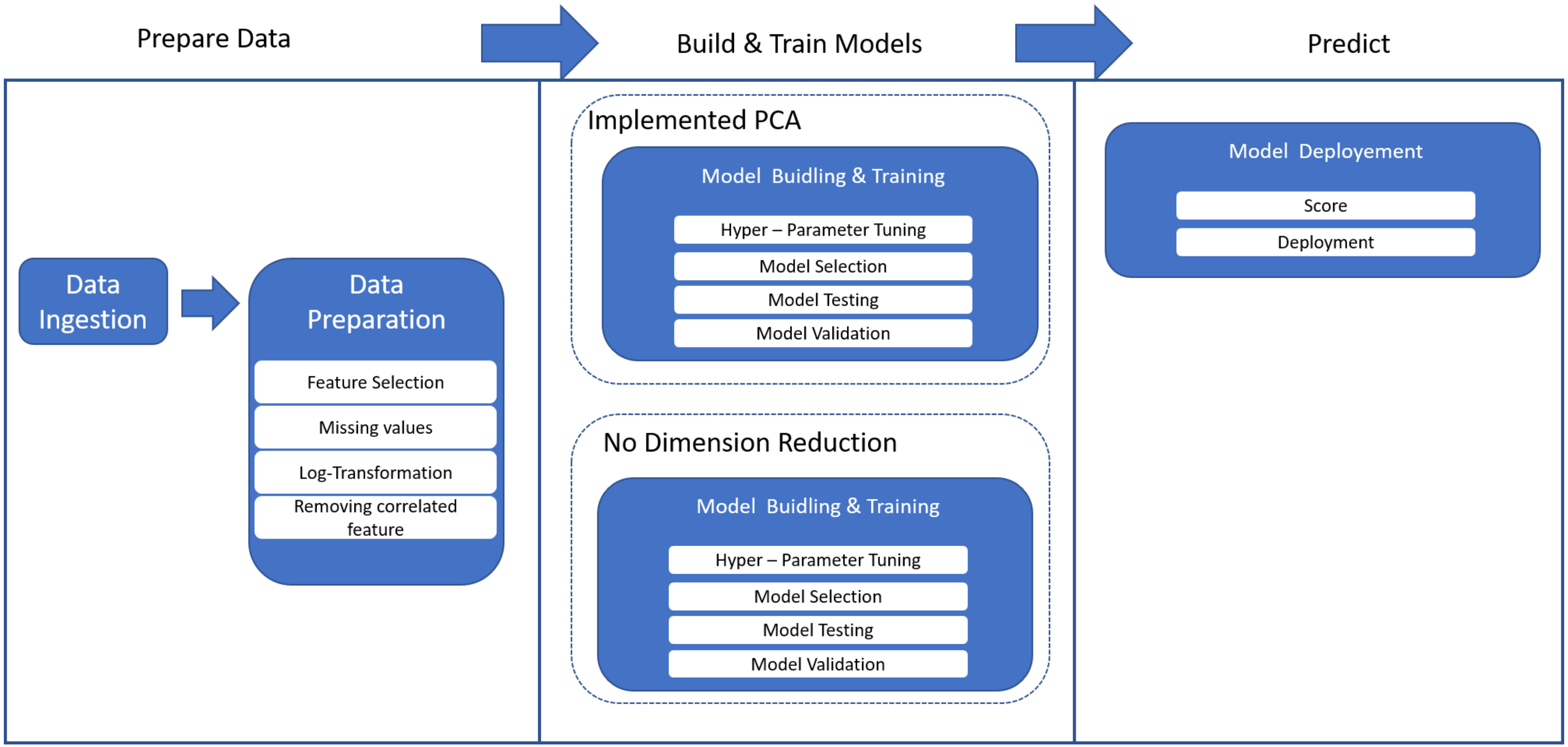
Objective

To build a regression model that best fit and predict the HousePricePred dataset

Pipeline



Data Preparation

- Feature selection

- Imputing missing values
- Log-transformation
- removing outliers

Models Used

- No Dimension Reduction
  - Linear Regression
  - XGBRegressor
  - CatBoostRegressor
  - Decision TreeRegressor
  - KNeighborsRegressor
  - LightGBM
  - SVM
  - Ridge
  - AutoML - 10 generation
  - AutoML - 15 generation
- With PCA
  - Linear Regression
  - XGBRegressor
  - CatBoostRegressor
  - Decision TreeRegressor
  - KNeighborsRegressor
  - LightGBM
  - SVM
  - Ridge

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from tqdm.notebook import tqdm
tqdm.pandas()

from scipy.stats import norm
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
pd.options.display.max_columns = 999
pd.options.display.max_rows = 20
```

In [3]:
```
df = pd.read_csv('data (1).csv')
```

In [4]:
```
df.head()
```

Out[4]:

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

In [6]:
```
df.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
```

In [7]:

```
for column in df:
    if df[column].dtype =="float":
        df[column]=pd.to_numeric(df[column],downcast="float")
    if df[column].dtype=="int64":
        df[column] = pd.to_numeric(df[column],downcast = "integer")
```

In [6]:
```
df.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   int16  
 1   MSSubClass     1460 non-null   int16  
 2   MSZoning       1460 non-null   object 
 3   LotFrontage    1201 non-null   float32
 4   LotArea        1460 non-null   int32  
 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   int8   
 18  OverallCond    1460 non-null   int8   
 19  YearBuilt      1460 non-null   int16  
 20  YearRemodAdd   1460 non-null   int16  
 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   float32
 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   int16
 35  BsmtFinType2   1422 non-null   object
 36  BsmtFinSF2     1460 non-null   int16
 37  BsmtUnfSF      1460 non-null   int16
 38  TotalBsmtSF    1460 non-null   int16
 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   int16
 44  2ndFlrSF       1460 non-null   int16
 45  LowQualFinSF   1460 non-null   int16
 46  GrLivArea      1460 non-null   int16
 47  BsmtFullBath   1460 non-null   int8
 48  BsmtHalfBath   1460 non-null   int8
 49  FullBath       1460 non-null   int8
 50  HalfBath       1460 non-null   int8
 51  BedroomAbvGr   1460 non-null   int8
 52  KitchenAbvGr   1460 non-null   int8
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int8
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int8
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float32
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int8
 62  GarageArea     1460 non-null   int16
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int16
 67  OpenPorchSF    1460 non-null   int16
 68  EnclosedPorch  1460 non-null   int16
 69  3SsnPorch      1460 non-null   int16
 70  ScreenPorch    1460 non-null   int16
 71  PoolArea       1460 non-null   int16
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int16
```

```
 76  MoSold         1460 non-null   int8
 77  YrSold         1460 non-null   int16
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int32
dtypes: float32(3), int16(21), int32(2), int8(12), object(43)
memory usage: 596.1+ KB
```

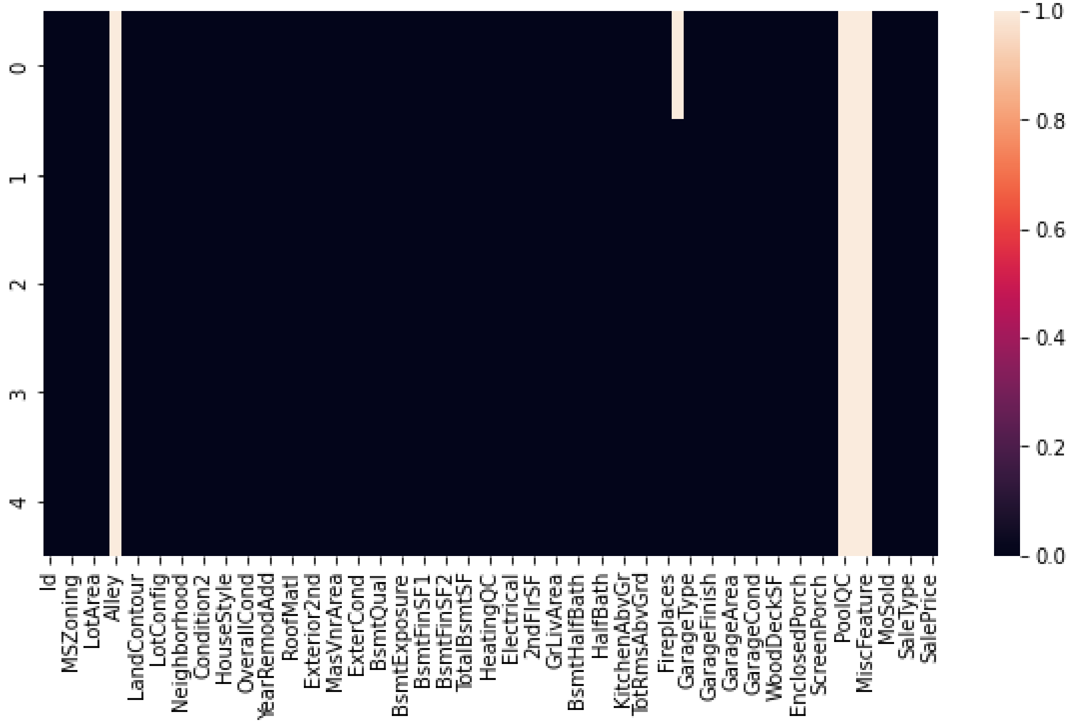
Benefit, increase the processing speed, decreasing file size

Basic Checks

Checking for missing values in the dataframe

In [8]:
```
plt.figure(figsize = (10,5))
sns.heatmap(df.head().isna())
```

Out[8]: <AxesSubplot:>



In [9]:
```
null_df = pd.DataFrame(df.isna().sum())
null_df.reset_index(drop = False, inplace = True)
null_df[null_df[0] != 0].rename(columns = {'index':'Feature', 0:'Count of missing values'})
```

Out[9]:

	Feature	Count of missing values
3	LotFrontage	259
6	Alley	1369
25	MasVnrType	8
26	MasVnrArea	8
30	BsmtQual	37
31	BsmtCond	37
32	BsmtExposure	38
33	BsmtFinType1	37
35	BsmtFinType2	38
42	Electrical	1
57	FireplaceQu	690
58	GarageType	81
59	GarageYrBlt	81
60	GarageFinish	81
63	GarageQual	81
64	GarageCond	81
72	PoolQC	1453
73	Fence	1179
74	MiscFeature	1406

- There is no values for Misc Feature
    - Drop columns since it does not have any values

In [10]:
```
df.drop(columns = ['MiscFeature'], inplace = True)
```

In [11]:
```
temp_dict = {'Feature':[], 'Len':[], 'Values':[]}
for x in df:
    temp_dict['Feature'].append(x)
    temp_dict['Len'].append(len(df[x].unique()))
    temp_dict['Values'].append(df[x].unique())
```

In [12]:
```
data_field = pd.DataFrame(temp_dict)
```

Checking the length, unique values, missing values, data types, and percentage of missing values

In [13]:
```
data_field = data_field.sort_values(by = ['Len'], ascending = False).reset_index(drop = True)
data_field = data_field.merge(null_df, left_on = 'Feature', right_on = 'index').drop(columns = 'index').rename(columns =
```

In [14]:
```
data_field['Data Types'] = data_field.progress_apply(lambda x: df[x['Feature']].dtype, axis = 1)
```

```
  0%|          | 0/80 [00:00<?, ?it/s]
```

In [15]:
```
data_field['Percentage of missing values'] = data_field.progress_apply(lambda x: x['Missing Values'] / len(df) * 100,axi
```

```
  0%|          | 0/80 [00:00<?, ?it/s]
```

In [16]:
```
data_field.head()
```

Out[16]:

	Feature	Len	Values	Missing Values	Data Types	Percentage of missing values
0	Id	1460	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	0	int16	0.0
1	LotArea	1073	[8450, 9600, 11250, 9550, 14260, 14115, 10084,...	0	int32	0.0
2	GrLivArea	861	[1710, 1262, 1786, 1717, 2198, 1362, 1694, 209...	0	int16	0.0
3	BsmtUnfSF	780	[150, 284, 434, 540, 490, 64, 317, 216, 952, 1...	0	int16	0.0
4	1stFlrSF	753	[856, 1262, 920, 961, 1145, 796, 1694, 1107, 1...	0	int16	0.0

Missing Data

Missing Data

In [17]:
```
missing_col_num_df = data_field[(data_field['Data Types'] != 'object') & (data_field['Missing Values'] != 0)]
missing_col_num = data_field[(data_field['Data Types'] != 'object') & (data_field['Missing Values'] != 0)]['Feature'].to
```

In [18]:
```
# numerical column missing values
missing_col_num_df
```

Out[18]:

	Feature	Len	Values	Missing Values	Data Types	Percentage of missing values
10	MasVnrArea	328	[196.0, 0.0, 162.0, 350.0, 186.0, 240.0, 286.0...	8	float32	0.547945
16	LotFrontage	111	[65.0, 80.0, 68.0, 60.0, 84.0, 85.0, 75.0, nan...	259	float32	17.739726
17	GarageYrBlt	98	[2003.0, 1976.0, 2001.0, 1998.0, 2000.0, 1993....	81	float32	5.547945

In [19]:
```
# replacing missing values in numeric columns with average value since the missing percentage is not too high
for x in missing_col_num:
    df[x].fillna(df[x].mean(), inplace = True)
```

In [20]:
```python
# categorical column missing values
missing_col_cat = data_field[(data_field['Data Types'] == 'object') & (data_field['Missing Values'] != 0)].sort_values(by
missing_col_cat
```

Out[20]:

	Feature	Len	Values	Missing Values	Data Types	Percentage of missing values
42	Electrical	6	[SBrkr, FuseF, FuseA, FuseP, Mix, nan]	1	object	0.068493
56	MasVnrType	5	[BrkFace, None, Stone, BrkCmn, nan]	8	object	0.547945
39	BsmtFinType1	7	[GLQ, ALQ, Unf, Rec, BLQ, nan, LwQ]	37	object	2.534247
59	BsmtQual	5	[Gd, TA, Ex, nan, Fa]	37	object	2.534247
60	BsmtCond	5	[TA, Gd, nan, Fa, Po]	37	object	2.534247
38	BsmtFinType2	7	[Unf, BLQ, nan, ALQ, Rec, LwQ, GLQ]	38	object	2.602740
61	BsmtExposure	5	[No, Gd, Mn, Av, nan]	38	object	2.602740
40	GarageType	7	[Attchd, Detchd, BuiltIn, CarPort, nan, Basmen...	81	object	5.547945
43	GarageQual	6	[TA, Fa, Gd, nan, Ex, Po]	81	object	5.547945
44	GarageCond	6	[TA, Fa, nan, Gd, Po, Ex]	81	object	5.547945
69	GarageFinish	4	[RFn, Unf, Fin, nan]	81	object	5.547945
48	FireplaceQu	6	[nan, TA, Gd, Fa, Ex, Po]	690	object	47.260274
50	Fence	5	[nan, MnPrv, GdWo, GdPrv, MnWw]	1179	object	80.753425
76	Alley	3	[nan, Grvl, Pave]	1369	object	93.767123
65	PoolQC	4	[nan, Ex, Fa, Gd]	1453	object	99.520548

In [21]:
```python
# segmenting the categorical columns with less than 10 % missing values
missing_col_cat_less10_df = missing_col_cat[missing_col_cat['Percentage of missing values'] < 10]
missing_col_cat_less10 = missing_col_cat_less10_df['Feature'].tolist()

# segmenting the categorical columns with more than 10 % missing values
missing_col_cat_great10_df = missing_col_cat[missing_col_cat['Percentage of missing values'] > 10]
missing_col_cat_great10 = missing_col_cat_great10_df['Feature'].tolist()
```

In [22]:
```
# Less than 10% missing values
missing_col_cat_less10_df
```

Out[22]:

	Feature	Len	Values	Missing Values	Data Types	Percentage of missing values
42	Electrical	6	[SBrkr, FuseF, FuseA, FuseP, Mix, nan]	1	object	0.068493
56	MasVnrType	5	[BrkFace, None, Stone, BrkCmn, nan]	8	object	0.547945
39	BsmtFinType1	7	[GLQ, ALQ, Unf, Rec, BLQ, nan, LwQ]	37	object	2.534247
59	BsmtQual	5	[Gd, TA, Ex, nan, Fa]	37	object	2.534247
60	BsmtCond	5	[TA, Gd, nan, Fa, Po]	37	object	2.534247
38	BsmtFinType2	7	[Unf, BLQ, nan, ALQ, Rec, LwQ, GLQ]	38	object	2.602740
61	BsmtExposure	5	[No, Gd, Mn, Av, nan]	38	object	2.602740
40	GarageType	7	[Attchd, Detchd, BuiltIn, CarPort, nan, Basmen...	81	object	5.547945
43	GarageQual	6	[TA, Fa, Gd, nan, Ex, Po]	81	object	5.547945
44	GarageCond	6	[TA, Fa, nan, Gd, Po, Ex]	81	object	5.547945
69	GarageFinish	4	[RFn, Unf, Fin, nan]	81	object	5.547945

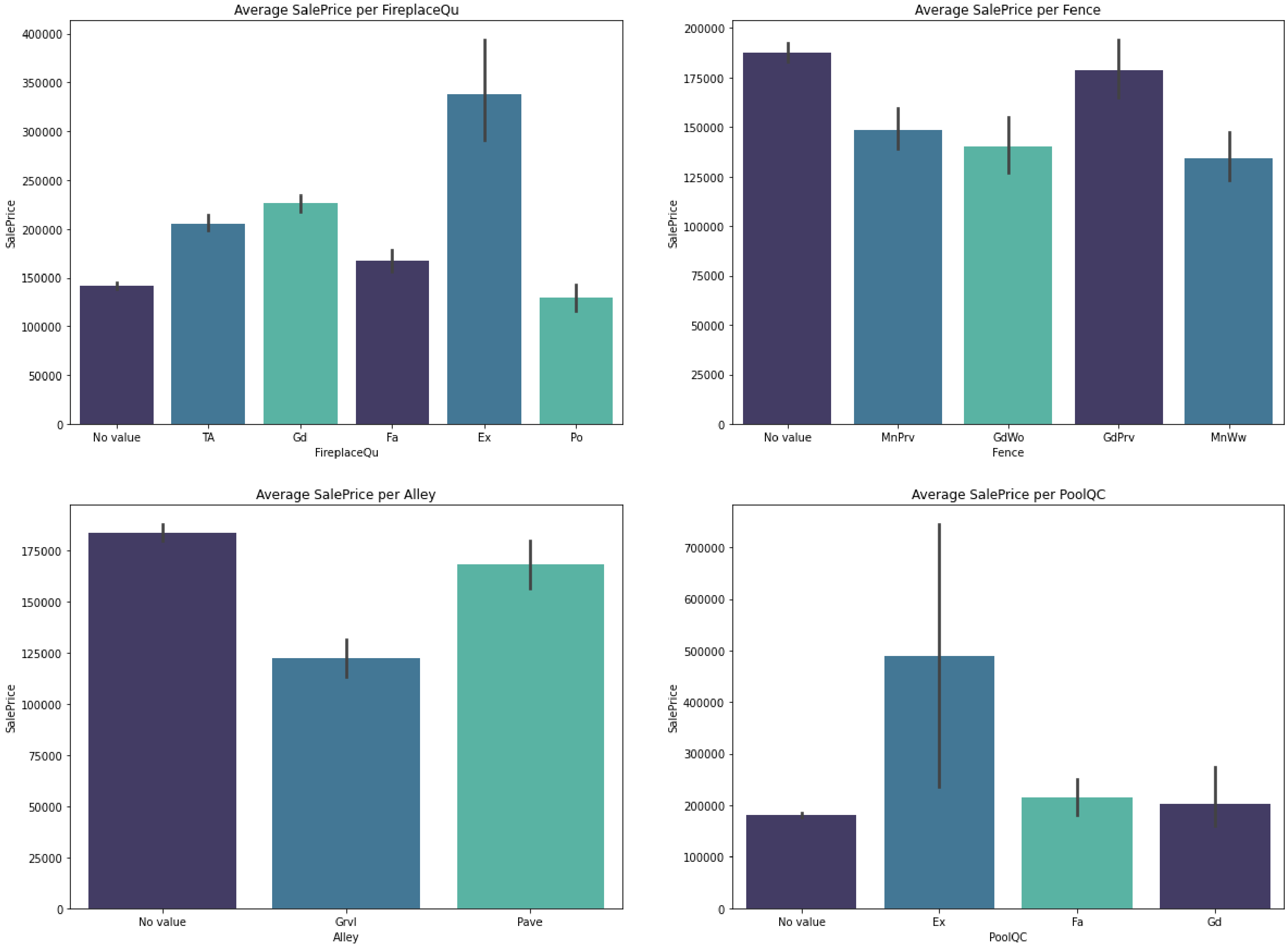
In [23]:
```
# for feature with less than 10% missing values, impute the mode directly since the missing percentage is not too high
for x in missing_col_cat_less10:
    df[x] = df[x].fillna(df[x].mode()[0])
```

In [24]:
```
# Greater than 10% missing values
missing_col_cat_great10_df
```

Out[24]:

	Feature	Len	Values	Missing Values	Data Types	Percentage of missing values
48	FireplaceQu	6	[nan, TA, Gd, Fa, Ex, Po]	690	object	47.260274
50	Fence	5	[nan, MnPrv, GdWo, GdPrv, MnWw]	1179	object	80.753425
76	Alley	3	[nan, Grvl, Pave]	1369	object	93.767123
65	PoolQC	4	[nan, Ex, Fa, Gd]	1453	object	99.520548

In [25]:
```
# Checking the releationship of high missing values column with the target
pal = sns.color_palette("mako", len(df[x].unique()))
plot_count = 1
plt.figure(figsize = (20,15))
for x in missing_col_cat_great10:
    plt.subplot(2,2,plot_count)
    plt.title("Average SalePrice per "+ x)
    sns.barplot(data = df[[x,'SalePrice']].fillna('No value'), x = x, y ='SalePrice', estimator = np.mean, palette = pal
    plot_count += 1
```



Based on the chart above, the columns with missing value has a significant effect to SalePrice value. The null values will be imputed as "No value".

add reason why not to drop

In [26]:
```
# for feature with greater than 10% missing values,
# it is confirmed that even the columns has high missing percentage, it is still significant. Impute the mode to missing
for x in missing_col_cat_great10:
    df[x] = df[x].fillna(df[x].mode()[0])
```

In [27]:
```
# This is the new dataset with no missing values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 80 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1460 non-null   int16  
 1   MSSubClass     1460 non-null   int16  
 2   MSZoning       1460 non-null   object 
 3   LotFrontage    1460 non-null   float32
 4   LotArea        1460 non-null   int32  
 5   Street         1460 non-null   object 
 6   Alley          1460 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   int8   
 18  OverallCond    1460 non-null   int8   
 19  YearBuilt      1460 non-null   int16  
 20  YearRemodAdd   1460 non-null   int16  
 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     1460 non-null   object 
 26  MasVnrArea     1460 non-null   float32
 27  ExterQual      1460 non-null   object 
 28  ExterCond      1460 non-null   object 
 29  Foundation     1460 non-null   object 
 30  BsmtQual       1460 non-null   object 
 31  BsmtCond       1460 non-null   object 
 32  BsmtExposure   1460 non-null   object 
```

```
 33  BsmtFinType1   1460 non-null   object 
 34  BsmtFinSF1     1460 non-null   int16  
 35  BsmtFinType2   1460 non-null   object 
 36  BsmtFinSF2     1460 non-null   int16  
 37  BsmtUnfSF      1460 non-null   int16  
 38  TotalBsmtSF    1460 non-null   int16  
 39  Heating        1460 non-null   object 
 40  HeatingQC      1460 non-null   object 
 41  CentralAir     1460 non-null   object 
 42  Electrical     1460 non-null   object 
 43  1stFlrSF       1460 non-null   int16  
 44  2ndFlrSF       1460 non-null   int16  
 45  LowQualFinSF   1460 non-null   int16  
 46  GrLivArea      1460 non-null   int16  
 47  BsmtFullBath   1460 non-null   int8   
 48  BsmtHalfBath   1460 non-null   int8   
 49  FullBath       1460 non-null   int8   
 50  HalfBath       1460 non-null   int8   
 51  BedroomAbvGr   1460 non-null   int8   
 52  KitchenAbvGr   1460 non-null   int8   
 53  KitchenQual    1460 non-null   object 
 54  TotRmsAbvGrd   1460 non-null   int8   
 55  Functional     1460 non-null   object 
 56  Fireplaces     1460 non-null   int8   
 57  FireplaceQu    1460 non-null   object 
 58  GarageType     1460 non-null   object 
 59  GarageYrBlt    1460 non-null   float32
 60  GarageFinish   1460 non-null   object 
 61  GarageCars     1460 non-null   int8   
 62  GarageArea     1460 non-null   int16  
 63  GarageQual     1460 non-null   object 
 64  GarageCond     1460 non-null   object 
 65  PavedDrive     1460 non-null   object 
 66  WoodDeckSF     1460 non-null   int16  
 67  OpenPorchSF    1460 non-null   int16  
 68  EnclosedPorch  1460 non-null   int16  
 69  3SsnPorch      1460 non-null   int16  
 70  ScreenPorch    1460 non-null   int16  
 71  PoolArea       1460 non-null   int16  
 72  PoolQC         1460 non-null   object 
 73  Fence          1460 non-null   object 
 74  MiscVal        1460 non-null   int16  
```

```
 75  MoSold         1460 non-null   int8   
 76  YrSold         1460 non-null   int16  
 77  SaleType       1460 non-null   object 
 78  SaleCondition  1460 non-null   object 
 79  SalePrice      1460 non-null   int32  
dtypes: float32(3), int16(21), int32(2), int8(12), object(42)
memory usage: 584.7+ KB
```

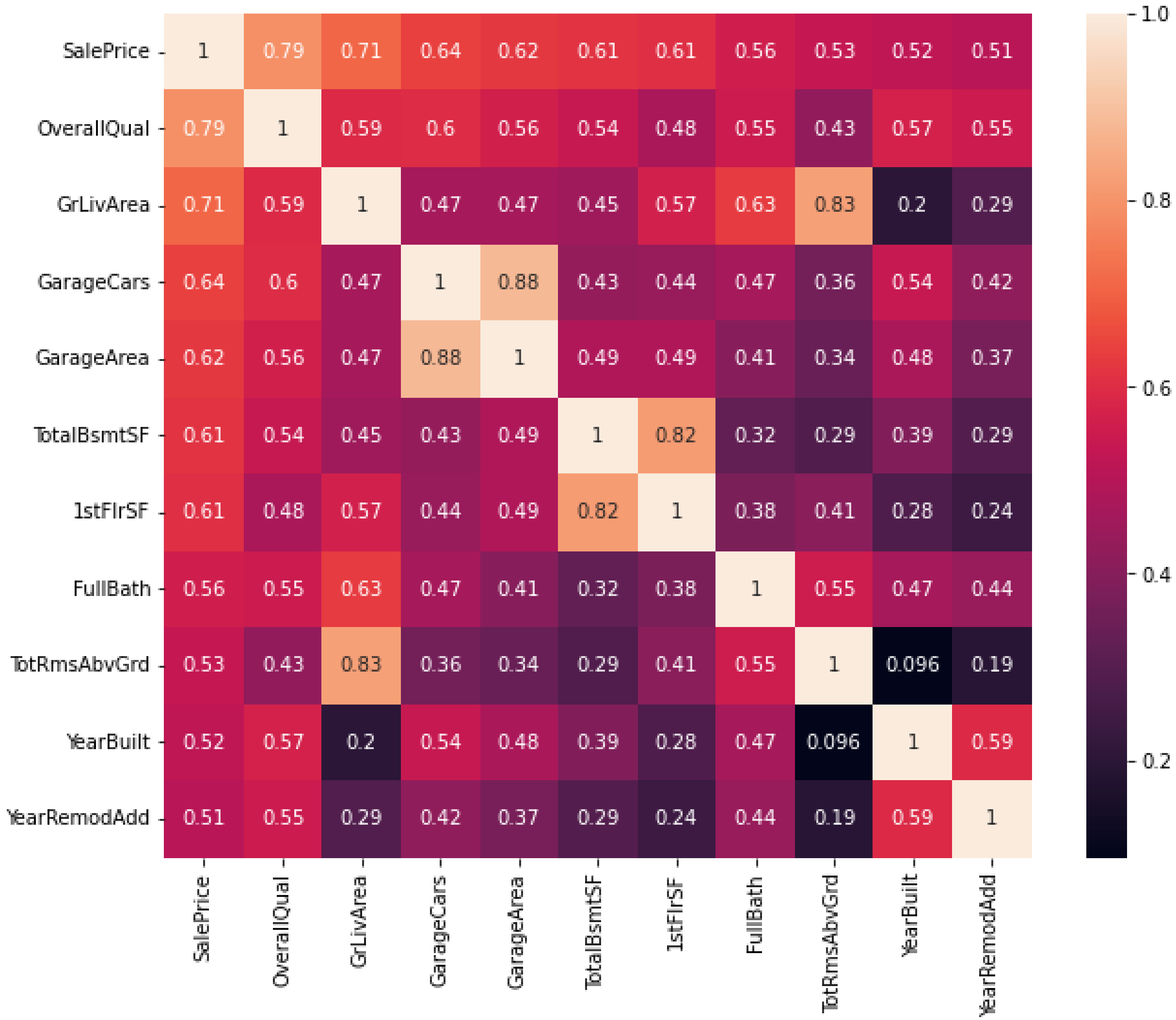
EDA

Correlation Plotting

In [28]:
```
# Checking the most correlated feature to the target variable
k = 11
cols = df.corr().nlargest(k, 'SalePrice')['SalePrice'].index
cm = np.corrcoef(df[cols].values.T)

plt.figure(figsize=[10, 8])
sns.heatmap(cm, yticklabels = cols.values, xticklabels = cols.values, annot = True)
```

Out[28]: <AxesSubplot:>



From the 10 largest correlated features, best correlated to saleprice are the following:

- OverallQual
- GrLivArea
- GarageCars
- TotalBsmtSF
- FullBath
- YearBuilt
- YearRemodAdd

The following features are skipped because they are highly correlated with other top feature and their correlation to saleprice is lower

- GarageArea
- 1stFlrSF
- TotRmsAbvGrd

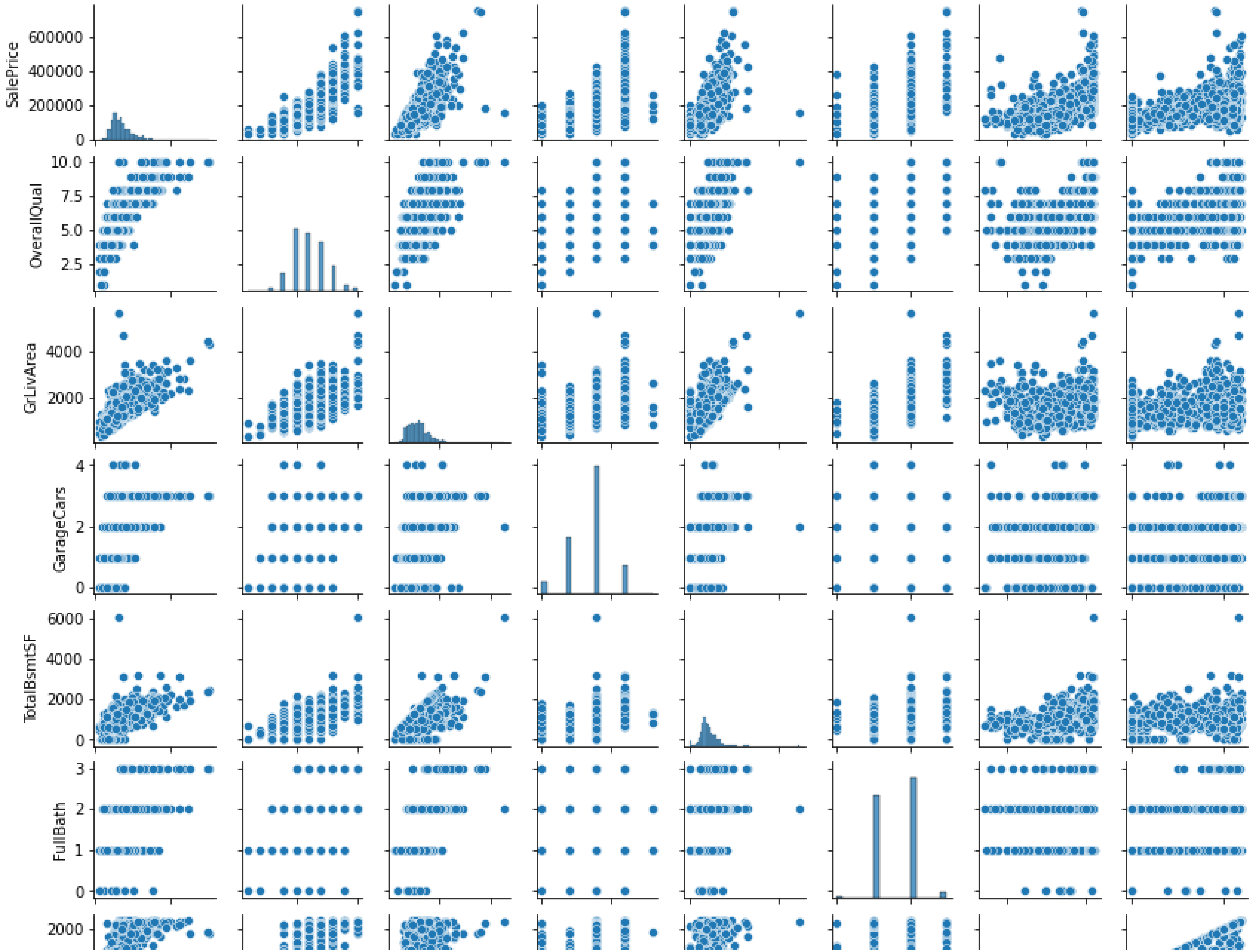
In [29]:
```
cols = """SalePrice
OverallQual
GrLivArea
GarageCars
TotalBsmtSF
FullBath
YearBuilt
YearRemodAdd"""
cols = cols.split('\n')
print(cols)
```

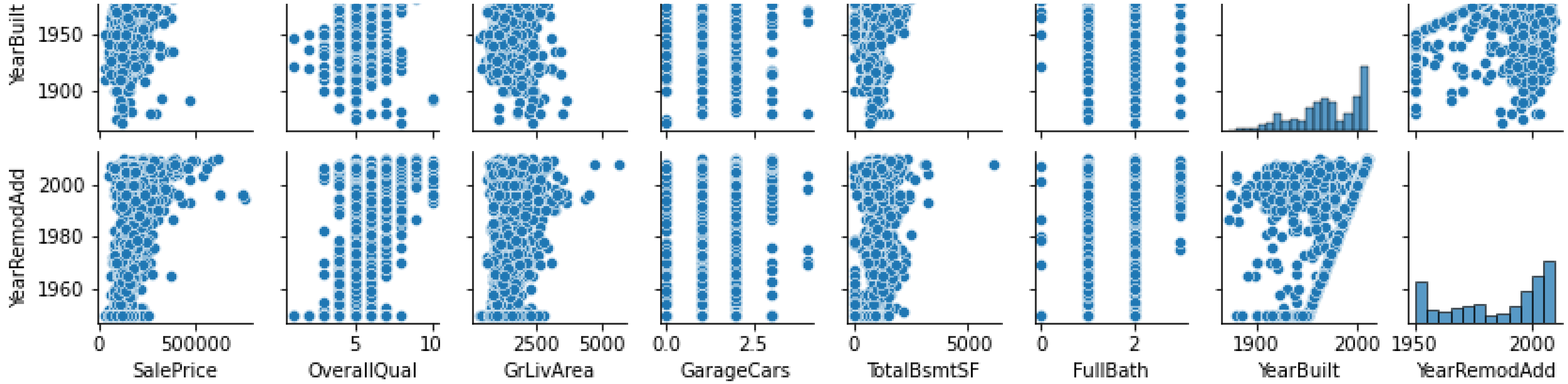
```
['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'FullBath', 'YearBuilt', 'YearRemodAdd']
```

Distribution of correlated features

In [30]:
```
sns.pairplot(df[cols], height = 1.5)
```

Out[30]: <seaborn.axisgrid.PairGrid at 0x21c546d41c0>





In [31]:
```
dist_df = pd.DataFrame(index = ['Skewness', 'Kurtosis'], data = {'Normal':[df['SalePrice'].skew(), df['SalePrice'].kurt(
                                                                  'log':[np.log1p(df['SalePrice']).skew(), np.log1p(df['SalePrice']
dist_df['% change'] = dist_df.apply(lambda x: (x['log'] / x['Normal'] - 1) * 100, axis = 1)
dist_df
```

Out[31]:

	Normal	log	% change
Skewness	1.882876	0.121347	-93.555251
Kurtosis	6.536282	0.809519	-87.614990

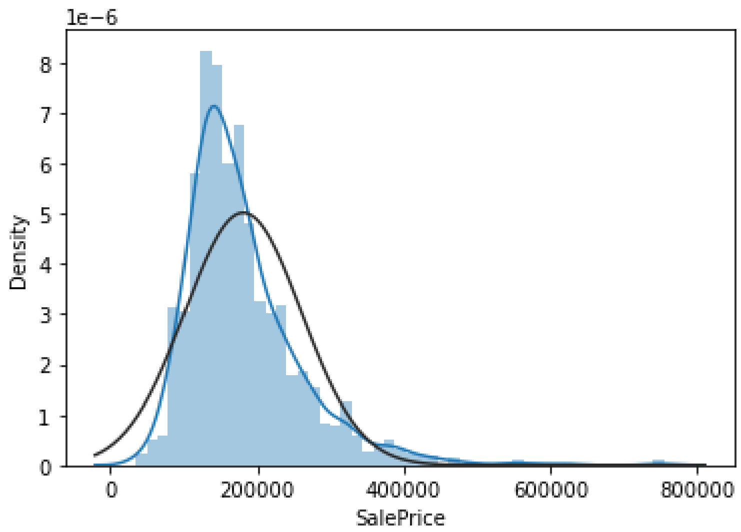
Implementing log-transformation in the dataset lowers the skewness and kurtosis.

Distribution of saleprice before and after log-transformation

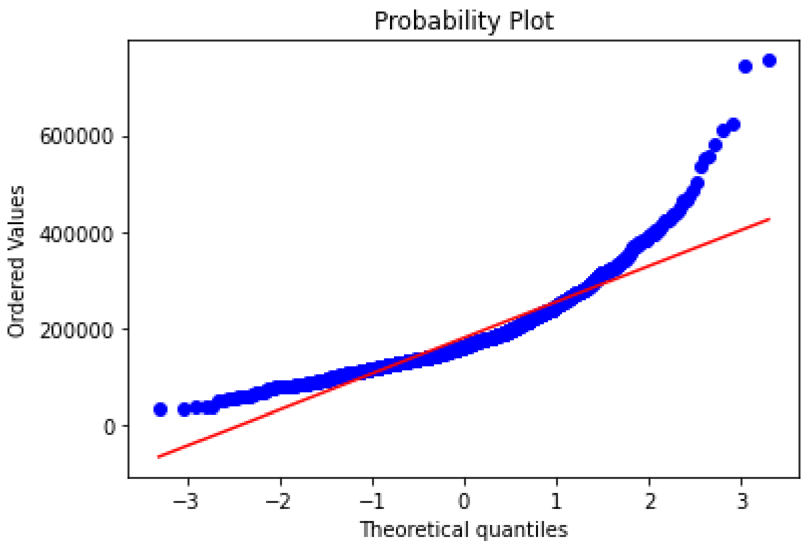
Before

In [32]:
```
sns.distplot(df['SalePrice'], fit = norm)
```

Out[32]: <AxesSubplot:xlabel='SalePrice', ylabel='Density'>



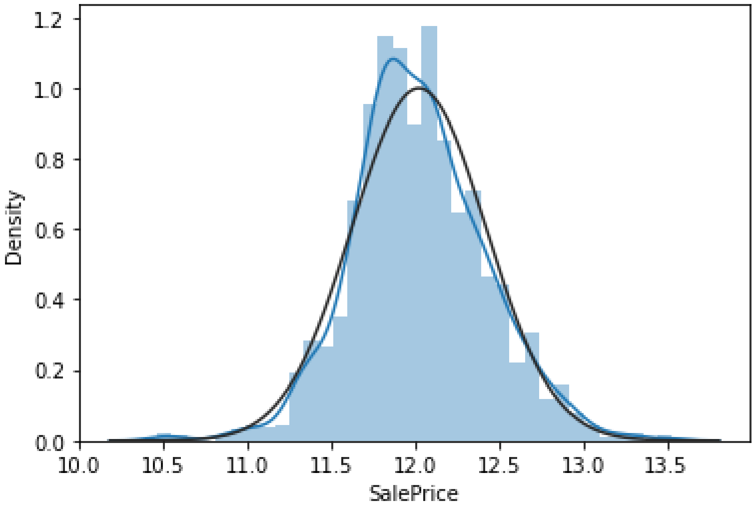
In [33]:
```
stats.probplot(df['SalePrice'], plot=plt)
plt.show()
```



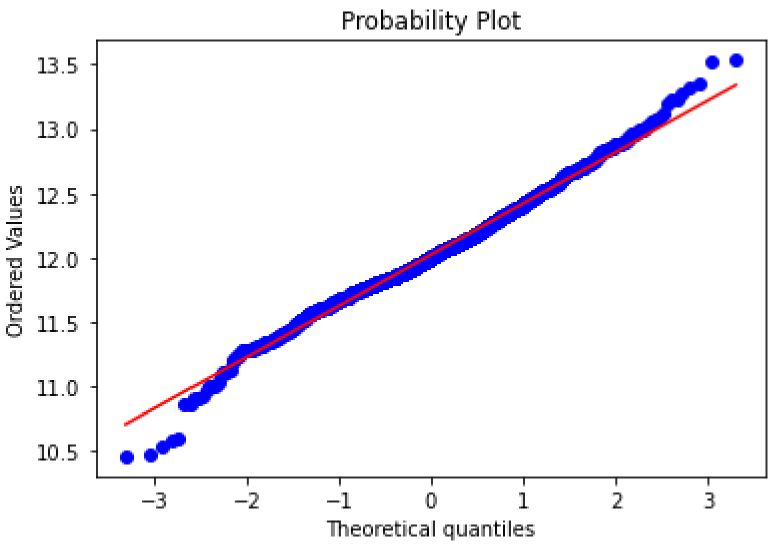
After

In [34]:
```
sns.distplot(np.log1p(df['SalePrice']), fit = norm)
```

Out[34]: <AxesSubplot:xlabel='SalePrice', ylabel='Density'>



In [35]:
```
stats.probplot(np.log1p(df['SalePrice']), plot=plt)
plt.show()
```

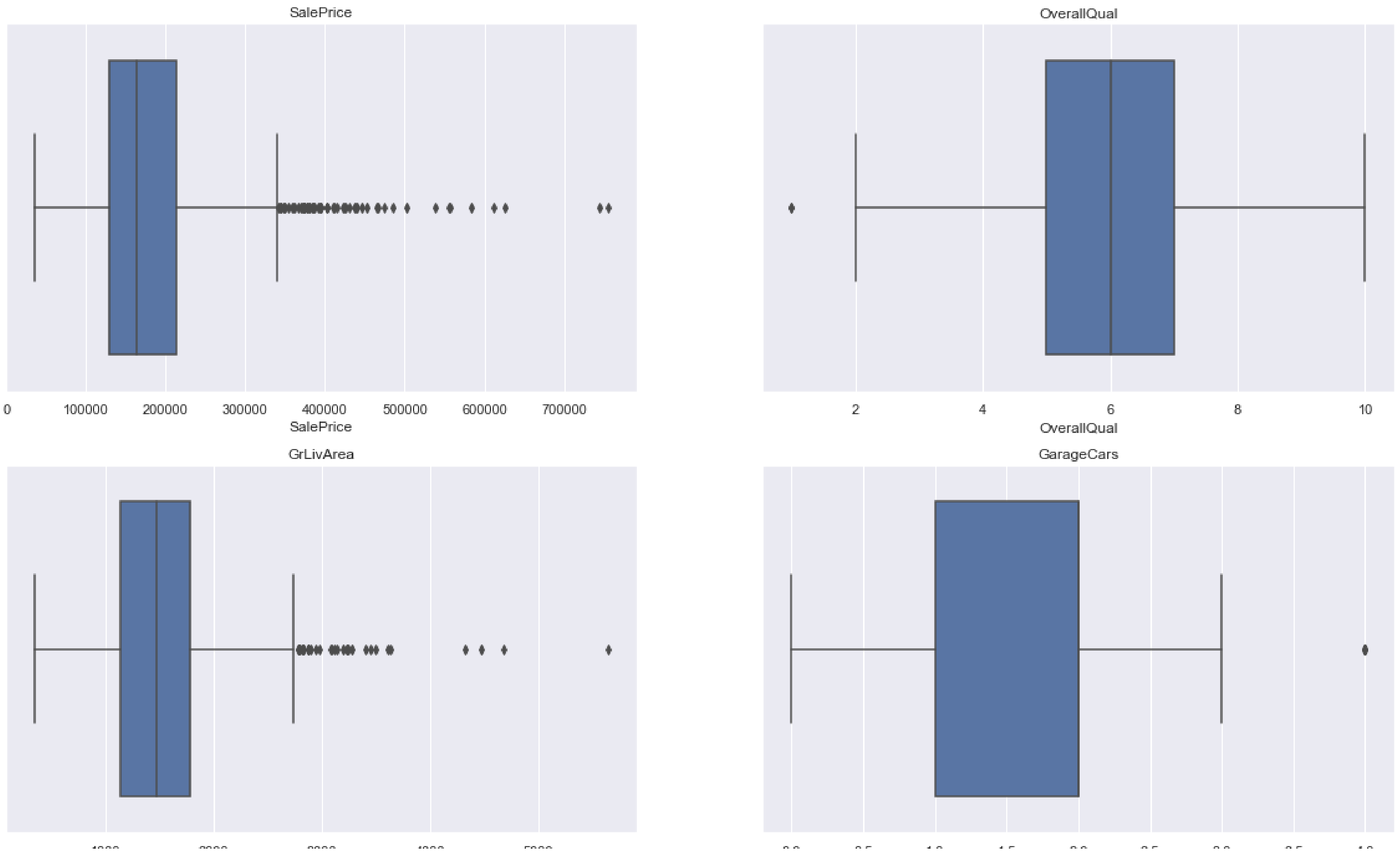


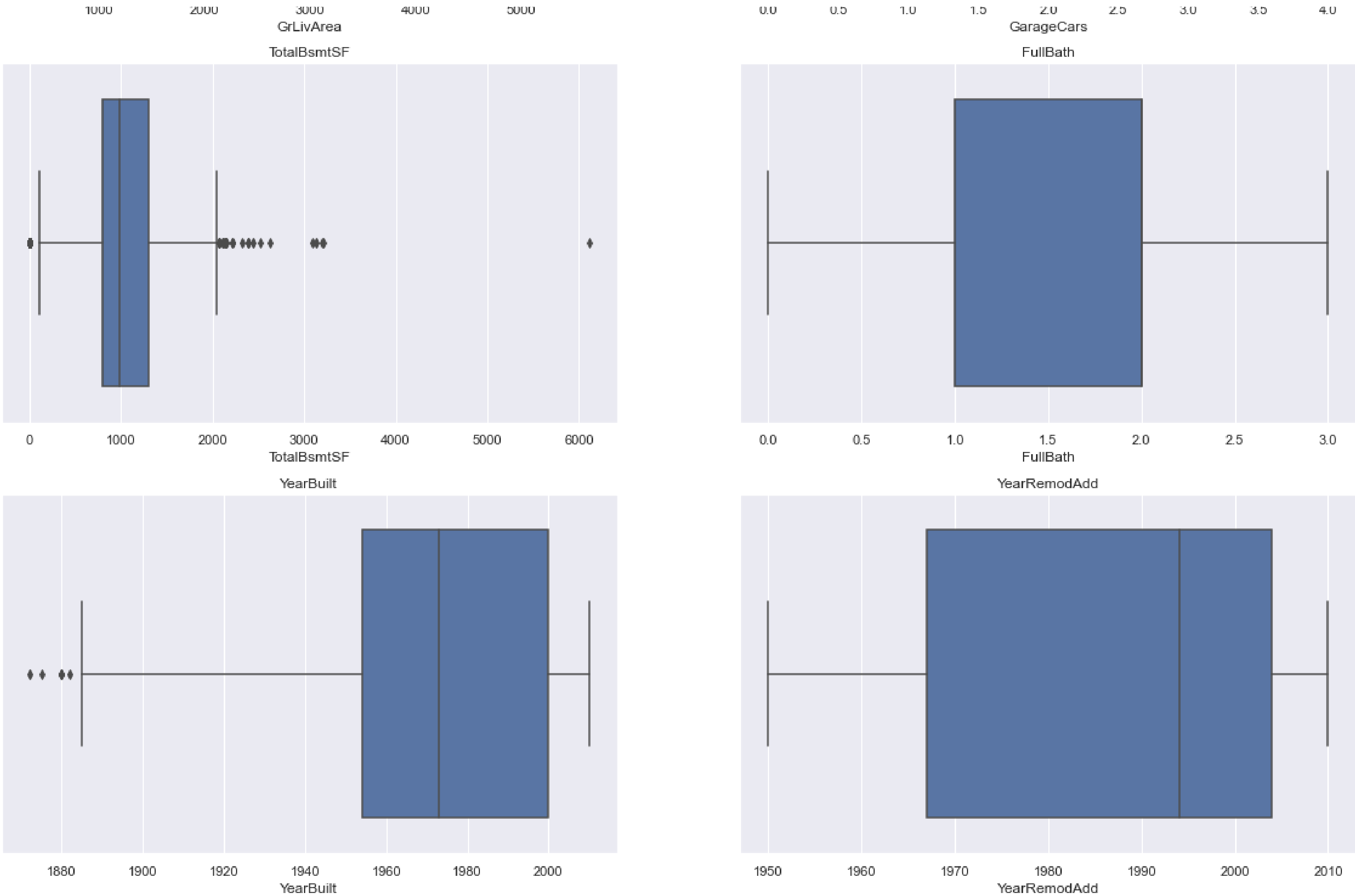
Based on the probability plot, it is more fit when log-transformed

Focusing on the before distribution, we could notice that it is right skewed. Therefore there maybe outliers in the dataset

Checking for outliers

In [36]:
```
sns.set()
plot_count = 1
plt.figure(figsize = (20,25))
for x in cols:
    plt.subplot(4,2,plot_count)
    plt.title(x)
    sns.boxplot(df[x])
    plot_count += 1
```





Based on the chart above, saleprice above 500k can be considered as an outlier

Summary:

1. Columns to remove because they are also correlated with other columns
   - GarageArea
   - 1stFlrSF
   - TotRmsAbvGrd
   - Id -> unique column
2. Log-tranformation decreases the skewness and kurtosis value of the dataset
3. The distribution of saleprice shows that above 500k is an outlier

Implementing insights to dataset

In [37]:
```
# filtering correlated columns
df_clean = df[[x for x in df.columns if x not in ['GarageArea', '1stFlrSF', 'TotRmsAbvGrd', 'Id']]]

# filterout above 500k saleprice
df_clean = df_clean[df_clean['SalePrice'] < 500000]

# log-transformation
df_clean['SalePrice'] = np.log1p(df_clean['SalePrice'])

#transform categorical columns
df_clean = pd.get_dummies(df_clean)
```

In [38]:
```
df_clean.drop('SalePrice', axis = 1)
```

Out[38]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF
0	60	65.0	8450	7	5	2003	2003	196.0	706	0	150
1	20	80.0	9600	6	8	1976	1976	0.0	978	0	284
2	60	68.0	11250	7	5	2001	2002	162.0	486	0	434
3	70	60.0	9550	7	5	1915	1970	0.0	216	0	540
4	60	84.0	14260	8	5	2000	2000	350.0	655	0	490
...	...	...	...	...	...	...	...	...	...	...	...
1455	60	62.0	7917	6	5	1999	2000	0.0	0	0	953
1456	20	85.0	13175	6	6	1978	1988	119.0	790	163	589
1457	70	66.0	9042	7	9	1941	2006	0.0	275	0	877
1458	20	68.0	9717	5	6	1950	1996	0.0	49	1029	0
1459	20	75.0	9937	5	6	1965	1965	0.0	830	290	136

1451 rows × 281 columns

Data Model

In [39]:
```
from sklearn.model_selection import train_test_split
# from sklearn.preprocessing import StandardScaler
```

Data splitting

In [40]:
```
X = df_clean.drop('SalePrice', axis = 1)
y = df_clean['SalePrice']

train_x, test_x, train_y, test_y = train_test_split(X, y, test_size=0.25, random_state=1)
```

Training and evaluation

In [41]:
```
# from sklearn.metrics import r2_score, mean_squared_error

# class Model:
#     scores = {'Model':[], 'r2_score':[], 'mse':[]}

#     def __init__(self, model, model_name):
#         self.model = model
#         self.model_name = model_name

#     def predict(self):
#         self.model.fit(train_x, train_y)
#         pred = self.model.predict(test_x)
#         r2 = r2_score(test_y, pred)
#         self.performance(pred, r2)

#     def performance(self, pred, r2):
#         mse = mean_squared_error(test_y, pred)
#         Model.scores['Model'].append(self.model_name)
#         Model.scores['r2_score'].append(r2)
#         Model.scores['mse'].append(mse)

#         print(f'r2_score: {r2}')
#         print(f'mse: {mse}\n')
```

In [42]:
```
from sklearn.metrics import r2_score, mean_squared_error

class Model:
    scores = {'Model':[], 'score-train':[], 'score-test':[],'r2_score-train':[], 'mse-train':[], 'r2_score-test':[], 'ms

    def __init__(self, model, model_name):
        self.model = model
        self.model_name = model_name

    def predict(self):
        self.model.fit(train_x, train_y)

        #training dataset pred
        pred_train = self.model.predict(train_x)
        score_train = self.model.score(train_x, train_y)
        r2_train = r2_score(train_y, pred_train)
        mse_train = mean_squared_error(train_y, pred_train)

        #testing dataset pred
        score_test = self.model.score(test_x, test_y)
        pred_test = self.model.predict(test_x)
        r2_test = r2_score(test_y, pred_test)
        mse_test = mean_squared_error(test_y, pred_test)

        self.performance(score_train, score_test, r2_train, mse_train, r2_test, mse_test)

    def performance(self, score_train, score_test, r2_train, mse_train, r2_test, mse_test):

        Model.scores['Model'].append(self.model_name)
        Model.scores['score-test'].append(score_test)
        Model.scores['score-train'].append(score_train)
        Model.scores['r2_score-test'].append(r2_test)
        Model.scores['r2_score-train'].append(r2_train)
        Model.scores['mse-test'].append(mse_test)
        Model.scores['mse-train'].append(mse_train)

        print("**Training**")
        print(f'score: {score_train}')
        print(f'r2_score: {r2_train}')
        print(f'mse: {mse_train}')
        print("===========================================================")
```

```
        print("**Test**")
        print(f'score: {score_test}')
        print(f'r2_score: {r2_test}')
        print(f'mse: {mse_test}')
```

Linear Regression

In [43]:
```
from sklearn.linear_model import LinearRegression

model = Model(LinearRegression(), 'Regression')
model.predict()
```

```
**Training**
score: 0.9475108900408803
r2_score: 0.9475108900408803
mse: 0.007476867104204432
==========================================================
**Test**
score: 0.8552351403715983
r2_score: 0.8552351403715983
mse: 0.02500968067460981
```

In [44]:
```
model.model.score(train_x, train_y)
```

Out[44]: 0.9475108900408803

In [45]:
```
model.model.score(test_x, test_y)
```

Out[45]: 0.8552351403715983

XGBRegressor

In [46]:
```
from xgboost import XGBRegressor

model = Model(XGBRegressor(), 'XGBRegressor')
model.predict()
```

```
**Training**
score: 0.9995090853908454
r2_score: 0.9995090853908454
mse: 6.992885371881096e-05
==========================================================
**Test**
score: 0.8782434390948566
r2_score: 0.8782434390948566
mse: 0.021034750533332428
```

In [47]:
```
model.model.score(train_x, train_y)
```

Out[47]:
```
0.9995090853908454
```

In [48]:
```
model.model.score(test_x, test_y)
```

Out[48]:
```
0.8782434390948566
```

In [ ]:
```

```

CatBoostRegressor

In [49]:
```
!pip install catboost
```

```
Requirement already satisfied: catboost in c:\users\dashs\anaconda3\lib\site-packages (1.0.4)
Requirement already satisfied: matplotlib in c:\users\dashs\anaconda3\lib\site-packages (from catboost) (3.4.3)
Requirement already satisfied: six in c:\users\dashs\anaconda3\lib\site-packages (from catboost) (1.16.0)
Requirement already satisfied: scipy in c:\users\dashs\anaconda3\lib\site-packages (from catboost) (1.7.1)
Requirement already satisfied: pandas>=0.24.0 in c:\users\dashs\anaconda3\lib\site-packages (from catboost) (1.3.4)
Requirement already satisfied: graphviz in c:\users\dashs\anaconda3\lib\site-packages (from catboost) (0.19.1)
Requirement already satisfied: numpy>=1.16.0 in c:\users\dashs\anaconda3\lib\site-packages (from catboost) (1.22.4)
Requirement already satisfied: plotly in c:\users\dashs\anaconda3\lib\site-packages (from catboost) (5.6.0)
Requirement already satisfied: pytz>=2017.3 in c:\users\dashs\anaconda3\lib\site-packages (from pandas>=0.24.0->catboos
t) (2021.3)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\dashs\anaconda3\lib\site-packages (from pandas>=0.24.
0->catboost) (2.8.2)
Requirement already satisfied: cycler>=0.10 in c:\users\dashs\anaconda3\lib\site-packages (from matplotlib->catboost)
(0.10.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\dashs\anaconda3\lib\site-packages (from matplotlib->catboost)
(8.4.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\dashs\anaconda3\lib\site-packages (from matplotlib->catboo
st) (1.3.1)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\dashs\anaconda3\lib\site-packages (from matplotlib->catboos
t) (3.0.4)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\dashs\anaconda3\lib\site-packages (from plotly->catboost)
(8.0.1)


WARNING: Ignoring invalid distribution -yping-extensions (c:\users\dashs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -yping-extensions (c:\users\dashs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -yping-extensions (c:\users\dashs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -yping-extensions (c:\users\dashs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -yping-extensions (c:\users\dashs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -yping-extensions (c:\users\dashs\anaconda3\lib\site-packages)
```

In [50]:
```
from catboost import CatBoostRegressor
model = Model(CatBoostRegressor(), 'CatBoostRegressor')
model.predict()
```

```
Learning rate set to 0.041492
0:	learn: 0.3679402	total: 156ms	remaining: 2m 35s
1:	learn: 0.3578436	total: 158ms	remaining: 1m 18s
2:	learn: 0.3500195	total: 161ms	remaining: 53.4s
3:	learn: 0.3409319	total: 163ms	remaining: 40.5s
4:	learn: 0.3322140	total: 165ms	remaining: 32.9s
5:	learn: 0.3244645	total: 168ms	remaining: 27.8s
6:	learn: 0.3165631	total: 170ms	remaining: 24.2s
7:	learn: 0.3095511	total: 173ms	remaining: 21.4s
8:	learn: 0.3019426	total: 176ms	remaining: 19.3s
9:	learn: 0.2949595	total: 179ms	remaining: 17.7s
10:	learn: 0.2878374	total: 181ms	remaining: 16.3s
11:	learn: 0.2808911	total: 184ms	remaining: 15.1s
12:	learn: 0.2753694	total: 186ms	remaining: 14.2s
13:	learn: 0.2699761	total: 190ms	remaining: 13.4s
14:	learn: 0.2644937	total: 192ms	remaining: 12.6s
15:	learn: 0.2592536	total: 194ms	remaining: 12s
16:	learn: 0.2540043	total: 197ms	remaining: 11.4s
17:	learn: 0.2489526	total: 199ms	remaining: 10.9s
```

DecisionTreeRegressor

In [51]:
```
from sklearn.tree import DecisionTreeRegressor

model = Model(DecisionTreeRegressor(), 'DecisionTreeRegressor')
model.predict()
```

```
**Training**
score: 1.0
r2_score: 1.0
mse: 8.70067174876116e-33
==========================================================
**Test**
score: 0.6948317099125783
r2_score: 0.6948317099125783
mse: 0.05272109202947579
```

KNeighborsRegressor

In [52]:
```
from sklearn.neighbors import KNeighborsRegressor

model = Model(KNeighborsRegressor(), 'KNeighborsRegressor')
model.predict()
```

```
**Training**
score: 0.7674984426150838
r2_score: 0.7674984426150838
mse: 0.03311893166871167
==========================================================
**Test**
score: 0.6670015779993883
r2_score: 0.6670015779993883
mse: 0.0575290455208671
```

LightGBM

In [53]:
```
!pip install lightgbm
```

```
Requirement already satisfied: lightgbm in c:\users\dashs\anaconda3\lib\site-packages (3.3.2)

WARNING: Ignoring invalid distribution -yping-extensions (c:\users\dashs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -yping-extensions (c:\users\dashs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -yping-extensions (c:\users\dashs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -yping-extensions (c:\users\dashs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -yping-extensions (c:\users\dashs\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -yping-extensions (c:\users\dashs\anaconda3\lib\site-packages)



Requirement already satisfied: wheel in c:\users\dashs\anaconda3\lib\site-packages (from lightgbm) (0.37.0)
Requirement already satisfied: numpy in c:\users\dashs\anaconda3\lib\site-packages (from lightgbm) (1.22.4)
Requirement already satisfied: scipy in c:\users\dashs\anaconda3\lib\site-packages (from lightgbm) (1.7.1)
Requirement already satisfied: scikit-learn!=0.22.0 in c:\users\dashs\anaconda3\lib\site-packages (from lightgbm) (1.0.
2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\dashs\anaconda3\lib\site-packages (from scikit-learn!=
0.22.0->lightgbm) (2.2.0)
Requirement already satisfied: joblib>=0.11 in c:\users\dashs\anaconda3\lib\site-packages (from scikit-learn!=0.22.0->l
ightgbm) (1.1.0)
```

In [54]:
```
from lightgbm import LGBMRegressor

model = Model(LGBMRegressor(), 'LGBMRegressor')
model.predict()
```

```
**Training**
score: 0.9884537132583779
r2_score: 0.9884537132583779
mse: 0.0016447230974459702
==========================================================
**Test**
score: 0.8987217370262491
r2_score: 0.8987217370262491
mse: 0.01749690513812874
```

In [ ]:

support vector machine regression

In [55]:
```
from sklearn import svm

model = Model(svm.SVR(), 'svm')
model.predict()
```

```
**Training**
score: 0.6875665040707979
r2_score: 0.6875665040707979
mse: 0.044504921683450445
==========================================================
**Test**
score: 0.7145062318398188
r2_score: 0.7145062318398188
mse: 0.049322107551551024
```

Ridge

In [56]:
```
from sklearn.linear_model import Ridge

model = Model(Ridge(alpha=1.0), 'Ridge')
model.predict()
```

```
**Training**
score: 0.9370060075568896
r2_score: 0.9370060075568896
mse: 0.008973246264362722
==========================================================
**Test**
score: 0.8833370754919374
r2_score: 0.8833370754919374
mse: 0.020154770266777675
```

In [ ]:
```
```

In [66]:
```
from sklearn.linear_model import Lasso

model = Model(Lasso(alpha=0), 'Lasso')
model.predict()
```

```
**Training**
score: 0.9999999982292352
r2_score: 0.9999999982292352
mse: 10.561815911670994
==========================================================
**Test**
score: 0.9999999990709155
r2_score: 0.9999999990709155
mse: 7.126381319063706
```

In [ ]:

In [67]:
```
from sklearn.linear_model import ElasticNet

model = Model(ElasticNet(), 'ElasticNet')
model.predict()
```

```
**Training**
score: 0.9999999982292352
r2_score: 0.9999999982292352
mse: 10.561815914611842
==========================================================
**Test**
score: 0.9999999990709151
r2_score: 0.9999999990709151
mse: 7.126384272257651
```

In [ ]:

In [68]:
```
from sklearn.svm import SVR
from sklearn.ensemble import BaggingRegressor

model = Model(BaggingRegressor(base_estimator=SVR(),n_estimators=10), 'BaggingRegressor')
model.predict()
```

```
**Training**
score: -0.04242026399727927
r2_score: -0.04242026399727927
mse: 6217568181.505436
==========================================================
**Test**
score: -0.022338150790116806
r2_score: -0.022338150790116806
mse: 7841667110.612656
```

In [69]:
```
from sklearn.ensemble import GradientBoostingRegressor

model = Model(GradientBoostingRegressor(), 'GradientBoostingRegressor')
model.predict()
```

```
**Training**
score: 0.9999339327871095
r2_score: 0.9999339327871095
mse: 394061.2197362208
==========================================================
**Test**
score: 0.9983118637773125
r2_score: 0.9983118637773125
mse: 12948555.510181868
```

Model Performance Summary

In [57]:
```
performance_df = pd.DataFrame(Model.scores)
performance_df.sort_values(by='r2_score-test', ascending=False, inplace=True)
performance_df.reset_index(drop = True, inplace = True)
performance_df
```

Out[57]:

	Model	score-train	score-test	r2_score-train	mse-train	r2_score-test	mse-test
0	CatBoostRegressor	0.993033	0.912745	0.993033	9.924118e-04	0.912745	0.015074
1	LGBMRegressor	0.988454	0.898722	0.988454	1.644723e-03	0.898722	0.017497
2	Ridge	0.937006	0.883337	0.937006	8.973246e-03	0.883337	0.020155
3	XGBRegressor	0.999509	0.878243	0.999509	6.992885e-05	0.878243	0.021035
4	Regression	0.947511	0.855235	0.947511	7.476867e-03	0.855235	0.025010
5	svm	0.687567	0.714506	0.687567	4.450492e-02	0.714506	0.049322
6	DecisionTreeRegressor	1.000000	0.694832	1.000000	8.700672e-33	0.694832	0.052721
7	KNeighborsRegressor	0.767498	0.667002	0.767498	3.311893e-02	0.667002	0.057529

From the model performance summary, CatBoostRegressor has the best performance with r2_score of 91.27% and mse of 0.015074

Using AutoML

This part will be one time running, as this only need to search for the most optimized parameters

In [55]:
```
# import tpot

# from tpot import TPOTRegressor
# from sklearn.model_selection import RepeatedStratifiedKFold

# cv = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=1)
# tpot = TPOTRegressor(generations=5, population_size=50, verbosity=2, random_state = 1, n_jobs = -1)

# tpot.fit(train_x, train_y)
```

In [56]:
```
# print(tpot.score(test_x, test_y))
# tpot.export('HousePricePred - AutoML.py')
```

In [57]:
```
# tpot = TPOTRegressor(generations=15, population_size=50, verbosity=2, random_state = 1, n_jobs = -1)

# tpot.fit(train_x, train_y)
# print(tpot.score(test_x, test_y))
# tpot.export('HousePricePred - AutoML gen15.py')
```

10 generation AutoML

In [59]:
```
!pip install tpot
```

```
Collecting tpot
  Downloading TPOT-0.11.7-py3-none-any.whl (87 kB)
Requirement already satisfied: scikit-learn>=0.22.0 in c:\users\administrator\anaconda3\lib\site-packages (from tpot)
(0.24.2)
Collecting deap>=1.2
  Downloading deap-1.3.1-cp39-cp39-win_amd64.whl (108 kB)
Requirement already satisfied: scipy>=1.3.1 in c:\users\administrator\anaconda3\lib\site-packages (from tpot) (1.7.1)
Requirement already satisfied: tqdm>=4.36.1 in c:\users\administrator\anaconda3\lib\site-packages (from tpot) (4.62.
3)
Requirement already satisfied: numpy>=1.16.3 in c:\users\administrator\anaconda3\lib\site-packages (from tpot) (1.20.
3)
Requirement already satisfied: pandas>=0.24.2 in c:\users\administrator\anaconda3\lib\site-packages (from tpot) (1.3.
4)
Collecting stopit>=1.1.1
  Downloading stopit-1.1.2.tar.gz (18 kB)
Collecting update-checker>=0.16
  Downloading update_checker-0.18.0-py3-none-any.whl (7.0 kB)
Requirement already satisfied: joblib>=0.13.2 in c:\users\administrator\anaconda3\lib\site-packages (from tpot) (1.1.
0)
Requirement already satisfied: xgboost>=1.1.0 in c:\users\administrator\anaconda3\lib\site-packages (from tpot) (1.5.
2)
Requirement already satisfied: pytz>=2017.3 in c:\users\administrator\anaconda3\lib\site-packages (from pandas>=0.24.
2->tpot) (2021.3)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\administrator\anaconda3\lib\site-packages (from pan
das>=0.24.2->tpot) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\administrator\anaconda3\lib\site-packages (from python-dateutil>=
2.7.3->pandas>=0.24.2->tpot) (1.16.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\administrator\anaconda3\lib\site-packages (from sciki
t-learn>=0.22.0->tpot) (2.2.0)
Requirement already satisfied: colorama in c:\users\administrator\anaconda3\lib\site-packages (from tqdm>=4.36.1->tpo
t) (0.4.4)
Requirement already satisfied: requests>=2.3.0 in c:\users\administrator\anaconda3\lib\site-packages (from update-che
cker>=0.16->tpot) (2.26.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\administrator\anaconda3\lib\site-packages (from requ
ests>=2.3.0->update-checker>=0.16->tpot) (1.26.7)
Requirement already satisfied: idna<4,>=2.5 in c:\users\administrator\anaconda3\lib\site-packages (from requests>=2.
3.0->update-checker>=0.16->tpot) (3.2)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\administrator\anaconda3\lib\site-packages (from request
s>=2.3.0->update-checker>=0.16->tpot) (2021.10.8)
```

```
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\administrator\anaconda3\lib\site-packages (from
requests>=2.3.0->update-checker>=0.16->tpot) (2.0.4)
Building wheels for collected packages: stopit
  Building wheel for stopit (setup.py): started
  Building wheel for stopit (setup.py): finished with status 'done'
  Created wheel for stopit: filename=stopit-1.1.2-py3-none-any.whl size=11952 sha256=86ec0d6d69329eea7629d4a696b967c9
14c619e3fd19bcdfca19c7c4be3b9890
  Stored in directory: c:\users\administrator\appdata\local\pip\cache\wheels\48\8c\93\3afb1916772591fe6bcc25cdf8b1c5b
dc362f0ec8e2f0fd413
Successfully built stopit
Installing collected packages: update-checker, stopit, deap, tpot
Successfully installed deap-1.3.1 stopit-1.1.2 tpot-0.11.7 update-checker-0.18.0
```

In [62]:
```
scores = {'Model':[], 'score-train':[], 'score-test':[],'r2_score-train':[], 'mse-train':[], 'r2_score-test':[], 'mse-te

import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LassoLarsCV
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline, make_union
from tpot.builtins import StackingEstimator
from tpot.export_utils import set_param_recursive

# Average CV score on the training set was: -0.013745162397721333
exported_pipeline = make_pipeline(
    StackingEstimator(estimator=LassoLarsCV(normalize=True)),
    RandomForestRegressor(bootstrap=True, max_features=0.8, min_samples_leaf=3, min_samples_split=13, n_estimators=100)
)
# Fix random state for all the steps in exported pipeline
set_param_recursive(exported_pipeline.steps, 'random_state', 1)

exported_pipeline.fit(train_x, train_y)
results = exported_pipeline.predict(test_x)

#training dataset pred
results = exported_pipeline.predict(train_x)
score_train = exported_pipeline.score(train_x, train_y)
r2_train = r2_score(train_y, results)
mse_train = mean_squared_error(train_y, results)

#testing dataset pred
results = exported_pipeline.predict(test_x)
score_test = exported_pipeline.score(test_x, test_y)
r2_test = r2_score(test_y, results)
mse_test = mean_squared_error(test_y, results)


scores['Model'].append('AutoML - 10 generation')
scores['score-train'].append(score_train)
scores['r2_score-train'].append(r2_train)
scores['mse-train'].append(mse_train)
scores['score-test'].append(score_test)
scores['r2_score-test'].append(r2_test)
```

```
scores['mse-test'].append(mse_test)

print("**Training**")
print(f'score: {score_train}')
print(f'r2_score: {r2_train}')
print(f'mse: {mse_train}')
print("==========================================================")
print("**Test**")
print(f'score: {score_test}')
print(f'r2_score: {r2_test}')
print(f'mse: {mse_test}')
```

```
**Training**
score: 0.9672551016600515
r2_score: 0.9672551016600515
mse: 0.004664381876910487
==========================================================
**Test**
score: 0.9171151034654166
r2_score: 0.9171151034654166
mse: 0.014319253998511885
```

15 generation AutoML

In [63]:
```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LassoLarsCV, RidgeCV
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline, make_union
from tpot.builtins import StackingEstimator
from tpot.export_utils import set_param_recursive

# NOTE: Make sure that the outcome column is labeled 'target' in the data file
# tpot_data = pd.read_csv('PATH/TO/DATA/FILE', sep='COLUMN_SEPARATOR', dtype=np.float64)
# features = tpot_data.drop('target', axis=1)
# training_features, testing_features, training_target, testing_target = \
#             train_test_split(features, tpot_data['target'], random_state=1)

# Average CV score on the training set was: -0.013366345704249877
exported_pipeline = make_pipeline(
    StackingEstimator(estimator=LassoLarsCV(normalize=True)),
    StackingEstimator(estimator=RandomForestRegressor(bootstrap=True, max_features=0.6500000000000001, min_samples_leaf=
    StackingEstimator(estimator=RidgeCV()),
    RandomForestRegressor(bootstrap=False, max_features=0.25, min_samples_leaf=4, min_samples_split=13, n_estimators=100
)
# Fix random state for all the steps in exported pipeline
set_param_recursive(exported_pipeline.steps, 'random_state', 1)

exported_pipeline.fit(train_x, train_y)
results = exported_pipeline.predict(test_x)

#training dataset pred
results = exported_pipeline.predict(train_x)
score_train = exported_pipeline.score(train_x, train_y)
r2_train = r2_score(train_y, results)
mse_train = mean_squared_error(train_y, results)

#testing dataset pred
results = exported_pipeline.predict(test_x)
score_test = exported_pipeline.score(test_x, test_y)
r2_test = r2_score(test_y, results)
mse_test = mean_squared_error(test_y, results)
```

```
scores['Model'].append('AutoML - 15 generation')
scores['score-train'].append(score_train)
scores['r2_score-train'].append(r2_train)
scores['mse-train'].append(mse_train)
scores['score-test'].append(score_test)
scores['r2_score-test'].append(r2_test)
scores['mse-test'].append(mse_test)

print("**Training**")
print(f'score: {score_train}')
print(f'r2_score: {r2_train}')
print(f'mse: {mse_train}')
print("==========================================================")
print("**Test**")
print(f'score: {score_test}')
print(f'r2_score: {r2_test}')
print(f'mse: {mse_test}')
```

```
**Training**
score: 0.9868423950782546
r2_score: 0.9868423950782546
mse: 0.001874249029677525
==========================================================
**Test**
score: 0.918976721072821
r2_score: 0.918976721072821
mse: 0.013997639609364369
```

In [64]:
```
performance_df = pd.concat([performance_df,pd.DataFrame(scores)]).sort_values(by = ['r2_score-test'], ascending = False)
performance_df.reset_index(drop = True, inplace = True)
performance_df['Dimension-Reduction'] = "None"
performance_df
```

Out[64]:

	Model	score-train	score-test	r2_score-train	mse-train	r2_score-test	mse-test	Dimension-Reduction
0	AutoML - 15 generation	0.986842	0.918977	0.986842	1.874249e-03	0.918977	0.013998	None
1	AutoML - 10 generation	0.967255	0.917115	0.967255	4.664382e-03	0.917115	0.014319	None
2	LGBMRegressor	0.988454	0.898722	0.988454	1.644723e-03	0.898722	0.017497	None
3	Ridge	0.937006	0.883337	0.937006	8.973246e-03	0.883337	0.020155	None
4	XGBRegressor	0.999509	0.878243	0.999509	6.992885e-05	0.878243	0.021035	None
5	Regression	0.947511	0.855235	0.947511	7.476867e-03	0.855235	0.025010	None
6	svm	0.687567	0.714506	0.687567	4.450492e-02	0.714506	0.049322	None
7	DecisionTreeRegressor	1.000000	0.682451	1.000000	8.700672e-33	0.682451	0.054860	None
8	KNeighborsRegressor	0.767498	0.667002	0.767498	3.311893e-02	0.667002	0.057529	None

After using autoML, it has increase the r2_score by 1% and based on the trend, the higher the generation, the higher the performance

With PCA

In [58]:
```
from sklearn.decomposition import PCA
```

In [59]:
```
# converted the dataframe categorical values to numerical
df_pca = pd.get_dummies(df)
df_pca.drop(columns = ['Id'], inplace = True)
df_pca
```

Out[59]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF
0	60	65.0	8450	7	5	2003	2003	196.0	706	0	150
1	20	80.0	9600	6	8	1976	1976	0.0	978	0	284
2	60	68.0	11250	7	5	2001	2002	162.0	486	0	434
3	70	60.0	9550	7	5	1915	1970	0.0	216	0	540
4	60	84.0	14260	8	5	2000	2000	350.0	655	0	490
...	...	...	...	...	...	...	...	...	...	...	...
1455	60	62.0	7917	6	5	1999	2000	0.0	0	0	953
1456	20	85.0	13175	6	6	1978	1988	119.0	790	163	589
1457	70	66.0	9042	7	9	1941	2006	0.0	275	0	877
1458	20	68.0	9717	5	6	1950	1996	0.0	49	1029	0
1459	20	75.0	9937	5	6	1965	1965	0.0	830	290	136

1460 rows × 285 columns

In [60]:
```
pca = PCA(n_components = 2)
x_new = pca.fit_transform(df_pca)
```

Explained variance ratio

In [61]:
```
pca.explained_variance_ratio_
```

Out[61]:
```
array([0.98536701, 0.01444069])
```

PC1 and PC2 can explain 99.97% of dataset

In [62]:
```
df_pca = pd.DataFrame(data = x_new, columns = ['PC1', 'PC2'])
```

In [63]:
```
from sklearn.model_selection import train_test_split

X = df_pca
y = df['SalePrice']

train_x, test_x, train_y, test_y = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [64]:
```
p_value = df_pca.join(df['SalePrice'])
```

In [65]:
```
import statsmodels.api as sm

all_columns = "+".join(df_pca.columns)

my_formula = "SalePrice~"+all_columns

lm = sm.OLS.from_formula(formula = my_formula, data = p_value)
result = lm.fit()

result.summary()
```

Out[65]:

OLS Regression Results

Dep. Variable:	SalePrice	**R-squared:**	1.000
Model:	OLS	**Adj. R-squared:**	1.000
Method:	Least Squares	**F-statistic:**	4.655e+11
Date:	Mon, 25 Jul 2022	**Prob (F-statistic):**	0.00
Time:	21:07:21	**Log-Likelihood:**	-3742.9
No. Observations:	1460	**AIC:**	7492.
Df Residuals:	1457	**BIC:**	7508.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.809e+05	0.082	2.2e+06	0.000	1.81e+05	1.81e+05
PC1	0.9994	1.04e-06	9.65e+05	0.000	0.999	0.999
PC2	-0.0337	8.56e-06	-3936.647	0.000	-0.034	-0.034

Omnibus:	1661.749	**Durbin-Watson:**	1.962
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	411892.815
Skew:	-5.279	**Prob(JB):**	0.00
Kurtosis:	84.605	**Cond. No.**	7.95e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.95e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [71]:
```
from sklearn.metrics import r2_score, mean_squared_error

class Model:
    scores = {'Model':[], 'score-train':[], 'score-test':[],'r2_score-train':[], 'mse-train':[], 'r2_score-test':[], 'ms

    def __init__(self, model, model_name):
        self.model = model
        self.model_name = model_name

    def predict(self):
        self.model.fit(train_x, train_y)

        #training dataset pred
        pred_train = self.model.predict(train_x)
        score_train = self.model.score(train_x, train_y)
        r2_train = r2_score(train_y, pred_train)
        mse_train = mean_squared_error(train_y, pred_train)

        #testing dataset pred
        score_test = self.model.score(test_x, test_y)
        pred_test = self.model.predict(test_x)
        r2_test = r2_score(test_y, pred_test)
        mse_test = mean_squared_error(test_y, pred_test)

        self.performance(score_train, score_test, r2_train, mse_train, r2_test, mse_test)

    def performance(self, score_train, score_test, r2_train, mse_train, r2_test, mse_test):

        Model.scores['Model'].append(self.model_name)
        Model.scores['score-test'].append(score_test)
        Model.scores['score-train'].append(score_train)
        Model.scores['r2_score-test'].append(r2_test)
        Model.scores['r2_score-train'].append(r2_train)
        Model.scores['mse-test'].append(mse_test)
        Model.scores['mse-train'].append(mse_train)

        print("**Training**")
        print(f'score: {score_train}')
        print(f'r2_score: {r2_train}')
        print(f'mse: {mse_train}')
        print("==============================================================")
```

```
        print("**Test**")
        print(f'score: {score_test}')
        print(f'r2_score: {r2_test}')
        print(f'mse: {mse_test}')
```

LinearRegression

In [72]:
```
from sklearn.linear_model import LinearRegression

model = Model(LinearRegression(), 'Regression')
model.predict()
```

```
**Training**
score: 0.9999999982292352
r2_score: 0.9999999982292352
mse: 10.561815911617607
==========================================================
**Test**
score: 0.9999999990709155
r2_score: 0.9999999990709155
mse: 7.126381247579847
```

XGBRegressor

In [73]:
```
from xgboost import XGBRegressor

model = Model(XGBRegressor(), 'XGBRegressor')
model.predict()
```

```
**Training**
score: 0.9999934633599985
r2_score: 0.9999934633599985
mse: 38988.11860368023
==========================================================
**Test**
score: 0.9995593541068126
r2_score: 0.9995593541068126
mse: 3379897.7426048173
```

CatBoostRegressor

In [74]:
```
from catboost import CatBoostRegressor
model = Model(CatBoostRegressor(), 'CatBoostRegressor')
model.predict()
```

```
Learning rate set to 0.04196
0:      learn: 74477.4685915    total: 3.27ms   remaining: 3.26s
1:      learn: 71903.6446028    total: 5.46ms   remaining: 2.73s
2:      learn: 69333.5010261    total: 7.66ms   remaining: 2.55s
3:      learn: 66987.3751442    total: 11ms     remaining: 2.75s
4:      learn: 64777.5284029    total: 13.2ms   remaining: 2.63s
5:      learn: 62621.0963121    total: 15.7ms   remaining: 2.6s
6:      learn: 60542.0822440    total: 17.8ms   remaining: 2.52s
7:      learn: 58549.4284258    total: 21.3ms   remaining: 2.64s
8:      learn: 56432.7405090    total: 23.8ms   remaining: 2.62s
9:      learn: 54502.0652475    total: 26.1ms   remaining: 2.58s
10:     learn: 52683.4362920    total: 28.5ms   remaining: 2.56s
11:     learn: 51000.3242281    total: 30.8ms   remaining: 2.54s
12:     learn: 49295.2861728    total: 34.1ms   remaining: 2.59s
13:     learn: 47768.2677398    total: 36.2ms   remaining: 2.55s
14:     learn: 46243.9846380    total: 38.3ms   remaining: 2.52s
15:     learn: 44642.2277410    total: 40.4ms   remaining: 2.49s
16:     learn: 43202.1239159    total: 43ms     remaining: 2.48s
17:     learn: 41777.8037193    total: 45.7ms   remaining: 2.49s
```

DecisionTreeRegressor

In [75]:
```
from sklearn.tree import DecisionTreeRegressor

model = Model(DecisionTreeRegressor(), 'DecisionTreeRegressor')
model.predict()
```

```
**Training**
score: 1.0
r2_score: 1.0
mse: 0.0
==========================================================
**Test**
score: 0.9995276557893082
r2_score: 0.9995276557893082
mse: 3623034.178082192
```

KNeighborsRegressor

In [76]:
```
from sklearn.neighbors import KNeighborsRegressor

model = Model(KNeighborsRegressor(), 'KNeighborsRegressor')
model.predict()

model = Model(LGBMRegressor(), 'LGBMRegressor')
```

```
**Training**
score: 0.9960334182281466
r2_score: 0.9960334182281466
mse: 23658876.81369863
==========================================================
**Test**
score: 0.9890078430990562
r2_score: 0.9890078430990562
mse: 84313429.15082192
```

LGBMRegressor

In [77]:
```
from lightgbm import LGBMRegressor

model = Model(LGBMRegressor(), 'LGBMRegressor')
model.predict()
```

```
**Training**
score: 0.9830626692284709
r2_score: 0.9830626692284709
mse: 101023562.68562302
==========================================================
**Test**
score: 0.9649516132635713
r2_score: 0.9649516132635713
mse: 268832559.30405885
```

SVM

In [78]:
```
from sklearn import svm

model = Model(svm.SVR(), 'svm')
model.predict()
```

```
**Training**
score: -0.040762344513l8905
r2_score: -0.04076234451318905
mse: 6207679437.216977
==========================================================
**Test**
score: -0.021242611886103324
r2_score: -0.021242611886103324
mse: 7833263969.846209
```

Ridge

In [79]:
```
from sklearn.linear_model import Ridge

model = Model(Ridge(alpha=1.0), 'Ridge')
model.predict()
```

```
**Training**
score: 0.9999999982292352
r2_score: 0.9999999982292352
mse: 10.561815911616721
==========================================================
**Test**
score: 0.9999999990709155
r2_score: 0.9999999990709155
mse: 7.126381247824536
```

Summary

In [82]:
```
performance_df_pca = pd.DataFrame(Model.scores)
performance_df_pca.sort_values(by='r2_score-test', ascending=False, inplace=True)

performance_df_pca['Dimension-Reduction'] = "pca"
```

In [83]:

```
performance_final = pd.concat([performance_df,performance_df_pca]).sort_values(by = ['mse-test','r2_score-test'], ascendi
performance_final
```

Out[83]:

	Model	score-train	score-test	r2_score-train	mse-train	r2_score-test	mse-test	Dimension-Reduction
0	AutoML - 15 generation	0.986842	0.918977	0.986842	1.874249e-03	0.918977	1.399764e-02	None
1	AutoML - 10 generation	0.967255	0.917115	0.967255	4.664382e-03	0.917115	1.431925e-02	None
2	LGBMRegressor	0.988454	0.898722	0.988454	1.644723e-03	0.898722	1.749691e-02	None
3	Ridge	0.937006	0.883337	0.937006	8.973246e-03	0.883337	2.015477e-02	None
4	XGBRegressor	0.999509	0.878243	0.999509	6.992885e-05	0.878243	2.103475e-02	None
5	Regression	0.947511	0.855235	0.947511	7.476867e-03	0.855235	2.500968e-02	None
6	svm	0.687567	0.714506	0.687567	4.450492e-02	0.714506	4.932211e-02	None
7	DecisionTreeRegressor	1.000000	0.682451	1.000000	8.700672e-33	0.682451	5.485995e-02	None
8	KNeighborsRegressor	0.767498	0.667002	0.767498	3.311893e-02	0.667002	5.752905e-02	None
9	Regression	1.000000	1.000000	1.000000	1.056182e+01	1.000000	7.126381e+00	pca
10	Ridge	1.000000	1.000000	1.000000	1.056182e+01	1.000000	7.126381e+00	pca
11	XGBRegressor	0.999993	0.999559	0.999993	3.898812e+04	0.999559	3.379898e+06	pca
12	DecisionTreeRegressor	1.000000	0.999528	1.000000	0.000000e+00	0.999528	3.623034e+06	pca
13	KNeighborsRegressor	0.996033	0.989008	0.996033	2.365888e+07	0.989008	8.431343e+07	pca
14	CatBoostRegressor	0.997999	0.984852	0.997999	1.193513e+07	0.984852	1.161924e+08	pca
15	LGBMRegressor	0.983063	0.964952	0.983063	1.010236e+08	0.964952	2.688326e+08	pca
16	svm	-0.040762	-0.021243	-0.040762	6.207679e+09	-0.021243	7.833264e+09	pca

This is rank by lowest mse then highest r2_score, result shows that AutoML - 15 generation has the most favorable