Department of Computer Science and Engineering (Data Science)

Subject: Applied Data Science (DJ19DSL703)

Experiment - 6

(Model Building)

SAP ID: 60009200056
NAME: Rishabh Patil
DIV./BATCH: D12

Aim: To build model using feature engineering.

Theory:

Feature engineering is a critical step in the data pre-processing process in data science and machine learning. It involves creating new features or modifying existing ones in your dataset to improve the performance of machine learning models. Proper feature engineering can lead to more accurate and robust models. Here are some key aspects of feature engineering:

- Feature Extraction: Feature extraction involves transforming raw data into a format that's suitable for machine learning. For example, extracting date features (e.g., year, month, day) from a timestamp, converting text data into numerical representations (e.g., TF-IDF, word embedding), or summarizing information in image data (e.g., color histograms, edge detection).
- Feature Transformation: This involves applying mathematical or statistical transformations to your features to make them more suitable for modelling. Common techniques include scaling (e.g., standardization or normalization), log or power transformations, and encoding categorical variables (e.g., one-hot encoding or label encoding).
- Feature Creation: Sometimes, you may need to create new features based on domain knowledge or insights gained during data exploration. This could involve combining existing features, creating interaction terms, or engineering new variables to capture specific patterns or relationships.
- Handling Missing Data: Dealing with missing data is also part of feature engineering. You can choose to impute missing values using techniques like mean, median, or predictive modelling. Sometimes, you may create binary flags indicating the presence of missing data.
- Feature Selection: Feature engineering also involves selecting the most relevant features for your model. This can be done through techniques like univariate feature selection, feature importance from tree-based models, or through domain knowledge.
- Text and NLP Feature Engineering: When working with text data, you may need to perform additional feature engineering, such as tokenization, stemming, lemmatization, and sentiment analysis.
- Handling Categorical Data: Categorical variables require special attention. You can use techniques like one-hot encoding, label encoding, or target encoding to represent categorical data numerically.
- Temporal Feature Engineering: For time-series data, creating lag features or time-based aggregations can be valuable.
- Geospatial Feature Engineering: For geospatial data, you might calculate distances, create spatial clusters, or derive location-based statistics.

- Domain-Specific Feature Engineering: In some cases, domain-specific knowledge can lead to unique feature engineering approaches. This might include creating custom metrics or indicators specific to your problem.
- Feature engineering is an iterative process, and it often requires experimenting with different feature combinations and transformations to find the most informative and predictive features for your machine learning model. Effective feature engineering can significantly impact the model's performance and its ability to uncover valuable insights from the data.

Model Evaluation / Performance Metrics

The choice of performance metrics depends on the type of machine learning problem. Here are some common metrics for different types of problems:

• For binary classification, you can use metrics like accuracy, precision, recall, F1-score, and the receiver operating characteristic (ROC) curve.
• For multi-class classification, metrics like accuracy, precision, recall, F1-score, and confusion matrices can be useful.
• For regression problems, metrics like mean squared error (MSE), mean absolute error (MAE), and R-squared (R2) are commonly used.

Identify and Minimize data leakage

•**Train-Test Split:** This is a common practice in which you divide your dataset into two subsets: one for training the model and the other for testing its performance. The split, often referred to as the training set and the test set, allows you to train your model on a portion of the data and then evaluate its performance on unseen data.

•**Cross-Validation:** Cross-validation is a more robust technique than a simple train-test split. It involves partitioning the dataset into multiple subsets (usually k subsets or "folds") and training and evaluating the model multiple times. The results are then averaged to provide a more reliable estimate of the model's performance. Common cross-validation methods include k-fold cross-validation and stratified k-fold cross-validation.

Lab Assignment:

- At this stage, we should be having a cleaned data. We will consider the cleaned data for feature engineering steps. Identify new features, make existing features better, remove unwanted or highly correlated features.
- We will perform train test split of the data. Train the model on the training data and evaluate the model on test data.
- Perform cross validation (changing train test split data) and then rechecking metrics to see if the model is not overfitting.

Assignment Examples to work with

1. House price prediction
2. Sales prediction
3. Credit card fraud detection
4. Customer segmentation
5. Diabetes prediction

In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

from xgboost import XGBRegressor
```

In [ ]:
```
df = pd.read_csv("HousePricePrediction.csv")
```

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

Out[ ]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utili
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllI
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllI
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllI
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllI
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllI

5 rows × 81 columns

Mapping Statistics of Data

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1460 non-null   int64  
 1   MSSubClass     1460 non-null   int64  
 2   MSZoning       1460 non-null   object 
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64  
 5   Street         1460 non-null   object 
 6   Alley          91 non-null     object 
 7   LotShape       1460 non-null   object 
 8   LandContour    1460 non-null   object 
 9   Utilities      1460 non-null   object 
 10  LotConfig      1460 non-null   object 
 11  LandSlope      1460 non-null   object 
 12  Neighborhood   1460 non-null   object 
 13  Condition1     1460 non-null   object 
 14  Condition2     1460 non-null   object 
 15  BldgType       1460 non-null   object 
 16  HouseStyle     1460 non-null   object 
 17  OverallQual    1460 non-null   int64  
 18  OverallCond    1460 non-null   int64  
 19  YearBuilt      1460 non-null   int64  
 20  YearRemodAdd   1460 non-null   int64  
 21  RoofStyle      1460 non-null   object 
 22  RoofMatl       1460 non-null   object 
 23  Exterior1st    1460 non-null   object 
 24  Exterior2nd    1460 non-null   object 
 25  MasVnrType     1452 non-null   object 
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object 
 28  ExterCond      1460 non-null   object 
 29  Foundation     1460 non-null   object 
 30  BsmtQual       1423 non-null   object 
 31  BsmtCond       1423 non-null   object 
 32  BsmtExposure   1422 non-null   object 
 33  BsmtFinType1   1423 non-null   object 
 34  BsmtFinSF1     1460 non-null   int64  
 35  BsmtFinType2   1422 non-null   object 
 36  BsmtFinSF2     1460 non-null   int64  
 37  BsmtUnfSF      1460 non-null   int64  
 38  TotalBsmtSF    1460 non-null   int64  
 39  Heating        1460 non-null   object 
 40  HeatingQC      1460 non-null   object 
 41  CentralAir     1460 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1460 non-null   int64  
 44  2ndFlrSF       1460 non-null   int64  
 45  LowQualFinSF   1460 non-null   int64  
 46  GrLivArea      1460 non-null   int64  
 47  BsmtFullBath   1460 non-null   int64  
 48  BsmtHalfBath   1460 non-null   int64  
 49  FullBath       1460 non-null   int64  
 50  HalfBath       1460 non-null   int64  
 51  BedroomAbvGr   1460 non-null   int64  
 52  KitchenAbvGr   1460 non-null   int64  
 53  KitchenQual    1460 non-null   object 
 54  TotRmsAbvGrd   1460 non-null   int64  
 55  Functional     1460 non-null   object 
 56  Fireplaces     1460 non-null   int64  
 57  FireplaceQu    770 non-null    object 
 58  GarageType     1379 non-null   object 
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object 
 61  GarageCars     1460 non-null   int64  
 62  GarageArea     1460 non-null   int64  
 63  GarageQual     1379 non-null   object 
 64  GarageCond     1379 non-null   object 
 65  PavedDrive     1460 non-null   object 
 66  WoodDeckSF     1460 non-null   int64  
 67  OpenPorchSF    1460 non-null   int64  
 68  EnclosedPorch  1460 non-null   int64  
 69  3SsnPorch      1460 non-null   int64  
 70  ScreenPorch    1460 non-null   int64  
 71  PoolArea       1460 non-null   int64  
 72  PoolQC         7 non-null      object 
 73  Fence          281 non-null    object 
 74  MiscFeature    54 non-null     object 
 75  MiscVal        1460 non-null   int64  
 76  MoSold         1460 non-null   int64  
 77  YrSold         1460 non-null   int64  
 78  SaleType       1460 non-null   object 
 79  SaleCondition  1460 non-null   object 
 80  SalePrice      1460 non-null   int64  
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

In [ ]:
```
df.describe().T
```

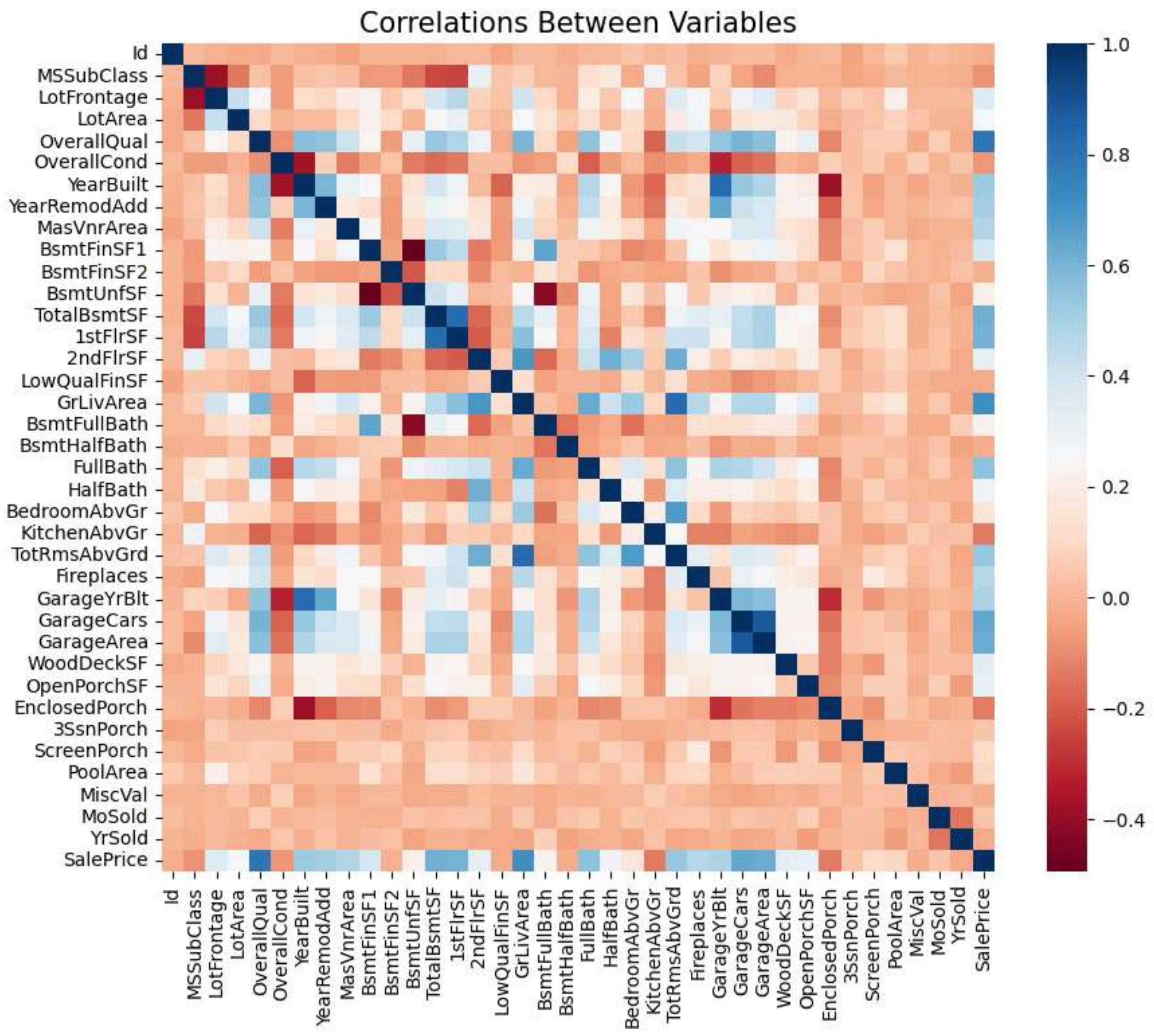
Out[ ]:

	count	mean	std	min	25%	50%	75%	
Id	1460.0	730.500000	421.610009	1.0	365.75	730.5	1095.25	
MSSubClass	1460.0	56.897260	42.300571	20.0	20.00	50.0	70.00	
LotFrontage	1201.0	70.049958	24.284752	21.0	59.00	69.0	80.00	
LotArea	1460.0	10516.828082	9981.264932	1300.0	7553.50	9478.5	11601.50	21
OverallQual	1460.0	6.099315	1.382997	1.0	5.00	6.0	7.00	
OverallCond	1460.0	5.575342	1.112799	1.0	5.00	5.0	6.00	
YearBuilt	1460.0	1971.267808	30.202904	1872.0	1954.00	1973.0	2000.00	
YearRemodAdd	1460.0	1984.865753	20.645407	1950.0	1967.00	1994.0	2004.00	
MasVnrArea	1452.0	103.685262	181.066207	0.0	0.00	0.0	166.00	
BsmtFinSF1	1460.0	443.639726	456.098091	0.0	0.00	383.5	712.25	
BsmtFinSF2	1460.0	46.549315	161.319273	0.0	0.00	0.0	0.00	
BsmtUnfSF	1460.0	567.240411	441.866955	0.0	223.00	477.5	808.00	
TotalBsmtSF	1460.0	1057.429452	438.705324	0.0	795.75	991.5	1298.25	
1stFlrSF	1460.0	1162.626712	386.587738	334.0	882.00	1087.0	1391.25	
2ndFlrSF	1460.0	346.992466	436.528436	0.0	0.00	0.0	728.00	
LowQualFinSF	1460.0	5.844521	48.623081	0.0	0.00	0.0	0.00	
GrLivArea	1460.0	1515.463699	525.480383	334.0	1129.50	1464.0	1776.75	
BsmtFullBath	1460.0	0.425342	0.518911	0.0	0.00	0.0	1.00	
BsmtHalfBath	1460.0	0.057534	0.238753	0.0	0.00	0.0	0.00	
FullBath	1460.0	1.565068	0.550916	0.0	1.00	2.0	2.00	
HalfBath	1460.0	0.382877	0.502885	0.0	0.00	0.0	1.00	
BedroomAbvGr	1460.0	2.866438	0.815778	0.0	2.00	3.0	3.00	
KitchenAbvGr	1460.0	1.046575	0.220338	0.0	1.00	1.0	1.00	
TotRmsAbvGrd	1460.0	6.517808	1.625393	2.0	5.00	6.0	7.00	
Fireplaces	1460.0	0.613014	0.644666	0.0	0.00	1.0	1.00	
GarageYrBlt	1379.0	1978.506164	24.689725	1900.0	1961.00	1980.0	2002.00	
GarageCars	1460.0	1.767123	0.747315	0.0	1.00	2.0	2.00	
GarageArea	1460.0	472.980137	213.804841	0.0	334.50	480.0	576.00	
WoodDeckSF	1460.0	94.244521	125.338794	0.0	0.00	0.0	168.00	
OpenPorchSF	1460.0	46.660274	66.256028	0.0	0.00	25.0	68.00	
EnclosedPorch	1460.0	21.954110	61.119149	0.0	0.00	0.0	0.00	
3SsnPorch	1460.0	3.409589	29.317331	0.0	0.00	0.0	0.00	
ScreenPorch	1460.0	15.060959	55.757415	0.0	0.00	0.0	0.00	
PoolArea	1460.0	2.758904	40.177307	0.0	0.00	0.0	0.00	
MiscVal	1460.0	43.489041	496.123024	0.0	0.00	0.0	0.00	1
MoSold	1460.0	6.321918	2.703626	1.0	5.00	6.0	8.00	

	count	mean	std	min	25%	50%	75%
YrSold	1460.0	2007.815753	1.328095	2006.0	2007.00	2008.0	2009.00
SalePrice	1460.0	180921.195890	79442.502883	34900.0	129975.00	163000.0	214000.00

Correlation Features

In [ ]:
```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), cmap="RdBu")
plt.title("Correlations Between Variables", size=15)
plt.show()
```



Feature Selection

In [ ]:
```
important_num_cols = list(df.corr()["SalePrice"][(df.corr()["SalePrice"]>0.50)
| (df.corr()["SalePrice"]<-0.50)].index)
# Only those columns are selected which can be important (Subjective)
cat_cols = ["MSZoning", "Utilities","BldgType","Heating","KitchenQual","SaleCo
ndition","LandSlope"]
important_cols = important_num_cols + cat_cols
df = df[important_cols]
df
```

Out[ ]:

	OverallQual	YearBuilt	YearRemodAdd	TotalBsmtSF	1stFlrSF	GrLivArea	FullBath	TotRms
0	7	2003	2003	856	856	1710	2	
1	6	1976	1976	1262	1262	1262	2	
2	7	2001	2002	920	920	1786	2	
3	7	1915	1970	756	961	1717	1	
4	8	2000	2000	1145	1145	2198	2	
...	...	...	...	...	...	...	...	
1455	6	1999	2000	953	953	1647	2	
1456	6	1978	1988	1542	2073	2073	2	
1457	7	1941	2006	1152	1188	2340	2	
1458	5	1950	1996	1078	1078	1078	1	
1459	5	1965	1965	1256	1256	1256	1	

1460 rows × 18 columns

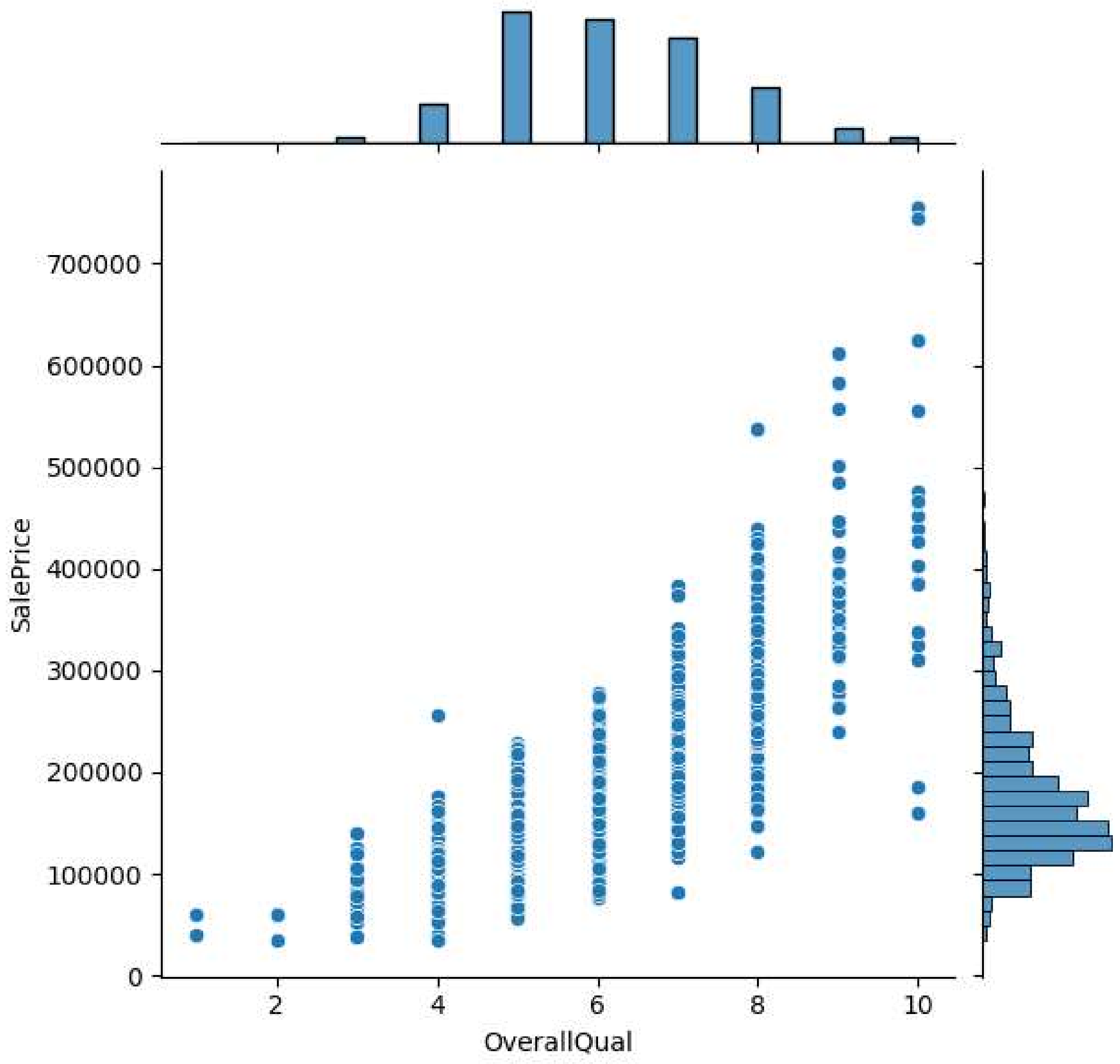
```
In [ ]: print("Missing Values by Column")
        print("-"*30)
        print(df.isna().sum())
        print("-"*30)
        print("TOTAL MISSING VALUES:",df.isna().sum().sum())
```

```
Missing Values by Column
------------------------------
OverallQual      0
YearBuilt        0
YearRemodAdd     0
TotalBsmtSF      0
1stFlrSF         0
GrLivArea        0
FullBath         0
TotRmsAbvGrd     0
GarageCars       0
GarageArea       0
SalePrice        0
MSZoning         0
Utilities        0
BldgType         0
Heating          0
KitchenQual      0
SaleCondition    0
LandSlope        0
dtype: int64
------------------------------
TOTAL MISSING VALUES: 0
```

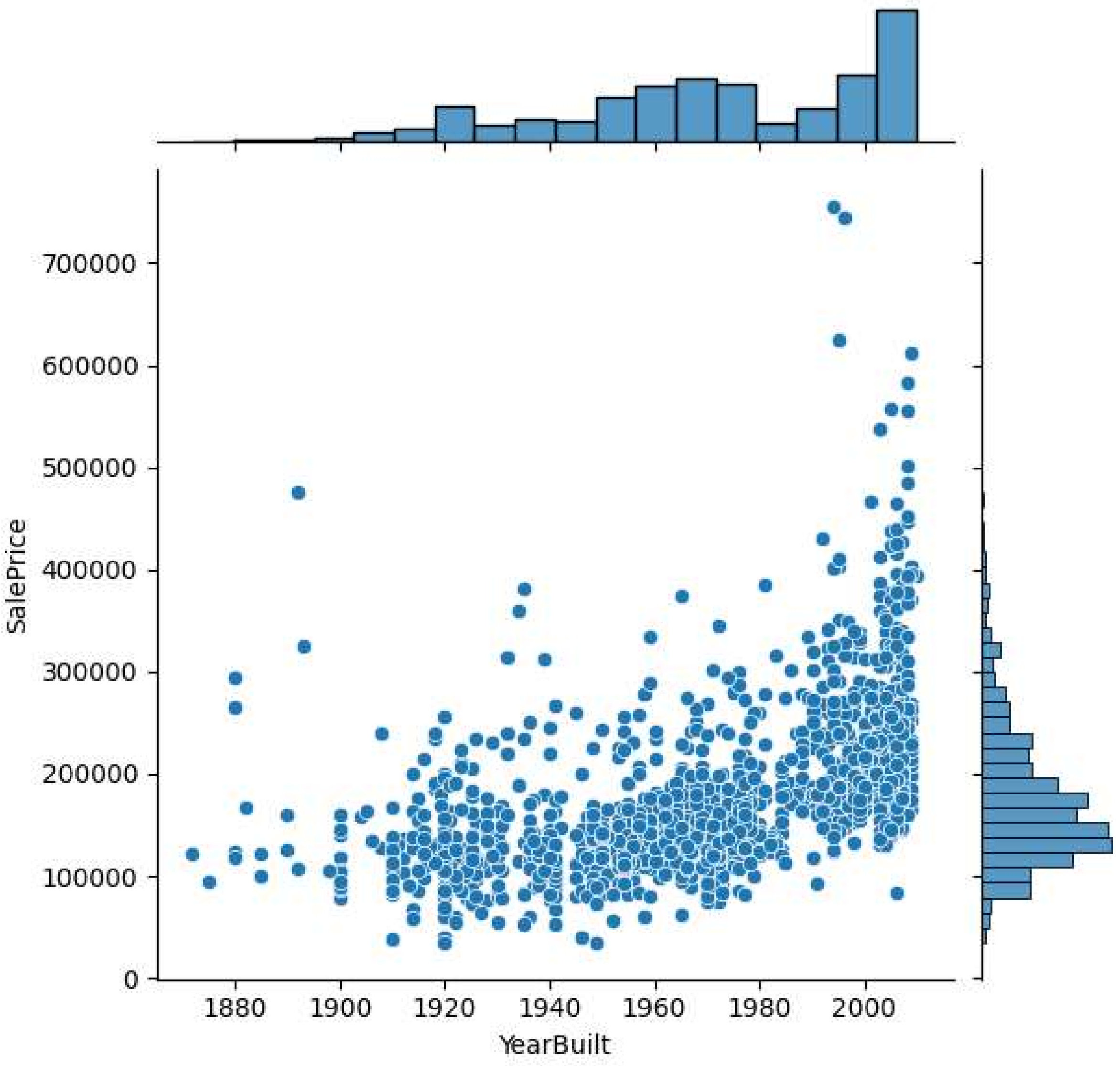
Correlation Using Joint Plots

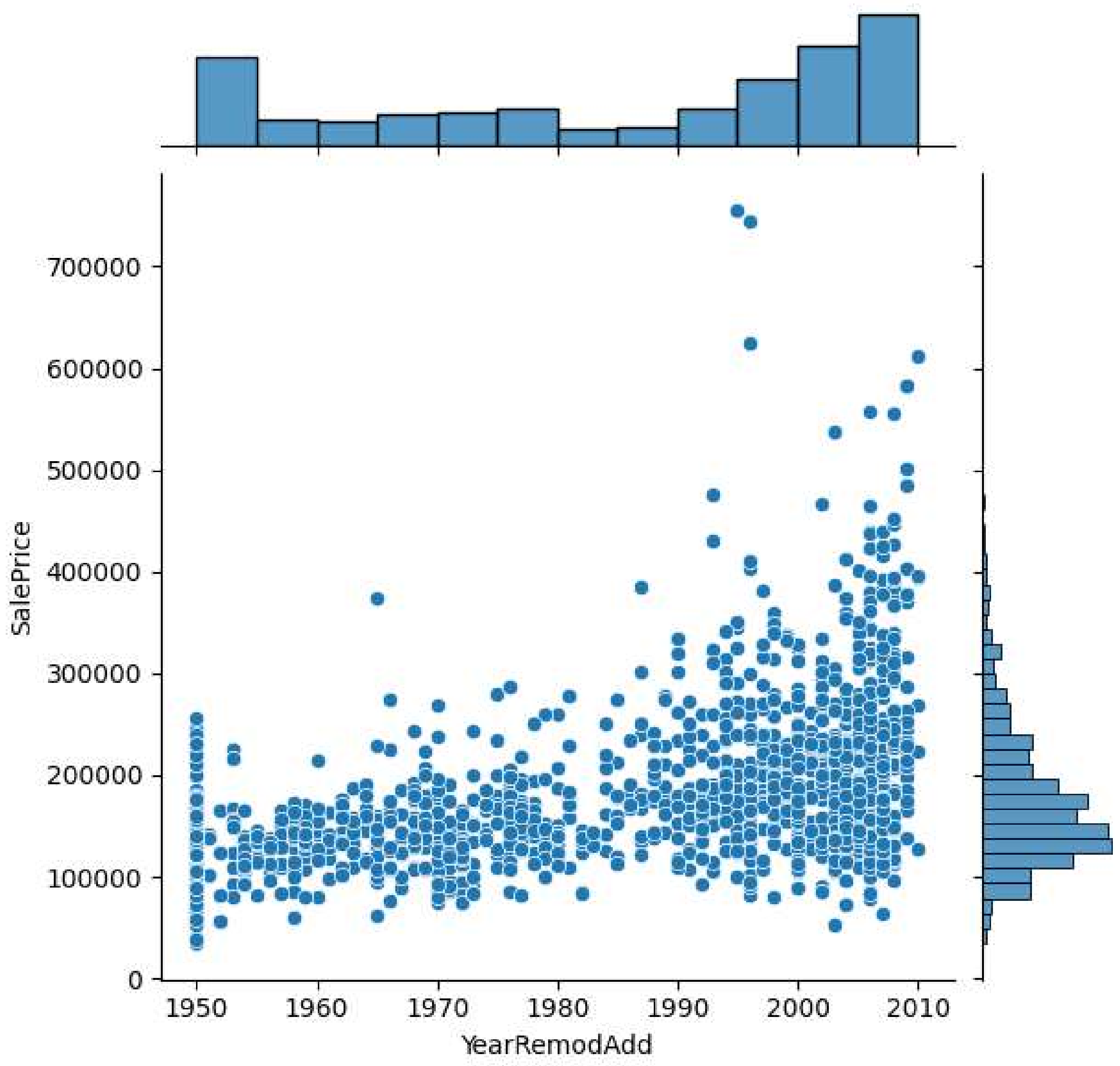
In [ ]:
```
plt.figure(figsize=(10,8))
for i in df.columns:
  if i not in cat_cols:
    sns.jointplot(x=df[i], y=df["SalePrice"])
    plt.show()
```

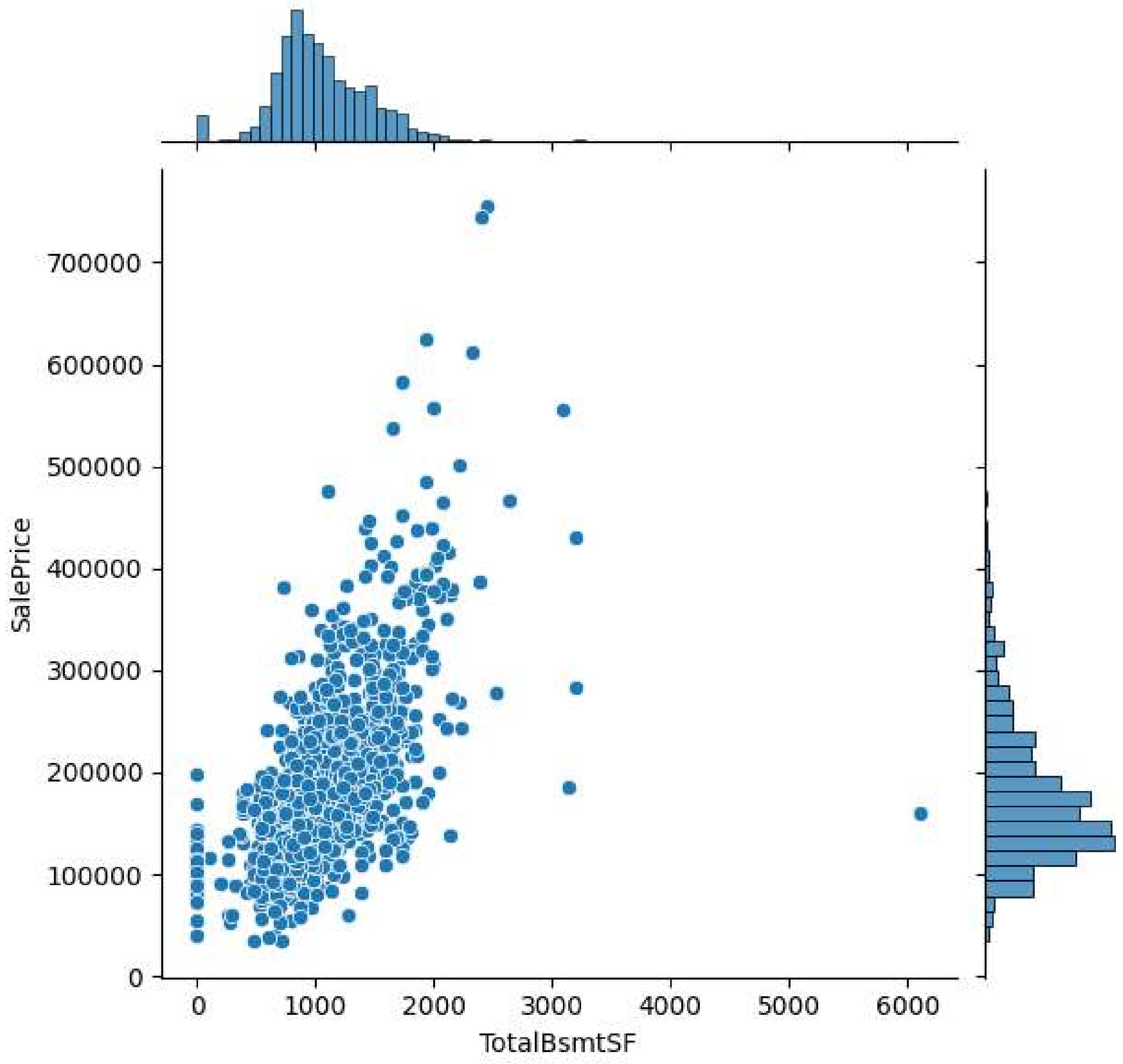
```
<Figure size 1000x800 with 0 Axes>
```

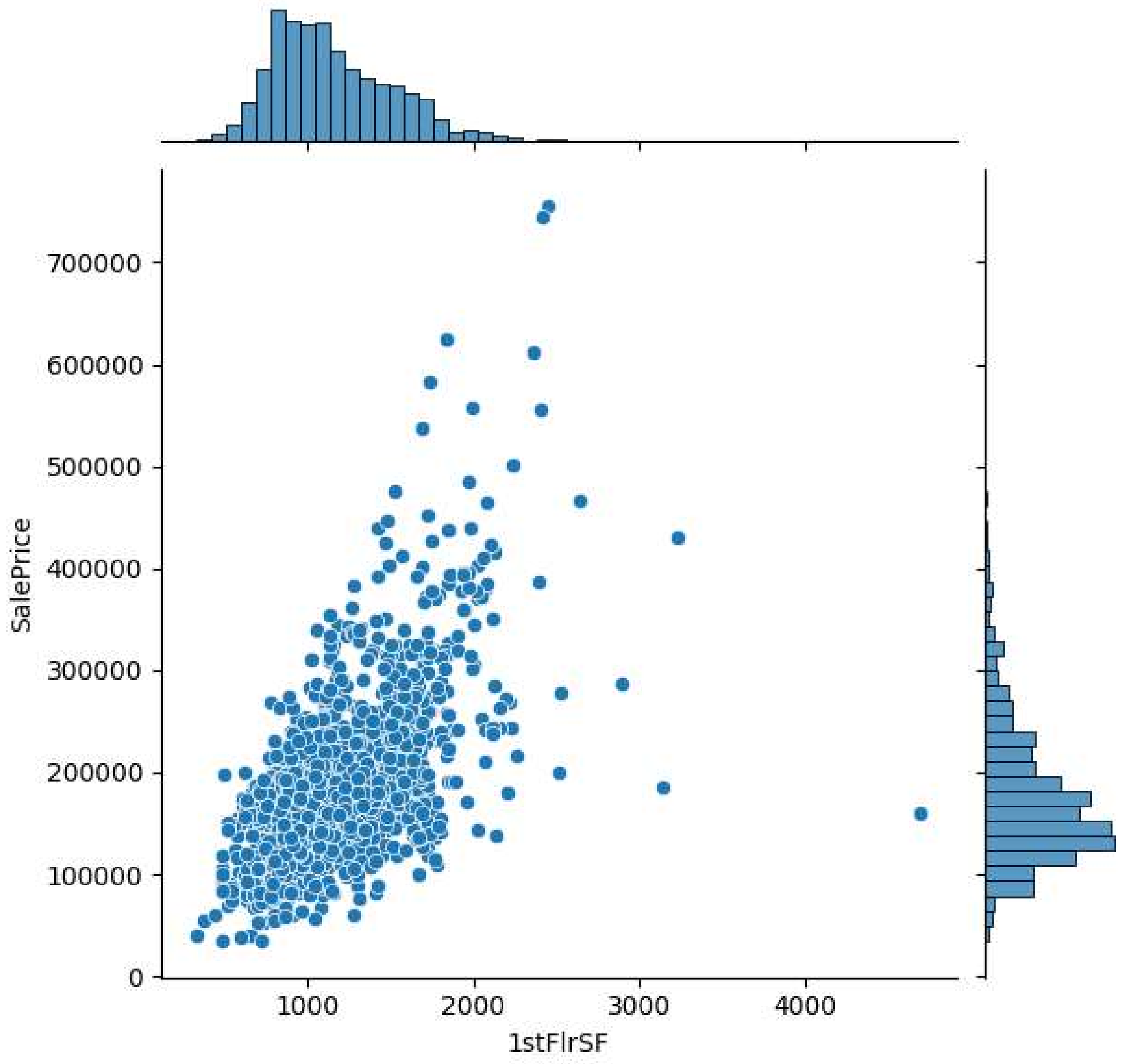


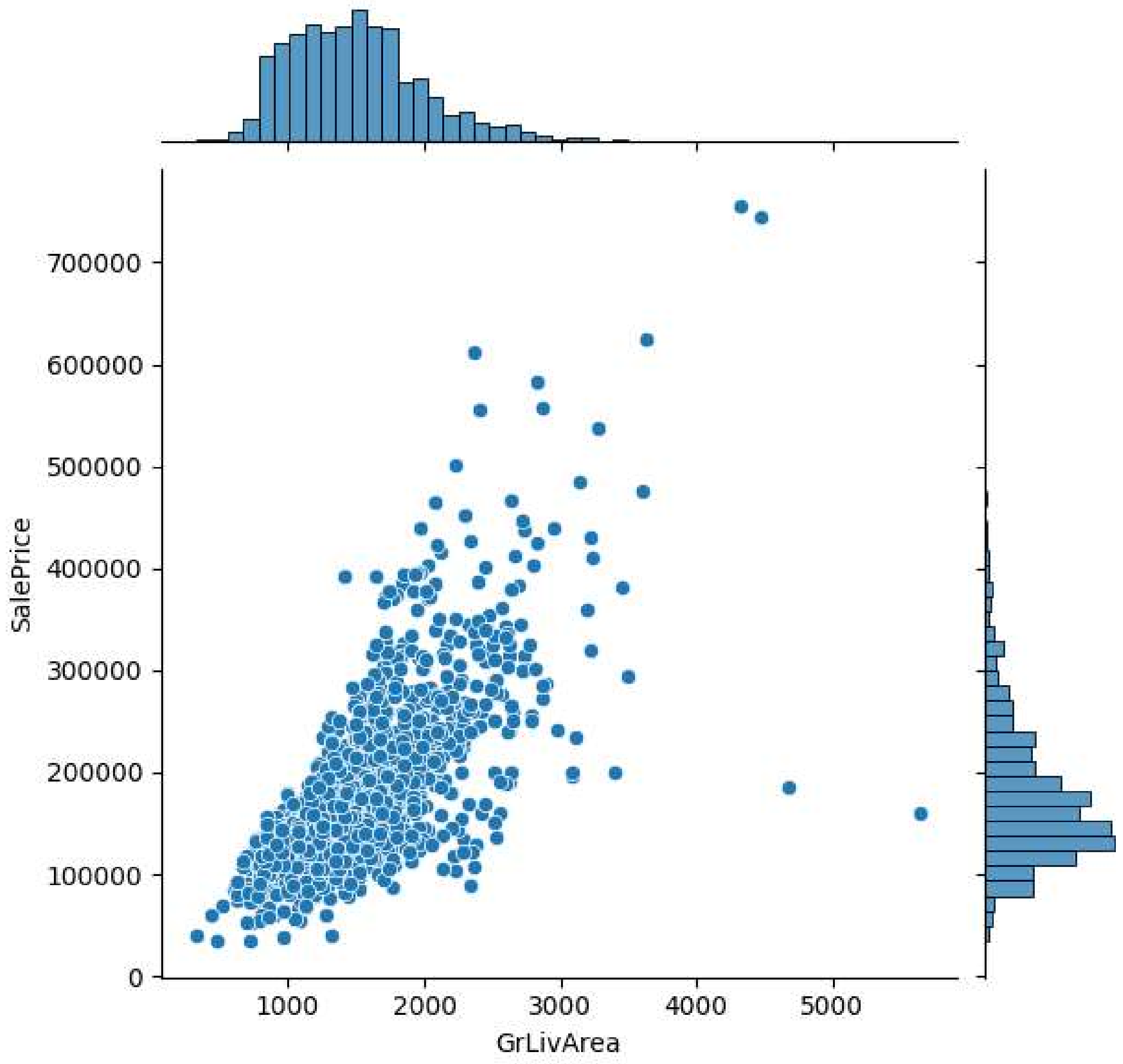
SalePrice
OverallQual
0
100000
200000
300000
400000
500000
600000
700000
2
4
6
8
10


SalePrice
0
100000
200000
300000
400000
500000
600000
700000
1880
1900
1920
1940
1960
1980
2000
YearBuilt

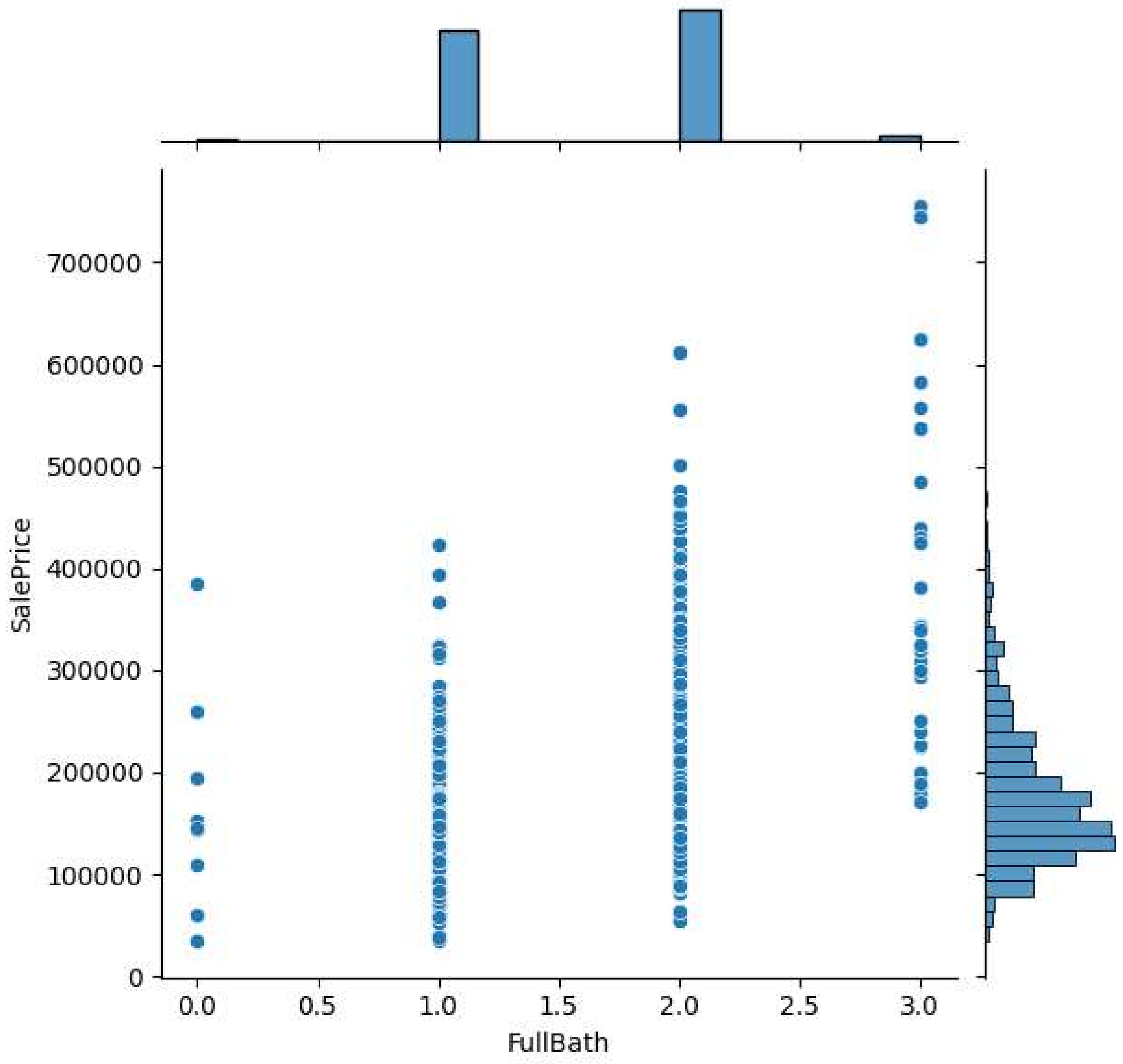

SalePrice
700000
600000
500000
400000
300000
200000
100000
0
1950
1960
1970
1980
1990
2000
2010
YearRemodAdd

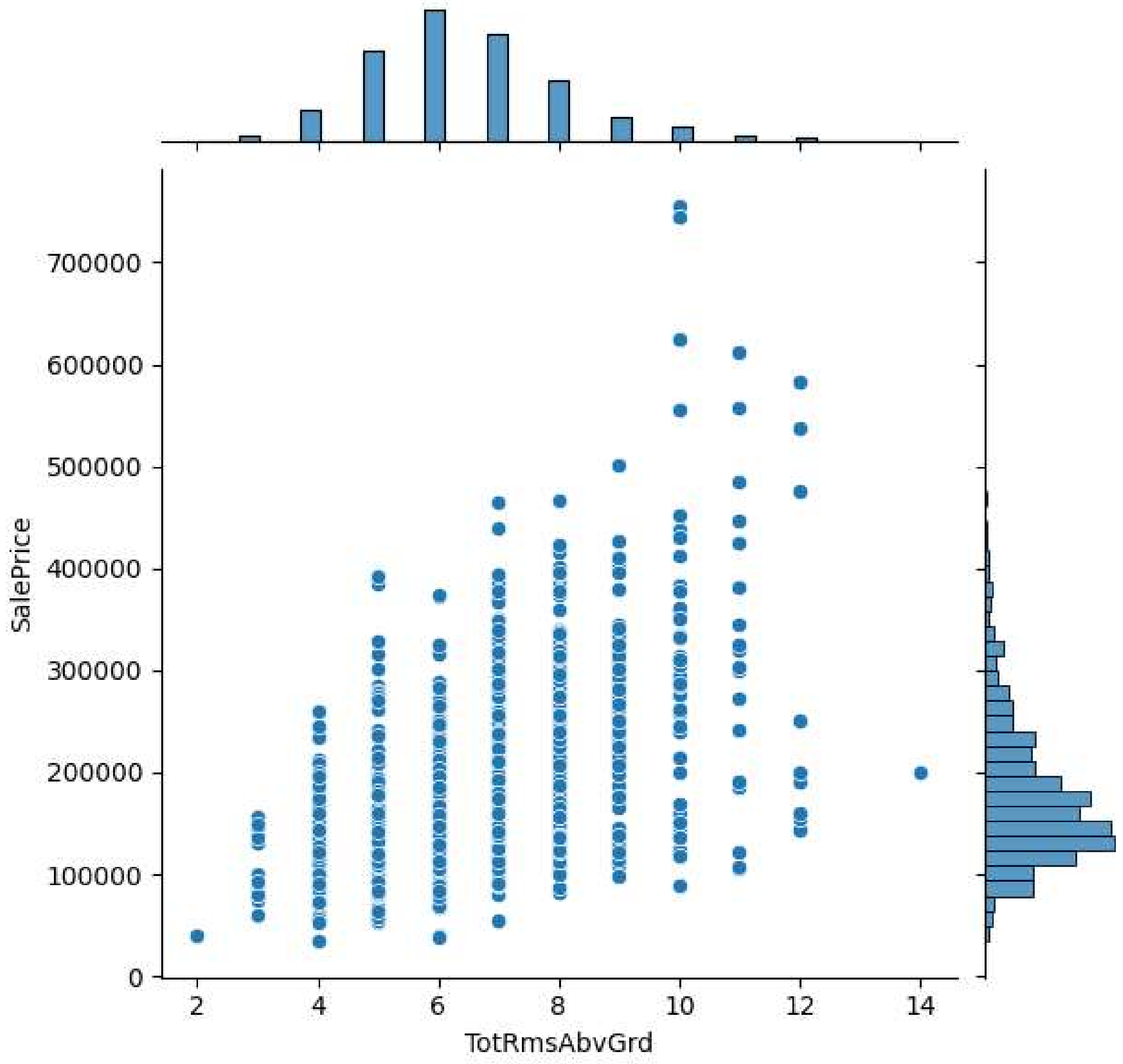

SalePrice
0
100000
200000
300000
400000
500000
600000
700000
TotalBsmtSF
0
1000
2000
3000
4000
5000
6000


SalePrice
1stFlrSF
0
100000
200000
300000
400000
500000
600000
700000
1000
2000
3000
4000

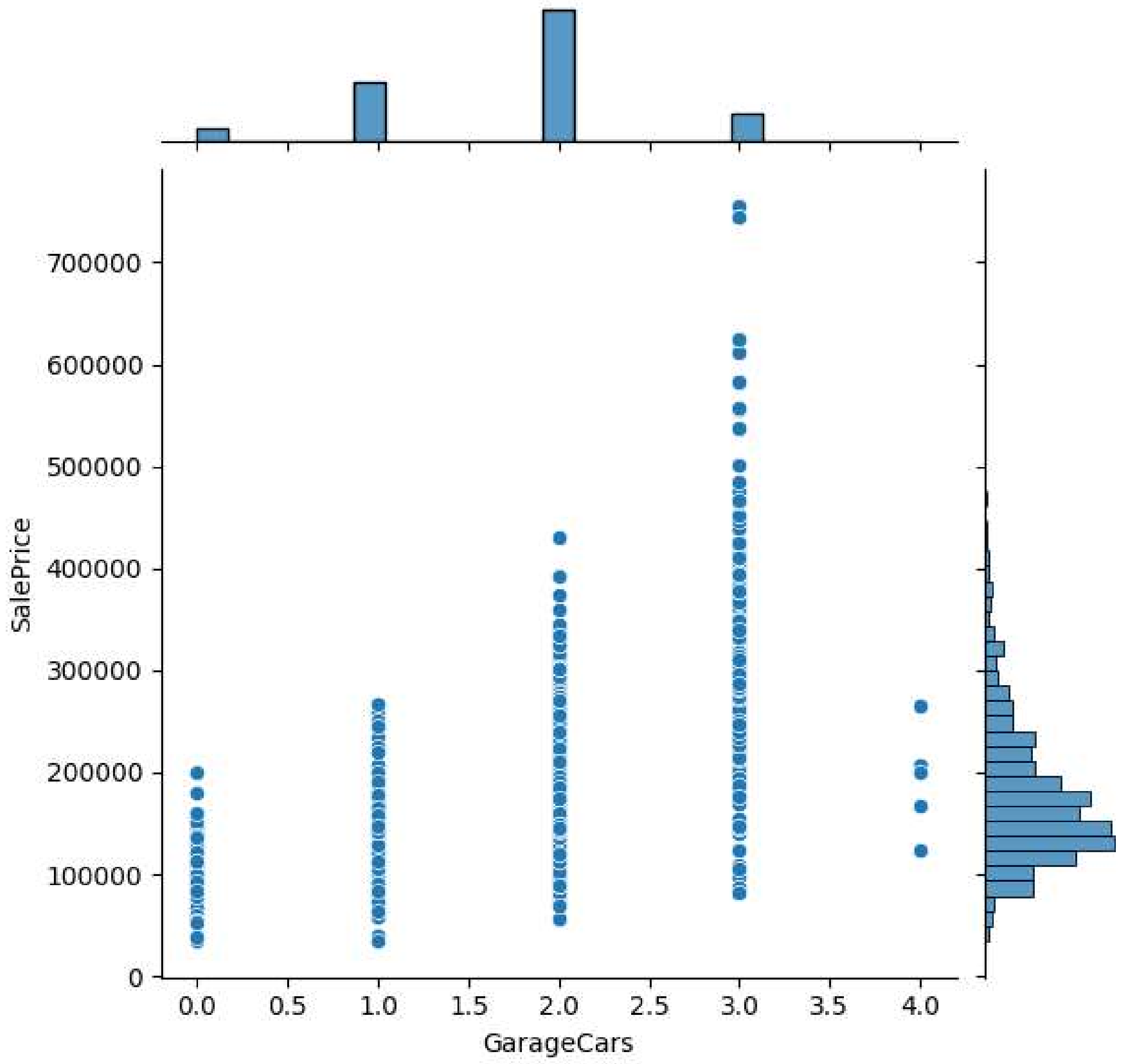


SalePrice
GrLivArea
0
100000
200000
300000
400000
500000
600000
700000
1000
2000
3000
4000
5000

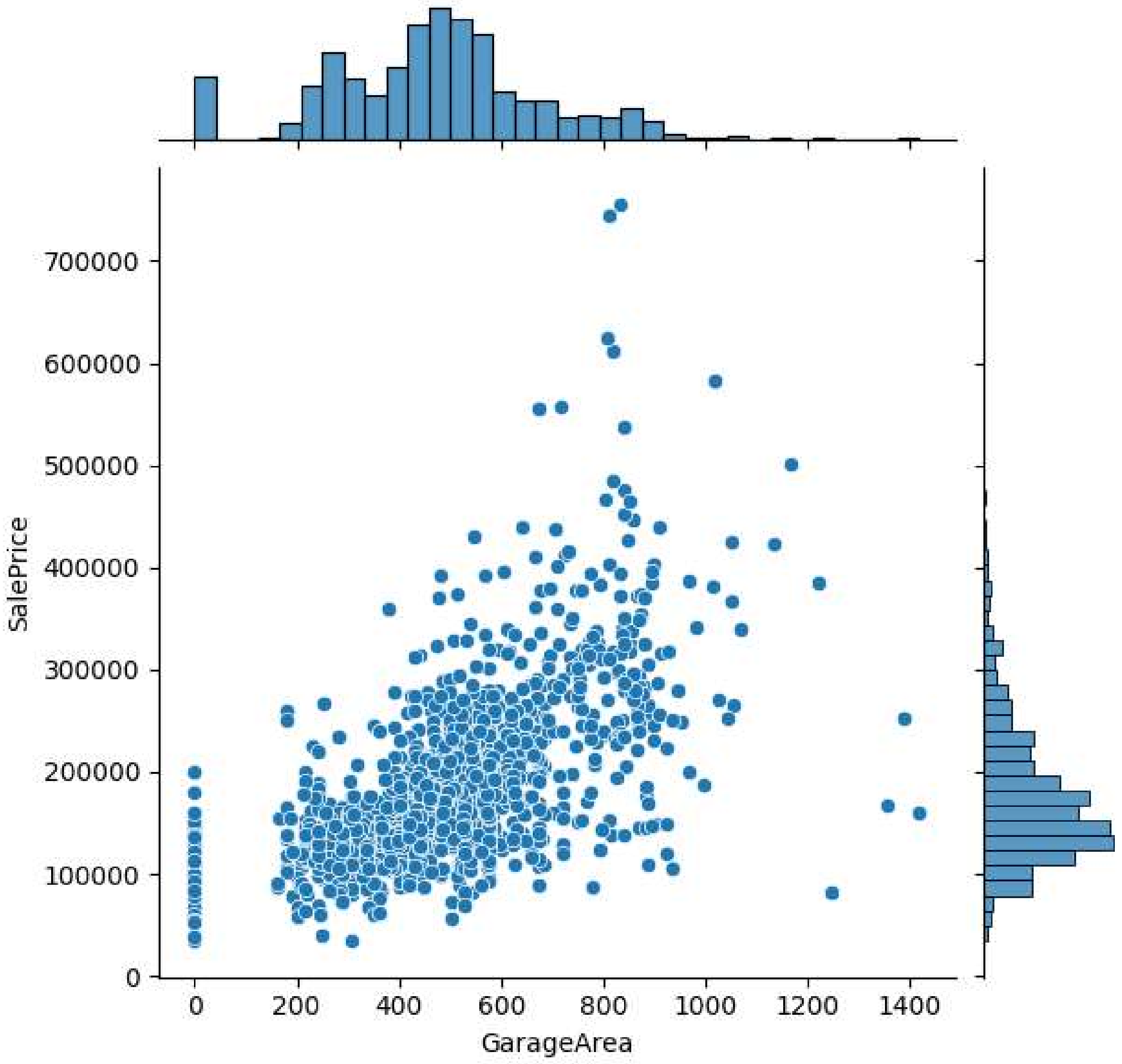

700000
600000
500000
400000
300000
200000
100000
0
SalePrice
0.0
0.5
1.0
1.5
2.0
2.5
3.0
FullBath

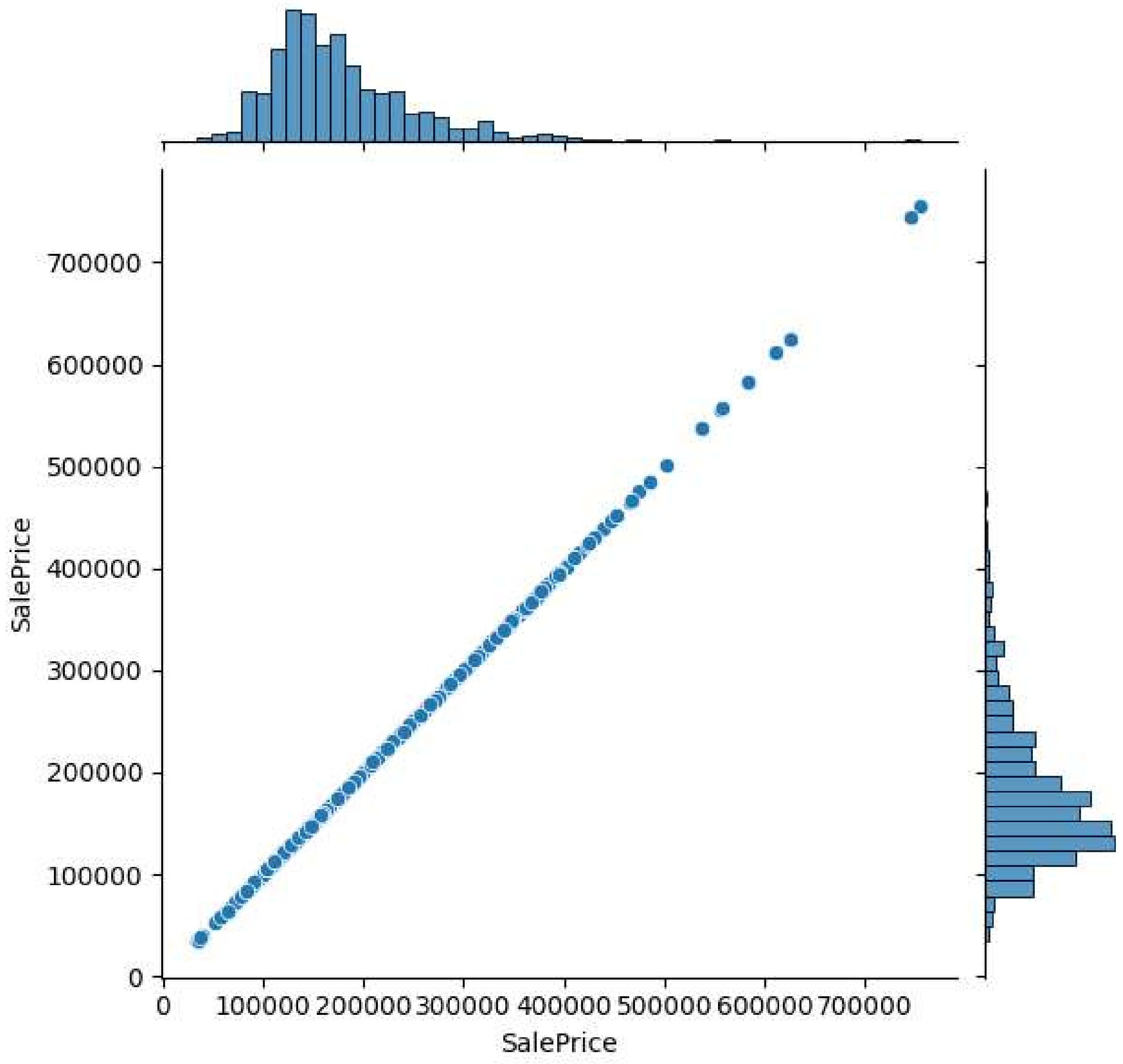


SalePrice
700000
600000
500000
400000
300000
200000
100000
0
2
4
6
8
10
12
14
TotRmsAbvGrd



SalePrice
GarageCars
0
100000
200000
300000
400000
500000
600000
700000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


SalePrice
GarageArea
0
100000
200000
300000
400000
500000
600000
700000
0
200
400
600
800
1000
1200
1400



One hot encoding

In [ ]:
```
df = pd.get_dummies(df, columns=cat_cols)
df
```

Out[ ]:

	OverallQual	YearBuilt	YearRemodAdd	TotalBsmtSF	1stFlrSF	GrLivArea	FullBath	TotRms
0	7	2003	2003	856	856	1710	2	
1	6	1976	1976	1262	1262	1262	2	
2	7	2001	2002	920	920	1786	2	
3	7	1915	1970	756	961	1717	1	
4	8	2000	2000	1145	1145	2198	2	
...	...	...	...	...	...	...	...	
1455	6	1999	2000	953	953	1647	2	
1456	6	1978	1988	1542	2073	2073	2	
1457	7	1941	2006	1152	1188	2340	2	
1458	5	1950	1996	1078	1078	1078	1	
1459	5	1965	1965	1256	1256	1256	1	

1460 rows × 42 columns

Scaling the numerical values

In [ ]:
```
important_num_cols.remove("SalePrice")

scaler = StandardScaler()
df[important_num_cols] = scaler.fit_transform(df[important_num_cols])
df
```

Out[ ]:

	OverallQual	YearBuilt	YearRemodAdd	TotalBsmtSF	1stFlrSF	GrLivArea	FullBath	TotR
0	0.651479	1.050994	0.878668	-0.459303	-0.793434	0.370333	0.789741	
1	-0.071836	0.156734	-0.429577	0.466465	0.257140	-0.482512	0.789741	
2	0.651479	0.984752	0.830215	-0.313369	-0.627826	0.515013	0.789741	
3	0.651479	-1.863632	-0.720298	-0.687324	-0.521734	0.383659	-1.026041	
4	1.374795	0.951632	0.733308	0.199680	-0.045611	1.299326	0.789741	
...	...	...	...	...	...	...	...	
1455	-0.071836	0.918511	0.733308	-0.238122	-0.542435	0.250402	0.789741	
1456	-0.071836	0.222975	0.151865	1.104925	2.355701	1.061367	0.789741	
1457	0.651479	-1.002492	1.024029	0.215641	0.065656	1.569647	0.789741	
1458	-0.795151	-0.704406	0.539493	0.046905	-0.218982	-0.832788	-1.026041	
1459	-0.795151	-0.207594	-0.962566	0.452784	0.241615	-0.493934	-1.026041	

1460 rows × 42 columns

Train Test Split

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

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
m_state=42)
```

Useful Metrics Calculations

In [ ]:
```
def evaluation(y, predictions):
    mae = mean_absolute_error(y, predictions)
    mse = mean_squared_error(y, predictions)
    rmse = np.sqrt(mean_squared_error(y, predictions))
    r_squared = r2_score(y, predictions)
    return mae, mse, rmse, r_squared
```

Modelling XG_Boost_Regressor

In [ ]:
```
xgb = XGBRegressor(n_estimators=1000, learning_rate=0.01)
xgb.fit(X_train, y_train)
predictions = xgb.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
```

```
MAE: 17985.48701038099
MSE: 837038411.4275047
RMSE: 28931.616121943563
R2 Score: 0.8908731664315402
```

Cross-Validation

In [ ]:
```
test_score = xgb.score(X_test, y_test)
test_score
```

Out[ ]:
```
0.8908731664315402
```

In [ ]:
```
k = 5
cross_val_scores = cross_val_score(xgb, X_train, y_train, cv=k)

print("Cross-Validation Scores:", cross_val_scores)
print("Mean CV Score:", np.mean(cross_val_scores))
```

```
Cross-Validation Scores: [0.82876146 0.67764145 0.86030628 0.85252966 0.86836
063]
Mean CV Score: 0.8175198945132556
```

In [ ]:
```
if np.mean(cross_val_scores) > test_score:
    print("The model may be overfitting.")
else:
    print("The model is not overfitting.")
```

```
The model is not overfitting.
```

In [ ]: