In [1]:

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

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from scipy.stats import skew, norm
from scipy.special import boxcox1p
from scipy.stats import boxcox_normmax
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA
#models
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegre
ssor, AdaBoostRegressor, BaggingRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.linear_model import Ridge, RidgeCV
from sklearn.linear_model import ElasticNet, ElasticNetCV
from sklearn.svm import SVR
from mlxtend.regressor import StackingCVRegressor
import lightgbm as lgb
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will
 list all files under the input directory

import os
```

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

# You can write up to 5GB to the current directory (/kaggle/working/) that
 gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be sa
ved outside of the current session
```

```
/kaggle/input/house-prices-advanced-regression-techniques/train.csv
/kaggle/input/house-prices-advanced-regression-techniques/data_descrip
tion.txt
/kaggle/input/house-prices-advanced-regression-techniques/test.csv
/kaggle/input/house-prices-advanced-regression-techniques/sample_submi
ssion.csv
```

In [2]:

```
train_df=pd.read_csv('/kaggle/input/house-prices-advanced-regression-tech
niques/train.csv')
test_df=pd.read_csv('/kaggle/input/house-prices-advanced-regression-techn
iques/test.csv')
```

In [3]:

```
train_df.describe()
```

Out[3]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallC
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.00
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.57534
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.11279
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.00000
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.00000
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.00000
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.00000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.00000

8 rows × 38 columns

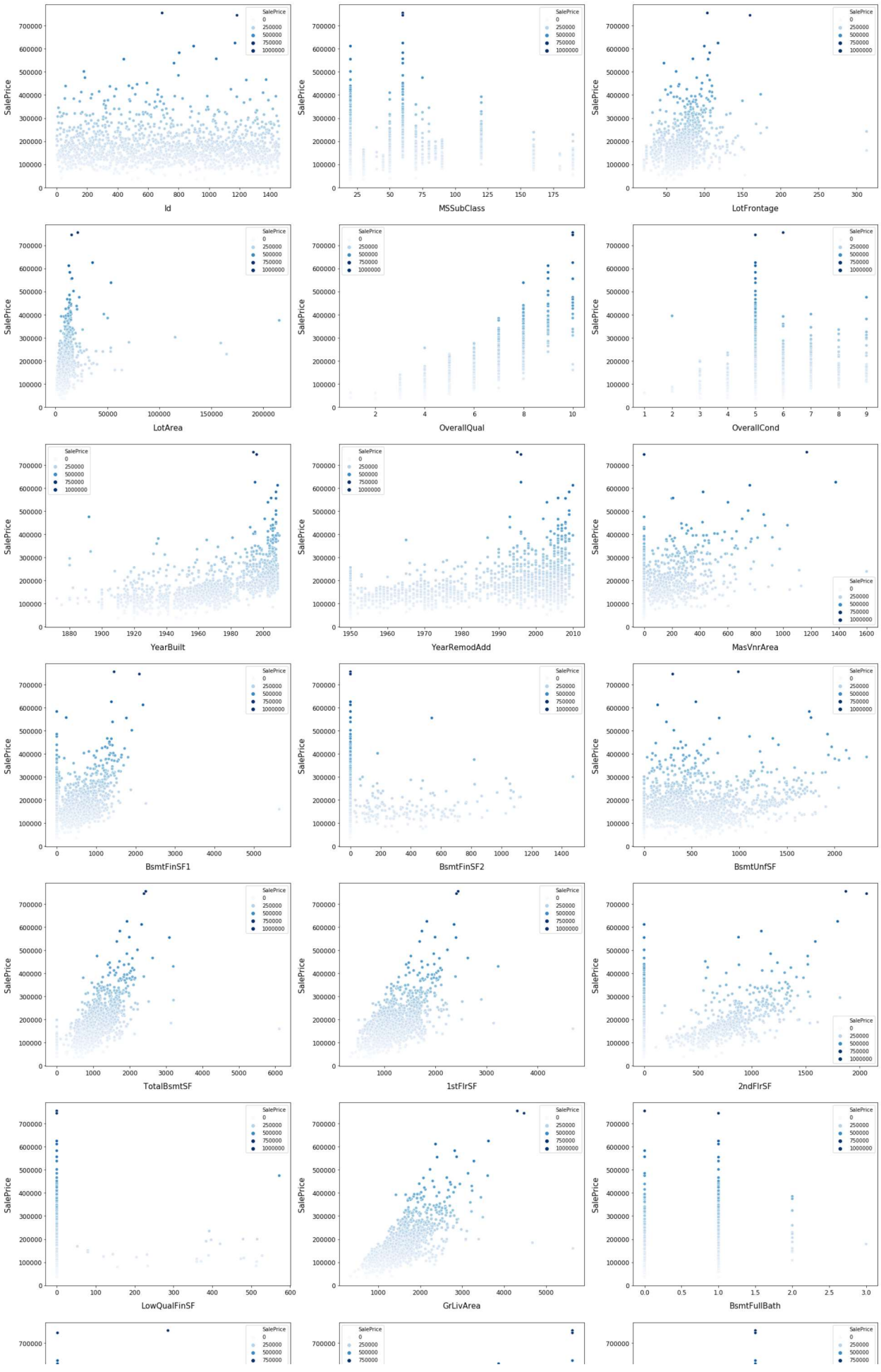
In [4]:

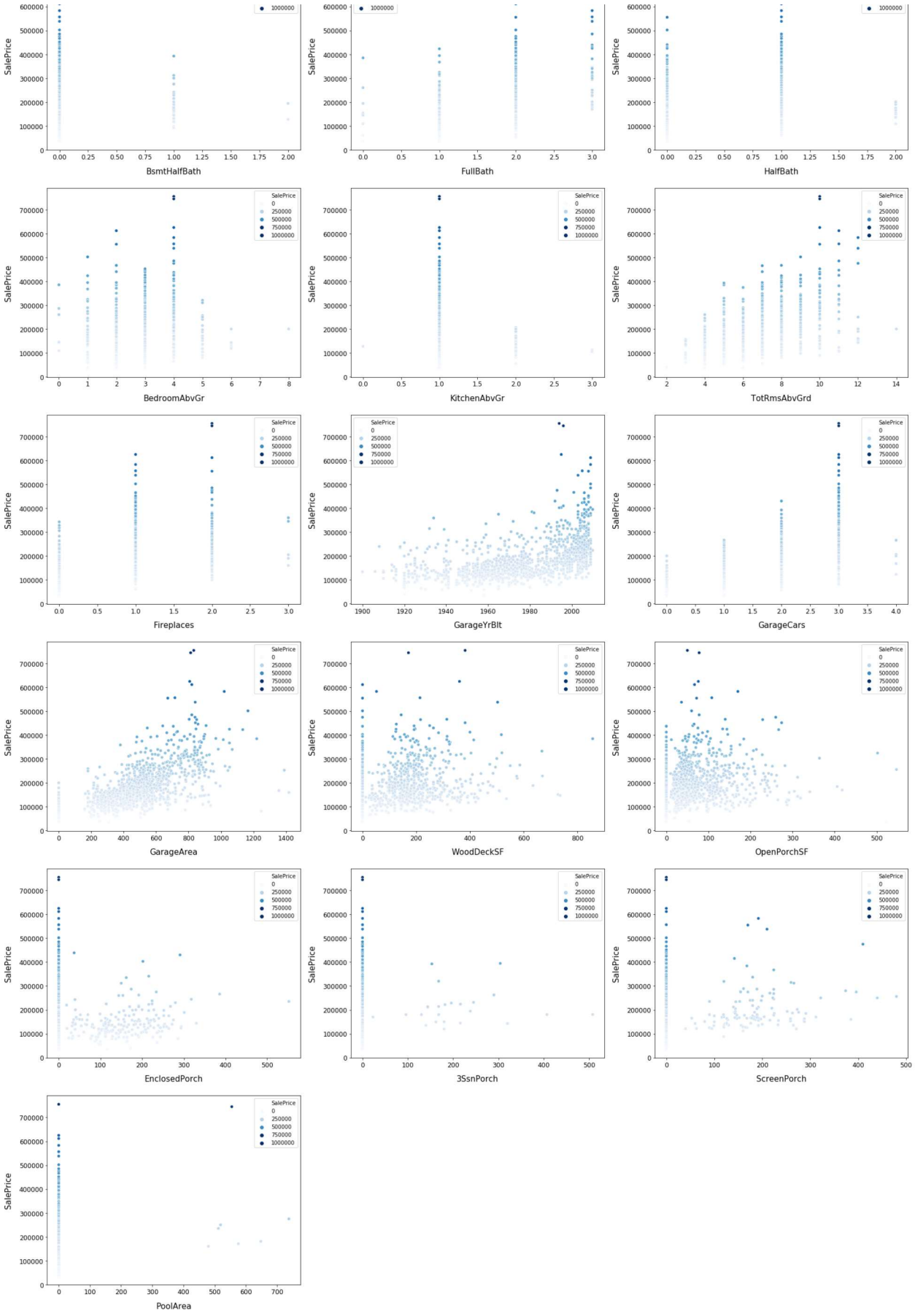
```
print(train_df.shape)
print(test_df.shape)
```

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

In [5]:

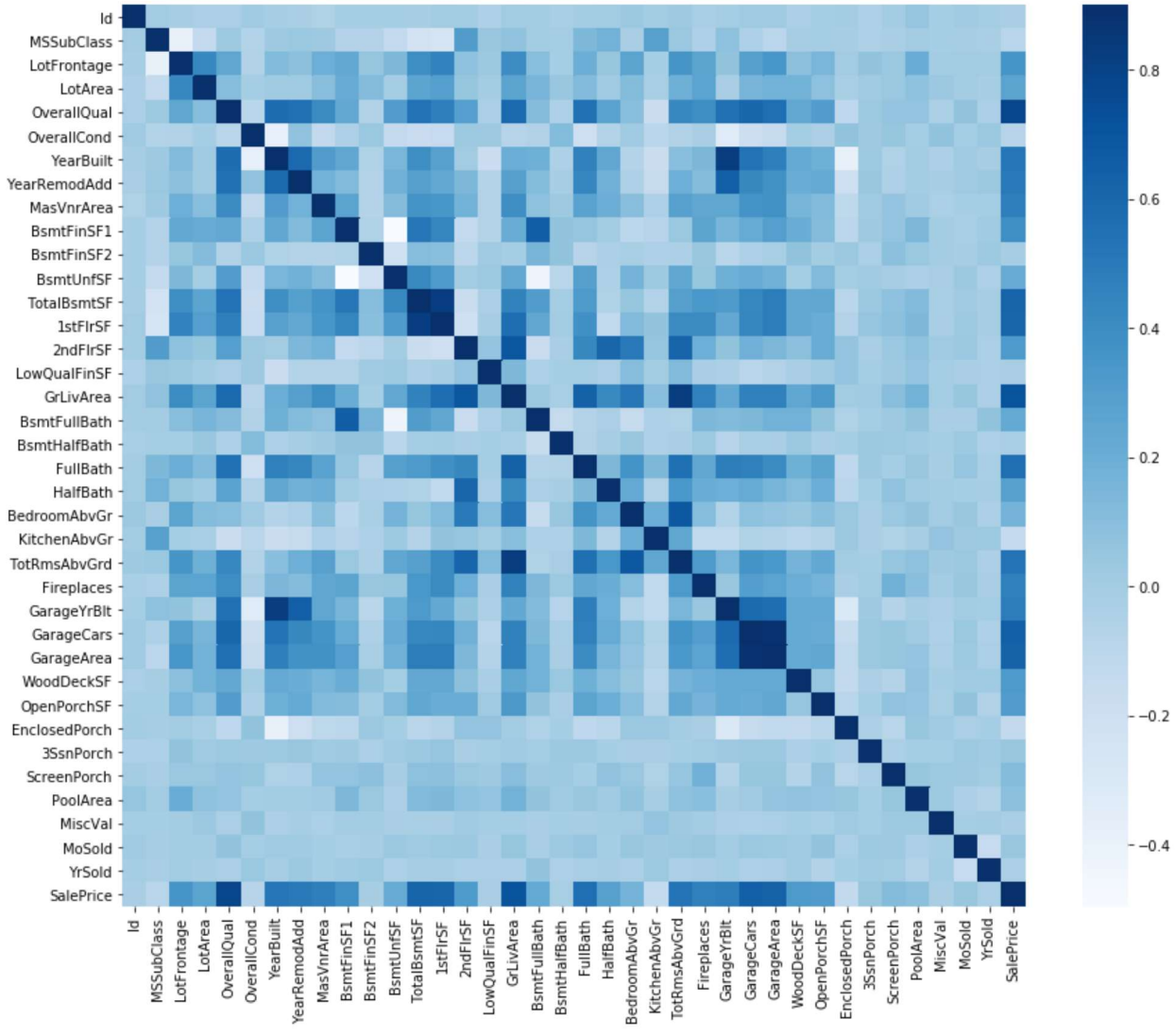
```
# Finding numeric features
numeric_dtypes = ['int16', 'int32', 'int64', 'float16', 'float32', 'float
64']
numeric = []
for i in train_df.columns:
    if train_df[i].dtype in numeric_dtypes:
        if i in ['TotalSF', 'Total_Bathrooms','Total_porch_sf','haspool',
'hasgarage','hasbsmt','hasfireplace']:
            pass
        else:
            numeric.append(i)
# visualising some more outliers in the data values
fig, axs = plt.subplots(ncols=2, nrows=0, figsize=(12, 120))
plt.subplots_adjust(right=2)
plt.subplots_adjust(top=2)
sns.color_palette("husl", 8)
for i, feature in enumerate(list(train_df[numeric]), 1):
    if(feature=='MiscVal'):
        break
    plt.subplot(len(list(numeric)), 3, i)
    sns.scatterplot(x=feature, y='SalePrice', hue='SalePrice', palette='B
lues', data=train_df)

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

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

    plt.legend(loc='best', prop={'size': 10})

plt.show()
```


SalePrice
Id
MSSubClass
LotFrontage
LotArea
OverallQual
OverallCond
YearBuilt
YearRemodAdd
MasVnrArea
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
LowQualFinSF
GrLivArea
BsmtFullBath


SalePrice
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageYrBlt
GarageCars
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
ScreenPorch
PoolArea

In [6]:
```
corr = train_df.corr()
plt.subplots(figsize=(15,12))
sns.heatmap(corr, vmax=0.9,cmap='Blues', square=True)
```

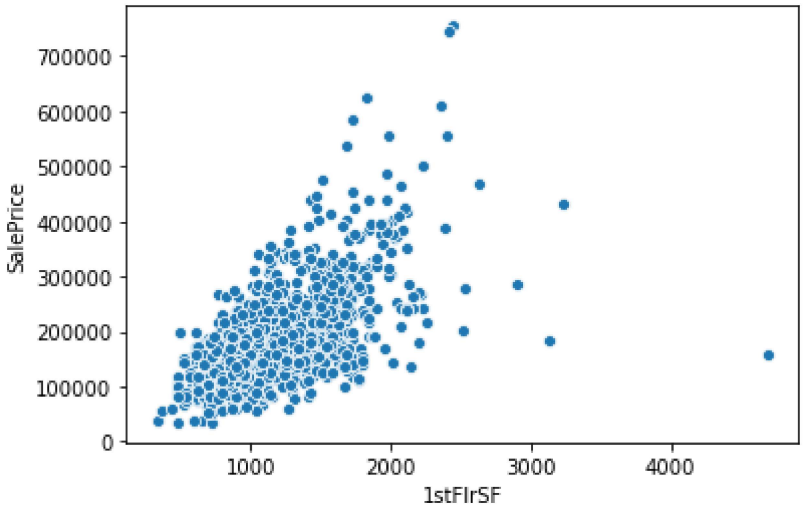
Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b1067e7d0>
```



Lets see how some of the features are related to SalePrice

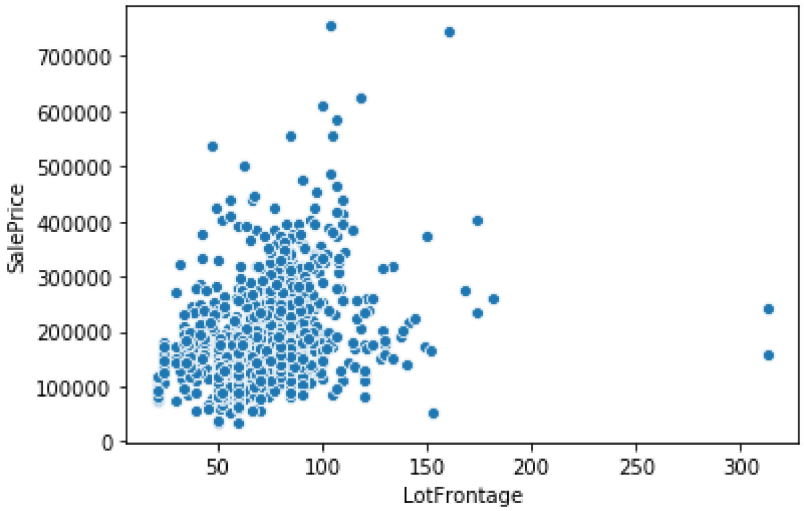
In [7]:
```
sns.scatterplot(x=train_df['1stFlrSF'],y=train_df['SalePrice'])
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b10b34a10>
```



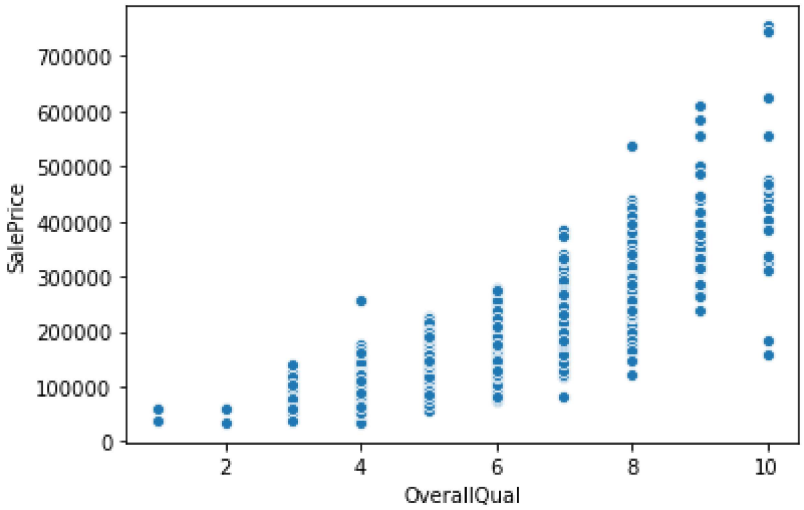
In [8]:
```
sns.scatterplot(x=train_df['LotFrontage'],y=train_df['SalePrice'])
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b10aa2b10>
```



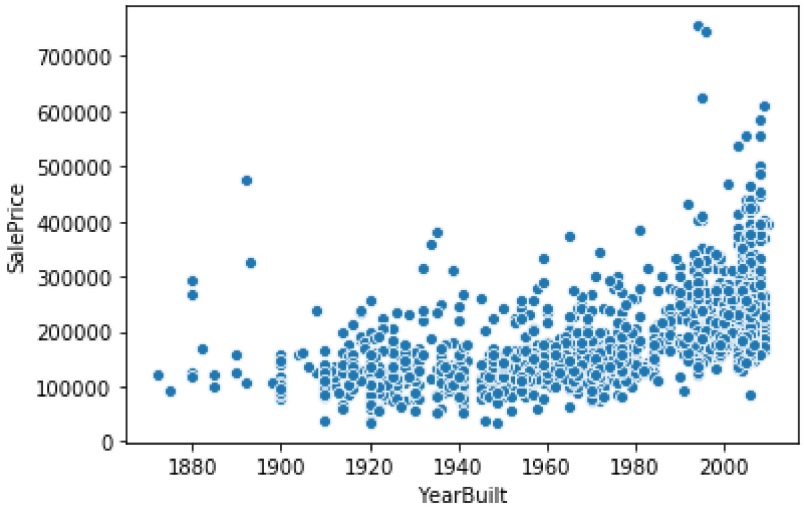
In [9]:
```
sns.scatterplot(x=train_df['OverallQual'],y=train_df['SalePrice'])
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b10a09a90>
```



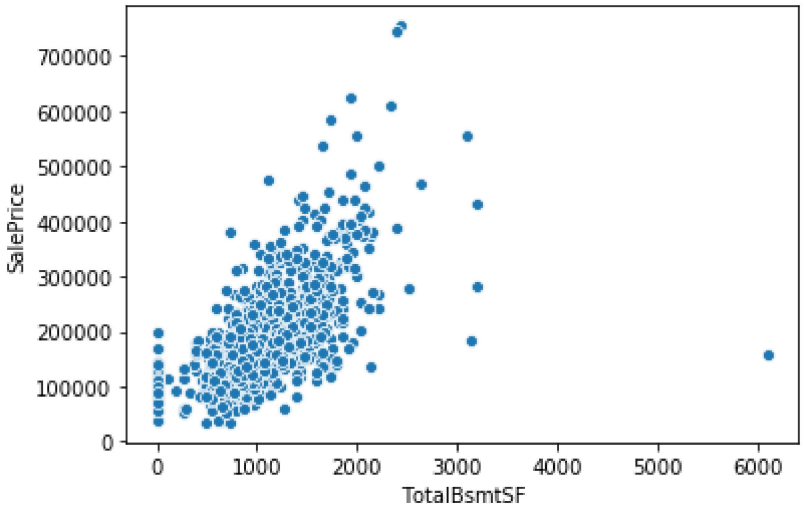
In [10]:
```
sns.scatterplot(x=train_df['YearBuilt'],y=train_df['SalePrice'])
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b109f16d0>
```



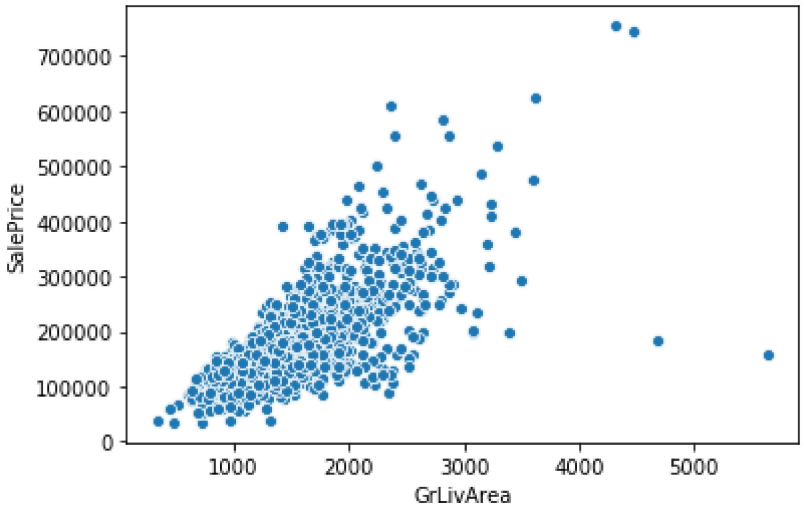
In [11]:
```
sns.scatterplot(x=train_df['TotalBsmtSF'],y=train_df['SalePrice'])
```

Out[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b10a83c90>
```



In [12]:
```
sns.scatterplot(x=train_df['GrLivArea'],y=train_df['SalePrice'])
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b108d3ed0>
```

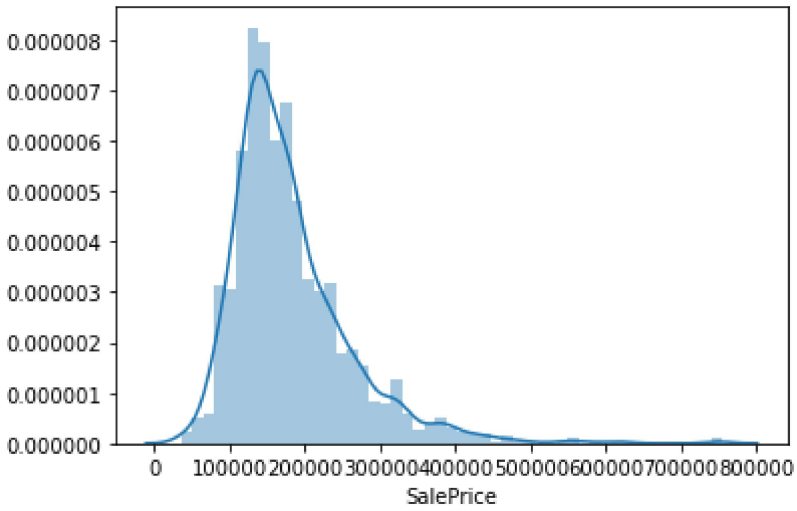


#Lets drop the unwanted features

In [13]:
```
train_df.drop(['Id'], axis=1, inplace=True)
test_df.drop(['Id'], axis=1, inplace=True)
```

In [14]:
```
sns.distplot(train_df['SalePrice'])
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b271f19d0>
```



In [15]:
```
train_df['SalePrice'] = np.log(train_df['SalePrice'])
```

In [16]:
```
cat_cols = [x for x in train_df.columns if train_df[x].dtype == 'object']
num_cols = [x for x in train_df.columns if train_df[x].dtype != 'object']
```

Removing Outliers

In [17]:
```
train_df.drop(train_df[(train_df['OverallQual']<5)&train_df['SalePrice']>
200000].index,inplace=True)
train_df.drop(train_df[(train_df['GrLivArea']<4500)&train_df['SalePrice']
>300000].index,inplace=True)
train_df.drop(train_df[(train_df['OpenPorchSF']<500)&train_df['SalePrice'
]>200000].index,inplace=True)
train_df.reset_index(drop=True,inplace=True)
```

In [18]:
```
y_train=train_df['SalePrice'].reset_index(drop=True)
X_train=train_df.drop(['SalePrice'],axis=1)
X_test=test_df
```

In [19]:
```
print('Unique values are:', train_df.MiscFeature.unique())
```

```
Unique values are: [nan 'Shed' 'Gar2' 'Othr' 'TenC']
```

In [20]:
```
#analysing the categorical data
categoricals = train_df.select_dtypes(exclude= [np.number])
categoricals.describe()
```

Out[20]:

	MSZoning	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	N
count	1460	1460	91	1460	1460	1460	1460	1460	1
unique	5	2	2	4	4	2	5	3	2
top	RL	Pave	Grvl	Reg	Lvl	AllPub	Inside	Gtl	N
freq	1151	1454	50	925	1311	1459	1052	1382	2

4 rows × 43 columns

In [21]:
```
#One-hot encoding to convert the categorical data into integer data
train_df['enc_street'] = pd.get_dummies(train_df.Street, drop_first= True
)
test_df['enc_street'] = pd.get_dummies(test_df.Street, drop_first= True)
```

In [22]:
```
print('Encoded: \n')
print(train_df.enc_street.value_counts())
```

```
Encoded:

1    1454
0       6
Name: enc_street, dtype: int64
```

In [23]:
```
def encode(x):
    if x == 'Partial':
        return 1
    else:
        return 0
#Treating partial as one class and other all sale condition as other
train_df['enc_condition'] = train_df.SaleCondition.apply(encode)
test_df['enc_condition'] = test_df.SaleCondition.apply(encode)
```

Filling Missing Values

In [24]:
```
df=pd.concat([X_train, X_test]).reset_index(drop=True)
df.shape
```

Out[24]:
```
(2919, 81)
```

In [25]:

```
def missing_val(dat):
    dict_x={}
    data = pd.DataFrame(dat)
    df_cols = list(pd.DataFrame(data))
    for i in range(0,len(df_cols)):
        dict_x.update({df_cols[i]:round(data[df_cols[i]].isnull().mean()*
100,2)})
    return dict_x
missing=missing_val(df)
df_missing=sorted(missing.items(),key=lambda x:x[1],reverse=True)
print('Percent of missing data')
df_missing[0:10]
```

```
Percent of missing data
```

Out[25]:

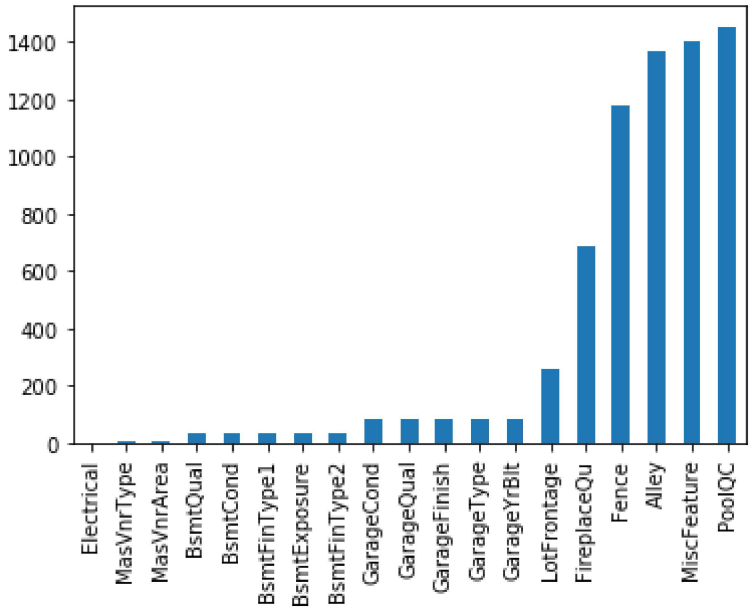
```
[('PoolQC', 99.66),
 ('MiscFeature', 96.4),
 ('Alley', 93.22),
 ('Fence', 80.44),
 ('enc_street', 50.02),
 ('enc_condition', 50.02),
 ('FireplaceQu', 48.65),
 ('LotFrontage', 16.65),
 ('GarageYrBlt', 5.45),
 ('GarageFinish', 5.45)]
```

In [26]:

```
missing = train_df.isnull().sum()
missing = missing[missing > 0]
missing.sort_values(inplace=True)
missing.plot.bar()
```

Out[26]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b107c4790>
```



In [27]:

```
def handle_missing(features):
    # the data description states that NA refers to typical ('Typ') values
    features['Functional'] = features['Functional'].fillna('Typ')
    # Replace the missing values in each of the columns below with their mo
de
    features['Electrical'] = features['Electrical'].fillna("SBrkr")
    features['KitchenQual'] = features['KitchenQual'].fillna("TA")
    features['Exterior1st'] = features['Exterior1st'].fillna(features['Ex
terior1st'].mode()[0])
    features['Exterior2nd'] = features['Exterior2nd'].fillna(features['Ex
terior2nd'].mode()[0])
    features['SaleType'] = features['SaleType'].fillna(features['SaleTyp
e'].mode()[0])
    features['MSZoning'] = features.groupby('MSSubClass')['MSZoning'].tra
nsform(lambda x: x.fillna(x.mode()[0]))

    # the data description stats that NA refers to "No Pool"
    features["PoolQC"] = features["PoolQC"].fillna("None")
    # Replacing the missing values with 0, since no garage = no cars in gar
age
    for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):
        features[col] = features[col].fillna(0)
    # Replacing the missing values with None
    for col in ['GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'
]:
        features[col] = features[col].fillna('None')
    # NaN values for these categorical basement features, means there's no
 basement
    for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1',
'BsmtFinType2'):
        features[col] = features[col].fillna('None')

    # Group the by neighborhoods, and fill in missing value by the median L
otFrontage of the neighborhood
    features['LotFrontage'] = features.groupby('Neighborhood')['LotFronta
ge'].transform(lambda x: x.fillna(x.median()))

    # We have no particular intuition around how to fill in the rest of the
categorical features
    # So we replace their missing values with None
    objects = []
```

```
    for i in features.columns:
        if features[i].dtype == object:
            objects.append(i)
    features.update(features[objects].fillna('None'))

    # And we do the same thing for numerical features, but this time with 0
s
    numeric_dtypes = ['int16', 'int32', 'int64', 'float16', 'float32', 'f
loat64']
    numeric = []
    for i in features.columns:
        if features[i].dtype in numeric_dtypes:
            numeric.append(i)
    features.update(features[numeric].fillna(0))
    return features
handle_missing(df)
```

Out[27]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour
0	60	RL	65.0	8450	Pave	None	Reg	Lvl
1	20	RL	80.0	9600	Pave	None	Reg	Lvl
2	60	RL	68.0	11250	Pave	None	IR1	Lvl
3	70	RL	60.0	9550	Pave	None	IR1	Lvl
4	60	RL	84.0	14260	Pave	None	IR1	Lvl
...	...	...	...	...	...	...	...	...
2914	160	RM	21.0	1936	Pave	None	Reg	Lvl
2915	160	RM	21.0	1894	Pave	None	Reg	Lvl
2916	20	RL	160.0	20000	Pave	None	Reg	Lvl
2917	85	RL	62.0	10441	Pave	None	Reg	Lvl
2918	60	RL	74.0	9627	Pave	None	Reg	Lvl

2919 rows × 81 columns

In [28]:
```
print('Number of Numerical Columns:',len(num_cols))
print('Number of Categorical Columns:',len(cat_cols))
```

```
Number of Numerical Columns: 37
Number of Categorical Columns: 43
```

In [29]:
```
num_cols.remove('SalePrice')
```

In [30]:
```
num_cat = ['MSSubClass', 'OverallQual', 'OverallCond', 'MoSold', 'YrSold'
]
for i in num_cat:
    df[i] = df[i].apply(str)
num_only = list(set(num_cols)-set(num_cat))
```

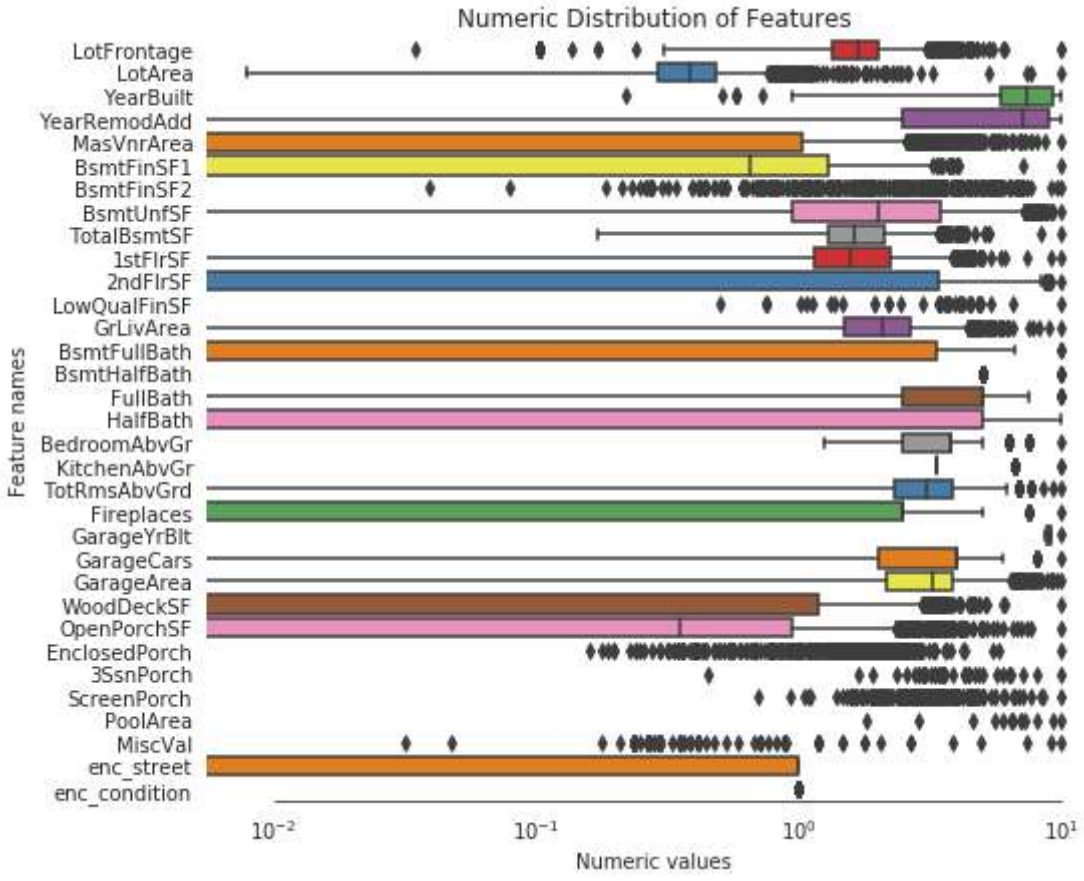
Feature Transformation

In [31]:
```
#Lets normalize numeric coloumns with Normalizer
from sklearn.preprocessing import MinMaxScaler
mm_scaler = MinMaxScaler(feature_range = (0,10))
df[num_only] = mm_scaler.fit_transform(df[num_only])
```

In [32]:
```
numeric_dtypes = ['int16', 'int32', 'int64', 'float16', 'float32', 'float
64']
numeric = []
for i in df.columns:
    if df[i].dtype in numeric_dtypes:
        numeric.append(i)
```

In [33]:

```
# Create box plots for all numeric features
sns.set_style("white")
f, ax = plt.subplots(figsize=(8, 7))
ax.set_xscale("log")
ax = sns.boxplot(data=df[numeric] , orient="h", palette="Set1")
ax.xaxis.grid(False)
ax.set(ylabel="Feature names")
ax.set(xlabel="Numeric values")
ax.set(title="Numeric Distribution of Features")
sns.despine(trim=True, left=True)
```



In [34]:
```
# Find skewed numerical features
skew_features = df[numeric].apply(lambda x: skew(x)).sort_values(ascendin
g=False)

high_skew = skew_features[skew_features > 0.5]
skew_index = high_skew.index

print("There are {} numerical features with Skew > 0.5 :".format(high_ske
w.shape[0]))
skewness = pd.DataFrame({'Skew' :high_skew})
skew_features.head(10)
```

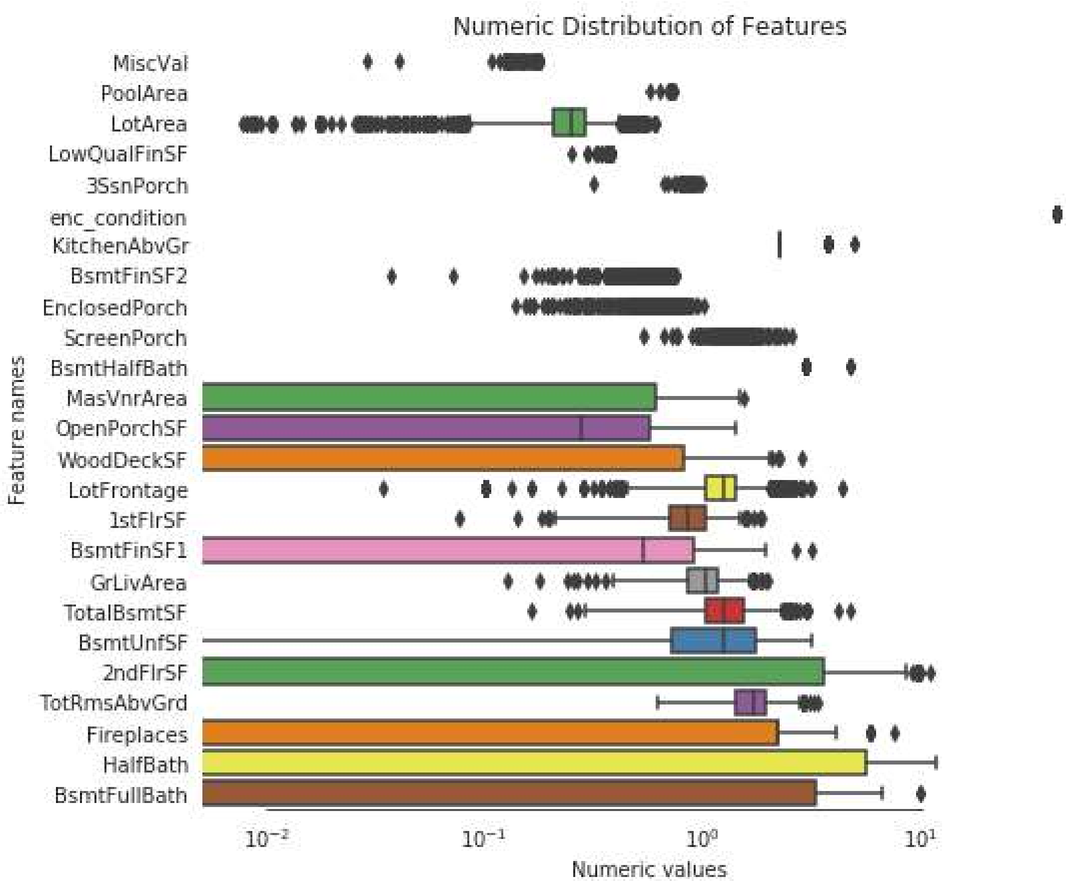
```
There are 25 numerical features with Skew > 0.5 :
```

Out[34]:
```
MiscVal          21.947195
PoolArea         16.898328
LotArea          12.822431
LowQualFinSF     12.088761
3SsnPorch        11.376065
enc_condition     4.622540
KitchenAbvGr      4.302254
BsmtFinSF2        4.146143
EnclosedPorch     4.003891
ScreenPorch       3.946694
dtype: float64
```

In [35]:

```
for i in skew_index:
    df[i] = boxcox1p(df[i], boxcox_normmax(df[i] + 1))
# Let's make sure we handled all the skewed values
sns.set_style("white")
f, ax = plt.subplots(figsize=(8, 7))
ax.set_xscale("log")
ax = sns.boxplot(data=df[skew_index] , orient="h", palette="Set1")
ax.xaxis.grid(False)
ax.set(ylabel="Feature names")
ax.set(xlabel="Numeric values")
ax.set(title="Numeric Distribution of Features")
sns.despine(trim=True, left=True)
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/categorical.py:483: Use
rWarning: Data has no positive values, and therefore cannot be log-sca
led.
  **kws)
/opt/conda/lib/python3.7/site-packages/seaborn/categorical.py:483: Use
rWarning: Data has no positive values, and therefore cannot be log-sca
led.
  **kws)
/opt/conda/lib/python3.7/site-packages/seaborn/categorical.py:483: Use
rWarning: Data has no positive values, and therefore cannot be log-sca
led.
  **kws)
/opt/conda/lib/python3.7/site-packages/seaborn/categorical.py:483: Use
rWarning: Data has no positive values, and therefore cannot be log-sca
led.
  **kws)
/opt/conda/lib/python3.7/site-packages/seaborn/categorical.py:483: Use
rWarning: Data has no positive values, and therefore cannot be log-sca
led.
  **kws)
```



In [36]:
```
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()
df[num_cat] = oe.fit_transform(df[num_cat])
```

In [37]:
```
df[num_cols].tail()
```

Out[37]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemod
2914	2.0	0.000000	0.028614	4.0	6.0	7.101449	3.333333
2915	2.0	0.000000	0.026791	4.0	4.0	7.101449	3.333333
2916	5.0	2.655371	0.395243	5.0	6.0	6.376812	7.666667
2917	14.0	1.073845	0.270858	5.0	4.0	8.695652	7.000000
2918	10.0	1.315769	0.255281	7.0	4.0	8.768116	7.333333

5 rows × 36 columns

In [38]:
```
from sklearn.feature_extraction import FeatureHasher
fh = FeatureHasher(n_features = 3, input_type = 'string')
df_cat_cols = pd.DataFrame()
```

In [39]:
```
for i in cat_cols:
    df_cat_cols = pd.concat([pd.DataFrame(fh.fit_transform(df[i]).toarray
()).add_prefix(i+'_'),df_cat_cols], axis = 1)

df_cat_cols
```

Out[39]:

	SaleCondition_0	SaleCondition_1	SaleCondition_2	SaleType_0	SaleType_1	SaleType_2
0	0.0	3.0	1.0	0.0	0.0	0.0
1	0.0	3.0	1.0	0.0	0.0	0.0
2	0.0	3.0	1.0	0.0	0.0	0.0
3	-1.0	4.0	0.0	0.0	0.0	0.0
4	0.0	3.0	1.0	0.0	0.0	0.0
...	...	...	...	...	...	...
2914	0.0	3.0	1.0	0.0	0.0	0.0
2915	-1.0	4.0	0.0	0.0	0.0	0.0
2916	-1.0	4.0	0.0	0.0	0.0	0.0
2917	0.0	3.0	1.0	0.0	0.0	0.0
2918	0.0	3.0	1.0	0.0	0.0	0.0

2919 rows × 129 columns

In [40]:
```
df = pd.concat([df[num_cols].reset_index(drop = True), df_cat_cols], axis
= 1)
```

In [41]:
```
train = df[:y_train.shape[0]]
test = df[y_train.shape[0]:]
```

In [42]:
```
train.shape,test.shape
```

Out[42]:
```
((1460, 165), (1459, 165))
```

In [43]:
```
for i in train.columns:
    if train[i].dtype == 'object':
        print(i)
```

no output means we are ready to train models

Train Models

In [44]:
```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(train, y_train, test_si
ze = 0.2)
```

In [45]:
```
X_train.shape,y_train.shape,X_val.shape,y_val.shape
```

Out[45]:
```
((1168, 165), (1168,), (292, 165), (292,))
```

In [46]:
```
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.model_selection import KFold, cross_val_score, RandomizedSea
rchCV
from sklearn.metrics import make_scorer, mean_squared_error
#function for error calculation
def rmsle_func(actual, pred):
    return np.sqrt(((np.log(pred + 1) - np.log(actual + 1))**2).mean())

rmsle = make_scorer(rmsle_func, greater_is_better = False)
```

Defining functions for error calculations

In [47]:

```
#function to get best parameters for our model
def generate_clf(clf, params, x, y):
    rs = RandomizedSearchCV(clf, params)
    rs_obj = rs.fit(x, y)
    best_rs = rs_obj.best_estimator_
    print('Best parameters:',rs_obj.best_params_)
    pred = rs.predict(X_valid)
    print('Training score:',rs.score(x, y))
    print('Validation score:',rs.score(X_valid, y_valid))
    print('Validation RMSLE error:',rmsle_func(pred, y_valid))
    kf = KFold(n_splits = 5, shuffle = True)
    return np.sqrt(-1 * cross_val_score(best_rs, x, y, cv = kf, scoring =
rmsle)) , rs_obj.best_params_
```

In [48]:

```
from lightgbm import LGBMRegressor

# lgbm_clf = LGBMRegressor()

# lgbm_params = {'application':['regression'],
#                'num_iterations':[500, 700, 1000, 1500],
#                'max_depth':[3, 5, 7, 9],
#                'min_data_in_leaf':[4, 5, 6, 7, 8],
#                'feature_fraction':[0.2, 0.3, 0.4],
#                'bagging_fraction':[0.6, 0.7, 0.8, 0.9],
#                 'num_leaves':[5, 7, 10],
#                'learning_rate':[0.01, 0.1, 0.02, 0.05, 0.03]}

# lgbm_score ,lgbm_best_params = generate_clf(lgbm_clf, lgbm_params, X_trai
n, y_train)
# print('LGBM RMSLE on Complete Train set:',lgbm_score.mean())

lgbm_clf_final = LGBMRegressor(application = 'regression', num_iterations
= 1000, max_depth = 7, num_leaves = 70)
# lgbm_clf_final = LGBMRegressor(**lgbm_best_params)
```

In [49]:
```
from xgboost import XGBRegressor

# xgb_clf = XGBRegressor()

# xgb_params = {'n_etimators':[2000, 2500, 3000, 3500, 4000],
#               'gamma':[0.02, 0.04, 0.05, 1],
#               'max_depth':[3, 5, 7, 9],
#               'alpha':[0.02, 0.04, 0.05, 1],
#               'eta':[0.02, 0.04, 0.05, 0.1]}

# xgb_score , xgb_best_params = generate_clf(xgb_clf, xgb_params, train, ta
rget)
# print('XGB RMSLE on Complete Train set:',xgb_score.mean())
xgb_clf_final = XGBRegressor(n_estimators = 3000)
# xgb_clf_final = XGBRegressor(**xgb_best_params)
```

In [50]:
```
from sklearn.linear_model import Lasso

# las_clf = Pipeline([('scaler', RobustScaler()),
#                     ('clf', Lasso())])
# las_clf = Lasso()

# las_params = {'alpha':[1e-4, 1e-3, 1e-2, 0.1, 0.05]}
# las_score , las_best_params = generate_clf(las_clf, las_params, train, ta
rget)
# print('Lasso RMSLE on Training set:',las_score.mean())

las_clf_final = Lasso(alpha = 0.01)
# las_clf_final = Lasso(**las_best_params)
```

In [51]:
```
target=train_df['SalePrice']
```

In [52]:
```
from sklearn.ensemble import StackingRegressor

my_estimators = [#('lgbm', lgbm_clf_final),
                 ('xgb', xgb_clf_final)]

streg_clf = StackingRegressor(estimators = my_estimators, final_estimator
= las_clf_final)
streg_clf.fit(train, target)
streg_clf_pred = np.exp(streg_clf.predict(test))
```

In [53]:
```
print(streg_clf.score(X_train, y_train))
print(streg_clf.score(X_val, y_val))
print(streg_clf.score(train, target))
```

```
0.9923364142632347
0.992341299927496
0.9923415235690766
```

In [54]:
```
final_results = streg_clf_pred
final_results
```

Out[54]:
```
array([133134.5 , 160585.2 , 188606.83, ..., 175485.98, 132887.28,
       217874.72], dtype=float32)
```

In [55]:
```
submission_data = pd.read_csv('/kaggle/input/house-prices-advanced-regression-techniques/sample_submission.csv')
submission_data['SalePrice'] = final_results
submission_data.to_csv('output.csv', index = False)

submission_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Id         1459 non-null   int64  
 1   SalePrice  1459 non-null   float32
dtypes: float32(1), int64(1)
memory usage: 17.2 KB
```

In [56]:
```
submission_data.head()
```

Out[56]:

	Id	SalePrice
0	1461	133134.500000
1	1462	160585.203125
2	1463	188606.828125
3	1464	198718.984375
4	1465	187197.265625

In [ ]:
```

```