Housing Market Analysis: Seattle CBSA

Gary Davis
Nikhil Golthi

September 18, 2021

Overview

Conducted model performance analysis for two sets of models built using Seattle-Tacoma-Bellevue CBSA housing data:

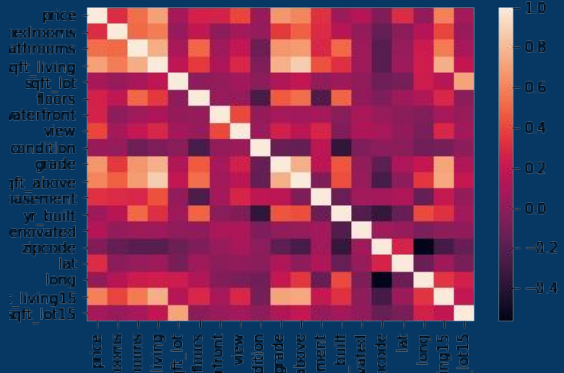
1) Comparison of Regression Models
2) Comparison of LSTM Models

Regression Analysis: King County Home Sales

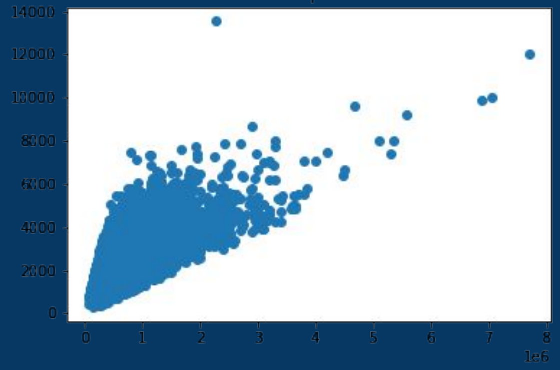
- Used dataset consisting of home sales and characteristics data for roughly 21,000 King County, WA properties to build regression models to predict home sales price
- Regression Methods:
  - Linear Regression
  - Multiple Linear Regression
  - Logistic Regression
  - Polynomial Regression
  - Random Forest Regression
- Compared performance of regression models to determine most effective method of regression

Linear Regression: Seattle CBSA Housing Inventory Data

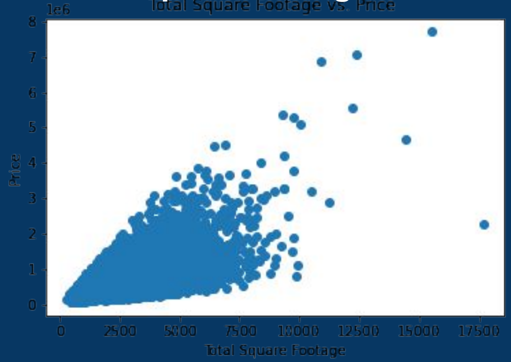
Correlation Heat Map



Square Foot vs. Price



Total Square Footage vs. Price



Linear Regression: Model Results

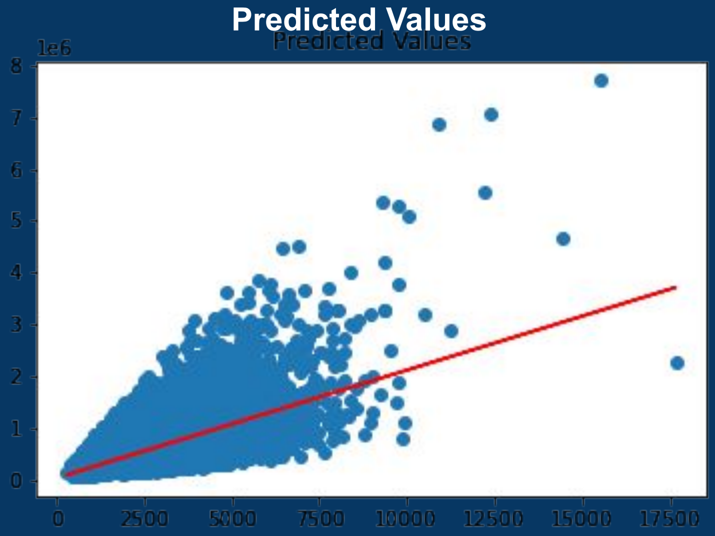
Coefficient = 207.69

Intercept = 4756.36

$R^2 = 0.446$

$y = mx + b$

$y = 207.69x + 4756.36$



Multiple Linear Regression: Definition & Results

Multiple linear regression is an extension of linear regression, but uses several variables (features) to predict the outcome.

Feature	Coefficient
bedrooms	-32878.696
bathrooms	25568.4758
sqft_living	-5.63E+18
sqft_lot	5799.76948
floors	-894.31225
waterfront	50526.919
view	36372.747
condition	21339.765
grade	115739.431
sqft_above	5.09E+18
sqft_basement	2.70E+18
yr_built	-74545.421
yr_renovated	6656.71045
lat	77616.1179
long	-17532.358
sqft_living15	17353.9688
sqft_lot15	-9729.2511

Mean Squared Error (MSE) = 40926641382.97
Root Mean Squared Error (RMSE) = 202303.34
Variance Score = 0.7016

Logistic Regression: Model Results

Training Data Score: 0.010117835770251096

Testing Data Score: 0.007772020725388601

	Prediction	Actual
0	650000.0	459000.0
1	350000.0	445000.0
2	590000.0	1057000.0
3	550000.0	732350.0
4	350000.0	235000.0
5	550000.0	555000.0
6	550000.0	365000.0
7	750000.0	685000.0
8	350000.0	525000.0
9	550000.0	449950.0
10	190000.0	280000.0
11	350000.0	428000.0
12	325000.0	575000.0
13	350000.0	313100.0
14	690000.0	637500.0
15	690000.0	732000.0
16	350000.0	400000.0
17	550000.0	829000.0
18	350000.0	469500.0
19	350000.0	537000.0

Polynomial Regression: Set-up

- Target variable: “price”
- Feature variables: all variables except “id”, “date”, “zipcode”, “long”, “condition”, “yr_built”, “sqft_lot15”, “sqft_lot”
- Used **PolynomialFeatures** from sklearn.preprocessing to scale data
- Model used:
  - **LinearRegression** model from sklearn.linear_model
- Model parameters:
  - PolynomialFeatures(Degree = 3)

```
poly_reg = PolynomialFeatures(degree=3)
X_train_scaled = poly_reg.fit_transform(X_train)
X_test_scaled = poly_reg.fit_transform(X_test)
polynomial_reg = LinearRegression()
```

Polynomial Regression: Model Performance

```
Polynomial Regression Out-of-Sample Results, Refined Features:
Polynomial Regression Out-of-Sample R2 Score: 66.6%
Polynomial Regression Out-of-Sample Root Mean Squared Error: $201339.04
Polynomial Regression Out-of-Sample Mean Absolute Error: $110501.39
Polynomial Regression Out-of-Sample Mean Absolute Percentage Error: 21.48%
Polynomial Regression Out-of-Sample Accuracy: 78.52%

Polynomial Regression In-Sample Results, Refined Features:
Polynomial Regression In-Sample R2 Score: 81.05%
Polynomial Regression In-Sample Root Mean Squared Error: $162424.9
Polynomial Regression In-Sample Mean Absolute Error: $105945.11
Polynomial Regression In-Sample Mean Absolute Percentage Error: 21.16%
Polynomial Regression In-Sample Accuracy: 78.84%
```

Random Forest Regression: Set-up

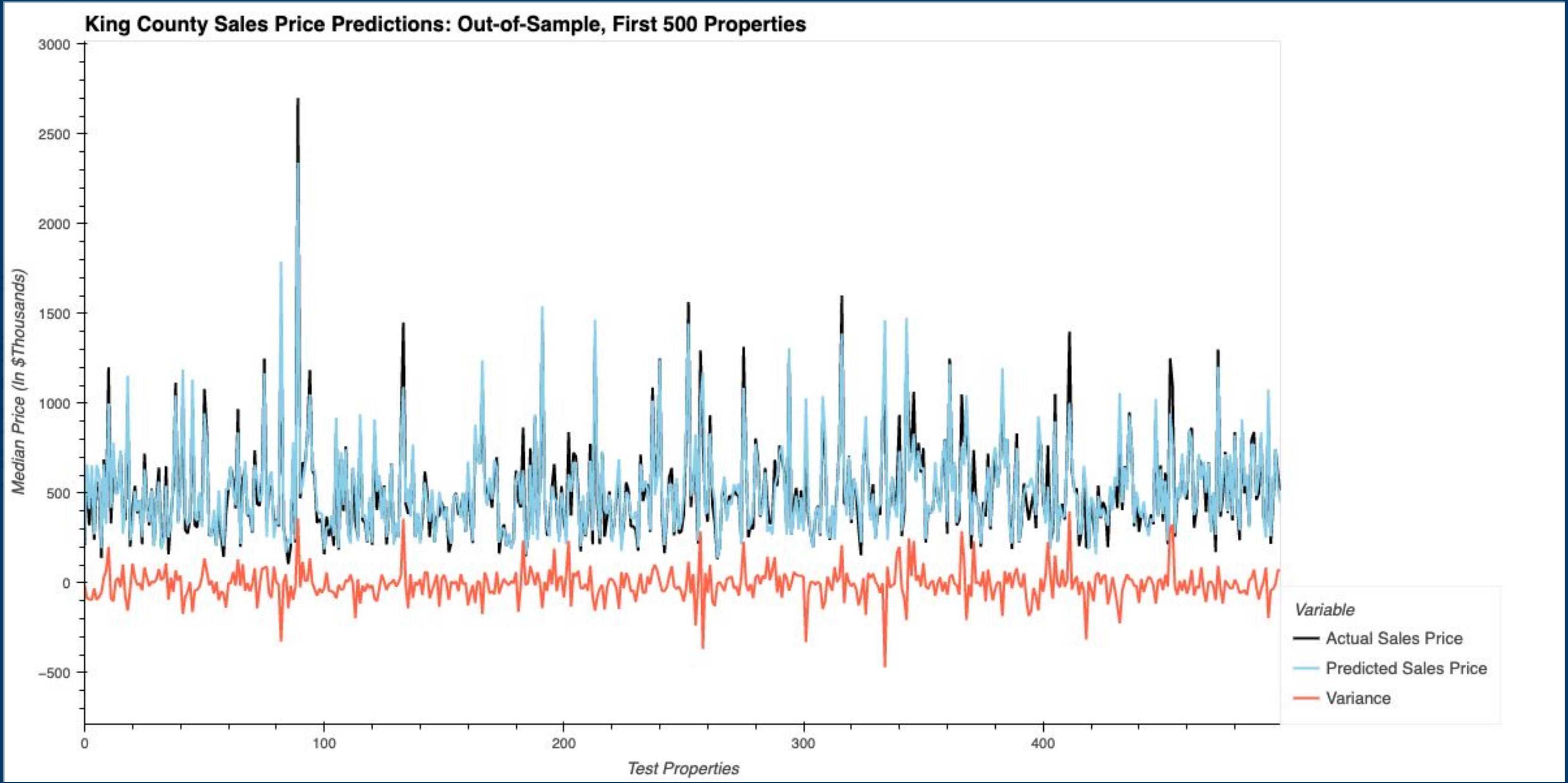
- Target variable: “price”
- Feature variables: all variables except “id” and “date”
- No scaling required for Random Forest Regressor
- Model(s) used: **RandomForestRegressor** from sklearn.ensemble
- Model parameters:
  - n_estimators = 1000

```
# Initiate Random Forest Regressor Model
rf_model = RandomForestRegressor(n_estimators=1000, random_state=23)
```

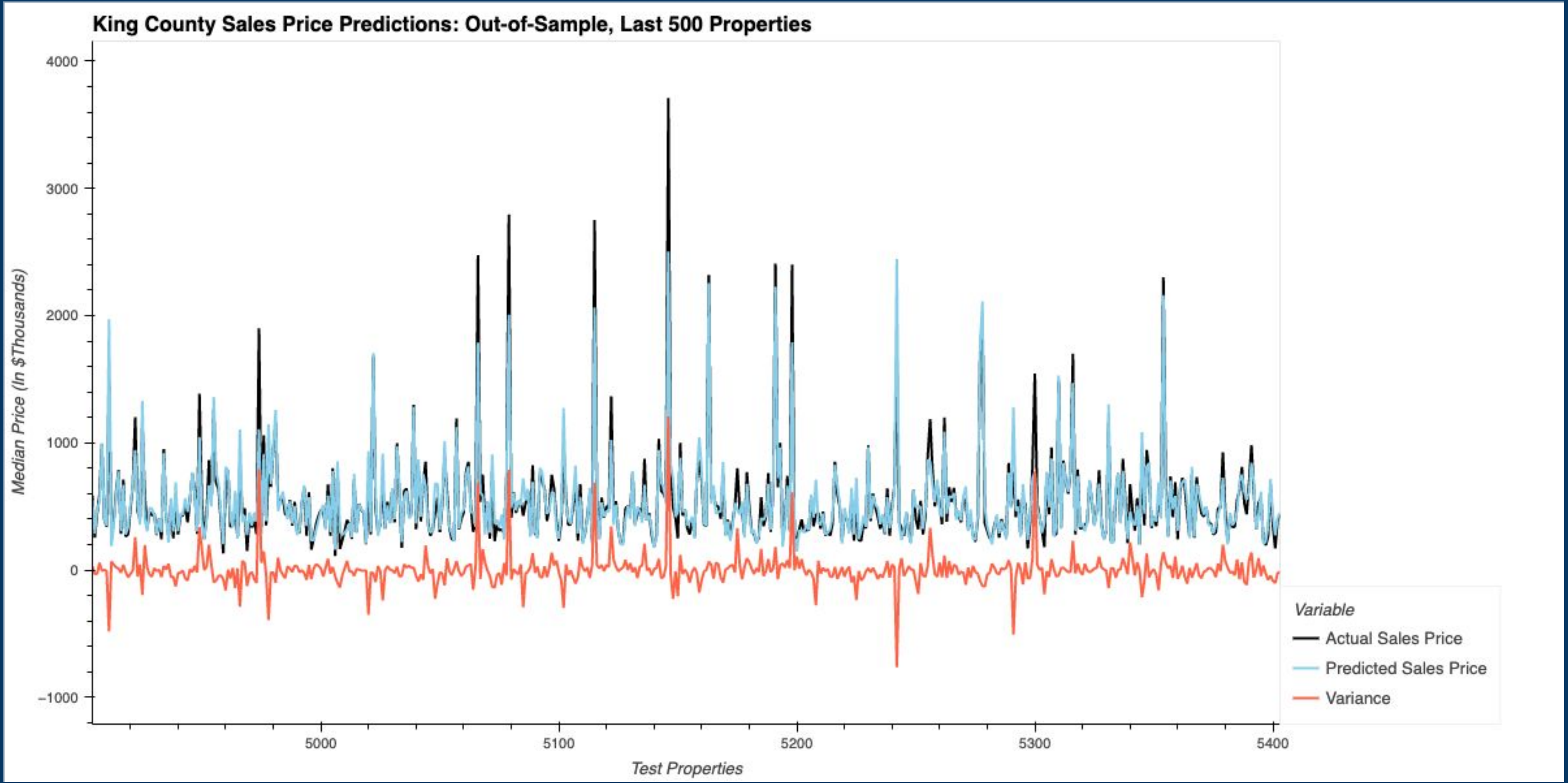
Random Forest Regression: Model Performance

```
Random Forest Regression Out-of-Sample Results:
RF Regression Out-of-Sample R2 Score: 87.1%
RF Regression Out-of-Sample Root Mean Squared Error: $125130.65
RF Regression Out-of-Sample Mean Absolute Error: $67752.31
RF Regression Out-of-Sample Mean Absolute Percentage Error: 12.82%
RF Regression Out-of-Sample Accuracy: 87.18%

Random Forest Regression In-Sample Results:
RF Regression In-Sample R2 Score: 98.31%
RF Regression In-Sample Root Mean Squared Error: $48472.69
RF Regression In-Sample Mean Absolute Error: $25692.84
RF Regression In-Sample Mean Absolute Percentage Error: 4.89%
RF Regression In-Sample Accuracy: 95.11%
```

Random Forest Regression: Model Performance



Random Forest Regression: Model Performance



Regression Analysis: Conclusion

- Random Forest Regression appears to build best model for predicting property sales price
  - Highest R-squared score & accuracy
  - Lowest MAPE & RMSE
- Multiple Linear Regression and Polynomial Regression produced strong models as well

LSTM Analysis: Seattle CBSA Housing Inventory Data

- Used Seattle-Tacoma-Bellevue CBSA housing inventory data to build LSTM models to predict monthly median home listing price
- Housing Inventory Data consisted of:
  - Monthly Median Home Listing Price
  - Monthly Active Listings Count
  - Monthly Median Days on Market
- LSTM Methods:
  - LSTM using median listing price as predictor
  - LSTM using active listings count as predictor
  - LSTM using median days on market as predictor
- Compared performance of LSTM models to determine which inventory statistic is best predictor of median home listing price

1) Active Listings Count LSTM: Set-up

- Target variable: “median_listing_price”
- Feature variable: “active_listings_count”
- Used **MinMaxScaler** from sklearn.preprocessing to scale data
- Model(s) used:
  - **Sequential** from tensorflow.keras.models
  - **LSTM, Dense, Dropout** from tensorflow.keras.layers
- Model / Run parameters:
  - Lookback window = 7 months
  - Dropout_fraction = 0.2
  - Used 3 layers
  - Optimizer / loss = “adam”, “mean_squared_error”
  - epochs = 100
  - Batch_size = 50

```
# Train the model
model.fit(X_train, y_train, epochs=100, shuffle=False, batch_size=50, verbose=1)
```

Active Listings Count LSTM: Model Performance

Mean Squared Error = 0.2258

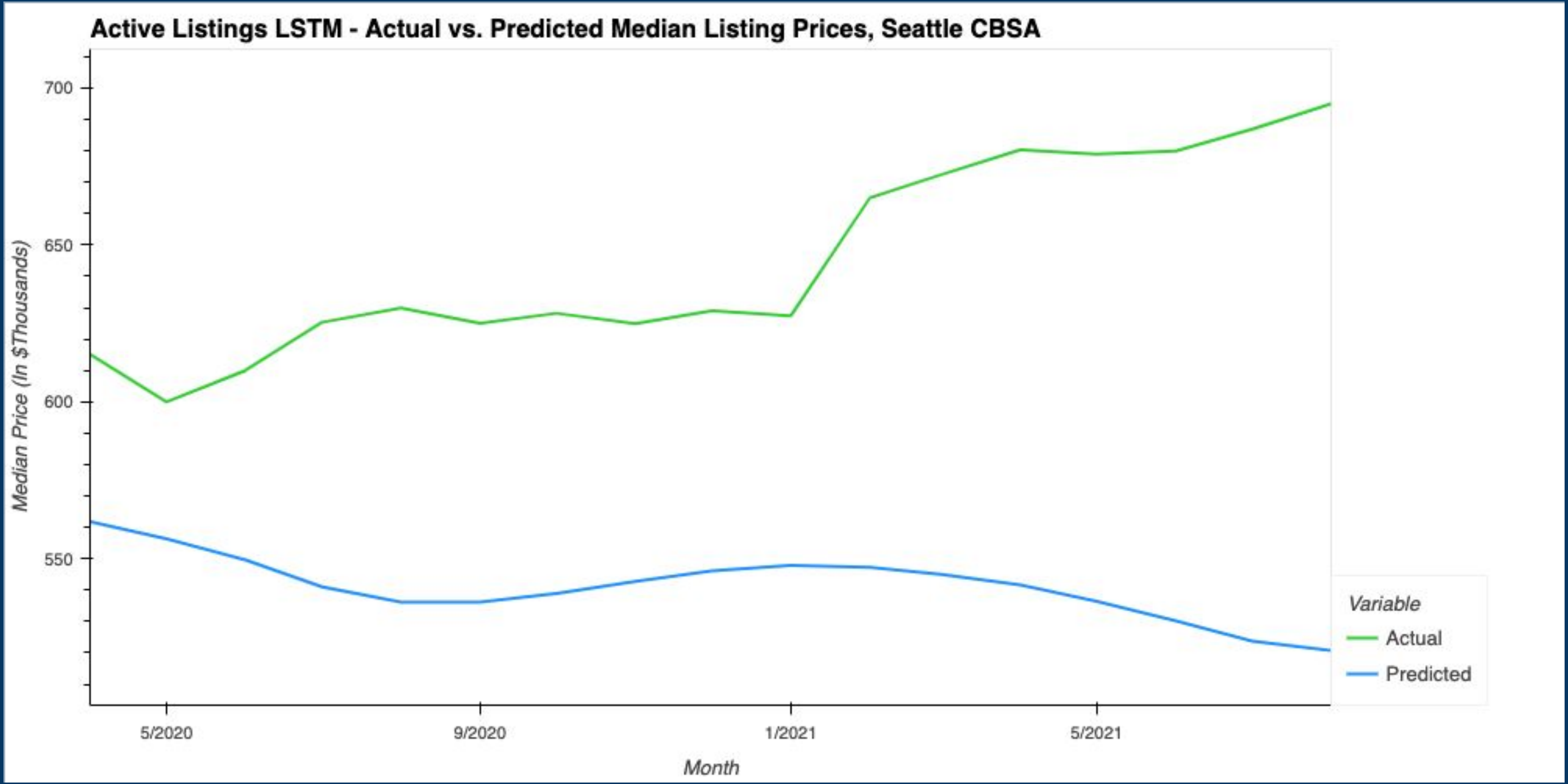
```
# Evaluate the model
model.evaluate(X_test, y_test)
```

```
1/1 [==============================] - 1s 1s/step - loss: 0.2258
0.225758895277977
```

Performance on Testing Data after re-scaling:

```
Active Listings Count LSTM Model Performance:

Active Listings Count LSTM Model RMSE: $110947.78125
Active Listings Count LSTM Model Mean Absolute Error: $104240.6875
Active Listings Count LSTM Model Mean Absolute Percentage Error: 15.916642189025879%
```

Active Listings Count LSTM: Model Performance



2) Time on Market LSTM: Set-up

- Target variable: “median_listing_price”
- Feature variable: “median_days_on_mkt”
- Used **MinMaxScaler** from sklearn.preprocessing to scale data
- Model(s) used:
  - **Sequential** from tensorflow.keras.models
  - **LSTM, Dense, Dropout** from tensorflow.keras.layers
- Model / Run parameters:
  - Lookback window = 7 months
  - Dropout_fraction = 0.2
  - Used 3 layers
  - Optimizer / loss = “adam”, “mean_squared_error”
  - epochs = 100
  - Batch_size = 30

```
# Train the model
model.fit(X_train, y_train, epochs=100, shuffle=False, batch_size=30, verbose=1)
```

Time on Market LSTM: Model Performance

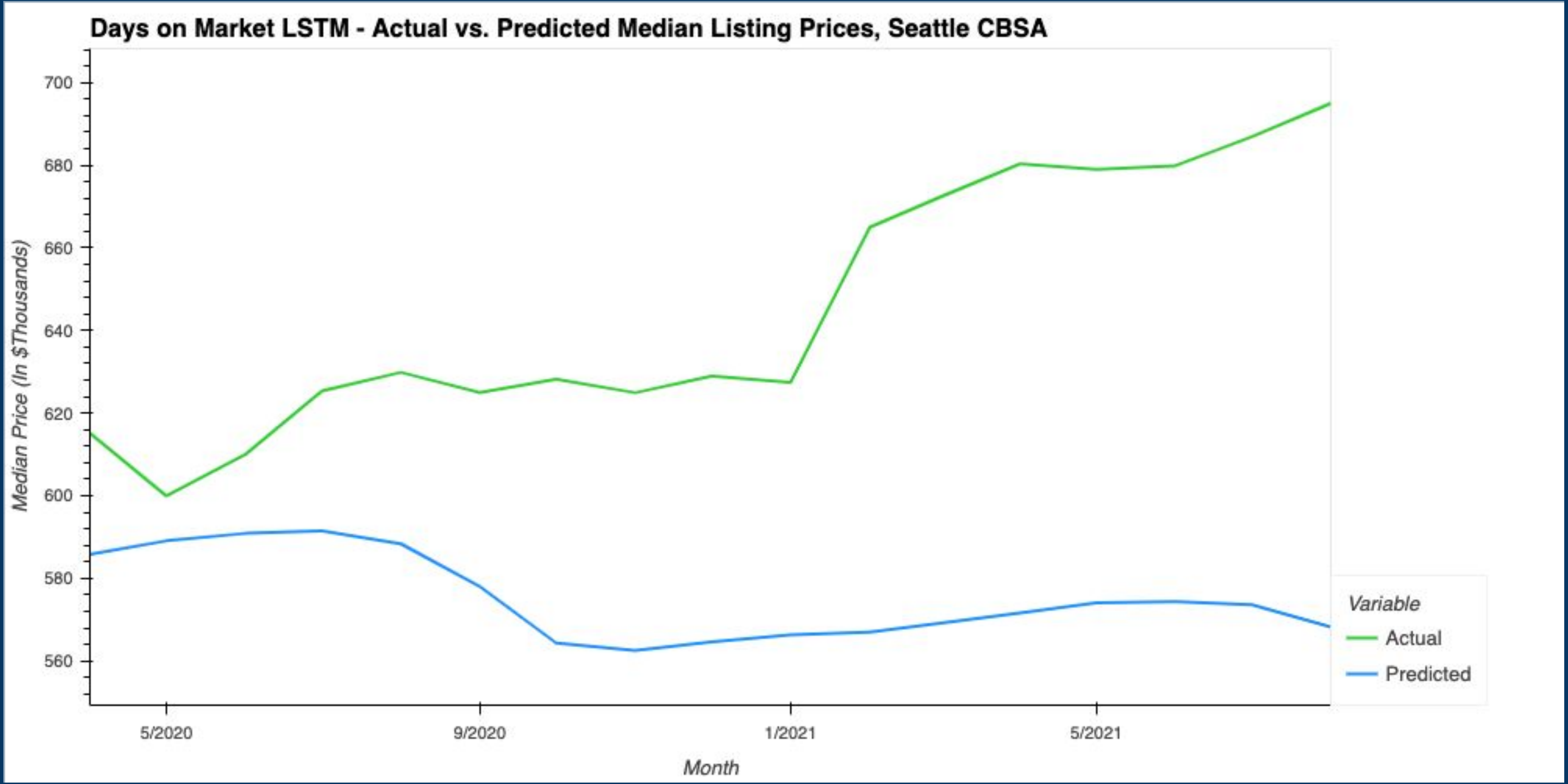
Mean Squared Error = 0.1137

```
# Evaluate the model
model.evaluate(X_test, y_test)

1/1 [==============================] - 2s 2s/step - loss: 0.1137
0.11367161571979523
```

Performance on Testing Data after re-scaling:

```
Time on Market LSTM Model Performance:

Time on Market LSTM Model RMSE: $78726.7265625
Time on Market LSTM Model Mean Absolute Error: $70213.609375
Time on Market LSTM Model Mean Absolute Percentage Error: 10.652839660644531%
```

Time on Market LSTM: Model Performance



3) Median Listing Price LSTM: Set-up

- Target variable: “median_listing_price”
- Feature variable: “median_listing_price”
- Used **MinMaxScaler** from sklearn.preprocessing to scale data
- Model(s) used:
  - **Sequential** from tensorflow.keras.models
  - **LSTM, Dense, Dropout** from tensorflow.keras.layers
- Model / Run parameters:
  - Lookback window = 9 months
  - Dropout_fraction = 0.2
  - Used 3 layers
  - Optimizer / loss = “adam”, “mean_squared_error”
  - epochs = 100
  - Batch_size = 50

```
# Train the model
model.fit(X_train, y_train, epochs=100, shuffle=False, batch_size=50, verbose=1)
```

Median Listing Price LSTM: Model Performance

Mean Squared Error = 0.0509

```
# Evaluate the model
model.evaluate(X_test, y_test)
```

```
1/1 [==============================] - 2s 2s/step - loss: 0.0509
0.05086769908666611
```

<u>Performance on Testing Data after re-scaling</u>:

```
Median Listing Price LSTM Model Performance:

Median Listing Price LSTM Model RMSE: $46799.2734375
Median Listing Price LSTM Model Mean Absolute Error: $41038.9921875
Median Listing Price LSTM Model Mean Absolute Percentage Error: 6.193641662597656%
```

Median Listing Price LSTM: Model Performance

LSTM Analysis: Conclusion

- LSTM model using median listing prices is best predictor for median listing prices, unsurprisingly
  - Lowest MSE (0.05)
  - Lowest RMSE (~$47k)
  - Lowest MAE (~$41k)
  - Lowest MAPE (~6%)
- Time on market LSTM is next best, active listings count LSTM comes in last
- Future considerations:
  - Find larger datasets of housing inventory statistics
  - Switch up target and feature variables:
    - Use median listing price and active listings count to predict median days on market