32, shape = [None, 1])
keep_prob = tf.placeholder(tf.float32)
is_training = tf.placeholder(tf.bool)

# Training Data
n_samples = y_trainRanked.shape[0]

# # Set model weights

# # the weights connecting the input to the hidden layer


W1 = tf.Variable(tf.ones([403, 50]), name='W1',dtype=tf.float32)
b1 = tf.Variable(tf.zeros([50]), name='b1',dtype=tf.float32)

# and the weights connecting the hidden layer to the output layer
W2 = tf.Variable(tf.ones([50, 50]), name='W2',dtype=tf.float32)
b2 = tf.Variable(tf.zeros([50]), name='b2',dtype=tf.float32)

# and the weights connecting the hidden layer to the output layer
W3 = tf.Variable(tf.ones([50, 1]), name='W3',dtype=tf.float32)
b3 = tf.Variable(tf.zeros([1]), name='b3',dtype=tf.float32)



# calculate the output of the hidden layer
hidden_out = tf.add(tf.matmul(X, W1), b1)
hidden_out = bn_layer_top(hidden_out,'hidden_out_bn', is_training=is_training)
hidden_out = tf.nn.relu(hidden_out)
drop_out = tf.nn.dropout(hidden_out, keep_prob)  # DROP-OUT here

# calculate the output of the hidden layer 2
hidden_out2 = tf.add(tf.matmul(drop_out, W2), b2)
hidden_out2 = bn_layer_top(hidden_out2,'hidden_out_bn2', is_training=is_training)
hidden_out2 = tf.nn.relu(hidden_out2)
drop_out2 = tf.nn.dropout(hidden_out2, keep_prob)  # DROP-OUT here
```

```
#Prediction

pred = tf.add(tf.matmul(drop_out2, W3), b3)




 # Define a loss function
deltas = tf.abs(pred - Y)
loss = tf.reduce_sum(deltas)/batch_size




#  Note, minimize() knows to modify W and b because Variable objects are
 trainable=True by default
optimiser = tf.train.AdamOptimizer(learning_rate).minimize(loss)




# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()

# Start training

# start the session
with tf.Session() as sess:
    # initialise the variables
    sess.run(init)
    total_batch = int(len(X_trainRanked) / batch_size)
    val_total_batch = int(len(X_valRanked) / batch_size)

    for epoch in range(epochs):
        avg_cost = 0
        avg_acc = 0
        avg_val_cost = 0
        avg_val_acc = 0
        for i in range(total_batch):
            batch_x, batch_y = next_batch(num=batch_size,data=X_trainRanked,labels=y_tr
ainRanked)
            _, c = sess.run([optimiser, loss],
                         feed_dict={X: batch_x, Y: batch_y, keep_prob : 0.90, is_traini
ng: True},)
            train_pred=sess.run(pred, feed_dict={X: batch_x,keep_prob : 1.0, is_trainin
g: False})
            c= rmse(batch_y,train_pred)
            a= r2_score(batch_y, train_pred, multioutput='variance_weighted')
            avg_cost += c / total_batch
            avg_acc += a / total_batch


        val_pred=sess.run(pred, feed_dict={X: X_valRanked,keep_prob : 1.0, is_training:
False})
        val_c= rmse(y_valRanked,val_pred)
        val_a= r2_score(y_valRanked, val_pred, multioutput='variance_weighted')
        avg_val_cost = val_c
        avg_val_acc = val_a
        print("Epoch:", (epoch + 1), "cost =", "{:.3f}".format(avg_cost))
```

```
        lossArr = np.append (lossArr, [avg_cost])
        accArr= np.append (accArr, [avg_acc])
        val_lossArr = np.append (val_lossArr, [avg_val_cost])
        val_accArr= np.append (val_accArr, [avg_val_acc])
        epochArr= np.append (epochArr, [epoch])



    #Predict for Evaluation
    pred_50X50_kp90['SalePrice'] = sess.run(pred, feed_dict={X: train_df_munged,keep_pr
ob : 1.0, is_training: False})



lcurves()
```





Cell Report:

1. This NN consists of Input layer->50 Neurons->50 Neurons->Single Output Neuron.
2. *Keep_prob* of Dropout is set to 0.90
3. A Learning Curve is also generated that will be discussed further on.

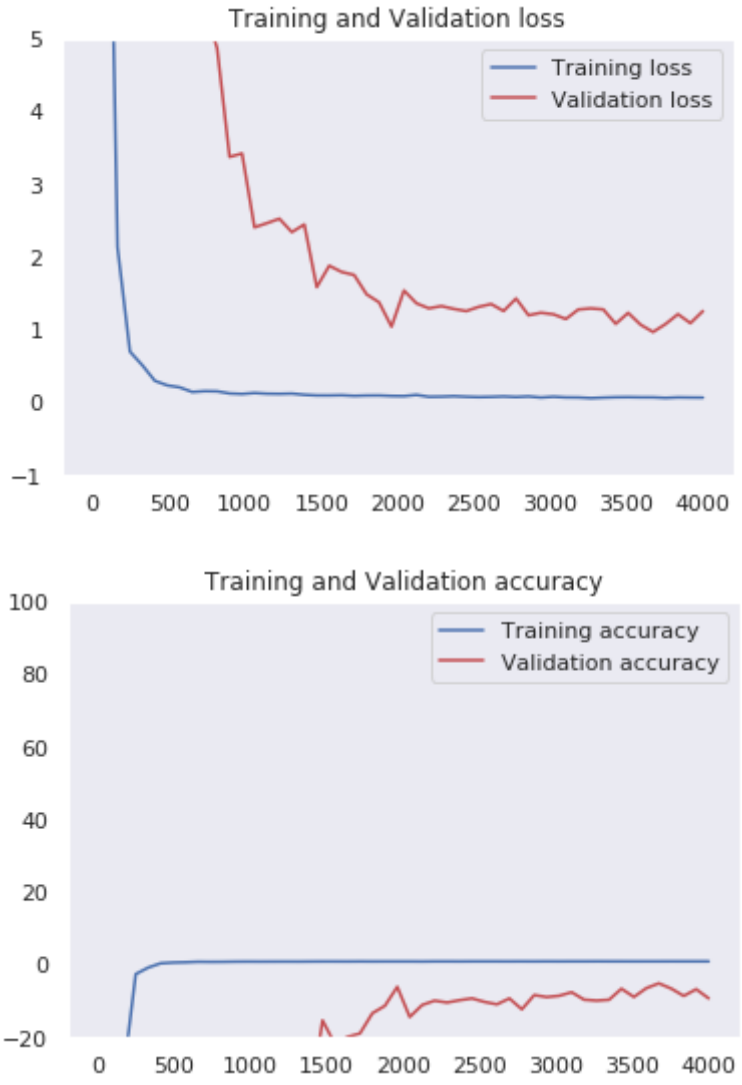
3.4 DNN 4

In [ ]:

```
lossArr =  np.array([])
accArr= np.array([])
val_lossArr =  np.array([])
val_accArr= np.array([])

epochArr=  np.array([])
pred_500_kp90 = pd.DataFrame()
pred_500_kp90['Id']= train_df['Id']



# Python optimisation variables
learning_rate = 0.001
epochs = 4000
batch_size = 80

#Reset TF graph
tf.reset_default_graph()


# placeholders for a tensor that will be always fed.
X = tf.placeholder(dtype=tf.float32, shape = [None, 403])
Y = tf.placeholder(dtype=tf.float32, shape = [None, 1])
keep_prob = tf.placeholder(tf.float32)
is_training = tf.placeholder(tf.bool)

# Training Data
n_samples = y_trainRanked.shape[0]

# # Set model weights

# # the weights connecting the input to the hidden layer


W1 = tf.Variable(tf.ones([403, 500]), name='W1',dtype=tf.float32)
b1 = tf.Variable(tf.zeros([500]), name='b1',dtype=tf.float32)

# and the weights connecting the hidden layer to the output layer
W2 = tf.Variable(tf.ones([500, 1]), name='W2',dtype=tf.float32)
b2 = tf.Variable(tf.zeros([1]), name='b2',dtype=tf.float32)


# calculate the output of the hidden layer
hidden_out = tf.add(tf.matmul(X, W1), b1)
hidden_out = bn_layer_top(hidden_out,'hidden_out_bn', is_training=is_training)
hidden_out = tf.nn.relu(hidden_out)
drop_out = tf.nn.dropout(hidden_out, keep_prob)  # DROP-OUT here



#Prediction

pred = tf.add(tf.matmul(drop_out, W2), b2)




 # Define a loss function
deltas = tf.abs(pred - Y)
```

```
loss = tf.reduce_sum(deltas)/batch_size




#  Note, minimize() knows to modify W and b because Variable objects are
 trainable=True by default
optimiser = tf.train.AdamOptimizer(learning_rate).minimize(loss)




# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()

# Start training

# start the session
with tf.Session() as sess:
    # initialise the variables
    sess.run(init)
    total_batch = int(len(X_trainRanked) / batch_size)
    val_total_batch = int(len(X_valRanked) / batch_size)

    for epoch in range(epochs):
        avg_cost = 0
        avg_acc = 0
        avg_val_cost = 0
        avg_val_acc = 0
        for i in range(total_batch):
            batch_x, batch_y = next_batch(num=batch_size,data=X_trainRanked,labels=y_tr
ainRanked)
            _, c = sess.run([optimiser, loss],
                         feed_dict={X: batch_x, Y: batch_y, keep_prob : 0.90, is_traini
ng: True},)
            train_pred=sess.run(pred, feed_dict={X: batch_x,keep_prob : 1.0, is_trainin
g: False})
            c= rmse(batch_y,train_pred)
            a= r2_score(batch_y, train_pred, multioutput='variance_weighted')
            avg_cost += c / total_batch
            avg_acc += a / total_batch


        val_pred=sess.run(pred, feed_dict={X: X_valRanked,keep_prob : 1.0, is_training:
False})
        val_c= rmse(y_valRanked,val_pred)
        val_a= r2_score(y_valRanked, val_pred, multioutput='variance_weighted')
        avg_val_cost = val_c
        avg_val_acc = val_a
        print("Epoch:", (epoch + 1), "cost =", "{:.3f}".format(avg_cost))
        lossArr = np.append (lossArr, [avg_cost])
        accArr= np.append (accArr, [avg_acc])
        val_lossArr = np.append (val_lossArr, [avg_val_cost])
        val_accArr= np.append (val_accArr, [avg_val_acc])
        epochArr= np.append (epochArr, [epoch])



    #Predict for Evaluation
```

```
    pred_500_kp90['SalePrice'] = sess.run(pred, feed_dict={X: train_df_munged,keep_prob
: 1.0, is_training: False})



lcurves(20,100,-100,100)
```



Cell Report:

1. This shallow NN consists of Input layer->500 Neurons->Single Output Neuron.
2. *Keep_prob* of Dropout is set to 0.90
3. A Learning Curve is also generated that will be discussed further on.

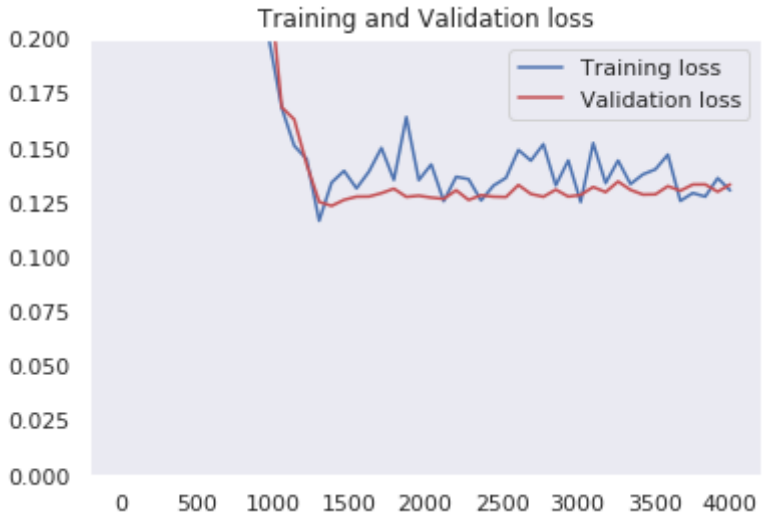
3.5 DNN 5

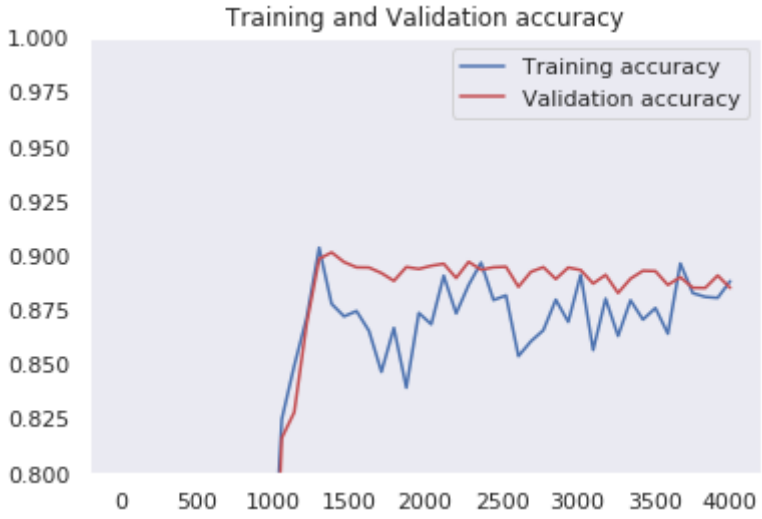
In [ ]:

```
lossArr =  np.array([])
accArr= np.array([])
val_lossArr =  np.array([])
val_accArr= np.array([])

epochArr=  np.array([])
pred_500X300_kp50 = pd.DataFrame()
pred_500X300_kp50['Id']= train_df['Id']



# Python optimisation variables
learning_rate = 0.001
epochs = 4000
batch_size = 80

#Reset TF graph
tf.reset_default_graph()


# placeholders for a tensor that will be always fed.
X = tf.placeholder(dtype=tf.float32, shape = [None, 403])
Y = tf.placeholder(dtype=tf.float32, shape = [None, 1])
keep_prob = tf.placeholder(tf.float32)
is_training = tf.placeholder(tf.bool)

# Training Data
n_samples = y_trainRanked.shape[0]

# # Set model weights

# # the weights connecting the input to the hidden layer


W1 = tf.Variable(tf.ones([403, 500]), name='W1',dtype=tf.float32)
b1 = tf.Variable(tf.zeros([500]), name='b1',dtype=tf.float32)

# and the weights connecting the hidden layer to the output layer
W2 = tf.Variable(tf.ones([500, 300]), name='W2',dtype=tf.float32)
b2 = tf.Variable(tf.zeros([300]), name='b2',dtype=tf.float32)

# and the weights connecting the hidden layer to the output layer
W3 = tf.Variable(tf.ones([300, 1]), name='W3',dtype=tf.float32)
b3 = tf.Variable(tf.zeros([1]), name='b3',dtype=tf.float32)



# calculate the output of the hidden layer
hidden_out = tf.add(tf.matmul(X, W1), b1)
hidden_out = bn_layer_top(hidden_out,'hidden_out_bn', is_training=is_training)
hidden_out = tf.nn.relu(hidden_out)
drop_out = tf.nn.dropout(hidden_out, keep_prob)  # DROP-OUT here

# calculate the output of the hidden layer 2
hidden_out2 = tf.add(tf.matmul(drop_out, W2), b2)
hidden_out2 = bn_layer_top(hidden_out2,'hidden_out_bn2', is_training=is_training)
hidden_out2 = tf.nn.relu(hidden_out2)
drop_out2 = tf.nn.dropout(hidden_out2, keep_prob)  # DROP-OUT here
```

```
#Prediction

pred = tf.add(tf.matmul(drop_out2, W3), b3)




 # Define a loss function
deltas = tf.abs(pred - Y)
loss = tf.reduce_sum(deltas)/batch_size




#  Note, minimize() knows to modify W and b because Variable objects are
 trainable=True by default
optimiser = tf.train.AdamOptimizer(learning_rate).minimize(loss)




# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()

# Start training

# start the session
with tf.Session() as sess:
    # initialise the variables
    sess.run(init)
    total_batch = int(len(X_trainRanked) / batch_size)
    val_total_batch = int(len(X_valRanked) / batch_size)

    for epoch in range(epochs):
        avg_cost = 0
        avg_acc = 0
        avg_val_cost = 0
        avg_val_acc = 0
        for i in range(total_batch):
            batch_x, batch_y = next_batch(num=batch_size,data=X_trainRanked,labels=y_tr
ainRanked)
            _, c = sess.run([optimiser, loss],
                         feed_dict={X: batch_x, Y: batch_y, keep_prob : 0.50, is_traini
ng: True},)
            train_pred=sess.run(pred, feed_dict={X: batch_x,keep_prob : 1.0, is_trainin
g: False})
            c= rmse(batch_y,train_pred)
            a= r2_score(batch_y, train_pred, multioutput='variance_weighted')
            avg_cost += c / total_batch
            avg_acc += a / total_batch


        val_pred=sess.run(pred, feed_dict={X: X_valRanked,keep_prob : 1.0, is_training:
False})
        val_c= rmse(y_valRanked,val_pred)
        val_a= r2_score(y_valRanked, val_pred, multioutput='variance_weighted')
        avg_val_cost = val_c
        avg_val_acc = val_a
        print("Epoch:", (epoch + 1), "cost =", "{:.3f}".format(avg_cost))
```

```
        lossArr = np.append (lossArr, [avg_cost])
        accArr= np.append (accArr, [avg_acc])
        val_lossArr = np.append (val_lossArr, [avg_val_cost])
        val_accArr= np.append (val_accArr, [avg_val_acc])
        epochArr= np.append (epochArr, [epoch])



    #Predict for Evaluation
    pred_500X300_kp50['SalePrice'] = sess.run(pred, feed_dict={X: train_df_munged,keep_
prob : 1.0, is_training: False})




lcurves()
```





Cell Report:

1. This NN consists of Input layer->500 Neurons->300 Neurons->Single Output Neuron.
2. *Keep_prob* of Dropout is set to 0.50
3. A Learning Curve is also generated that will be discussed further on.

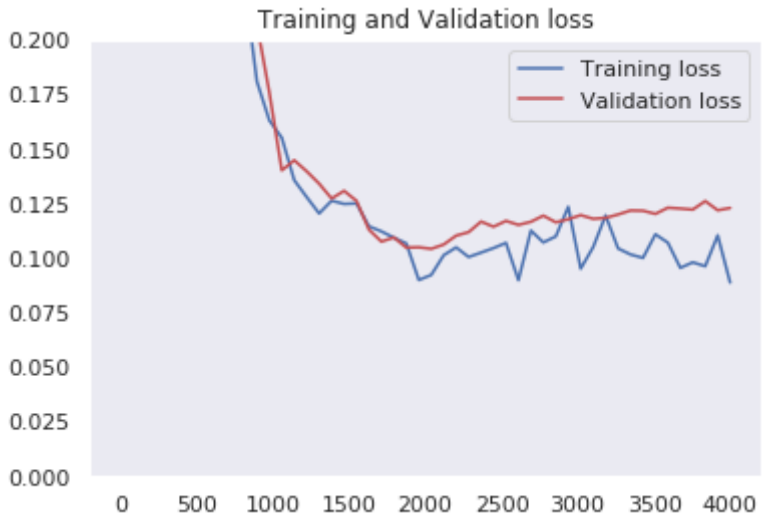
3.6 DNN 6

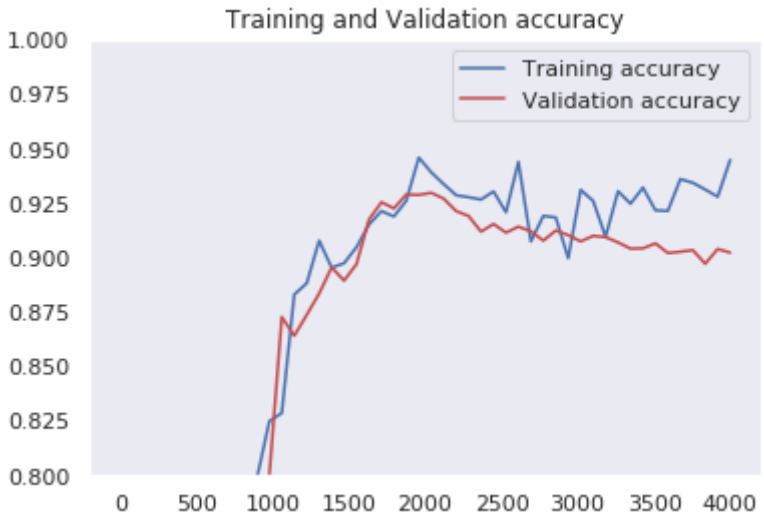
In [ ]:

```
lossArr =  np.array([])
accArr= np.array([])
val_lossArr =  np.array([])
val_accArr= np.array([])

epochArr=  np.array([])
pred_500X300_kp80 = pd.DataFrame()
pred_500X300_kp80['Id']= train_df['Id']



# Python optimisation variables
learning_rate = 0.001
epochs = 4000
batch_size = 80

#Reset TF graph
tf.reset_default_graph()


# placeholders for a tensor that will be always fed.
X = tf.placeholder(dtype=tf.float32, shape = [None, 403])
Y = tf.placeholder(dtype=tf.float32, shape = [None, 1])
keep_prob = tf.placeholder(tf.float32)
is_training = tf.placeholder(tf.bool)

# Training Data
n_samples = y_trainRanked.shape[0]

# # Set model weights

# # the weights connecting the input to the hidden layer


W1 = tf.Variable(tf.ones([403, 500]), name='W1',dtype=tf.float32)
b1 = tf.Variable(tf.zeros([500]), name='b1',dtype=tf.float32)

# and the weights connecting the hidden layer to the output layer
W2 = tf.Variable(tf.ones([500, 300]), name='W2',dtype=tf.float32)
b2 = tf.Variable(tf.zeros([300]), name='b2',dtype=tf.float32)

# and the weights connecting the hidden layer to the output layer
W3 = tf.Variable(tf.ones([300, 1]), name='W3',dtype=tf.float32)
b3 = tf.Variable(tf.zeros([1]), name='b3',dtype=tf.float32)



# calculate the output of the hidden layer
hidden_out = tf.add(tf.matmul(X, W1), b1)
hidden_out = bn_layer_top(hidden_out,'hidden_out_bn', is_training=is_training)
hidden_out = tf.nn.relu(hidden_out)
drop_out = tf.nn.dropout(hidden_out, keep_prob)  # DROP-OUT here

# calculate the output of the hidden layer 2
hidden_out2 = tf.add(tf.matmul(drop_out, W2), b2)
hidden_out2 = bn_layer_top(hidden_out2,'hidden_out_bn2', is_training=is_training)
hidden_out2 = tf.nn.relu(hidden_out2)
drop_out2 = tf.nn.dropout(hidden_out2, keep_prob)  # DROP-OUT here
```

```
#Prediction

pred = tf.add(tf.matmul(drop_out2, W3), b3)




 # Define a loss function
deltas = tf.abs(pred - Y)
loss = tf.reduce_sum(deltas)/batch_size




#  Note, minimize() knows to modify W and b because Variable objects are
 trainable=True by default
optimiser = tf.train.AdamOptimizer(learning_rate).minimize(loss)




# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()

# Start training

# start the session
with tf.Session() as sess:
    # initialise the variables
    sess.run(init)
    total_batch = int(len(X_trainRanked) / batch_size)
    val_total_batch = int(len(X_valRanked) / batch_size)

    for epoch in range(epochs):
        avg_cost = 0
        avg_acc = 0
        avg_val_cost = 0
        avg_val_acc = 0
        for i in range(total_batch):
            batch_x, batch_y = next_batch(num=batch_size,data=X_trainRanked,labels=y_tr
ainRanked)
            _, c = sess.run([optimiser, loss],
                         feed_dict={X: batch_x, Y: batch_y, keep_prob : 0.80, is_traini
ng: True},)
            train_pred=sess.run(pred, feed_dict={X: batch_x,keep_prob : 1.0, is_trainin
g: False})
            c= rmse(batch_y,train_pred)
            a= r2_score(batch_y, train_pred, multioutput='variance_weighted')
            avg_cost += c / total_batch
            avg_acc += a / total_batch


        val_pred=sess.run(pred, feed_dict={X: X_valRanked,keep_prob : 1.0, is_training:
False})
        val_c= rmse(y_valRanked,val_pred)
        val_a= r2_score(y_valRanked, val_pred, multioutput='variance_weighted')
        avg_val_cost = val_c
        avg_val_acc = val_a
        print("Epoch:", (epoch + 1), "cost =", "{:.3f}".format(avg_cost))
```

```
        lossArr = np.append (lossArr, [avg_cost])
        accArr= np.append (accArr, [avg_acc])
        val_lossArr = np.append (val_lossArr, [avg_val_cost])
        val_accArr= np.append (val_accArr, [avg_val_acc])
        epochArr= np.append (epochArr, [epoch])



    #Predict for Evaluation
    pred_500X300_kp80['SalePrice'] = sess.run(pred, feed_dict={X: X_valRanked,keep_prob
: 1.0, is_training: False})



lcurves()
```





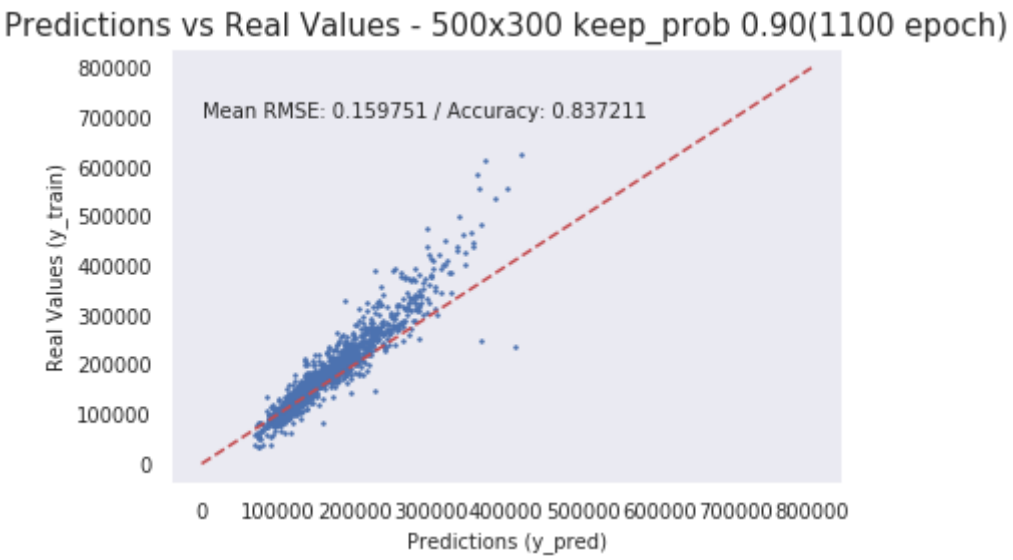
Cell Report:

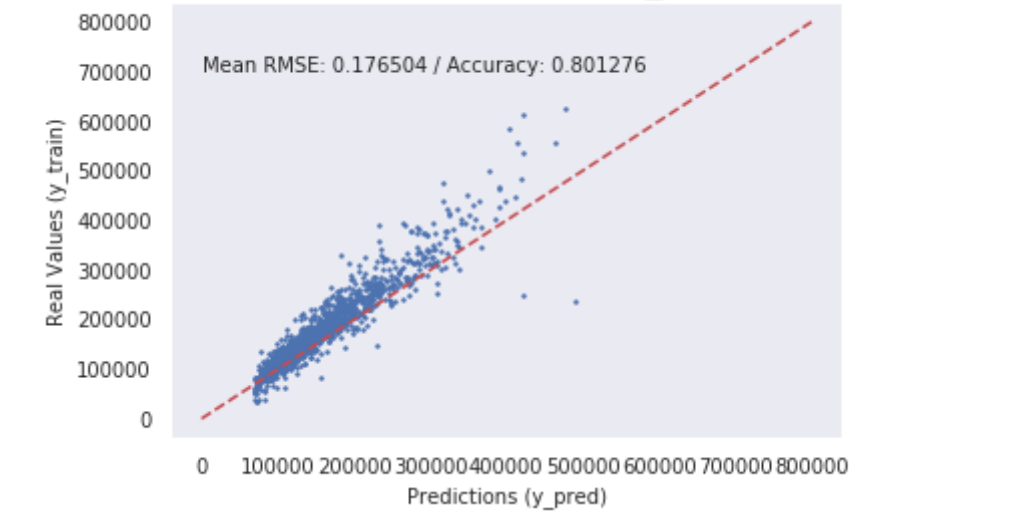
1. This NN consists of Input layer->500 Neurons->300 Neurons->Single Output Neuron.
2. *Keep_prob* of Dropout is set to 0.80
3. A Learning Curve is also generated that will be discussed further on.

3.7 Manual Search Conclusion & DNN Metrics

In [ ]:

```
############
plt.scatter(np.exp(pred_500X300_kp90['SalePrice']), np.exp(label_df["SalePrice"]),s=2)
plt.plot([0, 800000], [0, 800000], '--r')

#Score/Accuracy
print("RMSE --> ", rmse(label_df["SalePrice"],pred_500X300_kp90['SalePrice']))

print("Accuracy --> ", r2_score(label_df["SalePrice"], pred_500X300_kp90['SalePrice'],m
ultioutput='variance_weighted'))
plt.xlabel('Predictions (y_pred)', size=10)
plt.ylabel('Real Values (y_train)', size=10)
plt.tick_params(axis='x', labelsize=10)
plt.tick_params(axis='y', labelsize=10)

plt.title('Predictions vs Real Values - 500x300 keep_prob 0.90(1100 epoch)', size=15)
plt.text(0, 700000, 'Mean RMSE: {:.6f} / Accuracy: {:.6f}'.format(rmse(label_df["SalePr
ice"],pred_500X300_kp90['SalePrice']),
                                                                  r2_score(label_df["SalePri
ce"], pred_500X300_kp90['SalePrice'],multioutput='variance_weighted'), fontsize=15))
plt.show()
############

plt.scatter(np.exp(pred_500X300_kp80['SalePrice']), np.exp(label_df["SalePrice"]),s=2)
plt.plot([0, 800000], [0, 800000], '--r')

#Score/Accuracy
print("RMSE --> ", rmse(label_df["SalePrice"],pred_500X300_kp80['SalePrice']))

print("Accuracy --> ", r2_score(label_df["SalePrice"], pred_500X300_kp80['SalePrice'],m
ultioutput='variance_weighted'))
plt.xlabel('Predictions (y_pred)', size=10)
plt.ylabel('Real Values (y_train)', size=10)
plt.tick_params(axis='x', labelsize=10)
plt.tick_params(axis='y', labelsize=10)

plt.title('Predictions vs Real Values - 500x300 keep_prob 0.80(1100 epoch)', size=15)
plt.text(0, 700000, 'Mean RMSE: {:.6f} / Accuracy: {:.6f}'.format(rmse(label_df["SalePr
ice"],pred_500X300_kp80['SalePrice']),
                                                                  r2_score(label_df["SalePri
ce"], pred_500X300_kp80['SalePrice'],multioutput='variance_weighted'), fontsize=15))

plt.show()
```


Predictions vs Real Values - 500x300 keep_prob 0.90(1100 epoch)
Mean RMSE: 0.159751 / Accuracy: 0.837211
Real Values (y_train)
Predictions (y_pred)
0
100000
200000
300000
400000
500000
600000
700000
800000


Predictions vs Real Values - 500x300 keep_prob 0.80(1100 epoch)
Mean RMSE: 0.176504 / Accuracy: 0.801276
Real Values (y_train)
Predictions (y_pred)
0
100000
200000
300000
400000
500000
600000
700000
800000

Cell Report:

1. Rightaway judging by the Leanring Curves the models with Dropout *keep_prob* 0.5 and 0.9 were shown to be inferior. Models with *keep_prob* 0.5 were grossly underfitting the patterns in the data because of aggressive regularization as can be seen from the loss curves that flattened quickly without improving. Models with *keep_prob* faired better but their validation and train curves diverged away from each other quickly, meaing there were overfiting the patterns in the dataset. A middle point value of 0.8 was choosen for the final model.
2. Models In-500-300-Out and In-200-200-Out performed faily similarly, but the former gave a better Kaggle score. Model In-50-50-Out, which had two hidden layers with 50 neurons each had Learning Curves which flattened fairly quickly indicating the lack of capacity to learn. Model with only a single 500 neuron hidden layer was a bit immune to Dropout rates, and could have performed better with more neurons but the computer cost would have been too much and the same results as other NNs with less neurons with more layers.
3. The Shallow NN with the single hidden layer was not able to fit the data properly, adding more Neurons would fix it but compute cost would rise greatly.
4. Scattter plot shows better score for model with *keep_prob* rate 0.9 as it overfitted the training results.
5. Model In-500-300-Out was choosen as the final model as it gave the best Kaggle score(all of the models were tested with Kaggle Submit).

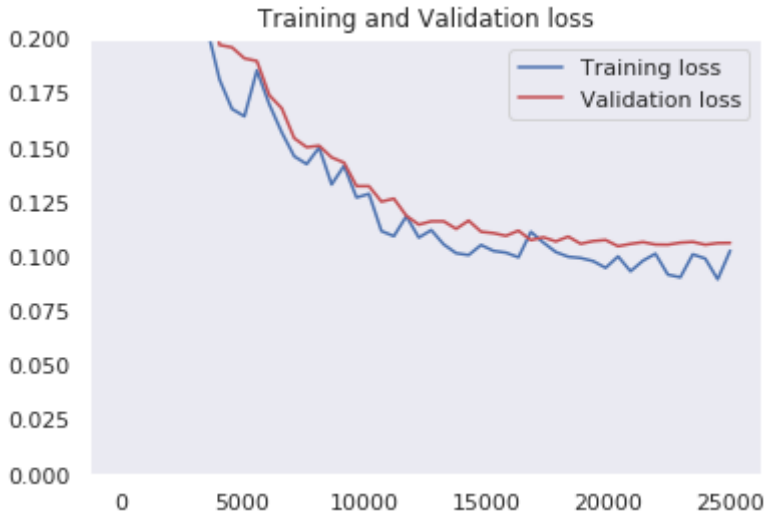
5. Final Model(Used for Kaggle Submission)

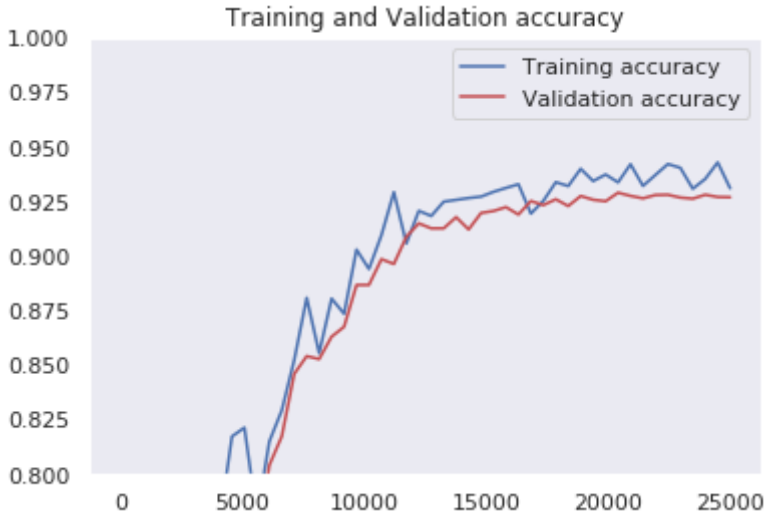
In [ ]:

```
ntrain = train_df_munged.shape[0]

y_trainRanked= label_df.values
y_trainRanked=np.reshape(y_trainRanked, (ntrain, 1))

X_testRanked= test_df_munged.values
test1X=X_testRanked

X_trainRanked= train_df_munged.values
yShape=y_trainRanked.shape
XShape=X_trainRanked.shape
print(XShape)
print(y_trainRanked.shape)
print(y_trainRanked[2])
def SaveResult2CSV(pred4Kaggle, epochNo):
    submission=pd.DataFrame()
    submission['Id']= test2.Id

    #Predict for Kaggle Test.CSV data
    pred_TestKaggle = pred4Kaggle



    #Revert from Log to Exp to get proper House Prices
    final_predictions= np.exp(pred_TestKaggle)
    #Kaggle cloud save
    submission['SalePrice']= final_predictions
    # Brutal approach to deal with predictions close to outer range
    q1 = submission['SalePrice'].quantile(0.0042)
    q2 = submission['SalePrice'].quantile(0.99)

    submission['SalePrice'] = submission['SalePrice'].apply(lambda x: x if x > q1 else
x*0.77)
    submission['SalePrice'] = submission['SalePrice'].apply(lambda x: x if x < q2 else
x*1.1)
    submission.head()
    create_download_link(submission)
    filename = "DNNKagglePredsMay{}.csv".format(epochNo)
    submission.to_csv(filename,index=False)

    print('Saved file: ' + filename)

# Python optimisation variables
learning_rate = 0.0001
epochs = 2
batch_size = 80


#Reset TF graph
tf.reset_default_graph()


# placeholders for a tensor that will be always fed.
X = tf.placeholder(dtype=tf.float32, shape = [None, 403])
Y = tf.placeholder(dtype=tf.float32, shape = [None, 1])
keep_prob = tf.placeholder(tf.float32)
is_training = tf.placeholder(tf.bool)

# Training Data
```

```
n_samples = y_trainRanked.shape[0]

# # Set model weights

# # the weights connecting the input to the hidden layer


W1 = tf.Variable(tf.ones([403, 500]), name='W1',dtype=tf.float32)
b1 = tf.Variable(tf.zeros([500]), name='b1',dtype=tf.float32)

# and the weights connecting the hidden layer to the output layer
W2 = tf.Variable(tf.ones([500, 300]), name='W2',dtype=tf.float32)
b2 = tf.Variable(tf.zeros([300]), name='b2',dtype=tf.float32)

# and the weights connecting the hidden layer to the output layer
W3 = tf.Variable(tf.ones([300, 1]), name='W3',dtype=tf.float32)
b3 = tf.Variable(tf.zeros([1]), name='b3',dtype=tf.float32)



# calculate the output of the hidden layer
hidden_out = tf.add(tf.matmul(X, W1), b1)
hidden_out = bn_layer_top(hidden_out,'hidden_out_bn', is_training=is_training)
hidden_out = tf.nn.relu(hidden_out)
drop_out = tf.nn.dropout(hidden_out, keep_prob)  # DROP-OUT here

# calculate the output of the hidden layer 2
hidden_out2 = tf.add(tf.matmul(drop_out, W2), b2)
hidden_out2 = bn_layer_top(hidden_out2,'hidden_out_bn2', is_training=is_training)
hidden_out2 = tf.nn.relu(hidden_out2)
drop_out2 = tf.nn.dropout(hidden_out2, keep_prob)  # DROP-OUT here

#Prediction

pred = tf.add(tf.matmul(drop_out2, W3), b3)



 # Define a loss function
deltas = tf.abs(pred - Y)
loss = tf.reduce_sum(deltas)/batch_size
# loss=tf.sqrt(loss)

cost = loss




#  Note, minimize() knows to modify W and b because Variable objects are
 trainable=True by default
optimiser = tf.train.AdamOptimizer(learning_rate).minimize(loss)




# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()
```

```
# Start training

# start the session
with tf.Session() as sess:
    # initialise the variables
    sess.run(init)
    total_batch = int(len(X_trainRanked) / batch_size)
    for epoch in range(epochs):
        avg_cost = 0
        for i in range(total_batch):
            batch_x, batch_y = next_batch(num=batch_size,data=X_trainRanked,labels=y_tr
ainRanked)
            _, c = sess.run([optimiser, loss],
                            feed_dict={X: batch_x, Y: batch_y, keep_prob : 0.80, is_traini
ng: True},)
            avg_cost += c / total_batch
        print("Epoch:", (epoch + 1), "cost =", "{:.3f}".format(avg_cost))
        if(epoch==11300 or epoch==14400 or epoch==16500 or epoch==18800):
            pred_TestKaggle = sess.run(pred, feed_dict={X: test1X,keep_prob : 1.0, is_t
raining: False})
            SaveResult2CSV(pred_TestKaggle,epoch)

    submission=pd.DataFrame()
    submission['Id']= test2.Id

    #Predict for Kaggle Test.CSV data
    pred_TestKaggle = sess.run(pred, feed_dict={X: test1X,keep_prob : 1.0, is_training:
False})



    #Revert from Log to Exp to get proper House Prices
    final_predictions= np.exp(pred_TestKaggle)
    #Kaggle cloud save
    submission['SalePrice']= final_predictions
    # Brutal approach to deal with predictions close to outer range
    q1 = submission['SalePrice'].quantile(0.0042)
    q2 = submission['SalePrice'].quantile(0.99)

    submission['SalePrice'] = submission['SalePrice'].apply(lambda x: x if x > q1 else
x*0.77)
    submission['SalePrice'] = submission['SalePrice'].apply(lambda x: x if x < q2 else
x*1.1)
    submission.head()
    create_download_link(submission)
    filename = 'DNNRegPredictionsMaykp80.csv'
    submission.to_csv(filename,index=False)

    print('Saved file: ' + filename)
```


Training and Validation loss
Training loss
Validation loss
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
5000
10000
15000
20000
25000


Training and Validation accuracy
Training accuracy
Validation accuracy
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0
5000
10000
15000
20000
25000

Cell Report:

1. Decision of algorithm and model(1st Submission of Assignment):

- Initially only categorical features were used as input and a low level API TensorFlow fully connected DNN model(80,200,200,1) was made, the Kaggle score was 0.44.
- Then various DNN models were attempted with different combinations of features with different feature engineering, trained using *Geoff Hinton's* rule of thumb.
- Dropout layer was added and improved Kaggle score slightly. Dropout was always set to 0.5.
- Shallow NN with single hidden layer(1000 Neurons) consistently gave best results arcross the board regardless of the level of feature engineering. This was a mistake as it had to do with the dropout rate of 0.5, which was found out later when adjusting hyperparameters.
- For the initial assigment subbmission, a shallow NN of input layer consisting of 126 nodes (one for each NP column), 1 densely connected hidden layer with 1000 nodes per layer was used. The output layer has only single node as this is a regression problem. The architecture of the network is from trial and error. The network was given ample enough size to "**learn**". [2]
- **ReLU** activation was used with the hidden layer, as it is the industry standard, with the dropout set to 0.8 to prevent **overfitting**. Other activation functions were tried but ReLU performed the best.
- **Dropout** was used for **normalization**.
- **Batch Norm** layer was introduced, and improved the score further, made the model more robust against bad initialization.
- The Kaggle score was: **0.129**.

2. For 2nd subbmission after receiving the 1st review:

- 1st subbmission's design decisions were mainly selected with dropout always set to 0.5, this was a big mistake as dropout at this rate was causing **underfitting**.
- Upon further investigation, dropout was set to 0.2(*keep_prob* 0.8) and it was found that shallow NN was not the best, a DNN (400,500,300,1) was selected as a baseline and gave satisfactory results on Kaggle, score **0.116**.

3. For this subbmission:

- ML approch was expanded upon, Learning Curves were added and model was kept the same as the Kaggle score was great.

4. Decision and optimization of hyper parameters:

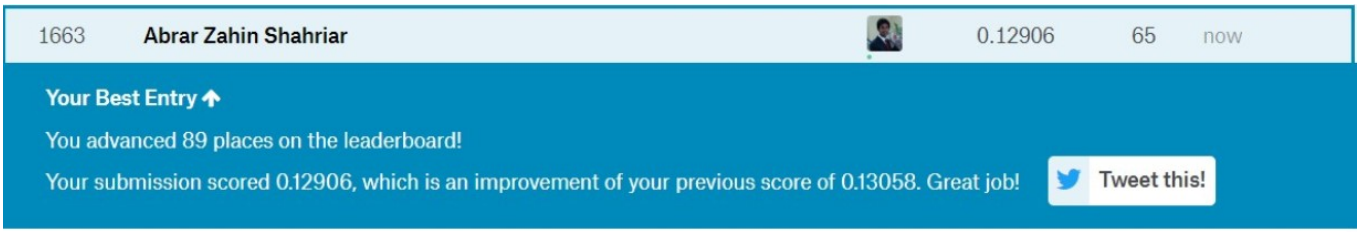
- Hyperpameters were tuned manually and by posting results to Kaggle at each step/change. **Mini-batch learning**(random normalized batch of size 80) was used to train the network, Cross Validation(Test-Train split) was was used and Learning Curve generator added in the style of Keras(avg. *train loss* caluated aross *Steps-Per-Epoch* and *validation loss* calulated at every Epoch end).
- Adam Optimizer was used as it performed the best thought trial and error method(other Optimizrs used were AdaGrad and RMSprop), with the industry standard **learning rate** and parameters. Learning rate for the final model was 0.0001 rather than the standard 0.001 because the higher rate was too jittery(as seen from the Learning Curves). Rate of 0.0001 made it easier for manual early stopping.
- Epochs of 20000 was selected with trail-and-error method based on observation of **Kaggle score** and the Learning Curve.
- Both RMSE and MAE were trialed as loss functions, MAE gave better results with the optimizer.
- Brutal approach from agehsbarg's Kaggle Kernel was to deal with predictions close to outer range as regression often does not deal well with edge cases. Using it resulted in jump of Kaggle score from **0.127** to **0.116**. [12]

5. Final model:

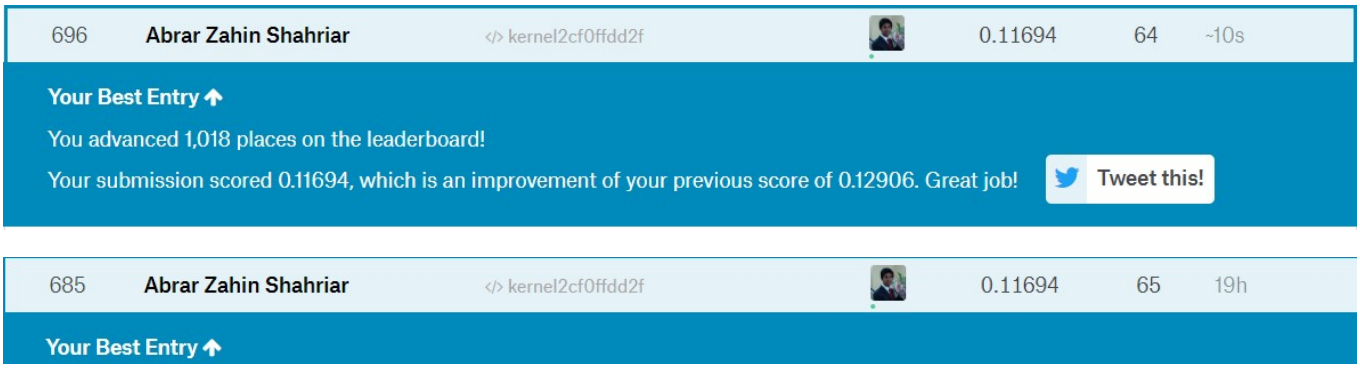
- Input->500 Neurons->300 Neurons->Single Output Neuron.
- Batch Norm at the two hidden layers.
- ReLU activation function at each layer except the final Output Neuron.
- Dropout(*keep_prob*:0.80) at two of the hidden layers.

Results:

1st Submission:



2nd Submission after last review:



Kaggle score of 0.11694 was received with TensorFlow NN (low level API).

Kaggle position: 685 (Top 15.6% at the time of Submission)

Discussion:

- Kaggle Score could have been improved with Grid Search but was abandoned in the interest of time.
- Feature Engineering, Outlier Removal, Skewness Reduction and Hyper Parameter Tuning had great impact on the improvement of the Final Score.
- K-fold CV could have been applied for better Learning Curves as Validation curves depend a lot on the distribution of data.
- More Advanced Regression Techniques like XGBoost would have been fruitful for a more Generic Model.
- Was not about to find a way to do cross validation with batch learning NN.
- TensorFlow Low Level API was difficult to get used to but cleared up many misconceptions that came from only using "blackbox" solutions.

Conclusion:

Neural Network: Neural Network has no real Theory that explains how to choose the number of Hidden Layers except the Thumb Rule and Trial & Error procedure. It takes a lot of time when the Input Data is large, needs powerful computing machines. It is difficult to interpret the results and very hard to interpret and measure the impact of individual predictors.

The Local Minima Issue: The Adam Optimizer produces the optimal weights for the Local Minima, the Global Minima of the Error Function is not guaranteed.

Citations-

1. https://www.kaggle.com/surya635/house-price-prediction (https://www.kaggle.com/surya635/house-price-prediction)
2. https://www.youtube.com/watch?v=UojVVG4PAG0 (https://www.youtube.com/watch?v=UojVVG4PAG0)
3. http://queirozf.com/entries/one-hot-encoding-a-feature-on-a-pandas-dataframe-an-example (http://queirozf.com/entries/one-hot-encoding-a-feature-on-a-pandas-dataframe-an-example)
4. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ (https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/)
5. https://www.dataquest.io/blog/kaggle-getting-started/ (https://www.dataquest.io/blog/kaggle-getting-started/)
6. http://aqibsaeed.github.io/2016-07-07-TensorflowLR/ (http://aqibsaeed.github.io/2016-07-07-TensorflowLR/)
7. https://stackoverflow.com/questions/40994583/how-to-implement-tensorflows-next-batch-for-own-data (https://stackoverflow.com/questions/40994583/how-to-implement-tensorflows-next-batch-for-own-data)
8. https://www.kaggle.com/humananalog/xgboost-lasso (https://www.kaggle.com/humananalog/xgboost-lasso)
9. https://towardsdatascience.com/choosing-the-right-encoding-method-label-vs-onehot-encoder-a4434493149b (https://towardsdatascience.com/choosing-the-right-encoding-method-label-vs-onehot-encoder-a4434493149b)
10. https://medium.com/deeper-learning/glossary-of-deep-learning-batch-normalisation-8266dcd2fa82 (https://medium.com/deeper-learning/glossary-of-deep-learning-batch-normalisation-8266dcd2fa82)
11. Zhongyu Kuang- https://stackoverflow.com/users/4678222/zhongyu-kuang?tab=topactivity (https://stackoverflow.com/users/4678222/zhongyu-kuang?tab=topactivity)
12. https://www.kaggle.com/agehsbarg/top-10-0-10943-stacking-mice-and-brutal-force (https://www.kaggle.com/agehsbarg/top-10-0-10943-stacking-mice-and-brutal-force)