Predictive Model for Utah Housing Values

Josh Funderburk

Western Governors University

Data Analytics Graduate Capstone (D214)

Task 2: Data Analytics Report and Executive Summary

January 28, 2025

Predictive Model for Utah Housing Values

A. Research Question

The primary research question for this analysis is "Can property features such as square footage, lot size, and city predict Utah housing unit values?"

This research question is justified by several key factors. First, Utah's real estate market is experiencing significant growth, ranking as a top 5 population growth state (Williams, 2024). This rapid growth creates a critical need for accurate property valuation models. Second, while traditional valuation methods rely heavily on comparative market analysis, a data-driven approach using property features could provide more objective valuations (Woodman, 2024). Third, understanding the relationship between property features and values is essential for multiple stakeholders, including real estate professionals, investors, lenders, and homeowners, who need reliable methods to make informed decisions.

The context for this research question exists within Utah's dynamic real estate market. As housing demand increases with population growth, the relationship between property features and values becomes increasingly complex. Real estate professionals and consumers need tools to understand and predict property values accurately. While various property valuation methods exist, there is a specific need to understand how physical property features influence values in Utah's unique market conditions.

The following hypotheses will guide this analysis:

- Null Hypothesis (H0): Property features (square footage, lot size, city) do not significantly predict housing prices.
- Alternative Hypothesis (H1): Property features (square footage, lot size, city) significantly predict housing prices.

The alternative hypothesis appears more likely given existing research and market understanding. Previous studies, such as Ayyalasomayajula et al. (2021), have demonstrated that property features can effectively predict housing values using Random Forest Regression

models. Additionally, general real estate principles suggest that physical characteristics and location significantly influence housing unit values.

B. Data Collection

The primary dataset for this analysis is the Utah Housing Unit Inventory, obtained from the Utah AGRC's Open Data Portal. This comprehensive dataset contains 688,270 records of housing units across Utah's urban counties and includes the following variables for this analysis:

Field	Type
TOT_VALUE (Total Value)	Continuous
ACRES	Continuous
APX_BLT_YR (Approx. Year Built)	Continuous
BLT_DECADE (Decade Built)	Continuous
CITY	Categorical
COUNTY	Categorical
DUA (Density in Unit Dwellings per Acre)	Continuous
IS_OUG (Owned Unit Group)	Categorical
TOT_BD_FT2 (Total Building Square Footage)	Continuous
UNIT_COUNT	Continuous

The data collection methodology involved a direct download of a CSV file from the official Utah Geospatial Resource Center (UGRC) portal.

One significant advantage of this data collection approach is its reliability and standardization. The dataset comes from a government source (UGRC) and is maintained under a Creative Commons BY 4.0 license, ensuring data quality and consistent formatting

across all records. This standardization is particularly valuable for the analysis as it reduces the need for overly extensive data cleaning and validation.

However, a notable disadvantage of the methodology is the file size and format limitations. Working with a CSV file containing 688,270 rows presents significant computational challenges. The CSV format lacks built-in data type preservation and can lead to memory issues when loading large datasets, occasionally resulting in system crashes or performance issues during initial data loading and processing.

During the data collection process, the main challenge encountered was locating the data dictionary necessary for understanding variable definitions and contexts. This challenge was overcome through research of the UGRC documentation. The extra effort in finding the data dictionary proved valuable as it provided essential context for variables like 'IS_OUG' (Owned Unit Group) and 'DUA' (Density in Unit Dwellings per Acre).

The dataset has limitations that were important to consider during collection:

- Exclusion of specialized housing types (nursing homes, rehabilitation centers, assisted living facilities)
- Omission of temporary residences (extended stay hotels, student housing)
- Limited coverage of Accessory Dwelling Units (ADUs) outside Salt Lake County
- Absence of detailed interior specifications (number of bedrooms, bathrooms, stories)

To maintain analysis integrity, a delimitation to focus solely on single-family housing units was applied, excluding multi-family properties to avoid introducing ambiguity into the predictive model.

C. Data Extraction and Preparation

The data extraction and preparation process consisted of several steps to ensure data quality and appropriateness for analysis. The preparation process followed these steps:

1. **Initial Data Loading and Filtering**

   The raw dataset was loaded using pandas' read_csv function:

```
df = pd.read_csv(r'C:\Users\funde\Desktop\WGU\D214\HousingUnitInventory.csv', index_col=0, low_memory=False)
```

```
df.head()
```

OBJECTID	UNIT_ID	TYPE	SUBTYPE	IS_OUG	UNIT_COUNT	DUA	ACRES	TOT_BD_FT2	TOT_VALUE	APX_BLT_YR
1	0	single_family	single_family	1.0	11	1.282095	8.579707	18703.0	8678200.0	2000.0

   The first critical preparation step involved filtering the dataset to include only single-family homes. This filtering reduced the dataset to 627,126 records, all representing single-family residences.

```
# Drop non-single family rows
df = df[df['TYPE'] == 'single_family']
df['TYPE'].value_counts()
```

```
TYPE
single_family    627126
Name: count, dtype: int64
```

2. **Data Cleaning**

   Unnecessary columns were removed from the DataFrame:

```
# Drop specified columns
columns_to_drop = ['TYPE', 'UNIT_ID', 'SUBCOUNTY', 'EVAL_DATE', 'Shape__Area', 'Shape__Length']
df = df.drop(columns=columns_to_drop)
```

   All null values were then removed to ensure data integrity:

```
# Drop null values from the dataframe
df.dropna(inplace=True)
df.isnull().sum()
```

```
SUBTYPE       0
IS_OUG        0
UNIT_COUNT    0
DUA           0
ACRES         0
TOT_BD_FT2    0
TOT_VALUE     0
APX_BLT_YR    0
BLT_DECADE    0
CITY          0
COUNTY        0
dtype: int64
```

Records with invalid values (zeros) in key fields were removed:

```
# Drop columns where total value is equal to 0
df = df[df['TOT_VALUE'] != 0]

# Drop columns where acres is equal to 0
df = df[df['ACRES'] != 0]

# Drop columns where total building square footage is equal to 0
df = df[df['TOT_BD_FT2'] != 0]
```

3. **Summary Statistics & Visualizing the Data**

   The TOT_VALUE and each feature variable had summary statistics performed.

   Univariate and bivariate visualizations were produced to help visualize the data. These visualizations are available in the accompanying Jupyter Notebook included alongside the submission.

4. **Outlier Treatment**

   Statistical outliers were identified and removed using the interquartile range method:

```python
# Log transform the values and remove outliers
log_values = np.log(df['TOT_VALUE'])
Q1 = log_values.quantile(0.25)
Q3 = log_values.quantile(0.75)
IQR = Q3 - Q1

lower_bound = np.exp(Q1 - 3.0 * IQR)
upper_bound = np.exp(Q3 + 3.0 * IQR)

# Print removal analysis
print(f"\nRemoval Analysis:")
print(f"Lower bound: ${lower_bound:.2f}")
print(f"Upper bound: ${upper_bound:.2f}")

# Identify outliers
outliers_low = df[df['TOT_VALUE'] < lower_bound]
outliers_high = df[df['TOT_VALUE'] > upper_bound]
print(f"Records below lower bound: {len(outliers_low)} ({len(outliers_low)/len(df)*100:.2f}%)")
print(f"Records above upper bound: {len(outliers_high)} ({len(outliers_high)/len(df)*100:.2f}%)")

# Create cleaned dataset
df = df[(df['TOT_VALUE'] >= lower_bound) & (df['TOT_VALUE'] <= upper_bound)]
print(f"\nCleaned dataset size: {len(df)}")
```

The removal results demonstrated the following bounds:

```
Removal Analysis:
Lower bound: $101541.46
Upper bound: $2651183.17
Records below lower bound: 6829 (1.22%)
Records above upper bound: 2766 (0.50%)

Cleaned dataset size: 549101
```

5. **Feature Engineering**

Categorical variables were encoded using one-hot encoding:

```python
# List of columns to encode
columns_to_encode = ['CITY', 'COUNTY', 'SUBTYPE']

# Create dummies for specified columns
df_encoded = pd.get_dummies(df, columns=columns_to_encode, drop_first=False)
```

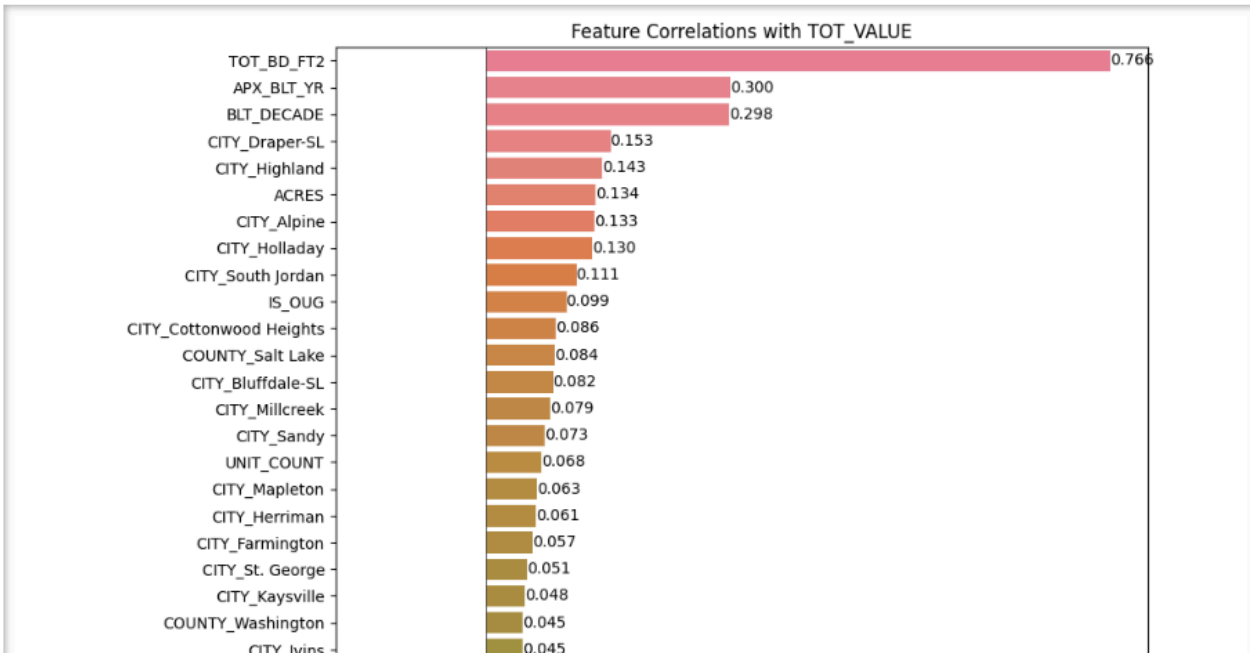
```python
df_encoded
```

CITY_American Fork	...	COUNTY_Morgan	COUNTY_Salt Lake	COUNTY_Tooele	COUNTY_Utah	COUNTY_Washington

6. Correlations

A correlation visualization was produced to help understand correlation between the feature variables and total value.

```
# Separate features and target
X = df_encoded.drop(['TOT_VALUE'], axis=1)
y = df_encoded['TOT_VALUE']

# Calculate correlations
correlations = pd.DataFrame({
    'Feature': X.columns,
    'Correlation': [X[col].corr(y) for col in X.columns]
}).sort_values('Correlation', ascending=False)

# Create a horizontal bar plot
plt.figure(figsize=(10, max(8, len(X.columns)/4)))
sns.barplot(data=correlations,
            x='Correlation',
            y='Feature',
            hue='Feature',
            legend=False)

# Add a vertical line at x=0 to make it easier to distinguish positive/negative correlations
plt.axvline(x=0, color='black', linestyle='-', linewidth=0.5)

# Customize the plot
plt.title('Feature Correlations with TOT_VALUE')
plt.xlabel('Correlation')

# Add value labels on the bars
for i, v in enumerate(correlations['Correlation']):
    plt.text(v, i, f'{v:.3f}', va='center')

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```



Note: For the entire visualization, please see the accompanying Jupyter Notebook.

7. **Scale the Data**

Numeric features were standardized using StandardScaler:

```
# Initialize scaler
scaler = StandardScaler()

# Scale all features
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
```

The tools and techniques used in this process were selected based on their effectiveness and reliability in handling large-scale housing data. Python was chosen as the primary programming language due to its faster and more efficient performance compared to R and its more comprehensive set of libraries (Saha, 2024b). Additionally, Python's intuitive syntax and public availability made it preferable to SAS, which has a steeper learning curve and requires paid licensing (Ochoa, 2024). The primary tools within the Python ecosystem used for data extraction and preparation included:

- Python's pandas library for data manipulation
- NumPy for numerical operations
- Scikit-learn for data preprocessing and standardization

The primary advantage of these tools is their robust handling of large datasets and built-in functions for data transformation. Pandas excels at handling mixed data types and provides efficient methods for data filtering and cleaning. The StandardScaler from scikit-learn ensures all numeric features are on the same scale, which is crucial for the subsequent machine learning analysis.

The main disadvantage of these tools emerge in the memory requirements for large datasets, particularly during one-hot encoding of categorical variables. The creation of multiple binary columns for each categorical variable significantly increases the dataset's dimensionality.

However, this trade-off was acceptable given the importance of proper categorical variable handling for machine learning.

Each step in the preparation process was documented with appropriate logging and verification to ensure data integrity was maintained throughout the transformations. The final prepared dataset contains 549,101 records with encoded categorical variables and standardized numeric features.

D. Analysis

The analysis phase employed Random Forest Regression (RFR) via Scikit-learn's Python library to predict housing unit values based on the prepared features. The analysis process was structured in three distinct phases: initial model development, hyperparameter tuning, and final model evaluation. This approach enabled thorough testing and optimization of the predictive model.

Analysis Phases

1. Initial Model Development

    During the initial model development phase, the prepared dataset was split into training and testing sets using Scikit-learn's train_test_split feature, with 80% (533,269 records) allocated for training and 20% (198,017 records) reserved for testing.

```python
# Set seed for reproducibility
SEED = 1

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    train_size=0.8,
    test_size=0.2,
    random_state=SEED
)

# Recombine features and target for each split into complete datasets
mod_train = pd.concat([X_train, y_train], axis=1)
mod_test = pd.concat([X_test, y_test], axis=1)

# Verify split sizes
print("Training set shape:", mod_train.shape)
print("Test set shape:", mod_test.shape)
```

```
Training set shape: (533269, 121)
Test set shape: (198017, 121)
```

The initial Random Forest model demonstrated strong predictive capability, demonstrating an R-squared score of 0.842. Additional performance metrics included a Mean Squared Error of 10,186,082,188.194, a Root Mean Squared Error of $100,926.122, and a Mean Absolute Error of $55,948.001. These results indicated that the model could explain approximately 84.2% of the variance in housing values.

```python
rfr = RandomForestRegressor(random_state=42, n_jobs=-1,)
rfr.fit(X_train, y_train)

# Make predictions
y_pred = rfr.predict(X_test)

# Calculate R-squared
r2 = r2_score(y_test, y_pred)
print(f"R-squared Score: {r2:.3f}\n")

# Calculate MSE
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.3f}")

# Calculate RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"Root Mean Squared Error: {rmse:.3f}\n")

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error: {mae:.3f}")
```

```
R-squared Score: 0.842

Mean Squared Error: 10186082188.194
Root Mean Squared Error: 100926.122

Mean Absolute Error: 55948.001
```

2. Hyperparameter Tuning

Following the initial modeling, comprehensive hyperparameter tuning was conducted using grid search cross-validation. The grid search explored various combinations of key parameters, including the number of trees (n_estimators: 100, 200, 300), maximum tree depth (max_depth: 10, 20, 30), minimum samples required for node splitting (min_samples_split: 2, 5, 10), and minimum samples per leaf node (min_samples_leaf: 1, 2, 3) (Nolan, 2023). This systematic optimization process identified the optimal parameter configuration: 300 trees, a maximum depth of 30, a minimum split threshold of 10 samples, and a minimum leaf size of 1 sample. Due to issues with computation, only 5% (21,964 samples) of the training data set were used for hyper parameter tuning. This is a significant limitation of this analysis as lack of computational memory and processing power eliminated the ability to tune parameters on the entire testing set. However, several tests did show consistency in the hyper parameters, so this analysis can be confident in trusting the results of the tuning.

```python
# Combine X_train and y_train into a single DataFrame to preserve index alignment
X_train.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
train_data = pd.concat([X_train, y_train], axis=1)

# Sample a fraction of the combined training data
sample_fraction = 0.05
sampled_data = train_data.sample(frac=sample_fraction, random_state=SEED)

# Separate the sampled features and target
X_train_sampled = sampled_data.drop(columns=y_train.name)
y_train_sampled = sampled_data[y_train.name]

# Verify the sampled sizes
print(f"Sampled training set shape: {X_train_sampled.shape}")

# Define parameters for hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 3]
}

# Set up grid search with 3-fold cross validation and r2 as the scoring metric
grid = GridSearchCV(
    rfr,
    param_grid,
    cv=2,
    scoring='r2',
    n_jobs=-1
)

# Perform the actual grid search
grid.fit(X_train_sampled, y_train_sampled)

# Print best parameters and score
print("\nBest parameters:", grid.best_params_)
print("Best Mean Absolute Error:", grid.best_score_)
```

```
Sampled training set shape: (21964, 120)

Best parameters: {'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 300}
Best Mean Absolute Error: 0.7989197157333751
```

3. Final Model

The final model implemented with these optimized parameters achieved slightly different performance metrics. The R-squared score adjusted to 0.805, with a Mean Squared Error of 12,540,528,306.686, a Root Mean Squared Error of $111,984.500, and a Mean Absolute Error of $62,548.030. While these metrics show a slight decrease in

performance compared to the initial model, they represent a more robust and generalizable solution, less likely to overfit the training data (Wikipedia, 2025).

```python
# Get the best model from grid search
best_rf = grid.best_estimator_

# Make predictions
y_pred = best_rf.predict(X_test)

# Calculate R-squared
r2 = r2_score(y_test, y_pred)
print(f"R-squared Score: {r2:.3f}\n")

# Calculate MSE
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.3f}")

# Calculate RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"Root Mean Squared Error: {rmse:.3f}\n")

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error: {mae:.3f}")
```

```
R-squared Score: 0.805

Mean Squared Error: 12540528306.686
Root Mean Squared Error: 111984.500

Mean Absolute Error: 62548.030
```

The selection of Random Forest Regression as the primary analytical technique was justified by several key factors. In fact, the advantages of the technique directly play into the reasons that the technique was justified. A crucial advantage of Random Forest Regression over other machine learning algorithms like Multiple Linear Regression is its ability to handle a large number of one-hot encoded categorical variables without issues of multicollinearity, which was essential given the dataset's numerous city and county features (*Won't highly-correlated variables in random forest distort accuracy and feature-selection?*, 2024). Additionally, the

algorithm's ability to capture non-linear relationships and handle complex interactions between variables made it particularly suitable for housing unit valuation, where total values often depend on intricate combinations of features. Random Forest's built-in feature importance assessment capabilities also provided valuable insights into the key drivers of housing values, enhancing the interpretability of results. Furthermore, the algorithm's ensemble nature provided protection against overfitting, a crucial consideration given the large number of features in the dataset (Lea, 2024).

However, Random Forest Regression also has several limitations as an analytical technique. Unlike linear regression, which provides clear coefficients showing the direct impact of each variable, Random Forest models make it difficult to understand exactly how predictions are made (Sajja, 2024). The technique does not provide confidence intervals for its predictions, limiting the ability to assess prediction uncertainty. Additionally, Random Forest models can be prone to overfitting when the number of trees is too large or the tree depth is too deep, requiring careful tuning of these parameters (Sajja, 2024). The method also tends to extrapolate poorly beyond the range of training data, as it can only make predictions within the bounds of values it has seen during training (*Random forest regression not predicting higher than training data*, 2020).

Despite these limitations, the analysis successfully demonstrated the capability to predict housing unit values using property features, providing a foundation for data-driven real estate valuation in Utah's market.

E. Data Summary and Implications

The analysis results support the alternative hypothesis that property features can significantly predict Utah housing unit values. The final Random Forest model achieved an R-squared value of 0.805, indicating that approximately 80.5% of the variance in housing values can be explained by the selected property features. However, with a Mean Absolute Error of

$62,548.03 on homes with a median value of approximately $509,000, the model's practical application is limited, as the error margin is roughly 12% of a typical property's value. This suggests that while the model demonstrates the predictive power of property features, there is significant room for improvement in prediction accuracy.

In the context of the original research question, the model's performance validates that square footage, lot size, and city location are meaningful predictors of housing values in Utah. The Random Forest model's feature importance analysis revealed that total building square footage and lot size (acres) were among the strongest predictors, while city locations showed varying levels of correlation depending on the city. This aligns with traditional real estate valuation principles while providing quantitative evidence of these relationships.

A significant limitation of this analysis is the absence of key predictor variables in the dataset. Critical features such as the number of bedrooms, bathrooms, garage capacity, and interior finish quality were not available in the UGRC data set. These features are traditionally strong indicators of home value in real estate appraisals. The inclusion of these variables would likely improve the model's prediction accuracy and reduce the current error margin, making it more practical for real-world applications.

Based on these results, the recommended course of action is to use the Random Forest model as a baseline for property valuations, while working to acquire more comprehensive property data. The model's current accuracy level suggests it should be used cautiously and only alongside traditional appraisal methods.

For future research,there are two promising directions:

1. Enhanced Feature Collection: Develop a more comprehensive dataset that includes key property characteristics such as number of bedrooms, bathrooms,

garage capacity, interior finish quality, and recent renovation status. This would address the current model's primary limitation and likely improve its predictive accuracy substantially.

2. Single-Unit Focus Study: Refine the analysis by focusing exclusively on single-family, single-unit properties to reduce variability in the dataset. By eliminating multi-unit properties, model performance could potentially improve even with the limitations of the current feature set. This could provide a stronger foundation before expanding into more complex market segments.

These research directions directly address the identified limitations while remaining practically achievable with appropriate data collection efforts.

In conclusion, this analysis has demonstrated both the potential and limitations of using property features to predict housing unit values in Utah's dynamic real estate market. While the Random Forest model's ability to explain 80.5% of housing value variance validates the relationship between physical properties and market values, the $62,548 mean absolute error highlights the complexity of real estate valuation. As Utah's housing market continues to evolve with its rapid population growth, the development of more intelligent predictive models incorporating additional property features could provide valuable tools for real estate owners, professionals, and investors. The path forward is to combine traditional real estate knowledge with advanced data analytics, ultimately working toward more accurate and reliable housing unit valuation methods.

F. References

Ayyalasomayajula, M., Bussa, S., & Ayyalasomayajula, S. (2021). Harnessing the Power of Big Data: The Evolution of AI and Machine Learning in Modern Times. ESP Journal of Engineering & Technology Advancements, 1(1), 125–133. https://www.espjeta.org/Volume1-Issue1/JETA-V1I1P114.pdf

Hummel, A. (2024, March 5). *Single-family vs. multi-family residences - decoding two types*. Real Estate Crunch. https://real-estate-crunch.com/single-family-vs-multi-family-residences-decoding-two-types

Lea, S. (2024, September 3). Beginner's Guide to Predicting House prices with Random Forest: Step-by-step introduction with A... Medium. Retrieved January 27, 2025, from https://medium.com/%40schuerch_sarah/beginners-guide-to-predicting-house-prices-with-random-forest-step-by-step-introduction-with-a-aee81daae3ee

Nolan, R. (2023, October 24). Random Forest Regressor in Python: A Step-by-Step Guide. YouTube. https://www.youtube.com/watch?v=YUsx5ZNlYWc

Ochoa, D. (2024, May 18). SAS vs python: A comparison for Data Science in 2024. Halfnine. https://www.halfnine.com/blog/post/sas-vs-python

Peng, X., & Lai, Y. (2022). Improving Data Cleaning Processes for Statistical Modeling. International Journal of Statistical Research, 28(3), 201–215. https://doi.org/10.1155/ijrstat/2022

Random forest regression not predicting higher than training data. Cross Validated. (2020, January 17). Retrieved January 27, 2025, fromhttps://stats.stackexchange.com/questions/235189/random-forest-regression-not-predicting-higher-than-training-data

Saha, M., Saha, S., Majumder, S. (2024a). House Price Prediction Using Random Forest Regression Considering the Nearest In-Demand Location. In: Dutta, S., Bhattacharya, A., Shahnaz, C., Chakrabarti, S. (eds) Cyber Intelligence and Information Retrieval. CIIR 2023. Lecture Notes in Networks and Systems, vol 1025. Springer, Singapore. https://doi.org/10.1007/978-981-97-3594-5_9

Saha, R. (2024b, January 27). Python vs. R for Predictive modeling: A comparative analysis. Analytics Insight. https://www.analyticsinsight.net/latest-news/python-vs-r-for-predictive-modeling-a-comparative-analysis

Sajja, R. K. (2024, August 20). Comparing linear regression and random forest regression using python. Medium. https://python.plainenglish.io/comparing-linear-regression-and-random-forest-regression-using-python-23cc1b8c5795

UGRC. (2025). *Utah Housing Unit Inventory*. Utah's State Geographic Information Database (AGRC). (2025, January 7). Retrieved January 19, 2025, from https://opendata.gis.utah.gov/datasets/utah::utah-housing-unit-inventory/

Wikimedia Foundation. (2025, January 27). Bias–variance tradeoff. Wikipedia. https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff

Williams, C. (2024, December 19). Utah returns to the top 5 in growth as US population soared in 2024. KSL.com. https://www.ksl.com/article/51214522/utah-returns-to-the-top-5-in-growth-as-us-population-soared-in-2024

Won't highly-correlated variables in random forest distort accuracy and feature-selection? Cross Validated. (2018, August 3). Retrieved January 26, 2025, from https://stats.stackexchange.com/questions/141619/wont-highly-correlated-variables-in-random-forest-distort-accuracy-and-feature

Woodman, C. (2024, December 2). AVM in real estate: Automated Valuation Models explained. New Silver. Retrieved January 26, 2025, from https://newsilver.com/the-lender/avm-in-real-estate