W207 Final Project

Predicting Sale Price

Allison Godfrey, Jackie Ma, Surya Gutta, and Ian Anderson

Overview of our process

EDA

What does our data look like? Missing values?

Data Cleaning

How to deal with missing values? Outliers?

Feature Engineering

Which features are most relevant? How to deal with categorical and numerical features?

Create Train/ Dev Data

How to split? Use CV?

Model Setup

How does each model perform? Should we use a blended model?

Model Analysis

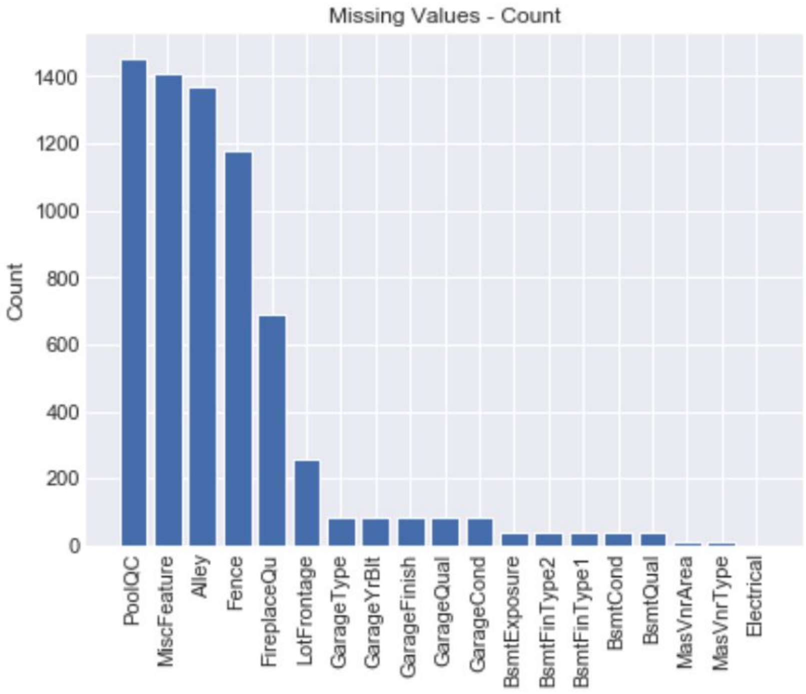
Which model is best? How do we measure this?

Final Submission

Base on the best performing model's RMSE

EDA and Data Cleaning: General Approach

- Look at shape of data
  - Train
    - 80 features
    - 1,460 examples
  - Test
    - 79 features
    - 1,459 examples
- Missing Values
  - See histogram
- Look at data structure
  - Mix of categorical and quantitative feature
- Exclude Outliers
  - GrLivArea > 4000 and SalePrice < 200000
  - LotArea > 50000
  - LotFrontage>200



An iterative process…

Dealing with Missing Values

- Exclude all columns with > 10% missing values
- Fill missing categorical values with NA
- Fill missing categorical values with the most common category
- Fill missing numerical values with 0
- Fill all missing numerical values with the median value
- Fill select features' missing values with median or mode

Encoding Non-Numerical Variables

- LabelEncoder()
- Hot_encode()
- Convert_ordinals()

Feature Selection

- No manual feature selection
- Take 20 most correlated features
- Add and delete features and evaluate RMSE
- Use L1 and L2 for model feature selection
- Remove multicollinear variables
- Include features with Spearman Rank > abs(0.05)

Selecting Models

- Adaboost (base estimator = LR
- Random Forest
- Bayesian Ridge
- Xgboost (base estimator = BR)
- Lasso Lars
- Linear
- Thielsen
- ARD
- Passive Aggressive
- Blended Model

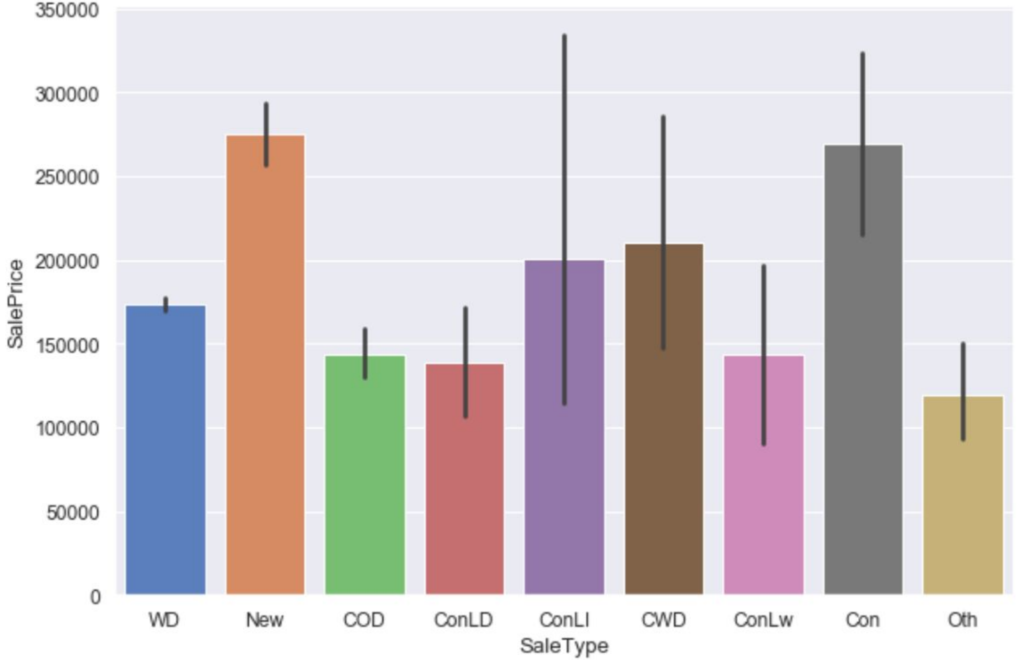
Other

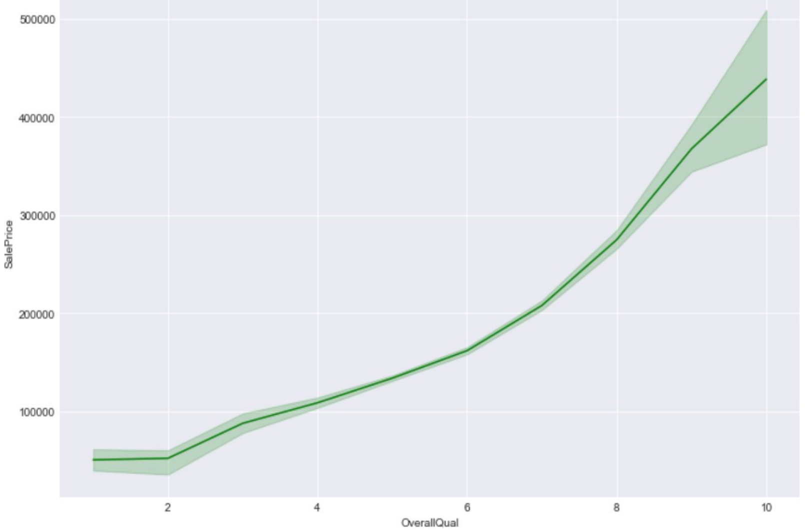
- Find and label duplicates
- Use log scale for Sale Price
- Use log scale for skewed features
- Use 5-fold cross validation instead of 80-20 split
- Tuning Hyperparameters for each model through CV
- Manually sift through data points whose predictions are most off & adjust features

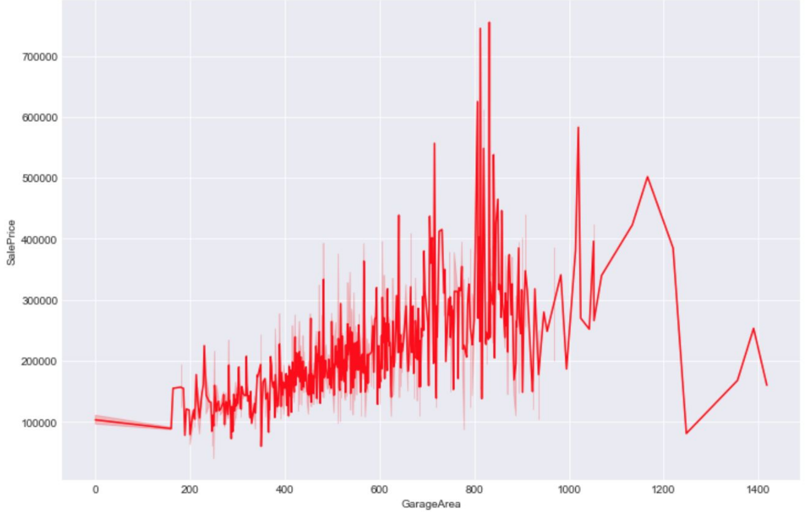
Features Chosen

- 1stFlrSF
- 2ndFlrSF
- BldgType_X
- BsmtExposure
- BsmtFinSF1
- BsmtQual_X
- GarageType_Attc
- SaleType_X
- YrSold_X
- TotalBsmtSF
- ExterQual
- FireplaceQu_Gd
- FireplaceQu_NA
- Fireplaces
- Foundation_X
- FullBath
- Functional
- GarageArea
- GarageCars
- total_porch_sf
- GarageType_Detc
- GrLivArea
- has_fireplace
- HeatingQC
- house_age
- KitchenQual
- LotArea
- LotFrontage
- LotShape
- MasVnrType_Non
- MasVnrType_Ston
- MSSubClass_X
- MSZoning_RM
- Neighborhood_X
- OverallCond
- OverallQual
- SaleCondition_Partial
- SaleType_New
- total_baths
- MasVnrArea
- total_sf
- TotRmsAbvGrd
- WoodDeckSF
- YearBuilt
- YearRemodAdd
- GarageFinish

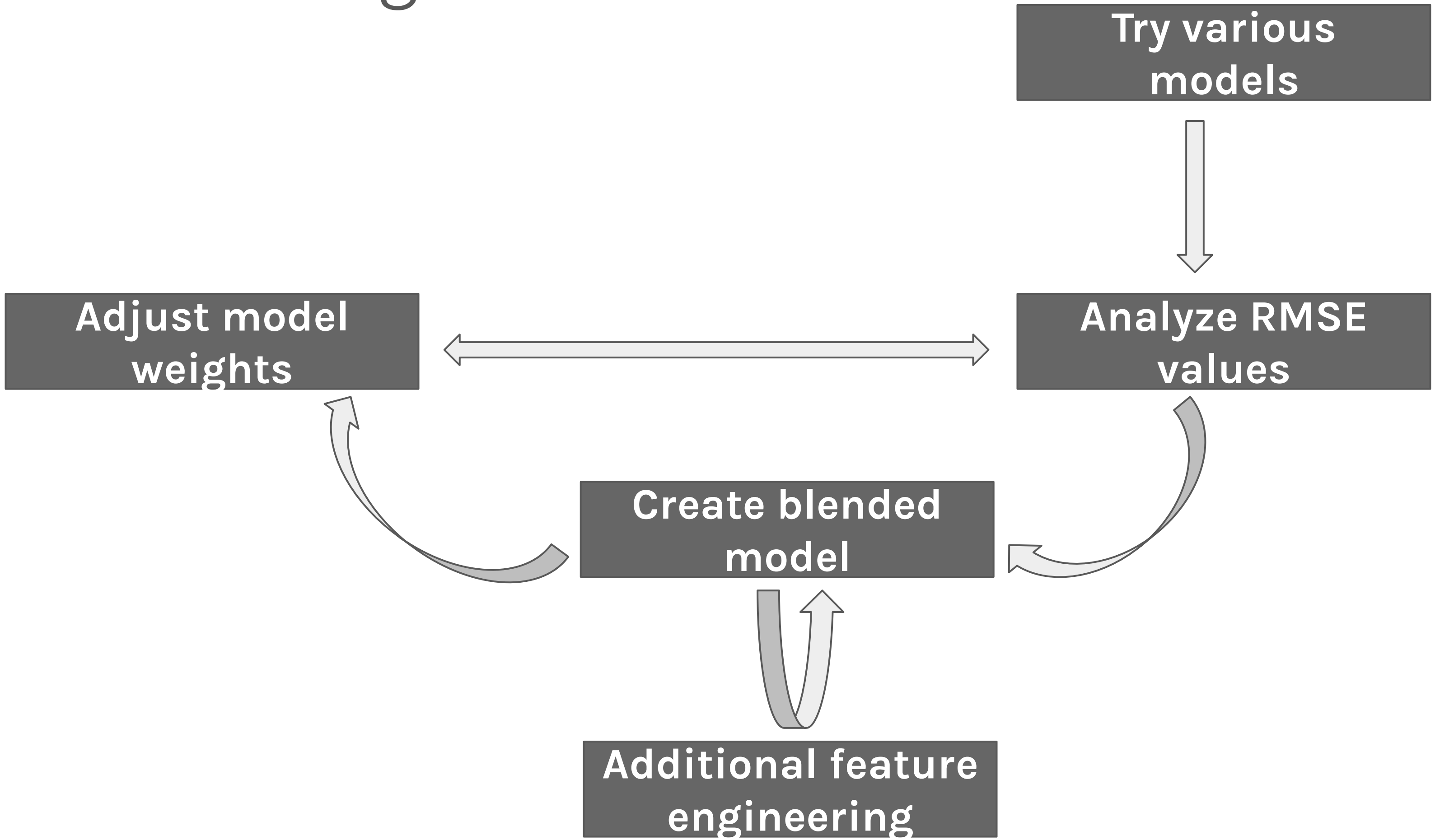
A closer look at specific features







Model Building Process



Converting Categorical/Ordinal Variables

```
def convert_ordinals(data):
    default_odinal = {
        'Ex': 5,  # Excellent
        'Gd': 4,  # Good
        'TA': 3,  # Average/Typical
        'Fa': 2,  # Fair
        'Po': 1,  # Poor
        'NA': 3
    }

    # LotShape: General shape of property
    lot_shape = {
        'Reg': 4,  # Regular
        'IR1': 3,  # Slightly irregular
        'IR2': 2,  # Moderately Irregular
        'IR3': 1   # Irregular
    }
```

```
def hot_encode(data):
    categorical_cols = data.select_dtypes(include=
['object'])
    return pd.get_dummies(data, columns = categoric
al_cols.columns)

test_new = hot_encode(test_new)
train_new = hot_encode(train_new)
train_new.head()
```

Using CV and tuning parameters

```
#Validation function
def cross_validation(model):
    kf = KFold(5, shuffle=True, random_state=42).get_n_splits(train_data)
    rmse= np.sqrt(abs(cross_val_score(model, train_data, train_labels, scoring="neg_mean_squared_e
rror", cv = kf)))
    return(rmse)
```

```
def parameter_tuning(model, parameters):
    clf = GridSearchCV(
        model, parameters, cv=5,scoring='neg_mean_squared_error', n_jobs = 5)

    clf.fit(train_data,train_labels)

    print(clf.best_params_)
    print(np.sqrt(-clf.best_score_))
```

Choosing the final weights

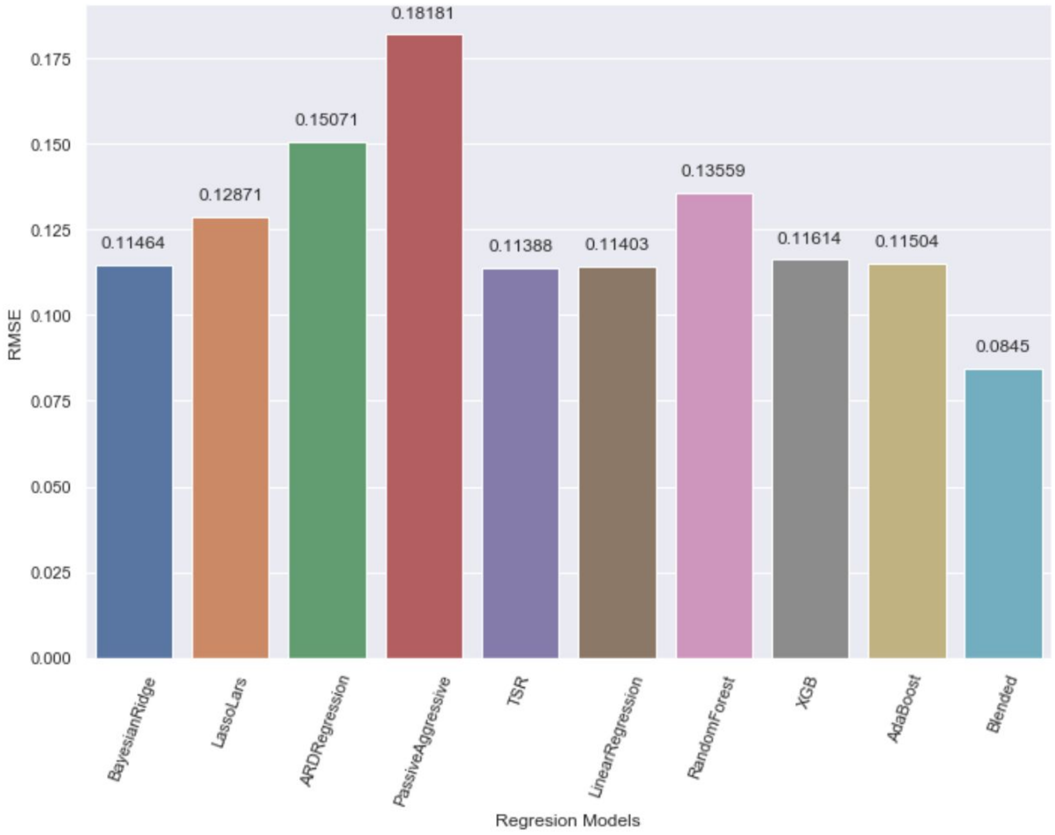
```
sorted_models = sorted(models_rmse.items(), key =
              lambda kv:(kv[1], kv[0]))
for item in sorted_models:
    if item[0] == 'Blended':
        continue
    print ('{:>18}  {:>12}'.format(item[0], item[1]))

regression_models = [
    {'name': 'AdaBoostRegressor', 'weight': 0.14, 'model': ada_model_fit},
    {'name': 'XGBoost', 'weight': 0.14, 'model': xgb_model_fit},
    {'name': 'BayesianRidge', 'weight': 0.14, 'model': br_model_fit},
    {'name': 'Linear', 'weight': 0.14, 'model': lr_model_fit},
    {'name': 'TheilSen', 'weight': 0.16, 'model': trs_model_fit},
    {'name': 'Random Forest', 'weight': 0.14, 'model': rfr_model_fit},
    {'name': 'ARD', 'weight': 0.0, 'model': ardr_model_fit},
    {'name': 'PassiveAggressive', 'weight': 0.0, 'model': par_model_fit},
    {'name': 'LassoLars', 'weight': 0.14, 'model': ll_model_fit},
]
```

Comparing the Models

Lowest RMSE (Blended):
0.0845

While our RMSE decreases with model adjustments, our score in Kaggle stays relatively constant

Are we overfitting our train data?



Final Kaggle Score

RMSE: 0.11743

Place: 525