Milestone Report: Model to estimate most affordable place to live in America

Introduction: Many job seekers in the US, especially when it comes to tech jobs, often think about where the most opportunities are, and flock to popular job markets located in places like New York, Los Angeles, or San Francisco. The problem with this approach is while they may have high paying jobs, the jobs in these places also are associated with inflated living and rent costs due to the demand for housing. As a result, these places can often be unaffordable or uneconomical to the average job seeker. This makes it harder to fulfill economic goals many of us have, whether it is to save enough for a property or to save for retirement.

In this project, I wanted to see where the most affordable places to live and find work are, factoring in income and rent prices. I also wanted to build a model that would predict housing prices given income data and population size of a particular area.

The objectives of this project are as follows:

* Build a model able to predict housing prices given income data and/or population size.
* Figure out what the most affordable/least affordable places are to live.

Potential clients:

* New job seekers hoping to find a job market that is advantageous to them
* Real estate clients looking for up and coming areas to build housing.

Datasets:

1. Zillow House Price Data: <https://www.kaggle.com/paultimothymooney/zillow-house-price-data/data#City_MedianRentalPrice_Sfr.csv>
2. 2015 Median Income by County Data

<https://data.world/tylerudite/2015-median-income-by-county/workspace/file?filename=2015+Median+Income+by+County.csv>

**Data Wrangling:**

I imported two datasets from the two links above and found the 2015 Median Income by County Data.csv, City\_MedianRentalPrice\_1Bedroom.csv, City\_MedianRentalPrice\_2Bedroom.csv, and City\_MedianRentalPrice\_3Bedroom.csv. I took the following steps to clean the data:

Creating the DataFrame: I loaded the datasets using the pd.read\_csv function of all the datasets I required.

Checking the DataFrames: Inside the DataFrames I saw that the rent prices were from cities while the income data was from counties. I also found that the rent prices had different entries for each month of the year.

Selecting Categories: For the rental sets, I selected the RegionName (the column names for the cities), CountyName, and State, as well as the months and years I would need. I first used the ones from 2019 months and I followed the exact same process for the 2015 months. While the 2015 months are the months of interest, since the Income Data is also from 2015, I used the 2019 months first, since it was a more complete dataset (had less NaN values). For the income data, I left it whole. I would merge it with the other dataset later. I added a CountyState column into the income data by combining the County and State columns. I would use this later to merge both datasets together.

Melting the DataFrames: I used the pd.melt function to get all of the months under one column, since the csv file kept them in each of their own columns.

Grouping Counties Together: Using the groupby function on the melted DataFrame with the mean function, I was able to get the 2019 means of the data for all of the rent DataFrames for each specific county. I used the CountyState column to avoid aggregating counties that had the same name, but in different states. I did the same with the 2015 data as well.

Merging 2015 and 2019 Data: Using the pd.merge function I combined with 2015 and 2019 rent datasets.

Filling NaN values: Using the ols function from the statsmodel library, I found that the bedroom prices from 1 bedroom prices in 2015 and 2019 were highly correlated, and used the coefficients of the regression to fill in the values in the 2015 dataset by multiplying the coefficients to the 2019 values. I also dropped an outlier in the 3 bedroom DataFrame.

Merging income data with rental price data: I used the merge function from the pandas library again to merge the income data with the rental data and dropped the redundant or unneeded data, I then merged each of the different rent price datasets together and filled NaN values where needed. I also added the columns ‘1B\_Price/Income\_Ratio, 2B\_Price/Income\_Ratio, and 3B\_Price/Income\_Ratio, which was created by dividing a year’s worth of rent from the income. Economists usually recommend people spend below 30% of their income on housing.

Evaluating merged DataFrame: Looking at the merged DataFrame, there were still NaN values, since some of the data of the different rent pricing did not overlap.

Filling more NaN values: To remedy the missing NaN values, I again used the ols function from the statsmodel library to find the R^2 and coefficients between 1 and 2 bedroom prices and 2 and 3 bedroom prices to fill the remaining NaN values. I used the same procedure as the first pass where I used the coefficients to fill the remaining values. I filled the NaN values in the Price/Income Ratios, by dividing the rent prices from the median household income for the year.

Adding more Features: To do data analysis on comparisons of different states, I created the State Code-cat variable, and added it to my model.

**Exploratory Data Analysis and Inferential Statistics:**

Checking collinearity with independent and dependent variables: I checked the collinearity using the ols function again with the bedroom prices vs income and population and found that they have a moderate correlation, but low R^2 values. I later confirmed this via hypothesis testing. I used the pearsonr function from scipy.stats to evaluate the correlation between bedroom rent prices and income and the p-value to hypothesis test.

Testing if data has a normal distribution: I graphed histograms of the bedroom rent prices and the0 Using the chi-square test, I found that my datasets were all not normally distributed. After taking the log of the data and rechecking with the chi-square test, I found out that the datasets would become normally distributed if I took the logs of them.

T testing: Using the t function from scipy.stats, I created a 95th percentile confidence interval for the bedroom rent pricing. Since the data looks to be one-tailed, I created the one tail instead of a normal two-tailed test.

Plots: I plotted scatter plots to visually see if there was correlation between bedroom prices and income and bedroom prices and population. I also drew a regression plot using seaborn of 2015 and 2019 rent data as well as 1 bedroom vs 2 bedroom prices and so on.

Questions answered so far: The least affordable county to live according to this data is in San Francisco County, California where occupants would spend about 51.7% of their income on a single bedroom apartment. The most affordable county would be in Cole County, Missouri, where occupants spend around 7.6% of their income on a single bedroom apartment. Averaging just the states, Washington DC is the least affordable while Missouri remains the most affordable.

Building the model: I would most likely try a Linear Regression model and a RandomForestRegressor to compare the two models in their ability to predict housing prices via population and median household income.

**Model Selection and Development**: Due to the nature of my model to try to predict housing prices in a particular place I elected to try two techniques, a Linear Regression Model and a Random Forest Regressor model, and see which model worked best on my dataset.

**Linear Regression**: Using a Linear Regression model to predict bedroom prices using State, Income, and Population, it actually performs pretty terribly, with an R^2 value of 0.261. The data was split into a training and testing set using train\_test\_split at 80% used for training and 20% used for testing. Since the model’s poor performance, I elected to try using a Random Forest Regressor, since the data does not necessarily need to be linearly correlated to be able to predict accurately. It is also quite robust and one of the more accurate algorithms out there.

**Random Forest Regressor:** I tried using the Random Forest Regressor untuned, with the same training/testing split and the model performed better than the Linear Regression with an R^2 square score of 0.487, that is, explains 48.7% of the variance. It also had a mean absolute error of 2392.5607538104355. However, after tuning the hyperparameters improved both the accuracy and error values of the model.

**Tuned Random Forest Regressor:** The hyperparameters used to train the dataset were as follows:

* max\_depth: A list from 1-20
* N\_estimators: a list from 1-20
* Max\_features: a list from 1-3

Using GridSearchCV with 3 partitions, the model was tuned with the training set, and had the following calibrations:

Tuned Random Forest Regressor Parameters: {'max\_depth': 9, 'max\_features': 1, 'n\_estimators': 15}

And a Best Score of : 0.5653799844397189

On the testing set, the model had a R^2 value of 0.6309356679115057 and a mean absolute error value of 1977.9496510089637. By tuning the model, the accuracy improved by 14.3% and had a reduced error.

**Possible biases:** The data available is more available in more popular, urban counties than other rural counties across the US.

**Next steps:** Next steps during this process would be to add more features and data to this project, maybe something like income tax rate or political party affiliation etc. to have a more applicable model. Other improvements include adding different time sets for the data in order to show different trends for different housing markets all over the country.