House Prices: Advanced Regression Techniques

Anindya Sanyal

TABLE OF CONTENT:

Abstract:

When a buyer buys his dream house, his price negotiation depends on various factors. This dataset has almost 79 explanatory variables that depends on how a customer will determine the house prices. We have to predict the price of 1460 houses given in the dataset. The dataset is divided into two parts where the training dataset contains the features and prices and the test dataset only haver the features but no pricing. We have to find the prices for the houses of the test dataset. We have to properly prepare the data and then feed the data to a regression algorithom or a deep neural network algorithm. In this report, The data preprocessing, training the regression algorithm and deep neural network model, predictions using the models are described step by step with source code.

Approach Summary:

Our goal is to determine the final house prices in our test dataset. We are given a train dataset that has 79 variables and corresponding house prices. But in the test dataset we are only given the 79 variables and we have to determine the house prices. The approach I will take is given in summary below:

- First of all we visualize our data to get a better understanding of our dataset. We will visualize our target variable 'SalePrice' and other Numerical and Catagorical Features.
- Then we will do correlation between the features and our target variable 'SalePrice' to understand which feature is more important and plays important role in case of determining the house prices.
- We will visualize the mostly correlated variables and their relations with the target variable 'SalePrice'.
- Then we will check the skewness of the data and log transform the data to remove skewness.
- Up next we will fill up the missing data according to the guideline of the dataset. Also we will drop the data that has a greater number of missing values.
- Then we will try six different regression algorithm to predict the house prices and will compare between them. The six regression algorithms used are:

```
* LASSO
* Elastic Net
* Kernel Ridge Regressor
* Gradient Boost
* XGBoost
* LightBGM
```

- Then we will take an average of this six regression algorithm and determine the house prices. It is just plain average of the obtained result from the six models. This model is known as averaged model and gives higher score than all the six models.
- We will use a stacked regression algorithm up next which will have three models average and a meta-model. This model also improves the score but the averaged model still scores the best.
- Then we will take an ensemble approach which will take four models except XGBoost and LightGBM and average them and will multiply two constants from the XGBoost and LightGBM and finally determine the result. This method also significantly improves score but the averaged model still scores the best.
- Then we will move forward to a Deep Neural Network approach where we will create our own model using Tensorflow low level API. We will feed the model from our preprocessed dataset and get the predicted result from it. The model needed Hyperparameter tuning which is also explained in detail with source code.
- All the regression models, averaged model, stack model, ensemble model and our own deep neural network model are cross-validated to check the performance parameter.
- As all the steps we found that averaged model performs best, so finally we made an ultimate model where we averaged the six regression models, average model of the six regression models, stacked model, ensemble model and our deep neural network model in total of ten models. And this ultimate model gives the best score in kaggle so this is out final submission.

Dataset:

The dataset is collected from the Kaggle competition page and the dataset has two CSv files named train and test. The train dataset has the features and prices but the test dataset only has the features but not the prices. The dataset has both Numerical and Catagorical features alongside it has outliers and missing values. A glimpse of the training data and testing data is given below (10 Samples):

In [86]:
```
import pandas as pd
train = pd.read_csv('../input/house-prices-advanced-regression-tech
niques/train.csv')
test = pd.read_csv('../input/house-prices-advanced-regression-techn
iques/test.csv')
```

In [87]:
```
train.head(10)
```

Out[87]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContou
0	1	60	RL	65.0	8450	Pave	NaN	Reg	L
1	2	20	RL	80.0	9600	Pave	NaN	Reg	L
2	3	60	RL	68.0	11250	Pave	NaN	IR1	L
3	4	70	RL	60.0	9550	Pave	NaN	IR1	L
4	5	60	RL	84.0	14260	Pave	NaN	IR1	L
5	6	50	RL	85.0	14115	Pave	NaN	IR1	L
6	7	20	RL	75.0	10084	Pave	NaN	Reg	L
7	8	60	RL	NaN	10382	Pave	NaN	IR1	L
8	9	50	RM	51.0	6120	Pave	NaN	Reg	L
9	10	190	RL	50.0	7420	Pave	NaN	Reg	L

10 rows × 81 columns

In [88]:
```
test.head(10)
```

Out[88]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandCon
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	
5	1466	60	RL	75.0	10000	Pave	NaN	IR1	
6	1467	20	RL	NaN	7980	Pave	NaN	IR1	
7	1468	60	RL	63.0	8402	Pave	NaN	IR1	
8	1469	20	RL	85.0	10176	Pave	NaN	Reg	
9	1470	20	RL	70.0	8400	Pave	NaN	Reg	

10 rows × 80 columns

Data Preprocessing and Feature Engineering:

Dataset preprocessing is an important part when it comes to feeding it to a regression algorithm or to a deep learning model. The dataset given from kaggle had several problems with it. It has many missing values, outliers, distortion from the normal distribution. These features are nedded to be dropped of handled manually to get appropriate result. In this process I took several approaches to deal with these data. For an example dropping highly correlated variables and observing the performance of regression algorithms or keeping them and filling the missing values.

Firstly we have to observe how many values are missing in the dataset and what is their impact on the final decision making. There are some columns which has almost 90% of missing value! In the dataset the target variable 'SalePrice' is distorted from normal distribution which has a severe impact on decision making.

There are some features that are highly correlated with the target variable 'SalePrice' and the correlation value is greater than 0.50 which indicates that these features have greater significance in the final predicted target variable 'SalePrice'. I have dropped the missing columns from these features as filling the missing data from them will make the result biased.

In case of missing values, the columns that has greater percentage of missing values were dropped as filling them up may cause a significant biased decision. Also missing featires were filled according to observation. Some features are filled with 1 or 0 and some features were filled by taking average or mean value. In case of catagorical features, they can not be filled up bu using numbers rather they are label encoded.

Skewness is a impact factor in decision making. The skewness value of this dataset is greater than 1 which indicates the data is positively skewed. I did log transform to correct the skewness. Also for the numeric features, which has a skewness of greater than 0.50 was also log transformed.

The whole process with source code is given below:

First of all lets import all the modules required and load the dataset

In [89]:
```
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
from matplotlib.legend_handler import HandlerBase
import seaborn as sns
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.preprocessing import LabelEncoder
from scipy import stats
from scipy.stats import norm, skew
%matplotlib inline
style.use('dark_background')
sns.set(style='darkgrid',color_codes=True)
```

Train dataset load and checking its columns

In [90]:
```
df_train = pd.read_csv('../input/house-prices-advanced-regression-t
echniques/train.csv')
df_train.columns
```

Out[90]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', '
Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig
',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'B
ldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'Y
earRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'Mas
VnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'Bsmt
Qual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '
Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFl
rSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath'
, 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu',
'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea',
'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'P
oolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'Sal
eType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [91]:
```
print(df_train.shape)
```

```
(1460, 81)
```

Test dataset load and checking its columns

In [92]:
```
df_test = pd.read_csv('../input/house-prices-advanced-regression-te
chniques/test.csv')
df_test.columns
```

Out[92]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', '
Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig
',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'B
ldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'Y
earRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'Mas
VnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'Bsmt
Qual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '
Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFl
rSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath'
, 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu',
'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea',
'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'P
oolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'Sal
eType',
       'SaleCondition'],
      dtype='object')
```

In [93]:
```
print(df_test.shape)
```

```
(1459, 80)
```

Our target variable is SalePrice. So, lets visualize the SalePrice variable

In [94]:
```
df_train['SalePrice'].describe()
```

Out[94]:
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

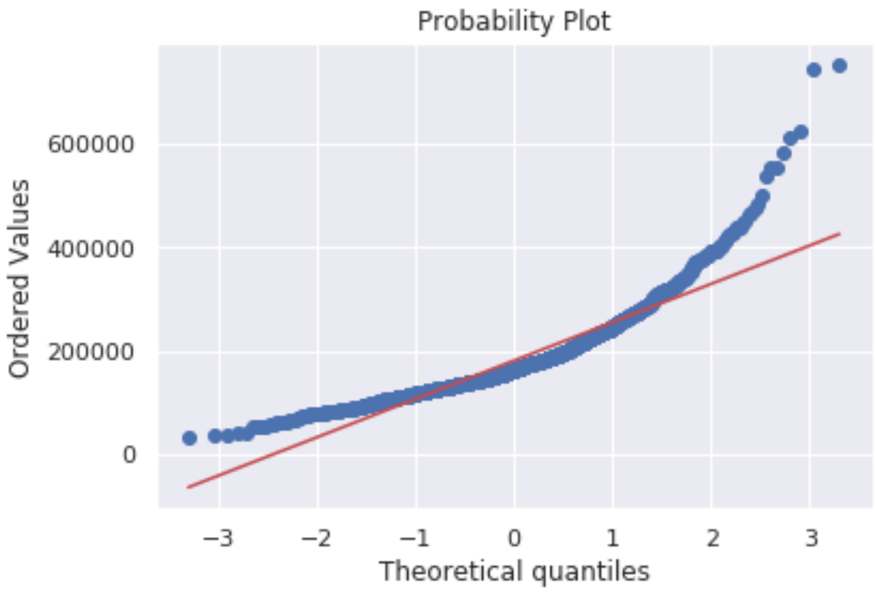
From here we can see the statistical analysis of our target variable 'SalePrice'. We can see the minimum sale price of house is 34900 and maximum is 755000. Also we can see other statistical features such as standard deviation, mean value etc for our target variable 'SalePrice' from the above column.

Lets check the histogram for our target variable 'SalePrice'

In [95]:
```
sns.distplot(df_train['SalePrice'],color='red').set_title('Sale Pri
ce');
```

Now lets draw the probability plot.

In [96]:
```
fig = plt.figure()
res = stats.probplot(train['SalePrice'], plot=plt)
plt.show()
```



From this histogram we can get a brief idea about our target variable 'SalePrice'. We can see that it has deviation from normal distribution, has showed peakedness and has positive skewness.

Lets check the value of Skewness and Kurtosis value

In [97]:
```
print("Skewness: %f" % df_train['SalePrice'].skew())
print("Kurtosis: %f" % df_train['SalePrice'].kurt())
```

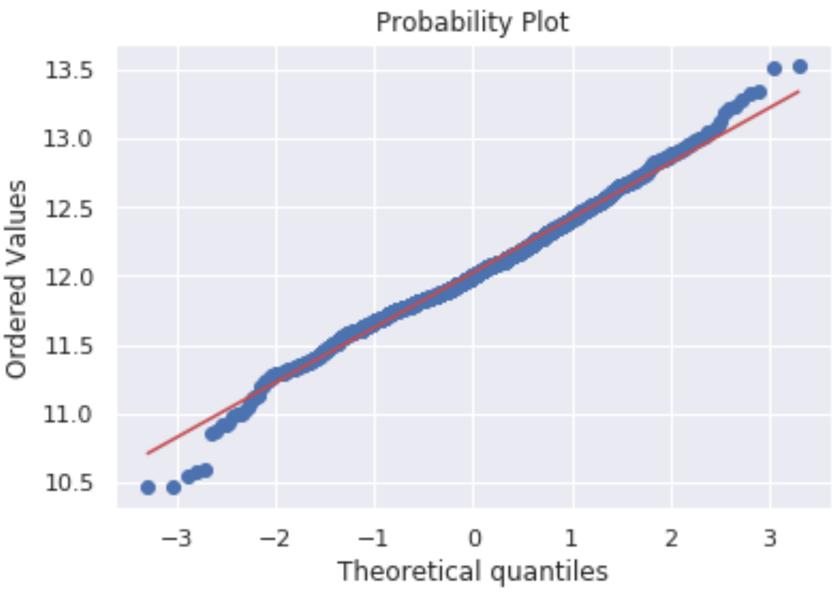
```
Skewness: 1.882876
Kurtosis: 6.536282
```

So, It has a skewness of 1.88 which is way greater than 1 hence it is positively skewed. We need to do log transform to reduce the skewness.

Lets log-transform the target variable

In [98]:
```
train["SalePrice"] = np.log1p(train["SalePrice"])

sns.distplot(train['SalePrice'] , fit=norm, color='red');

(mu, sigma) = norm.fit(train['SalePrice'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))

plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.fo
rmat(mu, sigma)],
            loc='best')
plt.ylabel('Frequency')
plt.title('SalePrice distribution')

fig = plt.figure()
res = stats.probplot(train['SalePrice'], plot=plt)
plt.show()
```

```
mu = 12.02 and sigma = 0.40
```



Now the probability plot looks much finer.

Now lets have a look at the Numerical features that are related to decision making for the target variable 'SalePrice'. Acording to my observation and knowledge four variables are really impactfull to our target variable 'SalePrice'. Among those, two are numerical and two are catagorical. They are:

Numerical Features:

1. GrLivArea
2. TotalBsmtSF

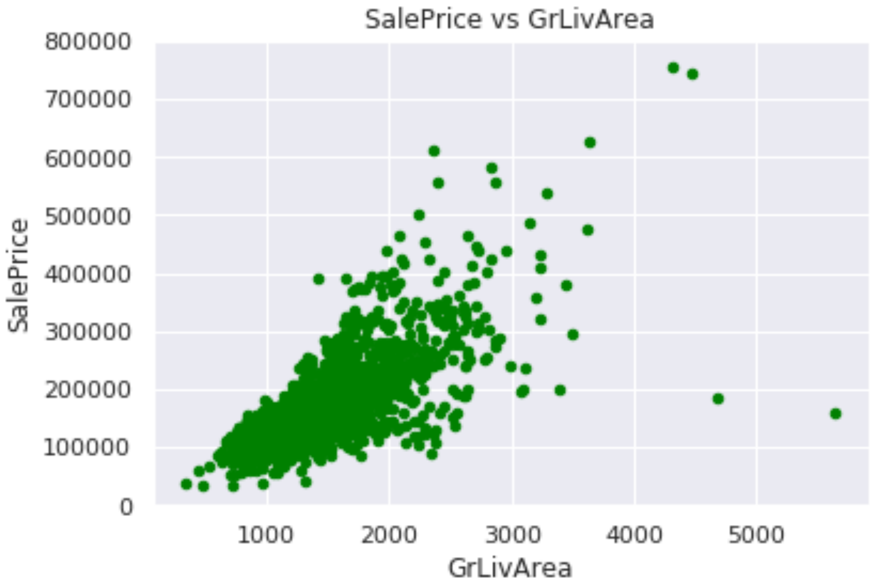
Catagorical Features:

1. OverallQual
2. YearBuilt

Lets check scatter plot for numerical features:

Relation of 'SalePrice' with 'GrLivArea':

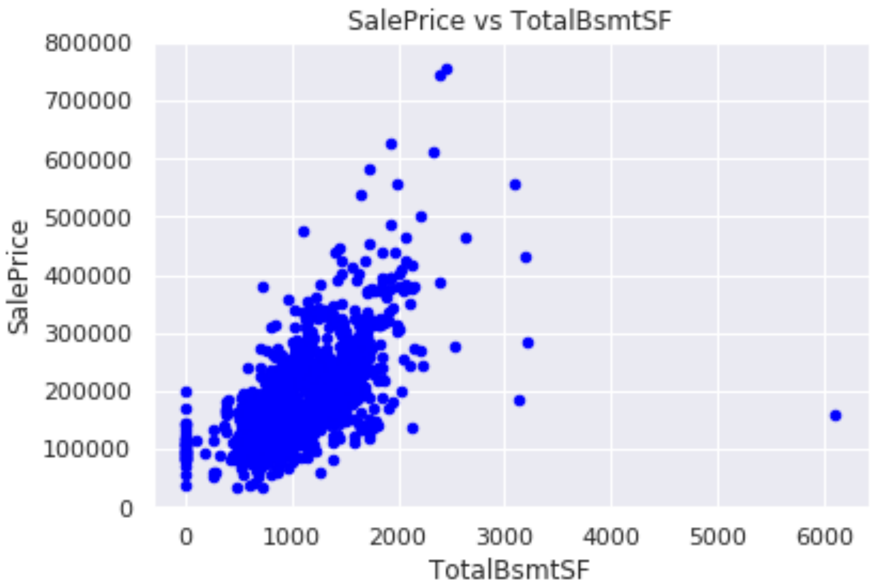
In [99]:
```
var = 'GrLivArea'
data = pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
data.plot.scatter(x=var, y='SalePrice', ylim=(0,800000), color='green').set_title('SalePrice vs GrLivArea');
```



We can see that 'SalePrice' and 'GrLivArea' have linear relationship. That indicates the greater the 'GrLivArea', the greater the 'SalePrice'!

Relation of 'SalePrice' with 'TotalBsmtSF':

In [100]:
```
var = 'TotalBsmtSF'
data = pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
data.plot.scatter(x=var, y='SalePrice', ylim=(0,800000), color='blu
e').set_title('SalePrice vs TotalBsmtSF');
```

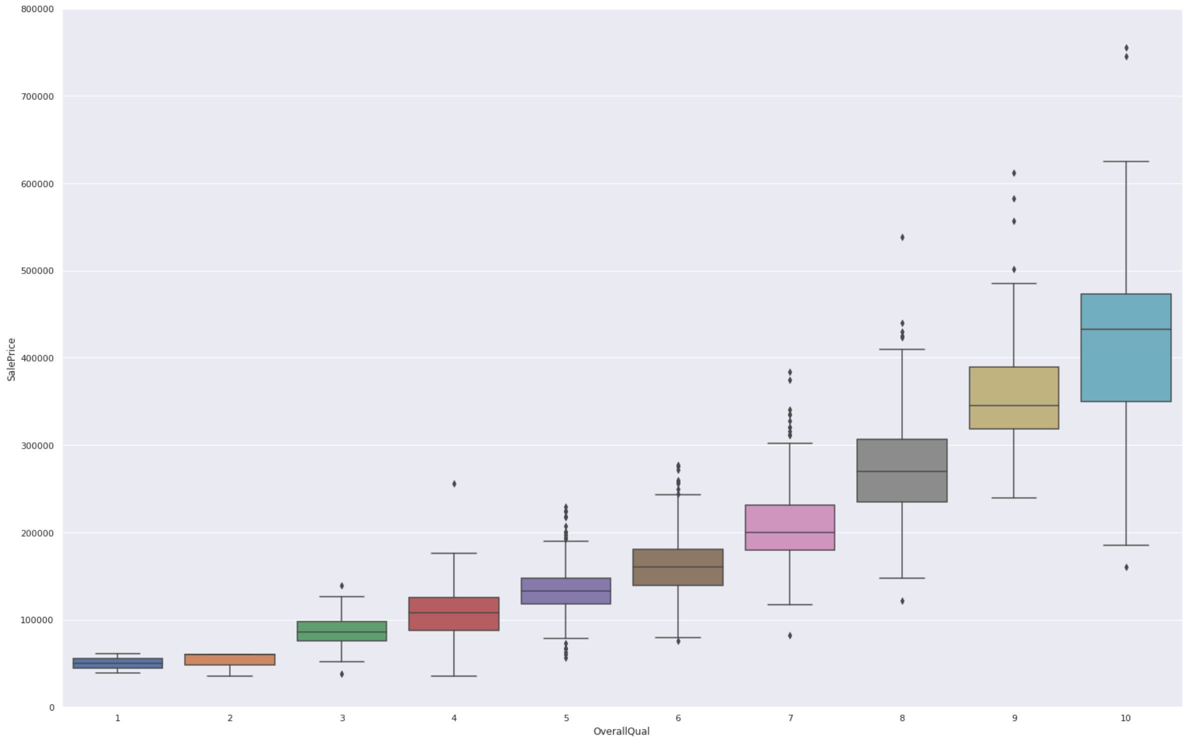


We can see that 'SalePrice' and 'TotalBsmtSF' has no known relationship pattern. Sometimes it is linear or sometimes it seems it is expotential. Or sometimes it has no significance on the target variable 'SalePrice'.

Lets check the relationship of catagorical features:

Firstly lets check 'Overallqual'

In [101]:
```
var = 'OverallQual'
data = pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
f, ax = plt.subplots(figsize=(25, 16))
fig = sns.boxplot(x=var, y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
```

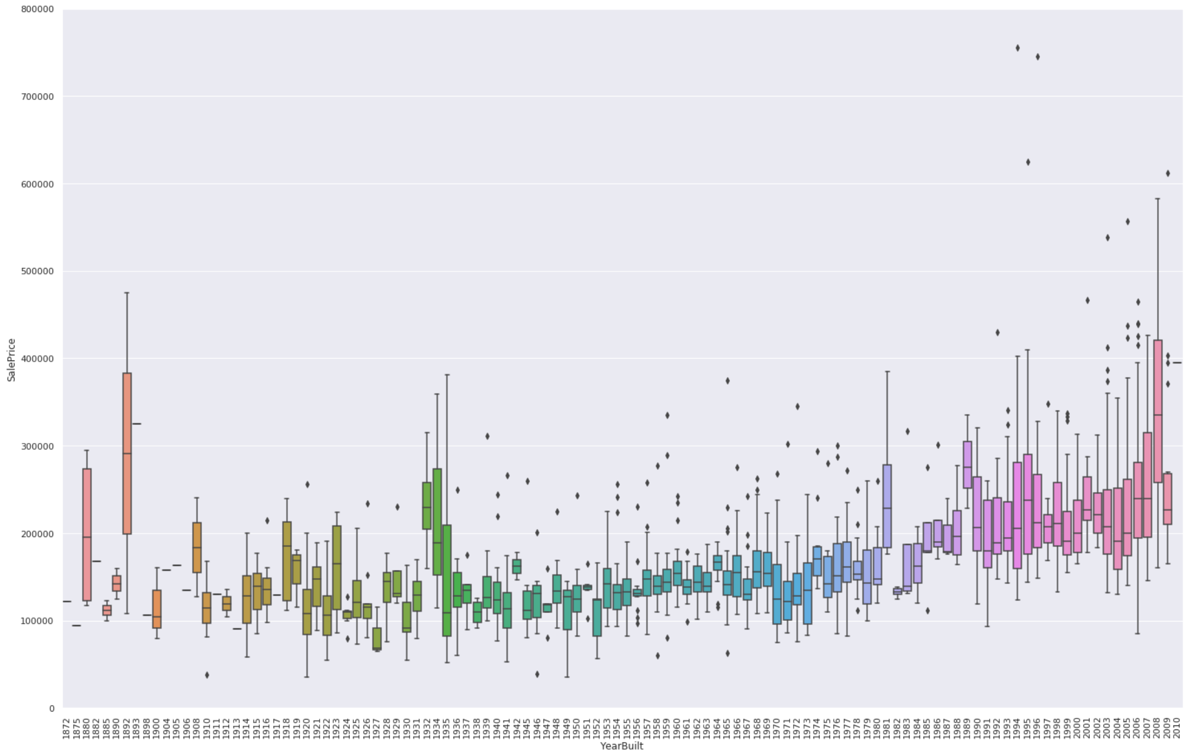


In [102]:
```
saleprice_overall_quality= train.pivot_table(index ='OverallQual',v
alues = 'SalePrice', aggfunc = np.median)
saleprice_overall_quality.plot(kind = 'bar',color = 'pink')
plt.xlabel('Overall Quality')
plt.ylabel('Median Sale Price')
plt.show()
```

So we can see that 'SalePrice' has significant dependancy on 'OverallQual'. The more the overall quality, the greater the house prices!

Now lets check 'YearBuilt'

In [103]:
```
var = 'YearBuilt'
data = pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
f, ax = plt.subplots(figsize=(25, 16))
fig = sns.boxplot(x=var, y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
plt.xticks(rotation=90);
```



The above cell indicates that the newer the house is, the greater the house price! It's usual as people prefer newer and modern houses.

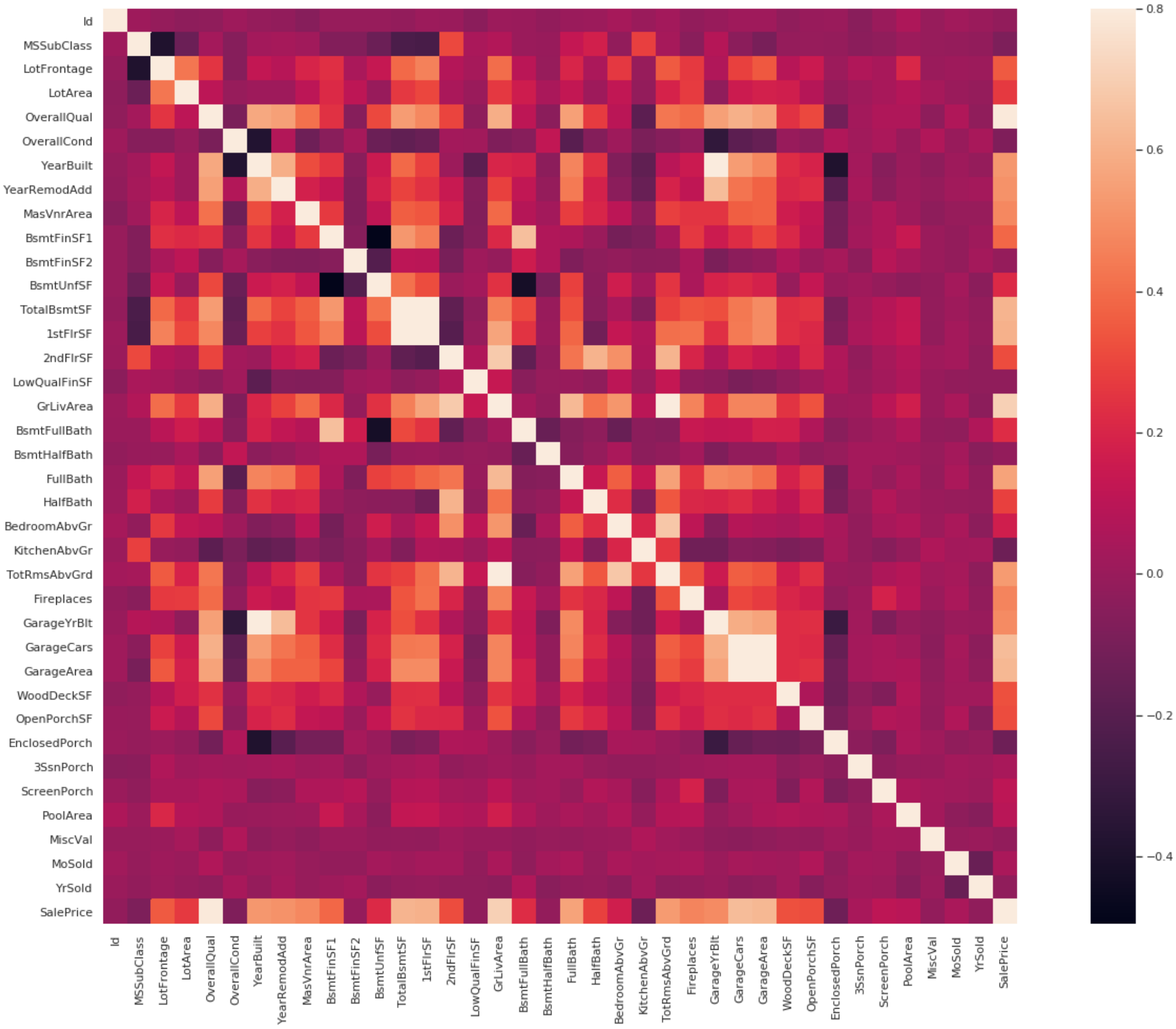
The house prices might have increase due to inflasion as inflasion plays a greater part in case of any pricing.

In summary:

1. 'GrLivArea' and 'TotalBsmtSF' seem to be linearly related with 'SalePrice'. Both relationships are positive, which means that as one variable increases, the other also increases. In the case of 'TotalBsmtSF', we can see that the slope of the linear relationship is particularly high.
2. 'OverallQual' and 'YearBuilt' also seem to be related with 'SalePrice'. The relationship seems to be stronger in the case of 'OverallQual', where the box plot shows how sales prices increase with the overall quality.

As of now I have only explored four variables. But there are more variables that seems to be highly correlated with the target variable 'SalePrice'. To check the correlation between other variables and target variables, we have to check the correlation heatmap between all variables first.

In [104]:
```python
corrmat = df_train.corr()
f, ax = plt.subplots(figsize=(25, 16))
sns.heatmap(corrmat, vmax=.8, square=True);
```



Now lets check the 'SalePrice' correlation matrix

In [105]:
```
k = 10
cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index
cm = np.corrcoef(df_train[cols].values.T)
f, ax = plt.subplots(figsize=(25, 16))
sns.set(font_scale=5)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yticklabels=cols.values, xticklabels=cols.values)
plt.show()
```

In Summary:

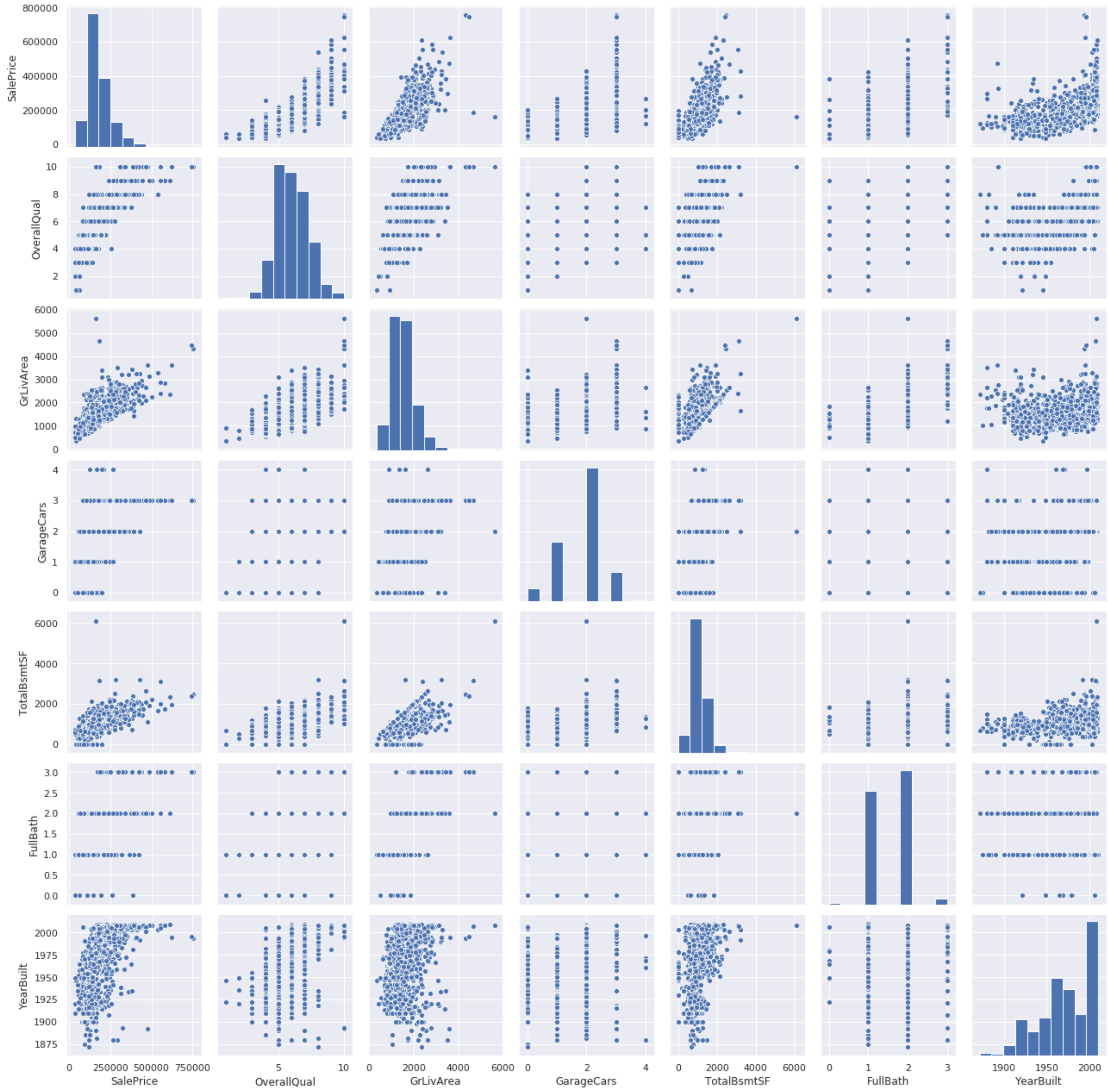
These are the variables most correlated with 'SalePrice'. My thoughts on this:

1. 'OverallQual', 'GrLivArea' and 'TotalBsmtSF' are strongly correlated with 'SalePrice'.
2. 'GarageCars' and 'GarageArea' are also some of the most strongly correlated variables. However, the number of cars that fit into the garage is a consequence of the garage area. 'GarageCars' and 'GarageArea' are almost identical. You'll never be able to distinguish them. Therefore, we just need one of these variables in our analysis so we can keep 'GarageCars' since it's correlation with 'SalePrice' is higher.
3. 'TotalBsmtSF' and '1stFloor' also seem to be identical variables. We can keep 'TotalBsmtSF'.
4. 'TotRmsAbvGrd' and 'GrLivArea', are almost identical variables.

Let's proceed to the pair plots.

Visualisation of 'SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'FullBath', 'YearBuilt' features with respect to SalePrice in the form of pair plot & scatter pair plot for better understanding.

In [106]:
```
sns.set()
cols = ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'Tot
alBsmtSF', 'FullBath', 'YearBuilt']
sns.pairplot(df_train[cols], size = 2.5, palette='grey')
plt.show();
```



Observation:

Although we already know some of the main figures, this pair plot gives us a reasonable overview insight about the correlated features. Here are some of my analysis:

1. This mega scatter plot gives us a reasonable idea about variables relationships.
2. One interesting observation is between 'SalePrice' and 'YearBuilt'. In the bottom of the 'dots cloud', we see what almost appears to be a exponential function.We can also see this same tendency in the upper limit of the 'dots cloud'
3. One more interesting observation is between 'TotalBsmtSF' and 'GrLiveArea'. In this figure we can see the dots drawing a linear line, which almost acts like a border. It totally makes sense that the majority of the dots stay below that line. Basement areas can be equal to the above ground living area, but it is not expected a basement area bigger than the above ground living area.
4. Last observation is that prices are increasing faster now with respect to previous years.

Missing Data Analysis:

Now lets check the missing data in the columns in the given train and test dataset. We have to fill them up with different strategies in order to get the perfect data to feed the models.

In [107]:
```
ntrain = train.shape[0]
ntest = test.shape[0]
y_train = train.SalePrice.values
all_data = pd.concat((train, test)).reset_index(drop=True)
all_data.drop(['SalePrice'], axis=1, inplace=True)
print("all_data size is : {}".format(all_data.shape))
```

```
all_data size is : (2919, 80)
```

Missing data with ascending missing ratio:

In [108]:
```
all_data_na = (all_data.isnull().sum() / len(all_data)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index)
.sort_values(ascending=False)[:30]
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data.head(20)
```

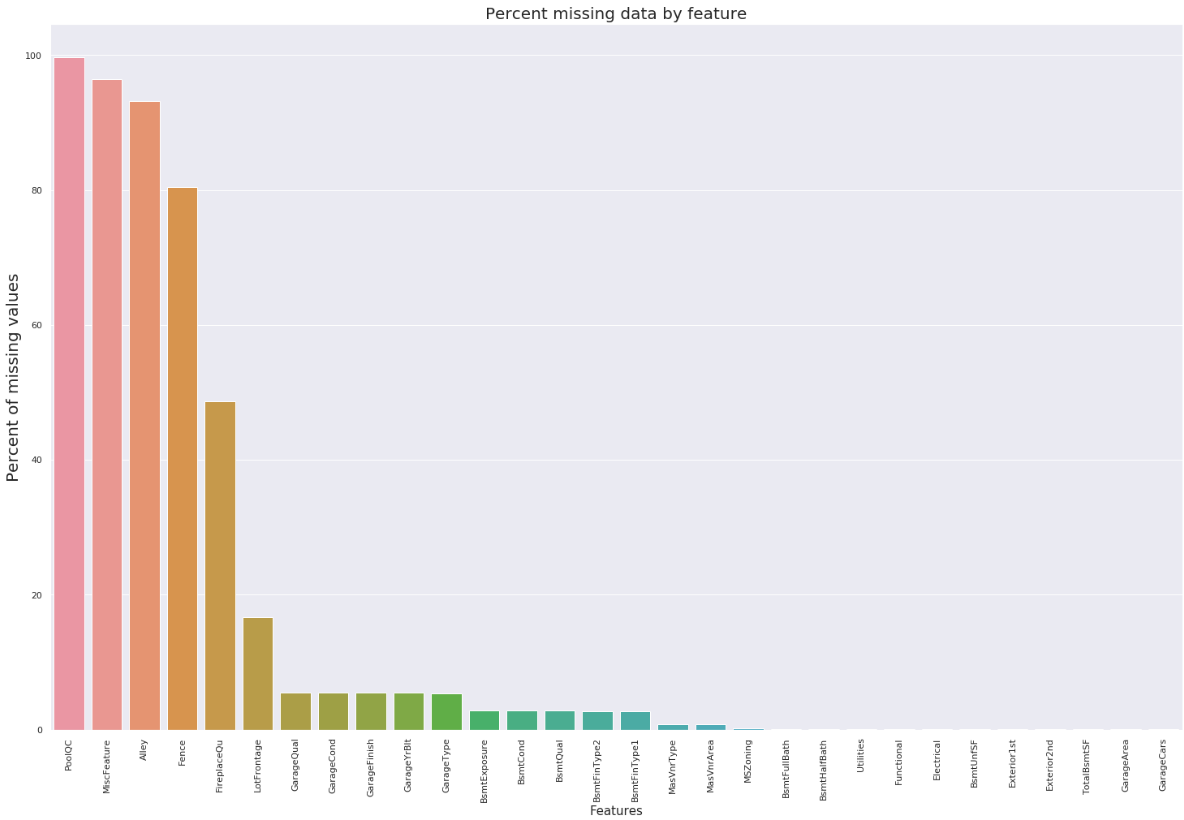
Out[108]:

	Missing Ratio
PoolQC	99.657417
MiscFeature	96.402878
Alley	93.216855
Fence	80.438506
FireplaceQu	48.646797
LotFrontage	16.649538
GarageQual	5.447071
GarageCond	5.447071
GarageFinish	5.447071
GarageYrBlt	5.447071
GarageType	5.378554
BsmtExposure	2.809181
BsmtCond	2.809181
BsmtQual	2.774923
BsmtFinType2	2.740665
BsmtFinType1	2.706406
MasVnrType	0.822199
MasVnrArea	0.787941
MSZoning	0.137033
BsmtFullBath	0.068517

To get a better understandation of missing data, we can use barplot to show the percentage of missing data

In [109]:
```
f, ax = plt.subplots(figsize=(25, 16))
plt.xticks(rotation='90')
sns.barplot(x=all_data_na.index, y=all_data_na)
plt.xlabel('Features', fontsize=15)
plt.ylabel('Percent of missing values', fontsize=20)
plt.title('Percent missing data by feature', fontsize=20)
```

Out[109]:
```
Text(0.5, 1.0, 'Percent missing data by feature')
```



Observation:

1. Here we can see the number of missing features by percentage and PoolQC, Alley and MiscFeature is nearly 100% missing.
2. We have to fill up this missing data according to the data description.

Let's analyse this to understand how to handle the missing data. We impute them by proceeding sequentially through features with missing values.

1. PoolQC : data description says NA means "No Pool".
2. MiscFeature : data description says NA means "no misc feature"
3. Alley : data description says NA means "no alley access"
4. Fence : data description says NA means "no fence
5. FireplaceQu : data description says NA means "no fireplace"
6. LotFrontage : Since the area of each street connected to the house property most likely have a similar area to other houses in its neighborhood , we can fill in missing values by the median LotFrontage of the neighborhood
7. GarageType, GarageFinish, GarageQual and GarageCond : Replacing missing data with None
8. GarageYrBlt, GarageArea and GarageCars : Replacing missing data with 0 (Since No garage = no cars in such garage.)
9. BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, BsmtFullBath and BsmtHalfBath : missing values are likely zero for having no basement
10. BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1 and BsmtFinType2 : For all these categorical basement-related features, NaN means that there is no basement.
11. MasVnrArea and MasVnrType : NA most likely means no masonry veneer for these houses. We can fill 0 for the area and None for the type.
12. MSZoning (The general zoning classification) : 'RL' is by far the most common value. So we can fill in missing values with 'RL'
13. Utilities : For this categorical feature all records are "AllPub", except for one "NoSeWa" and 2 NA . Since the house with 'NoSewa' is in the training set, this feature won't help in predictive modelling. We can then safely remove it.
14. Functional : data description says NA means typical
15. Electrical : It has one NA value. Since this feature has mostly 'SBrkr', we can set that for the missing value.
16. KitchenQual: Only one NA value, and same as Electrical, we set 'TA' (which is the most frequent) for the missing value in KitchenQual.
17. Exterior1st and Exterior2nd : Again Both Exterior 1 & 2 have only one missing value. We will just substitute in the most common string
18. SaleType : Fill in again with most frequent which is "WD"
19. MSSubClass : Na most likely means No building class. We can replace missing values with None

In [110]:
```
all_data["PoolQC"] = all_data["PoolQC"].fillna("None")
all_data["MiscFeature"] = all_data["MiscFeature"].fillna("None")
all_data["Alley"] = all_data["Alley"].fillna("None")
all_data["Fence"] = all_data["Fence"].fillna("None")
all_data["FireplaceQu"] = all_data["FireplaceQu"].fillna("None")
all_data["LotFrontage"] = all_data.groupby("Neighborhood")["LotFron
tage"].transform(
    lambda x: x.fillna(x.median()))
for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond
'):
    all_data[col] = all_data[col].fillna('None')
for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):
    all_data[col] = all_data[col].fillna(0)
for col in ('BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF',
'BsmtFullBath', 'BsmtHalfBath'):
    all_data[col] = all_data[col].fillna(0)
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1',
'BsmtFinType2'):
    all_data[col] = all_data[col].fillna('None')
all_data["MasVnrType"] = all_data["MasVnrType"].fillna("None")
all_data["MasVnrArea"] = all_data["MasVnrArea"].fillna(0)
all_data['MSZoning'] = all_data['MSZoning'].fillna(all_data['MSZoni
ng'].mode()[0])
all_data = all_data.drop(['Utilities'], axis=1)
all_data["Functional"] = all_data["Functional"].fillna("Typ")
all_data['Electrical'] = all_data['Electrical'].fillna(all_data['El
ectrical'].mode()[0])
all_data['KitchenQual'] = all_data['KitchenQual'].fillna(all_data['
KitchenQual'].mode()[0])
all_data['Exterior1st'] = all_data['Exterior1st'].fillna(all_data['
Exterior1st'].mode()[0])
all_data['Exterior2nd'] = all_data['Exterior2nd'].fillna(all_data['
Exterior2nd'].mode()[0])
all_data['SaleType'] = all_data['SaleType'].fillna(all_data['SaleTy
pe'].mode()[0])
all_data['MSSubClass'] = all_data['MSSubClass'].fillna("None")
```

As per our assumption there should be no missing values. We can check if there is any missing values left

In [111]:
```
all_data_na = (all_data.isnull().sum() / len(all_data)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index)
.sort_values(ascending=False)
missing_data = pd.DataFrame({'No More Missing Data' :all_data_na})
missing_data.head()
```

Out[111]:

No More Missing Data

So there is no missing data left!

Now lets label encode the catagorical features that contain information in ordering set

In [112]:
```
all_data['MSSubClass'] = all_data['MSSubClass'].apply(str)

all_data['OverallCond'] = all_data['OverallCond'].astype(str)

all_data['YrSold'] = all_data['YrSold'].astype(str)
all_data['MoSold'] = all_data['MoSold'].astype(str)

from sklearn.preprocessing import LabelEncoder
cols = ('FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond',
        'ExterQual', 'ExterCond','HeatingQC', 'PoolQC', 'KitchenQual', 'BsmtFinType1',
        'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish', 'LandSlope',
        'LotShape', 'PavedDrive', 'Street', 'Alley', 'CentralAir', 'MSSubClass', 'OverallCond',
        'YrSold', 'MoSold')

for c in cols:
    lbl = LabelEncoder()
    lbl.fit(list(all_data[c].values))
    all_data[c] = lbl.transform(list(all_data[c].values))


print('Shape all_data: {}'.format(all_data.shape))
```

```
Shape all_data: (2919, 79)
```

Since area related features are very important to determine house prices, we add one more feature which is the total area of basement, first and second floor areas of each house.

In [113]:
```
all_data['TotalSF'] = all_data['TotalBsmtSF'] + all_data['1stFlrSF'] + all_data['2ndFlrSF']
```

Lets check the skewness in the numeric features

In [114]:
```
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index

skewed_feats = all_data[numeric_feats].apply(lambda x: skew(x.dropn
a())).sort_values(ascending=False)
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(10)
```

```
Skew in numerical features:
```

Out[114]:

	Skew
MiscVal	21.947195
PoolArea	16.898328
LotArea	12.822431
LowQualFinSF	12.088761
3SsnPorch	11.376065
LandSlope	4.975157
KitchenAbvGr	4.302254
BsmtFinSF2	4.146143
EnclosedPorch	4.003891
ScreenPorch	3.946694

Now I am handling skewness of the features. For this I will take the BoxCox transform of the features with skewness > 0.5.

Reference: http://onlinestatbook.com/mobile/transformations/box-cox.html (http://onlinestatbook.com/mobile/transformations/box-cox.html)

BoxCox transform skewed numeric features:

1. We use the scipy function boxcox1p which computes the Box-Cox transformation of 1+x .
2. Note that setting λ=0 is equivalent to log1p used above for the target variable.

In [115]:
```
skewness = skewness[abs(skewness) > 0.5]
print("There are {} skewed numerical features to Box Cox transform"
.format(skewness.shape[0]))

from scipy.special import boxcox1p
skewed_features = skewness.index
lam = 0.15
for feat in skewed_features:
    #all_data[feat] += 1
    all_data[feat] = boxcox1p(all_data[feat], lam)

#all_data[skewed_features] = np.log1p(all_data[skewed_features])
```

```
There are 60 skewed numerical features to Box Cox transform
```

Getting dummy for catagorical features and new train and test datasets

In [116]:
```
all_data = pd.get_dummies(all_data)
print(all_data.shape)

train = all_data[:ntrain]
test = all_data[ntrain:]

dl_train=train
dl_test=test
```

```
(2919, 222)
```

Regression Approach:

Here we will take some regression algorithoms and get our predicted outputs from the models. We will do cross validation to check the RMSLE score. Lets start by importing the necessary libraries.

In [117]:
```
from sklearn.linear_model import ElasticNet, Lasso,  BayesianRidge,
LassoLarsIC
from sklearn.ensemble import RandomForestRegressor,  GradientBoosti
ngRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.base import BaseEstimator, TransformerMixin, Regressor
Mixin, clone
from sklearn.model_selection import KFold, cross_val_score, train_t
est_split
from sklearn.metrics import mean_squared_error
import xgboost as xgb
import lightgbm as lgb
```

Cross Validation:

We use the cross_val_score function of Sklearn. However this function has not a shuffle attribut, we add then one line of code, in order to shuffle the dataset prior to cross-validation

In [118]:
```
n_folds = 5

def rmsle_cv(model):
    kf = KFold(n_folds, shuffle=True, random_state=42).get_n_splits
(train.values)
    rmse= np.sqrt(-cross_val_score(model, train.values, y_train, sc
oring="neg_mean_squared_error", cv = kf))
    return(rmse)
```

Lets build the base models

LASSO Regression : This model may be very sensitive to outliers. So we need to made it more robust on them. For that we use the sklearn's Robustscaler() method on pipeline

In [119]:
```
lasso = make_pipeline(RobustScaler(), Lasso(alpha =0.0005, random_s
tate=1))
```

Elastic Net Regression : Again made robust to outliers hence we used RobustScaler() method.

In [120]:
```
ENet = make_pipeline(RobustScaler(), ElasticNet(alpha=0.0005, l1_ra
tio=.9, random_state=3))
```

Kernel Ridge Regression :

In [121]:
```
KRR = KernelRidge(alpha=0.6, kernel='polynomial', degree=2, coef0=2
.5)
```

Gradient Boosting Regression : With huber loss makes it robust to outliers

In [122]:
```
GBoost = GradientBoostingRegressor(n_estimators=3000, learning_rate
=0.05,
                                   max_depth=4, max_features='sqrt'
,
                                   min_samples_leaf=15, min_samples
_split=10,
                                   loss='huber', random_state =5)
```

XGBoost :

In [123]:
```
model_xgb = xgb.XGBRegressor(colsample_bytree=0.4603, gamma=0.0468,
                             learning_rate=0.05, max_depth=3,
                             min_child_weight=1.7817, n_estimators=
2200,
                             reg_alpha=0.4640, reg_lambda=0.8571,
                             subsample=0.5213, silent=1,
                             random_state =7, nthread = -1)
```

LightGBM :

In [124]:
```
model_lgb = lgb.LGBMRegressor(objective='regression',num_leaves=5,
                              learning_rate=0.05, n_estimators=720,
                              max_bin = 55, bagging_fraction = 0.8,
                              bagging_freq = 5, feature_fraction =
0.2319,
                              feature_fraction_seed=9, bagging_seed
=9,
                              min_data_in_leaf =6, min_sum_hessian_
in_leaf = 11)
```

Lets define a RMSLE function

In [125]:
```
def rmsle(y, y_pred):
    return np.sqrt(mean_squared_error(y, y_pred))
```

Model Scores: Let's see how these base models perform on the data by evaluating the cross-validation rmsle error

In [126]:
```
score = rmsle_cv(lasso)
```

In [126]:
```
score = rmsle_cv(lasso)
print("\nLasso score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

lasso.fit(train, y_train)
lasso_train_pred = lasso.predict(train)
lasso_pred = np.expm1(lasso.predict(test))
print(rmsle(y_train,lasso_train_pred))
print(' ')

score = rmsle_cv(ENet)
print("ElasticNet score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

ENet.fit(train, y_train)
ENet_train_pred = ENet.predict(train)
ENet_pred = np.expm1(ENet.predict(test))
print(rmsle(y_train, ENet_train_pred))
print(' ')

score = rmsle_cv(KRR)
print("Kernel Ridge score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

KRR.fit(train, y_train)
KRR_train_pred = KRR.predict(train)
KRR_pred = np.expm1(KRR.predict(test))
print(rmsle(y_train, KRR_train_pred))
print(' ')

score = rmsle_cv(GBoost)
print("Gradient Boosting score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

GBoost.fit(train, y_train)
GBoost_train_pred = GBoost.predict(train)
GBoost_pred = np.expm1(GBoost.predict(test))
print(rmsle(y_train, GBoost_train_pred))
print(' ')

score = rmsle_cv(model_xgb)
print("Xgboost score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

model_xgb.fit(train, y_train)
xgb_train_pred = model_xgb.predict(train)
xgb_pred = np.expm1(model_xgb.predict(test))
print(rmsle(y_train, xgb_train_pred))
print(' ')

score = rmsle_cv(model_lgb)
print("LGBM score: {:.4f} ({:.4f})\n" .format(score.mean(), score.std()))
```

```
model_lgb.fit(train, y_train)
lgb_train_pred = model_lgb.predict(train)
lgb_pred = np.expm1(model_lgb.predict(test))
print(rmsle(y_train, lgb_train_pred))
print(' ')
```

```
Lasso score: 0.1240 (0.0165)

0.10673452654040456

ElasticNet score: 0.1241 (0.0165)

0.10600525366156172

Kernel Ridge score: 0.1273 (0.0113)

0.0870512628850622

Gradient Boosting score: 0.1244 (0.0126)

0.05603455322072817

Xgboost score: 0.1216 (0.0102)

0.07824059899749358

LGBM score: 0.1236 (0.0091)

0.07426777676065932
```

Observation:

Al the regressor algorithms has very good output results. Now I will predict from all the regression algorithms and submit them to kaggle to check the scores and determine the best algorithm.

In [127]:
```
submission = pd.DataFrame()

test_data=pd.read_csv('../input/house-prices-advanced-regression-te
chniques/test.csv')

submission['Id'] = test_data['Id']

submission['SalePrice'] = lasso_pred = np.expm1(lasso.predict(test)
)

submission.to_csv('submission_lasso.csv' , index=False)
```

In [128]:
```
submission = pd.DataFrame()

test_data=pd.read_csv('../input/house-prices-advanced-regression-te
chniques/test.csv')

submission['Id'] = test_data['Id']

submission['SalePrice'] = ENet_pred = np.expm1(ENet.predict(test))

submission.to_csv('submission_ENet.csv' , index=False)
```

In [129]:
```
submission = pd.DataFrame()

test_data=pd.read_csv('../input/house-prices-advanced-regression-te
chniques/test.csv')

submission['Id'] = test_data['Id']

submission['SalePrice'] = KRR_pred = np.expm1(KRR.predict(test))

submission.to_csv('submission_KRR.csv' , index=False)
```

In [130]:
```
submission = pd.DataFrame()

test_data=pd.read_csv('../input/house-prices-advanced-regression-te
chniques/test.csv')

submission['Id'] = test_data['Id']

submission['SalePrice'] = GBoost_pred = np.expm1(GBoost.predict(tes
t))

submission.to_csv('submission_GBoost.csv' , index=False)
```

In [131]:
```
submission = pd.DataFrame()

test_data=pd.read_csv('../input/house-prices-advanced-regression-te
chniques/test.csv')

submission['Id'] = test_data['Id']

submission['SalePrice'] = xgb_pred = np.expm1(model_xgb.predict(tes
t))

submission.to_csv('submission_xgb.csv' , index=False)
```

In [132]:
```
submission = pd.DataFrame()

test_data=pd.read_csv('../input/house-prices-advanced-regression-te
chniques/test.csv')

submission['Id'] = test_data['Id']

submission['SalePrice'] = lgb_pred = np.expm1(model_lgb.predict(tes
t))

submission.to_csv('submission_lgb.csv' , index=False)
```

Kaggle Scores:

1. Lasso : 0.12039
2. Elastic Net : 0.12055
3. Kernel Ridge Regressor : 0.12523
4. Gradient Boost : 0.12495
5. XGBoost : 0.12646
6. LightGBM : 0.12290

6 submissions for Anindya Sanyal — Sort by: Public Score

All | Successful | Selected

Submission and Description	Public Score	Use for Final Score
submission_lasso.csv 9 minutes ago by Anindya Sanyal add submission details	0.12039	☐
submission_ENet.csv 8 minutes ago by Anindya Sanyal add submission details	0.12055	☐
submission_lgb.csv a few seconds ago by Anindya Sanyal add submission details	0.12290	☐
submission_GBoost.csv 2 minutes ago by Anindya Sanyal add submission details	0.12495	☐
submission_KRR.csv 7 minutes ago by Anindya Sanyal add submission details	0.12523	☐
submission_xgb.csv a minute ago by Anindya Sanyal add submission details	0.12646	☐

Observation:

1. The lasso model scores the best hence the best score in Kaggle.
2. All the regression algorithoms will be used later for a stacked approach.

Averaged Model:

We begin with this simple approach of averaging base models. We build a new class to extend scikit-learn with our model and also to laverage encapsulation and code reuse

In [133]:
```
class AveragingModels(BaseEstimator, RegressorMixin, TransformerMix
in):
    def __init__(self, models):
        self.models = models

    # we define clones of the original models to fit the data in
    def fit(self, X, y):
        self.models_ = [clone(x) for x in self.models]

        # Train cloned base models
        for model in self.models_:
            model.fit(X, y)

        return self

    #Now we do the predictions for cloned models and average them
    def predict(self, X):
        predictions = np.column_stack([
            model.predict(X) for model in self.models_
        ])
        return np.mean(predictions, axis=1)
```

In [134]:
```
averaged_models = AveragingModels(models = (ENet, GBoost, KRR, lass
o, model_xgb, model_lgb))

score = rmsle_cv(averaged_models)
print(" Averaged base models score: {:.4f} ({:.4f})\n".format(score
.mean(), score.std()))
```

```
 Averaged base models score: 0.1177 (0.0125)
```

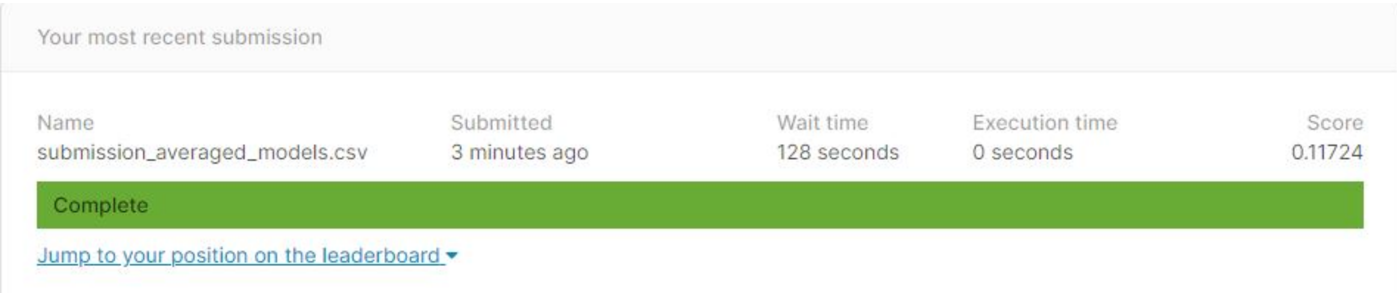
Lets predict using this Average Stack Model and save it to CSV for submission.

In [135]:
```
averaged_models.fit(train.values, y_train)
averaged_models_train_pred = averaged_models.predict(train.values)
averaged_models_pred = np.expm1(averaged_models.predict(test.values
))
print(rmsle(y_train, averaged_models_train_pred))
```

```
0.07883015000312331
```

In [136]:
```
submission = pd.DataFrame()

test_data=pd.read_csv('../input/house-prices-advanced-regression-techniques/test.csv')

submission['Id'] = test_data['Id']

submission['SalePrice'] = averaged_models_pred = np.expm1(averaged_models.predict(test.values))

submission.to_csv('submission_averaged_models.csv' , index=False)
```

Kaggle Score: 0.11724 which is better than any other regressor models.



Observation:

Averaging the models gives a significant boost to the results. So stacking models and ensembling approach is the right way to go. Lets create a meta-model to increase performance

In this approach, we add a meta-model on averaged base models and use the out-of-folds predictions of these base models to train our meta-model.

The procedure, for the training part, may be described as follows:

1. Split the total training set into two disjoint sets (here train and holdout)
2. Train several base models on the first part (train)
3. Test these base models on the second part (holdout)
4. Use the predictions from (3) (called out-of-folds predictions) as the inputs, and the correct responses (target variable 'SalePrice') as the outputs to train a higher level learner called meta-model.

The first three steps are done iteratively . If we take for example a 5-fold stacking , we first split the training data into 5 folds. Then we will do 5 iterations. In each iteration, we train every base model on 4 folds and predict on the remaining fold (holdout fold).

So, we will be sure, after 5 iterations , that the entire data is used to get out-of-folds predictions that we will then use as new feature to train our meta-model in the step 4.

For the prediction part , We average the predictions of all base models on the test data and used them as meta-features on which, the final prediction is done with the meta-model.

Stacking averaged Model:

In [137]:
```
class StackingAveragedModels(BaseEstimator, RegressorMixin, Transfo
rmerMixin):
    def __init__(self, base_models, meta_model, n_folds=5):
        self.base_models = base_models
        self.meta_model = meta_model
        self.n_folds = n_folds

    # We again fit the data on clones of the original models
    def fit(self, X, y):
        self.base_models_ = [list() for x in self.base_models]
        self.meta_model_ = clone(self.meta_model)
        kfold = KFold(n_splits=self.n_folds, shuffle=True, random_s
tate=156)

        # Train cloned base models then create out-of-fold predicti
ons
        # that are needed to train the cloned meta-model
        out_of_fold_predictions = np.zeros((X.shape[0], len(self.ba
se_models)))
        for i, model in enumerate(self.base_models):
            for train_index, holdout_index in kfold.split(X, y):
                instance = clone(model)
                self.base_models_[i].append(instance)
                instance.fit(X[train_index], y[train_index])
                y_pred = instance.predict(X[holdout_index])
                out_of_fold_predictions[holdout_index, i] = y_pred

        # Now train the cloned  meta-model using the out-of-fold pr
edictions as new feature
        self.meta_model_.fit(out_of_fold_predictions, y)
        return self

    #Do the predictions of all base models on the test data and use
the averaged predictions as
    #meta-features for the final prediction which is done by the me
ta-model
    def predict(self, X):
        meta_features = np.column_stack([
            np.column_stack([model.predict(X) for model in base_mod
els]).mean(axis=1)
            for base_models in self.base_models_ ])
        return self.meta_model_.predict(meta_features)
```

Stacking Averaged Model Scores: To make the two approaches comparable (by using the same number of models) , we just average Enet, KRR and Gboost, then we add lasso as meta-model.

In [138]:
```
stacked_averaged_models = StackingAveragedModels(base_models = (ENet, GBoost, KRR),
                                                 meta_model = lasso)

score = rmsle_cv(stacked_averaged_models)
print("Stacking Averaged models score: {:.4f} ({:.4f})".format(score.mean(), score.std()))
```

```
Stacking Averaged models score: 0.1193 (0.0139)
```

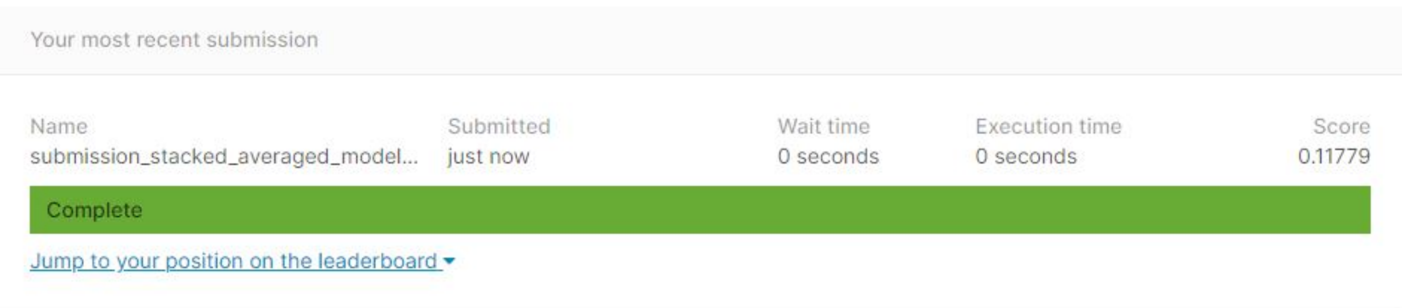
Prediction using meta-model:

In [139]:
```
stacked_averaged_models.fit(train.values, y_train)
stacked_train_pred = stacked_averaged_models.predict(train.values)
stacked_pred = np.expm1(stacked_averaged_models.predict(test.values))
print(rmsle(y_train, stacked_train_pred))
```

```
0.08173736258413 98
```

In [140]:
```
submission = pd.DataFrame()

test_data=pd.read_csv('../input/house-prices-advanced-regression-techniques/test.csv')

submission['Id'] = test_data['Id']

submission['SalePrice'] = stacked_averaged_models_pred = np.expm1(stacked_averaged_models.predict(test.values))

submission.to_csv('submission_stacked_averaged_models.csv' , index=False)
```

Kaggle Score: 0.11779



Ensemble approch prediction:

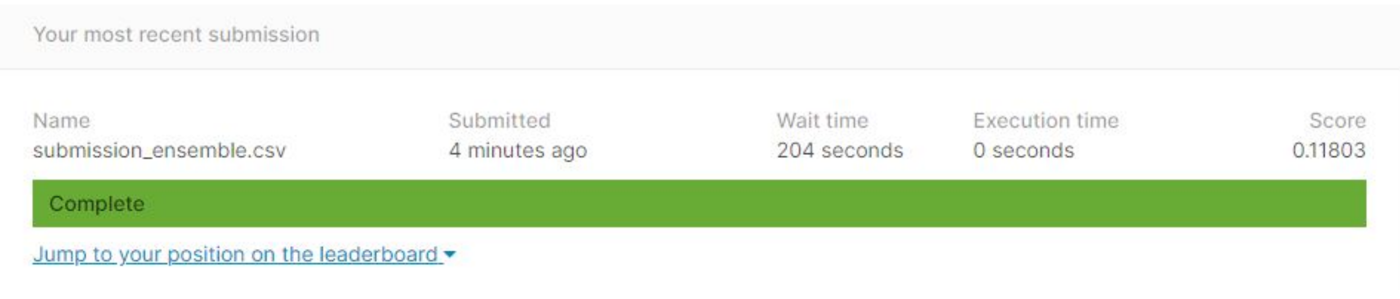
In the ensemble approach we will use a formula " **ensemble = stacked_pred*0.70 + xgb_pred*0.15 + lgb_pred*0.15** ** " to determine the house prices.

In [141]:
```
ensemble = stacked_pred*0.70 + xgb_pred*0.15 + lgb_pred*0.15

print('RMSLE score on train data on ensemble approch:')
print(rmsle(y_train,stacked_train_pred*0.70 + xgb_train_pred*0.15 +
lgb_train_pred*0.15 ))
```

```
RMSLE score on train data on ensemble approch:
0.0776014605600492
```

In [142]:
```
submission = pd.DataFrame()

test_data=pd.read_csv('../input/house-prices-advanced-regression-te
chniques/test.csv')

submission['Id'] = test_data['Id']

submission['SalePrice'] = ensemble
submission.to_csv('submission_ensemble.csv',index=False)
```

Kaggle Score: 0.11803



Deep Neural Network Approach:

First of all lets install tensorflow 1.4 as the latest version of tensorflow is not compitable with my code. Also we will check the current input and output directory.

In [143]:
```
!pip install tensorflow-gpu==1.4.0
!pip install tensorflow==1.4.0
import numpy as np
import pandas as pd


import os
for dirname, _, filenames in os.walk('/kaggle/'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
Requirement already satisfied: tensorflow-gpu==1.4.0 in /opt/conda
/lib/python3.6/site-packages (1.4.0)
Requirement already satisfied: tensorflow-tensorboard<0.5.0,>=0.4.
0rc1 in /opt/conda/lib/python3.6/site-packages (from tensorflow-gp
u==1.4.0) (0.4.0)
Requirement already satisfied: enum34>=1.1.6 in /opt/conda/lib/pyt
hon3.6/site-packages (from tensorflow-gpu==1.4.0) (1.1.10)
Requirement already satisfied: wheel>=0.26 in /opt/conda/lib/pytho
n3.6/site-packages (from tensorflow-gpu==1.4.0) (0.34.2)
Requirement already satisfied: numpy>=1.12.1 in /opt/conda/lib/pyt
hon3.6/site-packages (from tensorflow-gpu==1.4.0) (1.18.1)
Requirement already satisfied: protobuf>=3.3.0 in /opt/conda/lib/p
ython3.6/site-packages (from tensorflow-gpu==1.4.0) (3.11.3)
Requirement already satisfied: six>=1.10.0 in /opt/conda/lib/pytho
n3.6/site-packages (from tensorflow-gpu==1.4.0) (1.14.0)
Requirement already satisfied: markdown>=2.6.8 in /opt/conda/lib/p
ython3.6/site-packages (from tensorflow-tensorboard<0.5.0,>=0.4.0r
c1->tensorflow-gpu==1.4.0) (3.2.1)
Requirement already satisfied: bleach==1.5.0 in /opt/conda/lib/pyt
hon3.6/site-packages (from tensorflow-tensorboard<0.5.0,>=0.4.0rc1
->tensorflow-gpu==1.4.0) (1.5.0)
Requirement already satisfied: werkzeug>=0.11.10 in /opt/conda/lib
/python3.6/site-packages (from tensorflow-tensorboard<0.5.0,>=0.4.
0rc1->tensorflow-gpu==1.4.0) (1.0.0)
Requirement already satisfied: html5lib==0.9999999 in /opt/conda/l
ib/python3.6/site-packages (from tensorflow-tensorboard<0.5.0,>=0.
4.0rc1->tensorflow-gpu==1.4.0) (0.9999999)
Requirement already satisfied: setuptools in /opt/conda/lib/python
3.6/site-packages (from protobuf>=3.3.0->tensorflow-gpu==1.4.0) (4
5.2.0.post20200210)
Requirement already satisfied: tensorflow==1.4.0 in /opt/conda/lib
/python3.6/site-packages (1.4.0)
Requirement already satisfied: enum34>=1.1.6 in /opt/conda/lib/pyt
hon3.6/site-packages (from tensorflow==1.4.0) (1.1.10)
Requirement already satisfied: numpy>=1.12.1 in /opt/conda/lib/pyt
hon3.6/site-packages (from tensorflow==1.4.0) (1.18.1)
Requirement already satisfied: tensorflow-tensorboard<0.5.0,>=0.4.
0rc1 in /opt/conda/lib/python3.6/site-packages (from tensorflow==1
.4.0) (0.4.0)
Requirement already satisfied: wheel>=0.26 in /opt/conda/lib/pytho
n3.6/site-packages (from tensorflow==1.4.0) (0.34.2)
```

```
Requirement already satisfied: six>=1.10.0 in /opt/conda/lib/pytho
n3.6/site-packages (from tensorflow==1.4.0) (1.14.0)
Requirement already satisfied: protobuf>=3.3.0 in /opt/conda/lib/p
ython3.6/site-packages (from tensorflow==1.4.0) (3.11.3)
Requirement already satisfied: bleach==1.5.0 in /opt/conda/lib/pyt
hon3.6/site-packages (from tensorflow-tensorboard<0.5.0,>=0.4.0rc1
->tensorflow==1.4.0) (1.5.0)
Requirement already satisfied: werkzeug>=0.11.10 in /opt/conda/lib
/python3.6/site-packages (from tensorflow-tensorboard<0.5.0,>=0.4.
0rc1->tensorflow==1.4.0) (1.0.0)
Requirement already satisfied: html5lib==0.9999999 in /opt/conda/l
ib/python3.6/site-packages (from tensorflow-tensorboard<0.5.0,>=0.
4.0rc1->tensorflow==1.4.0) (0.9999999)
Requirement already satisfied: markdown>=2.6.8 in /opt/conda/lib/p
ython3.6/site-packages (from tensorflow-tensorboard<0.5.0,>=0.4.0r
c1->tensorflow==1.4.0) (3.2.1)
Requirement already satisfied: setuptools in /opt/conda/lib/python
3.6/site-packages (from protobuf>=3.3.0->tensorflow==1.4.0) (45.2.
0.post20200210)
/kaggle/lib/kaggle/gcp.py
/kaggle/input/house-prices-advanced-regression-techniques/sample_s
ubmission.csv
/kaggle/input/house-prices-advanced-regression-techniques/data_des
cription.txt
/kaggle/input/house-prices-advanced-regression-techniques/test.csv
/kaggle/input/house-prices-advanced-regression-techniques/train.cs
v
/kaggle/working/submission_ensemble.csv
/kaggle/working/submission_GBoost.csv
/kaggle/working/submission_ENet.csv
/kaggle/working/submission_KRR.csv
/kaggle/working/submission_xgb.csv
/kaggle/working/submission_averaged_models.csv
/kaggle/working/__notebook_source__.ipynb
/kaggle/working/submission_lasso.csv
/kaggle/working/submission_lgb.csv
/kaggle/working/submission_stacked_averaged_models.csv
```

Dataset loading from previous preprocessed data:

In this cell we will define the functions that are required for dataset loading and data pre-processing. The functions that will be defined here are:

1. **load_data**: This function will be called to load our dataset using pandas.
2. **output_submission**: This function will generate a name for our output CSV file.
3. **pre_process_data**: This will actually take the input data and create a data frame using pandas library.
4. **mini_batches**: This function will be called during the training times.

In [192]:

```
import csv
```

In [192]:
```
import csv
import pandas as pd


def load_data(train_path, test_path):


    train_data = pd.read_csv(train_path)
    test_data  =  pd.read_csv(test_path)

    print("number of training examples = " + str(train_data.shape[0
]))
    print("number of test examples = " + str(test_data.shape[0]))
    print("train shape: " + str(train_data.shape))
    print("test shape: " + str(test_data.shape))

    return train_data, test_data


def output_submission(test_ids, predictions, id_column, predction_c
olumn, file_name):

    print('Outputting submission...')
    with open('/kaggle/working/' + file_name, 'w') as submission:
        writer = csv.writer(submission)
        writer.writerow([id_column, predction_column])
        for test_id, test_prediction in zip(test_ids, predictions):
            writer.writerow([test_id, test_prediction])
    print('Output complete')


def pre_process_data(df):

    df = pd.get_dummies(df)

    return df


def mini_batches(train_set, train_labels, mini_batch_size):

    set_size = train_set.shape[0]
    batches = []
    num_complete_minibatches = set_size // mini_batch_size

    for k in range(0, num_complete_minibatches):
        mini_batch_x = train_set[k * mini_batch_size: (k + 1) * min
i_batch_size]
        mini_batch_y = train_labels[k * mini_batch_size: (k + 1) *
mini_batch_size]
        mini_batch = (mini_batch_x, mini_batch_y)
        batches.append(mini_batch)


    if set_size % mini_batch_size != 0:
```

```
        mini_batch_x = train_set[(set_size - (set_size % mini_batch
_size)):]
        mini_batch_y = train_labels[(set_size - (set_size % mini_ba
tch_size)):]
        mini_batch = (mini_batch_x, mini_batch_y)
        batches.append(mini_batch)

    return batches
```

Methods:

Here we will describe the functions that are required for our model to train. All the functions defined here are required for our model to train.

In [193]:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

with tf.device("/gpu:0"):

    def create_placeholders(input_size, output_size):


        x = tf.placeholder(shape=(None, input_size), dtype=tf.float
32, name="X")
        y = tf.placeholder(shape=(None, output_size), dtype=tf.floa
t32, name="Y")

        return x, y


    def forward_propagation(x, parameters, keep_prob=1.0, hidden_ac
tivation='relu'):


        a_dropout = x
        n_layers = len(parameters) // 2  # number of layers in the
neural network

        for l in range(1, n_layers):
            a_prev = a_dropout
            a_dropout = linear_activation_forward(a_prev, parameter
s['w%s' % l], parameters['b%s' % l], hidden_activation)

            if keep_prob < 1.0:
                a_dropout = tf.nn.dropout(a_dropout, keep_prob)

        al = tf.matmul(a_dropout, parameters['w%s' % n_layers]) + p
arameters['b%s' % n_layers]
```

```
        return al

    def linear_activation_forward(a_prev, w, b, activation):


        a = None
        if activation == "sigmoid":
            z = tf.matmul(a_prev, w) + b
            a = tf.nn.sigmoid(z)

        elif activation == "relu":
            z = tf.matmul(a_prev, w) + b
            a = tf.nn.relu(z)

        elif activation == "leaky relu":
            z = tf.matmul(a_prev, w) + b
            a = tf.nn.leaky_relu(z)

        return a


    def initialize_parameters(layer_dims):


        parameters = {}
        n_layers = len(layer_dims)

        for l in range(1, n_layers):
            parameters['w' + str(l)] = tf.get_variable('w' + str(l)
, [layer_dims[l - 1], layer_dims[l]],
                                                        initializer=
tf.contrib.layers.xavier_initializer())
            parameters['b' + str(l)] = tf.get_variable('b' + str(l)
, [layer_dims[l]], initializer=tf.zeros_initializer())

        return parameters


    def compute_cost(z3, y):


        cost = tf.sqrt(tf.reduce_mean(tf.square(y - z3)))

        return cost


    def predict(data, parameters):


        init = tf.global_variables_initializer()
        with tf.Session() as sess:
            sess.run(init)
```

```
            dataset = tf.cast(tf.constant(data), tf.float32)
            fw_prop_result = forward_propagation(dataset, parameter
s)
            prediction = fw_prop_result.eval()

        return prediction


    def rmse(predictions, labels):


        prediction_size = predictions.shape[0]
        prediction_cost = np.sqrt(np.sum(np.square(labels - predict
ions)) / prediction_size)

        return prediction_cost


    def rmsle(predictions, labels):


        prediction_size = predictions.shape[0]
        prediction_cost = np.sqrt(np.sum(np.square(np.log(predictio
ns + 1) - np.log(labels + 1))) / prediction_size)

        return prediction_cost


    def l2_regularizer(cost, l2_beta, parameters, n_layers):


        regularizer = 0
        for i in range(1, n_layers):
            regularizer += tf.nn.l2_loss(parameters['w%s' % i])

        cost = tf.reduce_mean(cost + l2_beta * regularizer)

        return cost


    def build_submission_name(layers_dims, num_epochs, lr_decay,
                              learning_rate, l2_beta, keep_prob, mi
nibatch_size, num_examples):


        submission_name = 'ly{}-epoch{}.csv' \
            .format(layers_dims, num_epochs)

        if lr_decay != 0:
            submission_name = 'lrdc{}-'.format(lr_decay) + submissi
on_name
        else:
```

```
            submission_name = 'lr{}-'.format(learning_rate) + submi
ssion_name

        if l2_beta > 0:
             submission_name = 'l2{}-'.format(l2_beta) + submission
_name

        if keep_prob < 1:
            submission_name = 'dk{}-'.format(keep_prob) + submissio
n_name
            submission_name = 'dk{}-'.format(keep_prob) + submissio
n_name

        if minibatch_size != num_examples:
            submission_name = 'mb{}-'.format(minibatch_size) + subm
ission_name

        return submission_name


    def plot_model_cost(train_costs, validation_costs, submission_n
ame):

        plt.plot(np.squeeze(train_costs), label='Train cost')
        plt.plot(np.squeeze(validation_costs), label='Validation co
st')
        plt.ylabel('cost')
        plt.xlabel('iterations (per tens)')
        plt.title("Model: " + submission_name)
        plt.legend()
        plt.show()
```

Model:

Here we will train our model using Tensorflow. All the required functions are already described in **METHODS**. We will calculate the training and validation costs which will be required to plot the **cost vs iteration** curve.

In [194]:

```
import tensorflow as tf
from tensorflow.python.framework import ops



def model(train_set, train_labels, validation_set, validation_label
s, layers_dims, learning_rate=0.01, num_epochs=1001,
          print_cost=True, plot_cost=True, l2_beta=0., keep_prob=1.
0, hidden_activation='relu', return_best=False,
          minibatch_size=0, lr_decay=0, evaluate=True):

    with tf.device("/gpu:0"):
```

```
        ops.reset_default_graph()

        input_size = layers_dims[0]
        output_size = layers_dims[-1]
        num_examples = train_set.shape[0]
        n_layers = len(layers_dims)
        train_costs = []
        validation_costs = []
        best_iteration = [float('inf'), 0]
        best_params = None

        if minibatch_size == 0 or minibatch_size > num_examples:
            minibatch_size = num_examples

        num_minibatches = num_examples // minibatch_size

        if num_minibatches == 0:
            num_minibatches = 1

        submission_name = build_submission_name(layers_dims, num_ep
ochs, lr_decay, learning_rate, l2_beta, keep_prob,
                                                minibatch_size, num
_examples)

        x, y = create_placeholders(input_size, output_size)
        tf_valid_dataset = tf.cast(tf.constant(validation_set), tf.
float32)
        parameters = initialize_parameters(layers_dims)

        fw_output_train = forward_propagation(x, parameters, keep_p
rob, hidden_activation)
        train_cost = compute_cost(fw_output_train, y)

        fw_output_valid = forward_propagation(tf_valid_dataset, par
ameters, 1, hidden_activation)
        validation_cost = compute_cost(fw_output_valid, validation_
labels)

        if l2_beta > 0:
            train_cost = l2_regularizer(train_cost, l2_beta, parame
ters, n_layers)
            validation_cost = l2_regularizer(validation_cost, l2_be
ta, parameters, n_layers)

        if lr_decay != 0:
            global_step = tf.Variable(0, trainable=False)
            learning_rate = tf.train.inverse_time_decay(learning_ra
te, global_step=global_step, decay_rate=lr_decay,
                                                        decay_steps
=1)
            optimizer = tf.train.AdamOptimizer(learning_rate).minim
ize(train_cost, global_step=global_step)
```

```
        else:
            optimizer = tf.train.AdamOptimizer(learning_rate).minim
ize(train_cost)



        init = tf.global_variables_initializer()




    with tf.Session() as sess:


        sess.run(init)

        for epoch in range(num_epochs):
            train_epoch_cost = 0.
            validation_epoch_cost = 0.

            minibatches = mini_batches(train_set, train_labels, min
ibatch_size)

            for minibatch in minibatches:


                (minibatch_X, minibatch_Y) = minibatch
                feed_dict = {x: minibatch_X, y: minibatch_Y}


                _, minibatch_train_cost, minibatch_validation_cost
= sess.run(
                    [optimizer, train_cost, validation_cost], feed_
dict=feed_dict)

                train_epoch_cost += minibatch_train_cost / num_mini
batches
                validation_epoch_cost += minibatch_validation_cost
/ num_minibatches

            if print_cost is True and epoch % 500 == 0:

                if evaluate is False:
                    print("Train cost after epoch %i: %f" % (epoch,
train_epoch_cost))
                    print("Validation cost after epoch %i: %f" % (e
poch, validation_epoch_cost))


            if plot_cost is True and epoch % 10 == 0:
                train_costs.append(train_epoch_cost)
                validation_costs.append(validation_epoch_cost)
```

```
            if return_best is True and validation_epoch_cost < best_iteration[0]:
                best_iteration[0] = validation_epoch_cost
                best_iteration[1] = epoch
                best_params = sess.run(parameters)

        if return_best is True:
            parameters = best_params
        else:
            parameters = sess.run(parameters)


        if evaluate is False:
            print("Parameters have been trained, getting metrics...")

        train_rmse = rmse(predict(train_set, parameters), train_labels)
        validation_rmse = rmse(predict(validation_set, parameters), validation_labels)
        train_rmsle = rmsle(predict(train_set, parameters), train_labels)
        validation_rmsle = rmsle(predict(validation_set, parameters), validation_labels)

        if evaluate is False:
            print('Train rmse: {:.4f}'.format(train_rmse))
            print('Validation rmse: {:.4f}'.format(validation_rmse))
            print('Train rmsle: {:.4f}'.format(train_rmsle))
            print('Validation rmsle: {:.4f}'.format(validation_rmsle))

        submission_name = 'tr_cost-{:.2f}-vd_cost{:.2f}-'.format(train_rmse, validation_rmse) + submission_name

        if return_best is True:
            print('Lowest rmse: {:.2f} at epoch {}'.format(best_iteration[0], best_iteration[1]))

        if plot_cost is True:
            plot_model_cost(train_costs, validation_costs, submission_name)

        return parameters, submission_name
```

Cross Validation Function:

The cross validation function for the Deep Neural Network Model is defined below:

In [195]:
```
from sklearn.metrics import mean_squared_log_error

n_folds = 5

def cross_validation(y, y_pred):
    kf = KFold(n_folds, shuffle=True, random_state=42).get_n_splits
(train.values)
    rmsle= np.sqrt(mean_squared_log_error(y, y_pred))
    return(rmsle)
```

This is out main cell. Here we will use all the functions we previously defined to feed data, train and predict from our neural network. This cell is time consuming but if we reduce the number of epoch's the required time will be decreased. But greater number of epoch's will increase efiiciency significantly. Then will will predict the house prices of all houses and create the CSV file to output them.

In [196]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

train_p=('../input/house-prices-advanced-regression-techniques/trai
n.csv')
test_p=('../input/house-prices-advanced-regression-techniques/test.
csv')

train, test = load_data(train_p,test_p)

train_raw_labels = train['SalePrice'].to_frame().as_matrix()


train_pre = pre_process_data(train)
test_pre = pre_process_data(test)


train_pre = train_pre.drop(['Id', 'SalePrice'], axis=1)
test_pre = test_pre.drop(['Id'], axis=1)


train_pre, test_pre = train_pre.align(test_pre, join='outer', axis=
1)


train_pre.replace(to_replace=np.nan, value=0, inplace=True)
test_pre.replace(to_replace=np.nan, value=0, inplace=True)

train_pre = train_pre.as_matrix().astype(np.float)
test_pre = test_pre.as_matrix().astype(np.float)


standard_scaler = preprocessing.StandardScaler()
train_pre = standard_scaler.fit_transform(train_pre)
test_pre = standard_scaler.fit_transform(test_pre)

X_train, X_valid, Y_train, Y_valid = train_test_split(train_pre, tr
ain_raw_labels, test_size=0.3, random_state=1)
```

```
number of training examples = 1460
number of test examples = 1459
train shape: (1460, 81)
test shape: (1459, 80)
```

Cross Validation:

To determine the models performance we have performed a cross validation and determined the score using the cross validation function we defined before.

In [149]:
```
input_size = train_pre.shape[1]
output_size = 1
num_epochs = 3000
learning_rate = 0.001
layers_dims = [input_size, 512, 256, 64, output_size]

parameters, submission_name = model(X_train, Y_train, X_valid, Y_va
lid, layers_dims, num_epochs=num_epochs,
                                    learning_rate=learning_rate, pr
int_cost=False, plot_cost=False, l2_beta=15,
                                    keep_prob=0.7, minibatch_size=0
, return_best=False, evaluate=True)

prediction = list(map(lambda val: float(val), predict(train_pre, pa
rameters)))

data=pd.read_csv('../input/house-prices-advanced-regression-techniq
ues/train.csv')

price= data['SalePrice']

rmsle=cross_validation(price,prediction)

print('Deep Neural Network Model Score: ',rmsle)
```

```
Deep Neural Network Model Score:  0.10786376950578942
```

Hyper-Parameter Tuning:

- #### Number of Layers:
  There's one additional rule of thumb that helps for supervised learning problems. The upper bound on the number of hidden neurons that won't result in over-fitting is:

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$$

- N_i = number of input neurons.
- N_o = number of output neurons.
- N_h = Number of Hidden Layers.
- N_s = number of samples in training data set.
- α = an arbitrary scaling factor usually 2-10.
- The target here is to optimize the value of α. According to rule α must be between 1 to 10. So for different values of α the model was checked. The model performed better for α = 2 resulting N_h = 3

- #### Number of Neurons:

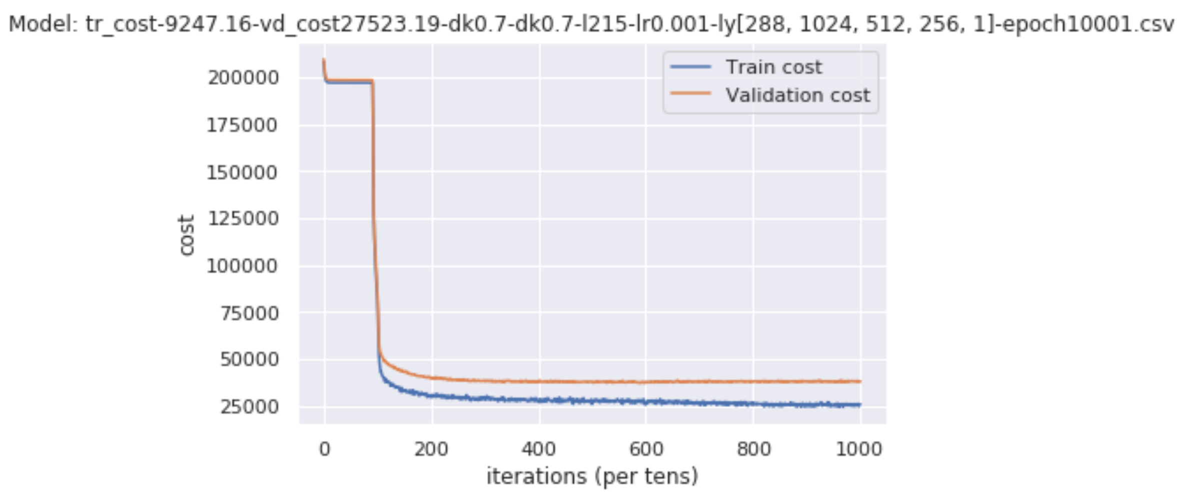
Number of Neurons was determinded by trial and error method. It is observed that increasing number of Neurons don't improve the models performance and the learning curve reflects that. So, by trial and error method the number of neurons is determinded (512,256,64) which performs the best. The input shape is 288 and output is 1. We will observe the learning curve to determine the number of neurons.

The source code is given below:

- Using Neuron (1024, 512, 256)

In [155]:
```
input_size = train_pre.shape[1]
output_size = 1
num_epochs = 10001
learning_rate = 0.001
layers_dims = [input_size, 1024, 512, 256, output_size]

parameters, submission_name = model(X_train, Y_train, X_valid, Y_va
lid, layers_dims, num_epochs=num_epochs,
                                    learning_rate=learning_rate, pr
int_cost=False, plot_cost=True, l2_beta=15,
                                    keep_prob=0.7, minibatch_size=0
, return_best=True, evaluate=False)
```

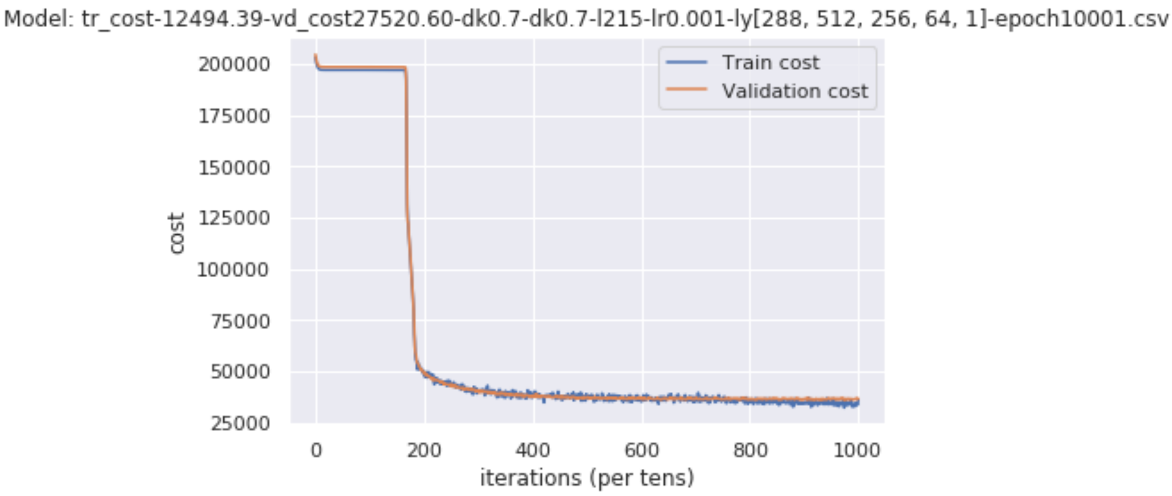
```
Parameters have been trained, getting metrics...
Train rmse: 9247.1603
Validation rmse: 27523.1918
Train rmsle: 0.0429
Validation rmsle: 0.1479
Lowest rmse: 37220.05 at epoch 5919
```



- Using Neuron (512, 256, 64)

In [161]:
```
input_size = train_pre.shape[1]
output_size = 1
num_epochs = 10001
learning_rate = 0.001
layers_dims = [input_size, 512, 256, 64, output_size]

parameters, submission_name = model(X_train, Y_train, X_valid, Y_va
lid, layers_dims, num_epochs=num_epochs,
                                    learning_rate=learning_rate, pr
int_cost=False, plot_cost=True, l2_beta=15,
                                    keep_prob=0.7, minibatch_size=0
, return_best=True, evaluate=False)
```

```
Parameters have been trained, getting metrics...
Train rmse: 12494.3935
Validation rmse: 27520.6008
Train rmsle: 0.0584
Validation rmsle: 0.1458
Lowest rmse: 35915.67 at epoch 9424
```

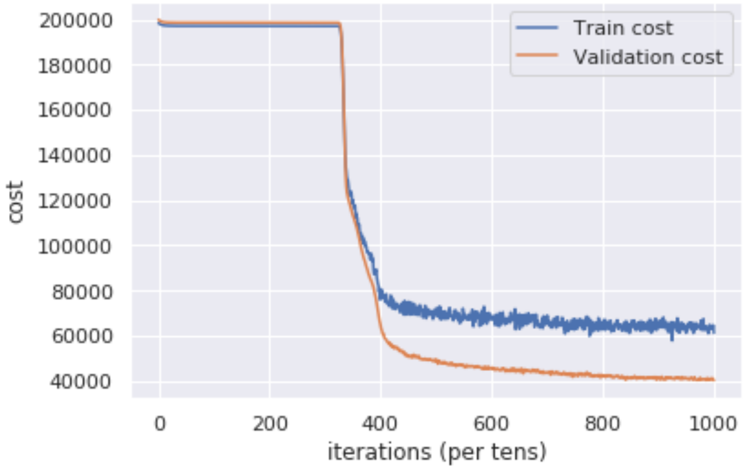


- Using Neuron (64, 32, 16)

In [167]:
```
input_size = train_pre.shape[1]
output_size = 1
num_epochs = 10001
learning_rate = 0.001
layers_dims = [input_size, 64, 32, 16, output_size]

parameters, submission_name = model(X_train, Y_train, X_valid, Y_va
lid, layers_dims, num_epochs=num_epochs,
                                    learning_rate=learning_rate, pr
int_cost=False, plot_cost=True, l2_beta=15,
                                    keep_prob=0.7, minibatch_size=0
, return_best=True, evaluate=False)
```

```
Parameters have been trained, getting metrics...
Train rmse: 22583.1513
Validation rmse: 32738.4378
Train rmsle: 0.1127
Validation rmsle: 0.1758
Lowest rmse: 40109.15 at epoch 9995
```

Model: tr_cost-22583.15-vd_cost32738.44-dk0.7-dk0.7-l215-lr0.001-ly[288, 64, 32, 16, 1]-epoch10001.csv



So, We can say that optimum Neuron is (512,256,64).

- #### Droupout and L2 Reghularization:

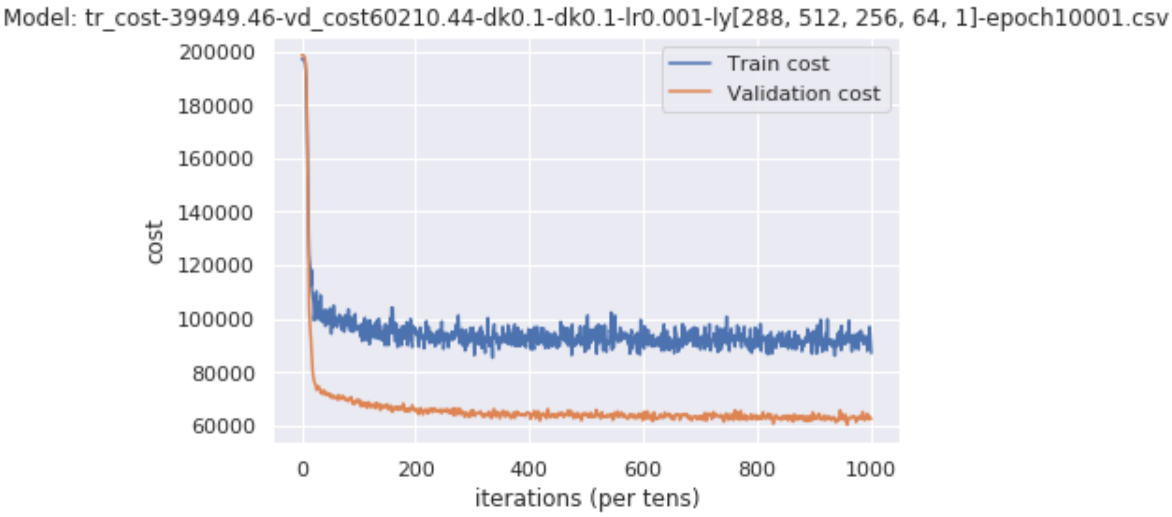
The dropout is controlled by the 'keep_prob' variable and L2 regularization is dependant on 'l2_beta' variable. Both of these was determinded using trial and error method. The optimum value for 'keep_prob' is found 0.7 and 'l2_beta' is 15. We determinded the best model by observing the learning curve. (Source: [https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/ (https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/)](https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/))

The source code for trial and error is given below:

- keep_prob=0.1
- l2_beta=0

In [173]:
```
input_size = train_pre.shape[1]
output_size = 1
num_epochs = 10001
learning_rate = 0.001
layers_dims = [input_size, 512, 256, 64, output_size]

parameters, submission_name = model(X_train, Y_train, X_valid, Y_va
lid, layers_dims, num_epochs=num_epochs,
                                    learning_rate=learning_rate, pr
int_cost=False, plot_cost=True, l2_beta=0,
                                    keep_prob=0.1, minibatch_size=0
, return_best=True, evaluate=False)
```

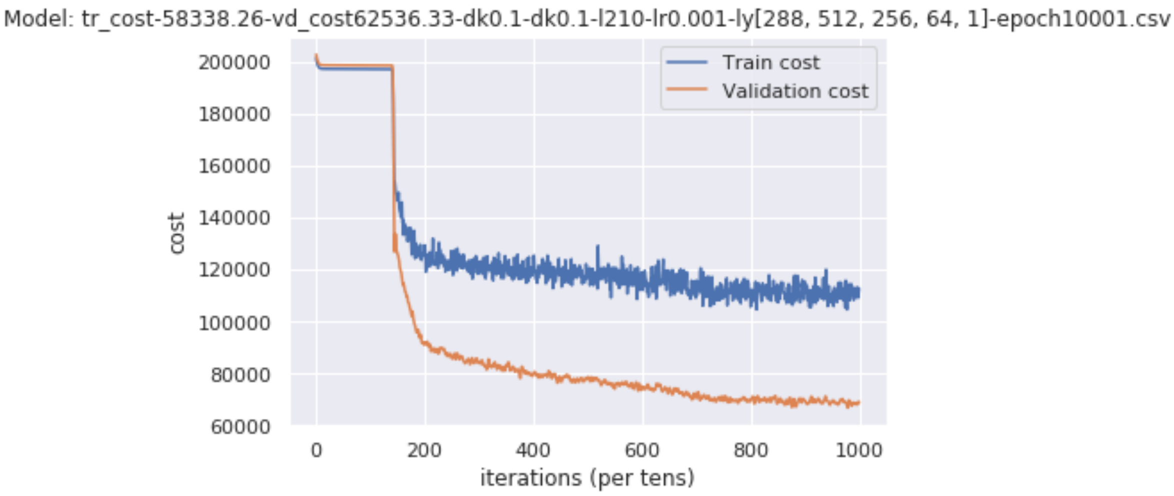
```
Parameters have been trained, getting metrics...
Train rmse: 39949.4591
Validation rmse: 60210.4375
Train rmsle: 0.2343
Validation rmsle: 0.5693
Lowest rmse: 60182.71 at epoch 9578
```



- keep_prob=0.1
- l2_beta=10

In [179]:
```
input_size = train_pre.shape[1]
output_size = 1
num_epochs = 10001
learning_rate = 0.001
layers_dims = [input_size, 512, 256, 64, output_size]

parameters, submission_name = model(X_train, Y_train, X_valid, Y_va
lid, layers_dims, num_epochs=num_epochs,
                                    learning_rate=learning_rate, pr
int_cost=False, plot_cost=True, l2_beta=10,
                                    keep_prob=0.1, minibatch_size=0
, return_best=True, evaluate=False)
```

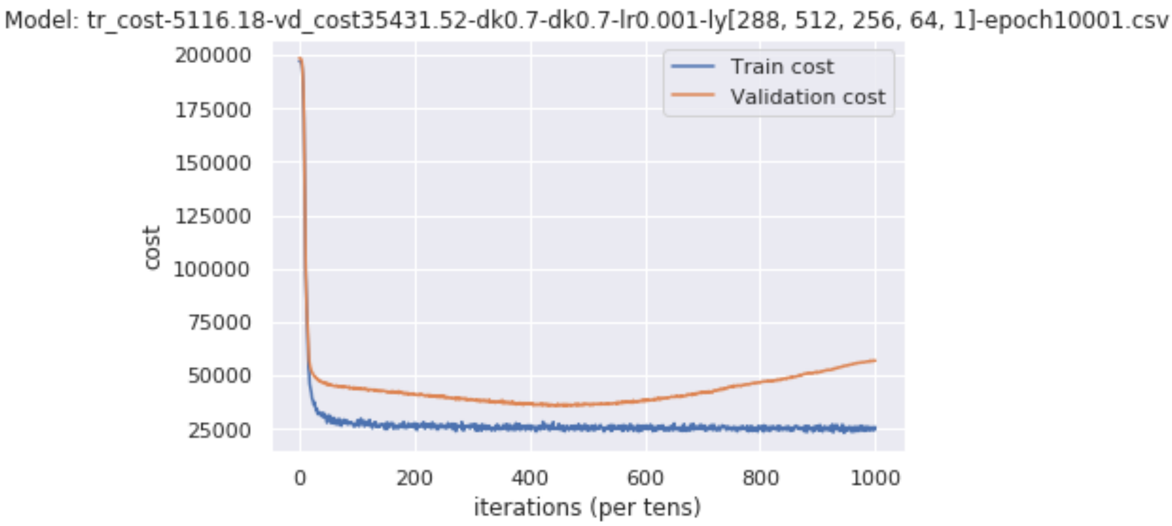
```
Parameters have been trained, getting metrics...
Train rmse: 58338.2628
Validation rmse: 62536.3288
Train rmsle: 0.3448
Validation rmsle: 0.3711
Lowest rmse: 66854.69 at epoch 8763
```



- keep_prob=0.7
- l2_beta=0

In [185]:
```
input_size = train_pre.shape[1]
output_size = 1
num_epochs = 10001
learning_rate = 0.001
layers_dims = [input_size, 512, 256, 64, output_size]

parameters, submission_name = model(X_train, Y_train, X_valid, Y_va
lid, layers_dims, num_epochs=num_epochs,
                                    learning_rate=learning_rate, pr
int_cost=False, plot_cost=True, l2_beta=0,
                                    keep_prob=0.7, minibatch_size=0
, return_best=True, evaluate=False)
```

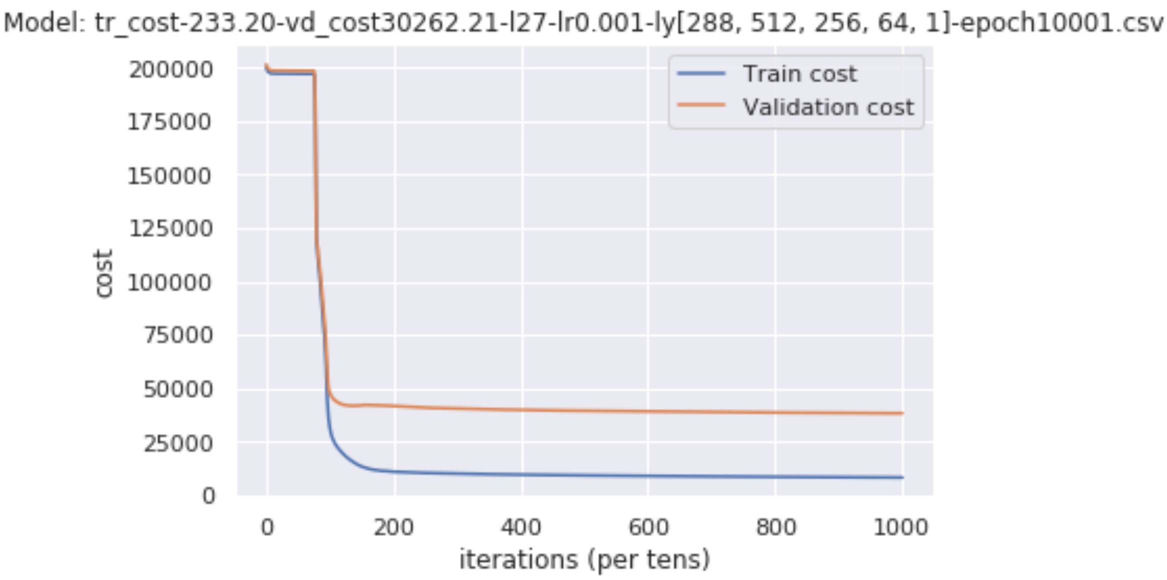
```
Parameters have been trained, getting metrics...
Train rmse: 5116.1812
Validation rmse: 35431.5189
Train rmsle: 0.0228
Validation rmsle: 0.2075
Lowest rmse: 35406.29 at epoch 4615
```



- keep_prob=1
- l2_beta=7

In [191]:
```
input_size = train_pre.shape[1]
output_size = 1
num_epochs = 10001
learning_rate = 0.001
layers_dims = [input_size, 512, 256, 64, output_size]

parameters, submission_name = model(X_train, Y_train, X_valid, Y_va
lid, layers_dims, num_epochs=num_epochs,
                                    learning_rate=learning_rate, pr
int_cost=False, plot_cost=True, l2_beta=7,
                                    keep_prob=1, minibatch_size=0,
return_best=True, evaluate=False)
```

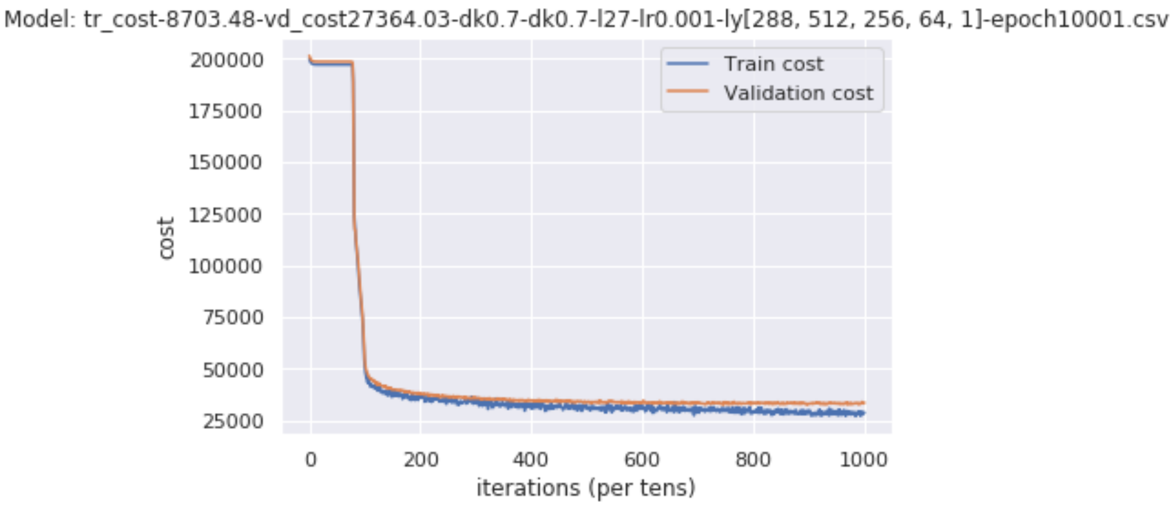
```
Parameters have been trained, getting metrics...
Train rmse: 233.2033
Validation rmse: 30262.2090
Train rmsle: 0.0014
Validation rmsle: 0.1690
Lowest rmse: 38313.16 at epoch 9996
```



- keep_prob=0.7
- l2_beta=7

In [197]:
```
input_size = train_pre.shape[1]
output_size = 1
num_epochs = 10001
learning_rate = 0.001
layers_dims = [input_size, 512, 256, 64, output_size]

parameters, submission_name = model(X_train, Y_train, X_valid, Y_va
lid, layers_dims, num_epochs=num_epochs,
                                    learning_rate=learning_rate, pr
int_cost=False, plot_cost=True, l2_beta=7,
                                    keep_prob=0.7, minibatch_size=0
, return_best=True, evaluate=False)
```

```
Parameters have been trained, getting metrics...
Train rmse: 8703.4829
Validation rmse: 27364.0315
Train rmsle: 0.0415
Validation rmsle: 0.1505
Lowest rmse: 32538.80 at epoch 7458
```



- keep_prob=0.7
- l2_beta=15

In [ ]:
```
input_size = train_pre.shape[1]
output_size = 1
num_epochs = 10001
learning_rate = 0.001
layers_dims = [input_size, 512, 256, 64, output_size]

parameters, submission_name = model(X_train, Y_train, X_valid, Y_va
lid, layers_dims, num_epochs=num_epochs,
                                    learning_rate=learning_rate, pr
int_cost=True, plot_cost=True, l2_beta=15,
                                    keep_prob=0.7, minibatch_size=0
, return_best=True, evaluate=False)
```

Here by the learning curve we can determine that using 'keep_prob=0.7' and 'l2_beta=15' we can get the best model.

Save the output to CSV and submission to Kaggle:

In [ ]:
```
prediction = list(map(lambda val: float(val), predict(test_pre, parameters)))

output_submission(test.Id.values, prediction, 'Id', 'SalePrice', submission_name)


submission = pd.DataFrame()

test_data=pd.read_csv('../input/house-prices-advanced-regression-techniques/test.csv')

submission['Id'] = test_data['Id']

submission['SalePrice'] = prediction

submission.to_csv('submission_DNN.csv' , index=False)
```

Summary of the approach:

Throughout this approach the main target was to utilize the tensorflow low level API to make the model creation robust and dynamic. Further changes can be made to make the model more robust. My main target was to make the model in a high level api (KERAS) style model creating system using low level API.

Observations:

1. The activation function 'ReLu' worked better than any other activation function.
2. The number of Neurons was determinded by trial and error method.
3. Adding L2 Regularization improves the performance.
4. 'l2_beta' variable is determinded by trial and error.
5. Tuning the droupout improves performance.
6. The droupout variable 'keep_prob' was determinded by trial and error.
7. 10000 epochs is the optimum number of epochs required to train the model.

Kaggle Score: 0.12039

Model Summary:

Model Summary

Layer	Type	Shape
Layer_1	Input	(288 , 512)
Activation_1	Relu	Layer_1
Dropout_1	Rate - 0.7	Layer_1
L2_Regularization_1	Beta - 15	Layer_1
Layer_2	Hidden	(512 , 256)
Activation_2	Relu	Layer_2
Dropout_2	Rate - 0.7	Layer_2
L2_Regularization_2	Beta - 15	Layer_2
Layer_3	Hidden	(256 , 64)
Activation_3	Relu	Layer_3
Dropout_1	Rate - 0.7	Layer_1
L2_Regularization_3	Beta - 15	Layer_3
Layer_4	Output	(64 , 1)
Activation_4	Relu	Layer_4

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission_DNN.csv	3 minutes ago	142 seconds	0 seconds	0.12039

Complete

Jump to your position on the leaderboard

Hyperparameters for Best Score using Deep Neural Network Model:

- Number of Layers: 3
- Minibatch Size: 0
- L2 Regularization Beta: 15
- Activation Function: Relu
- Learning Rate: 0.001 (RMSPropOptimizer)
- Epochs: 10001
- keep_prob: 0.7

These Parameters were found after several runs with parameter tuning as stated above.

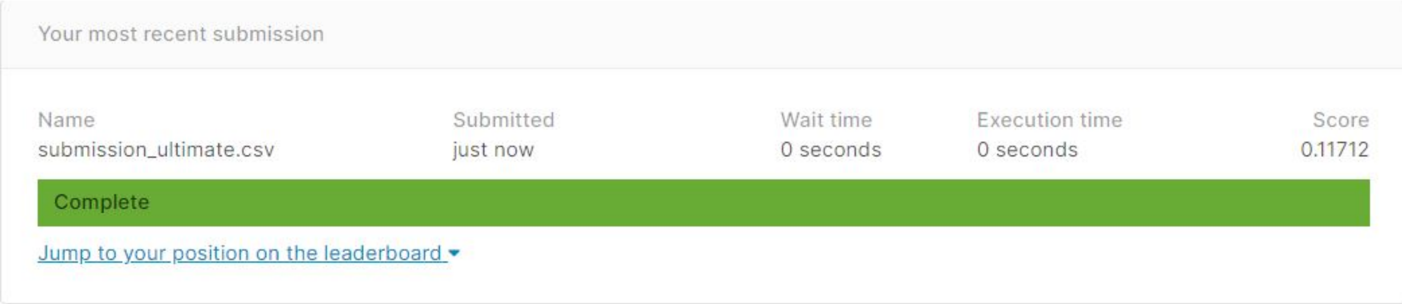
Ultimate Approach:

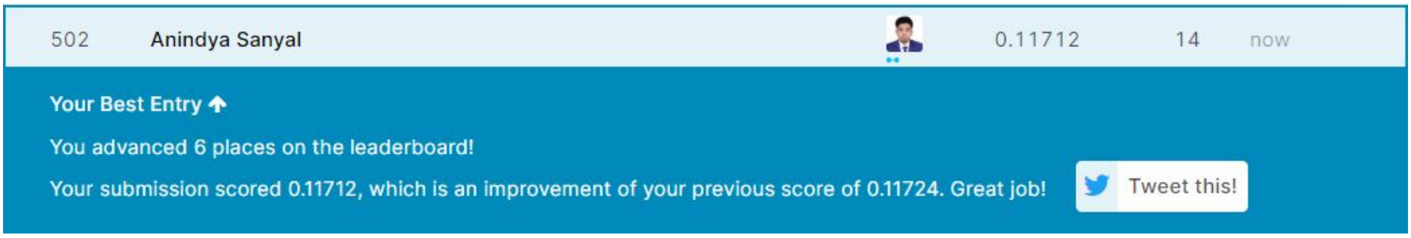
As we have seen in the upper sections, the averaged model performs best. So, here I will average all the TEN methods I have tried to determine the ultimate output.

Lets take an average of all the algorithoms to predict more accurate result! This will include all our previous 6 regression algorithm, averaged model, stacked model, ensemble approch and our deep learning model.

In [ ]:
```
submission = pd.DataFrame()

test_data=pd.read_csv('../input/house-prices-advanced-regression-techniques/test.csv')

submission['Id'] = test_data['Id']

submission['SalePrice'] = ((prediction + stacked_averaged_models_pred + ensemble + averaged_models_pred + lgb_pred + xgb_pred + GBoost_pred + ENet_pred + lasso_pred + KRR_pred)/10)

submission.to_csv('submission_ultimate.csv' , index=False)
```

Kaggle Score: 0.11712





Observation:

1. Averaging all the TEN models used before improves the performance significantly and gives the best Kaggle score too!

Summary Of The Report:

Here the primary and ultimate task was to determine house prices from the test dataset using 79 explanatory variables. The train dataset contained the 79 variables and the corresponding prices. To solve the problem at first I visualized the dataset, then I visualized the target variable 'SalePrice' using barplot, pairplot etc and calculated the correlation with other variables. Then I filled up the missing dataset and deleted data when there were too many missing data. Then I used six regression algorithms to calculate the house prices at first and then I made an avaerage model using those six methods. Then I also tried a stack model and ensemble model but the averaged model still performed better. Then I created my own deep neural network model to train the dataset and determine houseprices. I had to do Hyperparameter tuning to get accurate result.

Still I was not satisfied with my kaggle score. As averaged model performed best in earlier case, I decided to make an ultimate model where I averaged all TEN approches I made in previous cells. And this averaged model of TEN approaches gives the best result in Kaggle so I used this output as my final output.

Conclusion:

The best score achieved by Averaging model. The Deep Neural Network model performed quite well but using higher level API can make the code more robust and dynamic hence can bring good results. As the model was built from scratch, complexity arised. Adding dynamic regularization can also make the model more robust and bring good score.

Acknowledgement:

1. A Complete guide on Deep Learning with Python (https://www.analyticsvidhya.com/blog/2016/08/deep-learning-path/ (https://www.analyticsvidhya.com/blog/2016/08/deep-learning-path/))
2. House Price Prediction using Deep Learning (https://www.ijitee.org/wp-content/uploads/papers/v8i9/I7849078919.pdf (https://www.ijitee.org/wp-content/uploads/papers/v8i9/I7849078919.pdf))
3. Predict House Prices With Deep Learning (https://www.kaggle.com/yazeedalrubyli/predicting-houses-with-deep-learning (https://www.kaggle.com/yazeedalrubyli/predicting-houses-with-deep-learning))
4. How to use Learning Curves to Diagnose Machine Learning Model Performance (https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/ (https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/))
5. Skew and Kurtosis: 2 Important Statistics terms you need to know in Data Science (https://codeburst.io/2-important-statistics-terms-you-need-to-know-in-data-science-skewness-and-kurtosis-388fef94eeaa (https://codeburst.io/2-important-statistics-terms-you-need-to-know-in-data-science-skewness-and-kurtosis-388fef94eeaa))
6. Box-Cox Transformation (http://onlinestatbook.com/mobile/transformations/box-cox.html (http://onlinestatbook.com/mobile/transformations/box-cox.html))
7. Statistics and probability (https://www.khanacademy.org/math/statistics-probability (https://www.khanacademy.org/math/statistics-probability))
8. Comprehensive data exploration with Python (https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python (https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python))
9. Stacked Regressions (https://www.kaggle.com/serigne/stacked-regressions-top-4-on-leaderboard (https://www.kaggle.com/serigne/stacked-regressions-top-4-on-leaderboard))