Residential price prediction

Moussa Diop, Abdullah Jaura,
Bryan Johns, Bolima Tafah

Overview

Objective:

To determine residential prices in Ames, Iowa spanning 2006 to 2010 using machine learning models

Dataset

- Provided by kaggle
- Sourced from the Ames City Assessor's Office in 2011
- 1461 samples, with 80 categorical variables

Search this file

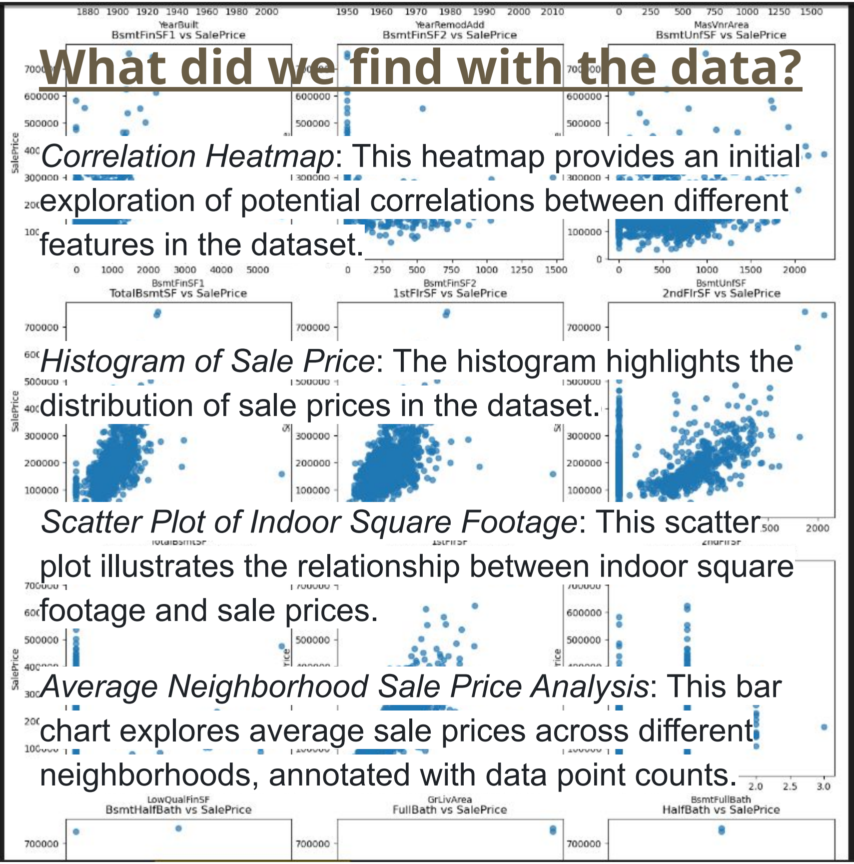
1	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	HouseStyle
2	1	60	RL	65	8450	Pave	NA	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam	2Story

rCond	Foundation	BsmtQual	BsmtCond	BsmtExposure	BsmtFinType1	BsmtFinSF1	BsmtFinType2	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	Heating	HeatingQC	CentralAir	Electrical	1stFlrSF	2ndFl
	PConc	Gd	TA	No	GLQ	706	Unf	0	150	856	GasA	Ex	Y	SBrkr	856	854

EDA

Graphs and variables

What did we find with the data?

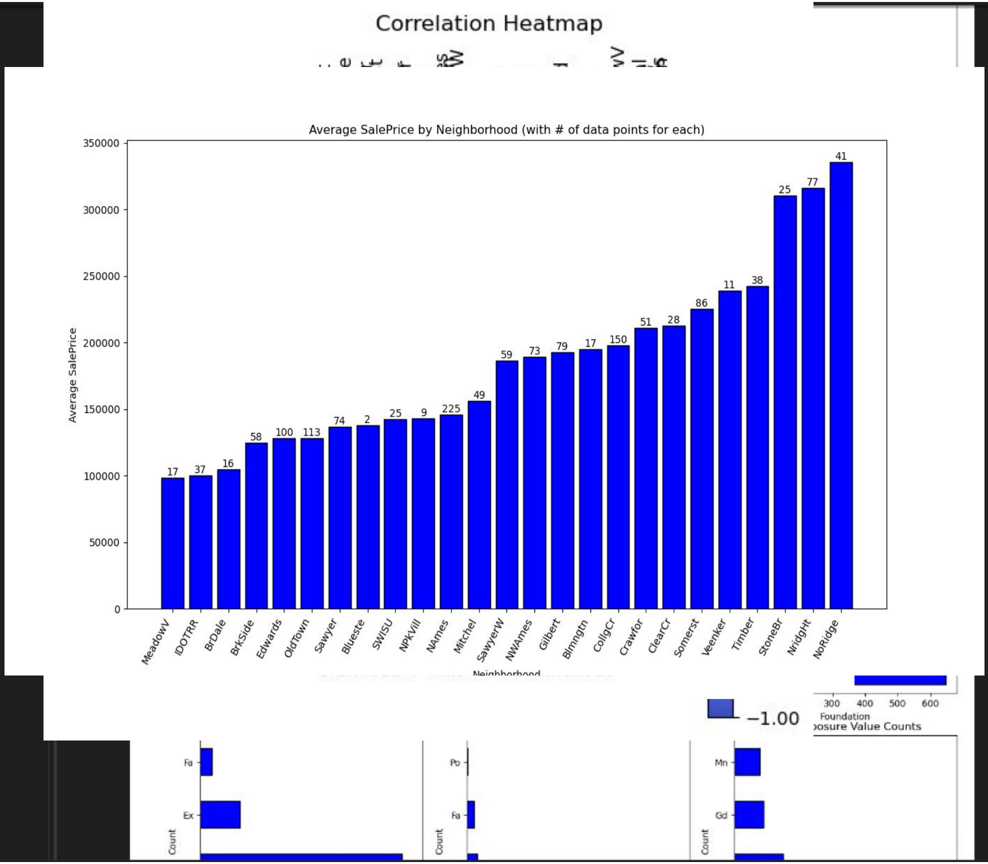


Correlation Heatmap: This heatmap provides an initial exploration of potential correlations between different features in the dataset.

Histogram of Sale Price: The histogram highlights the distribution of sale prices in the dataset.

Scatter Plot of Indoor Square Footage: This scatter plot illustrates the relationship between indoor square footage and sale prices.

Average Neighborhood Sale Price Analysis: This bar chart explores average sale prices across different neighborhoods, annotated with data point counts.



Machine learning

```python
# build keras-tuner function
def build_model(hp):
    nn_test = tf.keras.models.Sequential()

    # adds a range of 1 to 5 dense layers, allowable number of neurons (adjust based on features), activation functions
    for i in range(hp.Int("num_layers", min_value=1, max_value=5, step=1)):
        nn_test.add(
            tf.keras.layers.Dense(
                units=hp.Int(f"layer{i}", min_value=50, max_value=600, step=50),
                input_dim=len(X_train[0]),
                activation=hp.Choice(f"activation{i}", values=["relu", "tanh", "LeakyReLU"])
            )
        )

    # add final layer
    nn_test.add(tf.keras.layers.Dense(units=1, activation="linear"))

    # compile the model
    nn_test.compile(
        loss="mean_absolute_error",
        optimizer="adam",
        metrics=["mae"],
    )

    return nn_test
```

24	added KitchenQual	R-squared: 0.8360448908773381 Mean Squared Error: 600360601.224905 Mean Absolute Error: 17801.030104166668 Mean Percentage Error: -9.368243654875693	{'num_layers': 3, 'layer0': 166, 'activation0': 'tanh', 'layer1': 340, 'activation1': 'relu', 'layer2': 272, 'activation2': 'LeakyReLU'}	layers 1-5, neurons 50-600	
25	same as 16, added TotalRooms = BedroomAbvGr + KitchenAbvGr + FullBath + HalfBath + BsmtFullBath + BsmtHalfBath	R-squared: 0.8846981189499872 Mean Squared Error: 422205242.6421167 Mean Absolute Error: 15843.1384375 Mean Percentage Error: -10.23366429458488	{'num_layers': 5, 'layer0': 332, 'activation0': 'relu', 'layer1': 352, 'activation1': 'LeakyReLU', 'layer2': 358, 'activation2': 'relu', 'layer3': 302, 'activation3': 'LeakyReLU', 'layer4': 182, 'activation4': 'LeakyReLU'}	layers 1-5, neurons 40-400	
26	binned Neighborhood into 'Other' with cutoff of 30 (still dropped BlueSte)	R-squared: 0.857342506433154 Mean Squared Error: 522374319.8081996 Mean Absolute Error: 16166.054192708334 Mean Percentage Error: -10.69128643647996	{'num_layers': 5, 'layer0': 302, 'activation0': 'relu', 'layer1': 48, 'activation1': 'tanh', 'layer2': 66, 'activation2': 'LeakyReLU', 'layer3': 40, 'activation3': 'relu', 'layer4': 40, 'activation4': 'relu'}	layers 1-5, neurons 40-400	
27	dropped MasVnrType	R-squared: 0.8800672697299794 Mean Squared Error: 439162197.73051107 Mean Absolute Error: 15165.742708333333 Mean Percentage Error: -10.556236674703808	{'num_layers': 5, 'layer0': 276, 'activation0': 'relu', 'layer1': 40, 'activation1': 'relu', 'layer2': 40, 'activation2': 'relu', 'layer3': 40, 'activation3': 'relu', 'layer4': 40, 'activation4': 'relu'}	layers 1-5, neurons 40-400	
28	added random forest	R-squared: 0.8819972189597575 Mean Squared Error: 432095229.911556 Mean Absolute Error: 15199.682083333333 Mean Percentage Error: -10.992810695178374	{'num_layers': 4, 'layer0': 334, 'activation0': 'relu', 'layer1': 398, 'activation1': 'relu', 'layer2': 120, 'activation2': 'relu', 'layer3': 58, 'activation3': 'relu'}	layers 1-5, neurons 40-400	Random Forest R-squared 0.8707814716153914 Random Forest Mean Squ 473164354.59380394 Random Forest Mean Abs 15491.472666666667 Random Forest Mean Perc -0.6340198782811256

Trials and tribulations

- Tweaking model: We ran over 30 trials.
- Epochs - Training the Algorithm to go through the entire dataset.
- Layers - Vanishing/Exploding gradients, overfitting, complex architecture etc.
- Variables - (numeric, String, Boolean, objects etc)

- **In summary:** it was mostly adding and dropping features, feature engineering and adjusting parameters of keras tuner.

Results of NN and Random Forest

the diagrams above show a positive relationship between actual and predicted house prices(Positively correlated). This means any discrepancy in house prices could imply we used erroneous models. So our model was a success.

Neural Network

Random Forest

```python
# build keras-tuner function
def build_model(hp):
    nn_test = tf.keras.models.Sequential()

    # adds a range of 1 to 5 dense layers, allowable number of neurons (adjust based on featu
    for i in range(hp.Int("num_layers", min_value=1, max_value=5, step=1)):
        nn_test.add(
            tf.keras.layers.Dense(
                units=hp.Int(f"layer{i}", min_value=50, max_value=600, step=50),
                input_dim=len(X_train[0]),
                activation=hp.Choice(f"activation{i}", values=["relu", "tanh", "LeakyReLU"])
            )
        )

    # add final layer
    nn_test.add(tf.keras.layers.Dense(units=1, activation="linear"))

    # compile the model
    nn_test.compile(
        loss="mean_absolute_error",
        optimizer="adam",
        metrics=["mae"],
    )

    return nn_test


# define tuner / call the build_model function
tuner = RandomSearch(build_model, objective="mae", max_trials=10, overwrite=True)

# run the damn thing
tuner.search(
    X_train,
    y_train,
    epochs=100,
    validation_data=(X_val, y_val),
)
```

```python
# create random forest model
rf = RandomForestRegressor(random_state=42)

# train random forest model
rf.fit(X_train, y_train)

# predict
y_test_pred_rf = rf.predict(X_test)
```

References/acknowledgements

Dataset provided by [Kaggle]

Thanks to Justin Bisal, James Newman, and Geronimo Perez for feedback and assistance