1. The data

I chose **Ames Housing dataset** which was compiled by Dean De Cock for use in data science education.

My model is a prediction focused one and I applied 5 types of regression in this project.

- LinearRegression
- Ridge
- Lasso
- ElasticNet
- RandomForestRegressor

First we load both the test dataset and the training dataset:

In [3]:
```python
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')

full_data = pd.concat([train_df, test_df]).reset_index(drop=True)

print('train_df\t{}'.format(train_df.shape))
print('test_df \t{}'.format(test_df.shape))
print('full_data \t{}'.format(full_data.shape))

train_df.head()
```

```
train_df	(1460, 81)
test_df 	(1459, 80)
full_data 	(2919, 81)
```

Out[3]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSol
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	[illegible]
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	[illegible]
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	[illegible]
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	[illegible]
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	[illegible]

5 rows × 81 columns

Next we check the different columns between train and test columns:

In [4]:
```python
set(train_df.columns) - set(test_df.columns)
```

Out[4]:
```
{'SalePrice'}
```

We will drop the "ID" column:

In [5]:
```python
train_df.drop(columns = 'Id', inplace=True)
test_df.drop(columns = 'Id', inplace=True)
full_data.drop(columns = 'Id', inplace=True)
print('Drop column')
```

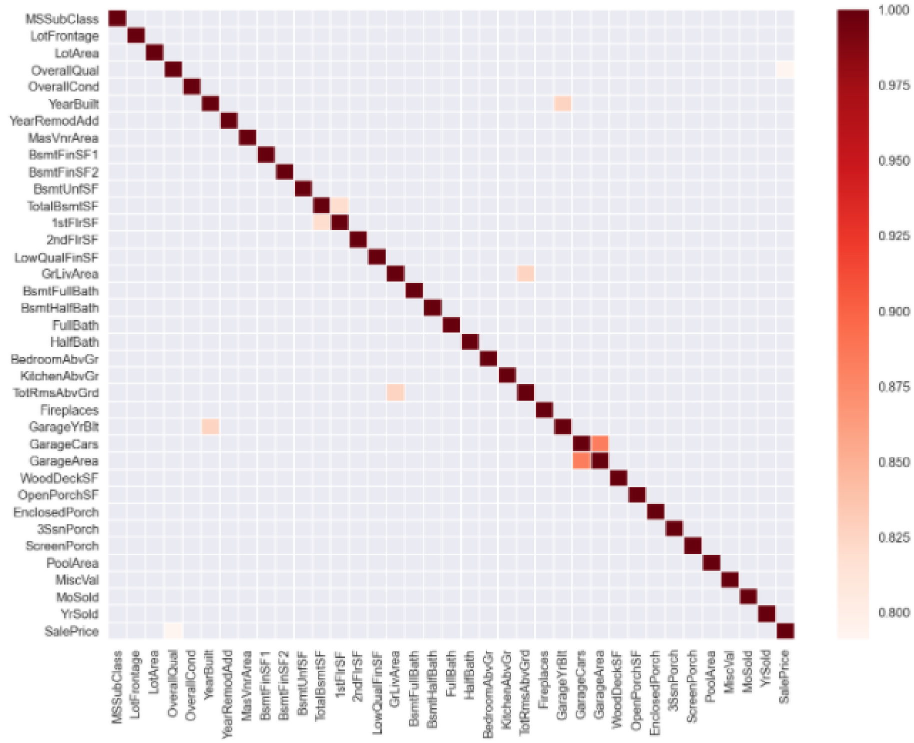
```
Drop column
```

A bit of exploring

Let's explore the data a bit in order to get a better understanding of what we are dealing with.

Next we will be plotting the overall data to see what variables are correlated:

In [6]:
```
plt.subplots(figsize=(12,9))

corrmat = train_df.corr()
sns.heatmap(corrmat, mask = corrmat < 0.75, linewidth = 0.5, cmap = 'Reds');
```

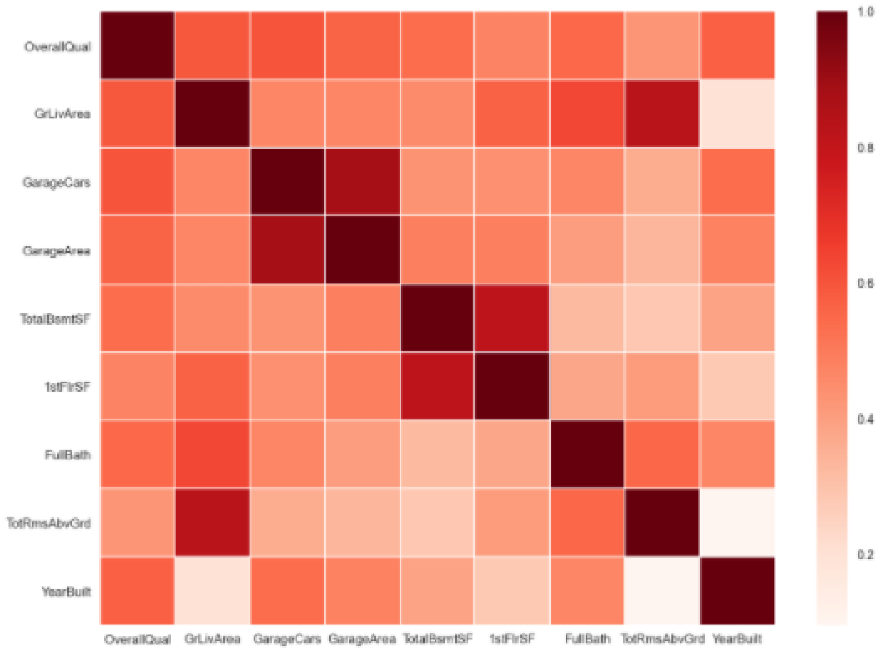


We will focus on the variables that are high correlation to target variable and then consider to eliminate outliers.

In [7]:
```
top_corr = abs(corrmat.SalePrice).sort_values(ascending=False).head(10)
top_corr_col = list(top_corr.index)
top_corr_col.remove('SalePrice')
top_corr
```

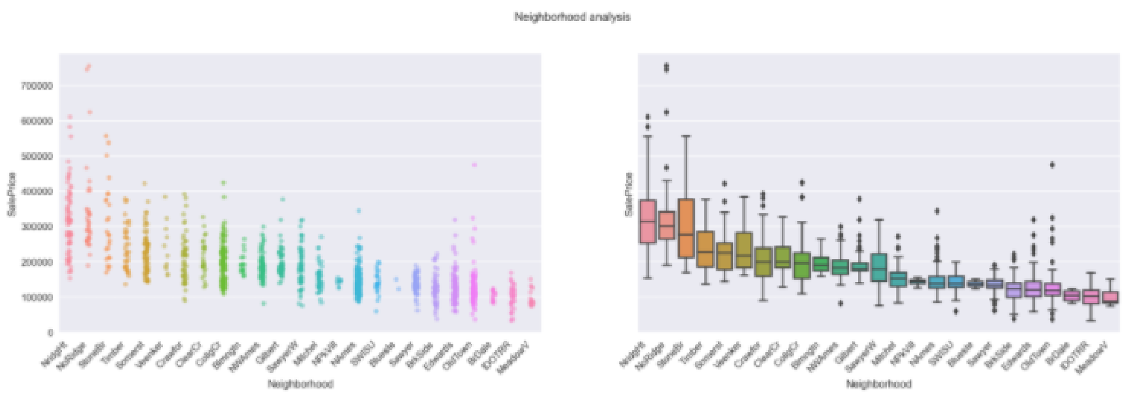
Out[7]:
```
SalePrice      1.000000
OverallQual    0.790982
GrLivArea      0.708624
GarageCars     0.640409
GarageArea     0.623431
TotalBsmtSF    0.613581
1stFlrSF       0.605852
FullBath       0.560664
TotRmsAbvGrd   0.533723
YearBuilt      0.522897
Name: SalePrice, dtype: float64
```

In [8]:
```
plt.subplots(figsize=(12,9))

corrmat = train_df[top_corr_col].corr()
sns.heatmap(corrmat, linewidth = 0.5, cmap = 'Reds');
```



In [10]:
```
sort_cate(train_df, 'Neighborhood')
```

```
Neighborhood  | type: object

           n      Ratio  TARGET_MEDIAN    Target_MEAN
NridgHt   77   5.273973         315000  316270.623377
NoRidge   41   2.808219         301500  335295.317073
StoneBr   25   1.712329         278000  310499.000000
Timber    38   2.602740         228475  242247.447368
Somerst   86   5.890411         225500  225379.837209
Veenker   11   0.753425         218000  238772.727273
Crawfor   51   3.493151         200624  210624.725490
ClearCr   28   1.917808         200250  212565.428571
CollgCr  150  10.273973         197200  197965.773333
Blmngtn   17   1.164384         191000  194870.882353
NWAmes    73   5.000000         182900  189050.068493
Gilbert   79   5.410959         181000  192854.506329
SawyerW   59   4.041096         179900  186555.796610
Mitchel   49   3.356164         153500  156270.122449
NPkVill    9   0.616438         146000  142694.444444
NAmes    225  15.410959         140000  145847.080000
SWISU     25   1.712329         139500  142591.360000
Blueste    2   0.136986         137500  137500.000000
Sawyer    74   5.068493         135000  136793.135135
BrkSide   58   3.972603         124300  124834.051724
Edwards  100   6.849315         121750  128219.700000
OldTown  113   7.739726         119000  128225.300885
BrDale    16   1.095890         106000  104493.750000
IDOTRR    37   2.534247         103000  100123.783784
MeadowV   17   1.164384          88000   98576.470588
```



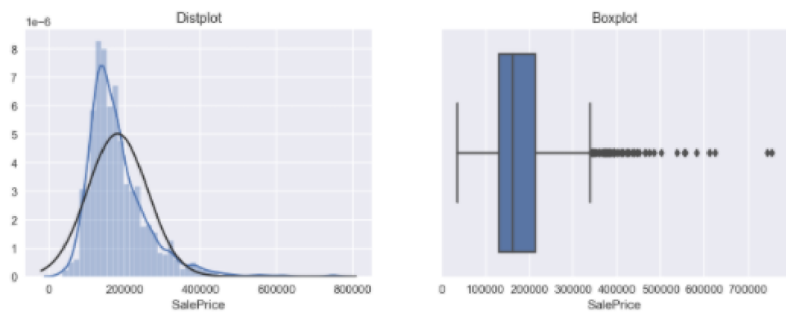
Dealing with Outliers

Some data points are far from other data points. Let’s remove outliers which are an issue since they can affect real results.

In [11]:
```
train_df = train_df.drop(train_df[(train_df['OverallQual'] < 5) & (train_df['SalePrice'] > 200000)].index)
train_df = train_df.drop(train_df[(train_df['OverallQual'] > 9) & (train_df['SalePrice'] < 200000)].index)


train_df = train_df.drop(train_df[(train_df['GrLivArea'] > 4000) & (train_df['SalePrice'] < 200000)].index)

train_df = train_df.drop(train_df[(train_df['GarageArea'] > 1200) & (train_df['SalePrice'] < 200000)].index)

train_df = train_df.drop(train_df[(train_df['TotalBsmtSF'] > 3000) & (train_df['SalePrice'] < 200000)].index)

train_df = train_df.drop(train_df[(train_df['1stFlrSF'] > 3000) & (train_df['SalePrice'] < 200000)].index)
```

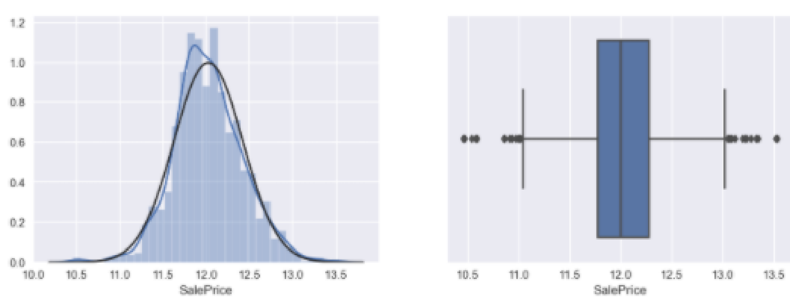
In [12]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12, 4))

ax1.set_title('Distplot')
sns.distplot(train_df['SalePrice'], fit=norm,  ax = ax1)

ax2.set_title('Boxplot')
sns.boxplot(train_df['SalePrice'], ax = ax2)


print ('Skewness: ', np.round(train_df['SalePrice'].skew(), 2))
print ('Kurtosis: ', np.round(train_df['SalePrice'].kurt(), 2))
```

```
Skewness:  1.88
Kurtosis:  6.53
```



In [13]:
```
train_df["SalePrice"] = np.log1p(train_df["SalePrice"])

fig, ax = plt.subplots(1, 2, figsize = (12, 4))
ax1.set_title('Distplot Log-transformation')
sns.distplot(train_df['SalePrice'],fit=norm,  ax = ax[0])

ax2.set_title('Boxplot Log-transformation')
sns.boxplot(train_df['SalePrice'], ax = ax[1])

print ('Skewness: ', np.round(train_df['SalePrice'].skew(), 2))
print ('Kurtosis: ', np.round(train_df['SalePrice'].kurt(), 2))
```

```
Skewness:  0.12
Kurtosis:  0.81
```



Missing values:

In [14]:
```
plot_missing(test_df)
```

Out[14]:

	Feature	% missing
PoolQC	1456.0	99.79
MiscFeature	1408.0	96.50
Alley	1352.0	92.67
Fence	1169.0	80.12
FireplaceQu	730.0	50.03
LotFrontage	227.0	15.56
GarageYrBlt	78.0	5.35
GarageCond	78.0	5.35
GarageQual	78.0	5.35
GarageFinish	78.0	5.35
GarageType	76.0	5.21
BsmtCond	45.0	3.08
BsmtExposure	44.0	3.02
BsmtQual	44.0	3.02
BsmtFinType1	42.0	2.88
BsmtFinType2	42.0	2.88
MasVnrType	16.0	1.10
MasVnrArea	15.0	1.03
MSZoning	4.0	0.27
BsmtFullBath	2.0	0.14
BsmtHalfBath	2.0	0.14
Utilities	2.0	0.14
Functional	2.0	0.14
Exterior2nd	1.0	0.07
Exterior1st	1.0	0.07
SaleType	1.0	0.07
BsmtFinSF1	1.0	0.07
BsmtFinSF2	1.0	0.07
BsmtUnfSF	1.0	0.07
KitchenQual	1.0	0.07
GarageCars	1.0	0.07
GarageArea	1.0	0.07
TotalBsmtSF	1.0	0.07

In [15]:
```
none_cols = ['Alley', 'PoolQC', 'MiscFeature', 'Fence', 'FireplaceQu', 'GarageType',
             'GarageFinish', 'GarageQual', 'GarageCond', 'BsmtQual', 'BsmtCond',
             'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'MasVnrType']

for col in none_cols:
    train_df[col].replace(np.nan, 'None', inplace=True)
    test_df[col].replace(np.nan, 'None', inplace=True)
```

In [16]:
```
bsm=['BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'BsmtFullBath','BsmtHalfBath','BsmtQual']
train_df[bsm].groupby('BsmtQual').sum()
```

Out[16]:

BsmtQual	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	BsmtFullBath	BsmtHalfBath
Ex	95591	2851	91428	189870	79	6
Fa	4575	276	20805	25656	6	2
Gd	290890	21417	385330	697637	293	27
None	0	0	0	0	0	0
TA	246479	43418	328032	617929	238	49

In [17]:
```
gar = ['GarageYrBlt', 'GarageArea', 'GarageCars','GarageQual']
train_df[gar].groupby('GarageQual').sum()
```

Out[17]:

GarageQual	GarageYrBlt	GarageArea	GarageCars
Ex	5967.0	2064	5
Fa	92817.0	14946	65
Gd	27733.0	7800	26
None	0.0	0	0
Po	5756.0	978	3
TA	2586149.0	659328	2467

In [18]:
```
# Replace features with 0
zero_cols = ['BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'BsmtFullBath','BsmtHalfBath', 'GarageYrBlt', 'GarageArea',

for col in zero_cols:
    train_df[col].replace(np.nan, 0, inplace=True)
    test_df[col].replace(np.nan, 0, inplace=True)
```

In [19]:
```
most_freq_cols = ['Electrical', 'Exterior1st', 'Exterior2nd', 'Functional', 'KitchenQual','SaleType', 'Utilities']

for col in most_freq_cols:
    train_df[col].replace(np.nan, train_df[col].mode()[0], inplace=True)
    test_df[col].replace(np.nan, test_df[col].mode()[0], inplace=True)
```

In [20]:
```
fig, ax = plt.subplots(1,1,figsize = (10, 8))
sns.countplot(x="MSSubClass", hue="MSZoning", data=train_df);
```

`MSZoning` should be considered to replace NaN by group of **`MSSubClass`**

`LotFrontage` relates to **`Neighborhood`** so we will replace NaN by group of `Neighborhood`

For **`MSSubClass`**, **`YrSold`**, **`MoSold`** should be changed to string.

In [21]:

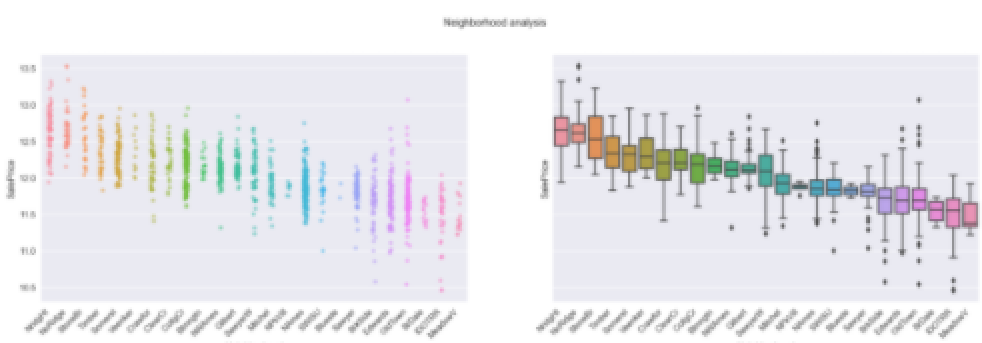
```
train_df['MSZoning'] = train_df.groupby('MSSubClass')['MSZoning'].apply(
    lambda x: x.fillna(x.mode()[0]))
test_df['MSZoning'] = test_df.groupby('MSSubClass')['MSZoning'].apply(
    lambda x: x.fillna(x.mode()[0]))


train_df["LotFrontage"] = train_df.groupby("Neighborhood")["LotFrontage"].transform(
    lambda x: x.fillna(x.median()))
test_df["LotFrontage"] = test_df.groupby("Neighborhood")["LotFrontage"].transform(
    lambda x: x.fillna(x.median()))


# type to string
chg_type = ['MSSubClass', 'YrSold', 'MoSold']
train_df[chg_type] = train_df[chg_type].astype(str)
test_df[chg_type] = test_df[chg_type].astype(str)
```

In [22]:

```
sort_cate(train_df, 'Neighborhood')
```

```
Neighborhood  | type: object
```

	n	Ratio	TARGET_MEDIAN	Target_MEAN
NridgHt	77	5.292006	12.660331	12.619415
NoRidge	41	2.817869	12.616529	12.676003
StoneBr	25	1.718213	12.535380	12.585490
Timber	38	2.611684	12.339184	12.363460
Somerst	86	5.910653	12.326077	12.296500
Veenker	11	0.756014	12.292255	12.344180
Crawfor	51	3.505155	12.209193	12.206664
ClearCr	27	1.855670	12.206078	12.232015
CollgCr	150	10.309278	12.191972	12.163647
Blmngtn	17	1.168385	12.160034	12.169421
NWAmes	73	5.017182	12.116700	12.130614
Gilbert	79	5.429553	12.106258	12.155809
SawyerW	59	4.054983	12.100162	12.090695
Mitchel	48	3.298969	11.932921	11.931917
NPkVill	9	0.618557	11.891369	11.866484
NAmes	225	15.463918	11.849405	11.868052
SWISU	25	1.718213	11.845827	11.838442
Blueste	2	0.137457	11.826543	11.826543
Sawyer	74	5.085911	11.813037	11.811475
BrkSide	58	3.986254	11.730225	11.679736
Edwards	98	6.735395	11.691071	11.705330
OldTown	113	7.766323	11.686887	11.703873
BrDale	16	1.099656	11.571204	11.547874
IDOTRR	36	2.474227	11.559202	11.450920
MeadowV	17	1.168385	11.385103	11.474533



Let's turn categorial features into ordinal features:

In [23]:
```
# Turn categorical into ordinal features
def cate_to_ordinal(df):

    n_map = {'MeadowV': 1,'IDOTRR': 1,'BrDale': 1,'OldTown': 2,'Edwards': 2,'BrkSide': 2,
             'Sawyer': 3,'Blueste': 3,'SWISU': 3,'NPkVill': 3,'NAmes': 3,'Mitchel': 4,
             'SawyerW': 5,'NWAmes': 5,'Gilbert': 5,'Blmngtn': 5,'CollgCr': 5,
             'ClearCr': 6,'Crawfor': 6,'Veenker': 7,'Somerst': 7,'Timber': 7,
             'StoneBr': 8,'NoRidge': 9,'NridgHt': 10}
    df['Neighborhood'] = df['Neighborhood'].map(n_map).fillna(0).astype('int')

    ext_map = {'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5}
    df['ExterQual'] = df['ExterQual'].map(ext_map).fillna(0).astype('int')

    ext_map = {'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5}
    df['ExterCond'] = df['ExterCond'].map(ext_map).fillna(0).astype('int')

    bsm_map = {'None': 0, 'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5}
    df['BsmtQual'] = df['BsmtQual'].map(bsm_map).fillna(0).astype('int')
    df['BsmtCond'] = df['BsmtCond'].map(bsm_map).astype('int')

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

    heat_map = {'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5}
    df['HeatingQC'] = df['HeatingQC'].map(heat_map).fillna(0).astype('int')
    df['KitchenQual'] = df['KitchenQual'].map(heat_map).fillna(0).astype('int')
    df['FireplaceQu'] = df['FireplaceQu'].map(bsm_map).fillna(0).astype('int')
    df['GarageCond'] = df['GarageCond'].map(bsm_map).fillna(0).astype('int')
    df['GarageQual'] = df['GarageQual'].map(bsm_map).fillna(0).astype('int')

cate_to_ordinal(train_df)
cate_to_ordinal(test_df)
```

Let's create new features from existing features in order to add more information to the target variable

In [25]:

```
train_df['TotalSF'] = train_df['BsmtFinSF1'] + train_df['BsmtFinSF2'] + train_df['1stFlrSF'] + train_df['2ndFlrSF']

test_df['TotalSF'] = test_df['BsmtFinSF1'] + test_df['BsmtFinSF2'] + test_df['1stFlrSF'] + test_df['2ndFlrSF']
```

If the values of a certain feature are skewed, depending on the model, skewness may violate model assumptions. In this case we will use Boxcox-Transformation to transform high skewness features.

In [25]:

```
num_col = list(train_df.select_dtypes(exclude='object').columns)
num_col.remove('SalePrice')

# Get skewness with num features
skew_feature = abs(train_df[num_col].apply(lambda x: skew(x))).sort_values(ascending = False)

# Filter skewness > 0.75
high_skew = skew_feature[skew_feature > 0.75]

# Boxcox

for f in high_skew.index:
    train_df[f] = boxcox1p(train_df[f], boxcox_normmax(train_df[f] + 1))
    test_df[f] = boxcox1p(test_df[f], boxcox_normmax(test_df[f] + 1))
```

In [26]:

```
train_df['Utilities'].value_counts(normalize = True).iloc[0]
```

Out[26]:

```
0.9993127147766323
```

In [28]:

```
drop_col += ['GarageYrBlt','TotRmsAbvGrd', '1stFlrSF', 'GarageCars']
# Drop column
train_df.drop(columns = drop_col, inplace=True)
test_df.drop(columns = drop_col, inplace=True)

print ('Drop Column')
```

```
Drop Column
```

In [29]:

```
y_train = train_df['SalePrice']
train_X = train_df.drop(columns = 'SalePrice')

print ('splitting completed')
```

```
splitting completed
```

In [30]:

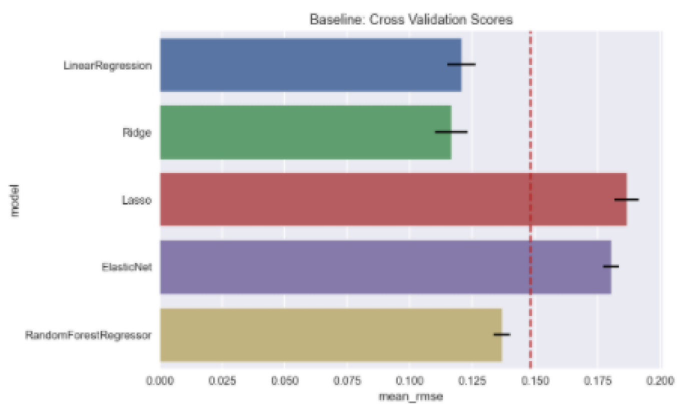
```
from sklearn.preprocessing import OneHotEncoder
# Get cat columns
cat_col = train_X.select_dtypes(include='object').columns
num_col = train_X.select_dtypes(exclude='object').columns

# instance of one-hot-encoder
enc = OneHotEncoder(handle_unknown='ignore',sparse=False)

# Apply one-hot encoder
OH_cols_train = pd.DataFrame(enc.fit_transform(train_X[cat_col]))
OH_cols_test = pd.DataFrame(enc.transform(test_df[cat_col]))


OH_cols_train.index = train_X.index
OH_cols_test.index = test_df.index


OH_cols_train.columns = enc.get_feature_names(cat_col)
OH_cols_test.columns = enc.get_feature_names(cat_col)

# Concat Cat with Num columns
X_train = pd.concat([train_X[num_col], OH_cols_train], axis=1)
X_test = pd.concat([test_df[num_col], OH_cols_test], axis=1)
```

Modeling

Here we will score our regression models

In [32]:
```
models = [('LinearRegression', LinearRegression()),
          ("Ridge", Ridge()),
          ("Lasso", Lasso()),
          ("ElasticNet", ElasticNet()),
          ('RandomForestRegressor', RandomForestRegressor())
         ]

print('='*30 +'RMSE BASE MODEL' + '='*30)

mean_rmse = []
sd_rmse = []
model_name = []
for name, model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    RMSE = -1*cross_val_score(model, X_train, y_train, cv=5, scoring='neg_root_mean_squared_error')
    print(f'{name:<25} {RMSE.mean():.5f} ±{RMSE.std():.3f}')

    mean_rmse.append(RMSE.mean())
    sd_rmse.append(RMSE.std())
    model_name.append(name)


base = pd.DataFrame({'model':model_name, 'mean_rmse':mean_rmse, 'sd_rmse':sd_rmse})
sns.barplot(x="mean_rmse", y="model", data=base, orient = 'h',**{'xerr': sd_rmse})
plt.title('Baseline: Cross Validation Scores')
plt.axvline(x = np.mean(mean_rmse), color = 'firebrick', linestyle = '--');
```

```
==============================RMSE BASE MODEL==============================
LinearRegression          0.12068 ±0.006
Ridge                     0.11665 ±0.006
Lasso                     0.18666 ±0.005
ElasticNet                0.18043 ±0.003
RandomForestRegressor     0.13684 ±0.003
```



In [33]:
```
# Ridge
pipe = make_pipeline(RobustScaler(), Ridge())

param_grid = {'ridge__alpha': [0.01, 0.1, 1, 10, 15 ,20, 25, 30]}

grid_rid = GridSearchCV(pipe, param_grid = param_grid, cv = 5, scoring = 'neg_root_mean_squared_error', verbose = False, n_jobs

best_grid_rid = grid_rid.fit(X_train, y_train)

rid_param = best_grid_rid.best_params_

print(f'best_params_: {rid_param}')
print(f'score: {-1*best_grid_rid.best_score_:.5f}')
```

```
best_params_: {'ridge__alpha': 25}
score: 0.11316
```

In [34]:
```
# Lasso
pipe = make_pipeline(Lasso())

param_grid = {'lasso__alpha': [5e-05, 0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008],
              'lasso__max_iter': [1e3, 1e5, 1e7]}

grid_las = GridSearchCV(pipe, param_grid = param_grid, cv = 5, scoring = 'neg_root_mean_squared_error', verbose = False, n_jobs

best_grid_las = grid_las.fit(X_train, y_train)

las_param = best_grid_las.best_params_

print(f'best_params_: {las_param}')
print(f'score: {-1*best_grid_las.best_score_:.5f}')
```

```
best_params_: {'lasso__alpha': 0.0004, 'lasso__max_iter': 1000.0}
score: 0.11141
```

In [35]:
```
# RandomForestRegressor
rf = RandomForestRegressor()

param_grid = {'bootstrap': [True, False],
              'max_depth': [10, 25, 50],
              'max_features': ['auto', 'sqrt'],
              'min_samples_leaf': [1, 2, 3],
              'min_samples_split': [2, 3],
              'n_estimators': [300,500]
             }

grid_rf = RandomizedSearchCV(rf, param_distributions = param_grid, cv = 5,scoring = 'neg_root_mean_squared_error', verbose = Fals

best_grid_rf = grid_rf.fit(X_train, y_train)

rf_param = best_grid_rf.best_params_

print(f'best_params_: {rf_param}')
print(f'score: {-1*best_grid_rf.best_score_:.5f}')
```

```
best_params_: {'n_estimators': 500, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 50, 'bo
otstrap': False}
score: 0.13061
```

In [36]:
```
# ElasticNet
pipe = make_pipeline(ElasticNet())

param_grid = {'elasticnet__alpha': [0.0001, 0.001, 0.01, 0.1, 1],
              'elasticnet__l1_ratio': [0.25, 0.5, 0.75]}

grid_en = GridSearchCV(pipe, param_grid = param_grid, cv = 5,scoring = 'neg_root_mean_squared_error', verbose = False, n_jobs =

best_grid_en = grid_en.fit(X_train, y_train)

en_param = best_grid_en.best_params_

print(f'best_params_: {en_param}')
print(f'score: {-1*best_grid_en.best_score_:.5f}')
```

```
best_params_: {'elasticnet__alpha': 0.001, 'elasticnet__l1_ratio': 0.5}
score: 0.11158
```

In [37]:
```
svr_scores = cross_val_score(best_grid_en.best_estimator_, X_train, y_train,
                             cv=5, n_jobs=-1, error_score="neg_root_mean_squared_error")

svr_scores.mean()
```

Out[37]:
```
0.9214997423540326
```

Let’s use stacking in an combined learning technique to combine multiple regression models through a meta-regressor. In this case we will use Lasso as our meta-regressor.

```
In [38]: stack = StackingCVRegressor(regressors = (best_grid_en.best_estimator_,
                                                   best_grid_rid.best_estimator_,
                                                   best_grid_las.best_estimator_,
                                                   best_grid_rf.best_estimator_),
                                     meta_regressor = best_grid_las.best_estimator_,
                                     use_features_in_secondary = True)


         stack.fit(X_train, y_train);

         stack_score = -cross_val_score(stack, X_train, y_train, scoring = 'neg_root_mean_squared_error', cv = 5)

         print(f'score: {stack_score.mean():.5f} ±{stack_score.std():.4f}')

         score: 0.11025 ±0.0057
```

```
In [39]: def blend_models_predict(X):

             return ((0.1 * best_grid_rid.predict(X)) +
                     (0.1 * best_grid_las.predict(X)) +
                     (0.1 * best_grid_en.predict(X)) +
                     (0.1 * best_grid_rf.predict(X)) +
                     (0.2 * stack.predict(X)))
```

```
In [41]: prediction = pd.read_csv("test.csv")

         prediction['SalePrice'] = np.floor(np.expm1(blend_models_predict(X_test)))

         prediction = prediction[['Id', 'SalePrice']]
         prediction.head()
```

Out[41]:

	Id	SalePrice
0	1461	3142.0
1	1462	3916.0
2	1463	4044.0
3	1464	3687.0
4	1465	2907.0

Conclusion

As you can see we successfully applied 5 regression techniques to our dataset. Even if the models aren’t the best we were able to obtain a prediction. Even if it’s inaccurate it shows how you can obtain valuable information from a large dataset and predict certain features of it.

I would choose the RandomForrest regression as a main form of regression for this dataset but there are other methods that are better-fitting and can produce truly accurate predictions.

Thank you!