Predicting House Prices with Linear Regression

Louis Bailey

introduction

- In this project the Ames, Iowa Housing Dataset is being used. This dataset is available on Kaggle. It includes 79 explanatory variables detailing nearly every facet of residential properties in Ames, Iowa. The goal is to predict housing prices by using Linear Regression.

- Contents:

libraries

In [1]:
```
import pandas as pd
import numpy as np
```

```
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, cross_val_score, KFold, GridSearchCV

from sklearn.metrics import r2_score, mean_squared_error

from imblearn.pipeline import make_pipeline
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.compose import TransformedTargetRegressor
from sklearn.decomposition import PCA

from sklearn.linear_model import LinearRegression

from scipy.stats import shapiro
```

import data

In [2]:
```
df = pd.read_excel('AmesHousing.xlsx')
```

In [3]:
```
df.head()
```

Out[3]:

	Order	PID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	...	Pool Area	Pool QC	Fence	Misc Feature	Misc Val	Mo Sold	Yr Sold
0	1	526301100	20	RL	141.0	31770	Pave	NaN	IR1	Lvl	...	0	NaN	NaN	NaN	0	5	2010
1	2	526350040	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	...	0	NaN	MnPrv	NaN	0	6	2010
2	3	526351010	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	...	0	NaN	NaN	Gar2	12500	6	2010
3	4	526353030	20	RL	93.0	11160	Pave	NaN	Reg	Lvl	...	0	NaN	NaN	NaN	0	4	2010
4	5	527105010	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	...	0	NaN	MnPrv	NaN	0	3	2010

5 rows × 82 columns

In [4]:
```
df.shape
```

Out[4]:
```
(2930, 82)
```

- The dataset is currently 2,930 x 82

---

1 | Cleaning & Preparing Data

% missing in columns with NaN

In [5]:
```
missing_percentage = df.isna().mean() * 100
missing_percentage = missing_percentage[missing_percentage > 0].round(2)

print(missing_percentage)
```

```
Lot Frontage      16.72
Alley             93.24
Mas Vnr Type      60.58
Mas Vnr Area       0.78
Bsmt Qual          2.73
Bsmt Cond          2.73
Bsmt Exposure      2.83
BsmtFin Type 1     2.73
BsmtFin SF 1       0.03
BsmtFin Type 2     2.76
BsmtFin SF 2       0.03
Bsmt Unf SF        0.03
Total Bsmt SF      0.03
Electrical         0.03
Bsmt Full Bath     0.07
Bsmt Half Bath     0.07
Fireplace Qu      48.53
Garage Type        5.36
Garage Yr Blt      5.43
Garage Finish      5.43
Garage Cars        0.03
Garage Area        0.03
Garage Qual        5.43
Garage Cond        5.43
Pool QC           99.56
Fence             80.48
Misc Feature      96.38
dtype: float64
```

- Six of the columns are missing more than 40% of their values.

---

duplicates

In [6]:
```
print(f'duplicate rows: {df.duplicated().sum()}')
print(f'duplicate columns: {df.columns.duplicated().sum()}')
```

```
duplicate rows: 0
duplicate columns: 0
```

- There are no duplicates.

---

dropping 'Order', 'PID' and columns missing more than 40% of values

In [7]:
```
df = df.drop(['Order', 'PID'], axis=1)
```

In [8]:
```
columns_to_drop = missing_percentage[missing_percentage > 40].index
df              = df.drop(columns=columns_to_drop, axis=1)

print(f"Dropped columns: {columns_to_drop.tolist()}")
```

```
Dropped columns: ['Alley', 'Mas Vnr Type', 'Fireplace Qu', 'Pool QC', 'Fence', 'Misc Feature']
```

In [9]:
```
df.shape
```

Out[9]:
```
(2930, 74)
```

- After dropping those 8 columns, the dataset now has 74 columns.

---

dropping rows with missing values

In [10]:
```
df.isna().sum().sum()
```

Out[10]:
```
1721
```

In [11]:
```
df.dropna(inplace=True)
```

In [12]:
```
df.isna().sum().sum()
```

Out[12]:
```
0
```

In [13]:
```
df.shape
```

Out[13]: (2218, 74)

- 712 rows were dropped, and the data is now 2,218 x 74.

---

dummy variables

In [14]:
```
categorical_columns = df.select_dtypes(include=['object']).columns
df                  = pd.get_dummies(df, columns=categorical_columns, drop_first=True)
```

In [15]:
```
print('Total columns:', len(df.columns))
print('Numerical columns:', len(df.select_dtypes(include=['number']).columns))
print('Boolean columns', len(df.select_dtypes(include=['bool']).columns))
```

```
Total columns: 227
Numerical columns: 37
Boolean columns 190
```

- Now there are a total of 227 columns - 37 numeric and 190 boolean.

---

variables with notable correlation to sale price

In [16]:
```
correlation_matrix = np.abs(df.corr())
correlation_matrix.SalePrice.sort_values(ascending=False)[:20]
```

Out[16]:

```
SalePrice               1.000000
Overall Qual            0.803153
Gr Liv Area             0.717469
Garage Cars             0.666589
Garage Area             0.651103
1st Flr SF              0.649510
Total Bsmt SF           0.648084
Exter Qual_TA           0.609356
Full Bath               0.567360
Year Built              0.557264
Kitchen Qual_TA         0.542506
Foundation_PConc        0.540228
Garage Yr Blt           0.539305
Year Remod/Add          0.535272
TotRms AbvGrd           0.533801
Mas Vnr Area            0.525153
Garage Finish_Unf       0.508507
Bsmt Qual_TA            0.503046
BsmtFin Type 1_GLQ      0.473115
Fireplaces              0.458617
Name: SalePrice, dtype: float64
```

variable pairs with correlation > .80

In [17]:

```
correlation_pairs = correlation_matrix.abs().unstack()
filtered_pairs    = correlation_pairs[(correlation_pairs > 0.8) & (correlation_pairs < 1)].sort_values(ascending=False)
filtered_pairs    = filtered_pairs.drop_duplicates()
correlation_list  = [(index[0], index[1], value) for index, value in filtered_pairs.items()]

for var1, var2, corr in correlation_list:
    print(f"{var1}, {var2}: {corr:.2f}")
```

```
Sale Condition_Partial, Sale Type_New: 0.99
Exterior 1st_CemntBd, Exterior 2nd_CmentBd: 0.98
Exterior 2nd_VinylSd, Exterior 1st_VinylSd: 0.98
Exterior 2nd_MetalSd, Exterior 1st_MetalSd: 0.97
Roof Style_Hip, Roof Style_Gable: 0.95
Total Bsmt SF, 1st Flr SF: 0.90
Exter Qual_TA, Exter Qual_Gd: 0.89
Exter Cond_TA, Exter Cond_Gd: 0.89
Exterior 1st_HdBoard, Exterior 2nd_HdBoard: 0.89
Garage Qual_TA, Garage Qual_Fa: 0.89
Exterior 2nd_Wd Sdng, Exterior 1st_Wd Sdng: 0.88
Exterior 2nd_Brk Cmn, Neighborhood_NPkVill: 0.86
Neighborhood_Somerst, MS Zoning_FV: 0.85
Garage Area, Garage Cars: 0.85
Garage Cond_TA, Garage Cond_Fa: 0.84
Garage Yr Blt, Year Built: 0.84
Garage Type_Attchd, Garage Type_Detchd: 0.83
TotRms AbvGrd, Gr Liv Area: 0.81
Kitchen Qual_Gd, Kitchen Qual_TA: 0.81
MS Zoning_RM, MS Zoning_RL: 0.81
House Style_2Story, 2nd Flr SF: 0.80
2nd Flr SF, House Style_1Story: 0.80
BsmtFin Type 2_Unf, BsmtFin SF 2: 0.80
SalePrice, Overall Qual: 0.80
```

- As shown above there is some severe multicollinearity. PCA will be used in the regression pipeline, so that should not be a problem.

---

features, target and train/test split

In [18]:
```
X = df.drop('SalePrice', axis=1)
y = df.SalePrice
```

In [19]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.33, shuffle=True, random_state=1)
```

2 | Baseline Model : Out-of-the-Box Linear Regression

linear regression

In [20]:
```
base = LinearRegression().fit(X_train,y_train)
```

In [21]:
```
y_pred_base       = base.predict(X_test)
y_train_pred_base = base.predict(X_train)
```

train vs test R2 and RMSE

In [22]:
```
print('Train R2', r2_score(y_train, y_train_pred_base))
print('Train RMSE', np.sqrt(mean_squared_error(y_train, y_train_pred_base)))

print('-'*40)

print('Test R2', r2_score(y_test, y_pred_base))
print('Test RMSE', np.sqrt(mean_squared_error(y_test, y_pred_base)))
```

```
Train R2 0.9340032048433237
Train RMSE 20918.818286344118
----------------------------------------
Test R2 0.7967898809430005
Test RMSE 39254.19396885059
```

- The baseline model appears to be overfitting the data.

R^2 cross val scores

In [23]:
```python
kf     = KFold(n_splits=5, shuffle=True, random_state=2)
scores = cross_val_score(base, X, y, cv=kf, scoring='r2')

print('R^2 scores for each fold:', scores)
print('STD of the R^2 scores:', round(np.std(scores),2))
```

```
R^2 scores for each fold: [0.64519534 0.83179435 0.89121931 0.72082189 0.78510724]
STD of the R^2 scores: 0.09
```

- The cross val scores for R^2 vary a lot which indicates the model is not consistent across different subsets of the data.
- The standard deviation for the cross val scores is 9 percentage points.

---

3 | Pipeline

In [24]:
```python
linreg_pipeline = make_pipeline(MinMaxScaler(), PCA(n_components=.95), LinearRegression())

linreg_pipeline_with_target_normalization=TransformedTargetRegressor(regressor=linreg_pipeline,
                                                                    transformer=StandardScaler())
```

- For the pipeline, a min max scaler is applied to the predictor variables and then PCA is performed such that 95% of the variance is retained. The target variable is transformed with a standard scaler.

---

linear regression model

In [25]:
```python
linreg_pipeline_with_target_normalization.fit(X_train, y_train)
y_train_pred_linreg = linreg_pipeline_with_target_normalization.predict(X_train)
y_pred_linreg       = linreg_pipeline_with_target_normalization.predict(X_test)
```

4 | Model Scores

In [26]:
```
class scores():
    def __init__(self, y_pred, y_train_pred):
        self.y_pred       = y_pred
        self.y_train_pred = y_train_pred

    def r2(self):
        print('Train R2', round(r2_score(y_train, self.y_train_pred),2))
        print('Test R2', round(r2_score(y_test, self.y_pred),2), '\n')

    def rmse(self):
        print('Train RMSE', round(np.sqrt(mean_squared_error(y_train, self.y_train_pred)),0))
        print('Test RMSE', round(np.sqrt(mean_squared_error(y_test,self.y_pred)),0), '\n')

    def r2_cross_val_std(self, model):
        scores = cross_val_score(model, X, y, cv=kf, scoring='r2')
        print('R^2 scores for each fold:', scores)
        print('STD of the R^2 scores:', round(np.std(scores),2))
```

linear regression scores

In [27]:
```
scores(y_pred_linreg, y_train_pred_linreg).r2()
scores(y_pred_linreg, y_train_pred_linreg).rmse()
scores(y_pred_linreg, y_train_pred_linreg).r2_cross_val_std(linreg_pipeline_with_target_normalization)
```

```
Train R2 0.9
Test R2 0.82

Train RMSE 25324.0
Test RMSE 36894.0

R^2 scores for each fold: [0.74850275 0.83304722 0.87125761 0.90270269 0.8852293 ]
STD of the R^2 scores: 0.05
```

- The linear regression model appears to be somewhat overfitting the training data.

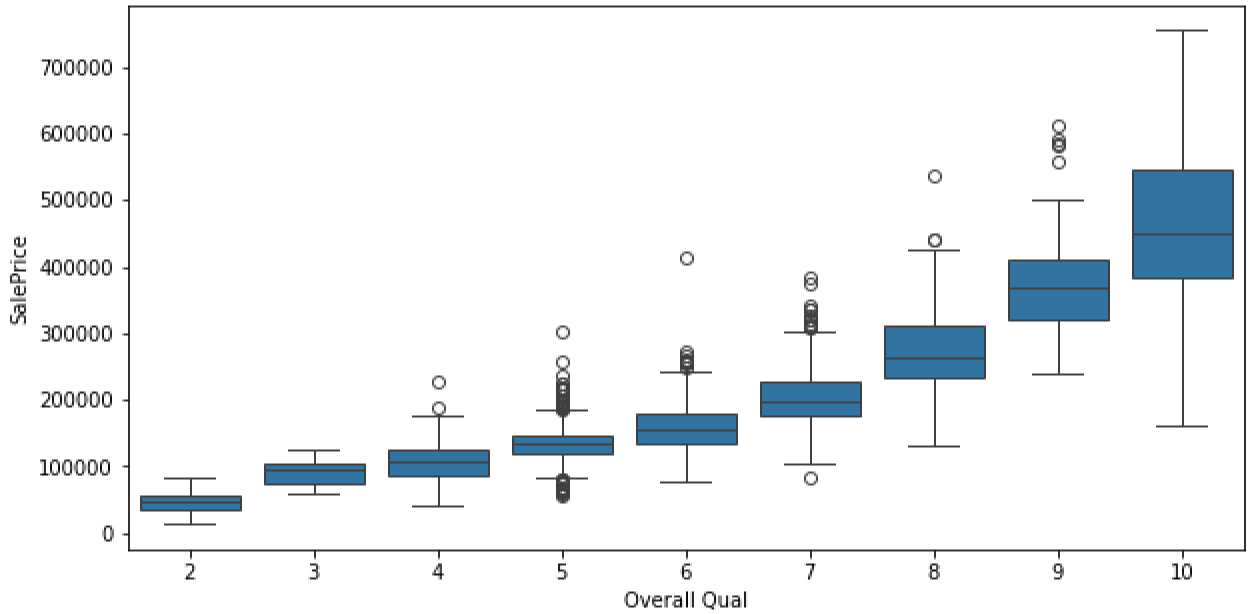
- The cross val scores are better than the base model, but vary enough to indicate the model is not consistent across different subsets of the data.
- The standard deviation of the cross val scores is 5 percentage points.

---

5 | Dropping Outliers

outlier indices for Saleprice by Overall Qual

In [28]:
```
plt.figure(figsize=(10,5))
sns.boxplot(x=df['Overall Qual'], y=df.SalePrice)
plt.show()
```

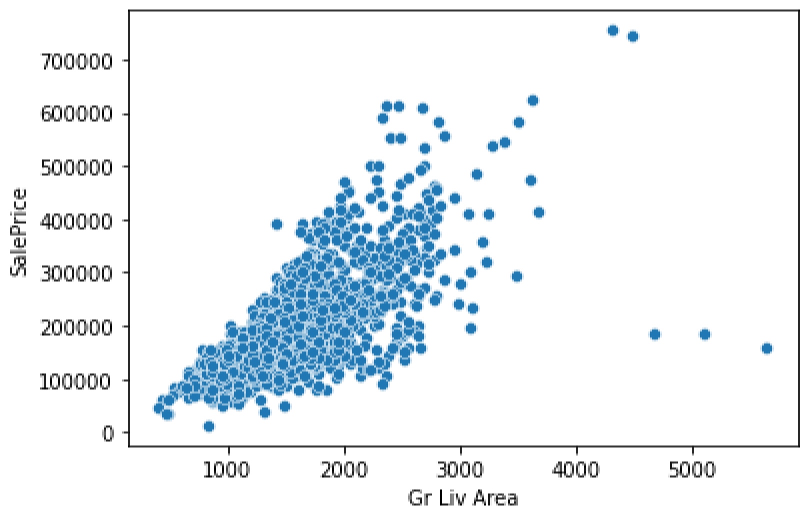


In [29]:
```
qual_outliers_indices = []
```

```
for qual in df['Overall Qual'].unique():
    data = df[df['Overall Qual'] == qual]['SalePrice']

    Q1  = data.quantile(0.25)
    Q3  = data.quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = data[(data < lower_bound) | (data > upper_bound)]

    qual_outliers_indices.extend(outliers.index.tolist())
```

outlier indices for SalePrice by Gr Liv Area

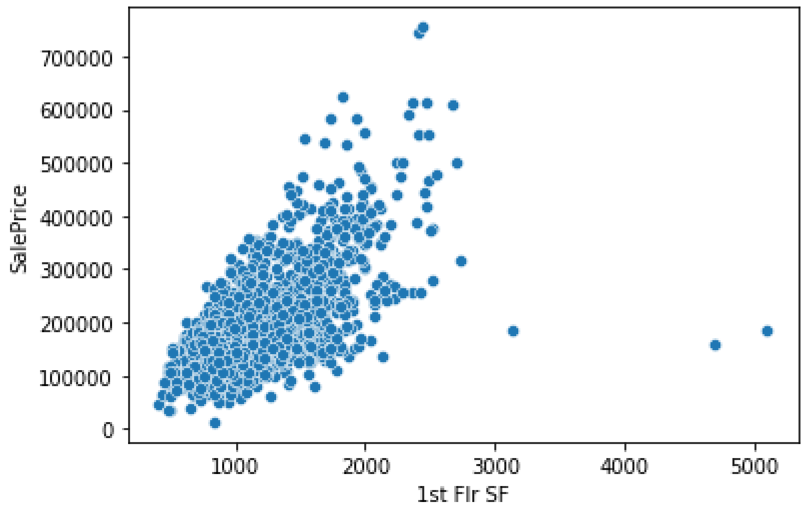
In [30]:
```
sns.scatterplot(x=df['Gr Liv Area'], y=df.SalePrice)
plt.show()
```



In [31]:
```
gr_outliers_indices = df[df['Gr Liv Area']>4000]['Gr Liv Area'].index
```

outlier indices for SalePrice by 1st Flr SF

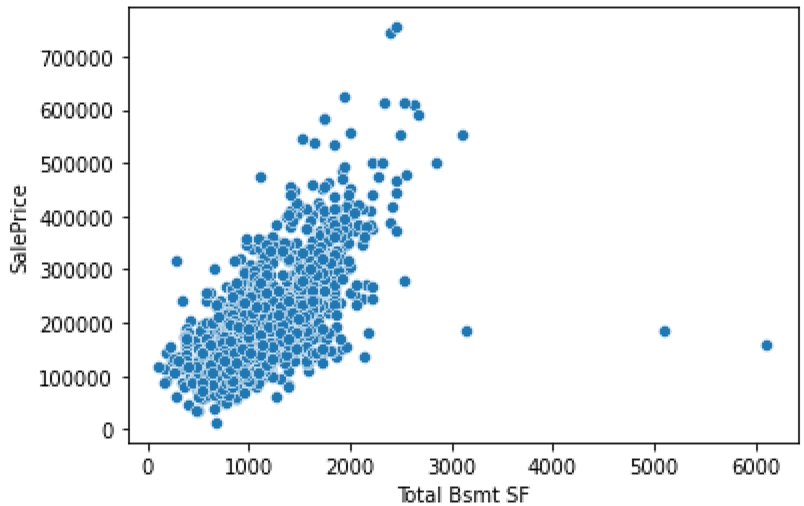
In [32]:
```
sns.scatterplot(x=df['1st Flr SF'], y=df.SalePrice)
plt.show()
```



In [33]:
```
garage_outliers_indices = df[df['1st Flr SF']>4000]['Gr Liv Area'].index
```

outlier indices for SalePrice by Total Bsmt SF

In [34]:
```
sns.scatterplot(x=df['Total Bsmt SF'], y=df.SalePrice)
plt.show()
```



In [35]:
```
bsmt_outliers_indices = df[df['Total Bsmt SF']>5000]['Gr Liv Area'].index
```

combining lists of outlier indices

In [36]:
```
all_outliers_indices = []

all_outliers_indices.extend(qual_outliers_indices)
all_outliers_indices.extend(gr_outliers_indices)
all_outliers_indices.extend(garage_outliers_indices)
all_outliers_indices.extend(bsmt_outliers_indices)
```

In [37]:
```
len(all_outliers_indices)
```

Out[37]:
```
79
```

dropping outlier indices from data

In [38]:
```
df = df.drop(index=all_outliers_indices)
```

6 | Retraining Linear Regression Model

In [39]:
```
X = df.drop('SalePrice', axis=1)
y = df.SalePrice
```

In [40]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.33, shuffle=True, random_state=1)
```

In [41]:
```
linreg_pipeline_with_target_normalization.fit(X_train, y_train)
y_train_pred_linreg = linreg_pipeline_with_target_normalization.predict(X_train)
y_pred_linreg       = linreg_pipeline_with_target_normalization.predict(X_test)
```

In [42]:
```
scores(y_pred_linreg, y_train_pred_linreg).r2()
scores(y_pred_linreg, y_train_pred_linreg).rmse()
scores(y_pred_linreg, y_train_pred_linreg).r2_cross_val_std(linreg_pipeline_with_target_normalization)
```

```
Train R2 0.92
Test R2 0.91

Train RMSE 21839.0
Test RMSE 22169.0

R^2 scores for each fold: [0.91017866 0.91139222 0.90076857 0.9124529  0.91176443]
STD of the R^2 scores: 0.0
```

- The linear regression model does not appear to be overfitting the training data.
- The cross val scores are very close together which indicates the model is consistent across different subsets of the data.
- The standard deviation of the cross val scores is less than 1 percentage point.

- An R2 of 0.91 means that 91% of the variance in the data is explained by the model.
- An RMSE of 22,169 means the model is on average off by about 22,169 dollars. As the range of house prices is 612,211 this could be described as an average error of about 3.6%.

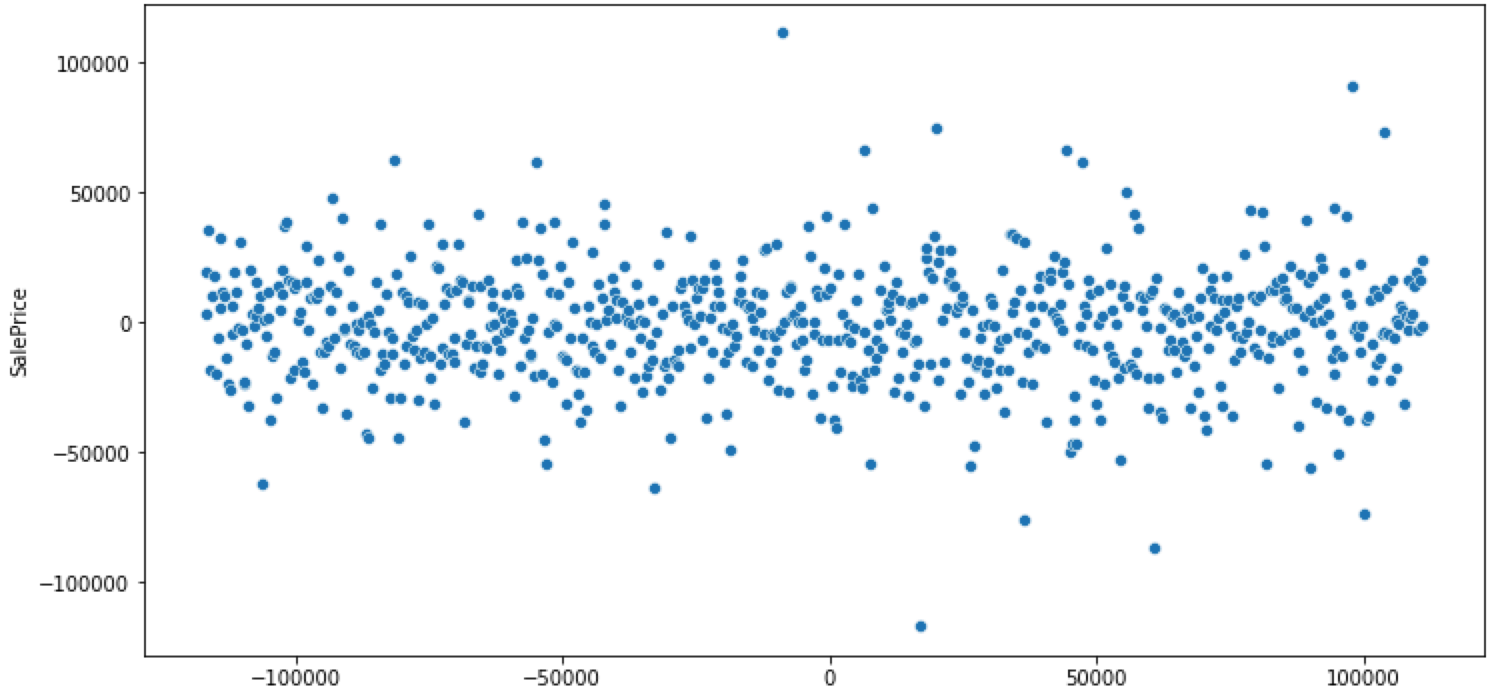
---

7 | Residuals of the Model

plot

In [43]:
```
resids = y_test-y_pred_linreg
```

In [44]:
```
x_axis= np.linspace(min(resids), max(resids), len(resids))

plt.figure(figsize=(12,6))
sns.scatterplot(x=x_axis, y=resids)
plt.show()
```



- Looking closely at the residuals, they appear to form nonlinear patterns in different spots. This could be a sign that a nonlinear model would better capture the data.

Shapiro-Wilk test

In [45]:
```
shapiro(resids)
```

Out[45]:
```
ShapiroResult(statistic=0.9714756011962891, pvalue=1.6150636383827077e-10)
```

- This is further backed up by the Shapiro-Wilk test which indicates the residuals are not normally distributed.

8 | Conclusion

- In conclusion the data was cleaned and prepared for training. Then an out of the box linear regression model was fit to the data. The model overfit the data and the cross val score varied a lot. The standard deviation of the cross val scores was 9 percentage points.

- Next a pipeline was created. For the pipeline, a min max scaler was applied to the predictor variables and then PCA was performed such that 95% of the variance was retained. The target variable was transformed with a standard scaler. That model somewhat overfit the data and the cross val scores varied enough to indicate the model was not consistent across different subsets of the data. The standard deviation of the cross val scores was 5 percentage points.

- Next outliers were dropped from SalePrice by Overall Qual, Gr Liv Area, 1st Flr SF and Total Bsmt SF. The pipeline created before was then retrained. The model did not overfit the data and the cross val scores were very close together. The standard deviation of the cross val scores was less than 1 percentage point. The test R2 was 0.91, the test RMSE was 22,169 - very good scores.

- Lastly, the residuals for the final model were plotted and there appeared to be nonlinear patterns. Furthermore, the Shapiro-Wilk test showed the residuals were not normally distributed. This indicates that while the linear regression model has good performance metrics, a nonlinear model may do a better job at capturing the patterns in the data.