Praktikum 5. EDA part 1

https://medium.com/data-folks-indonesia/memahami-data-dengan-exploratory-data-analysis-a53b230cce84

House Prices: EDA to ML

https://www.kaggle.com/code/dejavu23/house-prices-eda-to-ml-beginner/notebook

Table of Contents

Part 0 : Imports, Settings, Functions

Imports

In [1]:

```
import numpy as np
import pandas as pd
pd.set_option('max_columns', 105)
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
%matplotlib inline
sns.set()

import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
#warnings.filterwarnings("ignore")

from subprocess import check_output
print(check_output(["ls", "../input"]).decode("utf8"))
```

```
data_description.txt
sample_submission.csv
test.csv
train.csv
```

In [2]:

```
# setting the number of cross validations used in the Model part
nr_cv = 5

# switch for using log values for SalePrice and features
use_logvals = 1
# target used for correlation
target = 'SalePrice_Log'

# only columns with correlation above this threshold value
# are used for the ML Regressors in Part 3
min_val_corr = 0.4

# switch for dropping columns that are similar to others already used and show a high correlation to t
hese
drop_similar = 1
```

Some useful functions

In [3]:

```
def get_best_score(grid):

    best_score = np.sqrt(-grid.best_score_)
    print(best_score)
    print(grid.best_params_)
    print(grid.best_estimator_)

    return best_score
```

In [4]:

```
def print_cols_large_corr(df, nr_c, targ) :
    corr = df.corr()
    corr_abs = corr.abs()
    print (corr_abs.nlargest(nr_c, targ)[targ])
```

In [5]:

```
def plot_corr_matrix(df, nr_c, targ) :

    corr = df.corr()
    corr_abs = corr.abs()
    cols = corr_abs.nlargest(nr_c, targ)[targ].index
    cm = np.corrcoef(df[cols].values.T)

    plt.figure(figsize=(nr_c/1.5, nr_c/1.5))
    sns.set(font_scale=1.25)
    sns.heatmap(cm, linewidths=1.5, annot=True, square=True,
                fmt='.2f', annot_kws={'size': 10},
                yticklabels=cols.values, xticklabels=cols.values
               )
    plt.show()
```

Load data

In [6]:

```
df_train = pd.read_csv("../input/train.csv")
df_test = pd.read_csv("../input/test.csv")
```

Part 1: Exploratory Data Analysis

shape, info, head and describe

In [7]:

```
print(df_train.shape)
print("*"*50)
print(df_test.shape)
```

```
(1460, 81)
**************************************************
(1459, 80)
```

In [8]:

```
print(df_train.info())
print("*"*50)
print(df_test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
Id               1460 non-null int64
MSSubClass       1460 non-null int64
MSZoning         1460 non-null object
LotFrontage      1201 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null object
```

```
**************************************************
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 80 columns):
Id                 1459 non-null int64
MSSubClass         1459 non-null int64
MSZoning           1455 non-null object
LotFrontage        1232 non-null float64
LotArea            1459 non-null int64
Street             1459 non-null object
Alley              107 non-null object
LotShape           1459 non-null object
LandContour        1459 non-null object
Utilities          1457 non-null object
LotConfig          1459 non-null object
LandSlope          1459 non-null object
Neighborhood       1459 non-null object
Condition1         1459 non-null object
```

df train has 81 columns (79 features + id and target SalePrice) and 1460 entries (number of rows or house sales)

df test has 80 columns (79 features + id) and 1459 entries

There is lots of info that is probably related to the SalePrice like the area, the neighborhood, the condition and quality.

Maybe other features are not so important for predicting the target, also there might be a strong correlation for some of the features (like GarageCars and GarageArea).

For some columns many values are missing: only 7 values for Pool QC in df train and 3 in df test

In [9]:

```
df_train.head()
```

Out[9]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl

In [10]:

```
df_train.describe()
```

Out[10]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAd
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000

In [11]:

```
df_test.head()
```

Out[11]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlo
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	Inside	Gtl

In [12]:

```
df_test.describe()
```

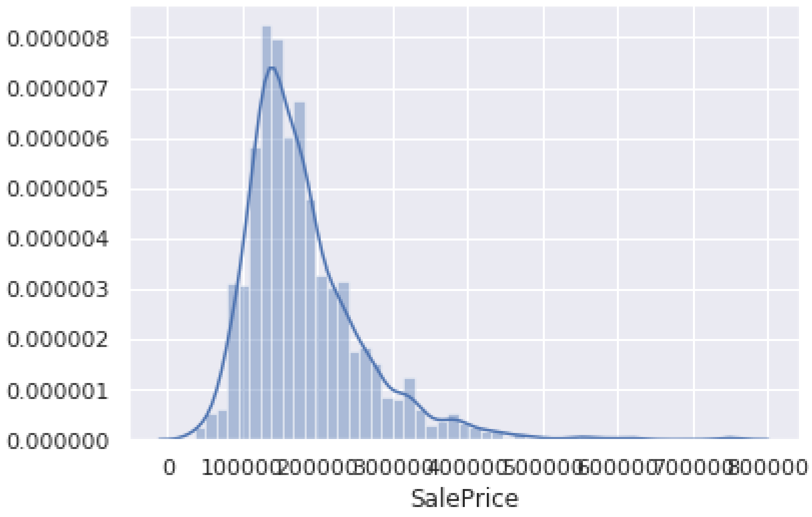
Out[12]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodA
count	1459.000000	1459.000000	1232.000000	1459.000000	1459.000000	1459.000000	1459.000000	1459.00000
mean	2190.000000	57.378341	68.580357	9819.161069	6.078821	5.553804	1971.357779	1983.66278
std	421.321334	42.746880	22.376841	4955.517327	1.436812	1.113740	30.390071	21.130467
min	1461.000000	20.000000	21.000000	1470.000000	1.000000	1.000000	1879.000000	1950.00000
25%	1825.500000	20.000000	58.000000	7391.000000	5.000000	5.000000	1953.000000	1963.00000
50%	2190.000000	50.000000	67.000000	9399.000000	6.000000	5.000000	1973.000000	1992.00000
75%	2554.500000	70.000000	80.000000	11517.500000	7.000000	6.000000	2001.000000	2004.00000

The target variable : Distribution of SalePrice

In [13]:
```
sns.distplot(df_train['SalePrice']);
#skewness and kurtosis
print("Skewness: %f" % df_train['SalePrice'].skew())
print("Kurtosis: %f" % df_train['SalePrice'].kurt())
```

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



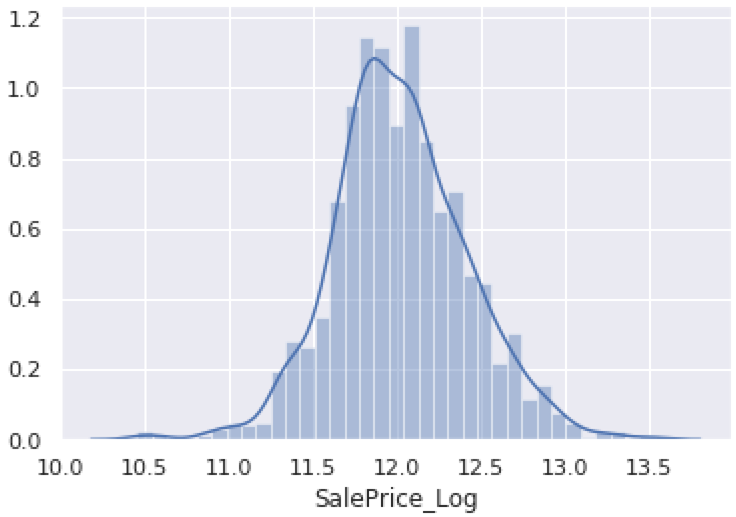
As we see, the target variable SalePrice is not normally distributed.
This can reduce the performance of the ML regression models because some assume normal distribution,
see sklearn info on preprocessing

Therefor we make a log transformation, the resulting distribution looks much better.

In [14]:

```
df_train['SalePrice_Log'] = np.log(df_train['SalePrice'])

sns.distplot(df_train['SalePrice_Log']);
# skewness and kurtosis
print("Skewness: %f" % df_train['SalePrice_Log'].skew())
print("Kurtosis: %f" % df_train['SalePrice_Log'].kurt())
# dropping old column
df_train.drop('SalePrice', axis= 1, inplace=True)
```

```
Skewness: 0.121335
Kurtosis: 0.809532
```



Numerical and Categorical features

In [15]:

```
numerical_feats = df_train.dtypes[df_train.dtypes != "object"].index
print("Number of Numerical features: ", len(numerical_feats))

categorical_feats = df_train.dtypes[df_train.dtypes == "object"].index
print("Number of Categorical features: ", len(categorical_feats))
```

```
Number of Numerical features:  38
Number of Categorical features:  43
```

In [16]:

```
print(df_train[numerical_feats].columns)
print("*"*100)
print(df_train[categorical_feats].columns)
```

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

In [17]:

```
df_train[numerical_feats].head()
```

Out[17]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtF
0	1	60	65.0	8450	7	5	2003	2003	196.0	706	0
1	2	20	80.0	9600	6	8	1976	1976	0.0	978	0
2	3	60	68.0	11250	7	5	2001	2002	162.0	486	0
3	4	70	60.0	9550	7	5	1915	1970	0.0	216	0
4	5	60	84.0	14260	8	5	2000	2000	350.0	655	0

In [18]:

```
df_train[categorical_feats].head()
```

Out[18]:

	MSZoning	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2
0	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm
1	RL	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr	Norm
2	RL	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm
3	RL	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm	Norm
4	RL	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm	Norm

List of features with missing values

In [19]:

```
total = df_train.isnull().sum().sort_values(ascending=False)
percent = (df_train.isnull().sum()/df_train.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
```

Out[19]:

	Total	Percent
PoolQC	1453	0.995205
MiscFeature	1406	0.963014
Alley	1369	0.937671
Fence	1179	0.807534
FireplaceQu	690	0.472603
LotFrontage	259	0.177397
GarageCond	81	0.055479
GarageType	81	0.055479
GarageYrBlt	81	0.055479
GarageFinish	81	0.055479

Filling missing values

For a few columns there is lots of NaN entries.

However, reading the data description we find this is not missing data:

For PoolQC, NaN is not missing data but means no pool, likewise for Fence, FireplaceQu etc.

In [20]:

```
# columns where NaN values have meaning e.g. no pool etc.
cols_fillna = ['PoolQC','MiscFeature','Alley','Fence','MasVnrType','FireplaceQu',
               'GarageQual','GarageCond','GarageFinish','GarageType', 'Electrical',
               'KitchenQual', 'SaleType', 'Functional', 'Exterior2nd', 'Exterior1st',
               'BsmtExposure','BsmtCond','BsmtQual','BsmtFinType1','BsmtFinType2',
               'MSZoning', 'Utilities']

# replace 'NaN' with 'None' in these columns
for col in cols_fillna:
    df_train[col].fillna('None',inplace=True)
    df_test[col].fillna('None',inplace=True)
```

In [21]:

```
total = df_train.isnull().sum().sort_values(ascending=False)
percent = (df_train.isnull().sum()/df_train.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(5)
```

Out[21]:

	Total	Percent
LotFrontage	259	0.177397
GarageYrBlt	81	0.055479
MasVnrArea	8	0.005479
SalePrice_Log	0	0.000000
ExterCond	0	0.000000

In [22]:

```
# fillna with mean for the remaining columns: LotFrontage, GarageYrBlt, MasVnrArea
df_train.fillna(df_train.mean(), inplace=True)
df_test.fillna(df_test.mean(), inplace=True)
```

In [23]:

```
total = df_train.isnull().sum().sort_values(ascending=False)
percent = (df_train.isnull().sum()/df_train.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(5)
```

Out[23]:

	Total	Percent
SalePrice_Log	0	0.0
Heating	0	0.0
RoofStyle	0	0.0
RoofMatl	0	0.0
Exterior1st	0	0.0

Missing values in train data ?

In [24]:
```
df_train.isnull().sum().sum()
```

Out[24]:
```
0
```

Missing values in test data ?

In [25]:
```
df_test.isnull().sum().sum()
```

Out[25]:
```
0
```

log transform

Like the target variable, also some of the feature values are not normally distributed and it is therefore better to use log values in df_train and df_test. Checking for skewness and kurtosis:

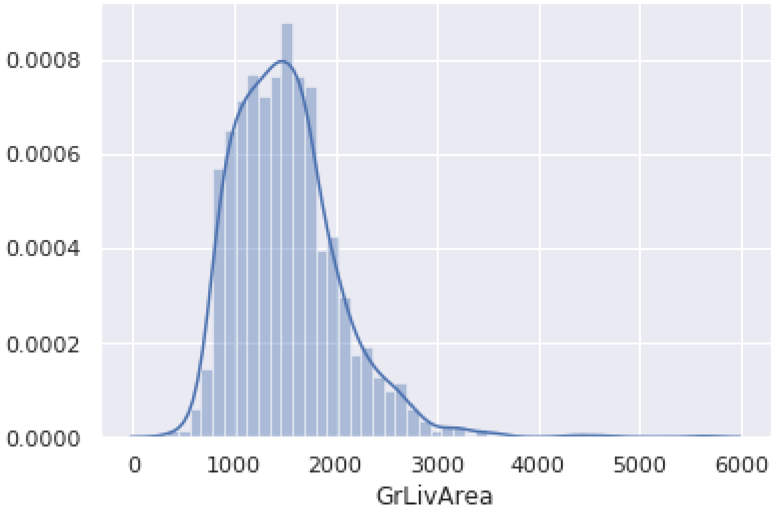
In [26]:
```
for col in numerical_feats:
    print('{:15}'.format(col),
          'Skewness: {:05.2f}'.format(df_train[col].skew()) ,
          '    ' ,
          'Kurtosis: {:06.2f}'.format(df_train[col].kurt())
         )
```

```
Id              Skewness: 00.00     Kurtosis: -01.20
MSSubClass      Skewness: 01.41     Kurtosis: 001.58
LotFrontage     Skewness: 02.38     Kurtosis: 021.85
LotArea         Skewness: 12.21     Kurtosis: 203.24
OverallQual     Skewness: 00.22     Kurtosis: 000.10
OverallCond     Skewness: 00.69     Kurtosis: 001.11
YearBuilt       Skewness: -0.61     Kurtosis: -00.44
YearRemodAdd    Skewness: -0.50     Kurtosis: -01.27
MasVnrArea      Skewness: 02.68     Kurtosis: 010.15
BsmtFinSF1      Skewness: 01.69     Kurtosis: 011.12
BsmtFinSF2      Skewness: 04.26     Kurtosis: 020.11
BsmtUnfSF       Skewness: 00.92     Kurtosis: 000.47
TotalBsmtSF     Skewness: 01.52     Kurtosis: 013.25
1stFlrSF        Skewness: 01.38     Kurtosis: 005.75
2ndFlrSF        Skewness: 00.81     Kurtosis: -00.55
LowQualFinSF    Skewness: 09.01     Kurtosis: 083.23
GrLivArea       Skewness: 01.37     Kurtosis: 004.90
BsmtFullBath    Skewness: 00.60     Kurtosis: -00.84
```

In [27]:

```python
sns.distplot(df_train['GrLivArea']);
#skewness and kurtosis
print("Skewness: %f" % df_train['GrLivArea'].skew())
print("Kurtosis: %f" % df_train['GrLivArea'].kurt())
```

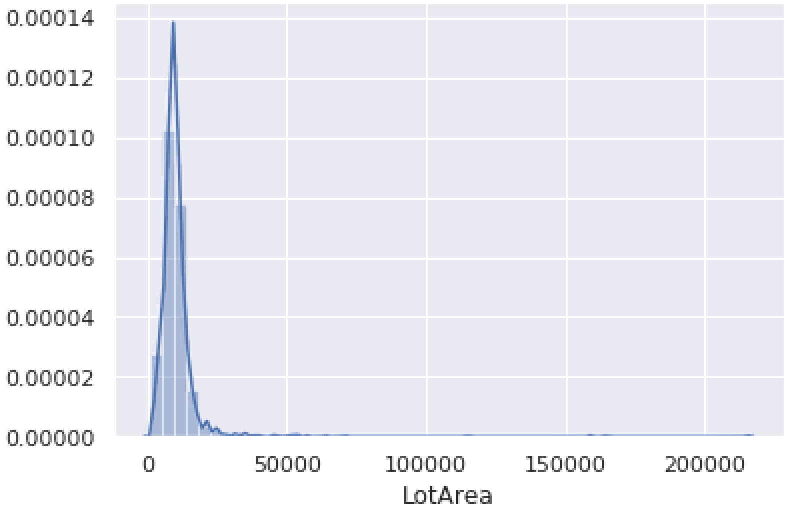
```
Skewness: 1.366560
Kurtosis: 4.895121
```



In [28]:

```python
sns.distplot(df_train['LotArea']);
#skewness and kurtosis
print("Skewness: %f" % df_train['LotArea'].skew())
print("Kurtosis: %f" % df_train['LotArea'].kurt())
```

```
Skewness: 12.207688
Kurtosis: 203.243271
```



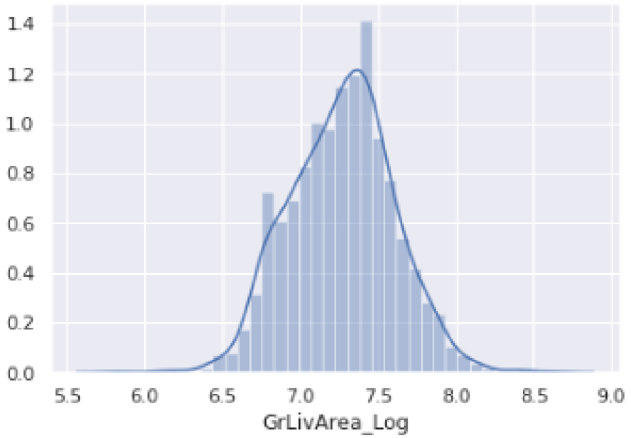
In [29]:

```
for df in [df_train, df_test]:
    df['GrLivArea_Log'] = np.log(df['GrLivArea'])
    df.drop('GrLivArea', inplace= True, axis = 1)
    df['LotArea_Log'] = np.log(df['LotArea'])
    df.drop('LotArea', inplace= True, axis = 1)



numerical_feats = df_train.dtypes[df_train.dtypes != "object"].index
```

In [30]:

```
sns.distplot(df_train['GrLivArea_Log']);
#skewness and kurtosis
print("Skewness: %f" % df_train['GrLivArea_Log'].skew())
print("Kurtosis: %f" % df_train['GrLivArea_Log'].kurt())
```

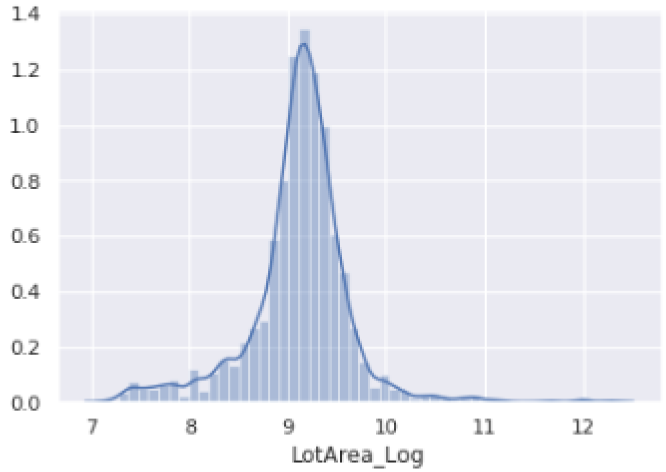
```
Skewness: -0.006995
Kurtosis: 0.282603
```



In [31]:

```
sns.distplot(df_train['LotArea_Log']);
#skewness and kurtosis
print("Skewness: %f" % df_train['LotArea_Log'].skew())
print("Kurtosis: %f" % df_train['LotArea_Log'].kurt())
```

```
Skewness: -0.137994
Kurtosis: 4.713358
```

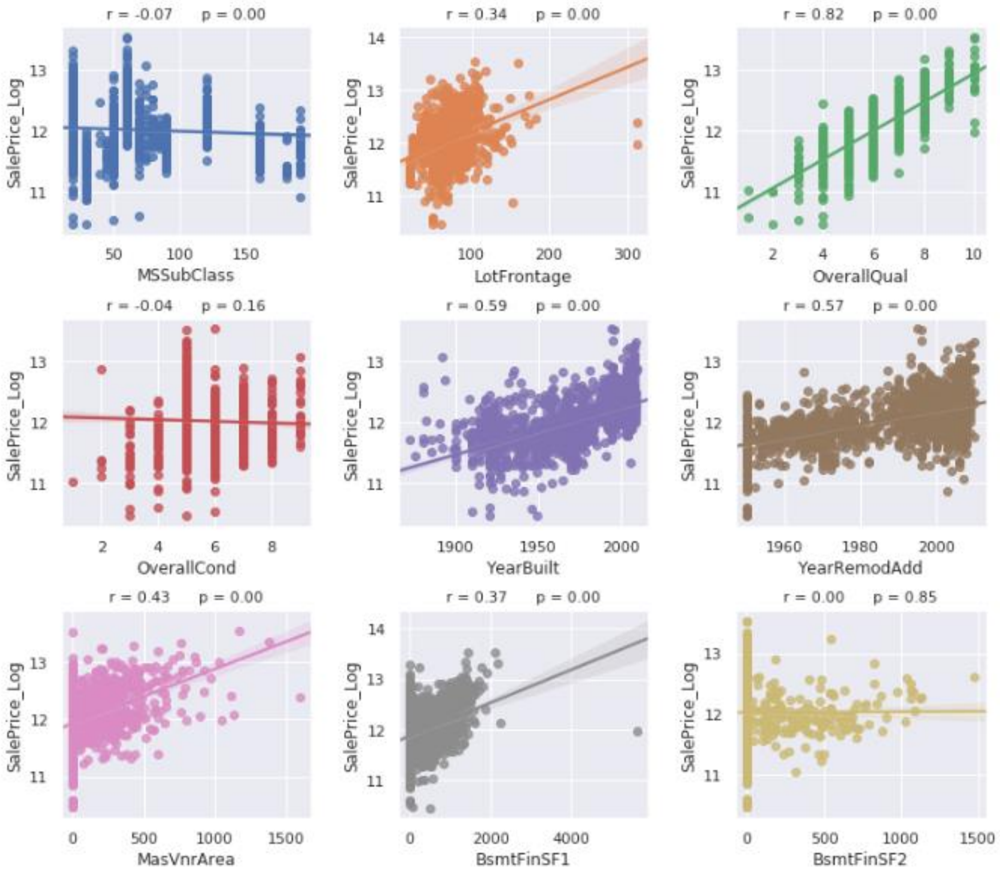


1.2 Relation of features to target (SalePrice_log)

Plots of relation to target for all numerical features

In [32]:

```
nr_rows = 12
nr_cols = 3

fig, axs = plt.subplots(nr_rows, nr_cols, figsize=(nr_cols*3.5,nr_rows*3))

li_num_feats = list(numerical_feats)
li_not_plot = ['Id', 'SalePrice', 'SalePrice_Log']
li_plot_num_feats = [c for c in list(numerical_feats) if c not in li_not_plot]


for r in range(0,nr_rows):
    for c in range(0,nr_cols):
        i = r*nr_cols+c
        if i < len(li_plot_num_feats):
            sns.regplot(df_train[li_plot_num_feats[i]], df_train[target], ax = axs[r][c])
            stp = stats.pearsonr(df_train[li_plot_num_feats[i]], df_train[target])
            #axs[r][c].text(0.4,0.9,"title",fontsize=7)
            str_title = "r = " + "{0:.2f}".format(stp[0]) + "      " "p = " + "{0:.2f}".format(stp
[1])
            axs[r][c].set_title(str_title,fontsize=11)

plt.tight_layout()
plt.show()
```



Conclusion from EDA on numerical columns:

1. We see that for some features like 'OverallQual' there is a strong linear correlation (0.79) to the target.
   For other features like 'MSSubClass' the correlation is very weak.
   For this kernel I decided to use only those features for prediction that have a correlation larger than a threshold value to SalePrice.
   This threshold value can be choosen in the global settings : min_val_corr

2. With the default threshold for min_val_corr = 0.4, these features are dropped in Part 2, Data Wrangling:
   'Id', 'MSSubClass', 'LotArea', 'OverallCond', 'BsmtFinSF2', 'BsmtUnfSF', 'LowQualFinSF', 'BsmtFullBath', 'BsmtHalfBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold'

3. We also see that the entries for some of the numerical columns are in fact categorical values.
   For example, the numbers for 'OverallQual' and 'MSSubClass' represent a certain group for that feature (see data description txt)

Outliers

In [33]:

```
df_train = df_train.drop(
    df_train[(df_train['OverallQual']==10) & (df_train['SalePrice_Log']<12.3)].index)
```

In [34]:

```
df_train = df_train.drop(
    df_train[(df_train['GrLivArea_Log']>8.3) & (df_train['SalePrice_Log']<12.5)].index)
```

Find columns with strong correlation to target
Only those with r > min_val_corr are used in the ML Regressors in Part 3
The value for min_val_corr can be chosen in global settings

List of numerical features and their correlation coefficient to target

In [35]:
```
corr = df_train.corr()
corr_abs = corr.abs()

nr_num_cols = len(numerical_feats)
ser_corr = corr_abs.nlargest(nr_num_cols, target)[target]

cols_abv_corr_limit = list(ser_corr[ser_corr.values > min_val_corr].index)
cols_bel_corr_limit = list(ser_corr[ser_corr.values <= min_val_corr].index)
```

In [36]:
```
print(ser_corr)
print("*"*30)
print("List of numerical features with r above min_val_corr :")
print(cols_abv_corr_limit)
print("*"*30)
print("List of numerical features with r below min_val_corr :")
print(cols_bel_corr_limit)
```

```
SalePrice_Log     1.000000
OverallQual       0.821404
GrLivArea_Log     0.737427
GarageCars        0.681033
GarageArea        0.656128
TotalBsmtSF       0.647563
1stFlrSF          0.620500
FullBath          0.595899
YearBuilt         0.587043
YearRemodAdd      0.565992
TotRmsAbvGrd      0.537702
GarageYrBlt       0.500842
Fireplaces        0.491998
MasVnrArea        0.433353
LotArea_Log       0.402814
BsmtFinSF1        0.392283
LotFrontage       0.352432
WoodDeckSF        0.334250
```

List of categorical features and their unique values

In [37]:

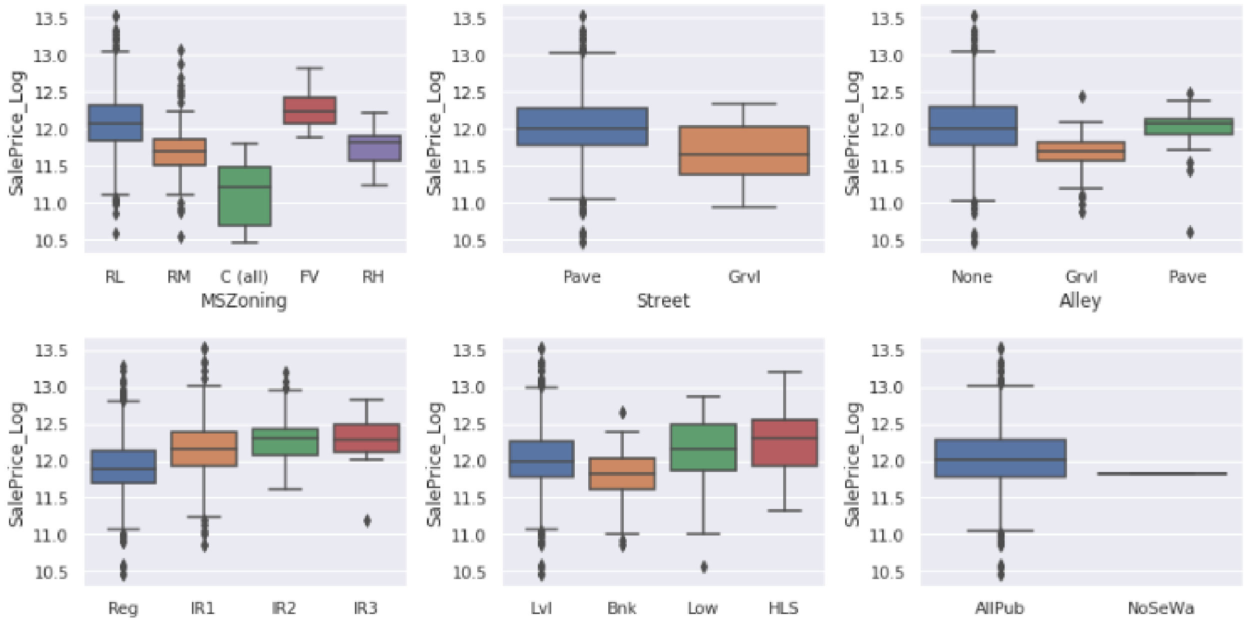
```
for catg in list(categorical_feats) :
    print(df_train[catg].value_counts())
    print('#'*50)
```

```
RL         1149
RM          218
FV           65
RH           16
C (all)      10
Name: MSZoning, dtype: int64
##################################################
Pave    1452
Grvl       6
Name: Street, dtype: int64
##################################################
None    1367
Grvl      50
Pave      41
Name: Alley, dtype: int64
##################################################
Reg    925
IR1    483
IR2     41
IR3      9
Name: LotShape, dtype: int64
##################################################
```

Relation to SalePrice for all categorical features

In [38]:

```
li_cat_feats = list(categorical_feats)
nr_rows = 15
nr_cols = 3

fig, axs = plt.subplots(nr_rows, nr_cols, figsize=(nr_cols*4,nr_rows*3))

for r in range(0,nr_rows):
    for c in range(0,nr_cols):
        i = r*nr_cols+c
        if i < len(li_cat_feats):
            sns.boxplot(x=li_cat_feats[i], y=target, data=df_train, ax = axs[r][c])

plt.tight_layout()
plt.show()
```



Conclusion from EDA on categorical columns:

1. For many of the categorical there is no strong relation to the target.
   However, for some fetaures it is easy to find a strong relation.
   From the figures above these are : 'MSZoning', 'Neighborhood', 'Condition2', 'MasVnrType', 'ExterQual', 'BsmtQual','CentralAir', 'Electrical', 'KitchenQual', 'SaleType'

Also for the categorical features, I use only those that show a strong relation to SalePrice.
So the other columns are dropped when creating the ML dataframes in Part 2 :

'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope', 'Condition1', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'ExterCond', 'Foundation', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Heating', 'HeatingQC', 'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature', 'SaleCondition'

In [39]:

```
catg_strong_corr = [ 'MSZoning', 'Neighborhood', 'Condition2', 'MasVnrType', 'ExterQual',
                     'BsmtQual','CentralAir', 'Electrical', 'KitchenQual', 'SaleType']

catg_weak_corr = ['Street', 'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
                  'LandSlope', 'Condition1',  'BldgType', 'HouseStyle', 'RoofStyle',
                  'RoofMatl', 'Exterior1st', 'Exterior2nd', 'ExterCond', 'Foundation',
                  'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Heating',
                  'HeatingQC', 'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish',
                  'GarageQual', 'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature',
                  'SaleCondition' ]
```

Correlation matrix 1

Features with largest correlation to SalePrice_Log

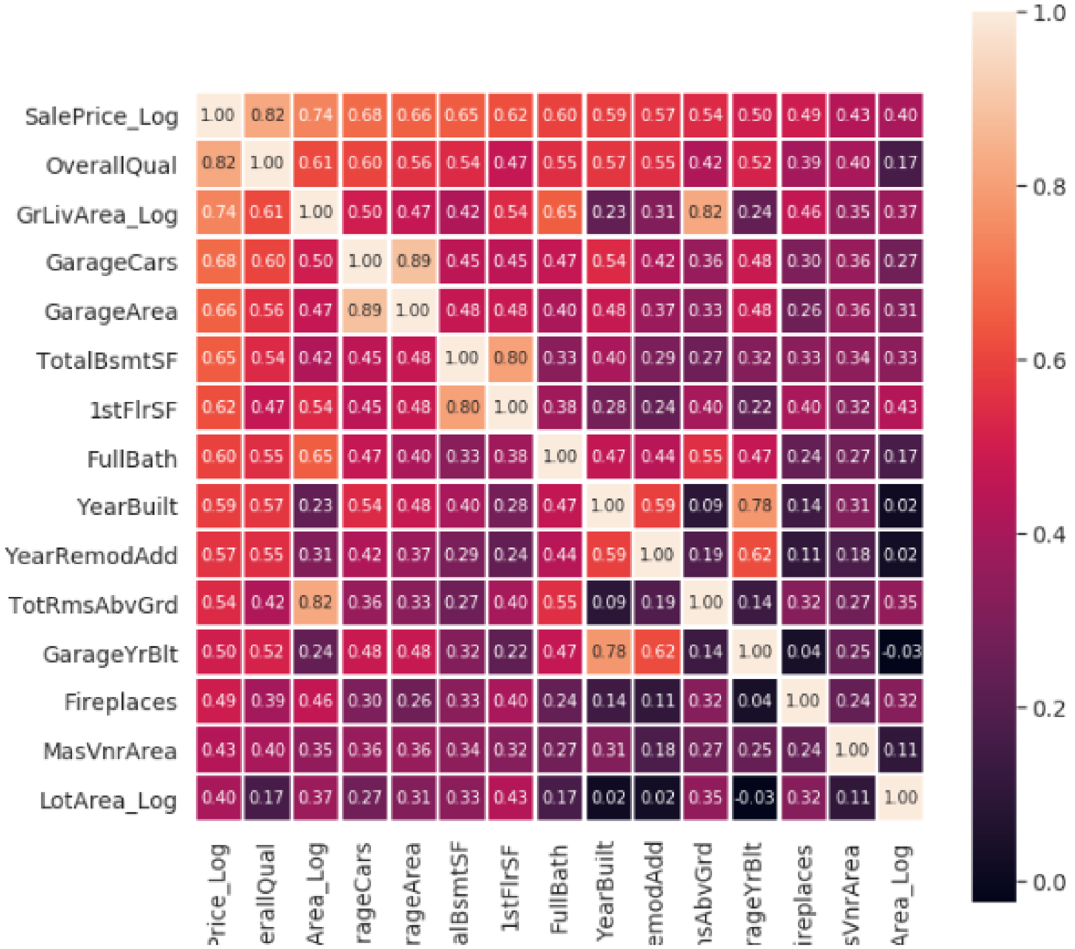
all numerical features with correlation coefficient above threshold

In [40]:

```
nr_feats = len(cols_abv_corr_limit)
```

In [41]:

```
plot_corr_matrix(df_train, nr_feats, target)
```



Of those features with the largest correlation to SalePrice, some also are correlated strongly to each other.

To avoid failures of the ML regression models due to multicollinearity, these are dropped in part 2.

This is optional and controlled by the switch drop_similar (global settings)