Machine Learning:
Real Estate Prediction Model

Joe Awanis
Manitha Ayaninilkkunnathil
Jean Paul Nduwayo Ntore
Cory Peacock

https://github.com/josephawanis30/MachineLearningRealEstatePredictions

Overview

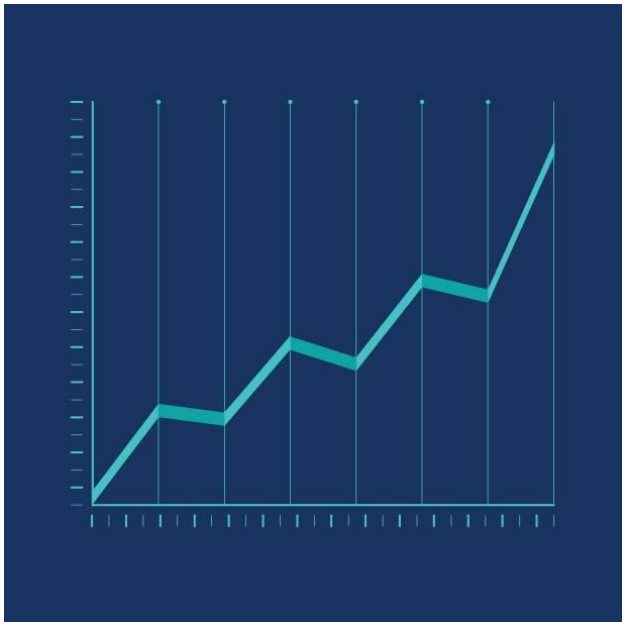
According to the May 6, 2022, Cromford Report, the housing market in Central Phoenix is just beginning to pick up again after a very slow year for Real Estate sales. Despite the slow year, the median sale price currently sits at $425,000, up 27.6% from May 5, 2021, with a monthly volume of 9,500 sales. This very active market warrants investigation.

Overview

There are, of course, numerous features that factor into the listing price for any realty listing. On one end of the spectrum, **large data-driven companies like Zillow are able to utilize their algorithms to price homes very accurately**. According to Zillow's website,2 their nationwide median error rate for an on-market home "Zestimate" is 1.9%. On the other end of the spectrum, **independent realtors utilize a mix of experiential intuition and "comps"** - hyper local comparable house listings. According to Brian Houle, a local Phoenix Realtor, finding six to ten homes within a two-mile radius with a similar square footage and similar housing condition / quality is a good starting place for finding accurate pricing. What independent Realtors have internalized as intuition, data-driven Realtors have made explicit in their algorithms, the features of which are typically not publicly available.

Overview

The primary purpose of this analysis is to create a machine learning model that can predict housing prices for single-family detached homes in the central Phoenix area utilizing features that are publicly available.



Data Sourcing

- Housing data sourced from the FlexMLS Realty portal
- Search conducted on May 6 and 7, 2022
  - Houses currently for sale or sold within the 14 days prior
  - Within a radius of 10 miles from Indian Steel Park in Central Phoenix
  - With an asking price of $1,300,000 or less
- More than 2,300 houses for sale or recently sold

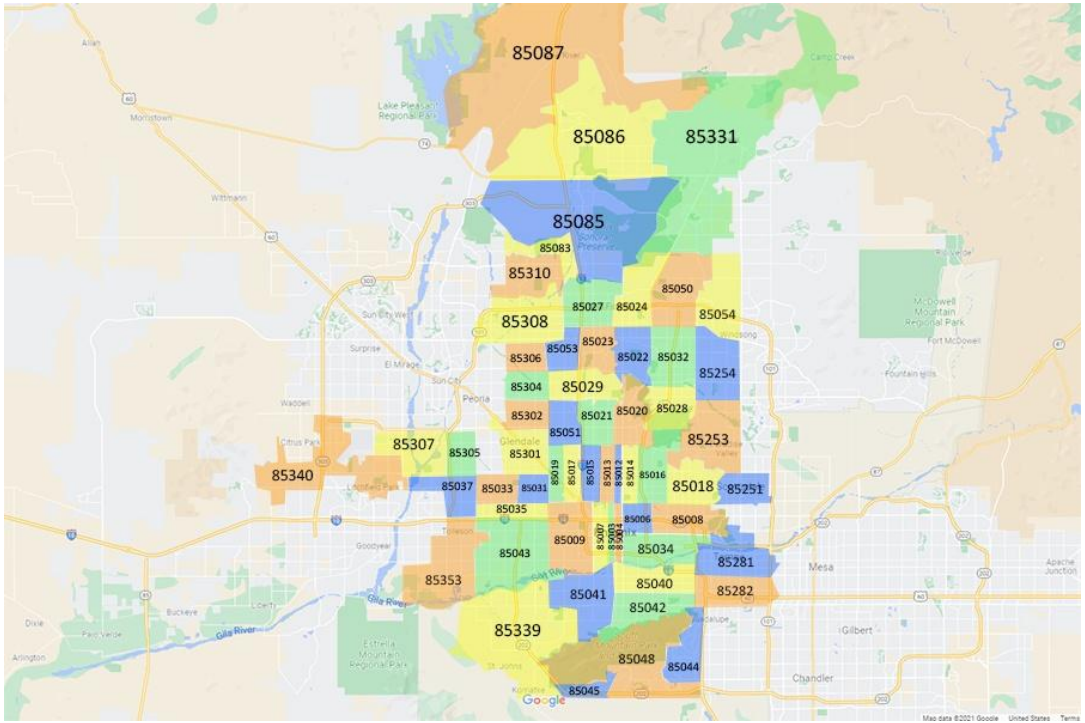
- The average sale price percentage compared to listed sale price is 101.83%.
- Little need to differentiate between houses listed for sale and those sold.

Data Sourcing

The features of each data point include:

- ID
- Price (Asking Price / Sale Price)
- Address
- Zip
- Status (Active, Under Contract, etc.)
- Subdivision
- Year built
- Bedrooms
- Bathrooms
- Approximate Square Footage

Data Sourcing



Additional data was retrieved from the Arizona Department of Education in order to provide an education score for each house listed in the dataset above. For each zip code in the dataset, the best scoring elementary school in the zip code's school district was attached.

Data Sourcing

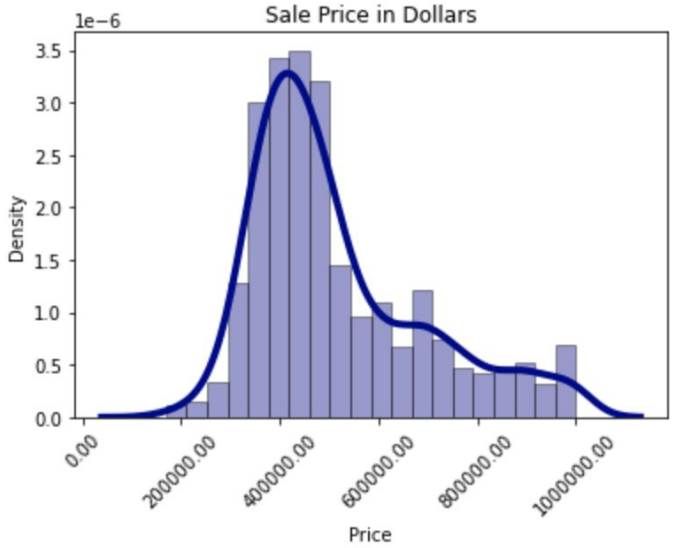
A crime score was also added by retrieving data from the crime statistics open data section of Phoenix.gov. Individual crime reports were grouped by zip code and the sum of crimes by zip code was divided by all crime reports in the dataset in order to give a crime score for each zip code. Where a zip code was not represented in the city of Phoenix crime dataset, due to the jurisdiction of another city, the median crime score was used.

Data Exploration

Initial data exploration included:

A density plot for price:

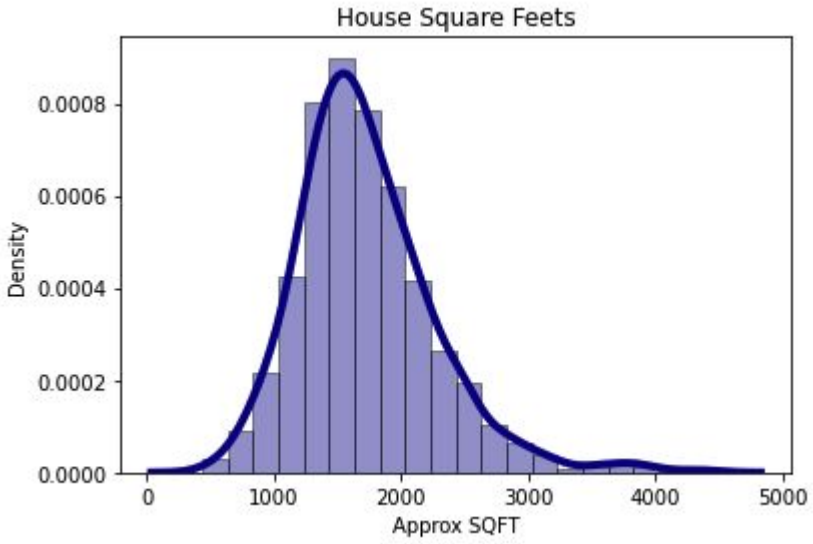
Min:	169000.000
Q1:	390000.000
Median:	462320.000
Q3:	600000.000
Max:	1000000.000
Mean:	515795.279
Mode:	450000.000



Data Exploration

A density plot for square footage:

Min:	442.000
Q1:	1372.000
Median:	1651.000
Q3:	2002.000
Max:	4423.000
Mean:	1726.404
Mode:	1260.000



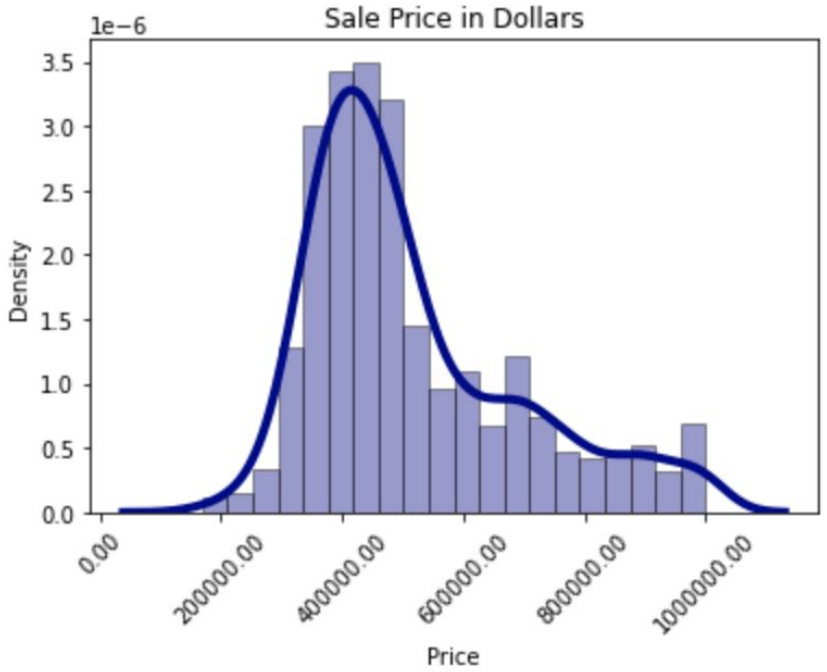
Data Exploration

A correlation matrix of numeric features:

Data Exploration

A density plot for bedrooms:

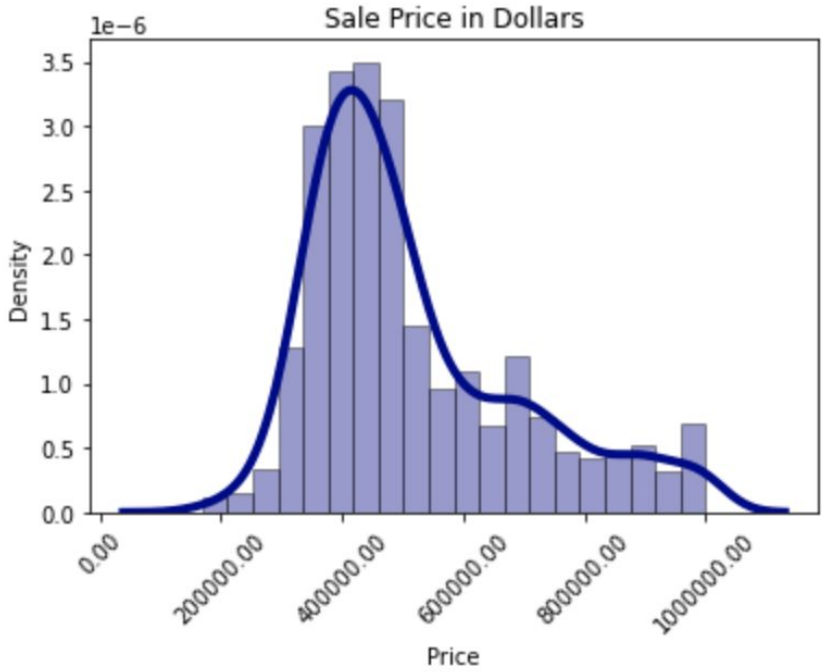
Min:	0.000
Q1:	3.000
Median:	3.000
Q3:	4.000
Max:	12.000
Mean:	3.381
Mode:	3.000



Data Exploration

A density plot for bathrooms:

Min:	0.000
Q1:	2.000
Median:	2.000
Q3:	2.000
Max:	6.000
Mean:	2.110
Mode:	2.000



Data Exploration

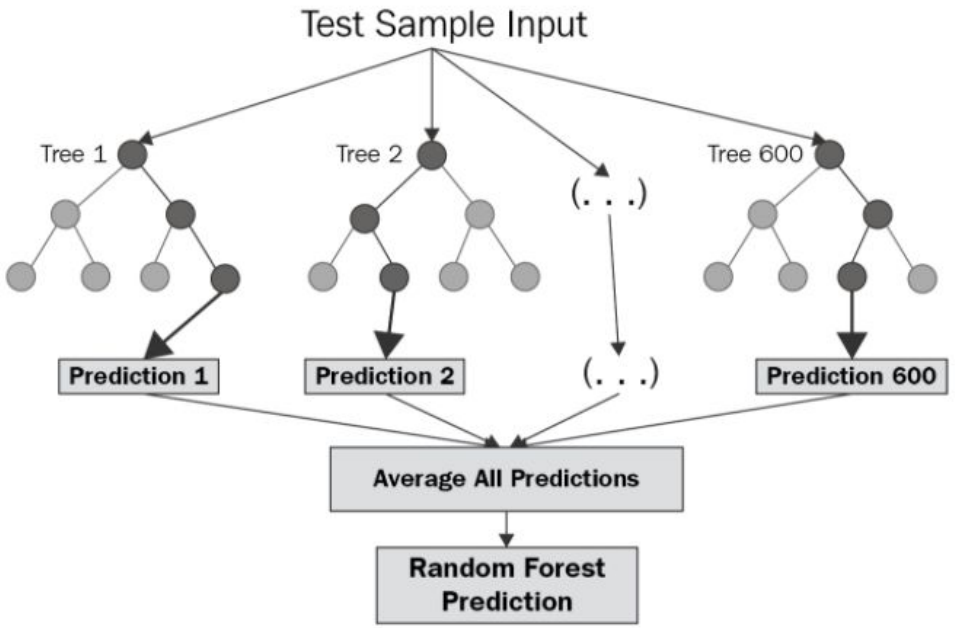
Additionally, we explored multiple linear regression using R, and a Neural Network using Python's TensorFlow in a Jupyter Notebook. Neither model suggested itself as helpful to this project.

```r
## statistical tests
ggplot(prop_filter, aes(x=Price)) + geom_density() # visualize distribution using density plot
shapiro.test(prop_filter$Price) # pvalue much less than 0.05, so NOT normal distribution
# note: strong right skew

# anova - considering all features are actually categorical even though they're numbers (sqft is exception)
summary(aov(Price ~ zip, data=prop_filter))
summary(aov(Price ~ zip + sqft, data = prop_filter))
summary(aov(Price ~ zip + sqft + year_built + Bedrooms + Bathrooms, data = prop_filter))

# multiple linear regression
# using only numerical values - that is, not year_built or zip
summary(lm(Price ~ sqft + Bedrooms + Bathrooms, data = prop_filter))
summary(lm(Price ~ sqft + Bedrooms + Bathrooms + age, data = prop_filter))
summary(lm(Price ~ sqft + Bedrooms + Bathrooms + age + ppsf, data = prop_filter)) # is ppsf redundant?

summary(lm(Price ~ sqft + Bedrooms + Bathrooms + age + zip, data = prop_filter))
```

```python
# Create a method that creates a new Sequential model with hyperparameter options
def create_model(hp):
    nn_model = tf.keras.models.Sequential()

    # Allow kerastuner to decide which activation function to use in hidden layers
    activation = hp.Choice('activation',['relu','tanh'])

    # Allow kerastuner to decide number of neurons in first layer
    nn_model.add(tf.keras.layers.Dense(units=hp.Int('first_units',
        min_value=1,
        max_value=30,
        step=5), activation=activation, input_dim=len(X_train[0])))

    # Allow kerastuner to decide number of hidden layers and neurons in hidden layers
    for i in range(hp.Int('num_layers', 1, 5)):
        nn_model.add(tf.keras.layers.Dense(units=hp.Int('units_' + str(i),
            min_value=1,
            max_value=30,
            step=5),
            activation=activation))

    nn_model.add(tf.keras.layers.Dense(units=1, activation="relu"))

    # Compile the model
    nn_model.compile(loss="binary_crossentropy", optimizer='adam', metrics=["accuracy"])

    return nn_model
```

Data Exploration

Given the dataset, which contains both numerical and categorical data, and the purpose of the project, which seeks to predict a house's price given various features, we landed on a Random Forest Regressor Model as most appropriate for this project.



Tools Used

Workflow

The primary file is: RandomForestRegressor.ipynb.
An outline of the workflow is as follows:

1. Import Dependencies
2. Import Data
3. Data Cleaning
4. Finding Best Parameters
5. Database - SQLite
6. Random Forest Model Creation
7. Export necessary information for storyboard / dashboard

1. Combined data
2. Education data
3. Crime data

1. Renaming columns
2. Dropping columns
3. Combining tables via joins

- Using RandomizedSearchCV

- Send to SQLite DB
- Retrieve from SQLite DB

1. Split into train/test
2. Fit & Evaluate
3. Predictions
4. Decision tree

Random Forest Model

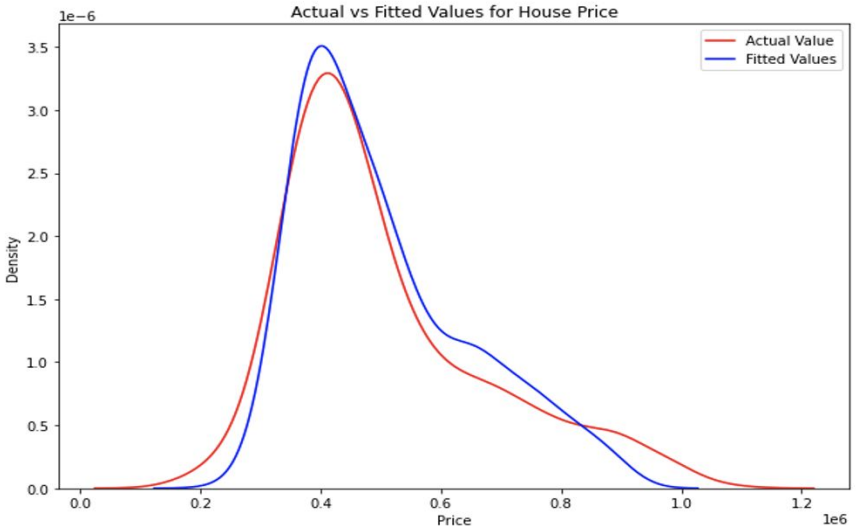
We used a RandomForestRegressor algorithm (i.e., a supervised learning algorithm and an ensemble of a decision trees algorithm) for modeling:

- It is supervised algorithm because during the model training, it learns the mappings between the input features (e.g., Number of bedrooms) and the outputs feature or target (House Price).
- It is an assemble algorithm because it combines or assembles multiple decision trees into a final decision to make a more accurate prediction than any underlying algorithm could on its own.

Random Forest Model

This model is one of the most popular algorithms for regression problems that is used to predict continuous outcomes due to its simplicity and high accuracy. This algorithm consists of two steps process, i.e., Building n decision tree regressors (estimators), and finding an average prediction across estimators.

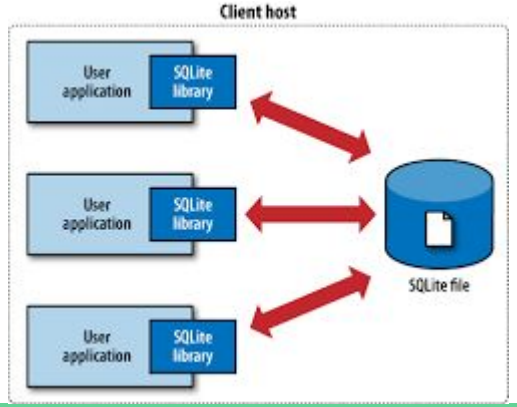
Random Forest Model



The regression using decision trees starts with the selection of attribute values to determine the best split, and once the best split is found, the dataset is split at that value (i.e., the root node) and the process is repeated for all the other ranges until a stopping condition (e.g., Maximum depth, or a minimum number of samples) is reached. Below is a graph depicting the actual house prices versus the prices predicted by the model.

SQLite

SQLite typically works well as the database engine for low to medium traffic websites (which is to say, 99.9% of all websites). SQLite is an open source software, also SQLite a an appropriate database tool to power our dashboard. Generally speaking, any site that gets fewer than 100K hits/day should work fine with SQLite. The 100K hits/day figure is a conservative estimate, not a hard upper bound. SQLite has been demonstrated to work with 10 times that amount of traffic.



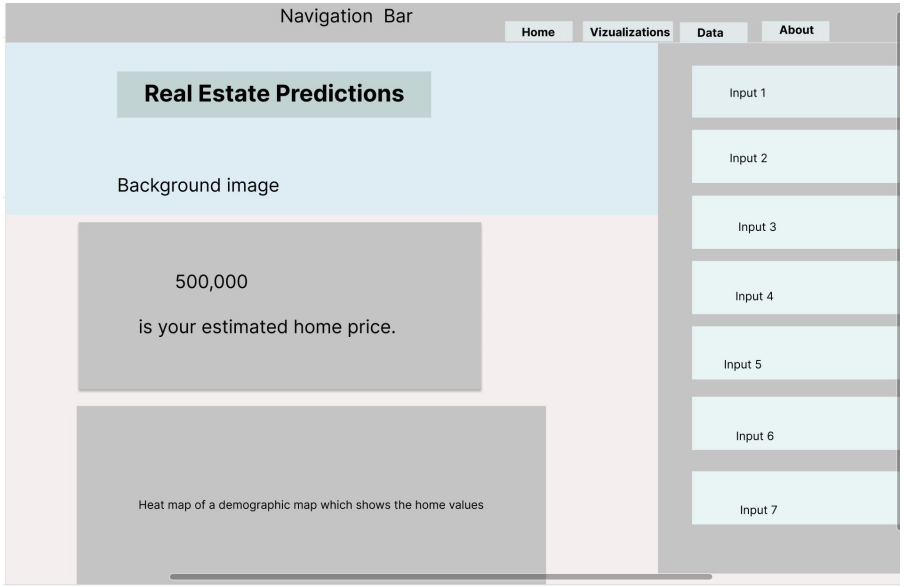
For this project, SQLite will serve to hold our final combined dataset in order to provide the data for visualizations on the dashboard in conjunction with the primary purpose - a predicted home price.

Dashboard

Using a Flask app, the site has a series of input boxes requesting the primary features of a house: zip code, square footage, bedrooms, bathrooms, and year built.

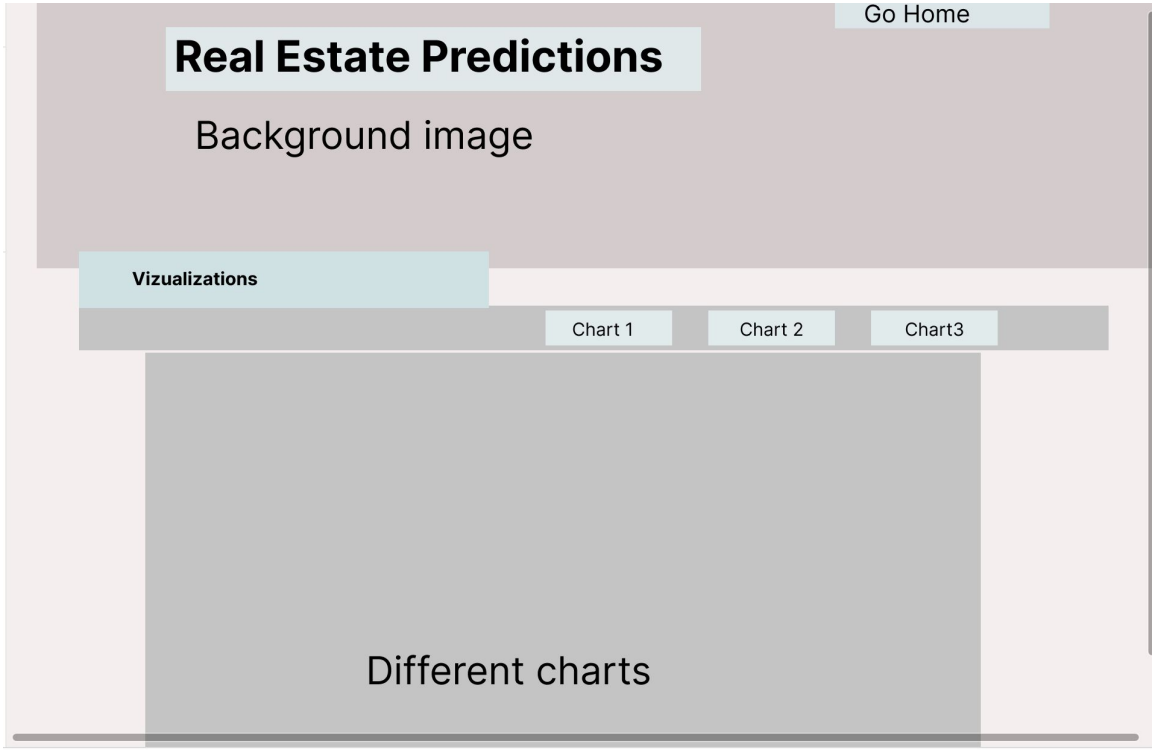
Submitting the search will result in calling the prediction model and returning a predicted price according to the parameters provided.

Page 1



Dashboard

Page 2



Dashboard

Page 3

Dashboard

Page 4