Mid-Term Project : Predicting House Prices

Noam Atias - 311394357

Channel Michaeli - 208491787

Our job is to predict the sales price for each house. For each Id in the test set, we need to predict the value of the SalePrice variable. Submissions are evaluated on Root-Mean-Squared-Error (RMSE) between the logarithm of the predicted value and the logarithm of the observed sales price. (Taking logs means that errors in predicting expensive houses and cheap houses will affect the result equally.) For this competition, we will use RidgeCV Regression to predict the house price.

Ridge regression is a model tuning method that is used to analyze any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values being far away from the actual values. Lambda is the penalty term. λ given here is denoted by an alpha parameter in the ridge function. So, by changing the values of alpha, we are controlling the penalty term. The higher the values of alpha, the bigger is the penalty and therefore the magnitude of coefficients is reduced.

Import Packages and Datasets

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import warnings
warnings.filterwarnings('ignore')
import joblib
import sys
sys.modules['sklearn.externals.joblib'] = joblib
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import RidgeCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from scipy import stats
from scipy.stats import norm, skew
```

In [ ]:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [ ]:

```
file_path1 = '/content/gdrive/My Drive/Colab Notebooks/house-prices-advanced-regression-techniques/train.csv'
file_path2 = '/content/gdrive/My Drive/Colab Notebooks/house-prices-advanced-regression-techniques/test.csv'
```

In [ ]:

```
train = pd.read_csv(file_path1)
test = pd.read_csv(file_path2)
```

Data Analysis

In [ ]:

```
print("The train data size before dropping Id feature is : {}  ".format(train.shape))
print("The test data size before dropping Id feature is : {} ".format(test.shape))
```

```
The train data size before dropping Id feature is : (1460, 81)
The test data size before dropping Id feature is : (1459, 80)
```

1. Sample Train Dataset

In [ ]:

```
train.head()
```

Out[ ]:

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

5 rows × 81 columns

2. Sample Test Dataset

In [ ]:

```
test.head()
```

Out[ ]:

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

3. The Feaures In The Dataset

In [ ]:

```
train.columns
```

Out[ ]:

```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
```

```
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

4. Train Dataset Statistics

In [ ]:

```
train.describe()
```

Out[ ]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasV
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.0
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.6
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.0
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.0
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.0
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.0
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.0
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.0

5. Summary statistics for categorical values

In [ ]:

```
display(train.describe(include= ['O']))
display(test.describe(include= ['O']))
```

	MSZoning	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Conditio
count	1460	1460	91	1460	1460	1460	1460	1460	1460	1460	14
unique	5	2	2	4	4	2	5	3	25	9	
top	RL	Pave	Grvl	Reg	Lvl	AllPub	Inside	Gtl	NAmes	Norm	No
freq	1151	1454	50	925	1311	1459	1052	1382	225	1260	14

	MSZoning	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Conditio
count	1455	1459	107	1459	1459	1457	1459	1459	1459	1459	14
unique	5	2	2	4	4	1	5	3	25	9	
top	RL	Pave	Grvl	Reg	Lvl	AllPub	Inside	Gtl	NAmes	Norm	No
freq	1114	1453	70	934	1311	1457	1081	1396	218	1251	14

6. Checking the Correlation between the variables and the target variable

In [ ]:

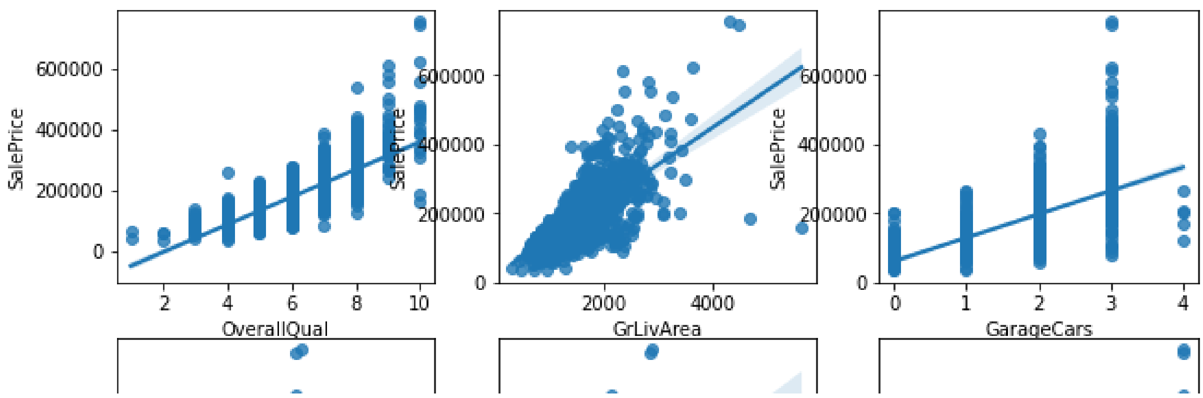
```
print(train.corr()['SalePrice'].sort_values(ascending=False))
```

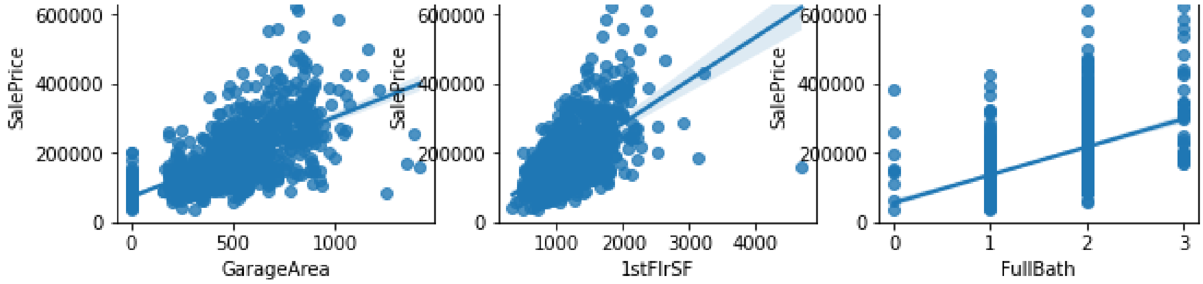
```
SalePrice        1.000000
OverallQual      0.790982
GrLivArea        0.708624
GarageCars       0.640409
GarageArea       0.623431
TotalBsmtSF      0.613581
1stFlrSF         0.605852
FullBath         0.560664
TotRmsAbvGrd     0.533723
YearBuilt        0.522897
YearRemodAdd     0.507101
GarageYrBlt      0.486362
MasVnrArea       0.477493
Fireplaces       0.466929
BsmtFinSF1       0.386420
LotFrontage      0.351799
WoodDeckSF       0.324413
2ndFlrSF         0.319334
OpenPorchSF      0.315856
HalfBath         0.284108
LotArea          0.263843
BsmtFullBath     0.227122
BsmtUnfSF        0.214479
BedroomAbvGr     0.168213
ScreenPorch      0.111447
PoolArea         0.092404
MoSold           0.046432
3SsnPorch        0.044584
BsmtFinSF2      -0.011378
BsmtHalfBath    -0.016844
MiscVal         -0.021190
Id              -0.021917
LowQualFinSF    -0.025606
YrSold          -0.028923
OverallCond     -0.077856
MSSubClass      -0.084284
EnclosedPorch   -0.128578
KitchenAbvGr    -0.135907
Name: SalePrice, dtype: float64
```

The top 6 variables with the best correlation :

In [ ]:

```
top_6_corr=['OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea', '1stFlrSF', 'FullBat
h']
prows = 2
pcols = 3
fig, axs = plt.subplots(prows, pcols, figsize=(pcols*3.5, prows*3))
for r in range(0,prows):
    for c in range(0,pcols):
        i = r*pcols+c
        col=top_6_corr[i]
        sns.regplot(x=train[col], y=train['SalePrice'], ax = axs[r][c])
```



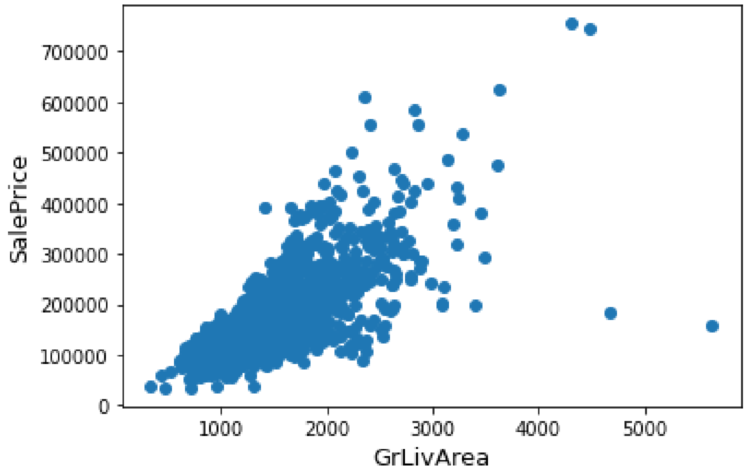


Data Preprocessing

1. Outliers

In [ ]:

```
fig, ax = plt.subplots()
ax.scatter(x = train['GrLivArea'], y = train['SalePrice'])
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)
plt.show()
```

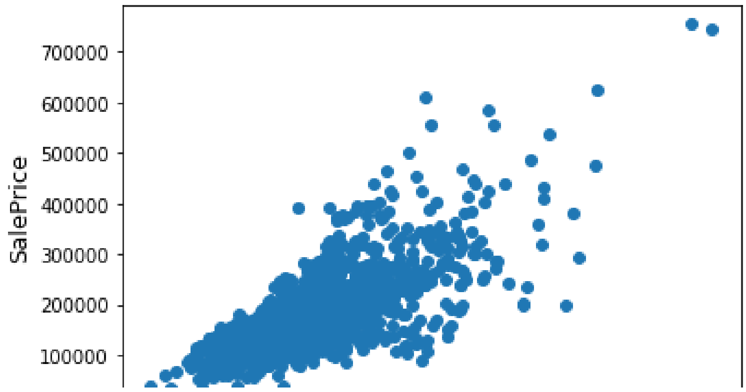


We can see that the points in the lower right corner have extremely large GrLivArea and at low low SalePrice, therefore those points are outliers and we can remove them. Then, we will show the plot one more time without those outliers:

In [ ]:

```
train= train.drop(train[(train['GrLivArea']>4000) & (train['SalePrice']<300000)].index)

fig, ax = plt.subplots()
ax.scatter(train['GrLivArea'], train['SalePrice'])
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)
plt.show()
print("The train data size is : {}  ".format(train.shape))
print("The test data size is : {} ".format(test.shape))
```




```
The train data size is : (1458, 81)
The test data size is : (1459, 80)
```

2. Saving the IDs and removing from datasets

In [ ]:

```
train_id = train['Id']
test_id = test['Id']
train = train.drop('Id',1)
test = test.drop('Id',1)
```

3. Splitting the target variable

In [ ]:

```
y_train = train['SalePrice']
x_train = train.drop('SalePrice', 1)
x_test = test
```

4. Dealing with missing values by each feature

In [ ]:

```
all_data = pd.concat((x_train, x_test)).reset_index(drop=True)
print("all_data size is : {}".format(all_data.shape))
```

```
all_data size is : (2917, 79)
```

In [ ]:

```
y_train.isnull().sum()
```

Out[ ]:

0

We've checked that there are no missing values in the target variable.

Now, we will calculate the number and the percentage of missing values by each feature :

In [ ]:

```
Total = all_data.isnull().sum().sort_values(ascending=False)
Percent = (all_data.isnull().sum()/all_data.shape[0]).sort_values(ascending=False)*100
missing_data = pd.concat([Total, Percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(35)
```

Out[ ]:

	Total	Percent
PoolQC	2908	99.691464
MiscFeature	2812	96.400411
Alley	2719	93.212204
Fence	2346	80.425094
FireplaceQu	1420	48.680151
LotFrontage	486	16.660953
GarageCond	159	5.450806

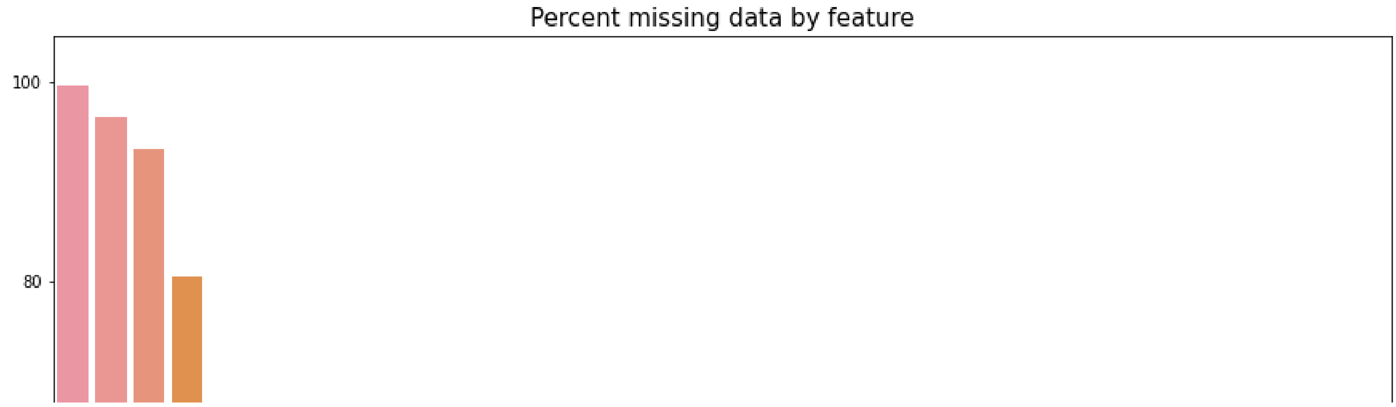
	Total	Percent
GarageQual	159	5.450806
GarageYrBlt	159	5.450806
GarageFinish	159	5.450806
GarageType	157	5.382242
BsmtCond	82	2.811107
BsmtExposure	82	2.811107
BsmtQual	81	2.776826
BsmtFinType2	80	2.742544
BsmtFinType1	79	2.708262
MasVnrType	24	0.822763
MasVnrArea	23	0.788481
MSZoning	4	0.137127
BsmtHalfBath	2	0.068564
Utilities	2	0.068564
Functional	2	0.068564
BsmtFullBath	2	0.068564
BsmtFinSF2	1	0.034282
BsmtFinSF1	1	0.034282
Exterior2nd	1	0.034282
BsmtUnfSF	1	0.034282
TotalBsmtSF	1	0.034282
Exterior1st	1	0.034282
SaleType	1	0.034282
Electrical	1	0.034282
KitchenQual	1	0.034282
GarageArea	1	0.034282
GarageCars	1	0.034282
OverallQual	0	0.000000

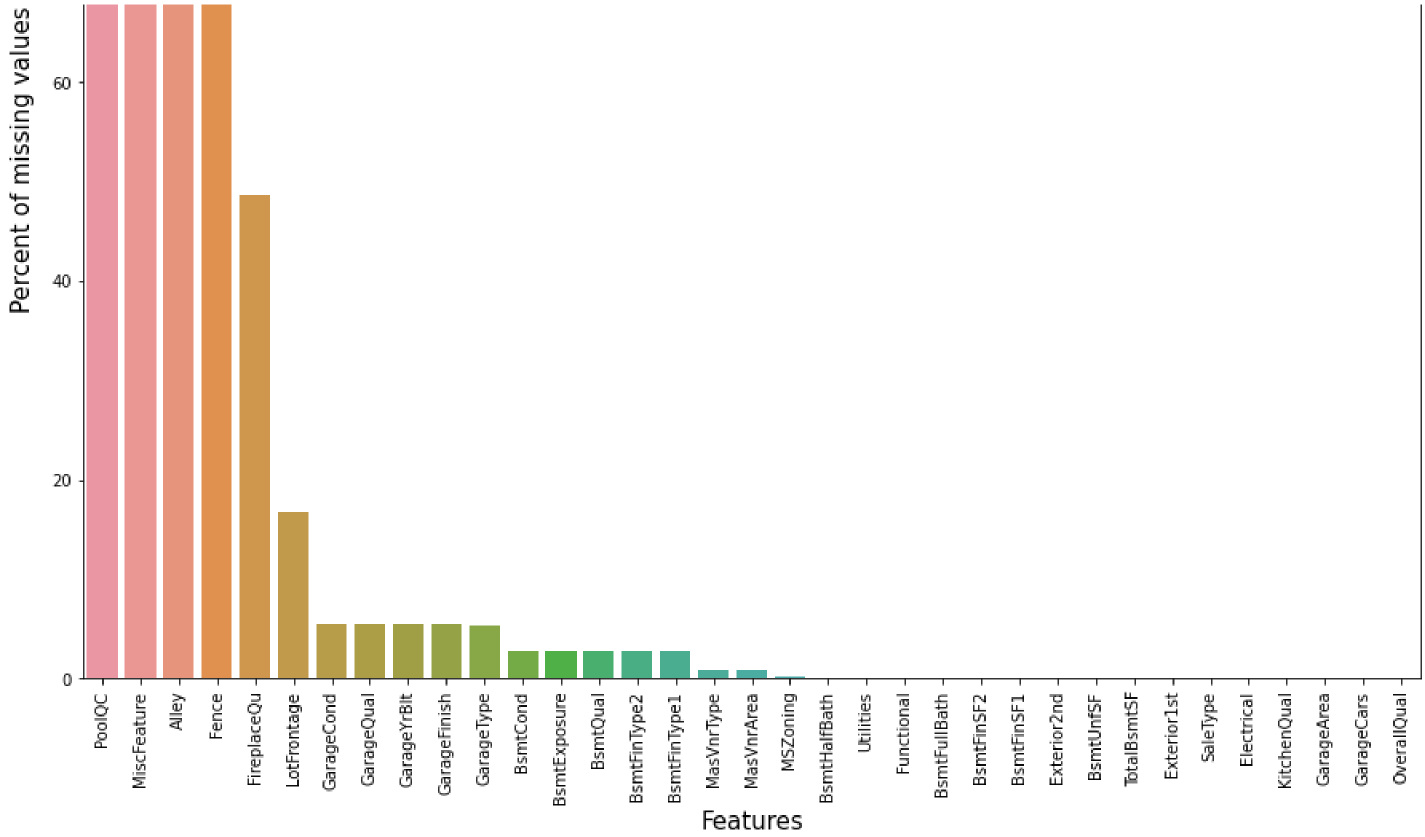
In [ ]:

```
f, ax = plt.subplots(figsize=(15, 12))
plt.xticks(rotation='90')
Percent = Percent[:35]
sns.barplot(x=Percent.index, y=Percent)
plt.xlabel('Features', fontsize=15)
plt.ylabel('Percent of missing values', fontsize=15)
plt.title('Percent missing data by feature', fontsize=15)
```

Out[ ]:

```
Text(0.5, 1.0, 'Percent missing data by feature')
```





First, we can impute some of the missing data in the features as following:

1. **PoolQC - 99% of the values are missing, and it makes sense that most of the houses does'nt have pool, so we can fill in the missing values with 'None' (no pool).**
2. **MiscFeature - NA means no misc feature, so we can fill in the missing values with 'None'.**
3. **Alley - NA means no alley access, so we can fill in the missing values with 'None'.**
4. **Fence - NA means no fence, so we can fill in the missing values with 'None'.**
5. **FireplaceQu - NA means no fireplace, so we can fill in the missing values with 'None'.**
6. **LotFrontage - replacing missing values with the most common value.**
7. **GarageType, GarageFinish, GarageQual, GarageCond - Replacing missing values with 'None'.**
8. **GarageYrBlt, GarageArea, GarageCars - Replacing missing values with 0.**
9. **BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, BsmtFullBath, BsmtHalfBath - Replacing missing values with 0 for having no basement.**
10. **BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1 and BsmtFinType2 - Replacing missing values with 'None' for having no basement.**
11. **Functional - NA means value 'typ'.**
12. **We will replace all other missing values with the most common value of each feature.**

In [ ]:

```
all_data["PoolQC"] = all_data["PoolQC"].fillna("None")
all_data["MiscFeature"] = all_data["MiscFeature"].fillna("None")
all_data["Alley"] = all_data["Alley"].fillna("None")
all_data["Fence"] = all_data["Fence"].fillna("None")
all_data["FireplaceQu"] = all_data["FireplaceQu"].fillna("None")
for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'):
    all_data[col] = all_data[col].fillna('None')
for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):
    all_data[col] = all_data[col].fillna(0)
for col in ('BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF', 'BsmtFullBath', 'Bsmt
HalfBath'):
    all_data[col] = all_data[col].fillna(0)
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
    all_data[col] = all_data[col].fillna('None')
```

Now all that is left is to replace all remaining missing values with the most common value of each feature. Let us check again the number of missing values for each variable:

In [ ]:

```
Total = all_data.isnull().sum().sort_values(ascending=False)
Percent = (all_data.isnull().sum()/all_data.shape[0]).sort_values(ascending=False)*100
missing_data = pd.concat([Total, Percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(12)
```

Out[ ]:

	Total	Percent
LotFrontage	486	16.660953
MasVnrType	24	0.822763
MasVnrArea	23	0.788481
MSZoning	4	0.137127
Functional	2	0.068564
Utilities	2	0.068564
Exterior1st	1	0.034282
SaleType	1	0.034282
Electrical	1	0.034282
Exterior2nd	1	0.034282
KitchenQual	1	0.034282
ExterCond	0	0.000000

In [ ]:

```
all_data = all_data.apply(lambda x:x.fillna(x.value_counts().index[0]))
print("number of missing values : " ,all_data.isnull().sum().max())
print("all_data size is : {}".format(all_data.shape))
```

```
number of missing values :  0
all_data size is : (2917, 79)
```

For all the other missing values, we replaced them with the most common value in each feature. Now, there are no missing values in the data.

5. Scaling datasets

We will scale the values of the numerical features by subtracting the mean and dividing by the standard deviation for both test and train features:

In [ ]:

```
for i in all_data.columns:
  if all_data[i].dtype != "object":
    all_data[i]=(all_data[i]-all_data[i].mean())/(all_data[i].std())
print('Shape of all_data dataset: {}'.format(all_data.shape))
all_data.head()
```

```
Shape of all_data dataset: (2917, 79)
```

Out[ ]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Nei
0	0.067343	RL	-0.125728	-0.216400	Pave	None	Reg	Lvl	AllPub	Inside	Gtl	
1	-0.873122	RL	0.585667	-0.069097	Pave	None	Reg	Lvl	AllPub	FR2	Gtl	
2	0.067343	RL	0.016551	0.142251	Pave	None	IR1	Lvl	AllPub	Inside	Gtl	

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Nei
3	0.302459	RL	-0.362689	-0.075501	Pave	None	IR1	Lvl	AllPub	Corner	Gtl	
4	0.067343	RL	0.775372	0.527801	Pave	None	IR1	Lvl	AllPub	FR2	Gtl	

6. Encoding the categorical features

In [ ]:

```
print('The categorial features are: {}'.format(all_data.select_dtypes(include='object').c
olumns))
print('The number of categorial features is: {}'.format(all_data.select_dtypes(include='o
bject').columns.shape[0]))
```

```
The categorial features are: Index(['MSZoning', 'Street', 'Alley', 'LotShape', 'LandConto
ur', 'Utilities',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation',
       'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
       'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
       'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual',
       'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature',
       'SaleType', 'SaleCondition'],
      dtype='object')
The number of categorial features is: 43
```

We can see in the description file that some categorical variables contain information in their ordering set. For example, the feature FireplaceQu has the categories:

Ex: Excellent - Exceptional Masonry Fireplace

Gd: Good - Masonry Fireplace in the main level

TA: Average - Prefabricated Fireplace in the main living area or Masonry Fireplace in basement

Fa: Fair - Prefabricated Fireplace in basement

Po: Poor - Ben Franklin Stove

None: No Fireplace

Therefore, we will do encoding for those variables in the dataset:

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
cols = ('FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond',
        'ExterQual', 'ExterCond','HeatingQC', 'PoolQC', 'KitchenQual', 'BsmtFinType1',
        'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish', 'LandSlop
e',
        'LotShape', 'PavedDrive', 'Street', 'Alley', 'CentralAir')

for c in cols:
    lbl = LabelEncoder()
    lbl.fit(list(all_data[c].values))
    all_data[c] = lbl.transform(list(all_data[c].values))

print('Shape of all dataset: {}'.format(all_data.shape))
all_data.head()
```

```
Shape of all dataset: (2917, 79)
```

Out[ ]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Nei

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Nei
0	0.067343	RL	-0.125728	0.216400	1	1	3	Lvl	AllPub	Inside	0	
1	-0.873122	RL	0.585667	-0.069097	1	1	3	Lvl	AllPub	FR2	0	
2	0.067343	RL	0.016551	0.142251	1	1	0	Lvl	AllPub	Inside	0	
3	0.302459	RL	-0.362859	-0.075501	1	1	0	Lvl	AllPub	Corner	0	
4	0.067343	RL	0.775372	0.527801	1	1	0	Lvl	AllPub	FR2	0	

Let us check how many categorical features are left:

In [ ]:

```
print('The categorial features are: {}'.format(all_data.select_dtypes(include='object').c
olumns))
print('The number of categorial features is: {}'.format(all_data.select_dtypes(include='o
bject').columns.shape[0]))
```

```
The categorial features are: Index(['MSZoning', 'LandContour', 'Utilities', 'LotConfig',
'Neighborhood',
       'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle',
       'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'Foundation',
       'Heating', 'Electrical', 'GarageType', 'MiscFeature', 'SaleType',
       'SaleCondition'],
      dtype='object')
The number of categorial features is: 21
```

For all remaining categorical features, we will use get dummies:

In [ ]:

```
all_data = pd.get_dummies(all_data)
print('Shape of all dataset: {}'.format(all_data.shape))
```

```
Shape of all dataset: (2917, 221)
```

7. Splitting the data to train, test and validation:

In [ ]:

```
#splitting all_data to train and test
x_train = all_data[:1458]
X_test = all_data[1458:]
#splitting train dataset to train and validation
X_train = x_train[:1210]
X_val = x_train[1210:]
Y_train = y_train[:1210]
Y_val = y_train[1210:]

print('Shape of test dataset: {}'.format(X_test.shape))
print('Shape of train dataset: {}'.format(X_train.shape))
print('Shape of validation dataset: {}'.format(X_val.shape))
print('Shape of train target variable: {}'.format(Y_train.shape))
print('Shape of validation target variable: {}'.format(Y_val.shape))
```

```
Shape of test dataset: (1459, 221)
Shape of train dataset: (1210, 221)
Shape of validation dataset: (248, 221)
Shape of train target variable: (1210,)
Shape of validation target variable: (248,)
```

In [ ]:

```
X_val.head()
```

Out[ ]:

	MSSubClass	LotFrontage	LotArea	Street	Alley	LotShape	LandSlope	OverallQual	OverallCond	YearBuilt	YearRem
1210	-0.167773	4.000359	0.255482	1	1	0	1	1.360350	1.288861	0.551789	0.9
1211	-0.638006	-0.837122	-0.102400	1	1	3	0	-1.483176	0.390723	-1.000032	-1.6
1212	0.537576	-0.362859	0.013649	1	1	0	0	-1.483176	3.085138	-0.207613	0.8
1213	0.655134	0.063978	0.008398	1	1	0	0	-0.772295	-0.507416	-0.306665	-1.0
1214	-0.873122	1.486766	-0.390089	1	1	0	0	-0.772295	-0.507416	-0.174595	-0.8

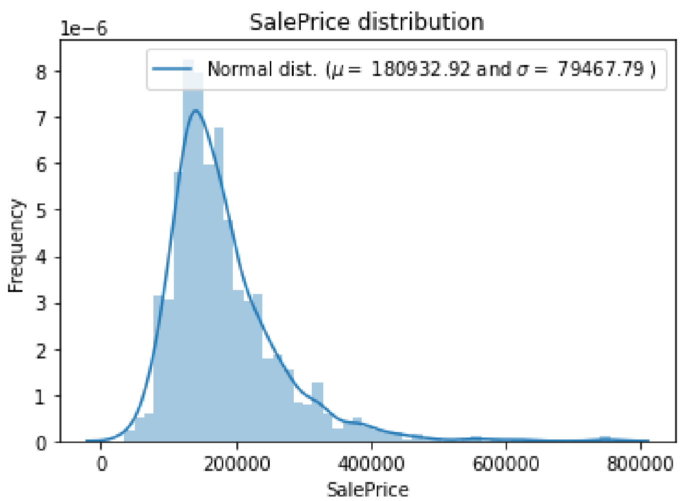
5 rows × 221 columns

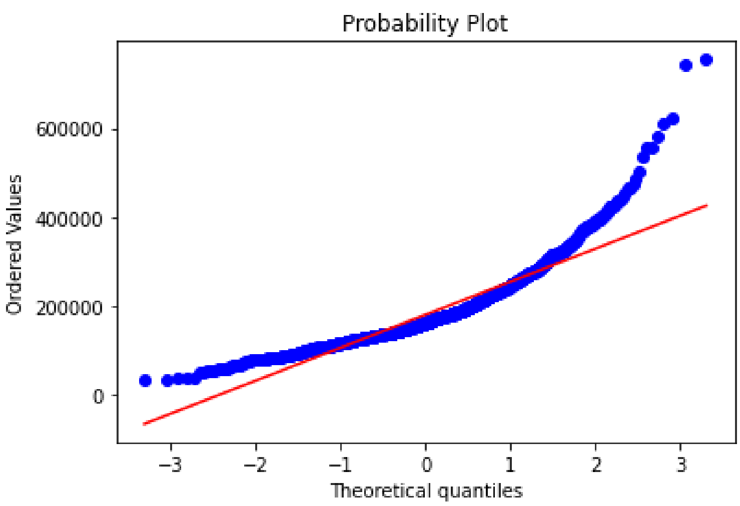
8. Log-transformation of the target variable

In [ ]:

```
sns.distplot(train['SalePrice'])
(mu, sigma) = norm.fit(train['SalePrice'])
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
            loc='best')
plt.ylabel('Frequency')
plt.title('SalePrice distribution')

fig = plt.figure()
res = stats.probplot(train['SalePrice'], plot=plt)
plt.show()
```





In [ ]:

```
print("Skewness: %f" % train['SalePrice'].skew())
```

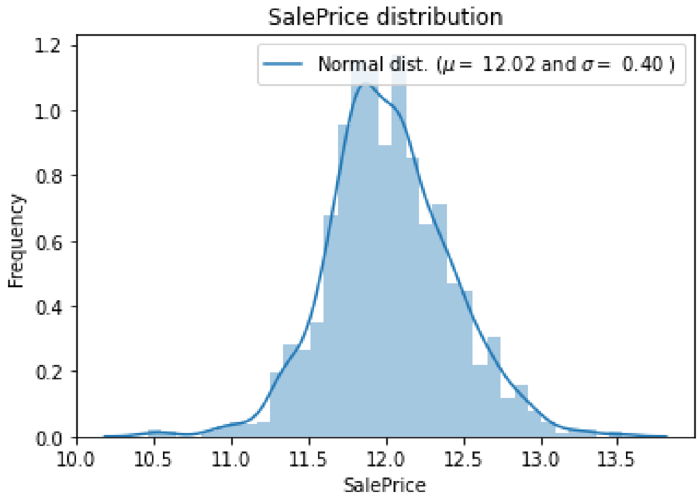
```
Skewness: 1.881296
```

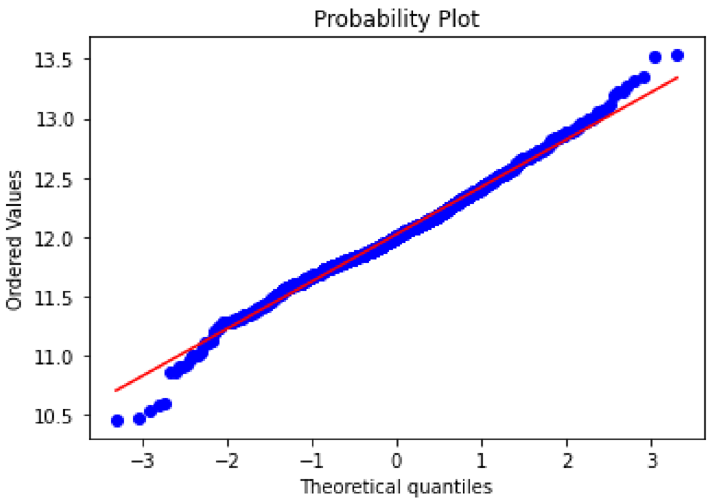
The target variable SalePrice is right-skewed, meaning that the mean is biased towards a higher price than the median. It is also deviating from the normal distribution. Below we see that log(y+1) provides a nice distribution:

In [ ]:

```
sns.distplot(np.log1p(train['SalePrice']))
(mu, sigma) = norm.fit(np.log1p(train['SalePrice']))
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
            loc='best')
plt.ylabel('Frequency')
plt.title('SalePrice distribution')

fig = plt.figure()
res = stats.probplot(np.log1p(train['SalePrice']), plot=plt)
plt.show()
```





We see that the target variable has a more symmetric distribution in log-space, and therefore we perform the following simple transformation:

In [ ]:

```
Y_train = np.log1p(Y_train)
Y_val = np.log1p(Y_val)
```

Model Building

In [ ]:

```
model = RidgeCV()
```

What makes this regression model more effective is its ability to regularize. The term "regularizing" stands for models' ability to structurally prevent overfitting by imposing a penalty on the coefficients. The main tuning parameter for the regularization model is alpha - a regularization parameter that measures how flexible our model is. When alpha is too large the regularization is too strong, and the model cannot capture all the complexities in the data. However, if we let the model be too flexible (alpha small) the model begins to overfit.

We will use K-fold technique for cross-validation:

In [ ]:

```
kfold=KFold(n_splits=5, random_state=100, shuffle=True)
```

Model Fitting and Evaluation

1. Make Predictions

We will find the optimal value of alpha and make predictions:

In [ ]:

```
alphas = list(np.arange(1e-3,20,1e-1))
clf = RidgeCV(alphas=alphas,cv=kfold).fit(X_train, Y_train)
y_train_pred = clf.predict(X_train)
y_val_pred = clf.predict(X_val)
y_test_pred = clf.predict(X_test)
```

2. Evaluate The Model

We will use the RMSE and R^2 and evaluate how good the value of alpha that is chosen:

In [ ]:

```
#best alpha
print('The best value of alpha is : {}'.format(clf.alpha_))
#R^2 score
print('R^2 score for training is : {}'.format(clf.score(X_train, Y_train)))
print('R^2 score for validation is : {}'.format(clf.score(X_val, Y_val)))

#rmse score
print('RMSE for training is : {}'.format(np.sqrt(mean_squared_error(Y_train, y_train_pred
))))
print('RMSE for validation is : {}'.format(np.sqrt(mean_squared_error(Y_val, y_val_pred))
))
```

```
The best value of alpha is : 18.801000000000002
R^2 score for training is : 0.9373253893260599
R^2 score for validation is : 0.9099295234081932
RMSE for training is : 0.10086817857730129
RMSE for validation is : 0.11489958962635259
```

We found that the optimal value of alpha, such that minimize the RMSE and get the R^2 closer to 1 is alpha = 18.8.

For the Ridge regression, we get a RMSE of about 0.1008 for training and 0.114 for validation.

3. Inverse-transformation of the target variable

We will inverse transfom the values of the target and the predictions:

In [ ]:

```
y_train1 = np.exp(Y_train)-1
y_val1 = np.exp(Y_val)-1
y_train_pred1 = np.exp(y_train_pred)-1
y_val_pred1 = np.exp(y_val_pred)-1
y_test_pred1 = np.exp(y_test_pred)-1
```

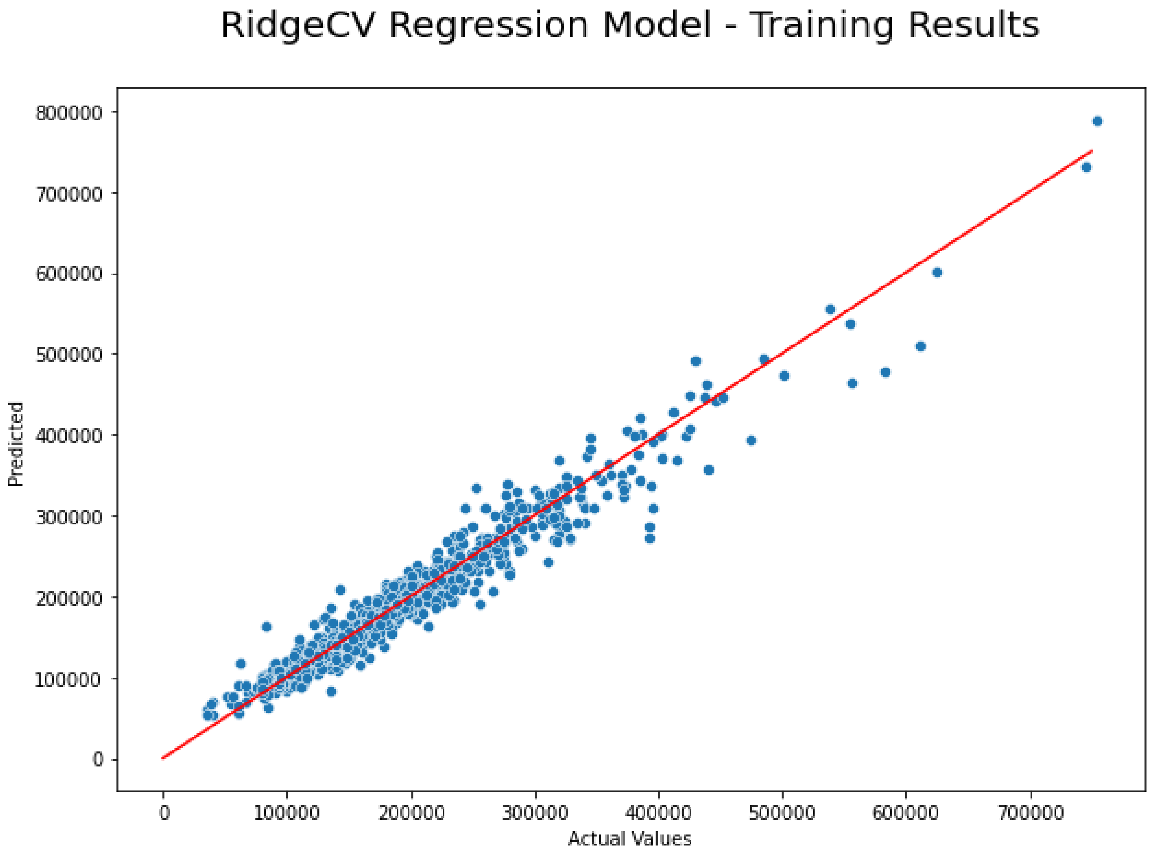
4. Results Presentation

In [ ]:

```
predict_data1 = pd.DataFrame({"Actual Values" : y_train1, "Predicted" : y_train_pred1})
plt.figure(figsize=(10, 7))
sns.scatterplot(data = predict_data1, x = "Actual Values", y = "Predicted")
sns.lineplot(x = [0, 750000], y = [0, 750000],  color = "red")
plt.title("RidgeCV Regression Model - Training Results\n", fontsize = 20)
```

Out[ ]:

```
Text(0.5, 1.0, 'RidgeCV Regression Model - Training Results\n')
```

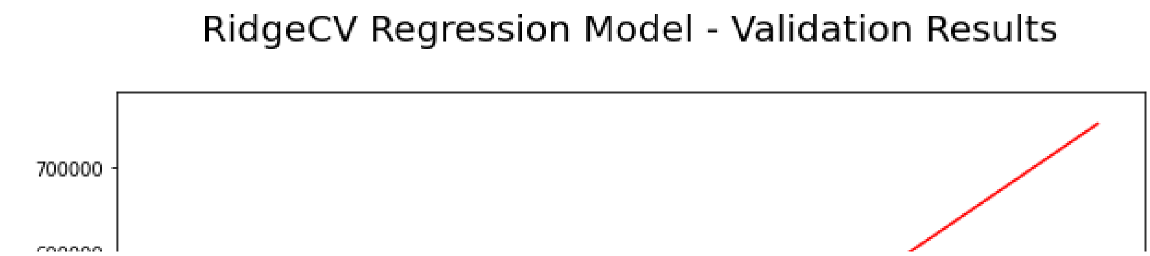


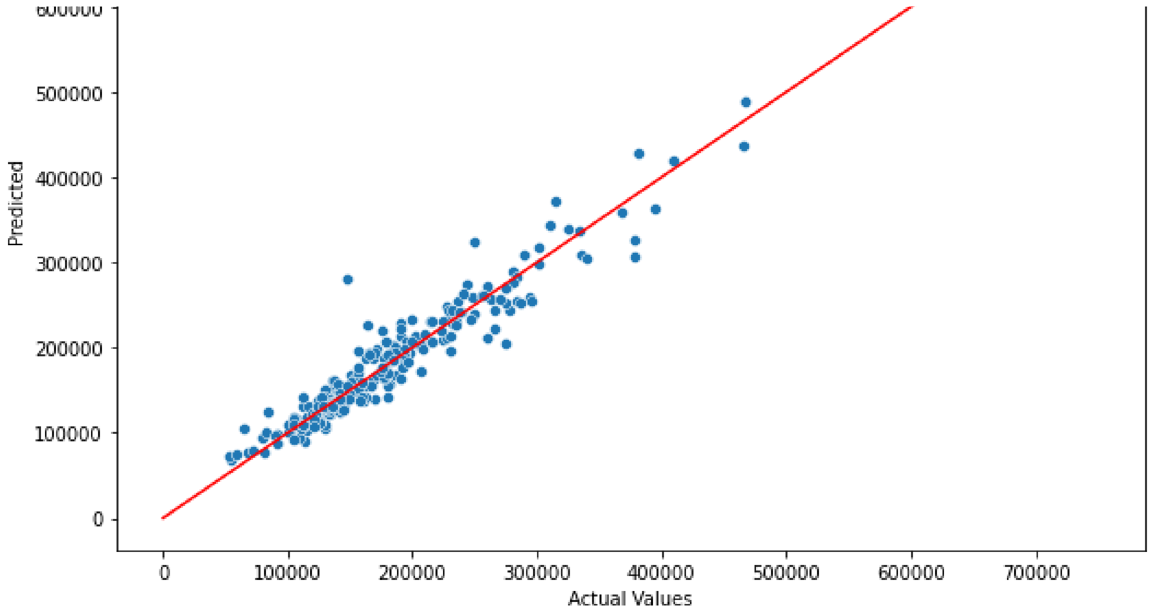
In [ ]:

```
predict_data = pd.DataFrame({"Actual Values" : y_val1, "Predicted" : y_val_pred1})
plt.figure(figsize=(10, 7))
sns.scatterplot(data = predict_data, x = "Actual Values", y = "Predicted")
sns.lineplot(x = [0, 750000], y = [0, 750000],  color = "red")
plt.title("RidgeCV Regression Model - Validation Results\n", fontsize = 20)
```

Out[ ]:

```
Text(0.5, 1.0, 'RidgeCV Regression Model - Validation Results\n')
```





We can see that there is a linear connection between the predicted values and the actual values in the training and the validation (besides a few outliers). Therefore, the model we build fits well.

5. Submission

In [ ]:

```
my_submission = pd.DataFrame({'Id': test_id, 'SalePrice': y_test_pred1})
print(my_submission)
my_submission.to_csv('submission.csv', index=False)
```

```
        Id      SalePrice
0     1461  117108.013634
1     1462  159831.138955
2     1463  177329.580431
3     1464  197569.065651
4     1465  194753.507250
...    ...            ...
1454  2915   87426.511591
1455  2916   83052.761238
1456  2917  176729.322211
1457  2918  114684.703011
1458  2919  230194.353121

[1459 rows x 2 columns]
```