MULTIVARIATE REGRESSION ON REAL ESTATE PRICE PREDICTION

MINH PHUONG

OBJECTIVE

- Apply what I learned in class
- Explore new models that I have not used
- Portfolio piece for job search (Yay employment!)

IDENTIFY PROBLEM

- Context: A local Vermont/New Hampshire real estate firm is looking into modeling closed prices for houses.
- Question: CAN YOU PREDICT CLOSED PRICE OF A PROPERTY USING THEIR LISTING FEATURES

WORKFLOW

- Train model
- Try different parameters, preprocessing and transformation (dimensionality reduction), inverse transformation
- Repeat until best model found; create ensemble
- Fine tune on testing accuracy

STEP 2: OBTAINING DATA

- Data was obtained from Kaggle.com
- This dataset contains features of houses in three towns in Vermont, which make up a sizable chunk of the real estate firm's business.
- MLS.com is the real estate information platform that is publicly available. Features were exported from an MLS web platform. (Perhaps webscraping)

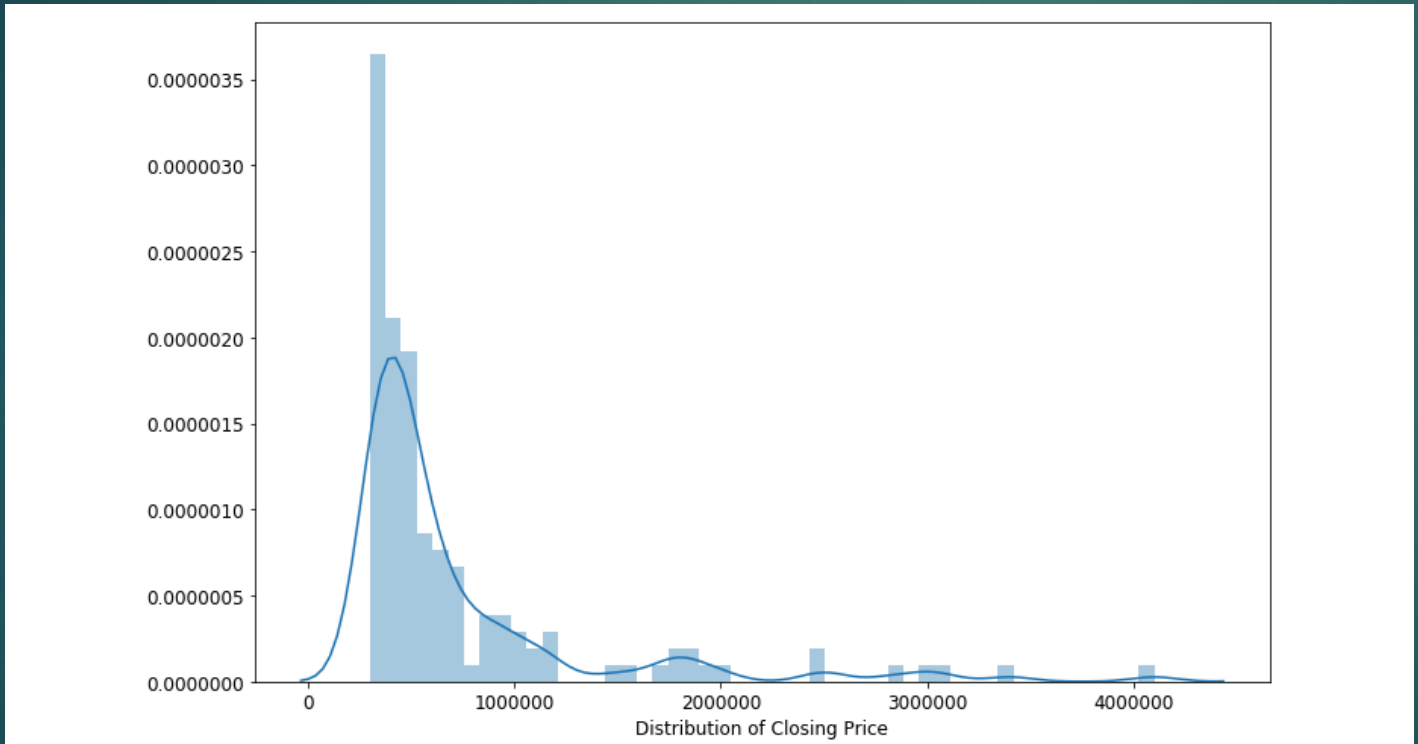
STEP 3: UNDERSTANDING THE DATA

- Features in Data Frame:

Features	Type
Id, bedrooms_total, baths_total, acres, sq_ft_tot_fn, tax_gross_amount, assessment_value_town, garage_capacity, year_built, total_stories, water_frontage_length, rooms_total,	Numeric
Address, city ,garage_type , surveyed , seasonal , water_body_type , short_sale , flood_zone , easements , current_use , covenants , common_land_acres, basement_access_type,basement,price_closed	String

STEP 3: UNDERSTANDING THE DATA

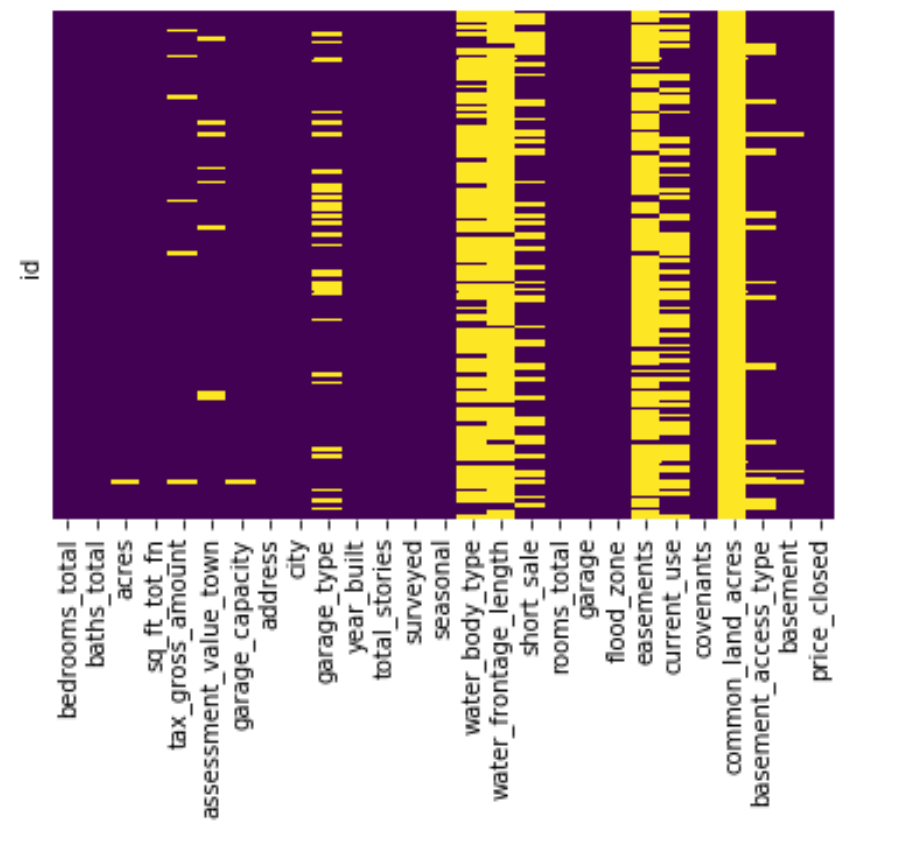
- Distribution of Closing Price



Almost normal distribution with a long tail

STEP 3: UNDERSTANDING THE DATA

- Missing values: Using Seaborn heatmap



Most prominent:

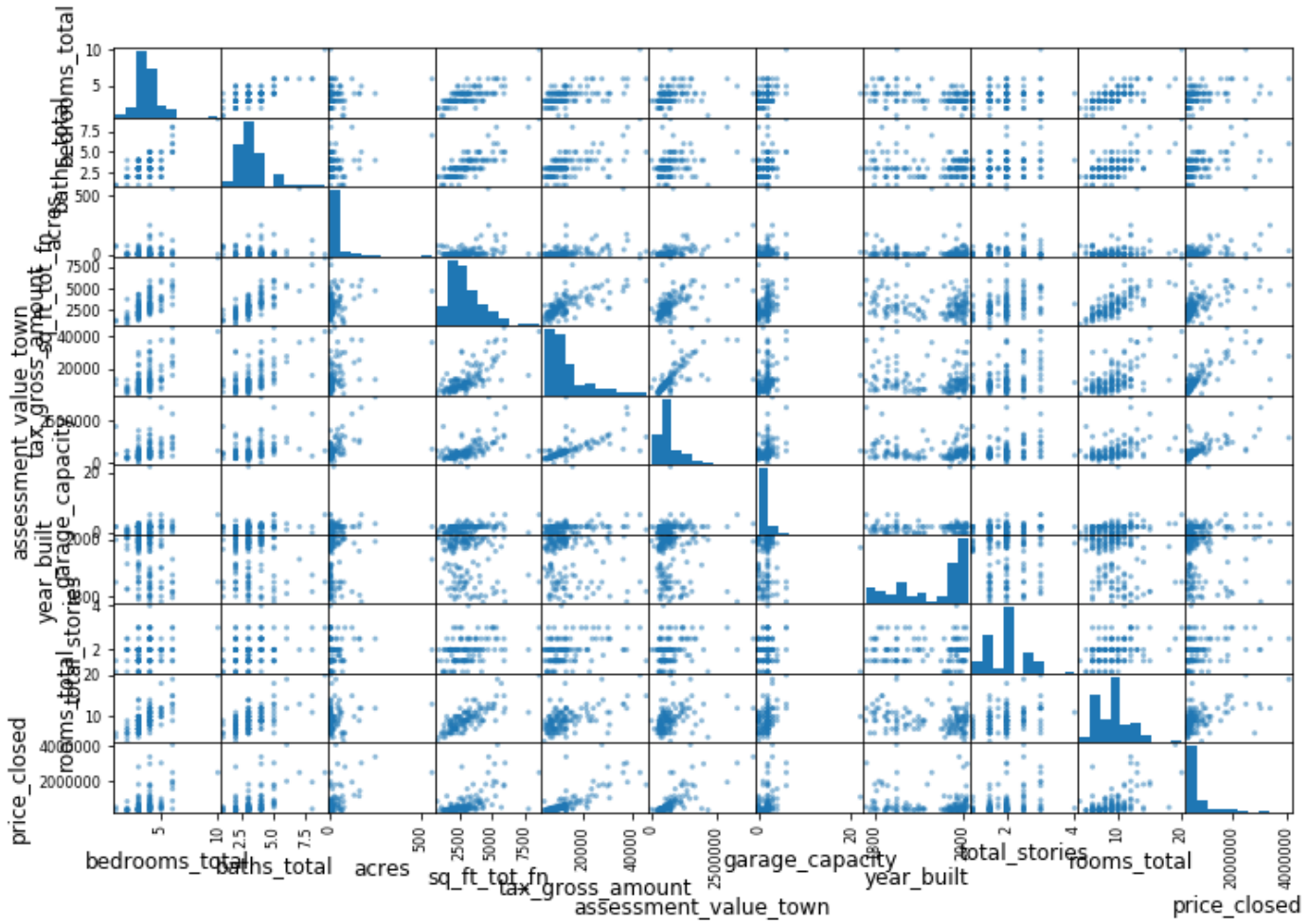
- common_land_acres
- water_frontage_length
- easements
- water_body_type

STEP 4: PREPARING THE DATA

- Drop features that are missing from more than 80% of the data set
  - df = df.drop(['common_land_acres','water_frontage_length','easements','water_body_type'],axis=1)
- Filled the rest with mean (numerical)
  - for col in ['acres','garage_capacity','tax_gross_amount','assessment_value_town']: df[col] = df[col].fillna(df[col].mean())

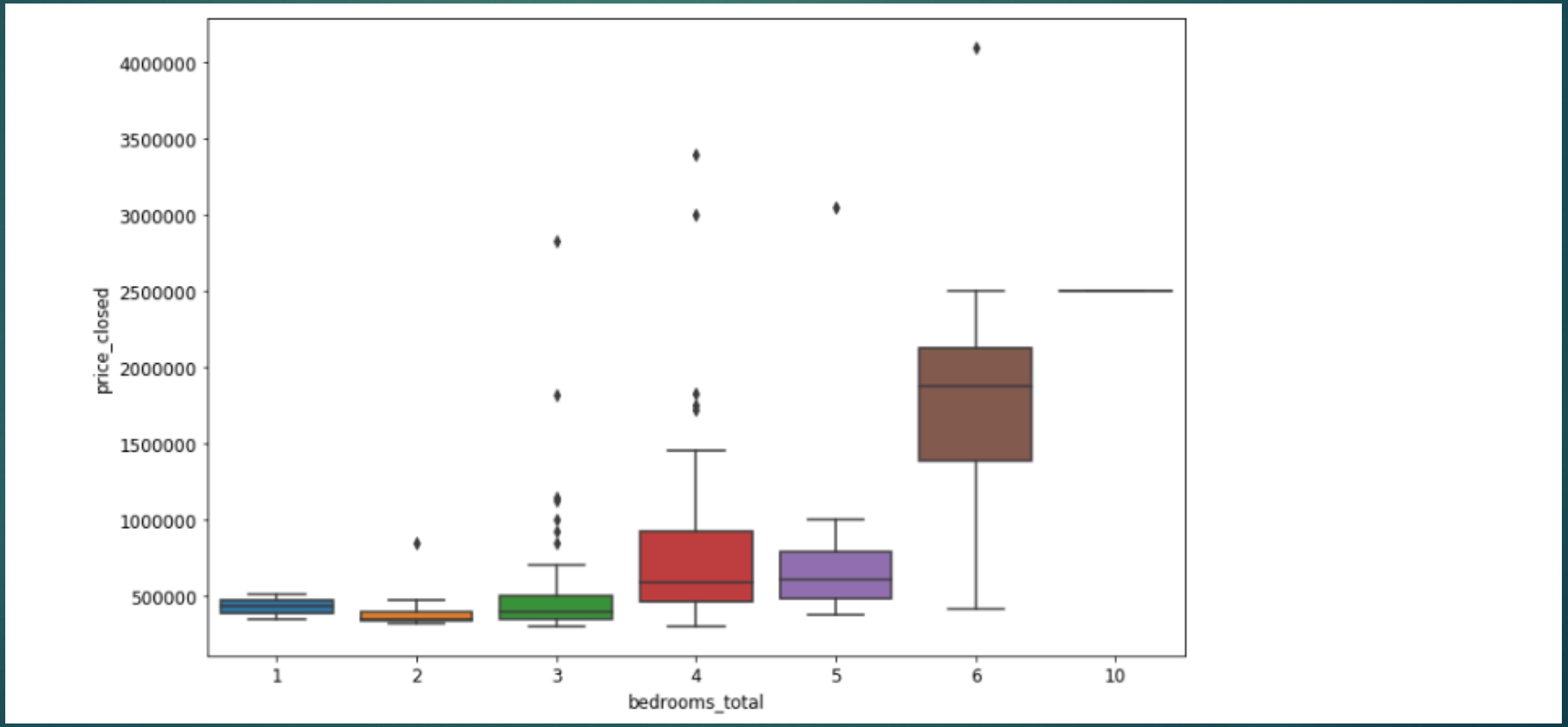
STEP 5: VISUALIZATION

- Explore relationships:



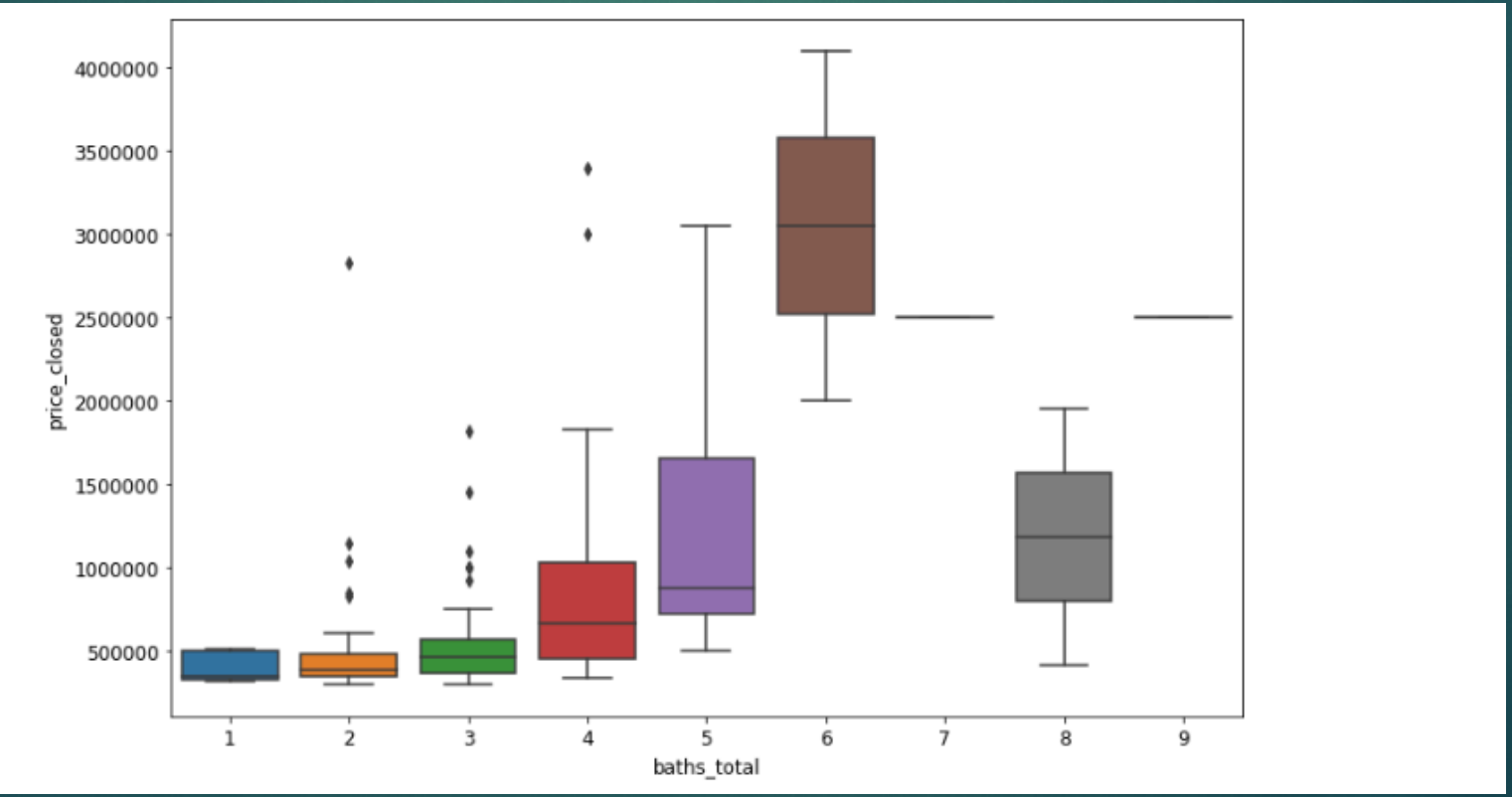
STEP 5: VISUALIZATION

- Narrow down to features with regression:



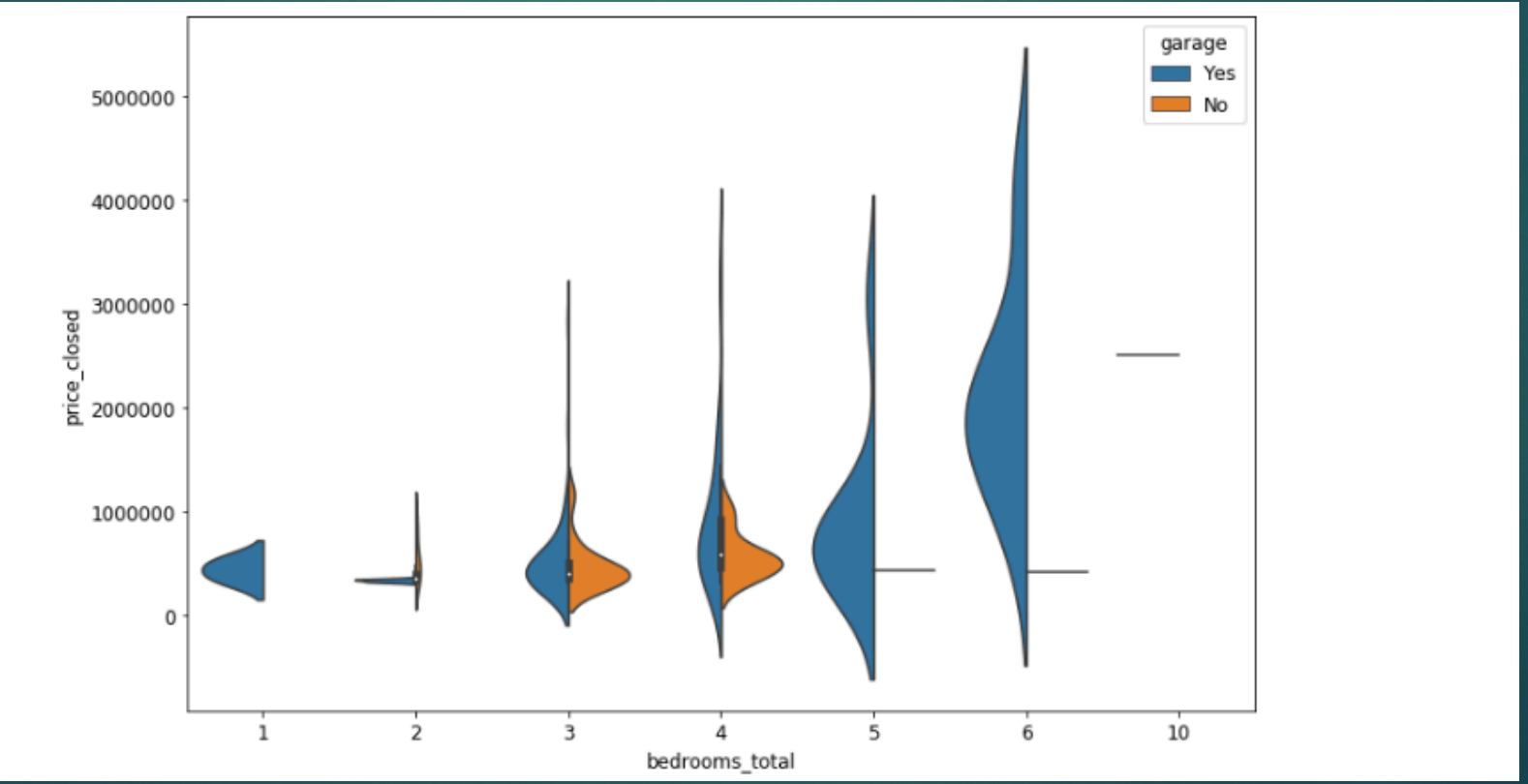
STEP 5: VISUALIZATION

- Narrow down to features with regression:



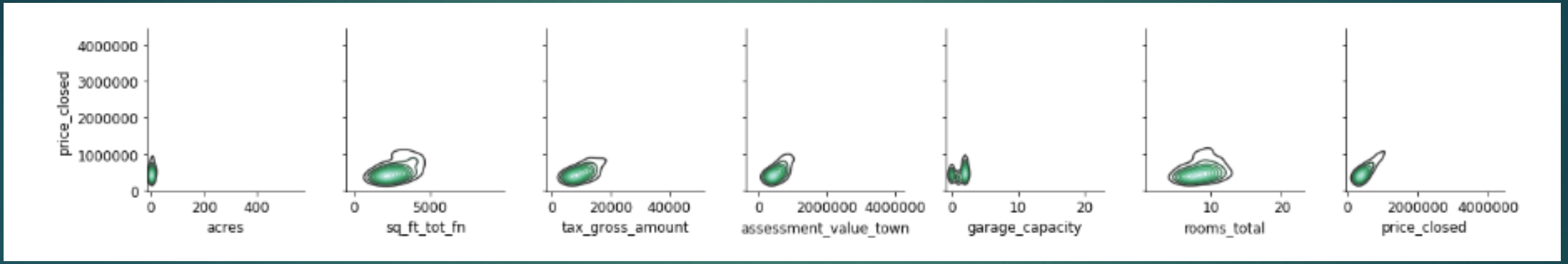
STEP 5: VISUALIZATION

- Narrow down to features with regression:



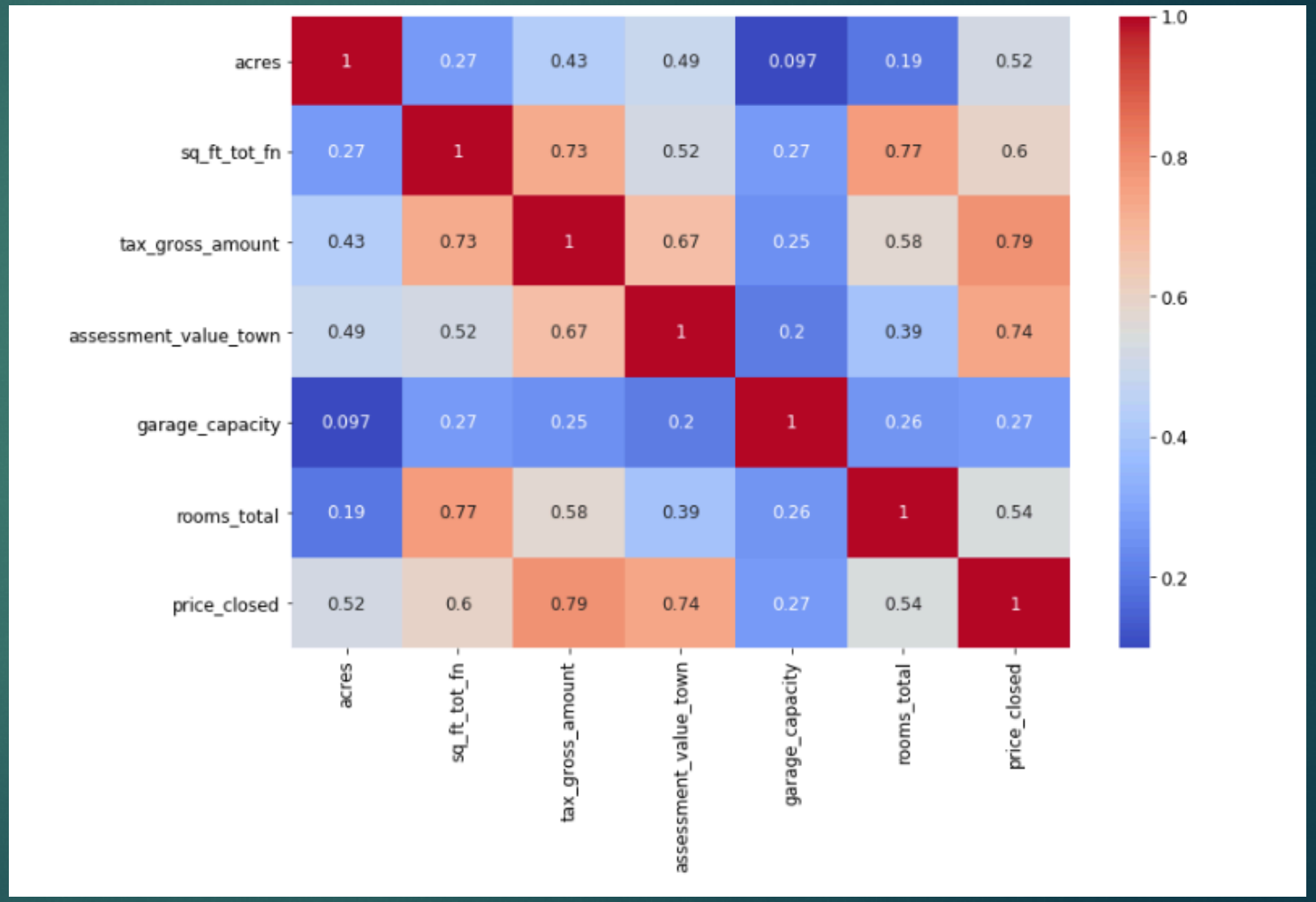
STEP 5: VISUALIZATION

- Narrow down to features with no clear regression:



STEP 5: VISUALIZATION

- Find out correlation:



Simple Linear Regression

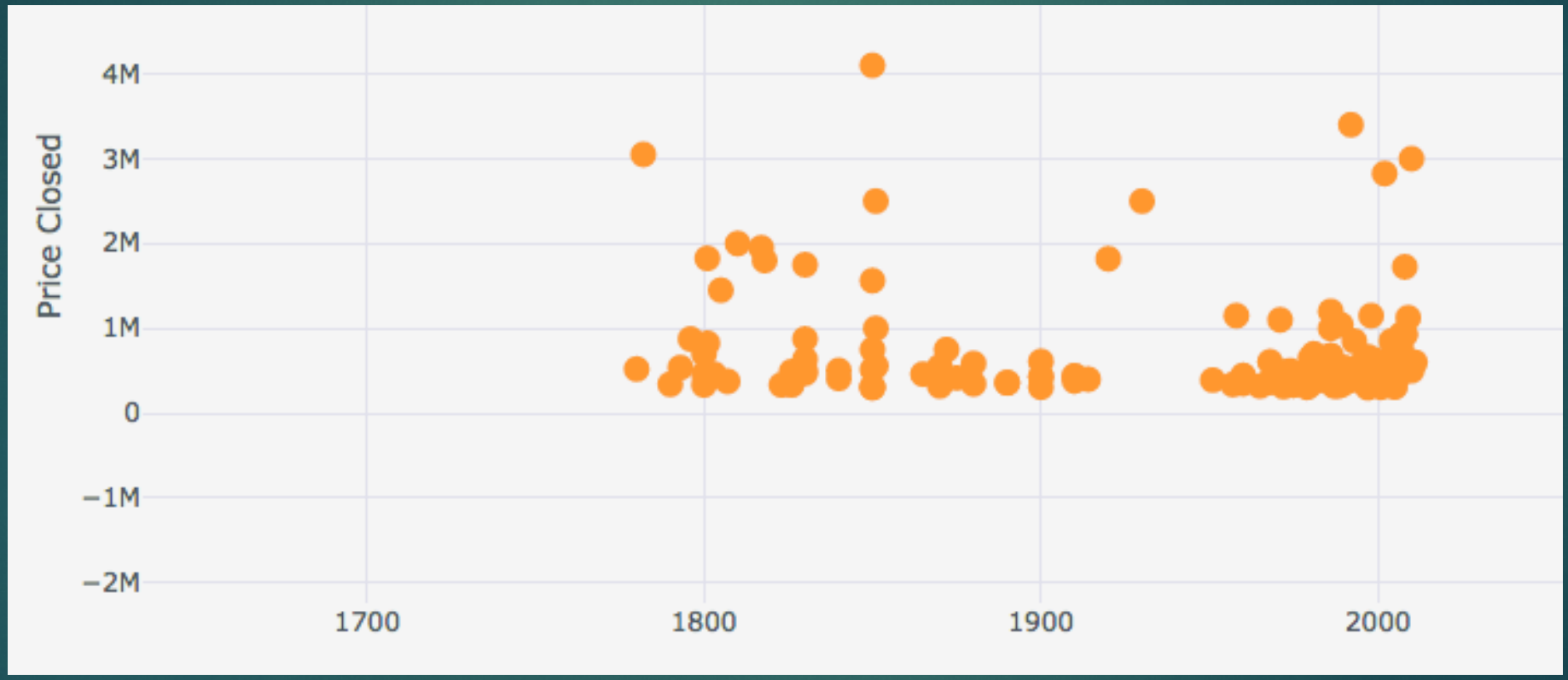
- Metrics such as mean absolute errors, mean squared errors and root mean squared errors are used to evaluate this first run:
  - MAE: 314120.6
  - MSE: 262121390014.0
  - RMSE: 511977.9
- Null Accuracy: 649444.9
- Testing Accuracy: RMSE == 451590.9 (NEEDS IMPROVEMENT)

Continue to clean up data set, Feature Engineering

- K Nearest Neighbors estimator is used to impute values for missing values using their closest 5 neighbor rows
- Mapping is also used on some of the smaller missing values on features
- Clean dataset is exported out to be used in the next step
- Entire process from preliminary and current step is repeated on the testing set

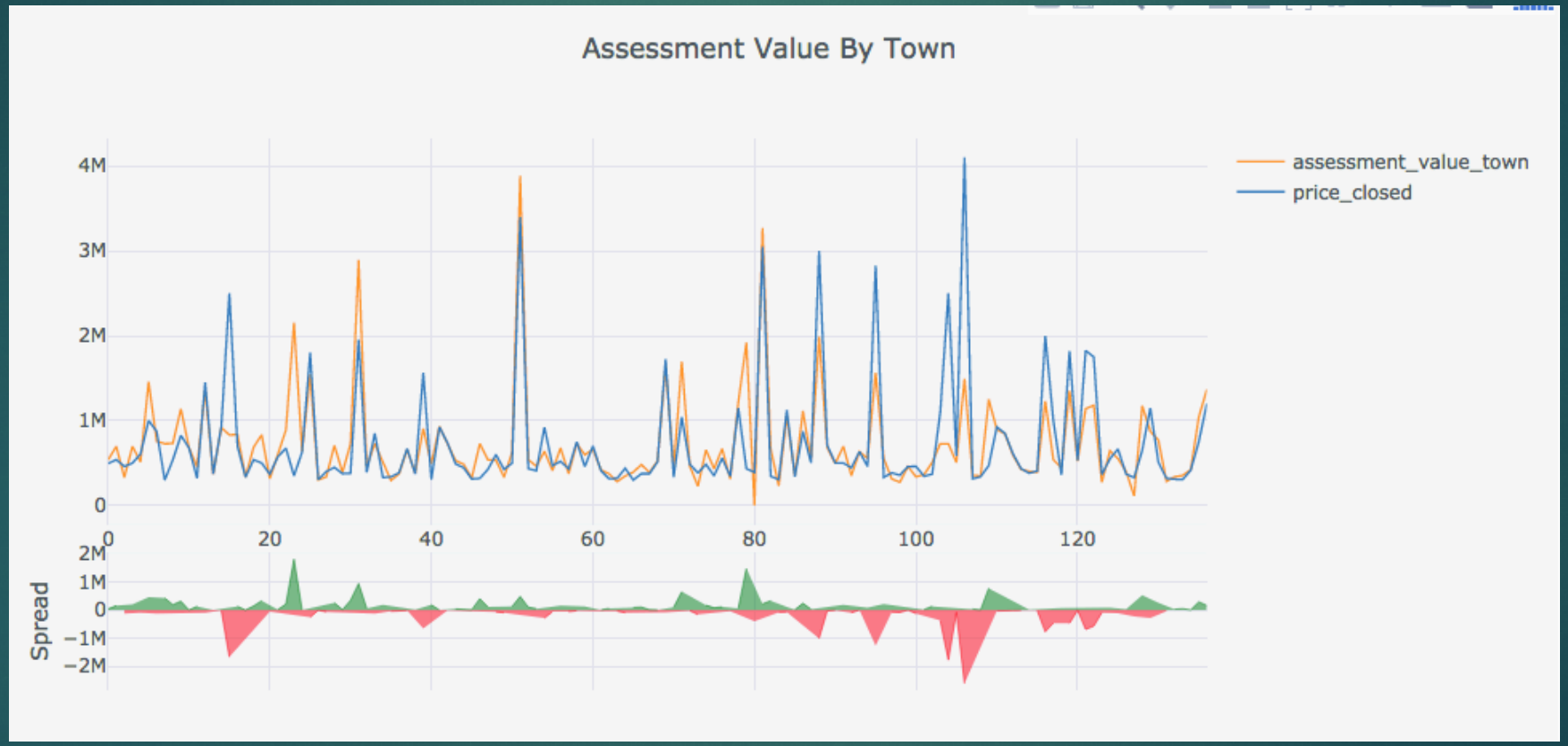
Further EDA

- Look for linear relationship, outliers and clusters



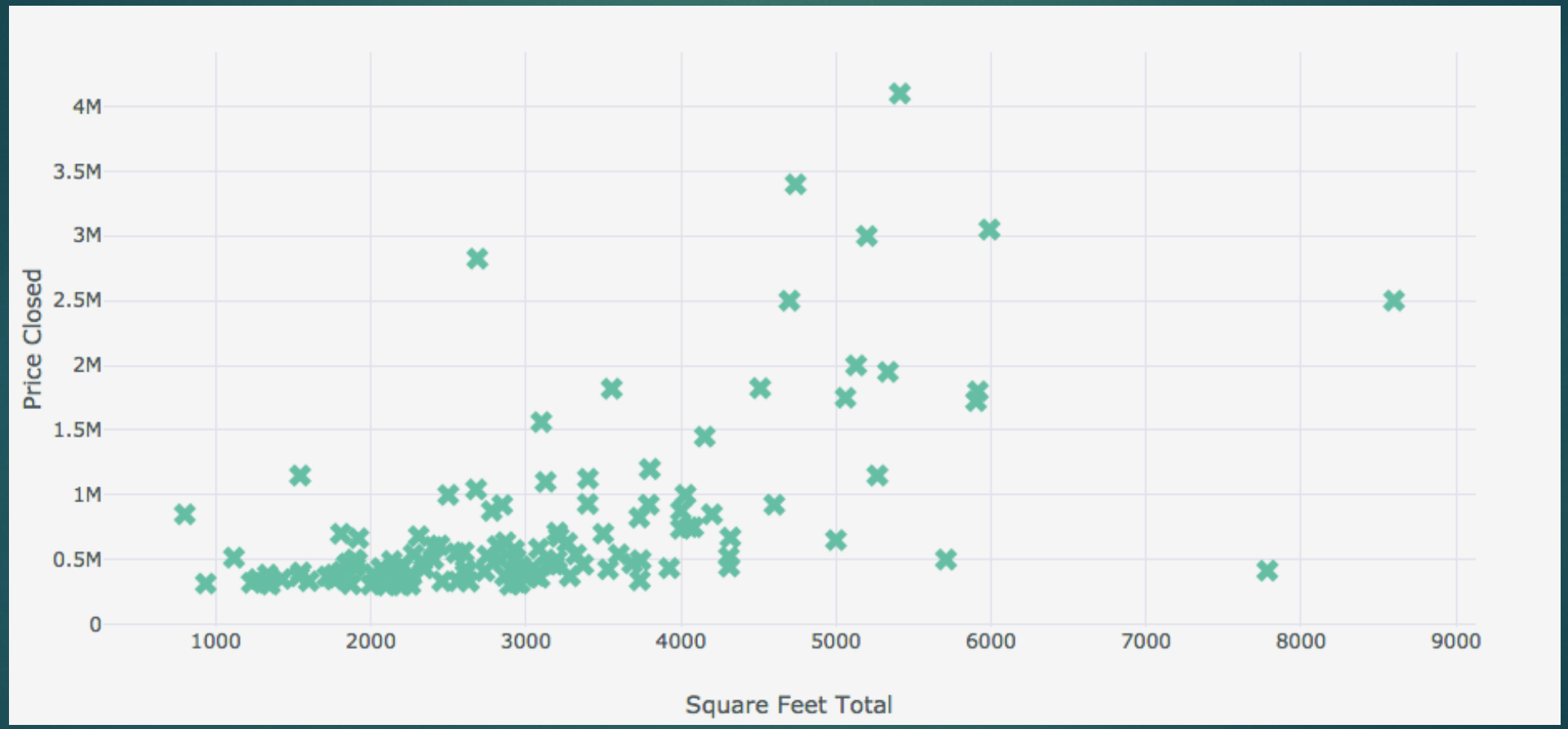
Further EDA

- Look for linear relationship, outliers and clusters



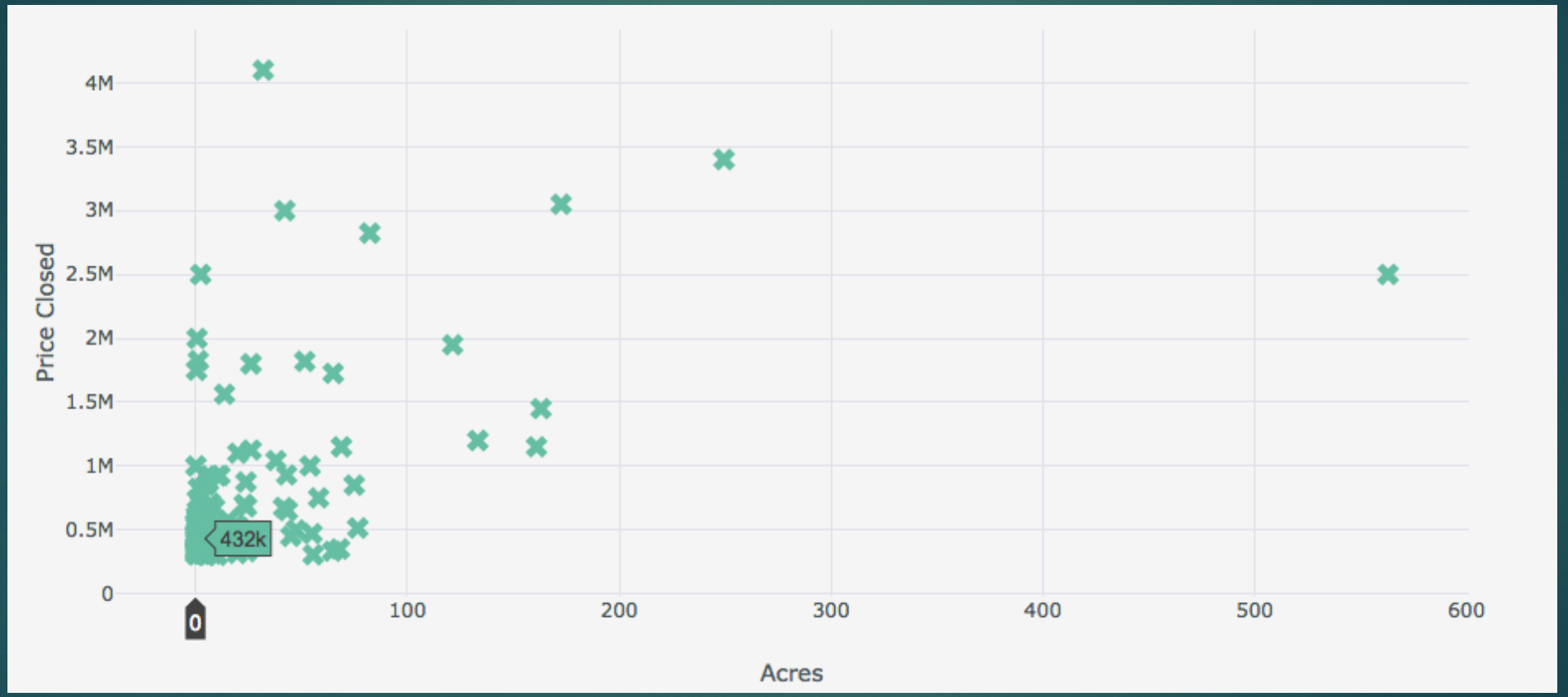
Further EDA

- Look for linear relationship, outliers and clusters



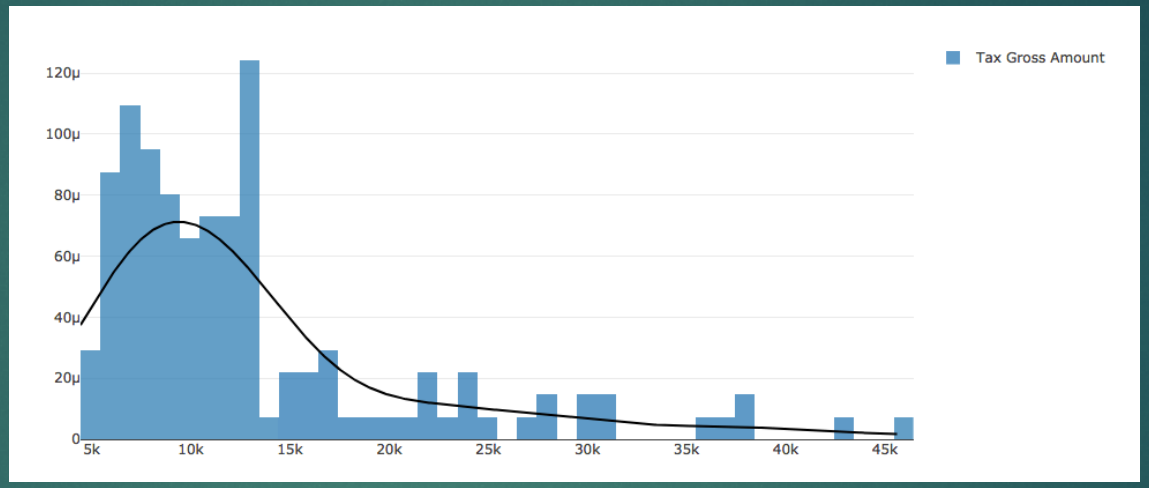
Further EDA

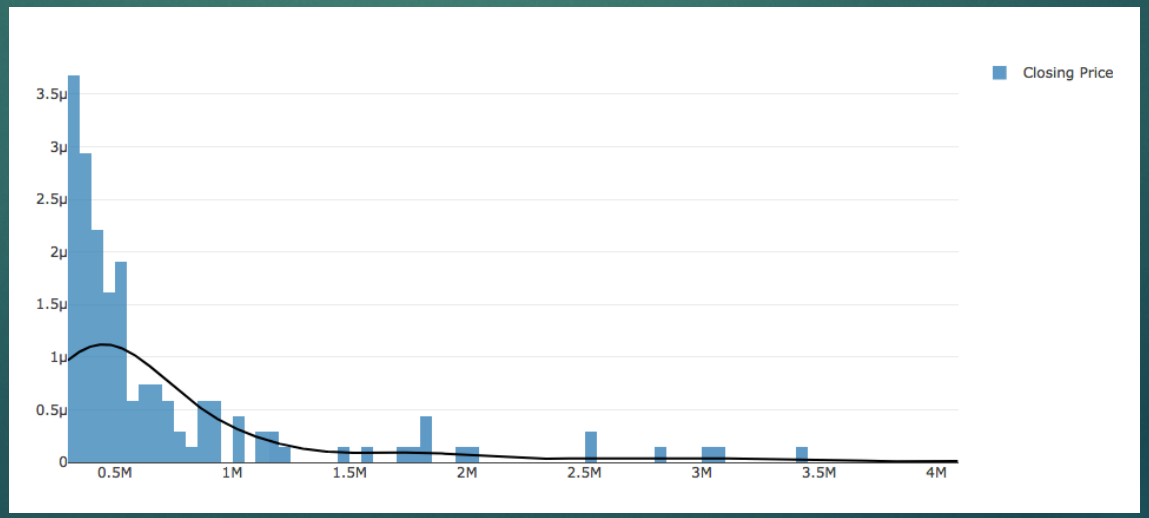
- Look for linear relationship, outliers and clusters



Further EDA

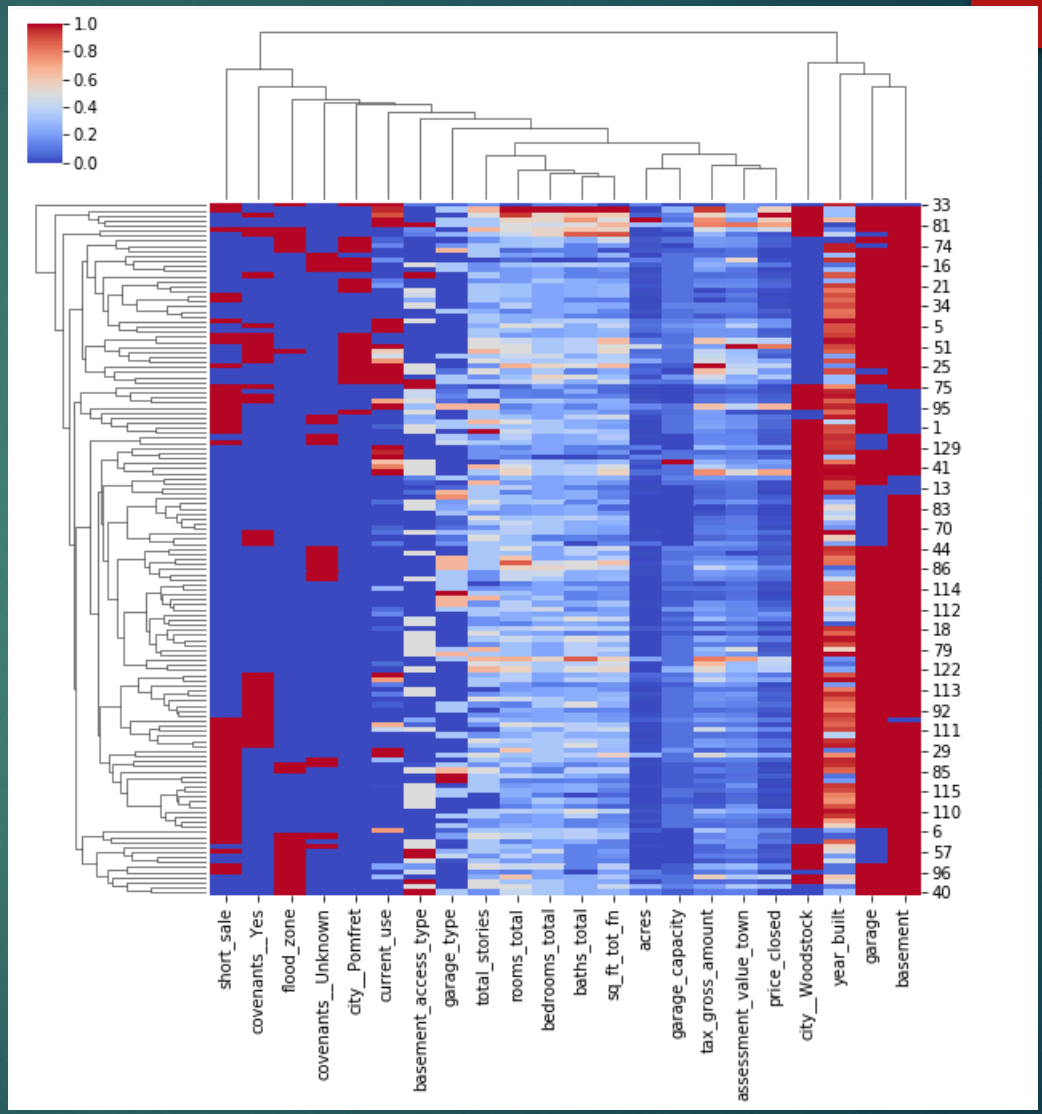
- Look for linear relationship, outliers and clusters





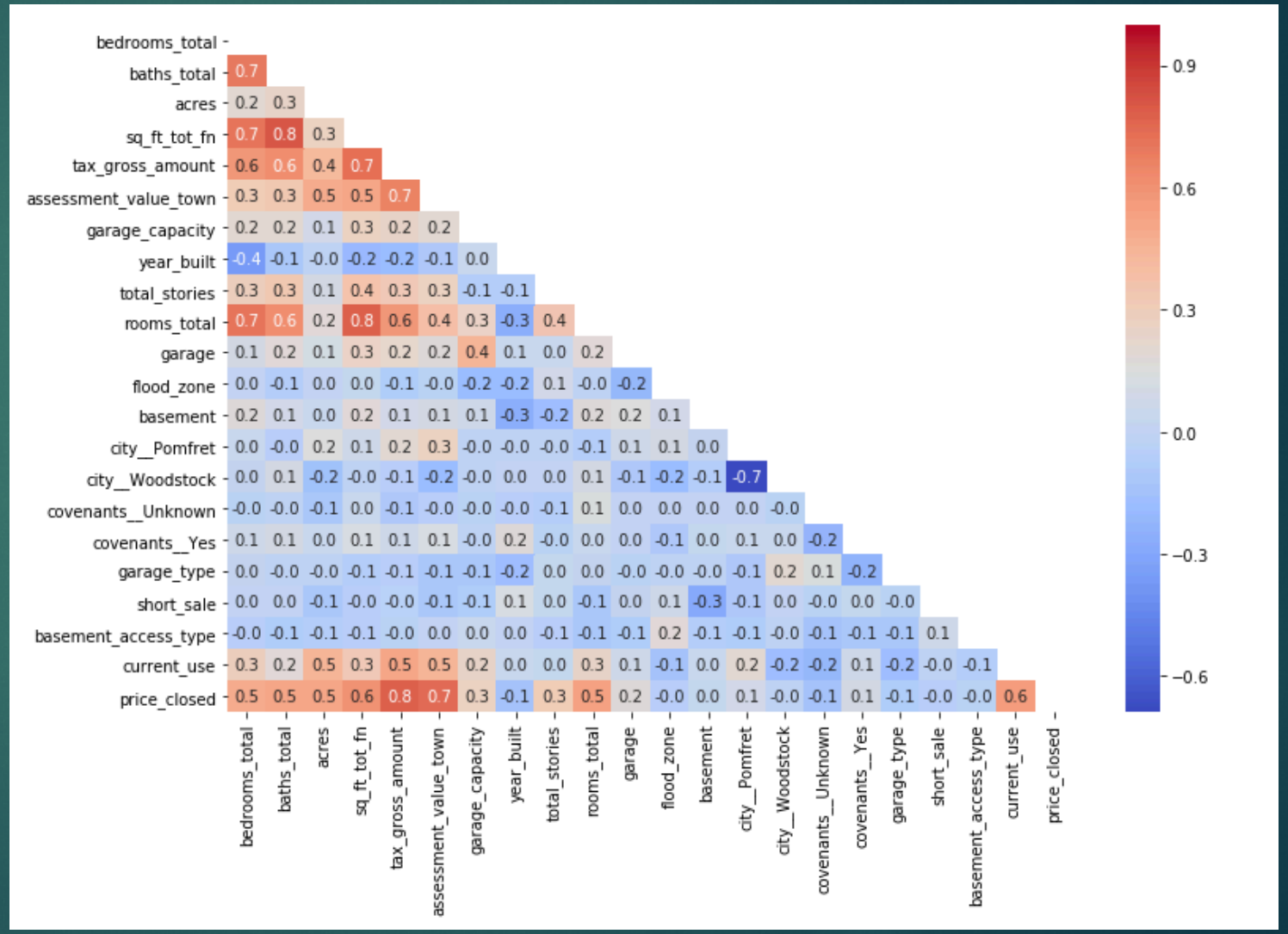
Further EDA

- Seaborn cluster: Potential challenge



Further EDA

- Correlation
- Important graph!



Dimensional Reduction
Data Normalization

- **Linear Regression** is run using newly engineered features, returned an improved result
- **Principal Component Analysis** is used to extract latent features and reduce noises
- **Standard Scaler** and **Principal Component Analysis** are then experimented separately, then combined, to improve the performance of our linear regression model.
- **Polynomial Feature** preprocessing is experimented at this step to confirm if it's necessary for that. It's not required since errors were higher

Proposal

- **Simple Model:**
  - Linear Regression
- **Alternative Complex Model:**
  - Random Forest Regressor
  - Support Vector Regression (Linear)

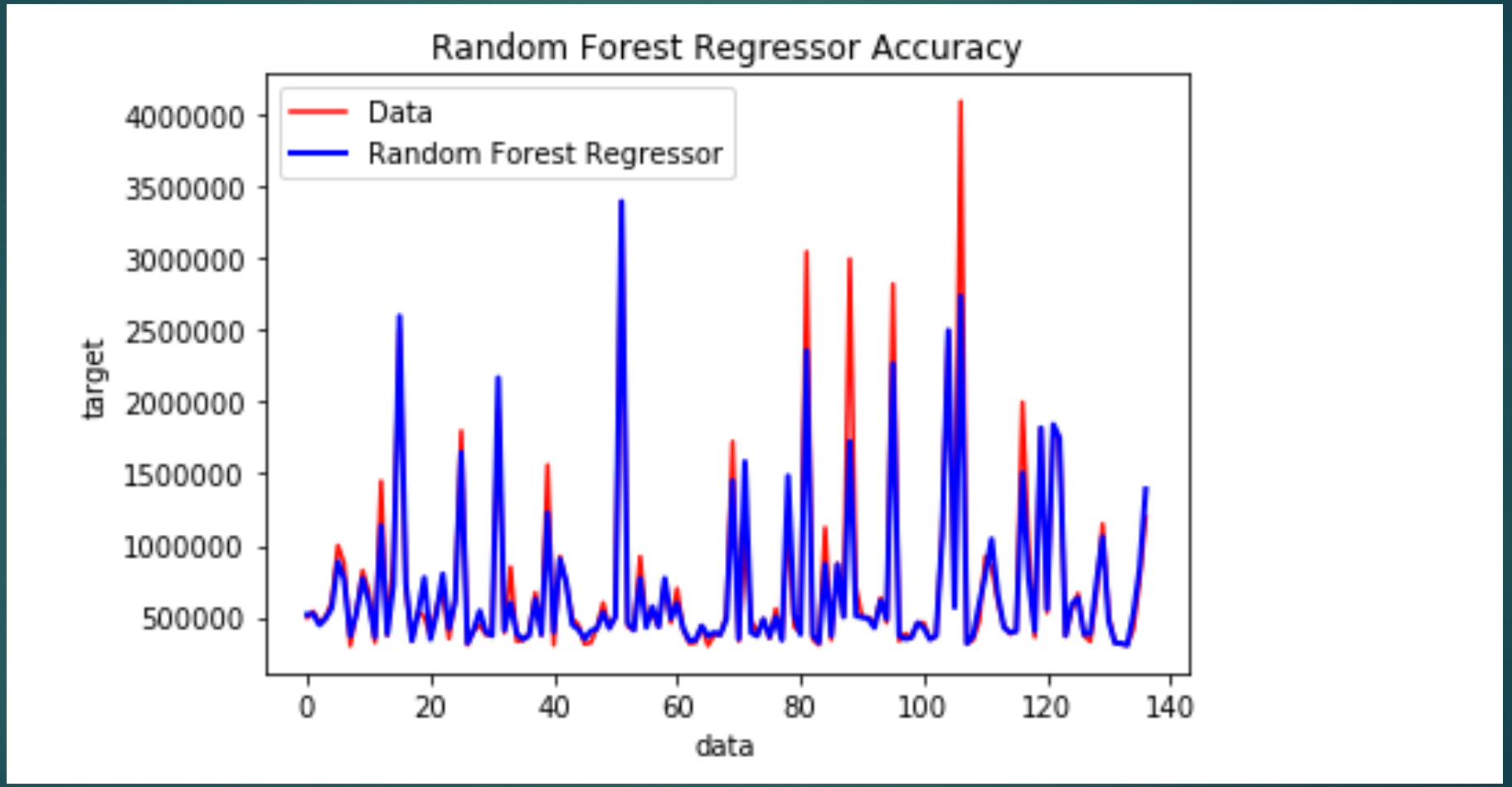
Random Forest Regressor

- **Random Forest Regressor** is chosen for an alternative model
- I am also interested in its ability to improve variance by averaging outcomes from multiple fully grown trees on variants of training set.
  - **Random subset** of the features at each split, as well as **bootstrapping** my datasets with replacement, to improve selection and validation for my model

Random Forest Regressor

- Random subset of the features at each split, as well as bootstrapping my datasets with replacement, to improve selection and validation for my model
- Simple loop and **GridSearch CV** is used to tune hyper parameters of the model
- Model is fitted with best parameters, calculation of **out-of-bag scores** and **features importance**. Rerun on filtered features

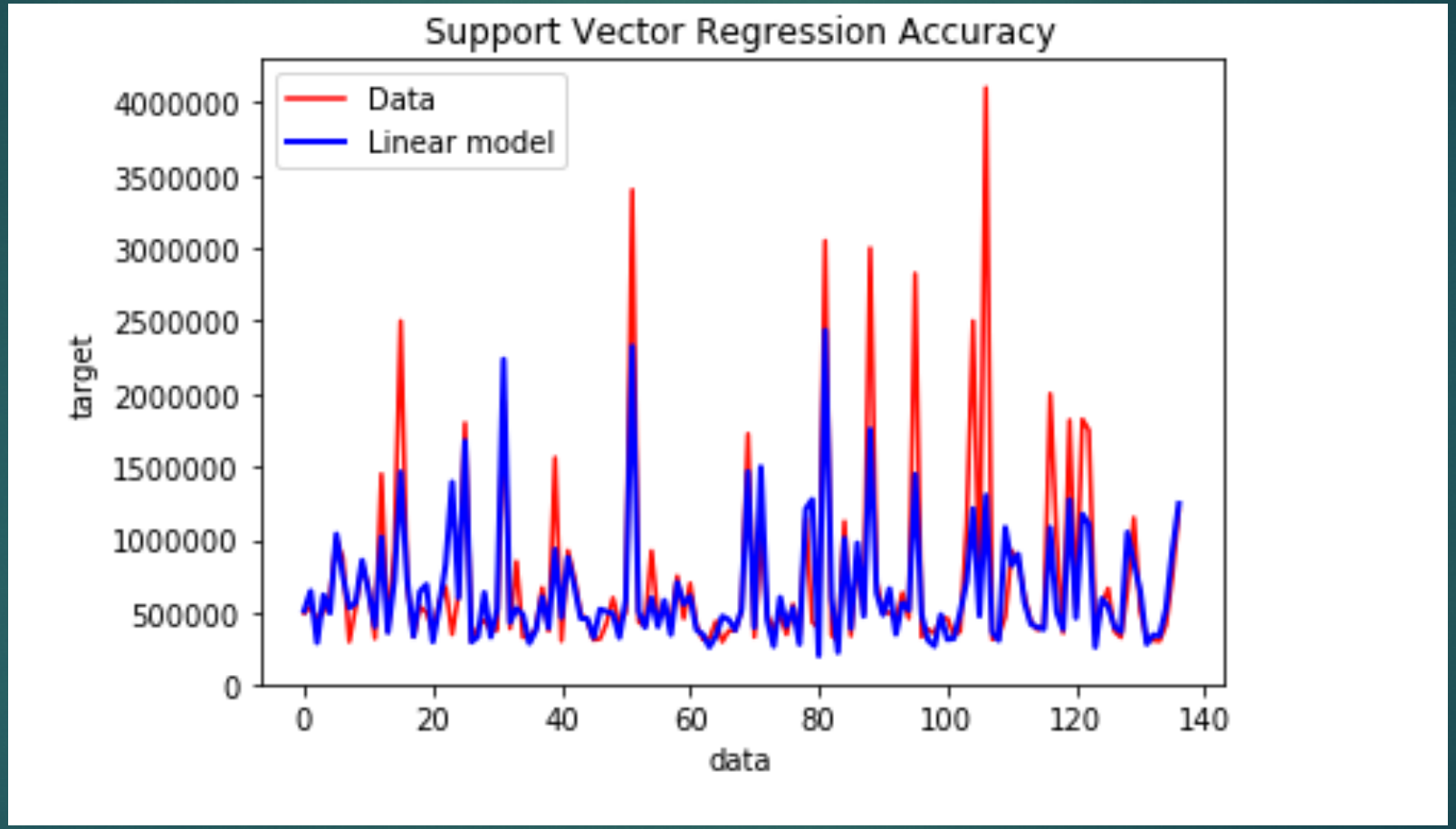
Overfitting complex models



Support Vector Regressor

- SVR is also chosen as a model. We can, however, see that there is a potential overfitting with low bias and high variance.

Overfitting complex models



Thus although the model performs well on training set, it might actually be worse off on new data

Results

- Simple Linear Regression is best performer with RMSE == 352325.48, PCA did not perform as well as expected
- Random Forest Regressor does not do well with RMSE == 424775.70, this confirms our hypothesis about the overfitting models underperform.
- Support Vector Regression, although takes the longest to train, is worst performer with RMSE == 763346.11
- Complexity of the models thus adversely affect its performance
- Stacking the estimators create a worse off result

Regularization

- Overfitting, as evident from our experiments, is a great issue. Largely due to our small datasets and many parameters relatively to the amount of data. Some of our estimators are overtuned on the training data and thus were not giving good prediction.
- Since we cannot obtain more data, we want to have a decent bias-variance tradeoff. Regularization is an approach that involves imposing a penalty on complex models. We want the least complex models to capture all the information in our data.

Regularization

- The data suffers from multicollinearity (independent variables are highly correlated).This is actually applicable because the numbers of bedrooms will highly correlate with numbers of rooms, areas, thus tax values, etc...
- Using *scipy*, some of the coefficients are truly significant and can affect the target response tremendously. The data set has a positive skew which means it has a longer right tail, which was evident from our visualization.
- Kurtosis is high as well, indicating great peakedness, which was also confirmed from our visualization in steps above. Some of the t and p values of features like acres, square foot total... are also small, indicating why our linear regression was able to perform so well based on highly correlated features to have high probability and low errors. We can reject the null hypothesis of independence for those features.

- We can proceed to experiment with models such as **Ridge Regression**, **LASSO** and **Elastic Net**. These estimators regularize by tweaking the coefficients of our model to prevent overfitting and reduce variance. Some bias will be introduced by default.

Regression, Ensemble and Stacking

- We continue to explore PCA and scaling on LASSO, Ridge Regression and Elastic Net, using the same procedures as with the previous models.
  - The effect is minimal as the model itself already regularize the dataset
- Best performance model is Elastic Net, which combines the power of LASSO and Ridge Regression. RMSE == 347195.12

Regression, Ensemble and Stacking

- Manual Ensemble:Using Object Oriented Programming, TransformerMixin and Feature Union inheritance classes are created to aggregate the accuracy of all the model.
  - By creating my custom class, I can also optimize model hyper parameters or pipeline. It can helps to save and load from this pre-trained models
- The Machine Learning Extension Library is also utilized to create a more sophisticated stacking of different regression models

Regression, Ensemble and Stacking

- The result is an improved performance in our testing prediction: We can therefore conclude that the issue of overfitting has been reduced.
  - RMSE Manual Ensemble == 348585.29
  - RMSE Stacking Regressor == 368604.82

Real time prediction

- Attempting to predict closing prices of current properties on the market using models we created. The actual price is obtain from the Internet from websites such as realtor.com, zillow.com, etc...
- Some models work better than others, Elastic Net seems to be the best performing one of all
- Linear Regression, Ridge and Lasso follows each other really closely
- Stacking Regression is more accurate for large real estate values
- Although there are errors, they are within the RMSE range, so this is an positive sign to continue fine tuning our models
- We did not include the most updated values, as well as accounting for the changes in time
- A lot of the listings do not have information available on the Internet

Next steps

- Further feature engineering
- Much more data is needed
- Enhance model's performance by fine tuning stacking and ensemble
- Explore boosting to reduce the overfitting issues of Random Forest Regressor, SVG, etc...
- Explore PCA of only important features in Random Forest Regressor
- Hyper parameter tuning for Elastic Net

THANK YOU. Q&A