Member-only story

WRANGLING THROUGH DATALAND

Cracking the Ames Housing Dataset with Linear Regression

Regression modelling for prediction and statistical inference

Alvin T. Tan, Ph.D. · Follow

Published in Towards Data Science · 14 min read · Nov 13, 2019

Image of Ames, Iowa by Tim Kiser from Creative Commons

The feature richness of the Ames housing dataset (2011) is both alluring and bewildering in equal measure. It is easy to become entagled in its bountiful features while trying to uncover its patterns. It is first and

foremost useful to understand that the Ames dataset fits into the long-established *hedonic pricing method* to analyzing housing prices. Some **domain knowledge** will go a long way.

I had previously studied statistics/econometrics in some detail. Cognizant of the "big data" revolution and intrigued by its promise, I have immersed myself in coding and machine learning these past few months. It was in this process that I encountered the Ames housing dataset. I approached it with a combination of **traditional econometric research and machine learning** tools.

I chose to stick to **transparent linear models** where the goals of both prediction and statistical inference may be pursued together. This meant running ordinary least squares (OLS), Ridge and Lasso regression models. Please note that the analysis I undertook was on the full **original** Ames dataset, and not the Kaggle version of the dataset.

Hedonic Pricing Method

In econometric research, one attempts to expand research horizons by standing on the shoulders of previous researchers in the subject matter. No two houses are exactly identical, and the basic idea of *hedonic price* modeling is that neighborhood-specific and unit-specific characteristics help determine house prices.

It is therefore useful to regurgitate the usual suspects affecting house prices. If the dependent (or target) variable is the sale price, then naturally the larger the house the higher the price, all else being equal. That should be as self-evident as asserting that an 18-inch pizza will probably cost more than a 12-inch pizza. So **property size measures** will be key variables, and there are several such variables in the Ames dataset. A more interesting question

might be how and why the price per square foot might differ from house to house, which is like asking why certain pizzerias are able to charge a higher price per slice. Real estate professionals too tend to focus on price per square foot (or meter) instead of the overall price. But that is not the goal in this article.

Any home buyer would have heard of the expression **"location, location, location"**, which also happens to be the title of a popular British TV show on house buyers that has been on air for 19 consecutive years. So one would expect neighborhood "location" and associated characteristics, such as access to amenities (e.g. good schools, leisure facilities), transport networks (e.g. near a metro station), neighborhood aesthetics (e.g. tree-lined streets, handsome houses), and socioeconomic prestige among others, to have an impact on house prices.

Another group of factors commonly said to be important for determining housing prices is the number of rooms, particularly bedrooms and bathrooms, and the condition of the kitchen. Beyond this, the age and physical condition of the house are also important, as are the construction materials and any structural improvements to the property.

The factors that effect housing prices may be summarized as **property size, location desirability, nearby amenities, number of rooms, construction materials, and age and condition of the structure**. Armed with this prior research, I took to analyzing the data using Python.

Data Cleaning & Outliers

The first task was data cleaning, as ever. The dataset had 2,930 observations initially, and I immediately dropped **three variables** that had less than 300 observations each. The "LotFrontage" (linear feet of street connected to

property) variable, which I thought could be important, was missing 490 observations. I filled the missing values with the average “LotFrontage” when grouped by their respective “LotShape” categories.

```
# Notice a large number of null values in 'LotFrontage'. Examine the 'Lots' grouping.
df_lots = df[['LotFrontage', 'LotArea', 'LotConfig', 'LotShape']]
grouped_lots = df_lots.groupby(['LotShape'])
grouped_lots.mean()
```

LotShape	LotFrontage	LotArea
IR1	74.768760	11506.414959
IR2	67.437500	18913.065789
IR3	117.636364	32064.187500
Reg	66.821387	8849.153301

Assign missing ‘LotFrontage’ by the mean value according to ‘LotShape’

I then rendered a histogram of the **dependent variable**, “SalePrice”. It was immediately obvious that it had extreme high values (a long right-tail) and was not distributed normally. This suggested that the data contained a lot of **outliers**. Dealing with outliers can be tricky in that some of them may provide important information, so one doesn’t want to define outliers too liberally and then chuck them all out.

Very skewed sale price target variable

Given the importance of **property size metrics,** I quickly zoomed in on "GrLivArea", "LotSize" and "GarageArea" as potential explanatory variables. It is uncommon in the US to include the basement square footage in the measure of a property's size. In fact, Fannie Mae and ANSI guidelines forbid the counting of basement square footage in appraising a property's living area, though one can imagine that it is still likely to exert some impact on property prices. I consequently chose to separate the basement size measure, and constructed a finished basement area variable, "BaseLivArea", from other basement measures.

It was now time to search for **outliers.** I looked for those observations in "SalePrice", "GrLivArea" and "BaseLivArea" respectively that were either under or above **3 standard deviations** away from their respective mean values. There were no observations in the "under" category, but there were **66 observations** in the "above" category across all three variables. I deleted

these 66 observations, which comprised approximately 2.3% of the sample, from the analysis.

Then, I looked into the "SaleCondition" variable, which recorded the type of sales for each transaction, Most observations were in the "Normal" sale category, but there were **218 observations** recorded as either "Abnormal" or "Family" (intra-family sale). I zoomed in on these, and noted that both their mean and median sale prices (and price per square foot) were well below those of the overall sample. This prompted me to exclude these 218 unusual sale transactions from the analysis. If these transactions were not primarily commercial in nature, then their inclusion in the dataset would introduce **bias** into a model trying to estimate the economic relationship between housing characteristics and house prices.

```
print(df_exout.SalePrice.mean())
print(df_exout.PriceSF.mean())
```

```
175876.4934744268
120.88661412002561
```

```
# 113 of these "abnormal" sale transactions are listed as "abnormal" or to family members,
# and their mean SalePrice is well below the overall mean SalePrice
print(df_exout.SalePrice[(df_exout.SaleCondition == 'Abnorml') | (df_exout.SaleCondition == 'Family')].count())
df_exout.SalePrice[(df_exout.SaleCondition == 'Abnorml') | (df_exout.SaleCondition == 'Family')].mean()
```

```
218

140131.2752293578
```

```
# Even the mean 'PriceSF' of these "abnormal" sales is well below the overall mean
df_exout.PriceSF[(df_exout.SaleCondition == 'Abnorml') | (df_exout.SaleCondition == 'Family')].mean()
```

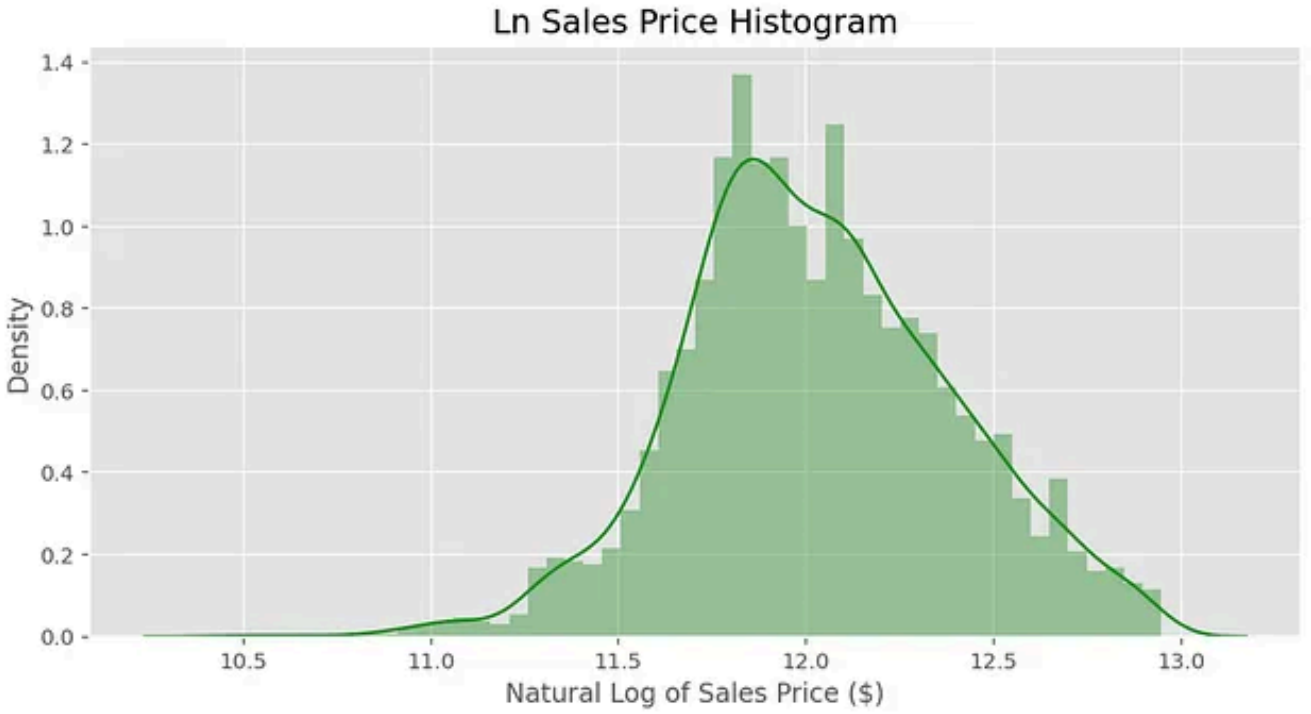
```
103.08140598125344
```

"Abnormal" and "Family" transactions had mean values that were well below the average sale price

Feature Engineering

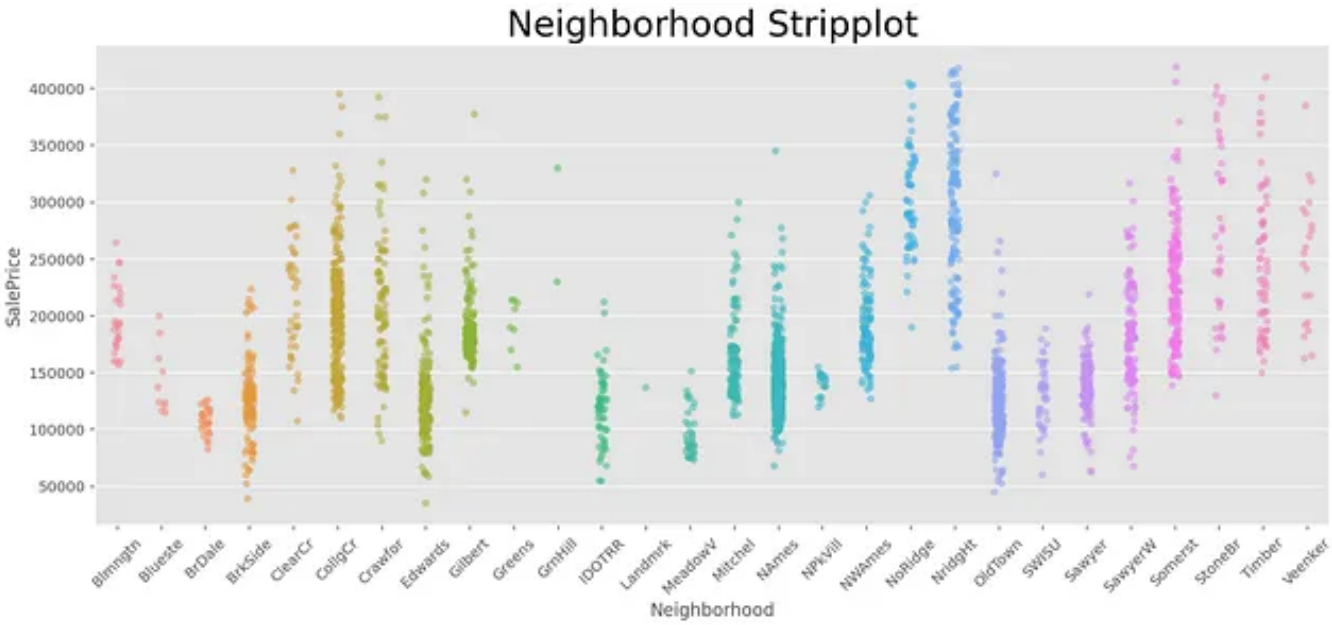
I had now reduced the number of observations from the original 2,930 to 2,617. I then constructed a variable to express the age of the property, "Age", before undertaking a **natural log-transformation** of "SalePrice" . The histogram of log sale prices definitely appeared more symmetrical, with less

extreme values. The resulting model is a *log-linear* model, meaning **a log dependent variable with linear explanatory variables.**



Distribution of the log sale price is much more symmetrical

The next problem was the dearth of a **location desirability variable.** There were **28 neighborhoods** listed in the "Neighborhood" variable, but it's unclear how to rank them. It might be obvious to rank these neighborhoods by their price per square foot, and use this as a location desirability measure, but this artificial construct would naturally be correlated to the house price. That's just not good research methodology because one would be constructing a composite feature variable created partly from the target variable itself. An obvious **proxy** for location desirability would be to rank each neighborhood by its average household income, but the dataset did not have such data available.



Too many neighborhoods and difficult to tell many of them apart by the price distribution

A simplistic way to deal with this would be to **dummify these 28 neighborhoods** and chuck them all into the model. The problems with having individual dummies for such a large number of neighborhoods are:

1. There are only a **handful of observations for some neighborhoods**, with less than 30 observations for eight of them, and less than 100 for the majority of them out of 2,617 sample observations;
2. There would be significant **multicollinearity** between certain neighborhoods that share similar characteristics and therefore have similar distribution of house prices, as may be observed in the stripplot above; and
3. If one had any interest in **statistical inference**, then one cannot draw much inference from such dummies without first having some prior knowledge of how these neighborhoods might rank in terms of desirability.

I chose instead to construct an **ordinal** neighborhood "Location" desirability variable out of various building quality and condition variables in the dataset, notably "OverallQual", "OverallCond", "ExterQual", "ExterCond" and "Functional". The motivation behind my methodology was the assumption that the **more desirable a neighborhood, the better the quality of its housing structures and their condition**.

```
df_exout['Location'] = ((df_exout['OverallQual']/df_exout['OverallQual'].mean())
                        + (df_exout['OverallCond']/df_exout['OverallCond'].mean())
                        + (df_exout['ExterQual_Num']/df_exout['ExterQual_Num'].mean())
                        + (df_exout['ExterCond_Num']/df_exout['ExterCond_Num'].mean())
                        + (df_exout['Functional_Num']/df_exout['Functional_Num'].mean()))
```

```
df_exout.Location.describe()
```

```
count    2617.000000
mean        5.000000
std         0.437242
min         2.087044
25%         4.747860
50%         5.041839
75%         5.269140
max         7.353587
Name: Location, dtype: float64
```

```
df_exout.Location.median()
```

```
5.041839076469422
```

Crafting a "location" score out of the quality and condition variables

With no additional information or insight, I decided that it was best to keep things simple and just allocate **4 ordinal values** for "Location" desirability: 1 (low), 2 (mid-low), 3 (mid-high) and 4 (high). The 28 neighborhoods were sorted by their mean "Location" scores, and I used the overall "Location" mean value to divide the neighborhoods into lower- and upper-half groupings.

The mean "Location" value conveniently split the 28 neighbourhoods into 14 neighborhoods at or above it, and 14 below it. These two groups were then halved again, so that the seven neighborhoods with the lowest mean "Location" scores were assigned "Location"=1, the subsequent seven

neighborhoods were assigned “Location”=2, the next seven were assigned “Location”=3, and the remaining seven were placed in “Location”=4.

Following these assignments, I checked the mean and median “SalePrice” and price per square foot of each “Location” rank, and there was a clear **positive correlation** between the “Location” score and both price variables. This verified that the “Location” variable as constructed could be a good proxy for neighborhood desirability.

```
# Positive correlation between 'SalePrice' and 'PriceSF' with 'Location'
print(df_exout['SalePrice'].groupby(df_exout.Location).mean())
print(df_exout['PriceSF'].groupby(df_exout.Location).mean())
```

```
Location
1    138194.593804
2    151184.290891
3    198714.606299
4    261270.218935
Name: SalePrice, dtype: float64
Location
1    108.533247
2    113.431800
3    129.737730
4    148.842905
Name: PriceSF, dtype: float64
```

‘Location’ ordinal variable has positive correlation to the sale price

That was the most work with a single variable in the exercise. I also created year dummies because the data straddled the **2008 financial crisis**, which had a major impact on US housing prices. Moreover, I created four dummies for each of the four zones (“Zoning”), one dummy for being near an artery road or railway line (“RoadRail”), one dummy being near positive amenities such as a park (“Amenities”), one for garages (“Garage”), one for flat roofs (“FlatRoof”), one for houses higher than one floor (“TwoStorey”), and one for a flat contour of the property lot (“FlatContour”).

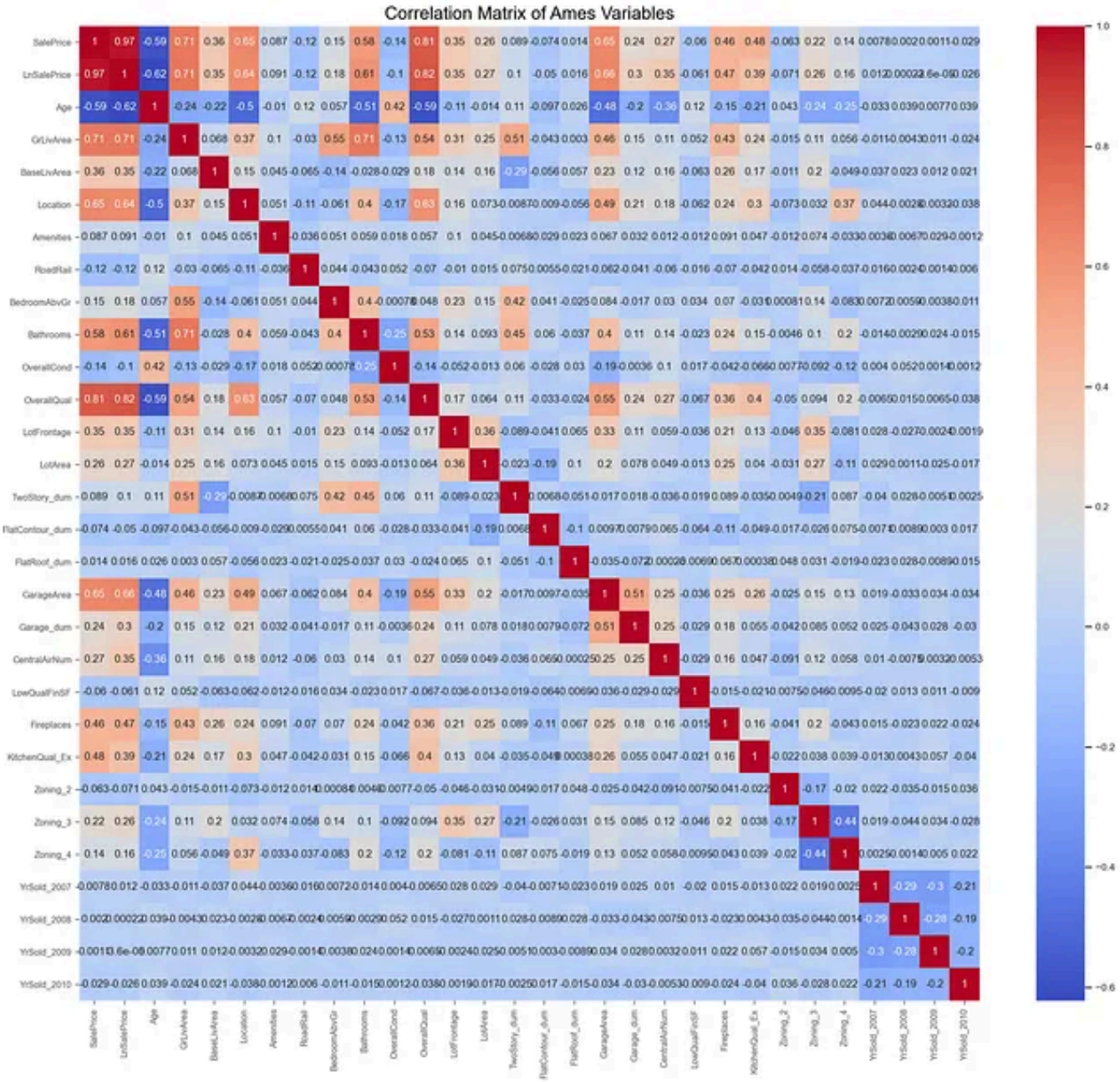
In terms of **indoor unit-specific** features (other than the size variables), I picked “OverallQual” as an ordinal variable, “OverallCond” as an ordinal variable, “Bedrooms” for number of bedrooms, “Bathrooms” for number of

bathrooms, “Fireplaces” for number of fireplaces, “CentralAirCond” as a dummy with 1 for having central air conditioning, and “ExcellentKitchen” as a dummy with 1 for a kitchen rated as “excellent”.

It is commonly said that **“the kitchen sells the house”**, and that fireplaces are particularly desirable. So we shall see.

Regression Analysis

The **correlation matrix** of the resultant engineered data showed that the correlations among the chosen **explanatory variables** were generally under +/-0.50. The only explanatory variable pair that exhibited correlations greater than +/-0.70 was the “GrLivArea-Bathrooms” pair. “Bathrooms” also showed elevated correlations (over 0.50) with “Age” and “OverallQual” separately. I consequently decided to **drop** the “Bathrooms” variable from the regression modeling to reduce **multicollinearity**.



The engineered data was split before running the regression analyses. The **2010** sales data were reserved as the **holdout test set**, while the **2006–2009** data were used for the **train-validation process.**

```
# Assigning the 2006-2009 data to another dataset
df_0609 = df.loc[df['YrSold_2010'] != 1]
df_0609.shape
```

```
(2307, 29)
```

```
# Assigning 2010 data to another dataset to be used as the holdout test set
df_2010 = df.loc[df['YrSold_2010'] == 1]
```

Separating the 2006–2009 data for train-validation, and reserving the 2010 data as the holdout test set

The regression model incorporated the following **explanatory variables**: Age, GrLivArea, BaseLivArea, LowQualFinSF, GarageArea, LotArea, LotFrontage, Location, Bedrooms, Fireplaces, OverallQual, OverallCond, Amenities (dummy), RoadRail (dummy), TwoStorey (dummy), FlatContour (dummy), FlatRoof (dummy), Garage (dummy), CentralAirCond (dummy), ExcellentKitchen (dummy), Zoning_2 (dummy), Zoning_3 (dummy), Zoning_4 (dummy), and the year dummies.

```
y_SP = df_0609['SalePrice']
y_lnSP = df_0609['LnSalePrice']
```

```
X = df_0609.drop(['SalePrice', 'LnSalePrice'], axis=1)
```

```
X.shape
```

```
(2307, 27)
```

```
# Train-test split of the 2006-2009 data
X_train, X_test, y_train, y_test = train_test_split(X, y_lnSP, test_size=0.3, random_state=8)
```

```
scaler = StandardScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns)
X_test = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns)
```

Splitting 2006–2009 data for train-validation procedure, and scaling them

The **target variable** was the natural log of "SalePrice". I used an 70–30 train-validation split for the 2006–2009 data. The explanatory variables were also **standardized** before running the regressions. The train-validation set was subsequently run through the **OLS, Ridge and Lasso linear regression models**. All three linear models provided train-test **scores of 0.91–0.92, MSE of approximately 0.011, and RMSE of approximately 0.106.**

```
# Perform 5-fold cross-validation on the training set
cv_scores = cross_val_score(ols, X_train, y_train, cv=5)

print('Training Score:', ols.score(X_train, y_train))
print('Cross validation scores:', cv_scores)
print('Mean cross validation score:', cv_scores.mean())
print('Test Score:', ols.score(X_test, y_test))
```

```
Training Score: 0.9159788945339753
Cross validation scores: [0.87891456 0.92745804 0.92723048 0.88813805 0.91258065]
Mean cross validation score: 0.9068643574389839
Test Score: 0.9141705342516765
```

```
# Shuffled 5-fold cross validation scores are rather similar
kf = KFold(n_splits=5, shuffle=True, random_state=1)
cv_scores_shuffled = cross_val_score(ols, X_train, y_train, cv=kf)

print('Shuffled cross validation score:', cv_scores_shuffled)
print('Mean shuffled cross validation score:', cv_scores_shuffled.mean())
```

```
Shuffled cross validation score: [0.91999086 0.89730729 0.90205275 0.92901312 0.89263224]
Mean shuffled cross validation score: 0.9081992502720686
```

```
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predictions_test))
print('Root Mean Squared Error:', (metrics.mean_squared_error(y_test, predictions_test))**0.5)
```

```
Mean Squared Error: 0.011182737514335173
Root Mean Squared Error: 0.10574846341358901
```

OLS regression scores, MSE and RMSE

```
# Confirmed similar to the above Ridge CV scores
ridge_mod = Ridge(alpha=21.54)

scores = cross_val_score(ridge_mod, X_train, y_train, cv=5)
print("Cross-validated training scores:", scores)
print("Mean cross-validated training score:", scores.mean())

ridge_mod.fit(X_train, y_train)
print("Training Score:", ridge_mod.score(X_train, y_train))
print("Test Score:", ridge_mod.score(X_test, y_test))
```

```
Cross-validated training scores: [0.88013216 0.92744452 0.92599438 0.88858094 0.91282888]
Mean cross-validated training score: 0.9069961767188179
Training Score: 0.9158691372500658
Test Score: 0.9140564666658226
```

```
ridge_predictions = ridge_mod.predict(X_test)
```

```
ridge_predictions = ridge_mod.predict(X_test)
print('Mean Squared Error:', metrics.mean_squared_error(y_test, ridge_predictions))
print('Root Mean Squared Error:', (metrics.mean_squared_error(y_test, ridge_predictions))**0.5)
```

```
Mean Squared Error: 0.011197599401920943
Root Mean Squared Error: 0.10581871007492458
```

```
# Lasso Cross-Validation
lasso_mod = LassoCV(alphas=np.logspace(-4, 4, 10), cv=5)
lasso_mod.fit(X_train, y_train)

print('Best Lasso alpha:', lasso_mod.alpha_)
print('Training score:', lasso_mod.score(X_train, y_train))
print("Test Score:", lasso_mod.score(X_test, y_test))
```

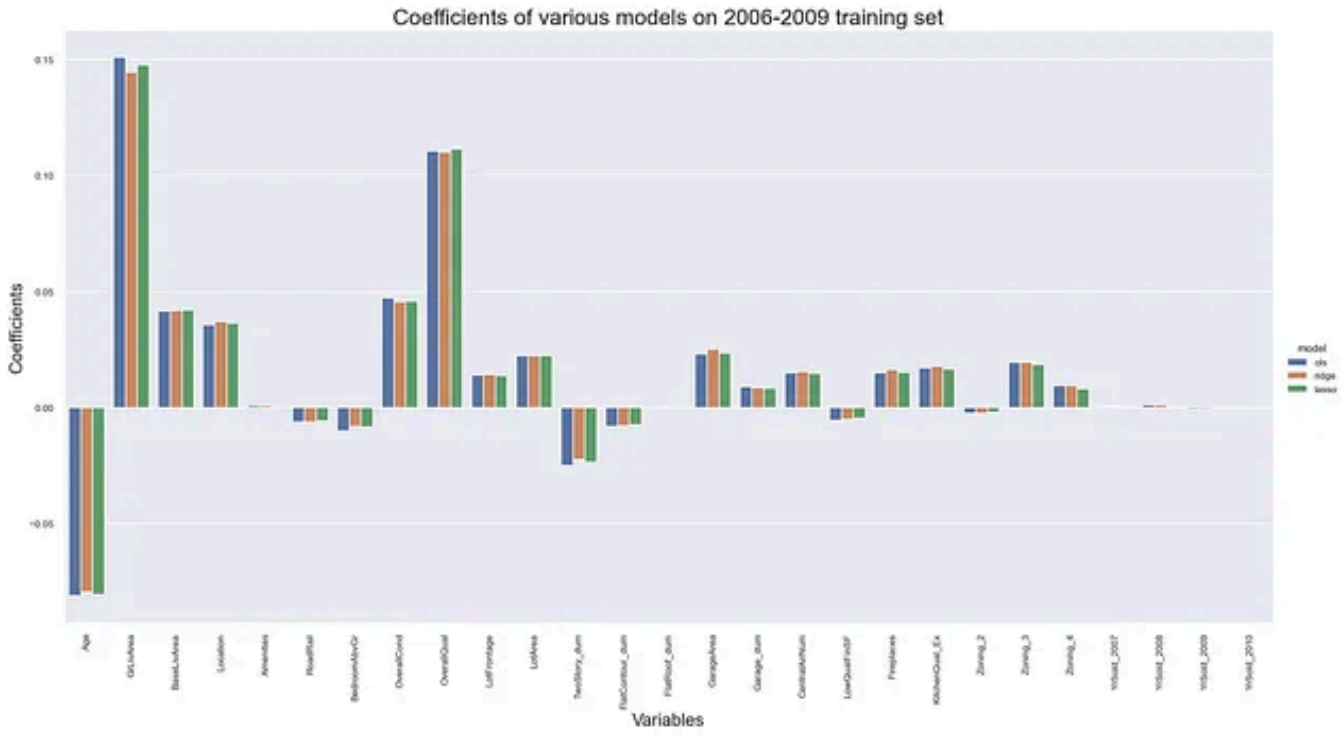
```
Best Lasso alpha: 0.000774263682681127
Training score: 0.915896320632999
Test Score: 0.9139118066533539
```

```
lasso_predictions = lasso_mod.predict(X_test)
print('Mean Squared Error:', metrics.mean_squared_error(y_test, lasso_predictions))
print('Root Mean Squared Error:', (metrics.mean_squared_error(y_test, lasso_predictions))**0.5)
```

```
Mean Squared Error: 0.011216447182623681
Root Mean Squared Error: 0.1059077295697707
```

Ridge/Lasso regression scores, MSE and RMSE

Thus, all three linear models produced rather **consistent scores.** The coefficients were also rather consistent across the models, as may be observed below. These results suggested very **stable model parameter estimates.** The model coefficients imply that the six largest effects on house prices in Ames were (in descending order) the **above ground living space** (positive), **overall quality** (positive), **house age** (negative), **overall condition** (positive), **basement living space** (positive) and **location ranking** (positive). These six variables were all also **statistically significant** at the 95% level, as I will discuss below.

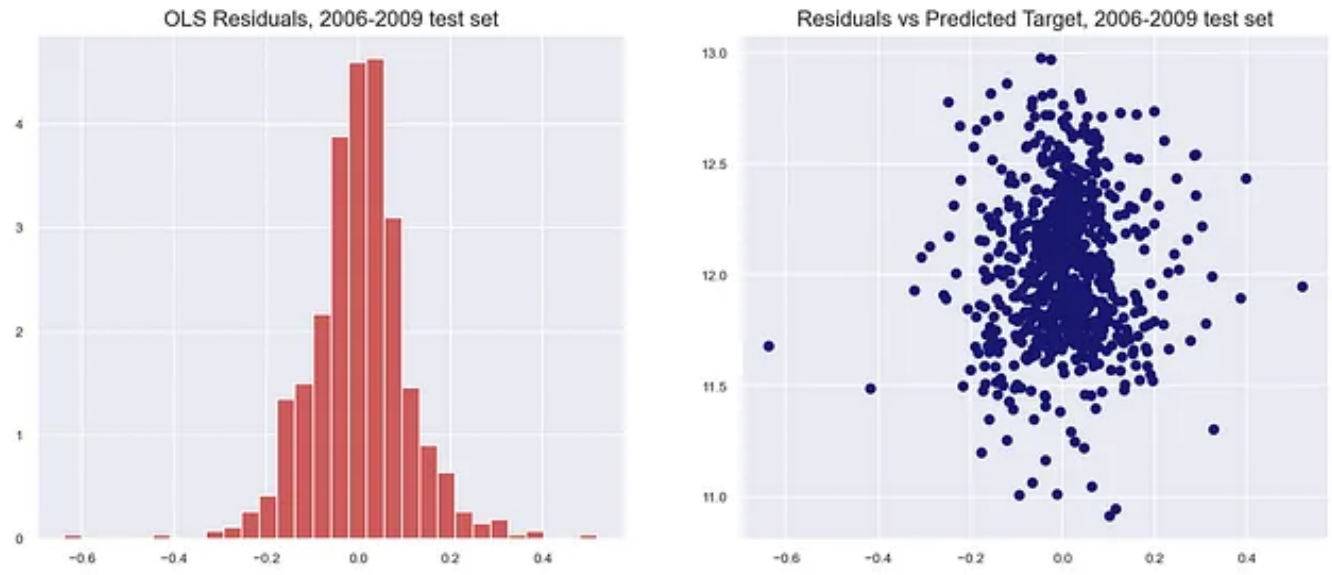


Consistent coefficients across the OLS, Ridge and Lasso regressions

Ordinary least squares regression assumes that the **residual errors** are independent of each other with a normal distribution, mean zero and homoskedastic. The **skew and kurtosis** indicated that the error distributions were approximately normal with mean zero. The residual plots also indicated that the various OLS assumptions regarding the distribution of the

residuals may be assumed to hold. The full residual analysis is available at my GitHub page with the link at the bottom of the article.

2006-2009 data	Train set residuals	Validation set residuals
Mean	0.0000	0.0033
Skew	-0.1695	-0.0394
Kurtosis	2.1447	3.4847



Residuals appear approximately normal and do not show a relationship with the target variable

Holdout Test Scores

For the moment of truth, I tested the model on the **2010 holdout data.** There was a slight drop in the R-squared for the 2010 holdout test set from the training (full 2006–2009) data (0.9014 versus 0.9160), but the scores were again rather **consistent across the OLS, Ridge and Lasso models.** The OLS regression's MSE and RMSE scores were 0.012 and 0.110 respectively on the holdout test set, again showing a slight drop from the train-validation results above.

```
# 5-fold cross-validation, with rather consistent results from the training set both individually and together
cv_scores = cross_val_score(ols, X_train, y_train, cv=5)

print('Training Score:', ols.score(X_train, y_train))
print('Cross validation score:', cv_scores)
print('Mean cross validation score:', cv_scores.mean())
print('2010 holdout data score:', ols.score(X_test, y_lnSP_2010))
```

```
Training Score: 0.9159759957848121
Cross validation score: [0.90818113 0.92403375 0.90683882 0.89583427 0.90777756]
Mean cross validation score: 0.9085331071418962
2010 holdout data score: 0.9014503210592073
```

```
print('Mean Squared Error:', metrics.mean_squared_error(y_lnSP_2010, predictions_test))
print('Root Mean Squared Error:', (metrics.mean_squared_error(y_lnSP_2010, predictions_test))**0.5)
```

```
Mean Squared Error: 0.012215827386758962
Root Mean Squared Error: 0.11052523416287777
```

OLS scores on the 2010 holdout set

```
ridge_mod = RidgeCV(alphas=np.logspace(-4, 4, 10), cv=5)
ridge_mod.fit(X_train, y_train)

print('Best Ridge alpha:', ridge_mod.alpha_)
print('Training score:', ridge_mod.score(X_train, y_train))
print("2010 holdout data score:", ridge_mod.score(X_test, y_lnSP_2010))
```

```
Best Ridge alpha: 21.54434690031882
Training score: 0.9159207883132006
2010 holdout data score: 0.9015978800682629
```

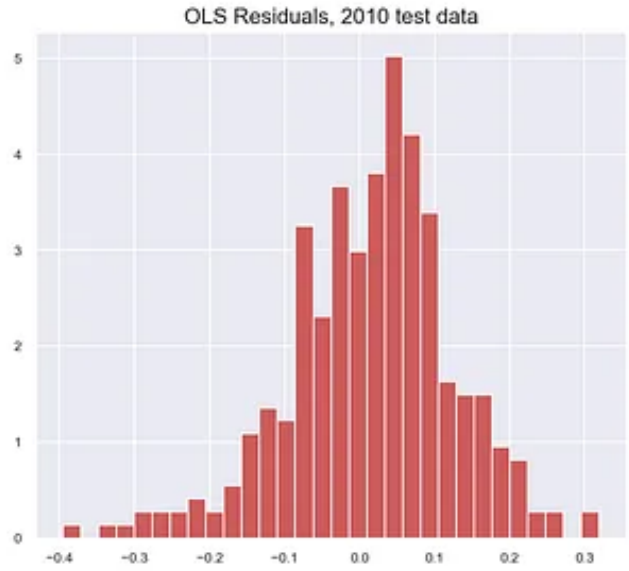
```
lasso_mod = LassoCV(alphas=np.logspace(-4, 4, 10), cv=5)
lasso_mod.fit(X_train, y_train)

print('Best Lasso alpha:', lasso_mod.alpha_)
print('Training score:', lasso_mod.score(X_train, y_train))
print("2010 holdout data score:", lasso_mod.score(X_test, y_lnSP_2010))
```

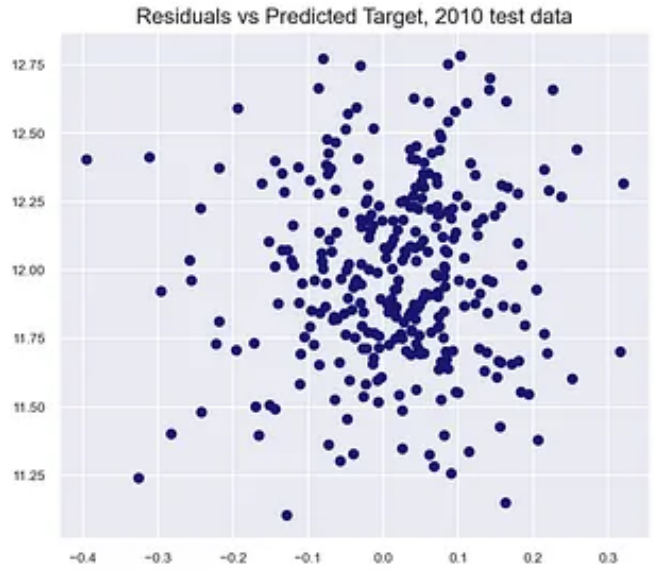
```
Best Lasso alpha: 0.000774263682681127
Training score: 0.9158861851228268
2010 holdout data score: 0.9013278388795367
```

Ridge/Lasso scores on the 2010 holdout set

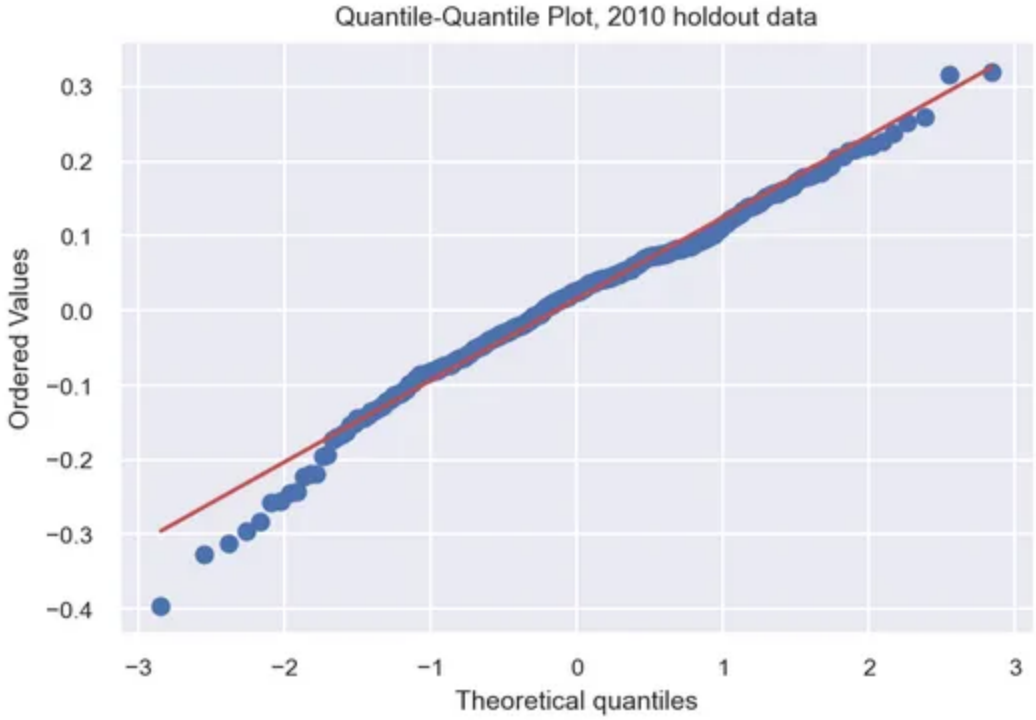
The residuals were also approximately normally distributed with mean zero. The **quantile-quantile plot** (qq-plot) of the test residuals in particular indicated a reasonable fit to a normal distribution.

2010 data	Holdout set residuals
Mean	0.0152
Skew	-0.4432
Kurtosis	0.9412





Residuals appear approximately normal and do not show a relationship with the target variable

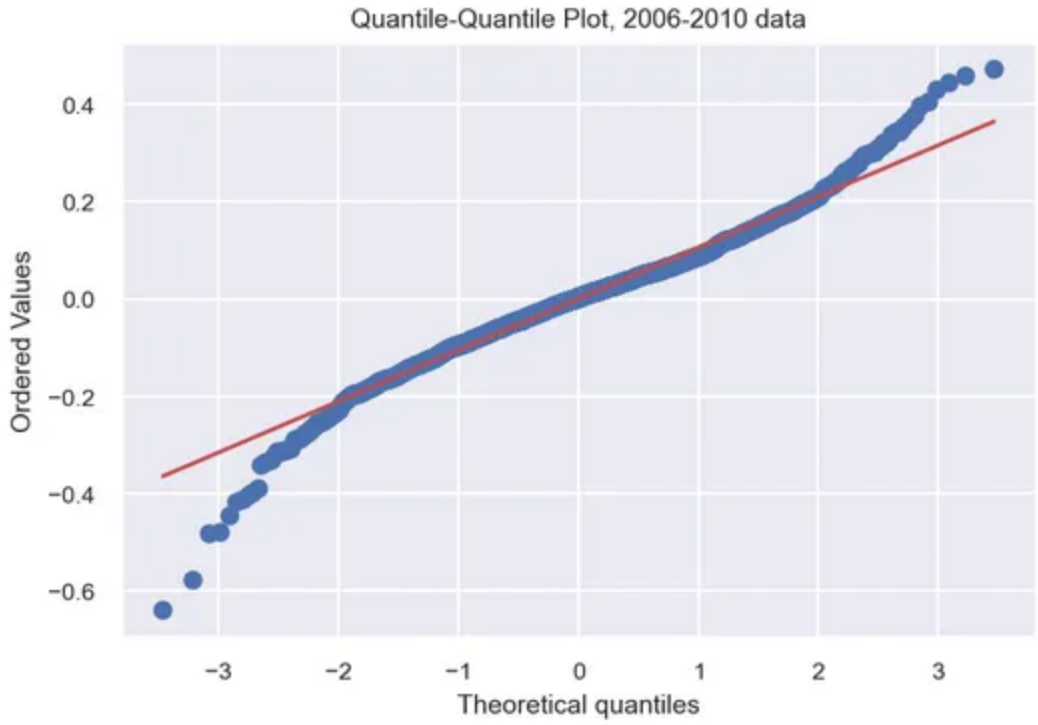


Statistical Inference

I utilized the **full 2006–2010 dataset** for statistical inference. This is one of the key differences between running a regression model for prediction purposes and doing it for causal inference. When we train a model for prediction, we **split the data** and segregate a test set to evaluate the accuracy

of the model predictions (as executed above). In statistical inference, on the other hand, the **full dataset is typically used** to derive the parameter estimates, which are evaluated against various test statistics.

The qq-plot for the model incorporating the full 2006–2010 dataset cautions against an overly optimistic use of the model to predict house prices. The qq-plot indicates that the model tends to **under-predict house prices in the highest quantile and over-predict those in the lowest quantile.** So while the final model may explain over 91% (R-squared) of the variation in house prices, it is not reliable when dealing with houses at the extremes of the price range. The model is most effective when targeting those properties with prices in the **25%-75% inter-quartile range.**



q-q plot of the residuals for the full 2006–2010 data indicates some sizeable skews

As for calculating exactly how the various explanatory variables might affect house prices, I would first need to compute the individual coefficient **standard errors** and **p-values** to determine if a particular variable is

statistically significant. I would also need to revert back to the original **unstandardized explanatory variables.** The resulting summary table of the OLS regression results on the **full 2006–2010 data** is below, from the *Statsmodels* module.

OLS Regression Results

Dep. Variable:	LnSalePrice	R-squared:	0.915
Model:	OLS	Adj. R-squared:	0.914
Method:	Least Squares	F-statistic:	1029.
Date:	Sun, 23 May 2021	Prob (F-statistic):	0.00
Time:	15:45:52	Log-Likelihood:	2154.3
No. Observations:	2616	AIC:	-4253.
Df Residuals:	2588	BIC:	-4088.
Df Model:	27		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	10.5389	0.024	445.575	0.000	10.492	10.585
Age	-0.0027	0.000	-23.277	0.000	-0.003	-0.002
GrLivArea	0.0003	9.35e-06	35.585	0.000	0.000	0.000
BaseLivArea	9.716e-05	5.44e-06	17.876	0.000	8.65e-05	0.000
Location	0.0333	0.003	11.353	0.000	0.028	0.039
Amenities	-0.0146	0.015	-0.961	0.337	-0.044	0.015
RoadRail	-0.0410	0.009	-4.616	0.000	-0.058	-0.024
BedroomAbvGr	-0.0117	0.004	-3.267	0.001	-0.019	-0.005
OverallCond	0.0457	0.002	19.981	0.000	0.041	0.050
OverallQual	0.0812	0.003	30.263	0.000	0.076	0.086
LotFrontage	0.0008	0.000	6.112	0.000	0.001	0.001
LotArea	3.185e-06	3.57e-07	8.916	0.000	2.48e-06	3.89e-06
TwoStory_dum	-0.0458	0.006	-7.563	0.000	-0.058	-0.034
FlatContour_dum	-0.0204	0.007	-2.737	0.006	-0.035	-0.006
FlatRoof_dum	0.0336	0.027	1.263	0.207	-0.019	0.086
GarageArea	0.0001	1.59e-05	8.228	0.000	9.94e-05	0.000
Garage_dum	0.0205	0.012	1.713	0.087	-0.003	0.044
CentralAirNum	0.0680	0.010	6.560	0.000	0.048	0.088
LowQualFinSF	-0.0001	5.37e-05	-2.682	0.007	-0.000	-3.87e-05
Fireplaces	0.0261	0.004	6.485	0.000	0.018	0.034
KitchenQual_Ex	0.0715	0.010	7.123	0.000	0.052	0.091
Zoning_2	-0.0376	0.025	-1.521	0.128	-0.086	0.011
Zoning_3	0.0497	0.007	7.037	0.000	0.036	0.063
Zoning_4	0.0565	0.012	4.586	0.000	0.032	0.081
YrSold_2007	0.0035	0.006	0.563	0.573	-0.009	0.016
YrSold_2008	0.0065	0.007	1.006	0.314	-0.006	0.019
YrSold_2009	-0.0064	0.006	-0.990	0.322	-0.019	0.006
YrSold_2010	0.0156	0.008	2.036	0.042	0.001	0.031

Omnibus:	158.447	Durbin-Watson:	1.682
Prob(Omnibus):	0.000	Jarque-Bera (JB):	592.121
Skew:	-0.176	Prob(JB):	2.65e-129
Kurtosis:	5.304	Cond. No.	1.55e+05

OLS regression on full 2006–2010 unscaled data

As may be observed, the overwhelming majority of the explanatory variables in the OLS model are **statistically significant** at the 95% level (i.e. *p-value* < 5%), as indicated by the individual t-statistics and associated p-values. The exceptions are the "RoadRail" and "LowQualFinSF" variables. The six variables found to have the largest impact on the house sale price are all statistically significant, as mentioned above.

```
import math

transformed_coef = []
for i in unscaled_coef:
    j = math.exp(i)
    transformed_coef.append(j)
print(transformed_coef)
```

```
[0.9972752408705468, 1.0003329291585632, 1.00009716923651, 1.0338247623381687, 0.9855426176035118, 0.959821470532034
3, 0.988369298063296, 1.0467486352268127, 1.0845439294576424, 1.000751771469628, 1.0000031846894666, 0.95525865324542
07, 0.9797963059158258, 1.0341265484179938, 1.000130531339637, 1.0206625196969974, 1.070345859587567, 0.9998559869822
011, 1.0264156398065571, 1.074077907916657, 0.963052462050285, 1.050908491836021, 1.0581614678622424, 1.0035490335757
256, 1.0065624160320006, 0.9936692254476065, 1.015710639164118]
```

```
coef_effect = [(i - 1)*df.SalePrice.mean() for i in transformed_coef]
```

Transforming the log-linear coefficients back to their original units

As a reminder, the model is *log-linear* where the target variable is the natural log of "SalePrice". To compute the impact of a unit change in each variable on the "SalePrice", we would first need the **exponent of each coefficient.** Moreover, the coefficients of a log-linear model imply the **percentage change** in the target variable for small unit changes in the explanatory variable. To get the specific dollar impact, it is best to use the **mean "SalePrice" as the reference comparison.**

	variable	1-unit change
0	Age	-487.364
1	GrLivArea	59.5493
2	BaseLivArea	17.3802
3	Location	6050.06
4	Amenities	-2585.92
5	RoadRail	-7186.52
6	BedroomAbvGr	-2080.32
7	OverallCond	8361.69
8	OverallQual	15121.9
9	LotFrontage	134.465
10	LotArea	0.569629
11	TwoStory_dum	-8002.65
12	FlatContour_dum	-3613.73
13	FlatRoof_dum	6104.04
14	GarageArea	23.3475
15	Garage_dum	3695.8
16	CentralAirNum	12582.4
17	LowQualFinSF	-25.7589
18	Fireplaces	4724.83
19	KitchenQual_Ex	13249.9
20	Zoning_2	-6608.61
21	Zoning_3	9105.74
22	Zoning_4	10403
23	YrSold_2007	634.797
24	YrSold_2008	1173.79
25	YrSold_2009	-1132.35
26	YrSold_2010	2810.08

Dollar impact of variables on “SalePrice”

The **dollar impact of a one-unit change in each explanatory variable on the average house price** in Ames is listed in the table on the left. To keep it brief, I will only discuss some of those variables found to be statistically significant.

For example, an increase in the **“Age”** of the housing unit by one year will reduce the average house price by $487, *all else being equal*. A one-category increase in the **“Location”** (remember that this was a four-category ordinal variable) by comparison will raise the average house price by $6,050 (the fourth variable), *all else being equal*.

Moreover, a **fireplace** will add $4,725 to the average house price, while a **kitchen rated “excellent”** will bump up the price by $13,250 (the 19th and 20th variables).

So indeed “***the kitchen sells the house***”!

Conclusion

Based on insights from the **hedonic pricing literature** on house prices, I zoomed in on various property size measures as likely important features for the regression model. I also undertook a **log transformation** of the highly skewed house price **target variable**.

There followed extensive feature engineering on various structural and internal factors expected to be significant determinants of house prices, principally a location desirability variable, zoning dummies, build quality and condition, proximity to amenities, and a highly-rated kitchen among

others. The engineered data was then run through **three linear regression models**: OLS, Ridge and Lasso.

Stable results and scores were found across the three linear models. The OLS regression coefficients nearly matched those derived from the Ridge and Lasso regressions. The modeling achieved an **R-squared of approximately 91-92%** on the training-validation sets, and this was **consistent** across the OLS, Ridge and Lasso regressions. It subsequently managed to secure a **90% score on the 2010 holdout test set**, along with MSE and RMSE scores of 0.012 and 0.110 respectively. The results were again consistent across the various algorithms.

The assorted **property size variables**, namely internal living area, basement finished area, lot area and garage area, were all found to be **positively correlated** to the sale price and **statistically significant. Build quality and condition** were also found to be among the most important determinants of house prices, as was **location desirability**.

There was no need for any abstruse black-box algorithm to achieve these results. Just straight-up **linear regression** that is **transparent and interpretable**, and where the individual explanatory variables may be subjected to **hypothesis testing**. Thus, the goals of prediction and statistical inference could be pursued together. More importantly, the linear regression results can be easily communicated to lay audiences, at both the model ***and*** individual variable levels.

(The full Python code and data for this exercise are available in my *GitHub* repository. If it is problematic rendering the GitHub notebook files directly, use nbviewer.)

If you saw value in reading articles like this, you may subscribe to Medium here to read other articles by me and countless other writers. Thank you.

Time & Seasonality Features in Time Series

Be data-focused, and include seasonality options in the model calibration process

towardsdatascience.com

Uncovering the Latent Factors Behind Stock Prices

Dynamic factor modelling of US large-cap equities

medium.datadriveninvestor.com

Tackling Imbalanced Data With Predicted Probabilities

A case study on the Portuguese bank marketing dataset

towardsdatascience.com

Hierarchical Clustering of the FX Market

Using unsupervised machine learning to identify behavioral currency clusters

towardsdatascience.com

Data Science | Machine Learning | Statistics | Dataland | Artificial Intelligence

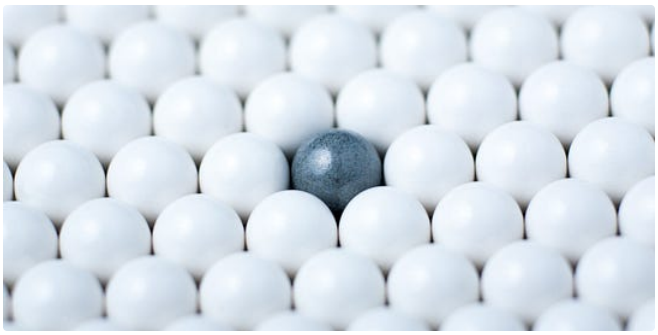
Written by Alvin T. Tan, Ph.D.

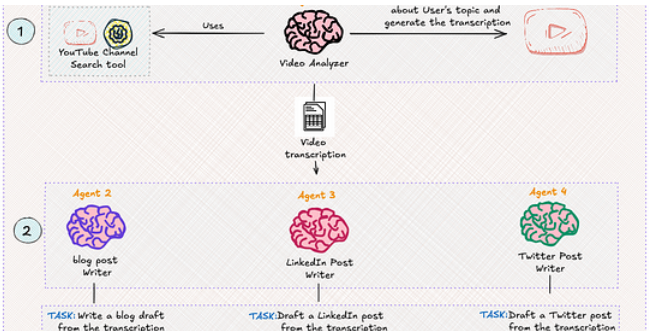
Follow

424 Followers · Writer for Towards Data Science

Financier by profession. Economist by training. Data scientist & essayist by inclination.

More from Alvin T. Tan, Ph.D. and Towards Data Science





Alvin T. Tan, Ph.D. in Towards Data Science

Tackling Imbalanced Data with Predicted Probabilities

A case study on optimizing class probability in the Portuguese bank marketing dataset

Apr 8, 2021 296 5

Zoumana Keita in Towards Data Science

AI Agents—From Concepts to Practical Implementation in Python

This will change the way you think about AI and its capabilities

Aug 12 1.1K 13

Ahmed Besbes in Towards Data Science

What Nobody Tells You About RAGs

A deep dive into why RAG doesn't always work as expected: an overview of the...

Aug 23 1.5K 22

Alvin T. Tan, Ph.D. in Towards Data Science

Time & Seasonality Features in Time Series

Be data-focused, and include seasonality options in the model calibration process

Mar 15, 2022 267

See all from Alvin T. Tan, Ph.D.

See all from Towards Data Science

Recommended from Medium

Wei Hutchinson

Mastering Outliers in Marketing Mix Modelling (MMM): What You...

Hello there, Dr Wei here! I know, I know—I haven't discussed MMM for a while. Conside...

Sep 3 · 7

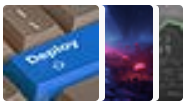
Jamie Crossman-Sm... in Low Code for Data Scien...

XGBoost Explained: A Beginner's Guide

Understand how XGBoost works, when to use it, and its advantages over other algorithms

Mar 24 · 271 · 2

Lists



Predictive Modeling w/ Python

20 stories · 1509 saves

Practical Guides to Machine Learning

10 stories · 1835 saves

Natural Language Processing

1687 stories · 1260 saves

ChatGPT prompts

48 stories · 1970 saves

Kevalsakhiya in The Deep Hub

Asish Biswas in AnalyticSoul

Bias-Variance Tradeoff

Bias, Variance, and Error in Machine Learning Models

Aug 23 50

PCA Dimensionality Reduction: The Complete Guide to Simplifying...

Learn how PCA can make your analyses more effective and insightful by reducing data...

Jun 4 17

Patwariraghottam in DevOps.dev

Mastering Feature Selection: Techniques to Boost Your Machin...

Aug 25 24 1

Playtika Tech & AI

ANALYZING UPLIFT MODELS

By Dvir Ben Or and Michael Kolomenkin

Apr 25 8

See more recommendations