Predictive Housing Price Model for Washington State

Group 3
Marriott, Bobby
Martinez, Patricio
Widjaja, Victoria (Tori)

1. Data Importing

Dataset description:

- 21 columns
- 21613 rows

Price Details:

```
count      21613.00
mean      540088.14
std       367127.20
min        75000.00
25%       321950.00
50%       450000.00
75%       645000.00
max      7700000.00
Name: price, dtype: object
```

Data Types:

```
id                 int64
date              object
price            float64
bedrooms         float64
bathrooms        float64
sqft_living      float64
sqft_lot         float64
floors           float64
waterfront         int64
view               int64
condition          int64
grade              int64
sqft_above         int64
sqft_basement      int64
yr_built           int64
yr_renovated       int64
zipcode            int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
dtype: object
```

Null Data Counts:

```
id                  0
date                0
price               0
bedrooms         1134
bathrooms        1068
sqft_living      1110
sqft_lot         1044
floors              0
waterfront          0
view                0
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated        0
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
dtype: int64
```

1. Cleaning and Wrangling
Standardizing the date format

```python
### working copy ###
date_time_cleaned = df_orig.copy()

### Fixing date format###
date_time_cleaned['date'] = pd.to_datetime(df_orig['date'], format='%Y%m%dT%H%M%S', errors='coerce')

### Save the updated DataFrame to a new CSV file ###
date_time_cleaned.to_csv(os.path.join(log_prefix, "house_file_v2.csv"), index=False)

### sample of the cleaned data ###
display(HTML("<u>Sample of Cleaned Data:</u>"))
print(date_time_cleaned.head())
```

```
           id       date     price  bedrooms  bathrooms  sqft_living  \
0  7129300520 2014-10-13 221900.00      3.00       1.00      1180.00
1  6414100192 2014-12-09 538000.00      3.00       2.25      2570.00
2  5631500400 2015-02-25 180000.00      2.00       1.00       770.00
3  2487200875 2014-12-09 604000.00      4.00       3.00      1960.00
4  1954400510 2015-02-18 510000.00      3.00       2.00      1680.00
```

1. Cleaning and Wrangling
Removing Bedrooms and Bathrooms with a value of 0

```
### Filtering and saving the data ###
date_time_cleaned = date_time_cleaned[(date_time_cleaned['bedrooms'] != 0) & (date_time_cleaned['bathrooms'] != 0)]

date_time_cleaned.to_csv(os.path.join(log_prefix, 'filtered_date_time_cleaned.csv'), index=False)
```

```
### Checking progress ###
date_time_cleaned.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built
0	7129300520	2014-10-13	221900.00	3.00	1.00	1180.00	5650.00	1.00	0	0	...	7	1180	0	1955
1	6414100192	2014-12-09	538000.00	3.00	2.25	2570.00	7242.00	2.00	0	0	...	7	2170	400	1951
2	5631500400	2015-02-25	180000.00	2.00	1.00	770.00	10000.00	1.00	0	0	...	6	770	0	1933

1. Cleaning and Wrangling
Removing the bottom and top 1% of data

```
### Removing the top and bottom 1% of data in the price column ###
working_data = pd.read_csv(os.path.join(log_prefix, 'filtered_date_time_cleaned.csv'))

### Establishing lower and upper bound using the bottom and upper 1% ##
upper_bound = working_data['price'].quantile(0.99)
lower_bound = working_data['price'].quantile(0.01)

display(HTML("<u>Defining Bounds to Remove Outliers:</u>"))
print("Upper bound =",round(upper_bound,2))
print("\nLower bound =",round(lower_bound,2))

### Only extracting the data in between the values above the lower bound and values lower than the upper bound ###
working_data2 = working_data[(working_data['price'] >= lower_bound) & (working_data['price'] <= upper_bound)]
```

Defining Bounds to Remove Outliers:

```
Upper bound = 1965006.6

Lower bound = 154000.0
```

1. Cleaning and Wrangling Imputing Part One

```
### Replace NaN values in 'bathrooms' with values from 'avg_bathrooms' ###
df_clean['bathrooms'] = df_clean['bathrooms'].fillna(df_clean['avg_bathrooms'])
```



```
### Replace NaN values in 'bathrooms' with values from 'avg_bathrooms' ###
df_clean['bathrooms'] = df_clean['bathrooms'].fillna(df_clean['avg_bathrooms'])
```



```
### Replace NaN values in 'SQFT' with values from 'avg_sqft_living_bin' ###
df_clean['sqft_living'] = df_clean['sqft_living'].fillna(df_clean['avg_sqft_living'])
```

```
### Replace NaN values in 'sqft_lot' with values from 'avg_sqft_lot' ###
df_clean['sqft_lot'] = df_clean['sqft_lot'].fillna(df_clean['avg_sqft_lot'])
```

1. Cleaning and Wrangling
Imputing Part Two

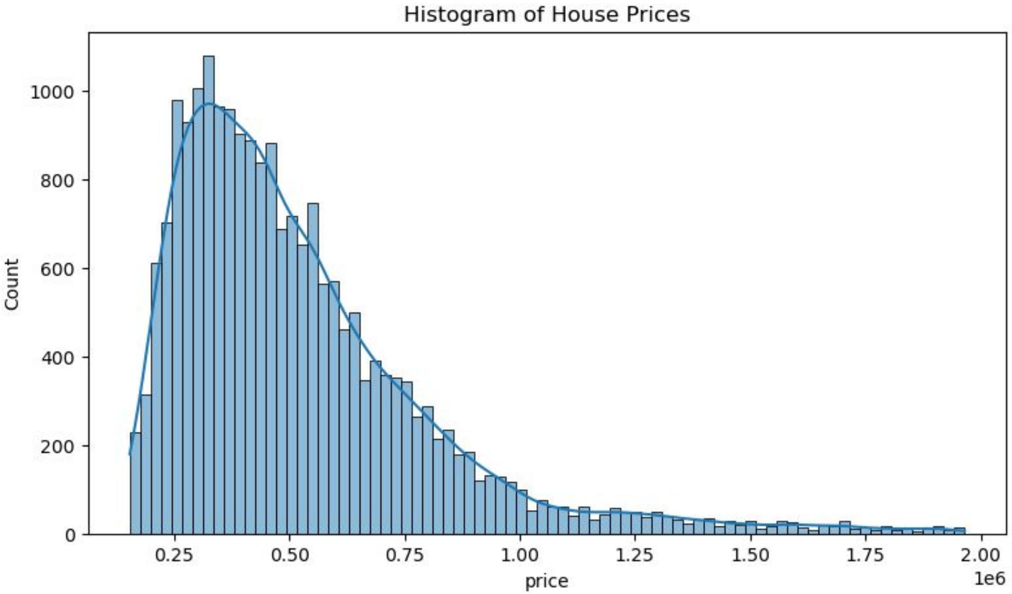
```python
### Check all Null values have been handled ###
null_data = df_clean.isnull().sum()
display(HTML("<u>Null Data Counts:</u>"))
print(null_data)
```

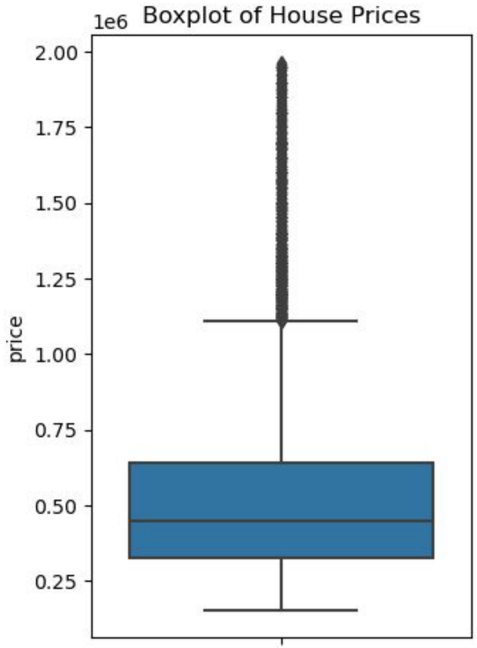
Null Data Counts:

```
id                  0
date                0
price               0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront          0
view                0
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated        0
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
price_group         0
avg_bedrooms        0
avg_bathrooms       0
avg_sqft_living     0
avg_sqft_lot        0
dtype: int64
```

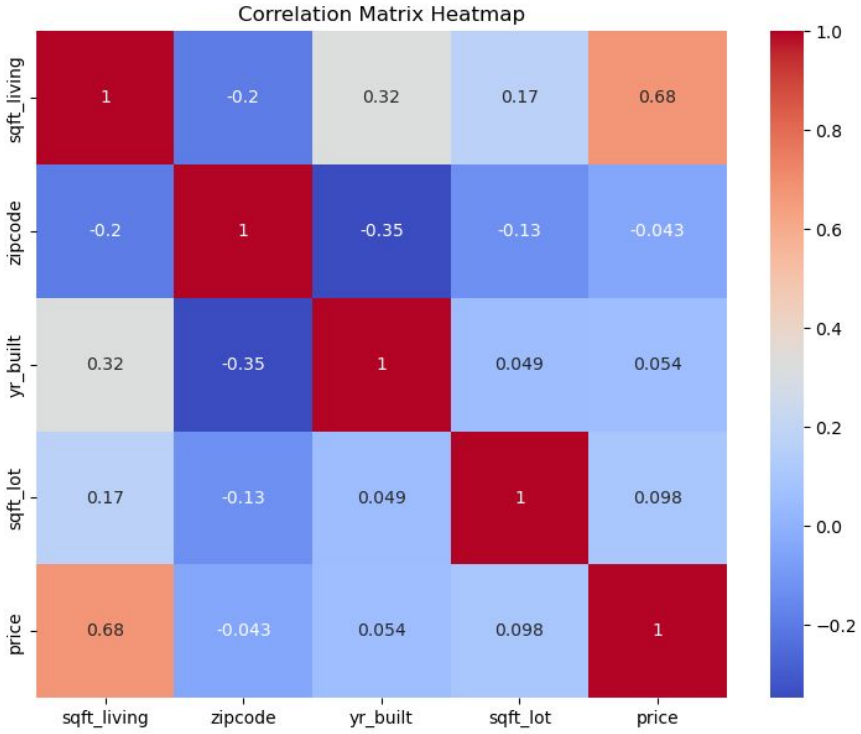
2. Data Analysis and Visualization

Our goal is to identify key housing features that significantly impact prices in Washington State.

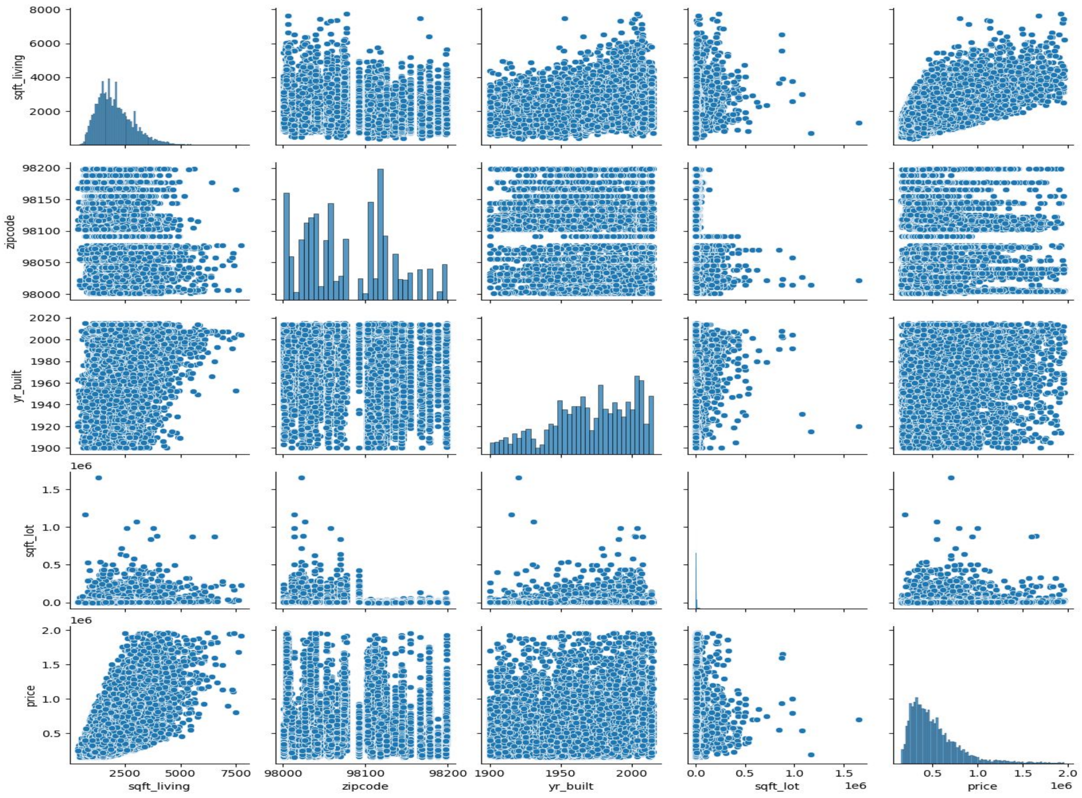




2. Data Analysis and Visualization



2. Data Analysis and Visualization



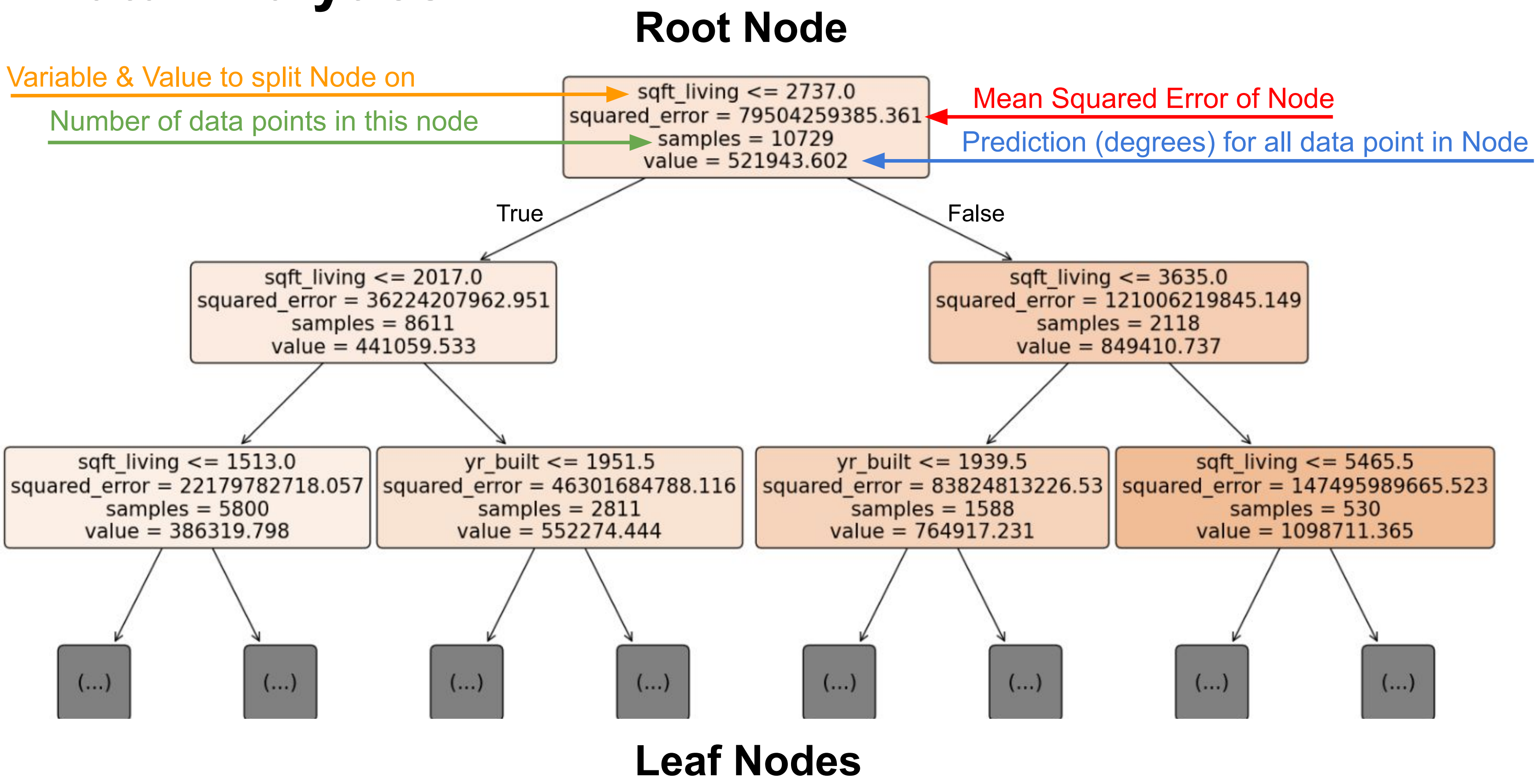
3. Data Analytics

- Final Algorithm Type: RandomForestRegressor from sklearn.ensemble

- Dependent Variable: price

- Independent Variables: sqft_living, zipcode, yr_built, sqft_lot

3. Data Analytics



3. Data Analytics

- Mean Absolute Error: $85,425.55
- Root Mean Squared Error (RMSE): $135,257.50
- Feature Importances:
  - 'sqft_living' 54.71%
  - 'zipcode' 24.54%
  - 'yr_built' 10.64%
  - 'sqft_lot' 10.11%
- R-squared (R2) Value: 0.7797
- Explained Variance Score: 0.7798

References

Bruce, P., Bruce, A., & Gedeck, P. (2020). Practical Statistics for Data Scientists (Second Edition ed.). O'Reilly Media, Inc.

Koehrsen, W. (2017). Random Forest in Python. Retrieved December 6, 2023 from https://towardsdatascience.com/random-forest-in-python-24d0893d51c0