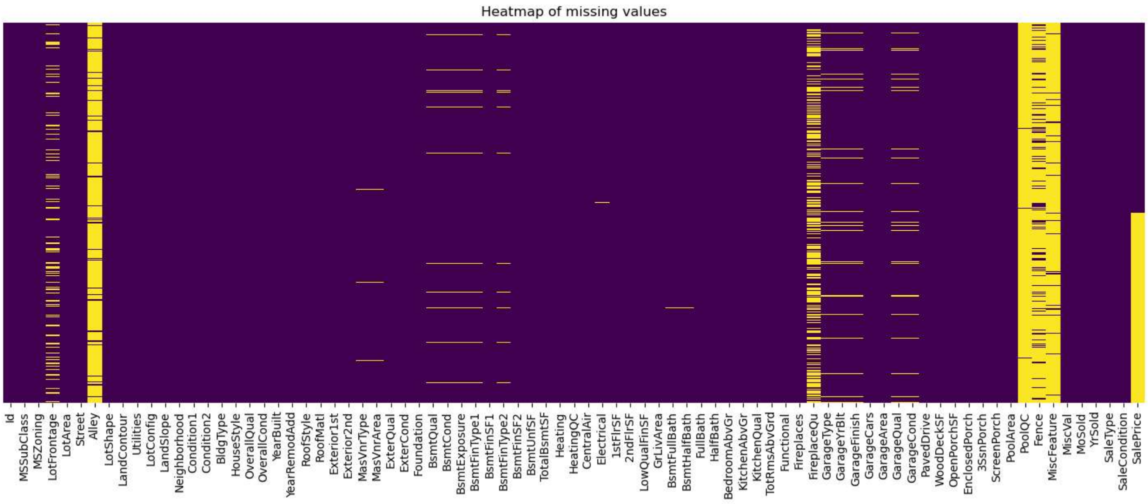
In [85]:
```python
#Loading required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.impute import SimpleImputer
from sklearn.ensemble import HistGradientBoostingRegressor
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import cross_val_score
import seaborn as sns
import statsmodels.api as sm

```

In [86]:
```python
train_data = pd.read_csv('train.csv')
test_data=pd.read_csv('test.csv')
data=pd.concat([train_data,test_data])
plt.figure(figsize=(18,6))
plt.title('Heatmap of missing values')
sns.heatmap(data.isnull(),yticklabels=False,cbar=False,cmap='viridis')

```

Out[86]:
```
<Axes: title={'center': 'Heatmap of missing values'}>
```



In [87]:

```
# separate numerical and categorical columns
numerical_cols = train_data.select_dtypes(include=['int64', 'float64'])
categorical_cols = train_data.select_dtypes(include=['object']).columns

# replace missing values with mean for numerical columns
imputer = SimpleImputer(strategy='mean')
train_data[numerical_cols] = imputer.fit_transform(train_data[numerical_

# replace missing values with mode for categorical columns
imputer = SimpleImputer(strategy='most_frequent')
train_data[categorical_cols] = imputer.fit_transform(train_data[categor

# verify that missing values are replaced
print(train_data.isnull().sum())



```

```
Id               0
MSSubClass       0
MSZoning         0
LotFrontage      0
LotArea          0
                ..
MoSold           0
YrSold           0
SaleType         0
SaleCondition    0
SalePrice        0
Length: 81, dtype: int64
```

In [88]:

```
# Select the predictor and target variable (for both training and testi
x_train = train_data[['LotArea', 'BedroomAbvGr', 'BsmtFullBath','BsmtHa
y_train = train_data['SalePrice']

x_test=test_data[['LotArea', 'BedroomAbvGr', 'BsmtFullBath','BsmtHalfBa
```

In [89]:

```
# separate numerical and categorical columns for test data
numerical_cols2 = x_test.select_dtypes(include=['int64', 'float64']).co

# replace missing values with mean for numerical columns
imputer = SimpleImputer(strategy='mean')
x_test[numerical_cols2] = imputer.fit_transform(x_test[numerical_cols2]

# verify that missing values are replaced
print(x_test.isnull().sum())
```

```
LotArea         0
BedroomAbvGr    0
BsmtFullBath    0
BsmtHalfBath    0
FullBath        0
HalfBath        0
dtype: int64

C:\Users\Lenovo\AppData\Local\Temp\ipykernel_8988\281113641.py:6: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  x_test[numerical_cols2] = imputer.fit_transform(x_test[numerical_cols2])
```

In [90]:

```
# Initializing a linear regression model and fit this model to train set
model = LinearRegression()
print(model)
model.fit(x_train, y_train)

```

```
LinearRegression()
```

Out[90]:

```
▾ LinearRegression
LinearRegression()
```

In [91]:

```
# Making predictions on the test set using the fitted model

y_pred = model.predict(x_test)
print(y_pred)

```

```
[112606.119798   145550.48551853 224695.03092123 ... 146582.41859758
 120117.45460539 219383.04414268]
```

In [92]:

```
#instead of simple linear regression we are using HistGradientBoosting w
#missing values are assigned to the left or right child accordingly1. Th
```

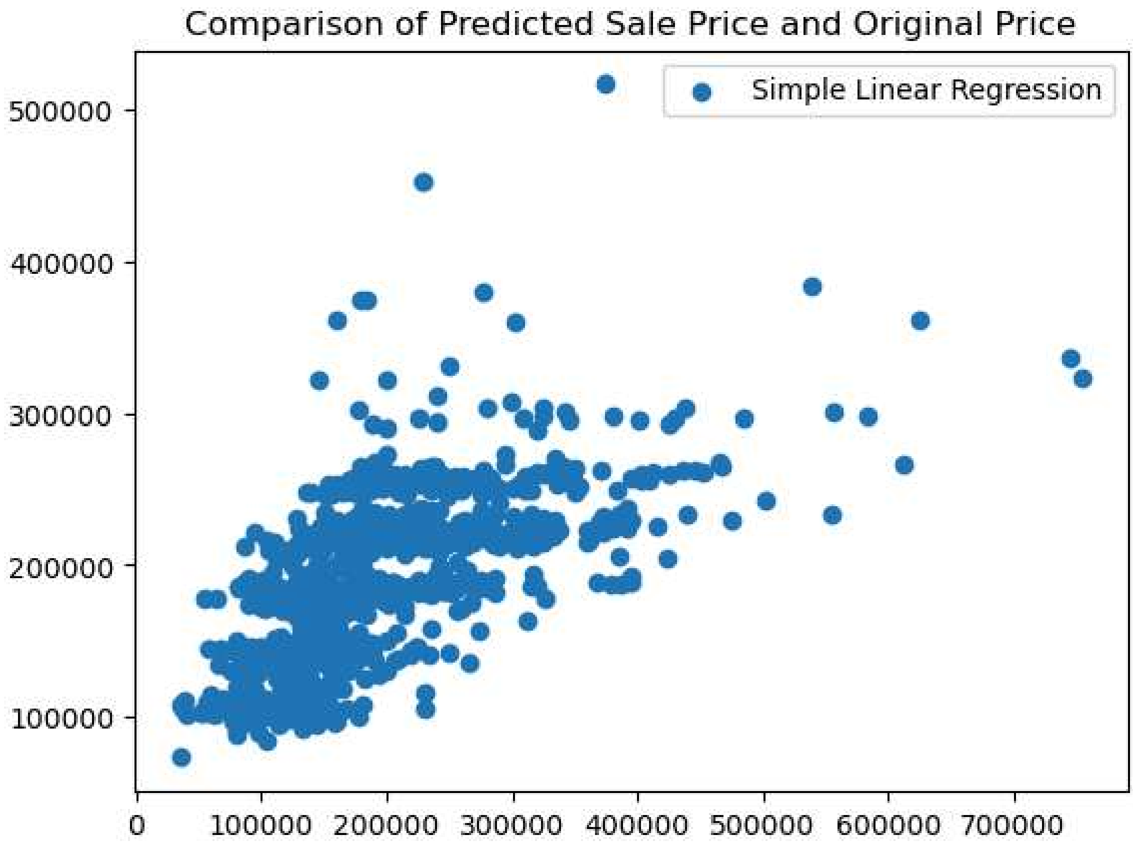
In [81]:

```
train_data = pd.read_csv('train.csv')
test_data=pd.read_csv('test.csv')

x_train = train_data[['LotArea', 'BedroomAbvGr', 'BsmtFullBath','BsmtHa
y_train = train_data['SalePrice']

x_test=test_data[['LotArea', 'BedroomAbvGr', 'BsmtFullBath','BsmtHalfBat
# Initializing a linear regression model and fit this model to train set
histgrad_model =  HistGradientBoostingRegressor().fit(x_train, y_train)

# Making predictions on the test set using the fitted model
y_pred = histgrad_model.predict(x_test)
print("predicted saleprices;",y_pred)

```

```
predicted saleprices; [129151.88642225 178516.40056678 253716.44635059 ...
199879.47712159
 137533.8702215  208269.17549794]
```

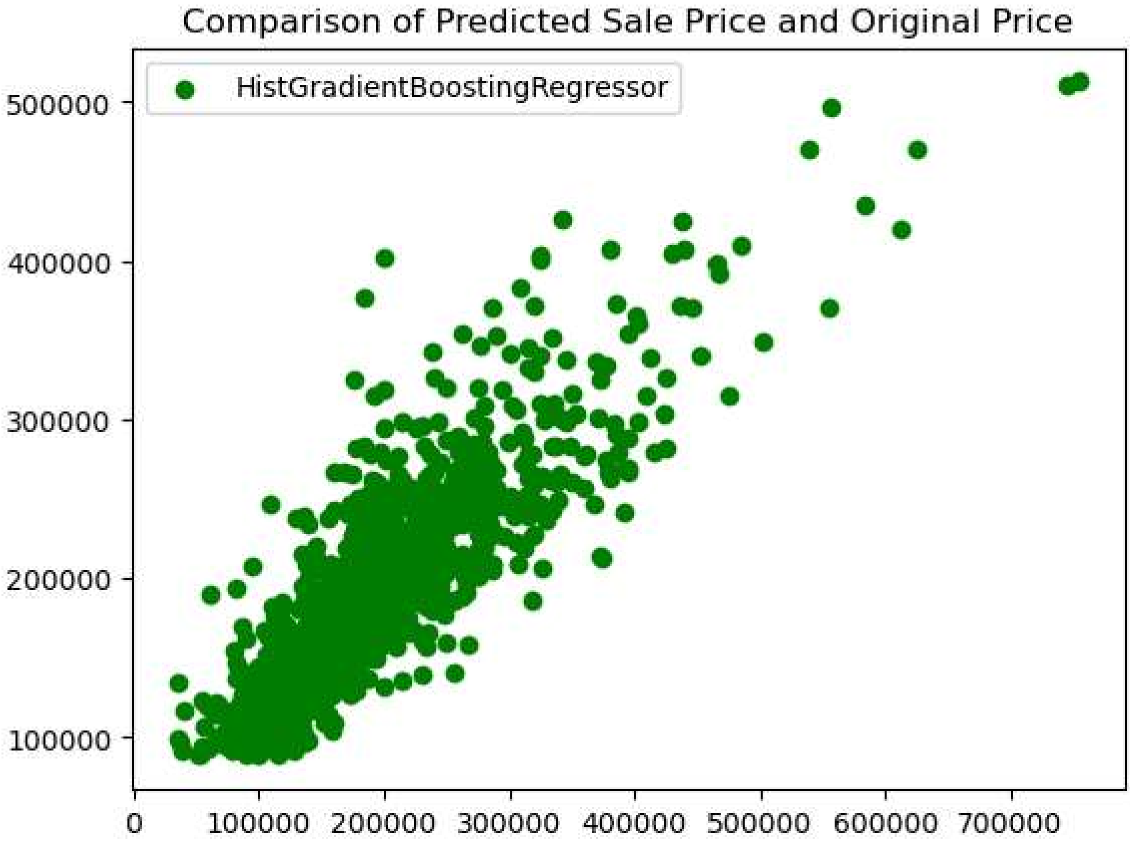
In [82]:

```
plt.figure()
plt.title('Comparison of Predicted Sale Price and Original Price')
plt.scatter(y_train, model.predict(x_train), label='Simple Linear Regres
plt.legend()
plt.show()

```



In [93]:

```
plt.figure()
plt.title('Comparison of Predicted Sale Price and Original Price')
plt.scatter(y_train, histgrad_model.predict(x_train), label='HistGradie
plt.legend()
plt.show()

```



In [96]:

```
#preparing the submission data
y_pred = model.predict(x_test)
sub_data=pd.DataFrame()
sub_data['Id']=test_data['Id']
sub_data['SalePrice']=y_pred
sub_data
sub_data.to_excel('submission.xlsx', index=False)

```

In [97]:

```
###LINEAR REGRESSION USING ORDINARY LEAST SQUARE METHOD

data = pd.read_csv('train.csv')

# Select the relevant features: square footage, number of bedrooms, and
X1 = data[['LotArea', 'BedroomAbvGr', 'BsmtFullBath','BsmtHalfBath','Ful
y1 = data['SalePrice']  # Target variable (house price)

# Add a constant term for the intercept in the regression model
X1 = sm.add_constant(X1)

# Fit the linear regression model
model = sm.OLS(y1, X1).fit()
model
# Print the summary of the model
print(model.summary())

```

```
                             OLS Regression Results
==============================================================================
===
Dep. Variable:              SalePrice   R-squared:                       0.
459
Model:                            OLS   Adj. R-squared:                  0.
457
Method:                 Least Squares   F-statistic:                     20
5.9
Date:                Tue, 04 Jun 2024   Prob (F-statistic):          4.28e-
190
Time:                        03:01:15   Log-Likelihood:                 -180
95.
No. Observations:                1460   AIC:                         3.620e
+04
Df Residuals:                    1453   BIC:                         3.624e
+04
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
=====
                  coef    std err          t      P>|t|      [0.025      
0.975]
------------------------------------------------------------------------------
-----
const         3.178e+04   6555.902      4.847      0.000    1.89e+04    4.4
6e+04
LotArea          1.2639      0.158      7.981      0.000       0.953       
1.575
BedroomAbvGr -6777.5551   2087.330     -3.247      0.001   -1.09e+04   -268
3.053
BsmtFullBath  3.694e+04   3074.859     12.014      0.000    3.09e+04     4.
3e+04
BsmtHalfBath  1.578e+04   6532.302      2.416      0.016    2967.754    2.8
6e+04
FullBath        7.97e+04   3014.745     26.436      0.000    7.38e+04    8.5
6e+04
HalfBath      3.638e+04   3134.939     11.604      0.000    3.02e+04    4.2
5e+04
==============================================================================
===
Omnibus:                      539.540   Durbin-Watson:                   1.
975
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              3595.
355
Skew:                           1.564   Prob(JB):                        
0.00
Kurtosis:                      10.023   Cond. No.                      6.70e
+04
==============================================================================
===

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is corr
ectly specified.
[2] The condition number is large, 6.7e+04. This might indicate that there 
are
strong multicollinearity or other numerical problems.
```

In [ ]:

```
1
```