In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as MSE
```

In [2]:
```
#pip install sklearn
```

In [3]:
```
#pip install lime
```

In [4]:
```
#pip install mlxtend
```

In [5]:
```
#pip install xgboost
```

In [6]:
```
#read the data
houses= pd.read_csv('./Data/train.csv')
```

In [7]:
```
#see the data
houses
```

Out[7]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub
...	...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub

1460 rows × 81 columns

In [8]:
```
houses.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 [9]:
```
#lets see if there are any columns with missing values
null_columns=houses.columns[houses.isnull().any()]
houses[null_columns].isnull().sum()
```

Out[9]:
```
LotFrontage      259
Alley           1369
MasVnrType         8
MasVnrArea         8
BsmtQual          37
BsmtCond          37
BsmtExposure      38
BsmtFinType1      37
BsmtFinType2      38
Electrical         1
FireplaceQu      690
GarageType        81
GarageYrBlt       81
GarageFinish      81
GarageQual        81
GarageCond        81
PoolQC          1453
Fence           1179
MiscFeature     1406
dtype: int64
```

In [10]:
```
StreetDict =   {'Grvl':0,
                "Pave":1,
                }
houses['Ordinal_Street'] = houses.Street.map(StreetDict)

houses = houses.drop('Street', axis = 1)
```

In [11]:
```
#Converted Alley to binary. Since most streets had no alley access.
AlleyDict =   {'Grvl':1,
               "Pave":1,
              0:0
               }

houses['Alley'].fillna(0, inplace = True)

houses['Binary_Alley'] = houses.Alley.map(AlleyDict)

houses = houses.drop('Alley', axis = 1)
```

In [12]:
```
#Change BsmtCond to ordinal Data, also NA meant no basement we replaced those with the m
#Because no basement is not necessarily worse than other conditions.
BsmtCondDict =   {'Ex':4,
                  'Gd':3,
                  'TA':2,
                  "Fa":1,
                  "Po":0,
                }
houses['Ordinal_BsmtCond'] = houses.BsmtCond.map(BsmtCondDict)
houses['Ordinal_BsmtCond'].fillna(houses['Ordinal_BsmtCond'].mean(), inplace = True)

houses = houses.drop('BsmtCond', axis = 1)
```

In [13]:
```
ExterQualDict =   {'Ex':4,
                  'Gd':3,
                  'TA':2,
                  "Fa":1,
                  "Po":0,
                }
houses['Ordinal_ExterQual'] = houses.ExterQual.map(ExterQualDict)

houses = houses.drop('ExterQual', axis = 1)
```

In [14]:
```
ExterCondDict =   {'Ex':4,
                  'Gd':3,
                  'TA':2,
                  "Fa":1,
                  "Po":0,
                }
houses['Ordinal_ExterCond'] = houses.ExterCond.map(ExterCondDict)

houses = houses.drop('ExterCond', axis = 1)
```

In [15]:
```
BsmtQualDict =   {'Ex':4,
                  'Gd':3,
                  'TA':2,
                  "Fa":1,
                  "Po":0,
                 }
houses['Ordinal_BsmtQual'] = houses.BsmtQual.map(BsmtQualDict)
houses['Ordinal_BsmtQual'].fillna(houses['Ordinal_BsmtQual'].mean(), inplace = True)

houses = houses.drop('BsmtQual', axis = 1)
```

In [16]:
```
# No basement scores were recorded as Na, which was confused as null value.
houses['BsmtExposure'].fillna("NoB", inplace = True)
houses['BsmtFinType1'].fillna(houses['BsmtFinType1'].mode(), inplace = True)
houses['BsmtFinType2'].fillna(houses['BsmtFinType2'].mode(), inplace = True)
```

In [17]:
```
HeatingQCDict =   {'Ex':4,
                  'Gd':3,
                  'TA':2,
                  "Fa":1,
                  "Po":0,
                }
houses['Ordinal_HeatingQC'] = houses.HeatingQC.map(HeatingQCDict)

houses = houses.drop('HeatingQC', axis = 1)
```

In [18]:
```
CentralAirDict = {'N':0,
                 "Y":1,
                 }
houses['Binary_CentralAir'] = houses.CentralAir.map(CentralAirDict)

houses = houses.drop('CentralAir', axis = 1)
```

In [19]:
```
KitchenQualDict =   {'Ex':4,
                  'Gd':3,
                  'TA':2,
                  "Fa":1,
                  "Po":0,
                }
houses['Ordinal_KitchenQual'] = houses.KitchenQual.map(KitchenQualDict)

houses = houses.drop('KitchenQual', axis = 1)
```

In [20]:
```
FireplaceQuDict =   {'Ex':4,
                    'Gd':3,
                    'TA':2,
                    "Fa":1,
                    "Po":0,
                   }
houses['Ordinal_FireplaceQu'] = houses.FireplaceQu.map(FireplaceQuDict)
houses['Ordinal_FireplaceQu'].fillna(houses['Ordinal_FireplaceQu'].mean(), inplace = Tru

houses = houses.drop('FireplaceQu', axis = 1)
```

In [21]:
```
# NA means no garage.
houses['GarageType'].fillna("NoG", inplace = True)
houses['GarageYrBlt'].fillna(houses['GarageYrBlt'].mode(), inplace = True)
```

In [22]:
```
GarageQualDict =   {'Ex':4,
                    'Gd':3,
                    'TA':2,
                    "Fa":1,
                    "Po":0,
                   }
houses['Ordinal_GarageQual'] = houses.GarageQual.map(GarageQualDict)
houses['Ordinal_GarageQual'].fillna(houses['Ordinal_GarageQual'].mean(), inplace = True)

houses = houses.drop('GarageQual', axis = 1)
```

In [23]:
```
GarageCondDict =   {'Ex':4,
                    'Gd':3,
                    'TA':2,
                    "Fa":1,
                    "Po":0,
                   }
houses['Ordinal_GarageCond'] = houses.GarageCond.map(GarageCondDict)
houses['Ordinal_GarageCond'].fillna(houses['Ordinal_GarageCond'].mean(), inplace = True)

houses = houses.drop('GarageCond', axis = 1)
```

In [24]:
```
#Converted Pool Quality to binary. Since most houses had no Pool
PoolQCDict =   {'Ex':1,
                'Gd':1,
                'TA':1,
                "Fa":1,
               0:0
               }

houses['PoolQC'].fillna(0, inplace = True)

houses['Binary_PoolQC'] = houses.PoolQC.map(PoolQCDict)

houses = houses.drop('PoolQC', axis = 1)
```

In [25]:
```
# No fence scores were recorded as Na, which was confused as null value.
houses['Fence'].fillna("NoF", inplace = True)
```

In [26]:
```
# No Masonry veneer type scores were recorded as Na, which was confused as null value.
houses['MasVnrType'].fillna("None", inplace = True)
```

In [27]:
```
# No basement fin scores were recorded as Na, which was confused as null value.
houses['Fence'].fillna("NoF", inplace = True)
```

In [28]:
```
#Converted Pool Quality to binary. Since most houses had no Pool
MiscFeatureDict =   {'Elev':1,
```

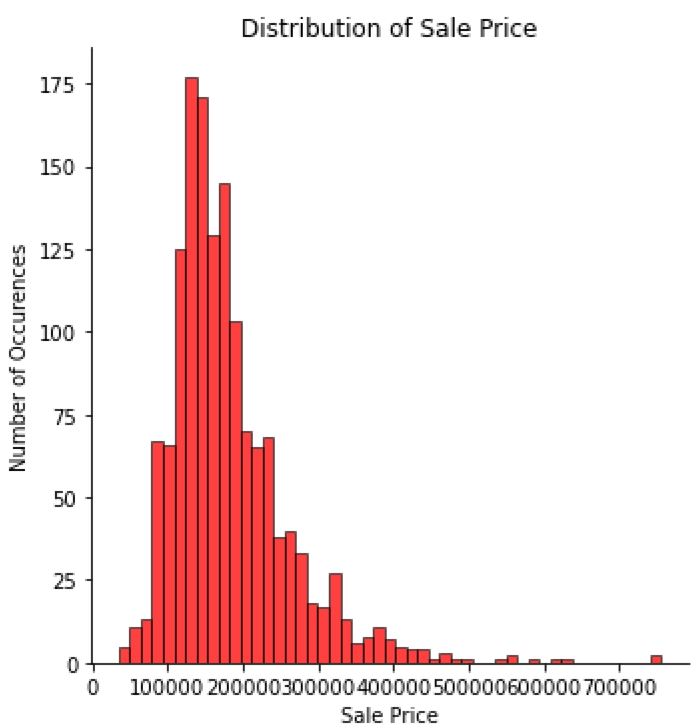
```
                  'Gar2':1,
                  'Othr':1,
                  "Shed":1,
                  "TenC":1,
                0:0
                }

houses['MiscFeature'].fillna(0, inplace = True)

houses['Binary_MiscFeature'] = houses.MiscFeature.map(MiscFeatureDict)

houses = houses.drop('MiscFeature', axis = 1)
```

In [29]:
```
# Handle numeric Na values.
houses['LotFrontage'].fillna(houses['LotFrontage'].mean(), inplace = True)
houses['MasVnrArea'].fillna(houses['MasVnrArea'].mean(), inplace = True)
houses['BsmtFinType1'].fillna(houses['BsmtFinType1'].mode(), inplace = True)
houses['BsmtFinType2'].fillna(houses['BsmtFinType2'].mode(), inplace = True)
houses['Electrical'].fillna(houses['Electrical'].mode(), inplace = True)
houses['GarageYrBlt'].fillna(houses['GarageYrBlt'].mean(), inplace = True)
houses['GarageFinish'].fillna(houses['GarageFinish'].mode(), inplace = True)
```

In [30]:
```
# Do one hot encoding for the rest of the categorical features
houses = pd.get_dummies(houses)
houses = houses.drop('Id', axis = 1)
```

In [31]:
```
#plot of the sale numbers due to price
sns.displot(houses['SalePrice'], color="r")
plt.title("Distribution of Sale Price")
plt.ylabel("Number of Occurences")
plt.xlabel("Sale Price")
```

Out[31]:
```
Text(0.5, 6.79999999999999, 'Sale Price')
```



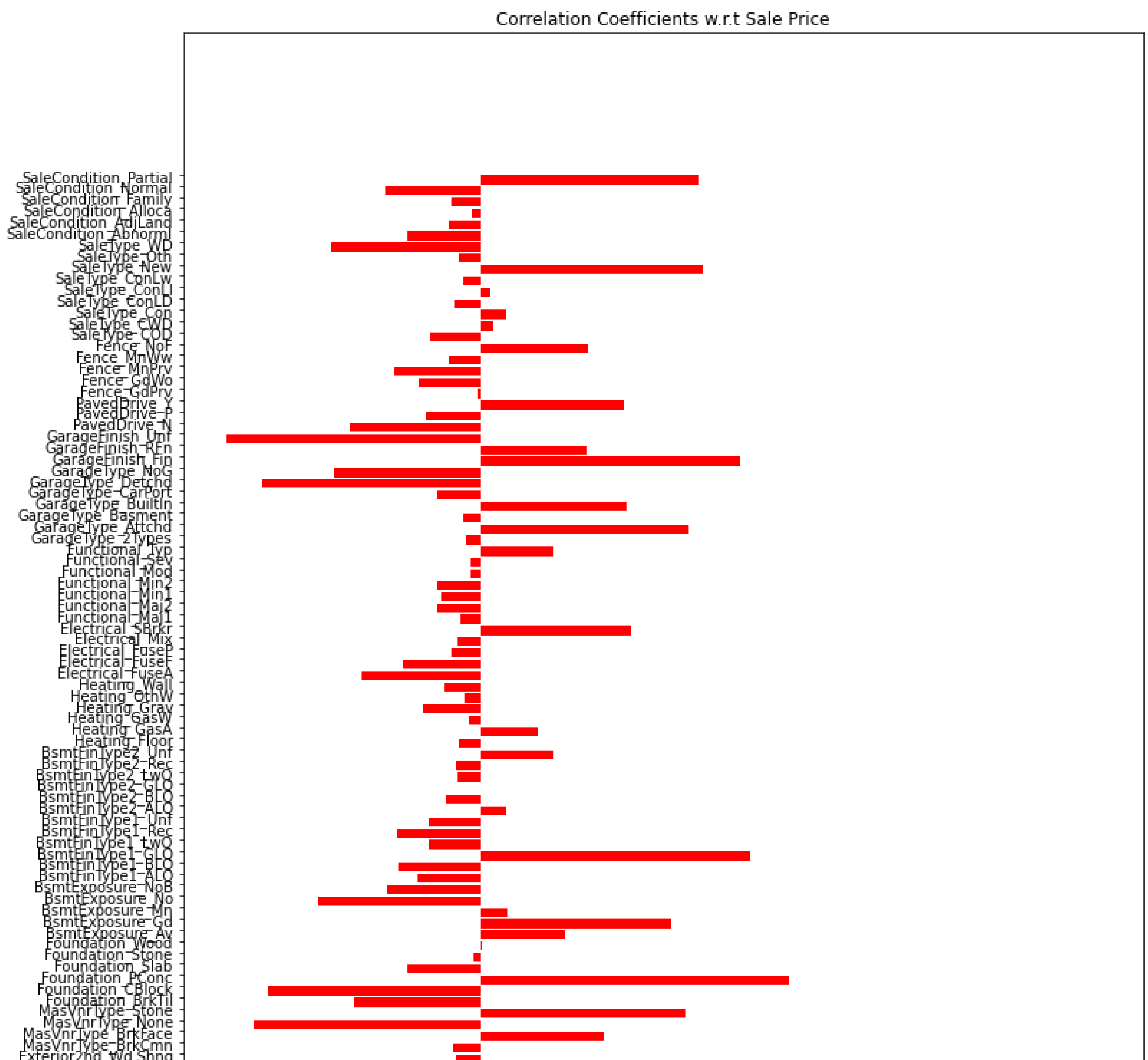
In [32]:
```
#finding corelations of features with SalePrice
corr = houses.corr()['SalePrice']
corr[np.argsort(corr, axis = 0)[::-1]]
```

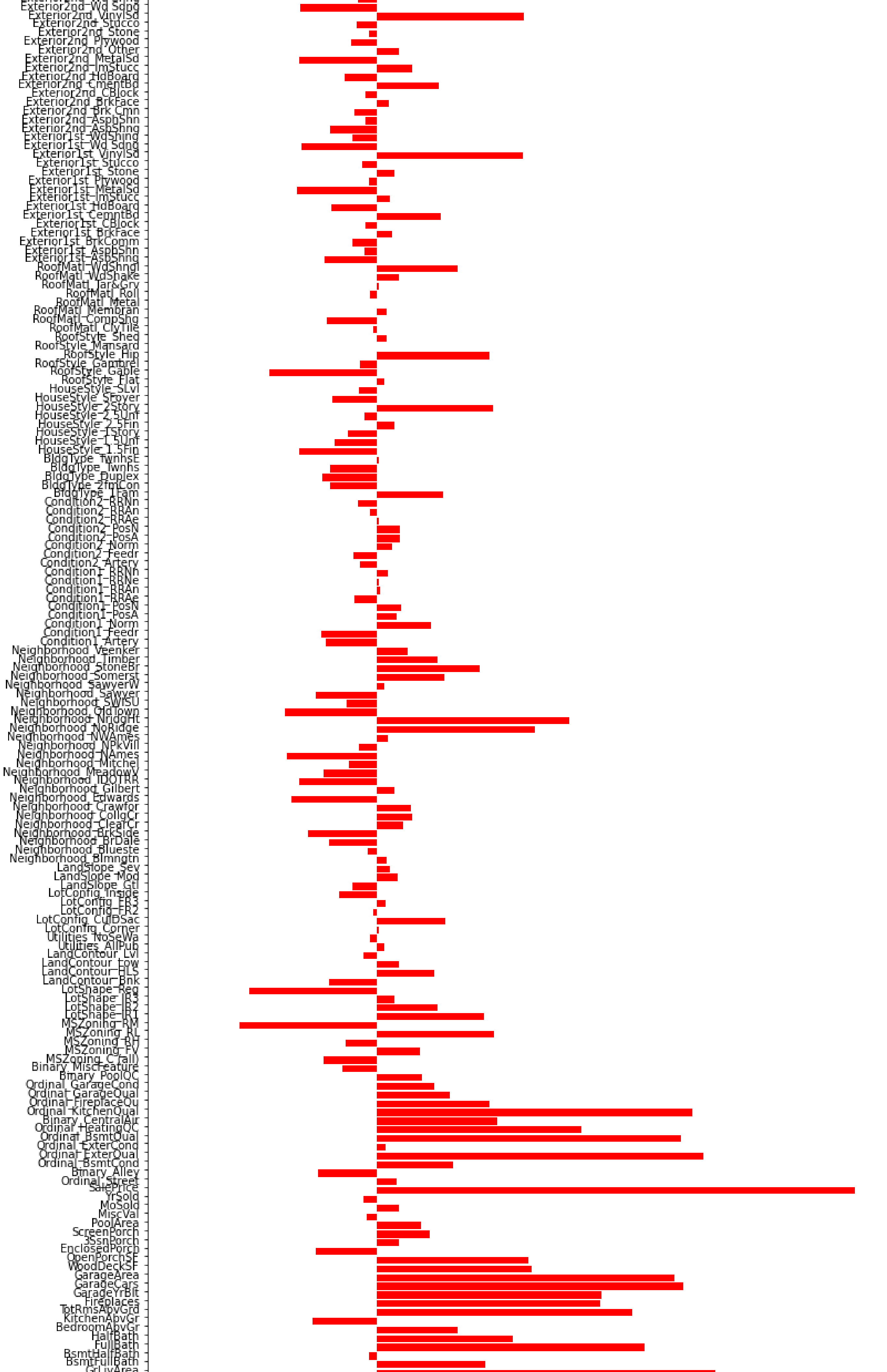
Out[32]:
```
SalePrice        1.000000
OverallQual      0.790982
GrLivArea        0.708624
```

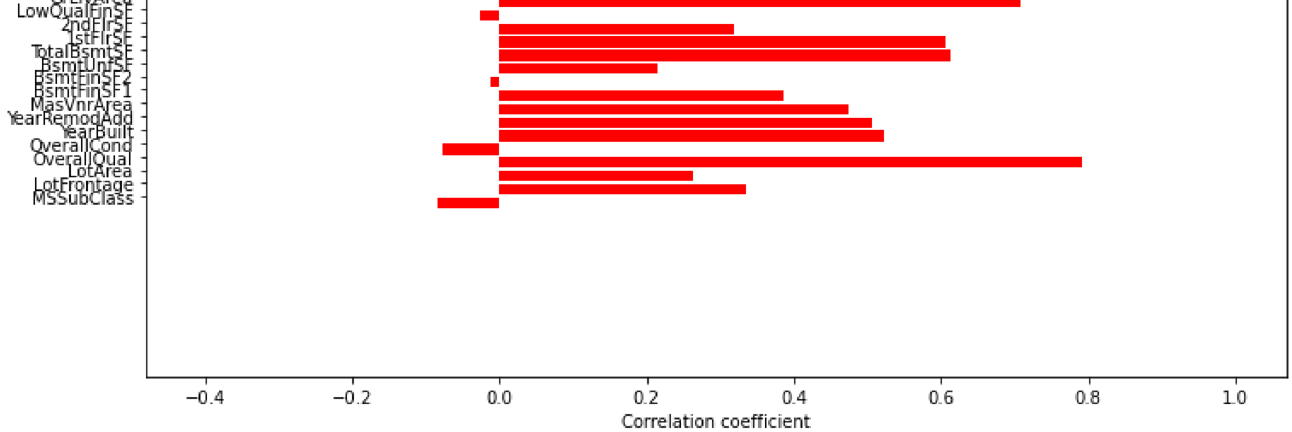
```
Ordinal_ExterQual       0.682639
Ordinal_KitchenQual     0.659600
                          ...
MSZoning_RM            -0.288065
Foundation_CBlock      -0.343263
GarageType_Detchd      -0.354141
MasVnrType_None        -0.367456
GarageFinish_Unf       -0.410608
Name: SalePrice, Length: 252, dtype: float64
```

In [33]:

```python
#plotting correlations
num_feat=houses.columns
labels = []
values = []
for col in num_feat:
    labels.append(col)
    values.append(np.corrcoef(houses[col].values, houses.SalePrice.values)[0,1])

ind = np.arange(len(labels))
width = 0.9
fig, ax = plt.subplots(figsize=(12,40))
rects = ax.barh(ind, np.array(values), color='red')
ax.set_yticks(ind+((width)/2.))
ax.set_yticklabels(labels, rotation='horizontal')
ax.set_xlabel("Correlation coefficient")
ax.set_title("Correlation Coefficients w.r.t Sale Price");
```




Exterior2nd_Wd Sdng
Exterior2nd_VinylSd
Exterior2nd_Stucco
Exterior2nd_Stone
Exterior2nd_Plywood
Exterior2nd_Other
Exterior2nd_MetalSd
Exterior2nd_ImStucc
Exterior2nd_HdBoard
Exterior2nd_CmentBd
Exterior2nd_CBlock
Exterior2nd_BrkFace
Exterior2nd_Brk Cmn
Exterior2nd_AsphShn
Exterior2nd_AsbShng
Exterior1st_WdShing
Exterior1st_Wd Sdng
Exterior1st_VinylSd
Exterior1st_Stucco
Exterior1st_Stone
Exterior1st_Plywood
Exterior1st_MetalSd
Exterior1st_ImStucc
Exterior1st_HdBoard
Exterior1st_CemntBd
Exterior1st_CBlock
Exterior1st_BrkFace
Exterior1st_BrkComm
Exterior1st_AsphShn
Exterior1st_AsbShng
RoofMatl_WdShngl
RoofMatl_WdShake
RoofMatl_Tar&Grv
RoofMatl_Roll
RoofMatl_Metal
RoofMatl_Membran
RoofMatl_CompShg
RoofMatl_ClyTile
RoofStyle_Shed
RoofStyle_Mansard
RoofStyle_Hip
RoofStyle_Gambrel
RoofStyle_Gable
RoofStyle_Flat
HouseStyle_SLvl
HouseStyle_SFoyer
HouseStyle_2Story
HouseStyle_2.5Unf
HouseStyle_2.5Fin
HouseStyle_1Story
HouseStyle_1.5Unf
HouseStyle_1.5Fin
BldgType_TwnhsE
BldgType_Twnhs
BldgType_Duplex
BldgType_2fmCon
BldgType_1Fam
Condition2_RRNn
Condition2_RRAn
Condition2_RRAe
Condition2_PosN
Condition2_PosA
Condition2_Norm
Condition2_Feedr
Condition2_Artery
Condition1_RRNn
Condition1_RRNe
Condition1_RRAn
Condition1_RRAe
Condition1_PosN
Condition1_PosA
Condition1_Norm
Condition1_Feedr
Condition1_Artery
Neighborhood_Veenker
Neighborhood_Timber
Neighborhood_StoneBr
Neighborhood_Somerst
Neighborhood_SawyerW
Neighborhood_Sawyer
Neighborhood_SWISU
Neighborhood_OldTown
Neighborhood_NridgHt
Neighborhood_NoRidge
Neighborhood_NWAmes
Neighborhood_NPkVill
Neighborhood_NAmes
Neighborhood_Mitchel
Neighborhood_MeadowV
Neighborhood_IDOTRR
Neighborhood_Gilbert
Neighborhood_Edwards
Neighborhood_Crawfor
Neighborhood_CollgCr
Neighborhood_ClearCr
Neighborhood_BrkSide
Neighborhood_BrDale
Neighborhood_Blueste
Neighborhood_Blmngtn
LandSlope_Sev
LandSlope_Mod
LandSlope_Gtl
LotConfig_Inside
LotConfig_FR3
LotConfig_FR2
LotConfig_CulDSac
LotConfig_Corner
Utilities_NoSeWa
Utilities_AllPub
LandContour_Lvl
LandContour_Low
LandContour_HLS
LandContour_Bnk
LotShape_Reg
LotShape_IR3
LotShape_IR2
LotShape_IR1
MSZoning_RM
MSZoning_RL
MSZoning_RH
MSZoning_FV
MSZoning_C (all)
Binary_MiscFeature
Binary_PoolQC
Ordinal_GarageCond
Ordinal_GarageQual
Ordinal_FireplaceQu
Ordinal_KitchenQual
Binary_CentralAir
Ordinal_HeatingQC
Ordinal_BsmtQual
Ordinal_ExterCond
Ordinal_ExterQual
Ordinal_BsmtCond
Binary_Alley
Ordinal_Street
SalePrice
YrSold
MoSold
MiscVal
PoolArea
ScreenPorch
3SsnPorch
EnclosedPorch
OpenPorchSF
WoodDeckSF
GarageArea
GarageCars
GarageYrBlt
Fireplaces
TotRmsAbvGrd
KitchenAbvGr
BedroomAbvGr
HalfBath
FullBath
BsmtHalfBath
BsmtFullBath



In [34]:

```
#detecting correlations between two feature which have correlation score above 0.5

correlations=houses.corr()
attrs = correlations # all except target

threshold = 0.5
important_corrs = (attrs[abs(attrs) > threshold][attrs != 1.0]).unstack().dropna().to_di

unique_important_corrs = pd.DataFrame(
    list(set([(tuple(sorted(key)), important_corrs[key])
    for key in important_corrs])),
        columns=['Attribute Pair', 'Correlation'])

    # sorted by absolute value
unique_important_corrs = unique_important_corrs.loc[
    abs(unique_important_corrs['Correlation']).argsort()[::-1]]

unique_important_corrs
```

Out[34]:

	Attribute Pair	Correlation
42	(Utilities_AllPub, Utilities_NoSeWa)	-1.000000
55	(Binary_PoolQC, PoolArea)	0.989665
98	(SaleCondition_Partial, SaleType_New)	0.986819
89	(Exterior1st_VinylSd, Exterior2nd_VinylSd)	0.977525
103	(Exterior1st_CemntBd, Exterior2nd_CmentBd)	0.974171
...	...	...
57	(GarageCars, Ordinal_KitchenQual)	0.509810
73	(SalePrice, YearRemodAdd)	0.507101
124	(Ordinal_HeatingQC, Ordinal_KitchenQual)	0.504228
125	(GarageYrBlt, Ordinal_KitchenQual)	0.503115
97	(2ndFlrSF, BedroomAbvGr)	0.502901

148 rows × 2 columns

In [35]:

```
#see the corelations of OverallQual and SalePrice
houses[['OverallQual','SalePrice']].groupby(['OverallQual'],
as_index=False).mean().sort_values(by='OverallQual', ascending=False)
```

Out[35]:

	OverallQual	SalePrice
9	10	438588.388889

8	9	367513.023256
7	8	274735.535714
6	7	207716.423197
5	6	161603.034759
4	5	133523.347607
3	4	108420.655172
2	3	87473.750000
1	2	51770.333333
0	1	50150.000000

In [36]:
```
#see the corelations of TotRmsAbvGrd and SalePrice
houses[['TotRmsAbvGrd','SalePrice']].groupby(['TotRmsAbvGrd'],
as_index=False).mean().sort_values(by='TotRmsAbvGrd', ascending=False)
```

Out[36]:

	TotRmsAbvGrd	SalePrice
11	14	200000.000000
10	12	280971.454545
9	11	318022.000000
8	10	296279.170213
7	9	252988.173333
6	8	213427.529412
5	7	196666.784195
4	6	161303.296020
3	5	141550.749091
2	4	122844.628866
1	3	111217.647059
0	2	39300.000000

In [37]:
```
#see the corelations of GarageCars and SalePrice
houses[['GarageCars','SalePrice']].groupby(['GarageCars'],
as_index=False).mean().sort_values(by='GarageCars', ascending=False)
```

Out[37]:

	GarageCars	SalePrice
4	4	192655.800000
3	3	309636.121547
2	2	183851.663835
1	1	128116.688347
0	0	103317.283951

In [38]:
```
# Normalize the data.
houses_min = houses.min()
houses_max = houses.max()
normalized_houses=(houses-houses.min())/(houses.max()-houses.min())
```

In [39]:
```
y = normalized_houses['SalePrice']
X = normalized_houses.drop("SalePrice", axis = 1)
train_X, test_X, train_y, test_y = train_test_split(X, y,
                    test_size = 0.3, random_state = 123)
```

In [40]:
```
#import packages for XGBoost model
import xgboost as xgb
```

In [41]:
```
#read the test data
#houses_test = pd.read_csv('./Data/test.csv')
```

In [42]:
```
#see the NaNs of test data
test_X.isnull().sum()
```

Out[42]:
```
MSSubClass              0
LotFrontage             0
LotArea                 0
OverallQual             0
OverallCond             0
                       ..
SaleCondition_AdjLand   0
SaleCondition_Alloca    0
SaleCondition_Family    0
SaleCondition_Normal    0
SaleCondition_Partial   0
Length: 251, dtype: int64
```

In [43]:
```
#creation of XGBoost model

model_xgb = xgb.XGBRegressor(objective = 'reg:squarederror', colsample_bytree = 0.3, lea
                            max_depth = 7, alpha = 5)
```

In [44]:
```
#fit the model and get prediction of model

model_xgb.fit(train_X, train_y)
preds_xgb = model_xgb.predict(test_X)
```

In [45]:
```
#see the predictions

preds_xgb
```

Out[45]:
```
array([0.25435543, 0.15370163, 0.18027733, 0.23362042, 0.13522966,
       0.3293924 , 0.3656502 , 0.15747403, 0.13513623, 0.15535003,
       0.15323596, 0.28300956, 0.16421331, 0.11781525, 0.25752452,
       0.20956232, 0.14809276, 0.4031963 , 0.26569512, 0.17573474,
       0.13300744, 0.20649043, 0.11202151, 0.15770926, 0.24060579,
       0.15207607, 0.21682484, 0.19619176, 0.11771087, 0.1360768 ,
       0.13450995, 0.22644556, 0.1436524 , 0.24420486, 0.48533168,
       0.21811263, 0.1620326 , 0.39908394, 0.19989333, 0.11166009,
       0.14430329, 0.24984547, 0.17483415, 0.20267752, 0.23881766,
       0.1120639 , 0.34746602, 0.31940833, 0.30325934, 0.11758024,
       0.22084855, 0.21746817, 0.15157646, 0.16487694, 0.11792441,
       0.15712404, 0.14510784, 0.15775107, 0.26901478, 0.4567657 ,
       0.1169848 , 0.3896869 , 0.19474229, 0.19049819, 0.22180913,
       0.12536818, 0.13260156, 0.25206017, 0.2596286 , 0.11799653,
       0.15452495, 0.20749056, 0.18008047, 0.3205597 , 0.19487551,
       0.13916421, 0.18719076, 0.14873461, 0.24026941, 0.17593619,
       0.37324232, 0.39214066, 0.11166009, 0.13722652, 0.25742194,
       0.12960711, 0.11540774, 0.12753496, 0.16231246, 0.29419565,
       0.13845575, 0.15839872, 0.18040302, 0.19918603, 0.14823003,
       0.12372634, 0.2669165 , 0.12235327, 0.27697396, 0.14061294,
       0.12273452, 0.47443306, 0.38298807, 0.16624881, 0.20489137,
       0.12565635, 0.13639951, 0.24365513, 0.19576591, 0.3974021 ,
       0.3241824 , 0.3299017 , 0.13615753, 0.3544948 , 0.20692264,
```

```
0.15024085, 0.1793143 , 0.41237187, 0.12118984, 0.16743918,
0.17511594, 0.2096722 , 0.23744957, 0.11179847, 0.31600133,
0.29909837, 0.15031934, 0.12030908, 0.21274096, 0.3854836 ,
0.14527051, 0.43667915, 0.3191888 , 0.18005921, 0.2323281 ,
0.18705052, 0.42476004, 0.12660737, 0.2535358 , 0.170651  ,
0.11710145, 0.25466928, 0.26450884, 0.14128567, 0.19822063,
0.27515823, 0.25228733, 0.15141834, 0.1858721 , 0.1351954 ,
0.21405792, 0.11975083, 0.4560937 , 0.19876708, 0.39594948,
0.15612134, 0.286362  , 0.13178435, 0.12418719, 0.1480183 ,
0.49151996, 0.25007984, 0.35577476, 0.12920512, 0.27503967,
0.11663721, 0.16439788, 0.11733755, 0.17352816, 0.113392  ,
0.13453381, 0.12937713, 0.23064487, 0.2592768 , 0.13457614,
0.36384645, 0.17336835, 0.12731011, 0.20028731, 0.25957012,
0.25283545, 0.15651733, 0.1460901 , 0.13466798, 0.17199902,
0.15445675, 0.13657741, 0.2358073 , 0.15833053, 0.24456878,
0.12227467, 0.13455291, 0.22174062, 0.2867214 , 0.13793792,
0.38337013, 0.12858853, 0.1126103 , 0.17423025, 0.13896024,
0.22057594, 0.41039896, 0.16909365, 0.37946078, 0.18160088,
0.13036568, 0.12481364, 0.3312688 , 0.16581625, 0.26198068,
0.18159023, 0.46014372, 0.13434818, 0.16837464, 0.14637055,
0.26774824, 0.20517851, 0.12880355, 0.12740843, 0.12342291,
0.18734723, 0.11612679, 0.17220667, 0.12693888, 0.12206154,
0.12372623, 0.18431406, 0.11055627, 0.13362995, 0.12863362,
0.12500048, 0.17278834, 0.1737294 , 0.12339629, 0.1175725 ,
0.18355496, 0.17637862, 0.12792827, 0.20535049, 0.11724872,
0.3154787 , 0.28838268, 0.21996346, 0.23230316, 0.12524353,
0.29707196, 0.17671348, 0.34394428, 0.13658124, 0.3243722 ,
0.250401  , 0.11739265, 0.22465354, 0.12648088, 0.18451512,
0.28976622, 0.3927311 , 0.24663766, 0.13579771, 0.1128139 ,
0.22797878, 0.21843053, 0.1776974 , 0.2044429 , 0.15995693,
0.15580568, 0.2807346 , 0.22853091, 0.11055627, 0.11147856,
0.1433973 , 0.11175127, 0.22697754, 0.11055627, 0.36015806,
0.17005706, 0.23854247, 0.24003197, 0.15207076, 0.11093754,
0.14017156, 0.13594282, 0.11100906, 0.13988051, 0.18808222,
0.12501286, 0.16307285, 0.15006036, 0.18714555, 0.12680884,
0.20717765, 0.11538392, 0.4032841 , 0.16266167, 0.28094485,
0.17462915, 0.17725413, 0.27324718, 0.13741827, 0.1351876 ,
0.42024636, 0.14595741, 0.26935473, 0.36064965, 0.42133167,
0.15744694, 0.3965404 , 0.24139209, 0.12533145, 0.11832815,
0.13727134, 0.1817058 , 0.13485864, 0.3389434 , 0.1550833 ,
0.29078344, 0.32390207, 0.12381407, 0.17505983, 0.16345482,
0.337972  , 0.18558523, 0.15607971, 0.21847351, 0.28998503,
0.17542784, 0.11805048, 0.18043597, 0.2247827 , 0.23036236,
0.14461888, 0.13750385, 0.22088344, 0.30679014, 0.128194  ,
0.3592595 , 0.31612858, 0.3182761 , 0.16012639, 0.13899101,
0.12631871, 0.47215247, 0.21883772, 0.13497572, 0.3617529 ,
0.11606141, 0.11708055, 0.15472756, 0.22143565, 0.11907743,
0.16992852, 0.19519784, 0.33591747, 0.172961  , 0.29051846,
0.16857252, 0.2056919 , 0.17320846, 0.12886116, 0.13101654,
0.13829125, 0.17718808, 0.17208758, 0.19007812, 0.14795542,
0.15034772, 0.12020964, 0.1923163 , 0.16673474, 0.2350452 ,
0.2538313 , 0.19587734, 0.11676142, 0.12197445, 0.14858751,
0.25862983, 0.13340017, 0.28902856, 0.22417144, 0.1988563 ,
0.16760322, 0.19537207, 0.21108313, 0.11055627, 0.17188528,
0.13565771, 0.26191095, 0.1120639 , 0.14735556, 0.11068544,
0.20632827, 0.19546005, 0.11102442, 0.18246567, 0.19030742,
0.18256873, 0.2667211 , 0.3097041 , 0.3463387 , 0.12441024,
0.17694508, 0.2775748 , 0.26326737, 0.15332516, 0.16021839,
0.1147529 , 0.23982832, 0.35558966, 0.47168958, 0.23197223,
0.15304866, 0.27034417, 0.18032555, 0.13215871, 0.40238455,
0.21739654, 0.14121065, 0.1260677 , 0.15175392, 0.13864271,
0.25394568, 0.24785395, 0.11665007, 0.21190825, 0.17288761,
0.24890417, 0.311241  , 0.36493552, 0.14128163, 0.12258954,
0.33057624, 0.2154137 , 0.17970864, 0.13972652, 0.12129143,
0.12358586, 0.34382394, 0.32480222], dtype=float32)
```

In [46]:
```
rmse = np.sqrt(MSE(test_y, preds_xgb))
print("RMSE : % f" %(rmse))
```

```
RMSE :  0.047111
```

In [47]:
```
#import packages for Support Vector Machine model

from sklearn.svm import SVR

#creation of SVM model

model_svm = SVR(kernel = 'poly')

#fit the SVM model

model_svm.fit(train_X, train_y.values.ravel())

#get and see the prediction of the SVM model

preds_svm=model_svm.predict(test_X)

# Calculate RMSE
rmse = np.sqrt(MSE(test_y, preds_svm))
print("RMSE : % f" %(rmse))
```

```
RMSE :  0.057991
```

In [48]:
```
#import packages for Random Forest model

from sklearn.ensemble import RandomForestRegressor as rfg

#creation of model for Random Forest model

model_rfg = rfg(n_estimators = 1000, random_state = 20 )

#fit the model and get predictions

model_rfg.fit(train_X, train_y.values.ravel())
preds_rfg = model_rfg.predict(test_X)

# Calculate RMSE
rmse = np.sqrt(MSE(test_y, preds_rfg))
print("RMSE : % f" %(rmse))
```

```
RMSE :  0.036052
```

In [49]:
```
# Gradient Boosting Regressor
from sklearn.ensemble import GradientBoostingRegressor

#Creation of model
model_gbr = GradientBoostingRegressor(n_estimators=20,
                                      learning_rate=0.1,
                                      max_depth=1)

#fit the model and get predictions

model_gbr.fit(train_X, train_y.values.ravel())
preds_gbr = model_gbr.predict(test_X)

# Calculate RMSE
rmse = np.sqrt(MSE(test_y, preds_gbr))
print("RMSE : % f" %(rmse))
```

```
RMSE :  0.061730
```

We now have four models with varying RMSEs before combining them in an ensemble model lets see if we can imrpove their RMSE's with PCA.

In [50]:
```
from sklearn.decomposition import PCA
pca=PCA()
PCA_train_X = pca.fit_transform(train_X)
```

In [51]:
```
np.cumsum(np.round(pca.explained_variance_ratio_, decimals = 4)*100)[0:]
```

Out[51]:
```
array([ 12.89,  19.33,  23.6 ,  27.55,  30.89,  34.05,  36.75,  39.21,
        41.6 ,  43.91,  46.09,  48.23,  50.24,  52.08,  53.84,  55.52,
        57.12,  58.61,  60.02,  61.33,  62.6 ,  63.82,  65.  ,  66.07,
        67.11,  68.11,  69.07,  69.99,  70.86,  71.71,  72.54,  73.34,
        74.1 ,  74.83,  75.55,  76.22,  76.87,  77.5 ,  78.11,  78.7 ,
        79.26,  79.81,  80.35,  80.87,  81.38,  81.86,  82.33,  82.79,
        83.24,  83.67,  84.1 ,  84.52,  84.92,  85.31,  85.68,  86.05,
        86.41,  86.76,  87.1 ,  87.43,  87.75,  88.06,  88.37,  88.67,
        88.95,  89.23,  89.51,  89.78,  90.04,  90.3 ,  90.56,  90.81,
        91.05,  91.29,  91.52,  91.75,  91.97,  92.19,  92.4 ,  92.6 ,
        92.8 ,  92.99,  93.18,  93.36,  93.54,  93.71,  93.88,  94.04,
        94.2 ,  94.36,  94.51,  94.65,  94.79,  94.93,  95.07,  95.2 ,
        95.33,  95.46,  95.58,  95.7 ,  95.82,  95.93,  96.04,  96.15,
        96.26,  96.36,  96.46,  96.56,  96.66,  96.76,  96.85,  96.94,
        97.03,  97.12,  97.21,  97.29,  97.37,  97.45,  97.53,  97.6 ,
        97.67,  97.74,  97.81,  97.88,  97.95,  98.01,  98.07,  98.13,
        98.19,  98.25,  98.31,  98.36,  98.41,  98.46,  98.51,  98.56,
        98.61,  98.66,  98.71,  98.76,  98.8 ,  98.84,  98.88,  98.92,
        98.96,  99.  ,  99.04,  99.07,  99.1 ,  99.13,  99.16,  99.19,
        99.22,  99.25,  99.28,  99.31,  99.34,  99.37,  99.4 ,  99.43,
        99.45,  99.47,  99.49,  99.51,  99.53,  99.55,  99.57,  99.59,
        99.61,  99.63,  99.65,  99.67,  99.69,  99.71,  99.73,  99.74,
        99.75,  99.76,  99.77,  99.78,  99.79,  99.8 ,  99.81,  99.82,
        99.83,  99.84,  99.85,  99.86,  99.87,  99.88,  99.89,  99.9 ,
        99.91,  99.92,  99.93,  99.94,  99.95,  99.96,  99.97,  99.98,
        99.99, 100.  , 100.  , 100.  , 100.  , 100.  , 100.  , 100.  ,
       100.  , 100.  , 100.  , 100.  , 100.  , 100.  , 100.  , 100.  ,
       100.  , 100.  , 100.  , 100.  , 100.  , 100.  , 100.  , 100.  ,
       100.  , 100.  , 100.  , 100.  , 100.  , 100.  , 100.  , 100.  ,
       100.  , 100.  , 100.  , 100.  , 100.  , 100.  , 100.  , 100.  ,
       100.  , 100.  , 100.  , 100.  , 100.  , 100.  , 100.  , 100.  ,
       100.  , 100.  , 100.  ])
```

It seems like we can capture all of the variety in the data with 201 features, this is not a small number but eliminating 50 features without a cost could increase our performance.

In [52]:
```
PCA_test_X = pca.fit_transform(test_X)
```

In [53]:
```
#fit the XG Boost model and see RMSE of model

model_xgb.fit(PCA_train_X[:,:201], train_y)
PCA_preds_xgb = model_xgb.predict(PCA_test_X[:,:201])

rmse = np.sqrt(MSE(test_y, PCA_preds_xgb))
print("RMSE : % f" %(rmse))
```

```
RMSE :  0.090426
```

In [54]:
```
#fit the SVM model

model_svm.fit(PCA_train_X[:,:201], train_y.values.ravel())

#get and see the prediction of the SVM model

PCA_preds_svm=model_svm.predict(PCA_test_X[:,:201])
```

```python
# Calculate RMSE
rmse = np.sqrt(MSE(test_y, PCA_preds_svm))
print("RMSE : % f" %(rmse))
```

RMSE : 0.094988

In [55]:
```python
#fit the Random Forest  and get predictions

model_rfg.fit(PCA_train_X[:,:201], train_y.values.ravel())
PCA_preds_rfg = model_rfg.predict(PCA_test_X[:,:201])

# Calculate RMSE
rmse = np.sqrt(MSE(test_y, PCA_preds_rfg))
print("RMSE : % f" %(rmse))
```

RMSE : 0.088877

It seems like PCA did not improve out performance, lets try eliminating useless features.

In [56]:
```python
from sklearn.feature_selection import VarianceThreshold
# Remove features with 0 variance.
constant_filter = VarianceThreshold(threshold=0)
constant_filter.fit(train_X)
len(train_X.columns[constant_filter.get_support()])
```

Out[56]: 246

In [57]:
```python
# Remove duplicate features
train_features_T = train_X.T
train_features_T.shape
# Find duplicate features
print(train_features_T.duplicated().sum())
```

6

In [58]:
```python
unique_train_X = train_features_T.drop_duplicates(keep='first').T
unique_train_X.shape
```

Out[58]: (1022, 245)

In [59]:
```python
correlated_features = set()
correlation_matrix = unique_train_X.corr()
correlation_matrix
```

Out[59]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRe
MSSubClass	1.000000	-0.347414	-0.128234	0.012387	-0.018605	0.015858	0
LotFrontage	-0.347414	1.000000	0.351524	0.258526	-0.056331	0.128215	C
LotArea	-0.128234	0.351524	1.000000	0.119127	-0.012136	0.026058	-0
OverallQual	0.012387	0.258526	0.119127	1.000000	-0.060721	0.578470	0
OverallCond	-0.018605	-0.056331	-0.012136	-0.060721	1.000000	-0.357318	(
...	...	...	...	...	...	...	
SaleCondition_AdjLand	0.007941	-0.021228	-0.015356	-0.054402	-0.043749	-0.062392	-0
SaleCondition_Alloca	0.027022	-0.013142	-0.002371	-0.037024	-0.006776	-0.017657	-0
SaleCondition_Family	0.024557	-0.027090	-0.017278	-0.013508	-0.016273	-0.034571	-0
SaleCondition_Normal	0.016932	-0.088886	-0.005082	-0.139858	0.163575	-0.162496	-C
SaleCondition_Partial	-0.064120	0.159939	0.040633	0.322663	-0.147666	0.353168	C

245 rows × 245 columns

In [60]:
```
# Find hihgly correlated features
for i in range(len(correlation_matrix .columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > 0.9:
            colname = correlation_matrix.columns[i]
            correlated_features.add(colname)
```

In [61]:
```
len(correlated_features)
```

Out[61]:
```
10
```

In [62]:
```
#Remove highly correlated features
unique_train_X.drop(labels=correlated_features, axis=1, inplace=True)
```

In [63]:
```
# Lets create remove all those from the test dataframe as well.
removed_features = train_X.drop(unique_train_X.columns, axis = 1)
unique_test_X = test_X.drop(removed_features, axis = 1)
```

In [64]:
```
#import packages for XAI method lime and define the explainer for training methods

from lime import lime_tabular
explainer = lime_tabular.LimeTabularExplainer(training_data = np.array(train_X), mode =
                                              feature_names = train_X.columns, categorica
```

In [65]:
```
#fit the XG Boost model and see RMSE of model

model_xgb.fit(unique_train_X, train_y)
UQ_preds_xgb = model_xgb.predict(unique_test_X)

rmse = np.sqrt(MSE(test_y, UQ_preds_xgb))
print("RMSE : % f" %(rmse))
```

```
RMSE :  0.046627
```

In [66]:
```
#fit the SVM model

model_svm.fit(unique_train_X, train_y.values.ravel())

#get and see the prediction of the SVM model

UQ_preds_svm=model_svm.predict(unique_test_X)

# Calculate RMSE
rmse = np.sqrt(MSE(test_y, UQ_preds_svm))
print("RMSE : % f" %(rmse))
```

```
RMSE :  0.057038
```

In [67]:
```
#fit the Random Forest  and get predictions

model_rfg.fit(unique_train_X, train_y.values.ravel())
UQ_preds_rfg = model_rfg.predict(unique_test_X)

# Calculate RMSE
rmse = np.sqrt(MSE(test_y, UQ_preds_rfg))
print("RMSE : % f" %(rmse))
```

```
RMSE :  0.036035
```

In [68]:
```
#fit the Gradiant Boosting Regressor model and get predictions
```

```
model_gbr.fit(unique_train_X, train_y.values.ravel())
UQ_preds_gbr = model_gbr.predict(unique_test_X)

# Calculate RMSE
rmse = np.sqrt(MSE(test_y, UQ_preds_gbr))
print("RMSE : % f" %(rmse))
```

```
RMSE :  0.061730
```

All models slightly benefitted from this approach. Lets create an ensemble model.

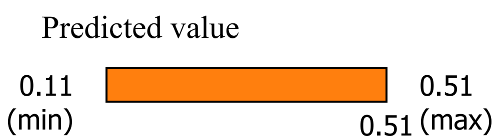
In [69]:
```
pred_final = (preds_xgb+(2*UQ_preds_rfg)+UQ_preds_gbr+UQ_preds_svm)/5.0
rmse = np.sqrt(MSE(test_y, pred_final))
print("RMSE : % f" %(rmse))
```

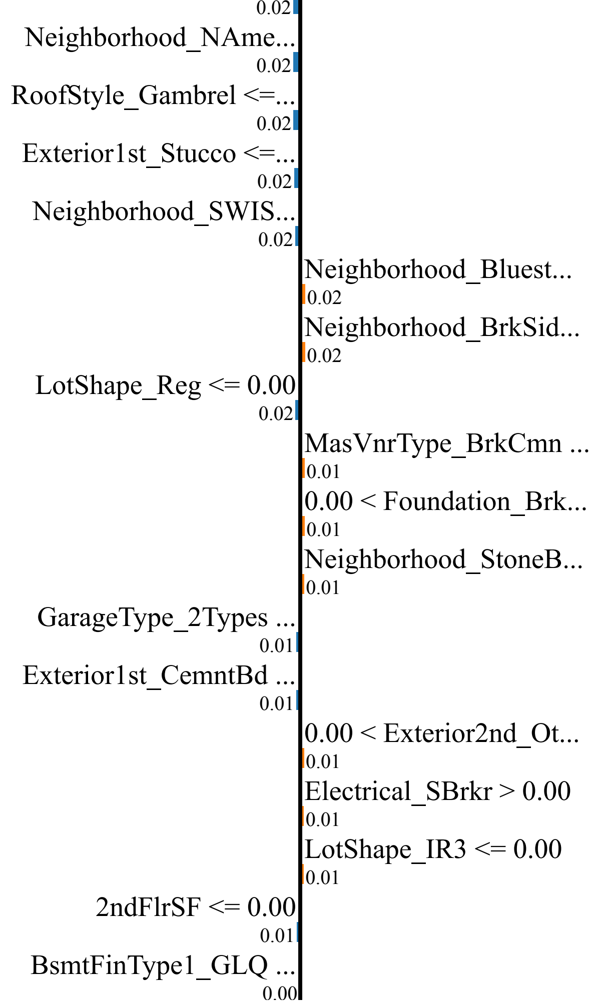
```
RMSE :  0.040434
```

In [70]:
```
# define explainer

explainer_1 = lime_tabular.LimeTabularExplainer(training_data = np.array(unique_train_X)
                                                feature_names = train_X.columns, categorica
```

In [71]:
```
exp = explainer_1.explain_instance(data_row = unique_test_X.iloc[201],
                                   predict_fn = model_svm.predict, num_features = 20)
exp.show_in_notebook(show_table = True)
```

```
/Users/bakolas/opt/anaconda3/envs/geopandas/lib/python3.7/site-packages/sklearn/base.py:
451: UserWarning: X does not have valid feature names, but SVR was fitted with feature n
ames
  "X does not have valid feature names, but"
```



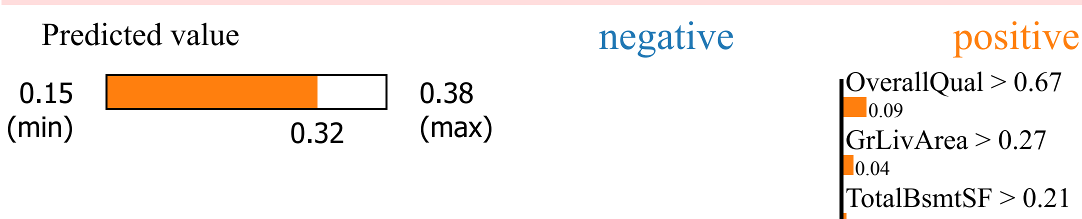


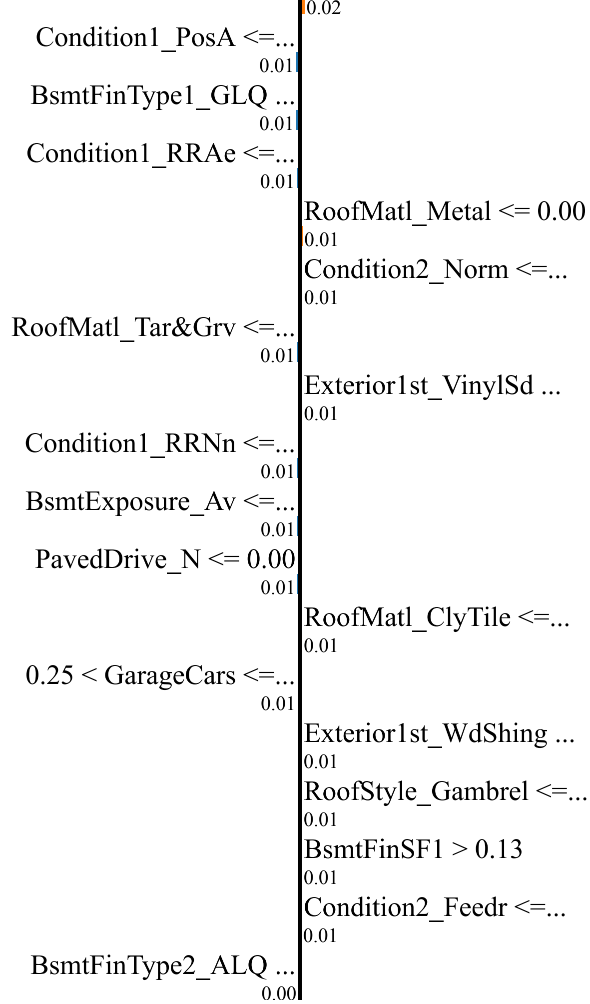
Feature	Value
Neighborhood_Mitchel	1.00
Exterior2nd_Wd Sdng	1.00
RoofMatl_CompShg	0.00
Neighborhood_NAmes	0.00
RoofStyle_Gambrel	0.00
Exterior1st_Stucco	0.00
Neighborhood_SWISU	0.00
Neighborhood_Blueste	0.00
Neighborhood_BrkSide	0.00
LotShape_Reg	0.00

In [72]:

```
exp = explainer_1.explain_instance(data_row = unique_test_X.iloc[201],
                                   predict_fn = model_gbr.predict, num_features = 20)
exp.show_in_notebook(show_table = True)
```

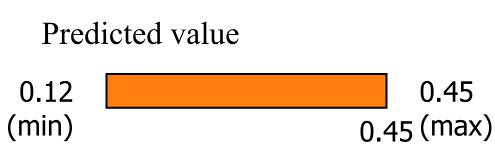
```
/Users/bakolas/opt/anaconda3/envs/geopandas/lib/python3.7/site-packages/sklearn/base.py:
451: UserWarning: X does not have valid feature names, but GradientBoostingRegressor was
fitted with feature names
  "X does not have valid feature names, but"
```

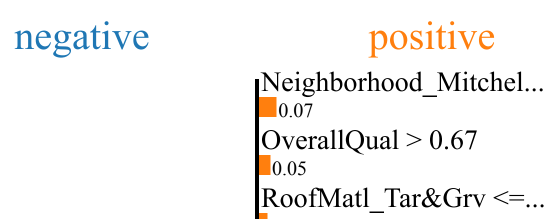


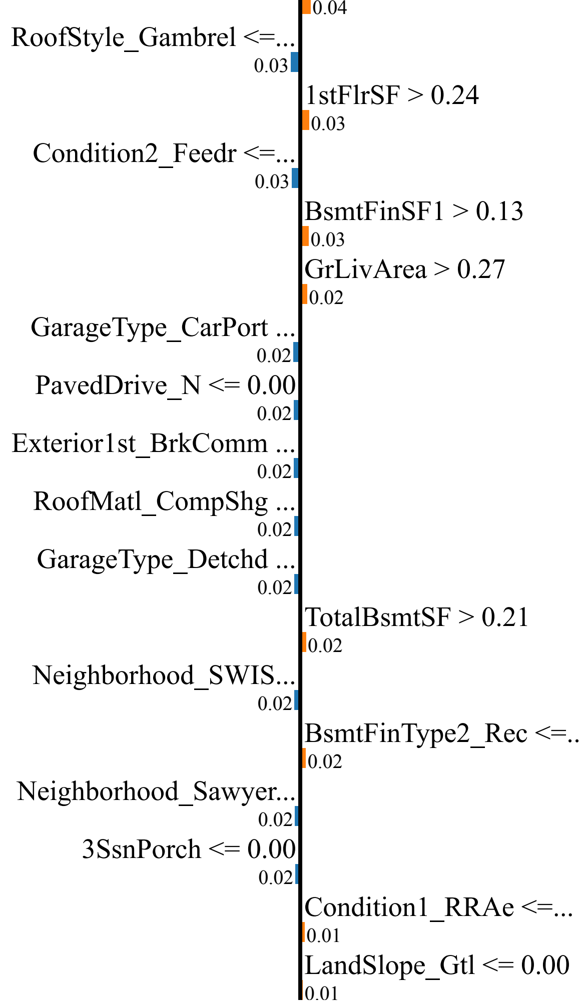


Feature	Value
OverallQual	0.78
GrLivArea	0.31
TotalBsmtSF	0.33
Condition1_PosA	0.00
BsmtFinType1_GLQ	0.00
Condition1_RRAe	0.00
RoofMatl_Metal	0.00
Condition2_Norm	0.00
RoofMatl_Tar&Grv	0.00
Exterior1st_VinylSd	0.00

In [73]:
```
exp = explainer_1.explain_instance(data_row = unique_test_X.iloc[201],
                                   predict_fn = model_xgb.predict, num_features = 20)
exp.show_in_notebook(show_table = True)
```



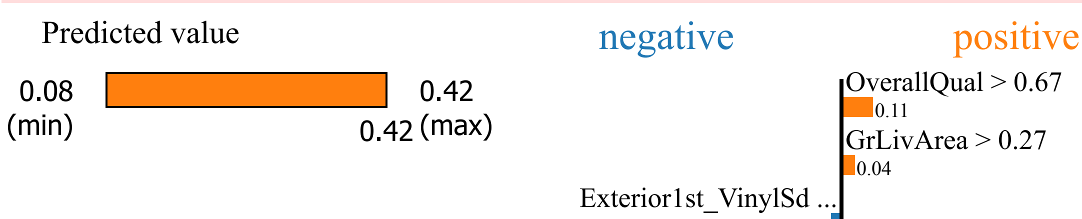


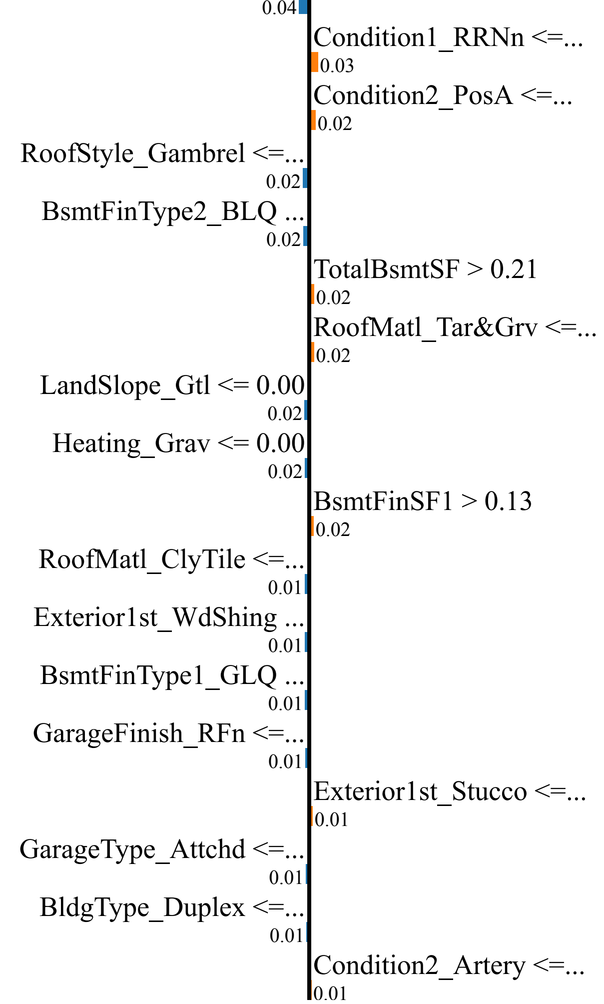


Feature	Value
Neighborhood_Mitchel	1.00
OverallQual	0.78
RoofMatl_Tar&Grv	0.00
RoofStyle_Gambrel	0.00
1stFlrSF	0.38
Condition2_Feedr	0.00
BsmtFinSF1	0.23
GrLivArea	0.31
GarageType_CarPort	0.00
PavedDrive_N	0.00

In [74]:
```
exp = explainer_1.explain_instance(data_row = unique_test_X.iloc[201],
                                   predict_fn = model_rfg.predict, num_features = 20)
exp.show_in_notebook(show_table = True)
```

```
/Users/bakolas/opt/anaconda3/envs/geopandas/lib/python3.7/site-packages/sklearn/base.py:
451: UserWarning: X does not have valid feature names, but RandomForestRegressor was fit
ted with feature names
  "X does not have valid feature names, but"
```





Feature	Value
OverallQual	0.78
GrLivArea	0.31
Exterior1st_VinylSd	0.00
Condition1_RRNn	0.00
Condition2_PosA	0.00
RoofStyle_Gambrel	0.00
BsmtFinType2_BLQ	0.00
TotalBsmtSF	0.33
RoofMatl_Tar&Grv	0.00
LandSlope_Gtl	0.00

In [131…

```
# Last test with better hyper parameters.
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
model_xgb = xgb.XGBRegressor(n_estimators=268,
                             learning_rate=0.05,
                             max_depth = 5,
                             min_child_weight=1,
                             subsample=1.0,
                             colsample_bytree=1.0)
model_svm = make_pipeline(RobustScaler(), SVR(C= 20, epsilon= 0.008, gamma=0.0003))
model_rfg = rfg(n_estimators=1000,
                          max_depth=50)
model_gbr = GradientBoostingRegressor(n_estimators=6000,
                                      learning_rate=0.01,
```

```
                                               max_depth=4,
                                               max_features='sqrt',
                                               min_samples_leaf=15,
                                               min_samples_split=10,
                                               loss='huber',
                                               random_state=42)
```

In [108…

```
#fit the XG Boost model and see RMSE of model

model_xgb.fit(unique_train_X, train_y)
UQ_preds_xgb = model_xgb.predict(unique_test_X)

rmse = np.sqrt(MSE(test_y, UQ_preds_xgb))
print("RMSE : % f" %(rmse))
```

```
RMSE :  0.034329
```

In [109…

```
model_svm.fit(unique_train_X, train_y.values.ravel())

#get and see the prediction of the SVM model

UQ_preds_svm=model_svm.predict(unique_test_X)

# Calculate RMSE
rmse = np.sqrt(MSE(test_y, UQ_preds_svm))
print("RMSE : % f" %(rmse))
```

```
RMSE :  0.033565
```

In [132…

```
#fit the Random Forest  and get predictions

model_rfg.fit(unique_train_X, train_y.values.ravel())
UQ_preds_rfg = model_rfg.predict(unique_test_X)

# Calculate RMSE
rmse = np.sqrt(MSE(test_y, UQ_preds_rfg))
print("RMSE : % f" %(rmse))
```

```
RMSE :  0.035956
```

In [87]:

```
#fit the Gradiant Boosting Regressor model and get predictions

model_gbr.fit(unique_train_X, train_y.values.ravel())
UQ_preds_gbr = model_gbr.predict(unique_test_X)

# Calculate RMSE
rmse = np.sqrt(MSE(test_y, UQ_preds_gbr))
print("RMSE : % f" %(rmse))
```

```
RMSE :  0.032174
```

In [133…

```
pred_final = (preds_xgb+UQ_preds_rfg+UQ_preds_gbr+UQ_preds_svm)/4.0
rmse = np.sqrt(MSE(test_y, pred_final))
print("RMSE : % f" %(rmse))
```

```
RMSE :  0.033885
```

In [134…

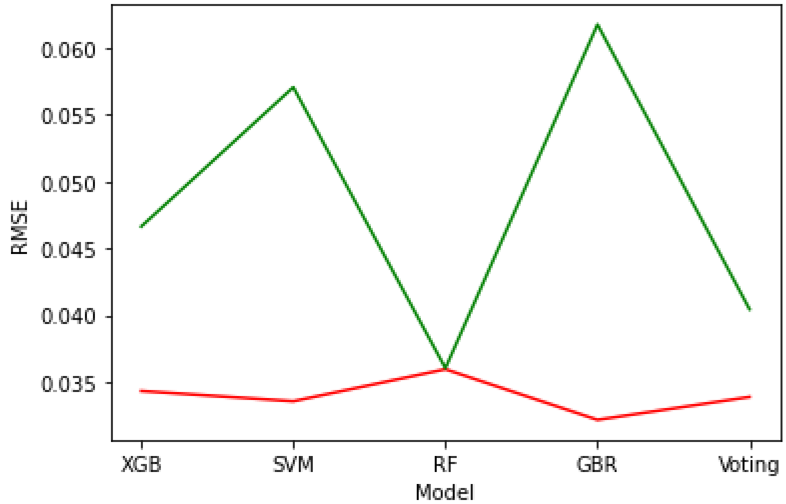
```
fresults = { "Model" : ["XGB","SVM","RF","GBR", "Voting"] , "RMSE" : [0.034329, 0.033565
results = { "Model" : ["XGB","SVM","RF","GBR", "Voting"] , "RMSE" : [0.046627, 0.057038,
```

In [135…

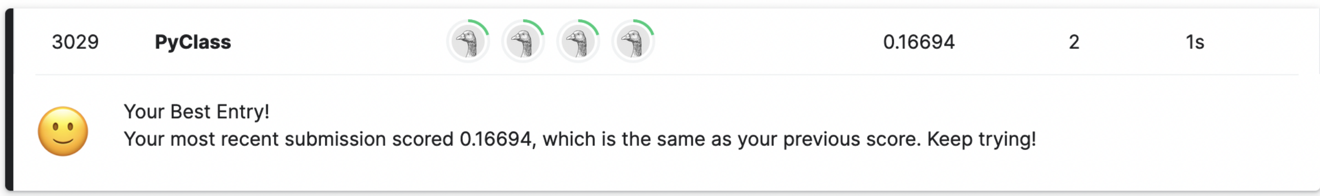
```
sns.lineplot(data=fresults, x='Model', y='RMSE', color="red")
sns.lineplot(data=results, x='Model', y='RMSE', color="green")
```

Out[135]:

```
<AxesSubplot:xlabel='Model', ylabel='RMSE'>
```



Our final best result is GBR with 0.033565 RMSE.



Not bad for our first competition submission.