House Prices

It includes the following approaches and techniques:

- EDA with Pandas and Seaborn
- Find features with strong correlation to target
- Data Wrangling, convert categorical to numerical
- apply the basic Regression models of sklearn
- use gridsearchCV to find the best parameters for each model
- compare the performance of the Regressors and choose best one

Part 0 : Imports, Settings, Functions

Imports

In [ ]:

```
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")
```

Settings and switches

Here one can choose settings for optimal performance and runtime.
For example, nr_cv sets the number of cross validations used in GridsearchCV, and
min_val_corr is the minimum value for the correlation coefficient to the target (only features with larger correlation will be used).

In [ ]:

```
# 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 these
drop_similar = 1
```

Some useful functions

In [ ]:

```
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 [ ]:

```
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 [ ]:

```
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 [ ]:

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

Part 1: Exploratory Data Analysis

1.1 Overview of features and relation to target

Let's get a first overview of the train and test dataset
How many rows and columns are there?
What are the names of the features (columns)?
Which features are numerical, which are categorical?
How many values are missing?

The **shape** and **info** methods answer these questions

head displays some rows of the dataset

describe gives a summary of the statistics (only for numerical columns)

shape, info, head and describe

In [ ]:

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

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

In [ ]:

```
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):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1460 non-null   int64  
 1   MSSubClass     1460 non-null   int64  
 2   MSZoning       1460 non-null   object 
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64  
 5   Street         1460 non-null   object 
 6   Alley          91 non-null     object 
 7   LotShape       1460 non-null   object 
 8   LandContour    1460 non-null   object 
 9   Utilities      1460 non-null   object 
 10  LotConfig      1460 non-null   object 
 11  LandSlope      1460 non-null   object 
 12  Neighborhood   1460 non-null   object 
 13  Condition1     1460 non-null   object 
 14  Condition2     1460 non-null   object 
 15  BldgType       1460 non-null   object 
 16  HouseStyle     1460 non-null   object 
 17  OverallQual    1460 non-null   int64  
 18  OverallCond    1460 non-null   int64  
 19  YearBuilt      1460 non-null   int64  
 20  YearRemodAdd   1460 non-null   int64  
 21  RoofStyle      1460 non-null   object 
 22  RoofMatl       1460 non-null   object 
 23  Exterior1st    1460 non-null   object 
 24  Exterior2nd    1460 non-null   object 
 25  MasVnrType     1452 non-null   object 
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object 
 28  ExterCond      1460 non-null   object 
 29  Foundation     1460 non-null   object 
 30  BsmtQual       1423 non-null   object 
 31  BsmtCond       1423 non-null   object 
 32  BsmtExposure   1422 non-null   object 
 33  BsmtFinType1   1423 non-null   object 
 34  BsmtFinSF1     1460 non-null   int64  
 35  BsmtFinType2   1422 non-null   object 
 36  BsmtFinSF2     1460 non-null   int64  
 37  BsmtUnfSF      1460 non-null   int64  
 38  TotalBsmtSF    1460 non-null   int64  
 39  Heating        1460 non-null   object 
 40  HeatingQC      1460 non-null   object 
 41  CentralAir     1460 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1460 non-null   int64  
 44  2ndFlrSF       1460 non-null   int64  
 45  LowQualFinSF   1460 non-null   int64  
 46  GrLivArea      1460 non-null   int64  
 47  BsmtFullBath   1460 non-null   int64  
 48  BsmtHalfBath   1460 non-null   int64  
 49  FullBath       1460 non-null   int64  
 50  HalfBath       1460 non-null   int64  
 51  BedroomAbvGr   1460 non-null   int64  
 52  KitchenAbvGr   1460 non-null   int64  
 53  KitchenQual    1460 non-null   object 
 54  TotRmsAbvGrd   1460 non-null   int64  
 55  Functional     1460 non-null   object 
 56  Fireplaces     1460 non-null   int64  
 57  FireplaceQu    770 non-null    object 
 58  GarageType     1379 non-null   object 
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object 
 61  GarageCars     1460 non-null   int64  
 62  GarageArea     1460 non-null   int64  
 63  GarageQual     1379 non-null   object 
 64  GarageCond     1379 non-null   object 
 65  PavedDrive     1460 non-null   object 
 66  WoodDeckSF     1460 non-null   int64  
 67  OpenPorchSF    1460 non-null   int64  
 68  EnclosedPorch  1460 non-null   int64  
 69  3SsnPorch      1460 non-null   int64  
 70  ScreenPorch    1460 non-null   int64  
 71  PoolArea       1460 non-null   int64  
 72  PoolQC         7 non-null      object 
 73  Fence          281 non-null    object 
 74  MiscFeature    54 non-null     object 
 75  MiscVal        1460 non-null   int64  
 76  MoSold         1460 non-null   int64  
 77  YrSold         1460 non-null   int64  
 78  SaleType       1460 non-null   object 
 79  SaleCondition  1460 non-null   object 
 80  SalePrice      1460 non-null   int64  
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
None
**************************************************
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 80 columns):
```

```
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1459 non-null   int64  
 1   MSSubClass     1459 non-null   int64  
 2   MSZoning       1455 non-null   object 
 3   LotFrontage    1232 non-null   float64
 4   LotArea        1459 non-null   int64  
 5   Street         1459 non-null   object 
 6   Alley          107 non-null    object 
 7   LotShape       1459 non-null   object 
 8   LandContour    1459 non-null   object 
 9   Utilities      1457 non-null   object 
 10  LotConfig      1459 non-null   object 
 11  LandSlope      1459 non-null   object 
 12  Neighborhood   1459 non-null   object 
 13  Condition1     1459 non-null   object 
 14  Condition2     1459 non-null   object 
 15  BldgType       1459 non-null   object 
 16  HouseStyle     1459 non-null   object 
 17  OverallQual    1459 non-null   int64  
 18  OverallCond    1459 non-null   int64  
 19  YearBuilt      1459 non-null   int64  
 20  YearRemodAdd   1459 non-null   int64  
 21  RoofStyle      1459 non-null   object 
 22  RoofMatl       1459 non-null   object 
 23  Exterior1st    1458 non-null   object 
 24  Exterior2nd    1458 non-null   object 
 25  MasVnrType     1443 non-null   object 
 26  MasVnrArea     1444 non-null   float64
 27  ExterQual      1459 non-null   object 
 28  ExterCond      1459 non-null   object 
 29  Foundation     1459 non-null   object 
 30  BsmtQual       1415 non-null   object 
 31  BsmtCond       1414 non-null   object 
 32  BsmtExposure   1415 non-null   object 
 33  BsmtFinType1   1417 non-null   object 
 34  BsmtFinSF1     1458 non-null   float64
 35  BsmtFinType2   1417 non-null   object 
 36  BsmtFinSF2     1458 non-null   float64
 37  BsmtUnfSF      1458 non-null   float64
 38  TotalBsmtSF    1458 non-null   float64
 39  Heating        1459 non-null   object 
 40  HeatingQC      1459 non-null   object 
 41  CentralAir     1459 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1459 non-null   int64  
 44  2ndFlrSF       1459 non-null   int64  
 45  LowQualFinSF   1459 non-null   int64  
 46  GrLivArea      1459 non-null   int64  
 47  BsmtFullBath   1457 non-null   float64
 48  BsmtHalfBath   1457 non-null   float64
 49  FullBath       1459 non-null   int64  
 50  HalfBath       1459 non-null   int64  
 51  BedroomAbvGr   1459 non-null   int64  
 52  KitchenAbvGr   1459 non-null   int64  
 53  KitchenQual    1458 non-null   object 
 54  TotRmsAbvGrd   1459 non-null   int64  
 55  Functional     1457 non-null   object 
 56  Fireplaces     1459 non-null   int64  
 57  FireplaceQu    729 non-null    object 
 58  GarageType     1383 non-null   object 
 59  GarageYrBlt    1381 non-null   float64
 60  GarageFinish   1381 non-null   object 
 61  GarageCars     1458 non-null   float64
 62  GarageArea     1458 non-null   float64
 63  GarageQual     1381 non-null   object 
 64  GarageCond     1381 non-null   object 
 65  PavedDrive     1459 non-null   object 
 66  WoodDeckSF     1459 non-null   int64  
 67  OpenPorchSF    1459 non-null   int64  
 68  EnclosedPorch  1459 non-null   int64  
 69  3SsnPorch      1459 non-null   int64  
 70  ScreenPorch    1459 non-null   int64  
 71  PoolArea       1459 non-null   int64  
 72  PoolQC         3 non-null      object 
 73  Fence          290 non-null    object 
 74  MiscFeature    51 non-null     object 
 75  MiscVal        1459 non-null   int64  
 76  MoSold         1459 non-null   int64  
 77  YrSold         1459 non-null   int64  
 78  SaleType       1458 non-null   object 
 79  SaleCondition  1459 non-null   object 
dtypes: float64(11), int64(26), object(43)
memory usage: 912.0+ KB
None
```

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

Out[ ]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	HouseStyle	OverallQua
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam	2Story	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr	Norm	1Fam	1Story	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam	2Story	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm	Norm	1Fam	2Story	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm	Norm	1Fam	2Story	

In [ ]:
```
df_train.describe()
```

Out[ ]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.639726	46.549315	567.240411	1057.429452	1162.626712	
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.098091	161.319273	441.866955	438.705324	386.587738	
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	0.000000	0.000000	0.000000	334.000000	
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.000000	0.000000	223.000000	795.750000	882.000000	
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000	0.000000	477.500000	991.500000	1087.000000	
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.250000	0.000000	808.000000	1298.250000	1391.250000	
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000	2336.000000	6110.000000	4692.000000	2

In [ ]:

```
df_test.head()
```

Out[ ]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	HouseStyle	OverallC
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	NAmes	Feedr	Norm	1Fam	1Story	
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	NAmes	Norm	Norm	1Fam	1Story	
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	Norm	1Fam	2Story	
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	Norm	1Fam	2Story	
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	Inside	Gtl	StoneBr	Norm	Norm	TwnhsE	1Story	

In [ ]:

```
df_test.describe()
```

Out[ ]:

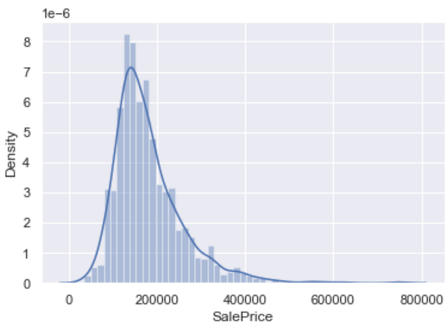
	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	
count	1459.000000	1459.000000	1232.000000	1459.000000	1459.000000	1459.000000	1459.000000	1459.000000	1444.000000	1458.000000	1458.000000	1458.000000	1458.000000	1459.000000	14
mean	2190.000000	57.378341	68.580357	9819.161069	6.078821	5.553804	1971.357779	1983.662783	100.709141	439.203704	52.619342	554.294925	1046.117970	1156.534613	3
std	421.321334	42.746880	22.376841	4955.517327	1.436812	1.113740	30.390071	21.130467	177.625900	455.268042	176.753926	437.260486	442.898624	398.165820	4
min	1461.000000	20.000000	21.000000	1470.000000	1.000000	1.000000	1879.000000	1950.000000	0.000000	0.000000	0.000000	0.000000	0.000000	407.000000	
25%	1825.500000	20.000000	58.000000	7391.000000	5.000000	5.000000	1953.000000	1963.000000	0.000000	0.000000	0.000000	219.250000	784.000000	873.500000	
50%	2190.000000	50.000000	67.000000	9399.000000	6.000000	5.000000	1973.000000	1992.000000	0.000000	350.500000	0.000000	460.000000	988.000000	1079.000000	
75%	2554.500000	70.000000	80.000000	11517.500000	7.000000	6.000000	2001.000000	2004.000000	164.000000	753.500000	0.000000	797.750000	1305.000000	1382.500000	6
max	2919.000000	190.000000	200.000000	56600.000000	10.000000	9.000000	2010.000000	2010.000000	1290.000000	4010.000000	1526.000000	2140.000000	5095.000000	5095.000000	18

The target variable : Distribution of SalePrice

In [ ]:

```
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
```

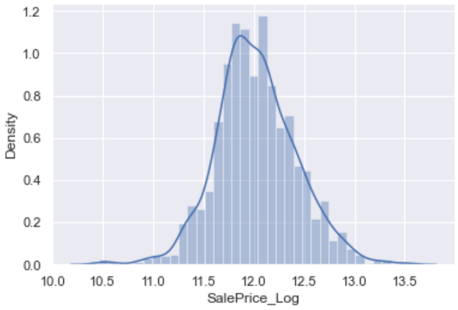


As we see, the target variable SalePrice is not normally distributed.
This can reduce the performance of the ML regression models because some assume normal distribution,
see [sklearn info on preprocessing](#)
Therfore we make a log transformation, the resulting distribution looks much better.

In [ ]:

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

Out[ ]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrL
0	1	60	65.0	8450	7	5	2003	2003	196.0	706	0	150	856	856	854	0	
1	2	20	80.0	9600	6	8	1976	1976	0.0	978	0	284	1262	1262	0	0	
2	3	60	68.0	11250	7	5	2001	2002	162.0	486	0	434	920	920	866	0	
3	4	70	60.0	9550	7	5	1915	1970	0.0	216	0	540	756	961	756	0	
4	5	60	84.0	14260	8	5	2000	2000	350.0	655	0	490	1145	1145	1053	0	

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

Out[ ]:

	MSZoning	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	HouseStyle	RoofStyle	RoofMatl	Exterior1st	Exterior2nd	Ma
0	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam	2Story	Gable	CompShg	VinylSd	VinylSd	
1	RL	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr	Norm	1Fam	1Story	Gable	CompShg	MetalSd	MetalSd	
2	RL	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam	2Story	Gable	CompShg	VinylSd	VinylSd	
3	RL	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm	Norm	1Fam	2Story	Gable	CompShg	Wd Sdng	Wd Shng	
4	RL	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm	Norm	1Fam	2Story	Gable	CompShg	VinylSd	VinylSd	

List of features with missing values

In [ ]:
```
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[ ]:

	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
GarageYrBlt	81	0.055479
GarageCond	81	0.055479
GarageType	81	0.055479
GarageFinish	81	0.055479

	Total	Percent
GarageQual	81	0.055479
BsmtFinType2	38	0.026027
BsmtExposure	38	0.026027
BsmtQual	37	0.025342
BsmtCond	37	0.025342
BsmtFinType1	37	0.025342
MasVnrArea	8	0.005479
MasVnrType	8	0.005479
Electrical	1	0.000685
Id	0	0.000000

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 [ ]:

```
# 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 [ ]:

```
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[ ]:

	Total	Percent
LotFrontage	259	0.177397
GarageYrBlt	81	0.055479
MasVnrArea	8	0.005479
Id	0	0.000000
KitchenAbvGr	0	0.000000

In [ ]:

```
# 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 [ ]:

```
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[ ]:

	Total	Percent
Id	0	0.0
CentralAir	0	0.0
GarageYrBlt	0	0.0
GarageType	0	0.0
FireplaceQu	0	0.0

Missing values in train data ?

In [ ]:

```
df_train.isnull().sum().sum()
```

Out[ ]:

```
0
```

Missing values in test data ?

In [ ]:

```
df_test.isnull().sum().sum()
```

Out[ ]:

```
0
```

In [ ]:

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 [ ]:

```
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())
         )
```

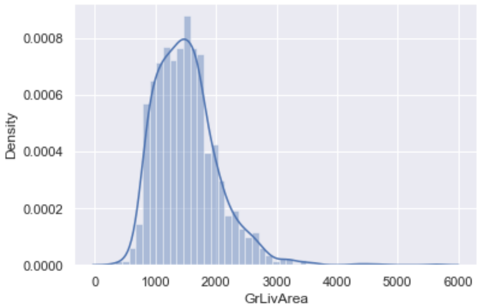
```
Id              Skewness: 00.00     Kurtosis: -01.20
MSSubClass      Skewness: 01.41     Kurtosis: 001.58
LotFrontage     Skewness: 02.38     Kurtosis: 021.85
```

```
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
BsmtHalfBath    Skewness: 04.10     Kurtosis: 016.40
FullBath        Skewness: 00.04     Kurtosis: -00.86
HalfBath        Skewness: 00.68     Kurtosis: -01.08
BedroomAbvGr    Skewness: 00.21     Kurtosis: 002.23
KitchenAbvGr    Skewness: 04.49     Kurtosis: 021.53
TotRmsAbvGrd    Skewness: 00.68     Kurtosis: 000.88
Fireplaces      Skewness: 00.65     Kurtosis: -00.22
GarageYrBlt     Skewness: -0.67     Kurtosis: -00.27
GarageCars      Skewness: -0.34     Kurtosis: 000.22
GarageArea      Skewness: 00.18     Kurtosis: 000.92
WoodDeckSF      Skewness: 01.54     Kurtosis: 002.99
OpenPorchSF     Skewness: 02.36     Kurtosis: 008.49
EnclosedPorch   Skewness: 03.09     Kurtosis: 010.43
3SsnPorch       Skewness: 10.30     Kurtosis: 123.66
ScreenPorch     Skewness: 04.12     Kurtosis: 018.44
PoolArea        Skewness: 14.83     Kurtosis: 223.27
MiscVal         Skewness: 24.48     Kurtosis: 701.00
MoSold          Skewness: 00.21     Kurtosis: -00.40
YrSold          Skewness: 00.10     Kurtosis: -01.19
SalePrice_Log   Skewness: 00.12     Kurtosis: 000.81
```

In [ ]:

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

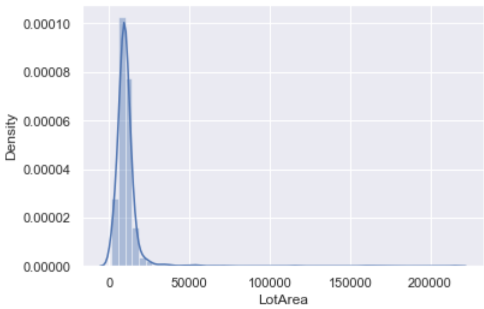
```
Skewness: 1.366560
Kurtosis: 4.895121
```



In [ ]:

```
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
```



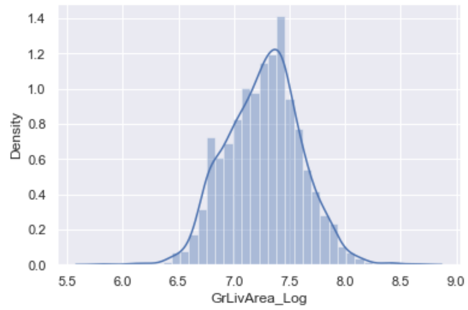
In [ ]:

```
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 [ ]:

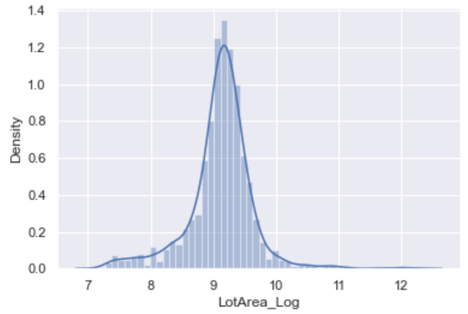
```
sns.distplot(df_train['GrLivArea_Log']);
#skewness and kurtosis
print("Skewness: %f" % df_train['GrLivArea_Log'].skew())
print("Kurtosis: %f" % df_train['GrLivArea_Log'].kurt())
```

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



In [ ]:
```
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())
```

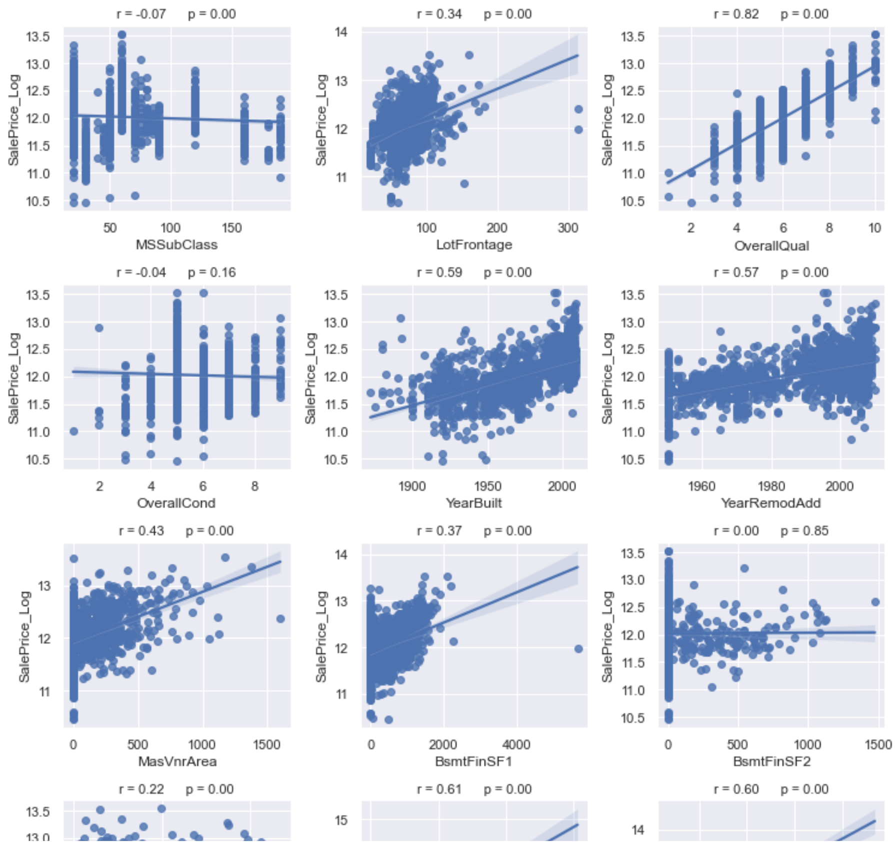
```
Skewness: -0.137994
Kurtosis: 4.713358
```

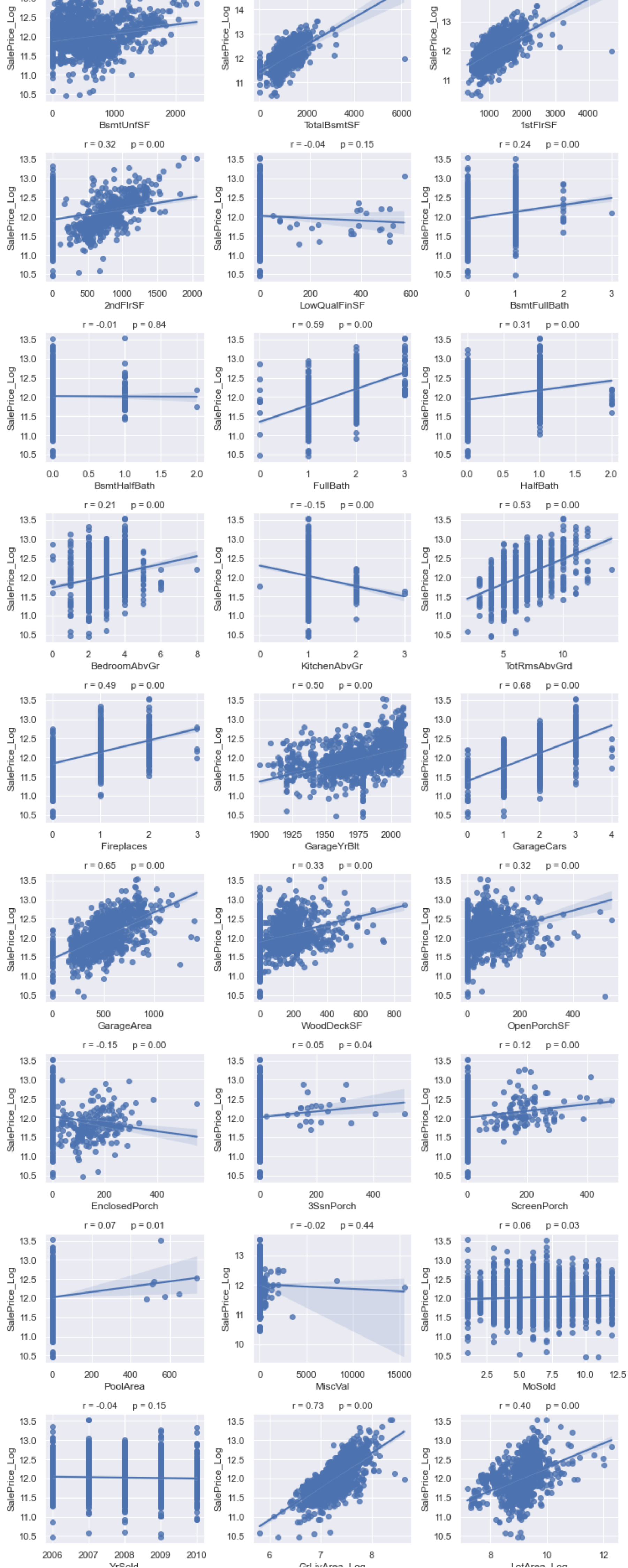


1.2 Relation of features to target (SalePrice_log)

Plots of relation to target for all numerical features

In [ ]:
```
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()
```





SalePrice_Log
BsmtUnfSF
TotalBsmtSF
1stFlrSF
r = 0.32 p = 0.00
2ndFlrSF
r = -0.04 p = 0.15
LowQualFinSF
r = 0.24 p = 0.00
BsmtFullBath
r = -0.01 p = 0.84
BsmtHalfBath
r = 0.59 p = 0.00
FullBath
r = 0.31 p = 0.00
HalfBath
r = 0.21 p = 0.00
BedroomAbvGr
r = -0.15 p = 0.00
KitchenAbvGr
r = 0.53 p = 0.00
TotRmsAbvGrd
r = 0.49 p = 0.00
Fireplaces
r = 0.50 p = 0.00
GarageYrBlt
r = 0.68 p = 0.00
GarageCars
r = 0.65 p = 0.00
GarageArea
r = 0.33 p = 0.00
WoodDeckSF
r = 0.32 p = 0.00
OpenPorchSF
r = -0.15 p = 0.00
EnclosedPorch
r = 0.05 p = 0.04
3SsnPorch
r = 0.12 p = 0.00
ScreenPorch
r = 0.07 p = 0.01
PoolArea
r = -0.02 p = 0.44
MiscVal
r = 0.06 p = 0.03
MoSold
r = -0.04 p = 0.15
YrSold
r = 0.73 p = 0.00
GrLivArea_Log
r = 0.40 p = 0.00
LotArea_Log

Conclusion from EDA on numerical columns:

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

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'

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

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

In [ ]:
```

```

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

In [ ]:
```
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)
```

List of numerical features and their correlation coefficient to target

In [ ]:
```
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
OpenPorchSF      0.325215
2ndFlrSF         0.319953
HalfBath         0.314186
BsmtFullBath     0.237099
BsmtUnfSF        0.221892
BedroomAbvGr     0.209036
EnclosedPorch    0.149029
KitchenAbvGr     0.147534
ScreenPorch      0.121245
PoolArea         0.074338
MSSubClass       0.073969
MoSold           0.057064
3SsnPorch        0.054914
LowQualFinSF     0.037951
YrSold           0.037151
OverallCond      0.036821
MiscVal          0.020012
Id               0.017774
BsmtHalfBath     0.005124
BsmtFinSF2       0.004863
Name: SalePrice_Log, dtype: float64
******************************
List of numerical features with r above min_val_corr :
['SalePrice_Log', 'OverallQual', 'GrLivArea_Log', 'GarageCars', 'GarageArea', 'TotalBsmtSF', '1stFlrSF', 'FullBath', 'YearBuilt', 'YearRemodAdd', 'TotRmsAbvGrd', 'Garage
YrBlt', 'Fireplaces', 'MasVnrArea', 'LotArea_Log']
******************************
List of numerical features with r below min_val_corr :
['BsmtFinSF1', 'LotFrontage', 'WoodDeckSF', 'OpenPorchSF', '2ndFlrSF', 'HalfBath', 'BsmtFullBath', 'BsmtUnfSF', 'BedroomAbvGr', 'EnclosedPorch', 'KitchenAbvGr', 'ScreenP
orch', 'PoolArea', 'MSSubClass', 'MoSold', '3SsnPorch', 'LowQualFinSF', 'YrSold', 'OverallCond', 'MiscVal', 'Id', 'BsmtHalfBath', 'BsmtFinSF2']
```

List of categorical features and their unique values

In [ ]:
```
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
##################################################
Lvl    1311
Bnk      61
HLS      50
Low      36
Name: LandContour, dtype: int64
##################################################
AllPub    1457
NoSeWa       1
Name: Utilities, dtype: int64
##################################################
Inside     1051
Corner      262
CulDSac      94
FR2          47
FR3           4
Name: LotConfig, dtype: int64
##################################################
Gtl    1380
Mod      65
Sev      13
Name: LandSlope, dtype: int64
##################################################
NAmes      225
CollgCr    150
OldTown    113
Edwards     98
Somerst     86
Gilbert     79
NridgHt     77
Sawyer      74
NWAmes      73
SawyerW     59
BrkSide     58
Crawfor     51
Mitchel     49
NoRidge     41
Timber      38
IDOTRR      37
ClearCr     28
StoneBr     25
SWISU       25
MeadowV     17
Blmngtn     17
BrDale      16
Veenker     11
NPkVill      9
Blueste      2
Name: Neighborhood, dtype: int64
##################################################
Norm      1260
Feedr       80
Artery      48
RRAn        26
PosN        18
RRAe        11
PosA         8
RRNn         5
RRNe         2
Name: Condition1, dtype: int64
##################################################
Norm      1444
Feedr        6
Artery       2
RRNn         2
PosA         1
PosN         1
RRAn         1
RRAe         1
Name: Condition2, dtype: int64
##################################################
1Fam      1218
TwnhsE     114
Duplex      52
Twnhs       43
2fmCon      31
Name: BldgType, dtype: int64
##################################################
1Story    726
2Story    443
1.5Fin    154
SLvl       65
SFoyer     37
1.5Unf     14
2.5Unf     11
2.5Fin      8
Name: HouseStyle, dtype: int64
##################################################
Gable      1141
Hip         284
Flat         13
Gambrel      11
Mansard       7
Shed          2
Name: RoofStyle, dtype: int64
```

```
####################################################
CompShg    1433
Tar&Grv      11
WdShngl       6
WdShake       5
Metal         1
Membran       1
Roll          1
Name: RoofMatl, dtype: int64
####################################################
VinylSd    515
HdBoard    222
MetalSd    220
Wd Sdng    206
Plywood    108
CemntBd     60
BrkFace     50
WdShing     26
Stucco      24
AsbShng     20
BrkComm      2
Stone        2
AsphShn      1
ImStucc      1
CBlock       1
Name: Exterior1st, dtype: int64
####################################################
VinylSd    504
MetalSd    214
HdBoard    207
Wd Sdng    197
Plywood    142
CmentBd     59
Wd Shng     38
BrkFace     25
Stucco      25
AsbShng     20
ImStucc     10
Brk Cmn      7
Stone        5
AsphShn      3
Other        1
CBlock       1
Name: Exterior2nd, dtype: int64
####################################################
None       872
BrkFace    445
Stone      126
BrkCmn      15
Name: MasVnrType, dtype: int64
####################################################
TA    906
Gd    488
Ex     50
Fa     14
Name: ExterQual, dtype: int64
####################################################
TA    1280
Gd     146
Fa      28
Ex       3
Po       1
Name: ExterCond, dtype: int64
####################################################
PConc     645
CBlock    634
BrkTil    146
Slab       24
Stone       6
Wood        3
Name: Foundation, dtype: int64
####################################################
TA      649
Gd      618
Ex      119
None     37
Fa       35
Name: BsmtQual, dtype: int64
####################################################
TA      1309
Gd        65
Fa        45
None      37
Po         2
Name: BsmtCond, dtype: int64
####################################################
No      953
Av      221
Gd      132
Mn      114
None     38
Name: BsmtExposure, dtype: int64
####################################################
Unf     430
GLQ     416
ALQ     220
BLQ     148
Rec     133
LwQ      74
None     37
Name: BsmtFinType1, dtype: int64
####################################################
Unf     1254
Rec       54
LwQ       46
None      38
BLQ       33
ALQ       19
GLQ       14
Name: BsmtFinType2, dtype: int64
####################################################
GasA     1426
GasW       18
Grav        7
Wall        4
```

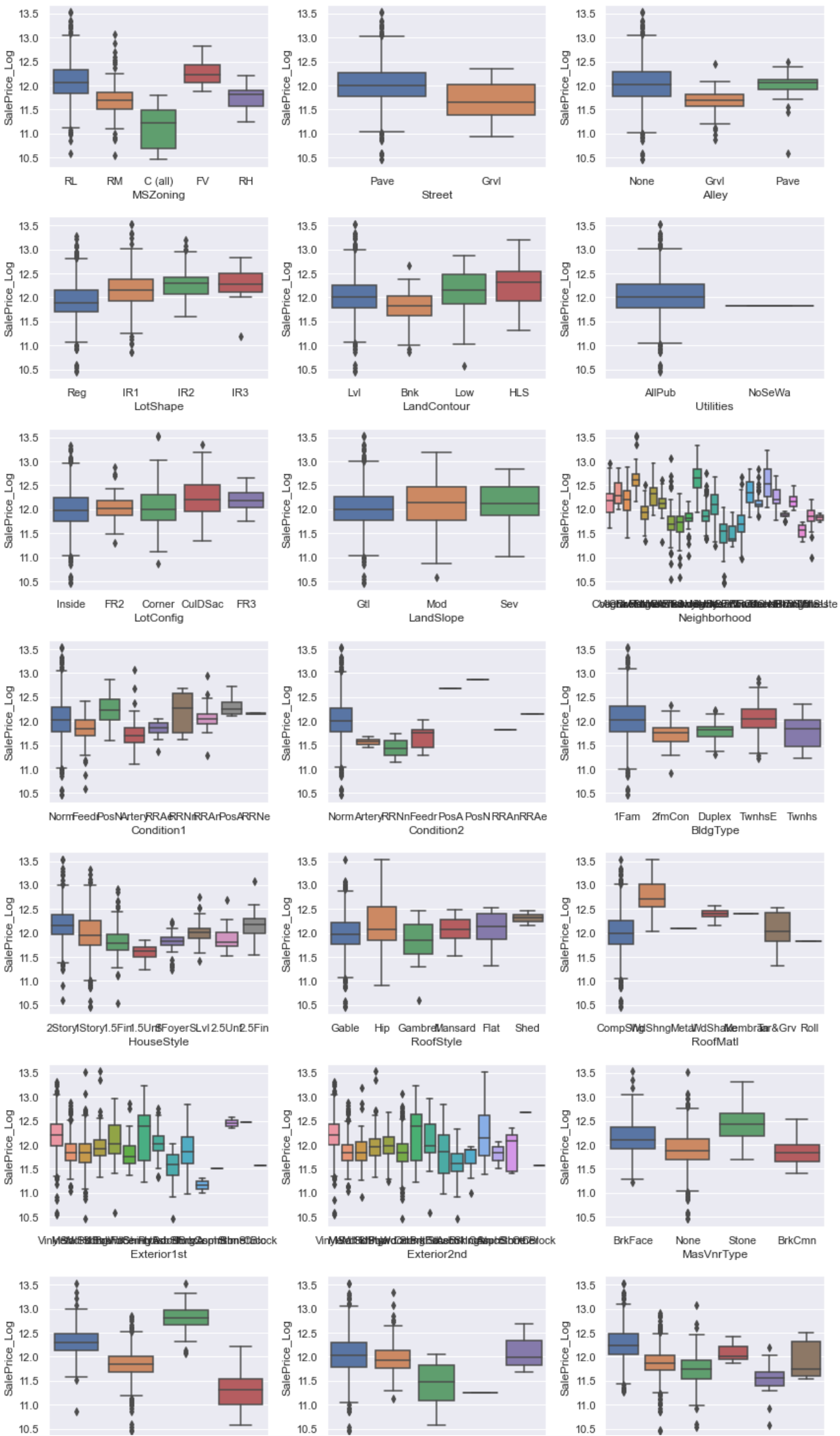
```
OthW        2
Floor       1
Name: Heating, dtype: int64
##################################################
Ex    739
TA    428
Gd    241
Fa     49
Po      1
Name: HeatingQC, dtype: int64
##################################################
Y    1363
N      95
Name: CentralAir, dtype: int64
##################################################
SBrkr    1332
FuseA      94
FuseF      27
FuseP       3
Mix         1
None        1
Name: Electrical, dtype: int64
##################################################
TA    735
Gd    586
Ex     98
Fa     39
Name: KitchenQual, dtype: int64
##################################################
Typ     1358
Min2      34
Min1      31
Mod       15
Maj1      14
Maj2       5
Sev        1
Name: Functional, dtype: int64
##################################################
None    690
Gd      378
TA      313
Fa       33
Ex       24
Po       20
Name: FireplaceQu, dtype: int64
##################################################
Attchd     869
Detchd     387
BuiltIn     87
None        81
Basment     19
CarPort      9
2Types       6
Name: GarageType, dtype: int64
##################################################
Unf     605
RFn     422
Fin     350
None     81
Name: GarageFinish, dtype: int64
##################################################
TA      1309
None      81
Fa        48
Gd        14
Ex         3
Po         3
Name: GarageQual, dtype: int64
##################################################
TA      1324
None      81
Fa        35
Gd         9
Po         7
Ex         2
Name: GarageCond, dtype: int64
##################################################
Y    1338
N      90
P      30
Name: PavedDrive, dtype: int64
##################################################
None    1452
Ex         2
Fa         2
Gd         2
Name: PoolQC, dtype: int64
##################################################
None     1177
MnPrv     157
GdPrv      59
GdWo       54
MnWw       11
Name: Fence, dtype: int64
##################################################
None    1404
Shed      49
Gar2       2
Othr       2
TenC       1
Name: MiscFeature, dtype: int64
##################################################
WD       1267
New       120
COD        43
ConLD       9
ConLI       5
ConLw       5
CWD         4
Oth         3
Con         2
Name: SaleType, dtype: int64
##################################################
Normal     1198
Partial     123
```

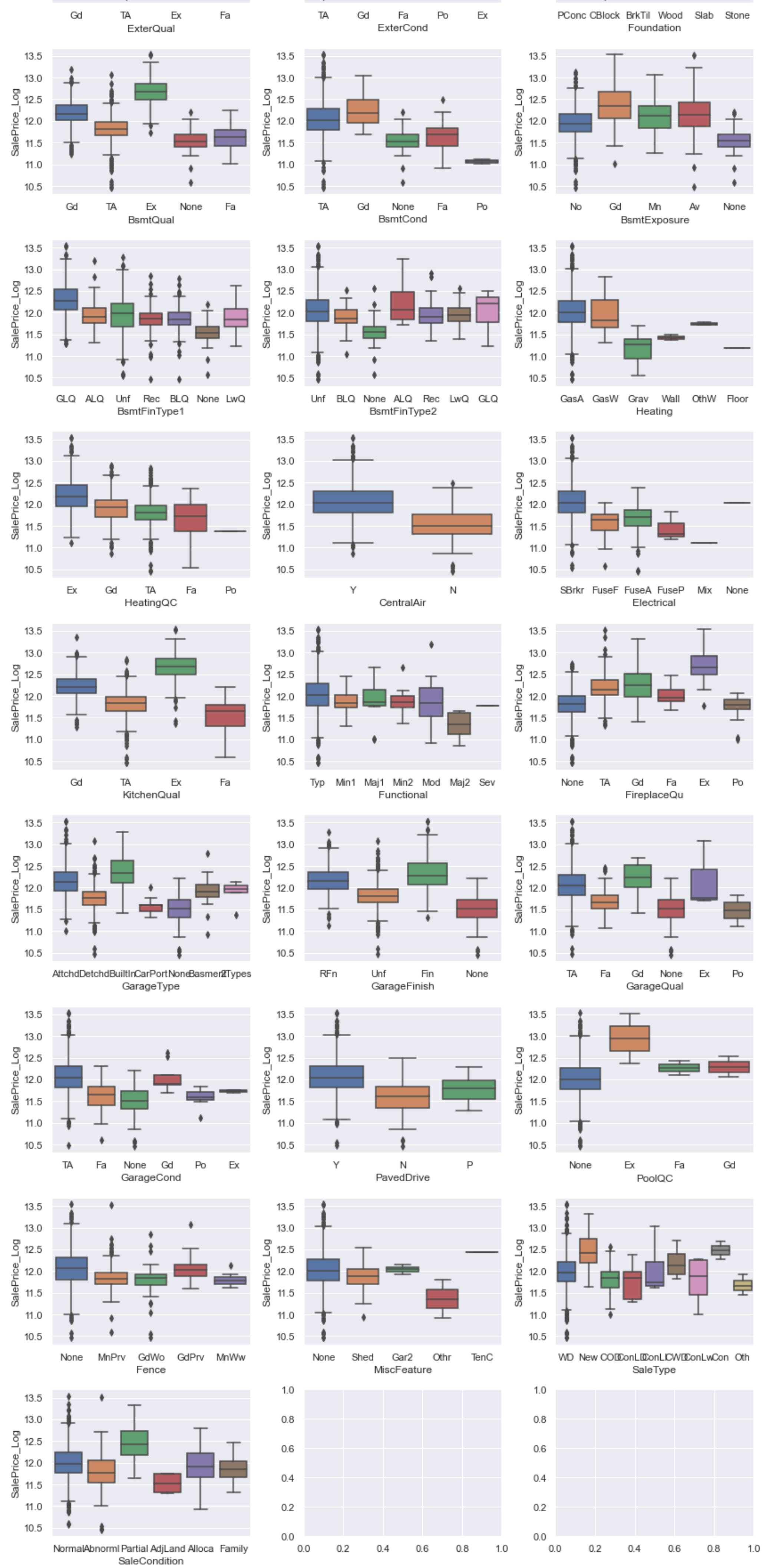
```
Abnorml     101
Family       20
Alloca       12
AdjLand       4
Name: SaleCondition, dtype: int64
#######################################################
```

Relation to SalePrice for all categorical features

In [ ]:

```
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:

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 [ ]:
```
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' ]
```

In [ ]:

In [ ]:

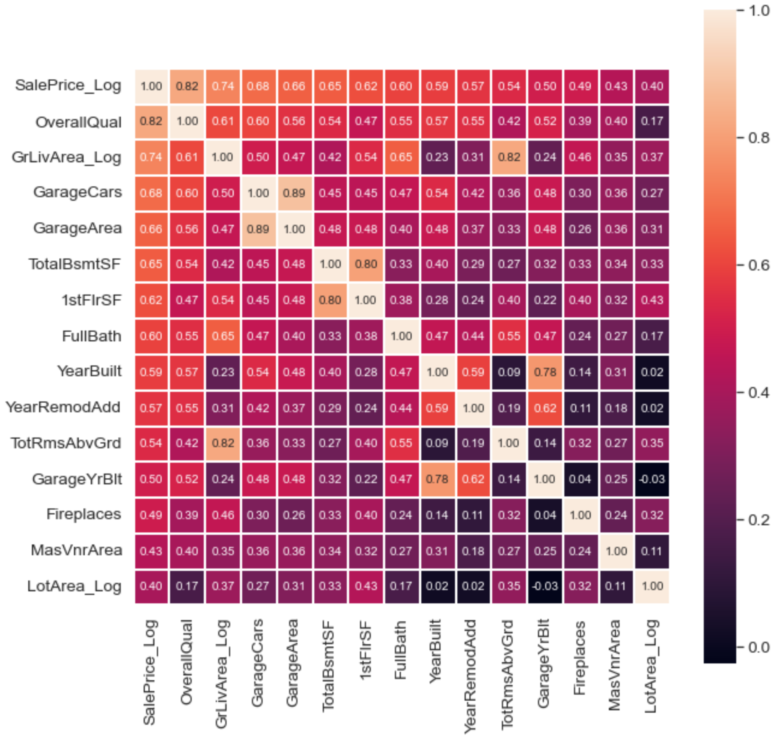
In [ ]:

Correlation matrix 1

Features with largest correlation to SalePrice_Log
all numerical features with correlation coefficient above threshold

In [ ]:
```
nr_feats = len(cols_abv_corr_limit)
```

In [ ]:
```
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)

In [ ]:

Part 2: Data wrangling

Drop all columns with only small correlation to SalePrice
Transform Categorical to numerical
Handling columns with missing data
Log values
Drop all columns with strong correlation to similar features

Numerical columns : drop similar and low correlation

Categorical columns : Transform to numerical

In [ ]:

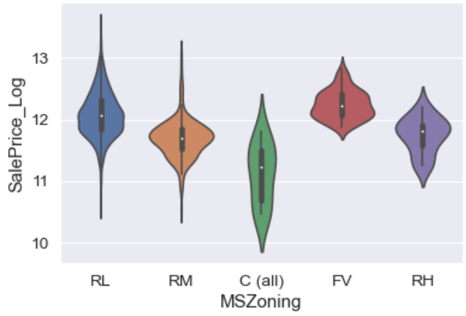
Dropping all columns with weak correlation to SalePrice

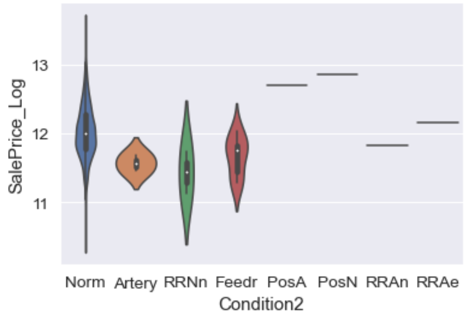
In [ ]:
```python
id_test = df_test['Id']

to_drop_num  = cols_bel_corr_limit
to_drop_catg = catg_weak_corr

cols_to_drop = ['Id'] + to_drop_num + to_drop_catg

for df in [df_train, df_test]:
    df.drop(cols_to_drop, inplace= True, axis = 1)
```

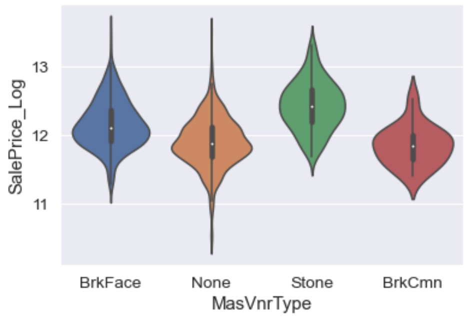
Convert categorical columns to numerical

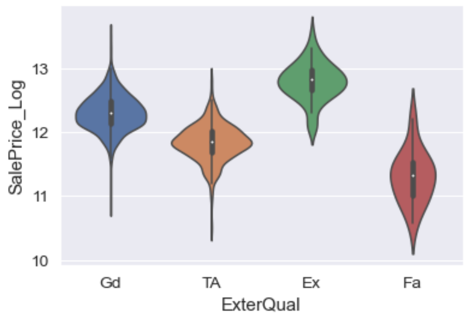
For those categorcial features where the EDA with boxplots seem to show a strong dependence of the SalePrice on the category, we transform the columns to numerical. To investigate the relation of the categories to SalePrice in more detail, we make violinplots for these features Also, we look at the mean of SalePrice as function of category.

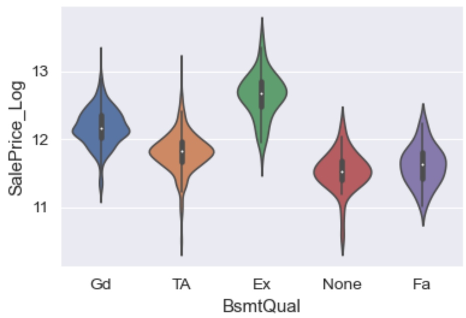
In [ ]:
```python
catg_list = catg_strong_corr.copy()
catg_list.remove('Neighborhood')

for catg in catg_list :
    #sns.catplot(x=catg, y=target, data=df_train, kind='boxen')
    sns.violinplot(x=catg, y=target, data=df_train)
    plt.show()
    #sns.boxenplot(x=catg, y=target, data=df_train)
    #bp = df_train.boxplot(column=[target], by=catg)
```

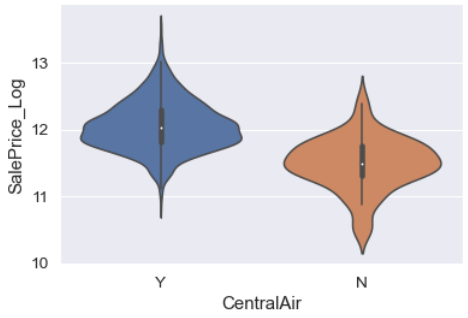


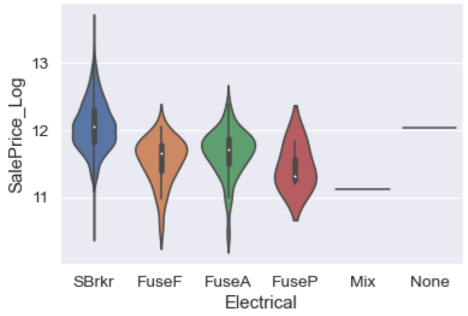


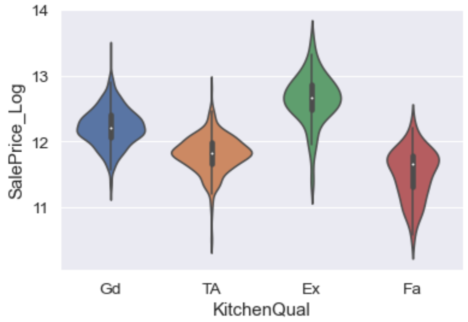


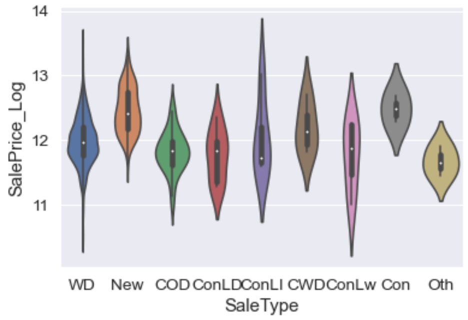






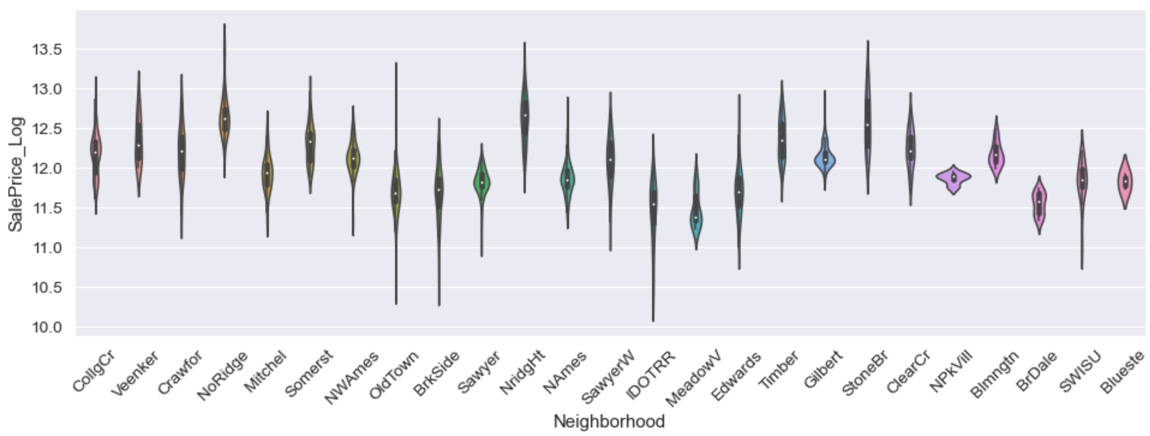






In [ ]:

```python
fig, ax = plt.subplots()
fig.set_size_inches(16, 5)
sns.violinplot(x='Neighborhood', y=target, data=df_train, ax=ax)
plt.xticks(rotation=45)
plt.show()
```



In [ ]:

```python
for catg in catg_list :
    g = df_train.groupby(catg)[target].mean()
    print(g)
```

```
MSZoning
C (all)    11.118259
FV         12.246616
RH         11.749840
RL         12.085939
RM         11.692893
Name: SalePrice_Log, dtype: float64
Condition2
Artery    11.570036
Feedr     11.670631
Norm      12.025925
```

```
PosA      12.691580
PosN      12.860999
RRAe      12.154779
RRAn      11.827043
RRNn      11.435329
Name: SalePrice_Log, dtype: float64
MasVnrType
BrkCmn     11.853239
BrkFace    12.163630
None       11.896884
Stone      12.431016
Name: SalePrice_Log, dtype: float64
ExterQual
Ex    12.792412
Fa    11.304541
Gd    12.311282
TA    11.837985
Name: SalePrice_Log, dtype: float64
BsmtQual
Ex      12.650235
Fa      11.617600
Gd      12.179882
None    11.529680
TA      11.810855
Name: SalePrice_Log, dtype: float64
CentralAir
N    11.491858
Y    12.061099
Name: SalePrice_Log, dtype: float64
Electrical
FuseA    11.660315
FuseF    11.539624
FuseP    11.446808
Mix      11.112448
None     12.028739
SBrkr    12.061474
Name: SalePrice_Log, dtype: float64
KitchenQual
Ex    12.645425
Fa    11.504581
Gd    12.222337
TA    11.810592
Name: SalePrice_Log, dtype: float64
SaleType
COD      11.827437
CWD      12.198344
Con      12.483911
ConLD    11.773000
ConLI    12.044878
ConLw    11.769706
New      12.466114
Oth      11.675295
WD       11.991061
Name: SalePrice_Log, dtype: float64
```

In [ ]:

```
# 'MSZoning'
msz_catg2 = ['RM', 'RH']
msz_catg3 = ['RL', 'FV']


# Neighborhood
nbhd_catg2 = ['Blmngtn', 'ClearCr', 'CollgCr', 'Crawfor', 'Gilbert', 'NWAmes', 'Somerst', 'Timber', 'Veenker']
nbhd_catg3 = ['NoRidge', 'NridgHt', 'StoneBr']

# Condition2
cond2_catg2 = ['Norm', 'RRAe']
cond2_catg3 = ['PosA', 'PosN']

# SaleType
SlTy_catg1 = ['Oth']
SlTy_catg3 = ['CWD']
SlTy_catg4 = ['New', 'Con']


#[]
```

In [ ]:

```
for df in [df_train, df_test]:

    df['MSZ_num'] = 1
    df.loc[(df['MSZoning'].isin(msz_catg2) ), 'MSZ_num'] = 2
    df.loc[(df['MSZoning'].isin(msz_catg3) ), 'MSZ_num'] = 3

    df['NbHd_num'] = 1
    df.loc[(df['Neighborhood'].isin(nbhd_catg2) ), 'NbHd_num'] = 2
    df.loc[(df['Neighborhood'].isin(nbhd_catg3) ), 'NbHd_num'] = 3

    df['Cond2_num'] = 1
    df.loc[(df['Condition2'].isin(cond2_catg2) ), 'Cond2_num'] = 2
    df.loc[(df['Condition2'].isin(cond2_catg3) ), 'Cond2_num'] = 3

    df['Mas_num'] = 1
    df.loc[(df['MasVnrType'] == 'Stone' ), 'Mas_num'] = 2

    df['ExtQ_num'] = 1
    df.loc[(df['ExterQual'] == 'TA' ), 'ExtQ_num'] = 2
    df.loc[(df['ExterQual'] == 'Gd' ), 'ExtQ_num'] = 3
    df.loc[(df['ExterQual'] == 'Ex' ), 'ExtQ_num'] = 4

    df['BsQ_num'] = 1
    df.loc[(df['BsmtQual'] == 'Gd' ), 'BsQ_num'] = 2
    df.loc[(df['BsmtQual'] == 'Ex' ), 'BsQ_num'] = 3

    df['CA_num'] = 0
    df.loc[(df['CentralAir'] == 'Y' ), 'CA_num'] = 1

    df['Elc_num'] = 1
    df.loc[(df['Electrical'] == 'SBrkr' ), 'Elc_num'] = 2


    df['KiQ_num'] = 1
    df.loc[(df['KitchenQual'] == 'TA' ), 'KiQ_num'] = 2
    df.loc[(df['KitchenQual'] == 'Gd' ), 'KiQ_num'] = 3
    df.loc[(df['KitchenQual'] == 'Ex' ), 'KiQ_num'] = 4
```

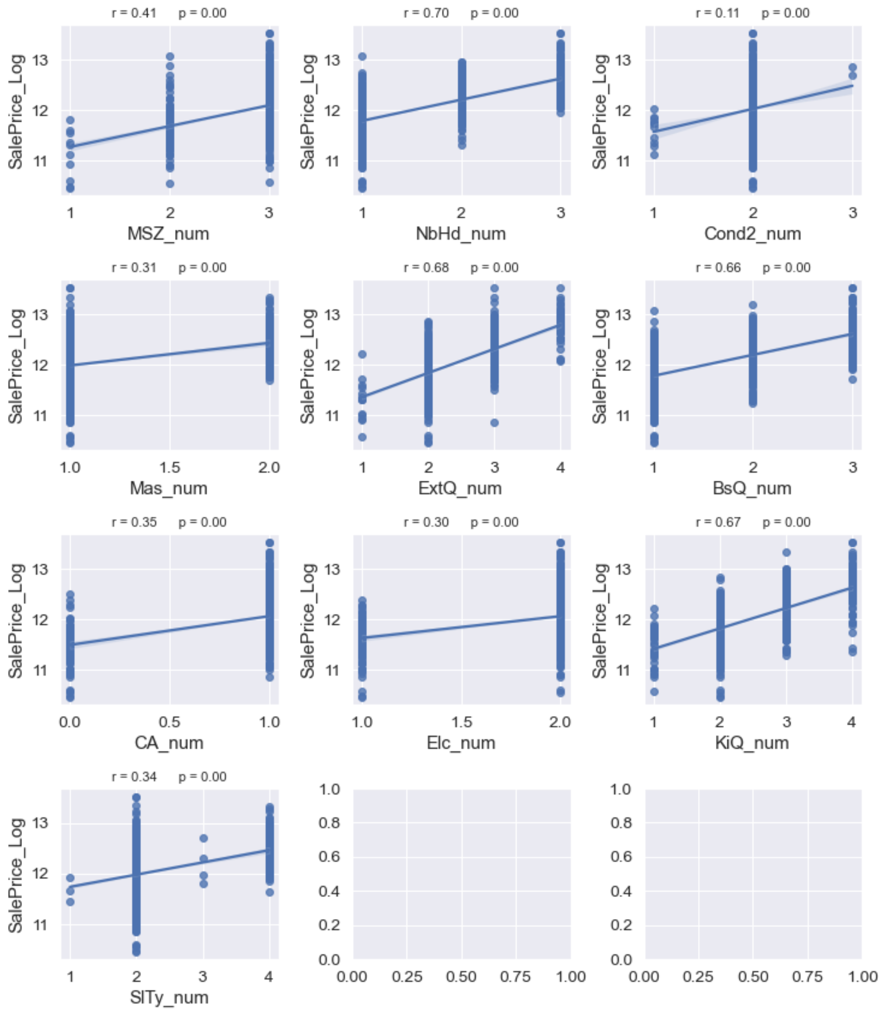
```
    df['SlTy_num'] = 2
    df.loc[(df['SaleType'].isin(SlTy_catg1) ), 'SlTy_num'] = 1
    df.loc[(df['SaleType'].isin(SlTy_catg3) ), 'SlTy_num'] = 3
    df.loc[(df['SaleType'].isin(SlTy_catg4) ), 'SlTy_num'] = 4
```

In [ ]:

Checking correlation to SalePrice for the new numerical columns

In [ ]:

```
new_col_num = ['MSZ_num', 'NbHd_num', 'Cond2_num', 'Mas_num', 'ExtQ_num', 'BsQ_num', 'CA_num', 'Elc_num', 'KiQ_num', 'SlTy_num']

nr_rows = 4
nr_cols = 3

fig, axs = plt.subplots(nr_rows, nr_cols, figsize=(nr_cols*3.5,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(new_col_num):
            sns.regplot(df_train[new_col_num[i]], df_train[target], ax = axs[r][c])
            stp = stats.pearsonr(df_train[new_col_num[i]], df_train[target])
            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()
```



There are few columns with quite large correlation to SalePrice (NbHd_num, ExtQ_num, BsQ_num, KiQ_num).
These will probably be useful for optimal performance of the Regressors in part 3.

Dropping the converted categorical columns and the new numerical columns with weak correlation

columns and correlation before dropping

In [ ]:

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

corr1 = df_train.corr()
corr_abs_1 = corr1.abs()

nr_all_cols = len(df_train)
ser_corr_1 = corr_abs_1.nlargest(nr_all_cols, target)[target]

print(ser_corr_1)
cols_bel_corr_limit_1 = list(ser_corr_1[ser_corr_1.values <= min_val_corr].index)


for df in [df_train, df_test] :
    df.drop(catg_cols_to_drop, inplace= True, axis = 1)
    df.drop(cols_bel_corr_limit_1, inplace= True, axis = 1)
```

```
SalePrice_Log     1.000000
OverallQual       0.821404
GrLivArea_Log     0.737427
NbHd_num          0.696962
```

```
ExtQ_num        0.682225
GarageCars      0.681033
KiQ_num         0.669989
BsQ_num         0.661286
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
MSZ_num         0.409423
LotArea_Log     0.402814
CA_num          0.351598
SlTy_num        0.337469
Mas_num         0.313280
Elc_num         0.304857
Cond2_num       0.107610
Name: SalePrice_Log, dtype: float64
```

columns and correlation after dropping

In [ ]:

```
corr2 = df_train.corr()
corr_abs_2 = corr2.abs()

nr_all_cols = len(df_train)
ser_corr_2 = corr_abs_2.nlargest(nr_all_cols, target)[target]

print(ser_corr_2)
```

```
SalePrice_Log   1.000000
OverallQual     0.821404
GrLivArea_Log   0.737427
NbHd_num        0.696962
ExtQ_num        0.682225
GarageCars      0.681033
KiQ_num         0.669989
BsQ_num         0.661286
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
MSZ_num         0.409423
LotArea_Log     0.402814
Name: SalePrice_Log, dtype: float64
```

new dataframes

In [ ]:

```
df_train.head()
```

Out[ ]:

	MSZoning	OverallQual	YearBuilt	YearRemodAdd	MasVnrArea	TotalBsmtSF	1stFlrSF	FullBath	TotRmsAbvGrd	Fireplaces	GarageYrBlt	GarageCars	GarageArea	SalePrice_Log	GrLivArea_Log	Lo
0	RL	7	2003	2003	196.0	856	856	2	8	0	2003.0	2	548	12.247694	7.444249	
1	RL	6	1976	1976	0.0	1262	1262	2	6	1	1976.0	2	460	12.109011	7.140453	
2	RL	7	2001	2002	162.0	920	920	2	6	1	2001.0	2	608	12.317167	7.487734	
3	RL	7	1915	1970	0.0	756	961	1	7	1	1998.0	3	642	11.849398	7.448334	
4	RL	8	2000	2000	350.0	1145	1145	2	9	1	2000.0	3	836	12.429216	7.695303	

In [ ]:

```
df_test.head()
```

Out[ ]:

	MSZoning	OverallQual	YearBuilt	YearRemodAdd	MasVnrArea	TotalBsmtSF	1stFlrSF	FullBath	TotRmsAbvGrd	Fireplaces	GarageYrBlt	GarageCars	GarageArea	GrLivArea_Log	LotArea_Log	MS
0	RH	5	1961	1961	0.0	882.0	896	1	5	0	1961.0	1.0	730.0	6.797940	9.360655	
1	RL	6	1958	1958	108.0	1329.0	1329	1	6	0	1958.0	1.0	312.0	7.192182	9.565704	
2	RL	5	1997	1998	0.0	928.0	928	2	6	1	1997.0	2.0	482.0	7.395722	9.534595	
3	RL	6	1998	1998	20.0	926.0	926	2	7	1	1998.0	2.0	470.0	7.380256	9.208138	
4	RL	8	1992	1992	0.0	1280.0	1280	2	5	0	1992.0	2.0	506.0	7.154615	8.518193	

In [ ]:

List of all features with strong correlation to SalePrice_Log

after dropping all coumns with weak correlation

In [ ]:

```
corr = df_train.corr()
corr_abs = corr.abs()

nr_all_cols = len(df_train)
print (corr_abs.nlargest(nr_all_cols, target)[target])
```

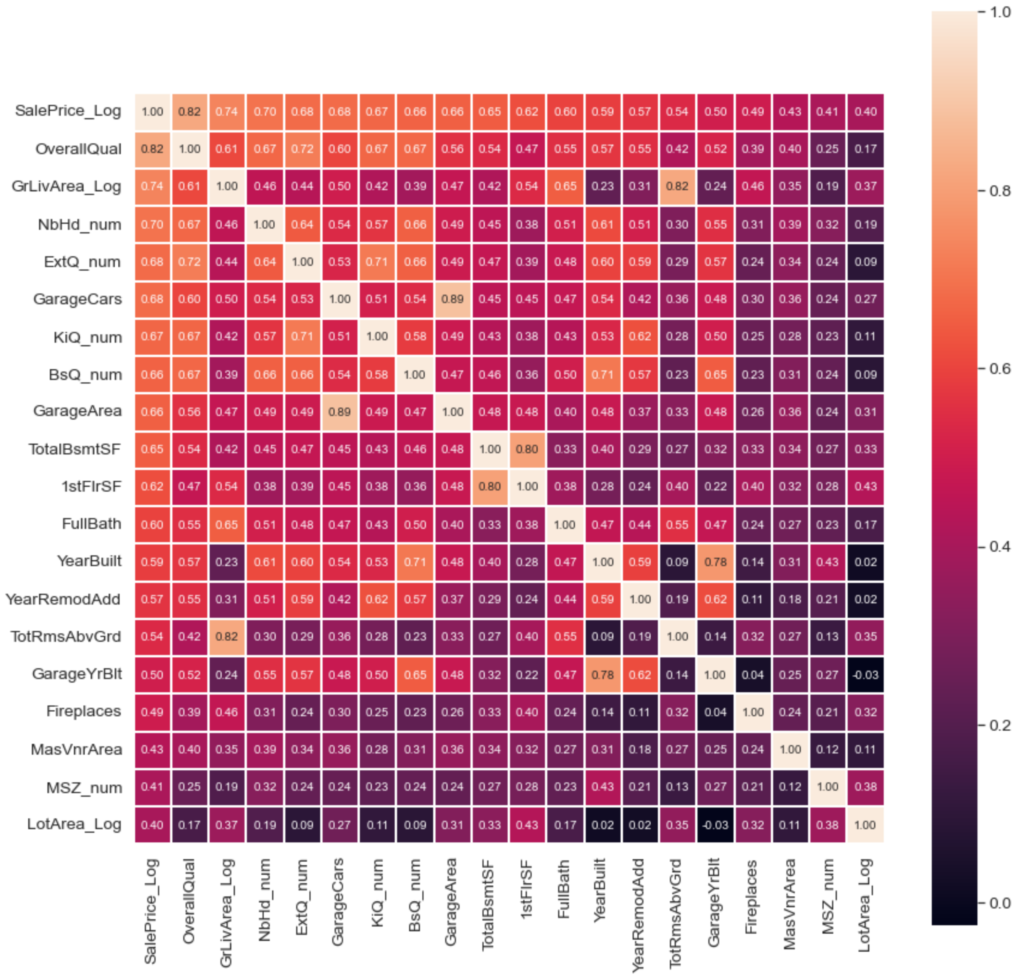
```
SalePrice_Log   1.000000
OverallQual     0.821404
GrLivArea_Log   0.737427
NbHd_num        0.696962
ExtQ_num        0.682225
GarageCars      0.681033
KiQ_num         0.669989
BsQ_num         0.661286
GarageArea      0.656128
TotalBsmtSF     0.647563
1stFlrSF        0.620500
FullBath        0.595899
```

```
YearBuilt        0.587043
YearRemodAdd     0.565992
TotRmsAbvGrd     0.537702
GarageYrBlt      0.500842
Fireplaces       0.491998
MasVnrArea       0.433353
MSZ_num          0.409423
LotArea_Log      0.402814
Name: SalePrice_Log, dtype: float64
```

Correlation Matrix 2 : All features with strong correlation to SalePrice

In [ ]:

```
nr_feats=len(df_train.columns)
plot_corr_matrix(df_train, nr_feats, target)
```



Check for Multicollinearity

Strong correlation of these features to other, similar features:

'GrLivArea_Log' and 'TotRmsAbvGrd'

'GarageCars' and 'GarageArea'

'TotalBsmtSF' and '1stFlrSF'

'YearBuilt' and 'GarageYrBlt'

Of those features we drop the one that has smaller correlation coeffiecient to Target.

In [ ]:

```
cols = corr_abs.nlargest(nr_all_cols, target)[target].index
cols = list(cols)

if drop_similar == 1 :
    for col in ['GarageArea','1stFlrSF','TotRmsAbvGrd','GarageYrBlt'] :
        if col in cols:
            cols.remove(col)
```

In [ ]:

```
cols = list(cols)
print(cols)
```

```
['SalePrice_Log', 'OverallQual', 'GrLivArea_Log', 'NbHd_num', 'ExtQ_num', 'GarageCars', 'KiQ_num', 'BsQ_num', 'TotalBsmtSF', 'FullBath', 'YearBuilt', 'YearRemodAdd', 'Fi
replaces', 'MasVnrArea', 'MSZ_num', 'LotArea_Log']
```

In [ ]:

List of features used for the Regressors in Part 3

In [ ]:

```
feats = cols.copy()
feats.remove('SalePrice_Log')

print(feats)
```

```
['OverallQual', 'GrLivArea_Log', 'NbHd_num', 'ExtQ_num', 'GarageCars', 'KiQ_num', 'BsQ_num', 'TotalBsmtSF', 'FullBath', 'YearBuilt', 'YearRemodAdd', 'Fireplaces', 'MasVn
rArea', 'MSZ_num', 'LotArea_Log']
```

In [ ]:

In [ ]:

```
df_train_ml = df_train[feats].copy()
df_test_ml  = df_test[feats].copy()

y = df_train[target]
```

Combine train and test data

for one hot encoding (use pandas get dummies) of all categorical features

uncommenting the following cell increases the number of features

up to now, all models in Part 3 are optimized for not applying one hot encoder

when applied, GridSearchCV needs to be rerun

In [ ]:

```
"""
all_data = pd.concat((df_train[feats], df_test[feats]))

li_get_dummies = ['OverallQual', 'NbHd_num', 'GarageCars','ExtQ_num', 'KiQ_num',
                  'BsQ_num', 'FullBath', 'Fireplaces', 'MSZ_num']
all_data = pd.get_dummies(all_data, columns=li_get_dummies, drop_first=True)

df_train_ml = all_data[:df_train.shape[0]]
df_test_ml  = all_data[df_train.shape[0]:]
"""
```

Out[ ]:

```
"\nall_data = pd.concat((df_train[feats], df_test[feats]))\n\nli_get_dummies = ['OverallQual', 'NbHd_num', 'GarageCars','ExtQ_num', 'KiQ_num',\n                  'BsQ_nu
m', 'FullBath', 'Fireplaces', 'MSZ_num']\nall_data = pd.get_dummies(all_data, columns=li_get_dummies, drop_first=True)\n\ndf_train_ml = all_data[:df_train.shape[0]]\ndf_
test_ml  = all_data[df_train.shape[0]:]\n"
```

In [ ]:

```
```

StandardScaler

In [ ]:

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
df_train_ml_sc = sc.fit_transform(df_train_ml)
df_test_ml_sc = sc.transform(df_test_ml)
```

In [ ]:

```
df_train_ml_sc = pd.DataFrame(df_train_ml_sc)
df_train_ml_sc.head()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0.658506	0.539624	0.658963	1.061109	0.313159	0.741127	0.648281	-0.473766	0.793546	1.052959	0.880362	-0.952231	0.521228	0.438861	-0.129585
1	-0.068293	-0.380198	0.658963	-0.689001	0.313159	-0.770150	0.648281	0.504925	0.793546	0.158428	-0.428115	0.605965	-0.574433	0.438861	0.118848
2	0.658506	0.671287	0.658963	1.061109	0.313159	0.741127	0.648281	-0.319490	0.793546	0.986698	0.831900	0.605965	0.331164	0.438861	0.427653
3	0.658506	0.551993	0.658963	-0.689001	1.652119	0.741127	-0.921808	-0.714823	-1.025620	-1.862551	-0.718888	0.605965	-0.574433	0.438861	0.108680
4	1.385305	1.299759	2.162512	1.061109	1.652119	0.741127	0.648281	0.222888	0.793546	0.953567	0.734975	0.605965	1.382104	0.438861	0.889271

Creating Datasets for ML algorithms

In [ ]:

```
X = df_train_ml.copy()
y = df_train[target]
X_test = df_test_ml.copy()

X_sc = df_train_ml_sc.copy()
y_sc = df_train[target]
X_test_sc = df_test_ml_sc.copy()

X.info()
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1458 entries, 0 to 1459
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   OverallQual    1458 non-null   int64
 1   GrLivArea_Log  1458 non-null   float64
 2   NbHd_num       1458 non-null   int64
 3   ExtQ_num       1458 non-null   int64
 4   GarageCars     1458 non-null   int64
 5   KiQ_num        1458 non-null   int64
 6   BsQ_num        1458 non-null   int64
 7   TotalBsmtSF    1458 non-null   int64
 8   FullBath       1458 non-null   int64
 9   YearBuilt      1458 non-null   int64
 10  YearRemodAdd   1458 non-null   int64
 11  Fireplaces     1458 non-null   int64
 12  MasVnrArea     1458 non-null   float64
 13  MSZ_num        1458 non-null   int64
 14  LotArea_Log    1458 non-null   float64
dtypes: float64(3), int64(12)
memory usage: 214.5 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   OverallQual    1459 non-null   int64
 1   GrLivArea_Log  1459 non-null   float64
 2   NbHd_num       1459 non-null   int64
 3   ExtQ_num       1459 non-null   int64
 4   GarageCars     1459 non-null   float64
 5   KiQ_num        1459 non-null   int64
 6   BsQ_num        1459 non-null   int64
 7   TotalBsmtSF    1459 non-null   float64
 8   FullBath       1459 non-null   int64
 9   YearBuilt      1459 non-null   int64
 10  YearRemodAdd   1459 non-null   int64
 11  Fireplaces     1459 non-null   int64
 12  MasVnrArea     1459 non-null   float64
 13  MSZ_num        1459 non-null   int64
 14  LotArea_Log    1459 non-null   float64
```

```
dtypes: float64(5), int64(10)
memory usage: 171.1 KB
```

In [ ]:

```
X.head()
```

Out[ ]:

	OverallQual	GrLivArea_Log	NbHd_num	ExtQ_num	GarageCars	KiQ_num	BsQ_num	TotalBsmtSF	FullBath	YearBuilt	YearRemodAdd	Fireplaces	MasVnrArea	MSZ_num	LotArea_Log
0	7	7.444249	2	3	2	3	2	856	2	2003	2003	0	196.0	3	9.041922
1	6	7.140453	2	2	2	2	2	1262	2	1976	1976	1	0.0	3	9.169518
2	7	7.487734	2	3	2	3	2	920	2	2001	2002	1	162.0	3	9.328123
3	7	7.448334	2	2	3	3	1	756	1	1915	1970	1	0.0	3	9.164296
4	8	7.695303	3	3	3	3	2	1145	2	2000	2000	1	350.0	3	9.565214

In [ ]:

```
X_sc.head()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0.658506	0.539624	0.658963	1.061109	0.313159	0.741127	0.648281	-0.473766	0.793546	1.052959	0.880362	-0.952231	0.521228	0.438861	-0.129585
1	-0.068293	-0.380198	0.658963	-0.689001	0.313159	-0.770150	0.648281	0.504925	0.793546	0.158428	-0.428115	0.605965	-0.574433	0.438861	0.118848
2	0.658506	0.671287	0.658963	1.061109	0.313159	0.741127	0.648281	-0.319490	0.793546	0.986698	0.831900	0.605965	0.331164	0.438861	0.427653
3	0.658506	0.551993	0.658963	-0.689001	1.652119	0.741127	-0.921808	-0.714823	-1.025620	-1.862551	-0.718888	0.605965	-0.574433	0.438861	0.108680
4	1.385305	1.299759	2.162512	1.061109	1.652119	0.741127	0.648281	0.222888	0.793546	0.953567	0.734975	0.605965	1.382104	0.438861	0.889271

In [ ]:

```
X_test.head()
```

Out[ ]:

	OverallQual	GrLivArea_Log	NbHd_num	ExtQ_num	GarageCars	KiQ_num	BsQ_num	TotalBsmtSF	FullBath	YearBuilt	YearRemodAdd	Fireplaces	MasVnrArea	MSZ_num	LotArea_Log
0	5	6.797940	1	2	1.0	2	1	882.0	1	1961	1961	0	0.0	2	9.360655
1	6	7.192182	1	2	1.0	3	1	1329.0	1	1958	1958	0	108.0	3	9.565704
2	5	7.395722	2	2	2.0	2	2	928.0	2	1997	1998	1	0.0	3	9.534595
3	6	7.380256	2	2	2.0	3	1	926.0	2	1998	1998	1	20.0	3	9.208138
4	8	7.154615	3	3	2.0	3	2	1280.0	2	1992	1992	0	0.0	3	8.518193

In [ ]:

Part 3: Scikit-learn basic regression models and comparison of results

Test simple sklearn models and compare by metrics

We test the following Regressors from scikit-learn:
LinearRegression
Ridge
Lasso
Elastic Net
Stochastic Gradient Descent
DecisionTreeRegressor
RandomForestRegressor
SVR

Model tuning and selection with GridSearchCV

In [ ]:

```
from sklearn.model_selection import GridSearchCV
score_calc = 'neg_mean_squared_error'
```

Linear Regression

In [ ]:

```
from sklearn.linear_model import LinearRegression

linreg = LinearRegression()
parameters = {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':[True, False]}
grid_linear = GridSearchCV(linreg, parameters, cv=nr_cv, verbose=1 , scoring = score_calc)
grid_linear.fit(X, y)

sc_linear = get_best_score(grid_linear)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
0.1362333768310373
{'copy_X': True, 'fit_intercept': True, 'normalize': True}
LinearRegression(normalize=True)
```

In [ ]:

```
linreg_sc = LinearRegression()
parameters = {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':[True, False]}
grid_linear_sc = GridSearchCV(linreg_sc, parameters, cv=nr_cv, verbose=1 , scoring = score_calc)
grid_linear_sc.fit(X_sc, y)

sc_linear_sc = get_best_score(grid_linear_sc)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
0.13623337683103734
{'copy_X': True, 'fit_intercept': True, 'normalize': True}
LinearRegression(normalize=True)
```

In [ ]:

```
linregr_all = LinearRegression()
#linregr_all.fit(X_train_all, y_train_all)
linregr_all.fit(X, y)
pred_linreg_all = linregr_all.predict(X_test)
pred_linreg_all[pred_linreg_all < 0] = pred_linreg_all.mean()
```

In [ ]:

```
sub_linreg = pd.DataFrame()
```

```
sub_linreg['Id'] = id_test
sub_linreg['SalePrice'] = pred_linreg_all
#sub_linreg.to_csv('linreg.csv',index=False)
```

Ridge

In [ ]:

```
from sklearn.linear_model import Ridge

ridge = Ridge()
parameters = {'alpha':[0.001,0.005,0.01,0.1,0.5,1], 'normalize':[True,False], 'tol':[1e-06,5e-06,1e-05,5e-05]}
grid_ridge = GridSearchCV(ridge, parameters, cv=nr_cv, verbose=1, scoring = score_calc)
grid_ridge.fit(X, y)

sc_ridge = get_best_score(grid_ridge)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
0.13620747796635556
{'alpha': 0.01, 'normalize': True, 'tol': 1e-06}
Ridge(alpha=0.01, normalize=True, tol=1e-06)
```

In [ ]:

```
ridge_sc = Ridge()
parameters = {'alpha':[0.001,0.005,0.01,0.1,0.5,1], 'normalize':[True,False], 'tol':[1e-06,5e-06,1e-05,5e-05]}
grid_ridge_sc = GridSearchCV(ridge_sc, parameters, cv=nr_cv, verbose=1, scoring = score_calc)
grid_ridge_sc.fit(X_sc, y)

sc_ridge_sc = get_best_score(grid_ridge_sc)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
0.13620747796635566
{'alpha': 0.01, 'normalize': True, 'tol': 1e-06}
Ridge(alpha=0.01, normalize=True, tol=1e-06)
```

In [ ]:

```
pred_ridge_all = grid_ridge.predict(X_test)
```

Lasso

In [ ]:

```
from sklearn.linear_model import Lasso

lasso = Lasso()
parameters = {'alpha':[1e-03,0.01,0.1,0.5,0.8,1], 'normalize':[True,False], 'tol':[1e-06,1e-05,5e-05,1e-04,5e-04,1e-03]}
grid_lasso = GridSearchCV(lasso, parameters, cv=nr_cv, verbose=1, scoring = score_calc)
grid_lasso.fit(X, y)

sc_lasso = get_best_score(grid_lasso)

pred_lasso = grid_lasso.predict(X_test)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
0.136455994502579 64
{'alpha': 0.001, 'normalize': False, 'tol': 0.0001}
Lasso(alpha=0.001, normalize=False)
```

Elastic Net

In [ ]:

```
from sklearn.linear_model import ElasticNet

enet = ElasticNet()
parameters = {'alpha' :[0.1,1.0,10], 'max_iter' :[1000000], 'l1_ratio':[0.04,0.05],
              'fit_intercept' : [False,True], 'normalize':[True,False], 'tol':[1e-02,1e-03,1e-04]}
grid_enet = GridSearchCV(enet, parameters, cv=nr_cv, verbose=1, scoring = score_calc)
grid_enet.fit(X_sc, y_sc)

sc_enet = get_best_score(grid_enet)

pred_enet = grid_enet.predict(X_test_sc)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
0.1371848369685215
{'alpha': 0.1, 'fit_intercept': True, 'l1_ratio': 0.04, 'max_iter': 1000000, 'normalize': False, 'tol': 0.01}
ElasticNet(alpha=0.1, l1_ratio=0.04, max_iter=1000000, normalize=False,
           tol=0.01)
```

SGDRegressor

Linear model fitted by minimizing a regularized empirical loss with SGD. SGD stands for Stochastic Gradient Descent: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). The regularizer is a penalty added to the loss function that shrinks model parameters towards the zero vector using either the squared euclidean norm L2 or the absolute norm L1 or a combination of both (Elastic Net).

In [ ]:

```
from sklearn.linear_model import SGDRegressor

sgd = SGDRegressor()
parameters = {'max_iter' :[10000], 'alpha':[1e-05], 'epsilon':[1e-02], 'fit_intercept' : [True]  }
grid_sgd = GridSearchCV(sgd, parameters, cv=nr_cv, verbose=1, scoring = score_calc)
grid_sgd.fit(X_sc, y_sc)

sc_sgd = get_best_score(grid_sgd)

pred_sgd = grid_sgd.predict(X_test_sc)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
0.1376250908227302
{'alpha': 1e-05, 'epsilon': 0.01, 'fit_intercept': True, 'max_iter': 10000}
SGDRegressor(alpha=1e-05, epsilon=0.01, max_iter=10000)
```

DecisionTreeRegressor

In [ ]:

```
from sklearn.tree import DecisionTreeRegressor

param_grid = { 'max_depth' : [7,8,9,10] , 'max_features' : [11,12,13,14] ,
               'max_leaf_nodes' : [None, 12,15,18,20] ,'min_samples_split' : [20,25,30],
                'presort': [False,True] , 'random_state': [5] }

grid_dtree = GridSearchCV(DecisionTreeRegressor(), param_grid, cv=nr_cv, refit=True, verbose=1, scoring = score_calc)
grid_dtree.fit(X, y)
```

```
sc_dtree = get_best_score(grid_dtree)

pred_dtree = grid_dtree.predict(X_test)
```

```
Fitting 5 folds for each of 480 candidates, totalling 2400 fits
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
C:\Users\AZMINE~1\AppData\Local\Temp/ipykernel_9036/3976936800.py in <module>
      6 
      7 grid_dtree = GridSearchCV(DecisionTreeRegressor(), param_grid, cv=nr_cv, refit=True, verbose=1, scoring = score_calc)
----> 8 grid_dtree.fit(X, y)
      9 
     10 sc_dtree = get_best_score(grid_dtree)

d:\SOFT\anconda\lib\site-packages\sklearn\model_selection\_search.py in fit(self, X, y, groups, **fit_params)
    873                 return results
    874 
--> 875             self._run_search(evaluate_candidates)
    876 
    877             # multimetric is determined here because in the case of a callable

d:\SOFT\anconda\lib\site-packages\sklearn\model_selection\_search.py in _run_search(self, evaluate_candidates)
   1373     def _run_search(self, evaluate_candidates):
   1374         """Search all candidates in param_grid"""
-> 1375         evaluate_candidates(ParameterGrid(self.param_grid))
   1376 
   1377 

d:\SOFT\anconda\lib\site-packages\sklearn\model_selection\_search.py in evaluate_candidates(candidate_params, cv, more_results)
    820                     )
    821 
--> 822                 out = parallel(
    823                     delayed(_fit_and_score)(
    824                         clone(base_estimator),

d:\SOFT\anconda\lib\site-packages\joblib\parallel.py in __call__(self, iterable)
   1041             # remaining jobs.
   1042             self._iterating = False
-> 1043             if self.dispatch_one_batch(iterator):
   1044                 self._iterating = self._original_iterator is not None
   1045 

d:\SOFT\anconda\lib\site-packages\joblib\parallel.py in dispatch_one_batch(self, iterator)
    859                 return False
    860             else:
--> 861                 self._dispatch(tasks)
    862                 return True
    863 

d:\SOFT\anconda\lib\site-packages\joblib\parallel.py in _dispatch(self, batch)
    777         with self._lock:
    778             job_idx = len(self._jobs)
--> 779             job = self._backend.apply_async(batch, callback=cb)
    780             # A job can complete so quickly than its callback is
    781             # called before we get here, causing self._jobs to

d:\SOFT\anconda\lib\site-packages\joblib\_parallel_backends.py in apply_async(self, func, callback)
    206     def apply_async(self, func, callback=None):
    207         """Schedule a func to be run"""
--> 208         result = ImmediateResult(func)
    209         if callback:
    210             callback(result)

d:\SOFT\anconda\lib\site-packages\joblib\_parallel_backends.py in __init__(self, batch)
    570         # Don't delay the application, to avoid keeping the input
    571         # arguments in memory
--> 572         self.results = batch()
    573 
    574     def get(self):

d:\SOFT\anconda\lib\site-packages\joblib\parallel.py in __call__(self)
    260         # change the default number of processes to -1
    261         with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 262             return [func(*args, **kwargs)
    263                     for func, args, kwargs in self.items]
    264 

d:\SOFT\anconda\lib\site-packages\joblib\parallel.py in <listcomp>(.0)
    260         # change the default number of processes to -1
    261         with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 262             return [func(*args, **kwargs)
    263                     for func, args, kwargs in self.items]
    264 

d:\SOFT\anconda\lib\site-packages\sklearn\utils\fixes.py in __call__(self, *args, **kwargs)
    115     def __call__(self, *args, **kwargs):
    116         with config_context(**self.config):
--> 117             return self.function(*args, **kwargs)
    118 
    119 

d:\SOFT\anconda\lib\site-packages\sklearn\model_selection\_validation.py in _fit_and_score(estimator, X, y, scorer, train, test, verbose, parameters, fit_params, return_
train_score, return_parameters, return_n_test_samples, return_times, return_estimator, split_progress, candidate_progress, error_score)
    672             cloned_parameters[k] = clone(v, safe=False)
    673 
--> 674         estimator = estimator.set_params(**cloned_parameters)
    675 
    676     start_time = time.time()

d:\SOFT\anconda\lib\site-packages\sklearn\base.py in set_params(self, **params)
    244             if key not in valid_params:
    245                 local_valid_params = self._get_param_names()
--> 246                 raise ValueError(
    247                     f"Invalid parameter {key!r} for estimator {self}. "
    248                     f"Valid parameters are: {local_valid_params!r}."

ValueError: Invalid parameter 'presort' for estimator DecisionTreeRegressor(max_depth=7, max_features=11, min_samples_split=20). Valid parameters are: ['ccp_alpha', 'cri
terion', 'max_depth', 'max_features', 'max_leaf_nodes', 'min_impurity_decrease', 'min_samples_leaf', 'min_samples_split', 'min_weight_fraction_leaf', 'random_state', 'sp
litter'].
```

In [ ]:

```
dtree_pred = grid_dtree.predict(X_test)
sub_dtree = pd.DataFrame()
sub_dtree['Id'] = id_test
```

```
sub_dtree['SalePrice'] = dtree_pred
#sub_dtree.to_csv('dtreeregr.csv',index=False)
```

In [ ]:

```

```

RandomForestRegressor

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor

param_grid = {'min_samples_split' : [3,4,6,10], 'n_estimators' : [70,100], 'random_state': [5] }
grid_rf = GridSearchCV(RandomForestRegressor(), param_grid, cv=nr_cv, refit=True, verbose=1, scoring = score_calc)
grid_rf.fit(X, y)

sc_rf = get_best_score(grid_rf)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:   16.6s finished
0.1465978663015509
{'min_samples_split': 4, 'n_estimators': 100, 'random_state': 5}
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=4,
           min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
           oob_score=False, random_state=5, verbose=0, warm_start=False)
```

In [ ]:

```
pred_rf = grid_rf.predict(X_test)

sub_rf = pd.DataFrame()
sub_rf['Id'] = id_test
sub_rf['SalePrice'] = pred_rf

if use_logvals == 1:
    sub_rf['SalePrice'] = np.exp(sub_rf['SalePrice'])

sub_rf.to_csv('rf.csv',index=False)
```

In [ ]:

```
sub_rf.head(10)
```

Out[ ]:

	Id	SalePrice
0	1461	121404.964212
1	1462	130824.396900
2	1463	183372.764889
3	1464	183944.210608
4	1465	198272.459357
5	1466	182039.290710
6	1467	164671.143500
7	1468	175829.325089
8	1469	180844.256443
9	1470	121240.046457

In [ ]:

```

```

KNN Regressor

In [ ]:

```
from sklearn.neighbors import KNeighborsRegressor

param_grid = {'n_neighbors' : [3,4,5,6,7,10,15] ,
              'weights' : ['uniform','distance'] ,
              'algorithm' : ['ball_tree', 'kd_tree', 'brute']}

grid_knn = GridSearchCV(KNeighborsRegressor(), param_grid, cv=nr_cv, refit=True, verbose=1, scoring = score_calc)
grid_knn.fit(X_sc, y_sc)

sc_knn = get_best_score(grid_knn)
```

```
Fitting 5 folds for each of 42 candidates, totalling 210 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
0.15615217437688825
{'algorithm': 'brute', 'n_neighbors': 5, 'weights': 'distance'}
KNeighborsRegressor(algorithm='brute', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=5, p=2,
          weights='distance')
[Parallel(n_jobs=1)]: Done 210 out of 210 | elapsed:    9.9s finished
```

In [ ]:

```
pred_knn = grid_knn.predict(X_test_sc)

sub_knn = pd.DataFrame()
sub_knn['Id'] = id_test
sub_knn['SalePrice'] = pred_knn

if use_logvals == 1:
    sub_knn['SalePrice'] = np.exp(sub_knn['SalePrice'])

sub_knn.to_csv('knn.csv',index=False)
```

In [ ]:

```
sub_knn.head(10)
```

Out[ ]:

	Id	SalePrice
0	1461	105027.859167
1	1462	123681.301052

	Id	SalePrice
2	1463	178767.921687
3	1464	194161.534320
4	1465	206225.287770
5	1466	177981.936038
6	1467	179348.288690
7	1468	175306.888377
8	1469	181974.660774
9	1470	119589.632069

In [ ]:

GaussianProcessRegressor

In [ ]:

```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import DotProduct, ConstantKernel

gpr = GaussianProcessRegressor(random_state=5, alpha=5e-9,
                                n_restarts_optimizer=0,
                               optimizer='fmin_l_bfgs_b',
                               copy_X_train=True)

param_grid = {'normalize_y' : [True,False],
              'kernel' : [DotProduct(), ConstantKernel(1.0, (1e-3, 1e3))] }

grid_gpr = GridSearchCV(gpr, param_grid, cv=nr_cv, verbose=1, scoring = score_calc)
grid_gpr.fit(X_sc, y_sc)

sc_gpr = get_best_score(grid_gpr)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
Fitting 5 folds for each of 4 candidates, totalling 20 fits
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:   53.4s finished
0.13623451472462014
{'kernel': DotProduct(sigma_0=1), 'normalize_y': False}
GaussianProcessRegressor(alpha=5e-09, copy_X_train=True,
             kernel=DotProduct(sigma_0=1), n_restarts_optimizer=0,
             normalize_y=False, optimizer='fmin_l_bfgs_b', random_state=5)
```

In [ ]:

```
pred_gpr = grid_gpr.predict(X_test_sc)

sub_gpr = pd.DataFrame()
sub_gpr['Id'] = id_test
sub_gpr['SalePrice'] = pred_gpr

if use_logvals == 1:
    sub_gpr['SalePrice'] = np.exp(sub_gpr['SalePrice'])

sub_gpr.to_csv('gpr.csv',index=False)
```

In [ ]:

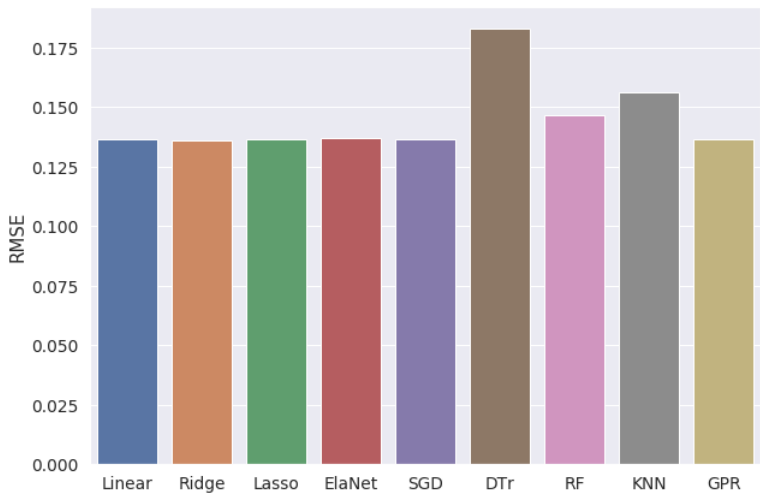
Comparison plot: RMSE of all models

In [ ]:

```
list_scores = [sc_linear, sc_ridge, sc_lasso, sc_enet,
               sc_sgd, sc_dtree, sc_rf, sc_knn, sc_gpr]
list_regressors = ['Linear','Ridge','Lasso','ElaNet','SGD','DTr','RF','KNN','GPR']
```

In [ ]:

```
fig, ax = plt.subplots()
fig.set_size_inches(10,7)
sns.barplot(x=list_regressors, y=list_scores, ax=ax)
plt.ylabel('RMSE')
plt.show()
```



The performance of all applied Regressors is very similar, except for Decision Tree which has larger RMSE than the other models.

In [ ]:

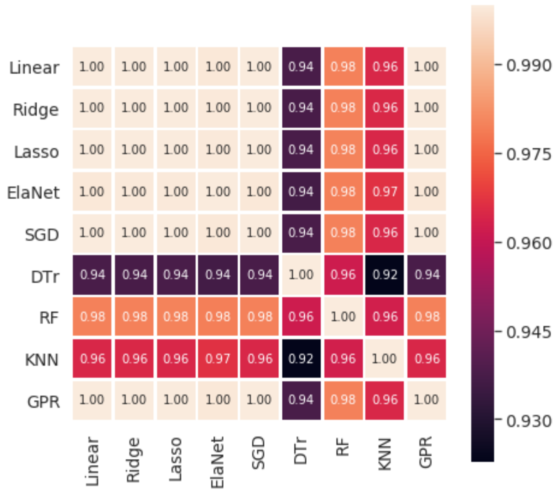
Correlation of model results

In [ ]:
```python
predictions = {'Linear': pred_linreg_all, 'Ridge': pred_ridge_all, 'Lasso': pred_lasso,
               'ElaNet': pred_enet, 'SGD': pred_sgd, 'DTr': pred_dtree, 'RF': pred_rf,
               'KNN': pred_knn, 'GPR': pred_gpr}
df_predictions = pd.DataFrame(data=predictions)
df_predictions.corr()
```

Out[ ]:

	Linear	Ridge	Lasso	ElaNet	SGD	DTr	RF	KNN	GPR
Linear	1.000000	0.999988	0.999809	0.999343	1.000000	0.937594	0.979555	0.964423	1.000000
Ridge	0.999988	1.000000	0.999835	0.999495	0.999990	0.937394	0.979495	0.964847	0.999988
Lasso	0.999809	0.999835	1.000000	0.999543	0.999813	0.937715	0.979863	0.964510	0.999809
ElaNet	0.999343	0.999495	0.999543	1.000000	0.999357	0.936547	0.979236	0.966557	0.999343
SGD	1.000000	0.999990	0.999813	0.999357	1.000000	0.937577	0.979552	0.964449	1.000000
DTr	0.937594	0.937394	0.937715	0.936547	0.937577	1.000000	0.961966	0.922761	0.937594
RF	0.979555	0.979495	0.979863	0.979236	0.979552	0.961966	1.000000	0.962788	0.979555
KNN	0.964423	0.964847	0.964510	0.966557	0.964449	0.922761	0.962788	1.000000	0.964423
GPR	1.000000	0.999988	0.999809	0.999343	1.000000	0.937594	0.979555	0.964423	1.000000

In [ ]:
```python
plt.figure(figsize=(7, 7))
sns.set(font_scale=1.25)
sns.heatmap(df_predictions.corr(), linewidths=1.5, annot=True, square=True,
                fmt='.2f', annot_kws={'size': 10},
                yticklabels=df_predictions.columns , xticklabels=df_predictions.columns
            )
plt.show()
```



For the first five models, the predictions show a very high correlation to each other (very close to 1.00).
Only for Random Forest and Decision Tree, the results are less correlated with the other Regressors.

mean of best models

In [ ]:
```python
sub_mean = pd.DataFrame()
sub_mean['Id'] = id_test
sub_mean['SalePrice'] = np.round( (pred_lasso + pred_enet + pred_rf + pred_sgd) / 4.0 )
sub_mean['SalePrice'] = sub_mean['SalePrice'].astype(float)
sub_mean.to_csv('mean.csv',index=False)
```

Conclusions

TODO

In [ ]:
```python

```

For some more advanced approaches on this task inluding Feature Engineering, Pipelines and methods like Stacking, Boosting and Voting have a look at [my second House Prices kernel]()