Data Analytics Final Project

Jacob Thomas

November 18, 2023

Abstract:

This housing prices dataset includes an abundant amount of a variety of features, both categorical and numerical, that one may use to analyze housing prices. The exploratory analysis of it will aim to factor out the useless features and locate the key features that I will be using to predict the sale price of a home. I will use a wide selection of visual tools such as scatterplots, boxplots, histograms, lineplots, heatmaps, countplots, and many others. Modeling will include simple techniques such as decision trees, random forests and k-nearest neighbors, then it will dive into more advanced models such as XGBoost and gradient boosting. I will be using RMSE and MAE for means of model evaluation.

Contents

Intro ... 3

About the data ... 3

Exploring Data ... 6

- LotFrontage vs SalePrice: ... 6
- LotArea vs SalePrice: ... 6
- YearBuilt vs SalePrice: ... 7
- BsmtQual vs SalePrice: ... 7
- BsmtCond vs SalePrice: ... 8
- TotalBsmtSf vs SalePrice: ... 8
- 1stFlrSF vs SalePrice: ... 9
- 2ndFlrSF vs SalePrice: ... 9
- PoolArea vs SalePrice: ... 10
- Fireplaces vs SalePrice: ... 10
- GarageCond vs SalePrice: ... 11
- OverallQual vs SalePrice ... 11

Modeling V1 ... 12

- Decision Tree Regression Model V1 ... 12
- Random Forest Regression Model V1 ... 12
- KNeighbors Regression Model V1 ... 12
- XGBoost Regression Model V1 ... 13
- Gradient Boosting Regression Model V1 ... 13
- Extra Trees Regression Model V1 ... 13
- AdaBoost Regression Model V1 ... 14
- Model V1 Conclusions ... 14

Modeling V2 ... 15

- Random Forest Regression Model V2 ... 19
- Extra Trees Regression Model V2 ... 19
- Gradient Boosting Regression Model V2 ... 19
- XGBoost Regression model V2 ... 20
- Modeling V2 Conclusions ... 20

Conclusion ... 21

Intro

Financial planning is crucial in today's day and age, and nothing breaks the bank more than buying a new home. Whether you're brand new to the housing market, a person who flips houses on the regular, or part of some financial agency, knowing the market is a must. The housing market is heavily data-driven, even if people don't realize it. For example, a realtor is easily able to predict the price of a home based on their experiences of selling past homes and what each one had to offer in terms of square feet, baths, etc. My models will act like someone with lots of experience in this field and be able to predict the prices of homes based on multiple features. The goal of this project is to be able to predict the sales price of a home based on numerous amounts of factors.

About the data

This data is from [Housing Prices Competition for Kaggle Learn Users | Kaggle](). It includes 1460 records and 81 variables about homes in Ames, Iowa. There are some missing values in the data,

- **SalePrice - the property's sale price in dollars. This is the target variable.**
- **MSSubClass: The building class**
- **MSZoning: The general zoning classification**
- **LotFrontage: Linear feet of street connected to property**
- **LotArea: Lot size in square feet**
- **Street: Type of road access**
- **Alley: Type of alley access**
- **LotShape: General shape of property**
- **LandContour: Flatness of the property**
- **Utilities: Type of utilities available**
- **LotConfig: Lot configuration**
- **LandSlope: Slope of property**
- **Neighborhood: Physical locations within Ames city limits**
- **Condition1: Proximity to main road or railroad**
- **Condition2: Proximity to main road or railroad (if a second is present)**
- **BldgType: Type of dwelling**
- **HouseStyle: Style of dwelling**
- **OverallQual: Overall material and finish quality**
- **OverallCond: Overall condition rating**
- **YearBuilt: Original construction date**
- **YearRemodAdd: Remodel date**
- **RoofStyle: Type of roof**
- **RoofMatl: Roof material**
- **Exterior1st: Exterior covering on house**
- **Exterior2nd: Exterior covering on house (if more than one material)**
- **MasVnrType: Masonry veneer type**
- **MasVnrArea: Masonry veneer area in square feet**

- **ExterQual: Exterior material quality**
- **ExterCond: Present condition of the material on the exterior**
- **Foundation: Type of foundation**
- **BsmtQual: Height of the basement**
- **BsmtCond: General condition of the basement**
- **BsmtExposure: Walkout or garden level basement walls**
- **BsmtFinType1: Quality of basement finished area**
- **BsmtFinSF1: Type 1 finished square feet**
- **BsmtFinType2: Quality of second finished area (if present)**
- **BsmtFinSF2: Type 2 finished square feet**
- **BsmtUnfSF: Unfinished square feet of basement area**
- **TotalBsmtSF: Total square feet of basement area**
- **Heating: Type of heating**
- **HeatingQC: Heating quality and condition**
- **CentralAir: Central air conditioning**
- **Electrical: Electrical system**
- **1stFlrSF: First Floor square feet**
- **2ndFlrSF: Second floor square feet**
- **LowQualFinSF: Low quality finished square feet (all floors)**
- **GrLivArea: Above grade (ground) living area square feet**
- **BsmtFullBath: Basement full bathrooms**
- **BsmtHalfBath: Basement half bathrooms**
- **FullBath: Full bathrooms above grade**
- **HalfBath: Half baths above grade**
- **Bedroom: Number of bedrooms above basement level**
- **Kitchen: Number of kitchens**
- **KitchenQual: Kitchen quality**
- **TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)**
- **Functional: Home functionality rating**
- **Fireplaces: Number of fireplaces**
- **FireplaceQu: Fireplace quality**
- **GarageType: Garage location**
- **GarageYrBlt: Year garage was built**
- **GarageFinish: Interior finish of the garage**
- **GarageCars: Size of garage in car capacity**
- **GarageArea: Size of garage in square feet**
- **GarageQual: Garage quality**
- **GarageCond: Garage condition**
- **PavedDrive: Paved driveway**
- **WoodDeckSF: Wood deck area in square feet**
- **OpenPorchSF: Open porch area in square feet**
- **EnclosedPorch: Enclosed porch area in square feet**
- **3SsnPorch: Three season porch area in square feet**
- **ScreenPorch: Screen porch area in square feet**
- **PoolArea: Pool area in square feet**
- **PoolQC: Pool quality**
- **Fence: Fence quality**
- **MiscFeature: Miscellaneous feature not covered in other categories**
- **MiscVal: $Value of miscellaneous feature**
- **MoSold: Month Sold**
- **YrSold: Year Sold**
- **SaleType: Type of sale**
- **SaleCondition: Condition of sale**

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	1460.000000	1460.000000
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.639726	46.549315	567.240411
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.098091	161.319273	441.866955
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	0.000000	0.000000
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.000000	0.000000	223.000000
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000	0.000000	477.500000
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.250000	0.000000	808.000000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000	2336.000000

	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	BedroomAbvGr	KitchenAbvGr	TotRmsAbvGrd
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000
mean	1057.429452	1162.626712	346.992466	5.844521	1515.463699	0.425342	0.057534	1.565068	0.382877	2.866438	1.046575	6.517808
std	438.705324	386.587738	436.528436	48.623081	525.480383	0.518911	0.238753	0.550916	0.502885	0.815778	0.220338	1.625393
min	0.000000	334.000000	0.000000	0.000000	334.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.000000
25%	795.750000	882.000000	0.000000	0.000000	1129.500000	0.000000	0.000000	1.000000	0.000000	2.000000	1.000000	5.000000
50%	991.500000	1087.000000	0.000000	0.000000	1464.000000	0.000000	0.000000	2.000000	0.000000	3.000000	1.000000	6.000000
75%	1298.250000	1391.250000	728.000000	0.000000	1776.750000	1.000000	0.000000	2.000000	1.000000	3.000000	1.000000	7.000000
max	6110.000000	4692.000000	2065.000000	572.000000	5642.000000	3.000000	2.000000	3.000000	2.000000	8.000000	3.000000	14.000000

	Fireplaces	GarageYrBlt	GarageCars	GarageArea	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	PoolArea	MiscVal	MoSold
count	1460.000000	1379.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000
mean	0.613014	1978.506164	1.767123	472.980137	94.244521	46.660274	21.954110	3.409589	15.060959	2.758904	43.489041	6.321918
std	0.644666	24.689725	0.747315	213.804841	125.338794	66.256028	61.119149	29.317331	55.757415	40.177307	496.123024	2.703626
min	0.000000	1900.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	0.000000	1961.000000	1.000000	334.500000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	5.000000
50%	1.000000	1980.000000	2.000000	480.000000	0.000000	25.000000	0.000000	0.000000	0.000000	0.000000	0.000000	6.000000
75%	1.000000	2002.000000	2.000000	576.000000	168.000000	68.000000	0.000000	0.000000	0.000000	0.000000	0.000000	8.000000
max	3.000000	2010.000000	4.000000	1418.000000	857.000000	547.000000	552.000000	508.000000	480.000000	738.000000	15500.000000	12.000000

	YrSold	SalePrice
count	1460.000000	1460.000000
mean	2007.815753	180921.195890
std	1.328095	79442.502883
min	2006.000000	34900.000000
25%	2007.000000	129975.000000
50%	2008.000000	163000.000000
75%	2009.000000	214000.000000
max	2010.000000	755000.000000

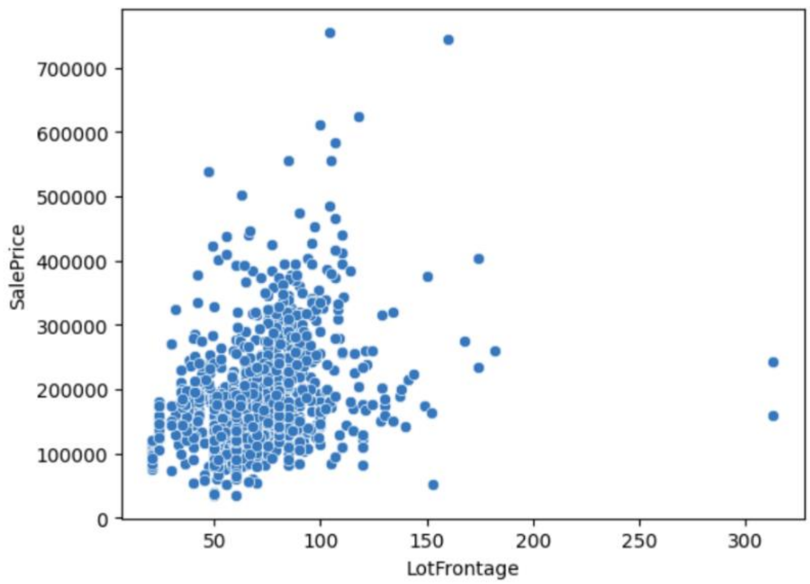
```
train.shape
```

```
(1460, 81)
```

Exploring Data

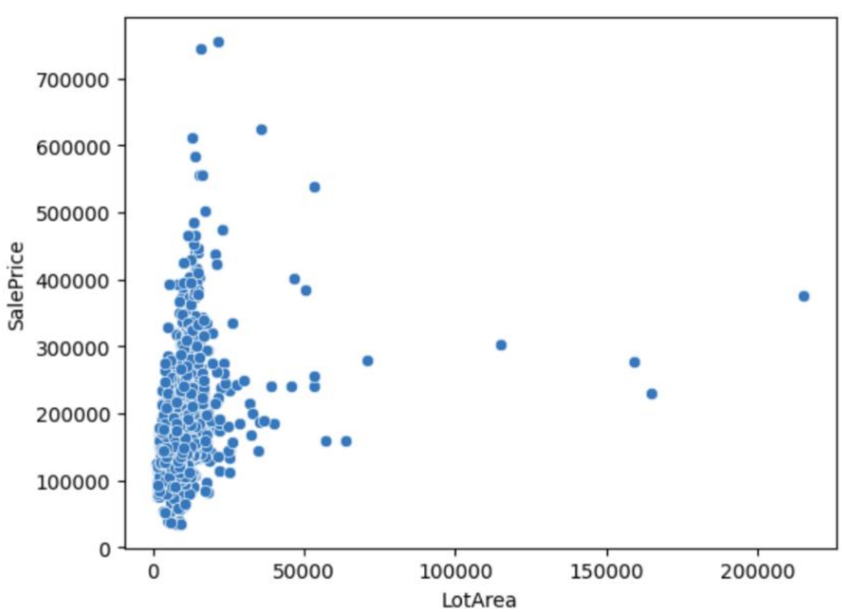
LotFrontage vs SalePrice:

Feet of street connected to property vs sale price



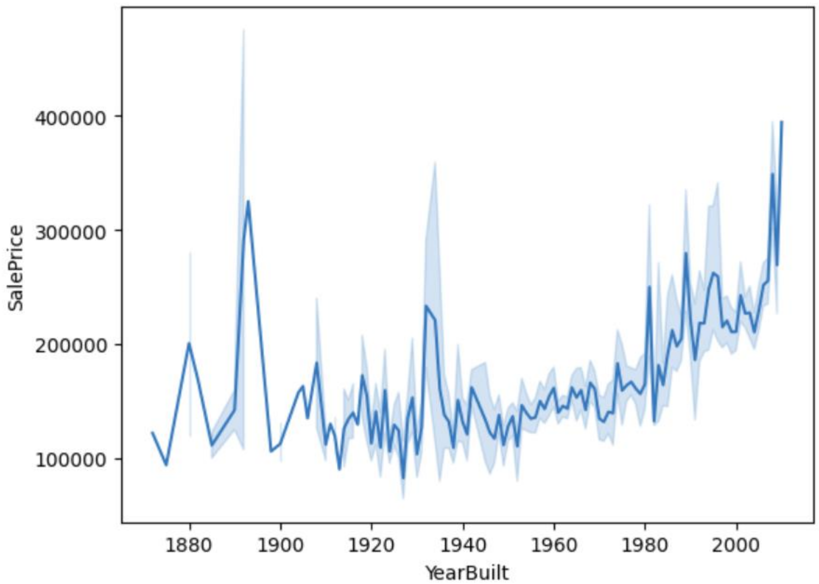
LotArea vs SalePrice:

Lot size in square feet vs sale price



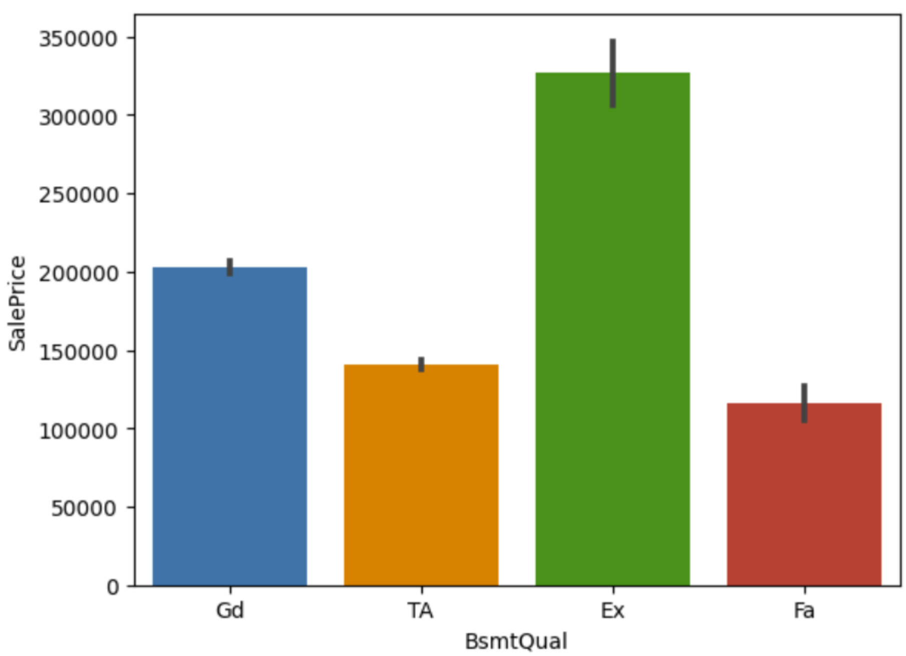
YearBuilt vs SalePrice:

Year the house was built in vs sale price. We see a gradual incline after the 1970s.



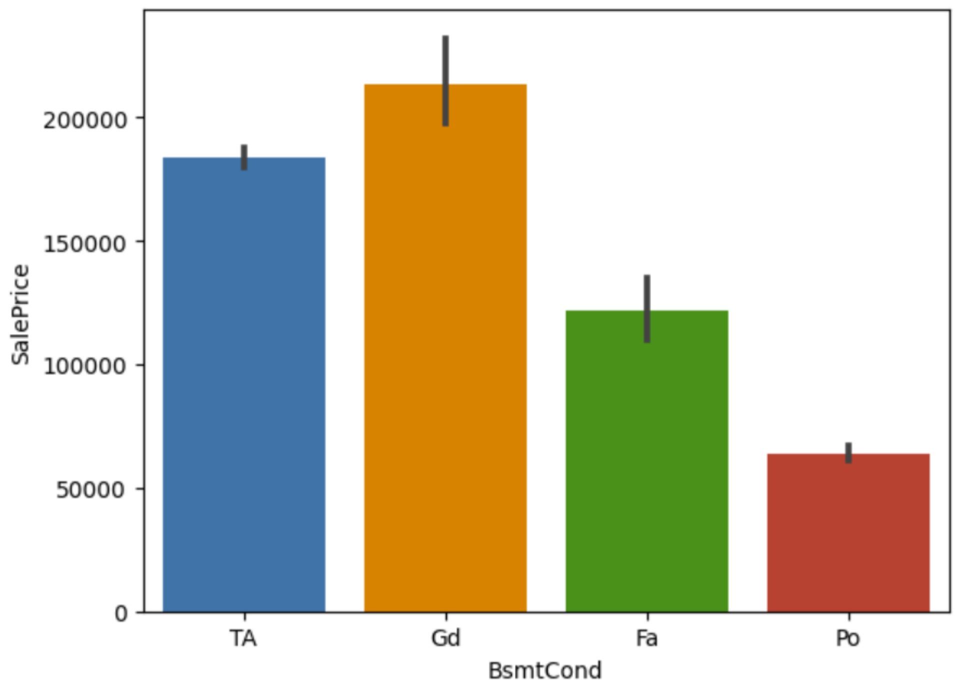
BsmtQual vs SalePrice:

The height of the basement vs sale price. We see an excellent quality (100+ inches tall) sells for significantly higher



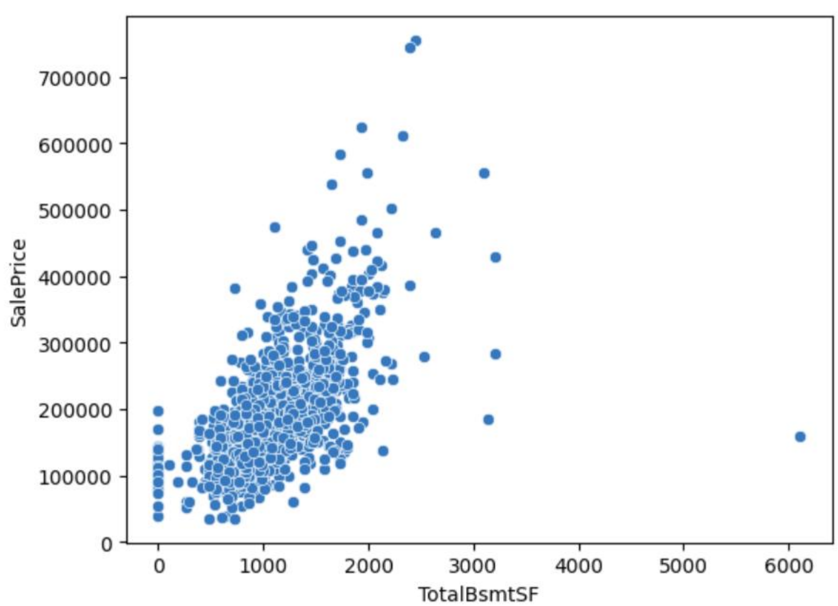
BsmtCond vs SalePrice:

The condition of the basement vs sale price. Poor (Po) basement quality drops the price by a lot



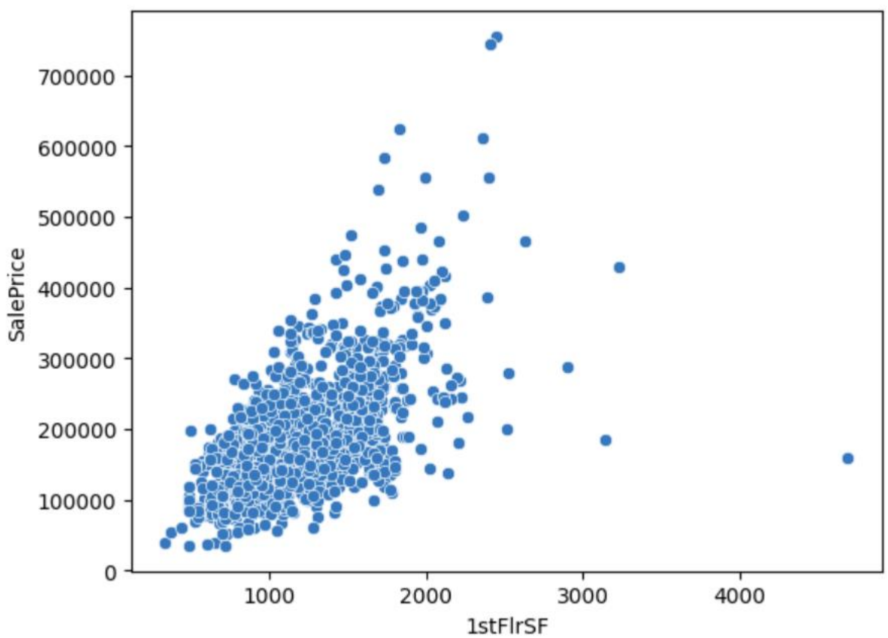
TotalBsmtSf vs SalePrice:

The total square feet of the basement vs sale price



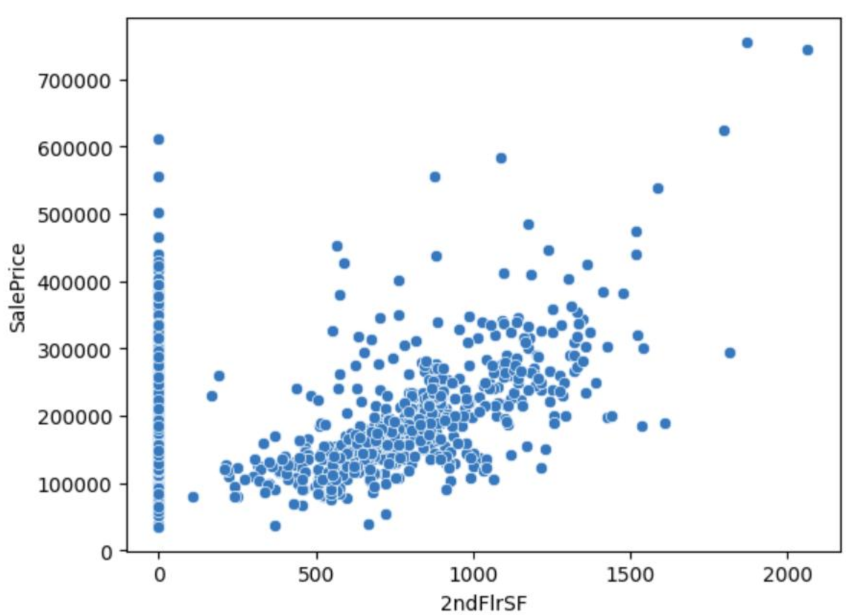
1stFlrSF vs SalePrice:

1st floor square feet vs sale price



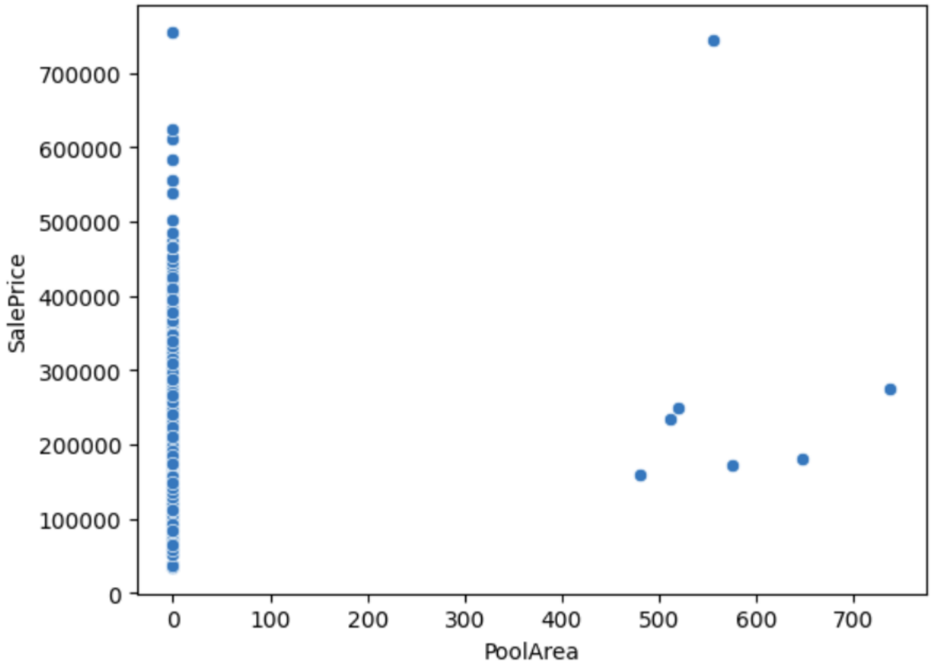
2ndFlrSF vs SalePrice:

2nd floor square feet vs sale price. Some homes won't have a second floor



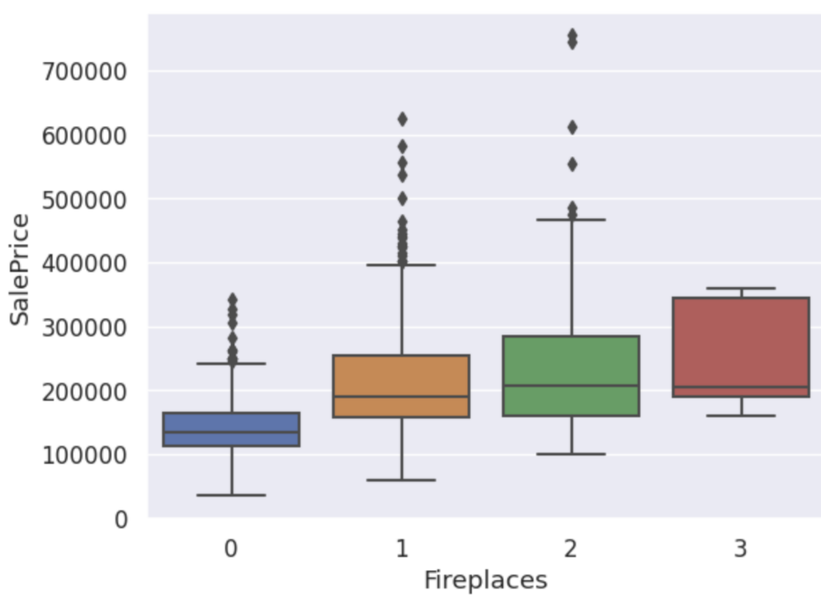
PoolArea vs SalePrice:

Pool area in square feet vs sale price. Not a lot of pools, not a good predicting variable.



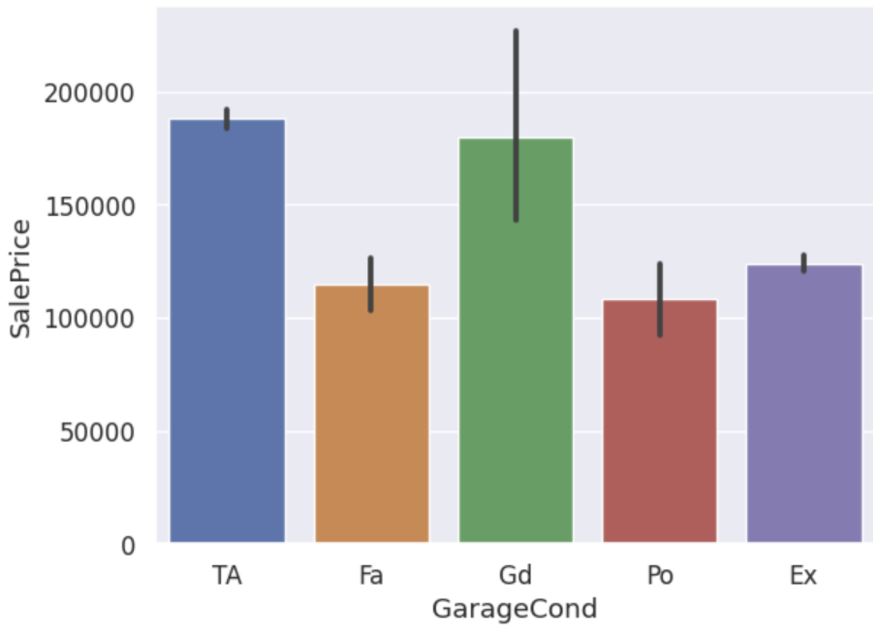
Fireplaces vs SalePrice:

The number of fireplaces vs sale price. We see on average the price goes up



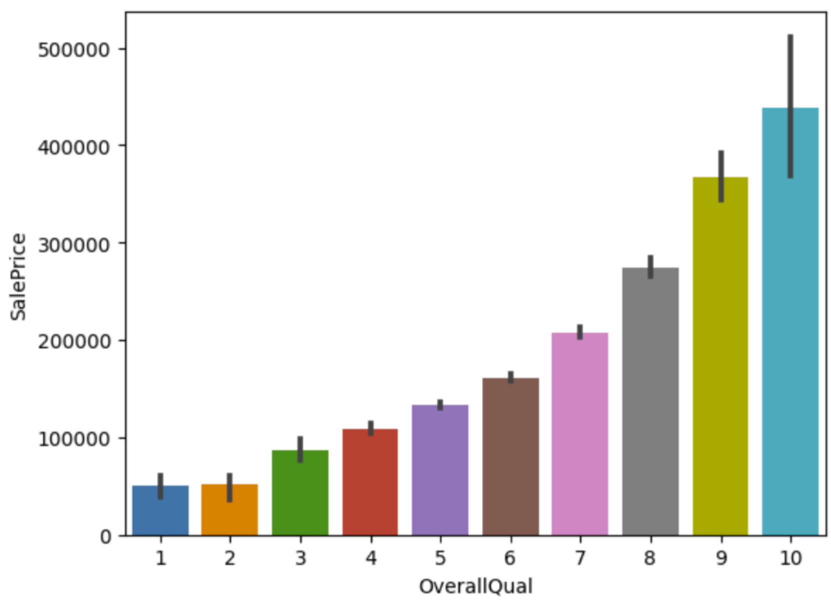
GarageCond vs SalePrice:

Garage condition vs sale price. Not a good predictor since Ex (excellent condition) is almost even with the poorest condition and fair condition. Average and good are weirdly selling for more



OverallQual vs SalePrice

The overall material and finish of the house vs sale price. Very positive relationship, quality goes up, then price goes up.



Modeling V1

Split the data into 80% train and 20% test. Target variable is SalePrice. All of these models will contain only numeric and non-problematic data (no missing values). There is no hyperparameter tuning or feature selection for the initial models, I want to see how they compare.

Decision Tree Regression Model V1

```
dt_md = DecisionTreeRegressor(random_state = 42)

#Fit dt model
dt_md.fit(X_train, Y_train)

#Predict
dt_pred = (dt_md.predict(X_test))

#MSE and MAE
print(f"The MSE of the Decision Tree model is {mean_squared_error(Y_test, dt_pred, squared = False)}")
print(f"The MAE of the Decision Tree model is {mean_absolute_error(Y_test, dt_pred)}")
```

```
The MSE of the Decision Tree model is 40088.6296067423
The MAE of the Decision Tree model is 26362.77397260274
```

Random Forest Regression Model V1

```
rf_md = RandomForestRegressor(random_state = 42)

#Fit rf model
rf_md.fit(X_train, Y_train)

#Predict
rf_pred = (rf_md.predict(X_test))

#MSE and MAE
print(f"The MSE of the Random Forest model is {mean_squared_error(Y_test, rf_pred, squared = False)}")
print(f"The MAE of the Random Forest model is {mean_absolute_error(Y_test, rf_pred)}")
```

```
The MSE of the Random Forest model is 30578.81927422495
The MAE of the Random Forest model is 19412.62626712329
```

KNeighbors Regression Model V1

```
#Fit knn model
knn_md = KNeighborsRegressor().fit(X_train, Y_train)

#Predict
knn_pred = knn_md.predict(X_test)

print(f"The MSE of the knn model is {mean_squared_error(Y_test, knn_pred, squared = False)}")
print(f"The MAE of the knn model is {mean_absolute_error(Y_test, knn_pred)}")
```

```
The MSE of the knn model is 56061.868756054646
The MAE of the knn model is 34496.572602739725
```

XGBoost Regression Model V1

```
#Fit xgb model
xgb_md = XGBRegressor(random_state = 42).fit(X_train, Y_train)

#Predict
xgb_pred = xgb_md.predict(X_test)

print(f"The MSE of the xgb model is {mean_squared_error(Y_test, xgb_pred, squared = False)}")
print(f"The MAE of the xgb model is {mean_absolute_error(Y_test, xgb_pred)}")
```

```
The MSE of the xgb model is 31472.14500881965
The MAE of the xgb model is 19744.879454730308
```

Gradient Boosting Regression Model V1

```
#Fit gb model
gb_md = GradientBoostingRegressor(random_state = 42).fit(X_train, Y_train)

#Predict
gb_pred = gb_md.predict(X_test)

print(f"The MSE of the gb model is {mean_squared_error(Y_test, gb_pred, squared = False)}")
print(f"The MAE of the gb model is {mean_absolute_error(Y_test, gb_pred)}")
```

```
The MSE of the gb model is 29377.712223096038
The MAE of the gb model is 19494.582560415183
```

Extra Trees Regression Model V1

```
#Fit et model
et_md = ExtraTreesRegressor(random_state = 42).fit(X_train, Y_train)

#Predict
et_pred = et_md.predict(X_test)

print(f"The MSE of the et model is {mean_squared_error(Y_test, et_pred, squared = False)}")
print(f"The MAE of the et model is {mean_absolute_error(Y_test, et_pred)}")
```

```
The MSE of the et model is 28114.369108441155
The MAE of the et model is 18224.241952054792
```

AdaBoost Regression Model V1

```
#Fit ada model
ada_md = AdaBoostRegressor(random_state = 42).fit(X_train, Y_train)

#Predict
ada_pred = ada_md.predict(X_test)

print(f"The MSE of the ada model is {mean_squared_error(Y_test, ada_pred, squared = False)}")
print(f"The MAE of the ada model is {mean_absolute_error(Y_test, ada_pred)}")
```

```
The MSE of the ada model is 37641.267194955624
The MAE of the ada model is 26453.934049700423
```

Model V1 Conclusions

My Random Forest, XGBoost, Gradient Boost, and Extra Trees models performed the best, while the Decision Tree and AdaBoost models didn’t do so well. The KNeighbors model did poorly, so we won’t focus on it or the other two for future modeling. The Extra Trees model performed the best with an MAE of 18,224.

Modeling V2

For version 2 of these models, we will focus on the 4 that did the best in version 1 of our modeling. What's different is we will drop missing values, do some feature selection, and finally some hyperparameter tuning.

```
#What columns contain null values
columns = []
for col in train.columns:
    if train[col].isnull().sum() > 0 :
        columns.append(col)
        print(col)
        #print(train[col].unique())
```

```
LotFrontage
Alley
MasVnrType
MasVnrArea
BsmtQual
BsmtCond
BsmtExposure
BsmtFinType1
BsmtFinType2
Electrical
FireplaceQu
GarageType
GarageYrBlt
GarageFinish
GarageQual
GarageCond
PoolQC
Fence
MiscFeature
```

```
#Drop null value columns
train = train.drop(columns = ['LotFrontage', 'Alley', 'MasVnrType', 'MasVnrArea', 'BsmtQual', 'BsmtCond', 'BsmtExposure',
                              'BsmtFinType1', 'BsmtFinType2', 'Electrical', 'FireplaceQu', 'GarageType', 'GarageYrBlt',
                              'GarageFinish', 'GarageQual', 'GarageCond', 'PoolQC', 'Fence', 'MiscFeature'], axis = 1)

X = pd.concat([pd.get_dummies(train)])
```

Above, we dropped missing values.

```python
#Store variable importance here
var_importance = list()

#Repeat 25 times
for i in range(0, 25):

    #Input and target
    X = pd.get_dummies(train.drop(columns = ['SalePrice', 'Id']))
    Y = train['SalePrice']

    #Split data
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = .2)


    ### Decision Tree ###
    dt_md = DecisionTreeRegressor().fit(X_train, Y_train)

    #Extract importance
    var_importance.append(dt_md.feature_importances_)


    ### Random Forest ###
    rf_md = RandomForestRegressor().fit(X_train, Y_train)

    #Extract importance
    var_importance.append(rf_md.feature_importances_)


    ### XGBoost ###
    xgb_md = XGBRegressor().fit(X_train, Y_train)

    #Extract importance
    var_importance.append(xgb_md.feature_importances_)
```

```python
    ### Gradient Boosting ###
    gb_md = GradientBoostingRegressor().fit(X_train, Y_train)

    #Extract importance
    var_importance.append(gb_md.feature_importances_)


    ### Extra Trees ###
    et_md = ExtraTreesRegressor().fit(X_train, Y_train)

    #Extract importance
    var_importance.append(et_md.feature_importances_)


    ### AdaBoost ###
    ada_md = AdaBoostRegressor().fit(X_train, Y_train)

    #Extract importance
    var_importance.append(ada_md.feature_importances_)



#Variable Importance Scores
var_importance = pd.DataFrame(var_importance, columns = X.columns)
var_importance = pd.DataFrame(var_importance.mean()).T
var_importance.sort_values(by='OverallQual', ascending = False)
```

	MSSubClass	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	BedroomAbvGr	KitchenAbvGr	TotRmsAbvGrd	Fireplaces
0	0.001178	0.013571	0.426864	0.003731	0.010095	0.010013	0.026143	0.000987	0.003535	0.038551	0.018127	0.043565	0.00031	0.095024	0.002742	0.000474	0.013521	0.002795	0.003666	0.005606	0.009696	0.010182

Above, we found the most important features.

```
def expand_grid(dictionary):
    return pd.DataFrame([row for row in product(*dictionary.values())],
                        columns = dictionary.keys())


### Random Forest ###
rf_dictionary = {'n_estimators': [50, 100, 125, 200, 250, 500, 1000],
                 'max_depth': [1,2,3,4,5,6,7,8,9,10]}

rf_parameters = expand_grid(rf_dictionary)
rf_parameters['MSE'] = np.nan
rf_parameters['MAE'] = np.nan


### Extra Trees ###
et_dictionary = {'n_estimators': [50, 100, 125, 200, 250, 500, 1000],
                 'max_depth': [1,2,3,4,5,6,7,8,9,10]}

et_parameters = expand_grid(et_dictionary)
et_parameters['MSE'] = np.nan
et_parameters['MAE'] = np.nan


### Gradient Boosting ###
gb_dictionary = {'n_estimators': [50, 100, 125, 200, 250, 500, 1000],
                 'max_depth': [1,2,3,4,5,6,7,8,9,10],
                 'learning_rate': [0.1, 0.01, 0.001, .25, .5, .8, .05]}

gb_parameters = expand_grid(gb_dictionary)
gb_parameters['MSE'] = np.nan
gb_parameters['MAE'] = np.nan


### XGBoost ###
xgb_dictionary = {'n_estimators': [50, 100, 125, 200, 250, 500, 1000],
                  'max_depth': [1,2,3,4,5,6,7,8,9,10],
                  'eta': [0.1, 0.01, 0.001, .25, .5, .8, .05, .25, .3, .4]}

xgb_parameters = expand_grid(xgb_dictionary)
xgb_parameters['MSE'] = np.nan
xgb_parameters['MAE'] = np.nan
```

The code above is the best hyperparameters we want to find for each model, and below, is training the version 2 models using the new features we picked. There are 3 more models just like below that are finding the best hyperparameters, just changing variable names.

```
#Input and target
X = pd.get_dummies(train[['MSSubClass', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'BsmtFinSF1',
                          'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea',
                          'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
                          'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
                          'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'MSZoning',
                          'LotShape', 'LandContour', 'LotConfig', 'LandSlope']])
Y = train['SalePrice']

#Store results here
rf_results = []

#Run 10 times
#for j in range(0, 10):

#Split data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = .2)

num_models = rf_parameters.shape[0]
for i in range(0, num_models):

    #Random Forest
    rf_md = RandomForestRegressor(max_depth = rf_parameters.loc[i, 'max_depth'],
                                  n_estimators = rf_parameters.loc[i, 'n_estimators']).fit(X_train, Y_train)

    #Predict
    rf_pred = rf_md.predict(X_test)

    #MSE and MAE
    rf_results.append([rf_parameters.loc[i, 'max_depth'], rf_parameters.loc[i, 'n_estimators'],
                       mean_squared_error(Y_test, rf_pred, squared = False), mean_absolute_error(Y_test, rf_pred)])
```

Random Forest best hyperparameters

	max_depth	n_estimators	MSE	MAE	mse_mae
0	10	125	31482.517730	18562.596929	50045.114660
1	10	500	31841.073280	18651.694223	50492.767503
2	10	1000	31598.092310	18661.276450	50259.368761
3	10	250	31456.653498	18684.843651	50141.497148
4	10	50	32177.527552	18721.670196	50899.197748

Extra Trees best hyperparameters

	max_depth	n_estimators	MSE	MAE	mse_mae
0	10	125	33958.240704	17880.019550	51838.260254
1	9	125	32205.030525	17952.186372	50157.216897
2	10	200	33215.166919	18089.292253	51304.459172
3	10	1000	33698.166440	18182.618575	51880.785015
4	10	250	33494.756924	18278.960801	51773.717726

Gradient Boosting best hyperparameters

	max_depth	n_estimators	learning_rate	MSE	MAE	mse_mae
0	3	500	0.05	27148.015299	17191.174329	44339.189628
1	3	1000	0.05	27059.842935	17239.267694	44299.110629
2	3	250	0.05	27628.712590	17381.292191	45010.004781
3	3	200	0.10	27845.266541	17442.741743	45288.008284
4	3	100	0.25	26668.171962	17478.766886	44146.938848

XGBoost best hyperparameters

	max_depth	n_estimators	eta	MSE	MAE	mse_mae
0	5	1000	0.01	23682.626412	15197.281411	38879.907822
1	5	500	0.10	23710.701187	15232.845034	38943.546222
2	5	1000	0.10	23755.540561	15257.220288	39012.760850
3	5	250	0.10	23773.907370	15317.504120	39091.411490
4	5	200	0.10	23862.391712	15419.747726	39282.139438

Random Forest Regression Model V2

```
X = pd.get_dummies(train[['MSSubClass', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'BsmtFinSF1',
                          'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea',
                          'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
                          'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
                          'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'MSZoning',
                          'LotShape', 'LandContour', 'LotConfig', 'LandSlope']])
Y = train['SalePrice']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = .2)

rf_md = RandomForestRegressor(max_depth = 10, n_estimators = 125).fit(X_train, Y_train)

rf_pred = rf_md.predict(X_test)

print(f"The MSE of the rf model is {mean_squared_error(Y_test, rf_pred, squared = False)}")
print(f"The MAE of the rf model is {mean_absolute_error(Y_test, rf_pred)}")
```

```
The MSE of the rf model is 34670.69428995096
The MAE of the rf model is 19127.288432784444
```

Extra Trees Regression Model V2

```
X = pd.get_dummies(train[['MSSubClass', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'BsmtFinSF1',
                          'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea',
                          'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
                          'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
                          'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'MSZoning',
                          'LotShape', 'LandContour', 'LotConfig', 'LandSlope']])
Y = train['SalePrice']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = .2)

et_md = ExtraTreesRegressor(max_depth = 10, n_estimators = 125).fit(X_train, Y_train)

et_pred = et_md.predict(X_test)

print(f"The MSE of the et model is {mean_squared_error(Y_test, et_pred, squared = False)}")
print(f"The MAE of the et model is {mean_absolute_error(Y_test, et_pred)}")
```

```
The MSE of the et model is 24833.873710586413
The MAE of the et model is 16481.381664312394
```

Gradient Boosting Regression Model V2

```
X = pd.get_dummies(train[['MSSubClass', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'BsmtFinSF1',
                          'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea',
                          'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
                          'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
                          'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'MSZoning',
                          'LotShape', 'LandContour', 'LotConfig', 'LandSlope']])
Y = train['SalePrice']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = .2)

gb_md = GradientBoostingRegressor(max_depth = 3, n_estimators = 500, learning_rate = .05).fit(X_train, Y_train)

gb_pred = gb_md.predict(X_test)

print(f"The MSE of the gb model is {mean_squared_error(Y_test, gb_pred, squared = False)}")
print(f"The MAE of the gb model is {mean_absolute_error(Y_test, gb_pred)}")
```

```
The MSE of the gb model is 33414.58654066964
The MAE of the gb model is 17019.61061834213
```

XGBoost Regression model V2

```
X = pd.get_dummies(train[['MSSubClass', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'BsmtFinSF1',
                          'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea',
                          'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
                          'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
                          'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'MSZoning',
                          'LotShape', 'LandContour', 'LotConfig', 'LandSlope']])
Y = train['SalePrice']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = .2)

xgb_md = XGBRegressor(max_depth = 5, n_estimators = 1000, eta = .01).fit(X_train, Y_train)

xgb_pred = xgb_md.predict(X_test)

print(f"The MSE of the xgb model is {mean_squared_error(Y_test, xgb_pred, squared = False)}")
print(f"The MAE of the xgb model is {mean_absolute_error(Y_test, xgb_pred)}")
```

```
The MSE of the xgb model is 25123.569594107586
The MAE of the xgb model is 16397.58201787243
```

Modeling V2 Conclusions

For simplicity sake, I will compare the results side by side. On the left is version 1, on the right is version 2. As we can see, the results of the models have improved. The model with the lowest MAE is now the XGBoost model.

Random Forest

The MSE of the Random Forest model is 30578.81927422495 The MAE of the Random Forest model is 19412.62626712329	The MSE of the rf model is 34670.69428995096 The MAE of the rf model is 19127.288432784444

XGBoost

The MSE of the xgb model is 31472.14500881965 The MAE of the xgb model is 19744.879454730308	The MSE of the xgb model is 25123.569594107586 The MAE of the xgb model is 16397.58201787243

Gradient Boosting

The MSE of the gb model is 29377.712223096038 The MAE of the gb model is 19494.582560415183	The MSE of the gb model is 33414.58654066964 The MAE of the gb model is 17019.61061834213

Extra Trees

The MSE of the et model is 28114.369108441155 The MAE of the et model is 18224.241952054792	The MSE of the et model is 24833.873710586413 The MAE of the et model is 16481.381664312394

Conclusion

In this project, I explored the Housing Prices dataset using visualizations such as scatterplots, bar plots, lineplots, boxplots, and a vast variety of regression models like random forest, extra trees, xgboost, and much more. I found OverallQual to be the main crucial feature out of all the columns through data visualization, plus a few other smaller ones. I made some very simple models to begin with using numerical data that had no missing values, to see how each model compared to one another without going too in-depth. Then, for the more advanced models, I decided to drop the columns with missing values and narrow down the number of input variables in my models using feature selection. I plugged in the new input data into the top four models from my first modeling version, then decided to loop some hyperparameter data into the models to see what hyperparameters are best. After finding the optimal hyperparameters, I plugged them into the models and used RMSE and MAE as the way to evaluate their performance compared to the first models. After comparing the results from both versions of the models, version two clearly has a lower MAE, meaning that it is better.