**University of North Texas**

**ADTA 5340-412**

**Discovery and Learning with Big Data/Data Discovery**

**Predicting Median Housing Prices**

Group Members - Group 2: Binod Nepal, Sharon Schoolcraft, Biniam Abebe and Maybel Hernandez

**Abstract:**

This project focused on predicting median housing prices in the United States, with the goal of providing valuable insights for stakeholders such as homebuyers, sellers, investors, and urban planners. By analyzing a real estate dataset from Realtor.com, we can uncover important market trends and factors influencing housing prices.

We used both linear and polynomial regression models, finding that the polynomial regression model performed better at capturing data patterns and making more accurate predictions.

Moving forward, we recommend expanding data sources to include additional factors, using more advanced data preparation techniques, exploring ensemble methods and neural networks, creating user-friendly interfaces for model deployment, and maintaining thorough documentation for ongoing improvement.

This research sets the stage for further investigation into regional market variations and how different factors impact prices. Sharing these insights can significantly advance real estate analytics research.

**Keywords:** *housing price prediction, real estate analytics, data analysis, machine learning, linear regression, polynomial regression, model performance, data collection, data preparation, model deployment, documentation*

Contents

[Abstract: 1](#_Toc165823151)

[I. Introduction: 3](#_Toc165823152)

[II. Data Description: 3](#_Toc165823153)

[III. Data Limitation: 4](#_Toc165823154)

[IV. Exploratory Data Analysis 5](#_Toc165823155)

[V. Methodology: 10](#_Toc165823156)

[VI. Model Development: 13](#_Toc165823157)

[References: 20](#_Toc165823158)

[Appendix: 20](#_Toc165823159)

1. **Introduction:**

Having an accurate prediction of housing prices would be beneficial for multiple stakeholders which include home buyers, house sellers, investors, and urban planners, when making important decisions for the future. This project revolves around exploring market data of the real estate industry by researching and forecasting property values and their impact on home prices.    
Our project is centered around studying median housing prices using data from realestate.com’s Real Estate Data and Market Trends. The goal is to explore the numerous factors that can impact housing prices and better decipher the housing market. Using our research and analysis, we aim to provide valuable insights that can empower individuals and communities allowing them to successfully navigate the real estate market with confidence helping them make well-informed decisions.

1. **Data Description:**

The original dataset is a historical view of the monthly median listing prices by counties. The dataset is 288386 rows by 40 columns. The last month of entry for the dataset is March 2024 and the oldest entry is for July 2017. The dataset provides valuable information to help predict prices like time on the market and reductions in cost. The dataset also flags outliers in the final column. The dataset covers listing features of the homes that can account for pricing differences. Using the mean and median of several attributes we will be able to construct a model to help with predicting the median housing price by county in the United States. The dataset will need some cleaning as some columns have percent changes, and normalized values/standardized values. These values can be cleaned and used to enrich the dataset further. There are many categorical variables for analysis within the dataset. We are going to focus on the changes within the counts of the dataset to show how the status of the listing is changing utilizing the proportions of the listings.

**Where to Find the Data:** ("Residential data.") Link: Realtor.com Real Estate Data and Market Trends for Download File Location: Select Monthly Inventory>Historical Data> View County Data. After cleaning and enriching the data set, 14 variables remained. Of these 16 variables, 6 are qualitative and 10 are quantitative.

1. **Data Limitation:**

The data from Relator.com uses various sources of information to get a robust approach to home valuation. Public data sources heavily influence the data estimates generated by realtor.com. Some of the sources that are used are based on recent sales in the area, details about the housing itself, and overall market trends. However, we need to recognize that it is just an estimate, consider its limitations, and view it as a starting point rather than a reliable verdict. No matter how modern computer models are used, they cannot account for everything. Recent renovations, Property condition, and distinctive features are not fully reflected in their assessments for the data from relator.com. Also, if the data in a particular area is limited, they might be outdated, which their estimates may not fully consider. These online valuations provide a starting point but do not consider the home's specifics that might affect its value. Considering this, the estimates might be off. Also, we need to consider that Realtor.com is a marketing platform. The realtor.com data does not include the interest rates. Interest rates are the critical component in the affordability equation for homebuyers. Buying power decreases when interest rates are high, which might lower the actual price of the house.

1. **Exploratory Data Analysis**

**Qualitative Variables:** *Median Market Time:*   
We transformed the variable Median\_Days\_on\_Market by reducing the variable into a categorical feature (Median\_Market\_Time) by rebinning the information into four labels: Rapid (less than 45 days), Normal (46 to 90 days), Long (91 to 145 days), and Stale (longer than 146 days (about 5 months).) With these four labels we can see that most homes are on the market for 46 to 90 days (about 3 months) with a count of 153,665 passing through the market at a normal pace, which is about 53.8%. The next highest value is the Stale. These median market times have been on the market for over 146 days (about 5 months) with a frequency of 75,321 that have prolonged or abnormal time in the marketplace. Stale make up about 26%. Our label ranked third is Rapid. These homes move through the marketplace in 45 days (about 1 and a half months) or less. We see a frequency of 56,738 median market times rapidly moving through the marketplace with 19.6%. Finally, we see the last label Long with the lowest count of 2661. This makes up an insignificant amount with a portion of less than 1% of the median market times.

A graph with blue squares

Description automatically generated

*Number of Houses Listed by Season:*

The variable month\_date\_yyyymm was split and sectioned off into months and years. Then the months were reduced by binning into seasons. December to February is considered winter months, with a count of 74,394. March to May is considered spring months, with a count of 68,198. June to July is considered summer months, with a count of 71,331. September to November is considered fall months, with a count of 74,462. The bar chart shows the highest number of listings occurring in the fall with the smallest number of listings occurring in the spring.

A graph of different colored squares

Description automatically generated

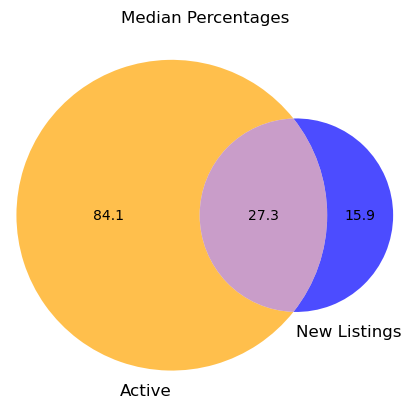
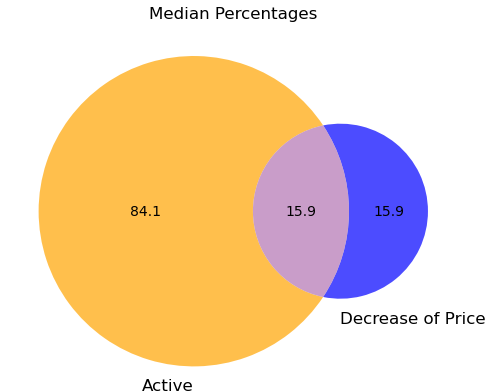
**Quantitative Variables:**  
*New Listing Proportions:* The number listing proportion was derived from the new listing count divided by total counts. For a house to be considered a new listing is must be less than 30 days (about 4 and a half weeks) old. The median proportion for new listing in the marketplace is 2.4%.

*Price Reduced Proportions:* Price Reduced and New Listings have a mutually exclusive relationship (assumed as it is typically in industry and only occurs as an error otherwise.) Typically, a price reduction occurs after a listing has lost interest and is not attracting buyers. This proportion was generated using the Price Decrease Count divided by the total count. 1.3% of listings are receiving a price deduction.

*Price Increase Proportions:* There is not a median amount of price increase occurring within the marketplace. The proportion was calculated by placing the Price Increase Count in the numerator and the Total Listing Count in the denominator. The proportion was equal to 0%, meaning we are seeing almost no increases within the listings.

*Active Listing Proportions:* Active listings and Pending Listings have a mutually exclusive relationship (assumed as it is typically in industry and only occurs as an error otherwise.) The Proportion was taken by putting the Active Count from the original data set over the total count to find the proportion. The active listings have a median proportion of 76.2%.

*Pending Listing Proportions:* The Proportion was taken by putting the Pending Listing Count from the original data set over the total count to find the proportion. The Pending Listings have a median proportion of 28.9%.



*Venn Diagrams:* were created by taking the mutually exclusive relationships between variables and scaling the value to 100. The overlapping region is the area of New Listings or Price Decreases in both regions. This gives a proportional relationship with respect to the relationship each value holds in the marketplace mapped in a visual. The Reduced Listings shows how much of the listings receive a price reduction. This visual aid was used utilizing the median counts of occurrences and confirmed with the proportions from the enriched dataset.

*Distribution of Housing Median Prices*

A graph of a distribution of housing median prices

Description automatically generated  
*Histogram*: The histogram visualizes the distribution of median\_listing\_price, with the x-axis representing house prices and the y-axis representing the count of the houses falling within each price interval. The histogram reveals a right-skewed distribution, characterized by a longer tail on the right side and a concentration of prices on the lower end of the distribution. This skewness suggests that a few houses have significantly higher prices compared to most properties, indicating potential market outliers or premium real estate segments.

*Interpretation*: The right-skewed distribution of median\_listing\_price implies that while most houses are priced relatively lower, there exist a few high-value properties that contribute to the overall skewness of the distribution. Understanding the underlying factors driving these high-priced properties is essential for identifying market trends, assessing risk, and developing targeted marketing or investment strategies.

1. **Methodology:**

##### **Data Preprocessing**

##### **Dummy variables**

Using Panda's Python library, we preprocessed the Home Dataset by creating dummy variables for the categorical features 'season' and 'median\_market\_time.' This transformation was crucial as it converted categorical data into a numerical format that could be processed more efficiently by our subsequent machine-learning models. The get\_dummies function was used for this purpose, which generated a new DataFrame where each unique category in the specified columns would be represented as a binary (0 or 1) column within the DataFrame. We could easily compare different categorical values since the columns were directly comparable to one another, making it easier for us to compare them.

**Code Snippet**

*#dummy variables*

*HomesEnriched\_dummies = pd.get\_dummies(HomesEnriched, columns=['season','median\_market\_time'])*

*HomesEnriched\_dummies.head()*

##### **Scale the data**

Our next step was to implement data standardization as a key step in preprocessing data. To achieve this, the StandardScaler class of the sklearn-preprocessing module in Python was used. By using the StandardScaler, features are standardized by removing the mean from the data (scaling to unit variance) and removing the standard deviation from the data (centering the data around zero). The StandardScaler has been instantiated, and it has been applied to several columns of our DataFrame HomesEnriched\_dummies, including 'median\_listing\_price', 'active\_listing\_prop', 'median\_days\_on\_market', 'new\_listing\_prop', 'price\_increased\_prop', 'price\_reduced\_prop', 'pending\_listing\_prop', 'median\_square\_feet', 'average\_listing\_price', and 'total\_listing\_count'. We used the fit\_transform method to fit the scaler to the data and then transform the data based on the fit. After the transformed (standardized) data had been re-assigned to the corresponding columns of the HomesEnriched\_dummies DataFrame, it was ready to be analyzed. The purpose of this step was to transform the data into a format that would allow our upcoming machine-learning models to better process the data, thereby improving the performance of our subsequent models.

**Code Snippet**

*# Scale the data*

*from sklearn.preprocessing import StandardScaler*

*scaler = StandardScaler()*

##### **Features selection**

Finally, we used a feature selection process to identify the most relevant predictors of our target variable, 'median\_listing\_price'. The SelectKBest class from the sklearn.feature\_selection module selects features based on the k highest scores of a scoring function. For our case, we used the f\_regression scoring function and set k=5 to select the top 5 features.

We prepared our dataset by splitting it into a feature matrix X and a target variable Y. We then divided these into training and test sets using train\_test\_split.

After fitting the SelectKBest selector to our training data, we used the inverse\_transform method to zero out all non-selected features. By identifying the non-zero variance features in the transformed feature matrix, we were able to easily identify the selected features.

In our data preprocessing, feature selection was critical as focusing on the most relevant features allowed us to reduce the dimensionality of our data and potentially improve the performance of our machine-learning models.

**Code Snippet**

*#Find the best features*

*from sklearn.feature\_selection import SelectKBest, f\_regression*

*from sklearn.feature\_selection import chi2*

*# Split the data*

*X = HomesEnriched\_dummies.drop(['year', 'month', 'county', 'state', 'median\_listing\_price'], axis=1)*

*y = HomesEnriched\_dummies['median\_listing\_price']*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

*# Find the best features*

*selector\_f = SelectKBest(f\_regression, k=5)*

*X\_new = selector\_f.fit\_transform(X\_train, y\_train)*

*# Get back the features we have kept, zero out all other features*

*selected\_features = pd.DataFrame(selector\_f.inverse\_transform(X\_new),*

*index=X\_train.index,*

*columns=X\_train.columns)*

*selected\_columns = selected\_features.columns[selected\_features.var()! = 0]*

*selected\_columns*

1. **Model Development:**

For multiple regression, we are using scikit-learn library. It is a comprehensive machine learning library that provides efficient tools for data analysis and modeling. For the model selection we imported train\_test\_split function. Which is commonly used in machine learning for splitting a dataset into two subsets: one for training the model and another for testing its performance. This function helps in evaluating the performance of the model on unseen data, which is crucial for assessing its generalization ability and detecting overfitting. For Linear Regression we imported class LinearRegression specifically designed for modeling the relationship between one or more independent variables (features) and a continuous dependent variable. And, to evaluate performance of regression models we imported function mean\_squared\_error. It calculates the mean squared error (MSE) between the actual values of the target variable and the predicted values produced by the model.

**Code Snippet**

X = HomesEnriched\_dummies[selected\_columns]

y = HomesEnriched\_dummies['median\_listing\_price']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a linear regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Make predictions

y\_pred = model.predict(X\_test)

# Create a polynomial regression model

poly = PolynomialFeatures(degree=2)

X\_poly = poly.fit\_transform(X\_train)

poly.fit(X\_poly, y\_train)

**Performance Metrics**

**Mean Squared Error:** MSE measures the average squared difference between actual and predicted values. Which will help to see how well the model fits the data. The lower MSE value means Model predictions are closer to the actual values and higher MSE values mean larger discrepancies, which means poor model performance.

**Root Mean Squared Error:** RMSE is the square root of the MSE, providing an interpretable measure in the same units as the dependent variable. Lower RMSE indicates better model performance and higher RMSE indicates predictions have larger discrepancies from the actual values. That means the model has poorer performance in terms of accuracy.

**Mean Absolute Error:** MAE is a metric used in regression analysis to measure the average absolute difference between the actual and predicted values of the target variable. Lower MAE indicates better model performance which reflects smaller discrepancies between prediction and actual values.

**The Coefficient of Determination (R-squared):** R-Squared represents the proportion of the variance in the dependent variable explained by the independent variables in a regression Model. It ranges from 0 to one, where 0 indicates that model does not explain any variability in the dependent variable and 1 indicates that it explains perfectly all the variability.

**Code Snippet**

*# Evaluate the model*

*mse = metrics.mean\_squared\_error(y\_test, y\_pred)*

*rmse = metrics.mean\_squared\_error(y\_test, y\_pred, squared=False)*

*mae = metrics.mean\_absolute\_error(y\_test, y\_pred)*

*r2 = metrics. r2\_score (y\_test, y\_pred)*

Results

| **Model** | **MSE** | **RMSE** | **MAE** | **R2** |
| --- | --- | --- | --- | --- |
| Linear Regression | 0.397758 | 0.630680 | 0.307621 | 0.629589 |
| Polynomial Regression | 0.205956 | 0.453824 | 0.205571 | 0.808204 |

What these metrics indicate about the performance of each model:

**MSE (Mean Squared Error):**

Linear Regression: 0.397758

Polynomial Regression: 0.205956

MSE measures the average squared difference between the estimated and actual values. A lower MSE indicates a model with better accuracy. Here, the Polynomial performs significantly better than the Linear Regression model.

**RMSE (Root Mean Squared Error):**

Linear Regression: 0.630680

Polynomial Regression: 0.453824

RMSE is the square root of MSE and measures the average magnitude of the error. As with MSE, lower values are better. Polynomial Regression again shows superior performance over Linear Regression.

**MAE (Mean Absolute Error):**

Linear Regression: 0.307621

Polynomial Regression: 0.205571

MAE calculates the average magnitude of errors in a set of forecasts without regard for direction (i.e., no squaring). Lower MAE values suggest that the model's predictions are more accurate. Polynomial models have lower MAE, indicating that they are more accurate.

**R² (R-squared):**

Linear Regression: 0.629589

Polynomial Regression: 0.808204

The R2 statistic indicates how much of the variation in the dependent variable can be predicted from the independent variables as a function of the independent variables. The higher the value, the better the fit. From the result table above, the polynomial regression model has much higher R2 values than the linear regression model, indicating that it explains more of the dataset's variability.

1. **Conclusion and Recommendations**

Using a structured and iterative approach, our project focused on predicting median housing prices, which is an important factor for stakeholders like home buyers, sellers, investors, and urban planners. After thoroughly exploring and analyzing data, we have discovered valuable insights that can help inform decisions in the real estate market.

During our data preparation phase, we carefully cleaned and improved the data from Realtor.com to make sure it was ready for modeling. This included dealing with missing information, adjusting variables, and creating new features by augmenting and adding data to help make our models more accurate. We tested both linear and polynomial regression methods, thoroughly reviewing their effectiveness through metrics like Mean Squared Errors (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and the Coefficient of Determination (R2).

Our analysis showed that the Polynomial Regression model performed better than the Linear Regression model, indicating its superior capability to capture the underlying data patterns and offer more precise predictions.

**Recommendations:**

To further enhance our ability to predict outcomes, we propose the following recommendations:

*Data Collection and Understanding*: Enhance data collection methods by including a variety of sources and factors, like neighborhood amenities, crime rates, and economic indicators. Work with local real estate agencies and government bodies to gather thorough datasets that provide a well-rounded view of housing markets.

*Data Preparation*: Enhance data processing skills by incorporating advanced techniques that can effectively handle complex data structures, outliers, and missing values. Additionally, consider feature engineering strategies to extract valuable insights from data and improve the performance of models.

*Modeling and Evaluation*: Explore different machine learning algorithms, like ensemble methods and neural networks, this can help determine the most cost-effective models for predicting housing prices. Using techniques such as cross-validation can provide a more accurate assessment of models’ reliability and performance.

*Deployment and Monitoring*: Create easy-to-use interfaces or dashboards for deploying predictive models and allowing stakeholders to interactively explore housing market trends and predictions. Set up monitoring tools to track model performance and make it easier to retrain the model with new data.

*Documentation and Communication*: Maintain detailed records of the data mining process, from data collection to modeling and evaluation. Share findings and insights with stakeholders in clear and easy-to-understand reports, presentations, and visualizations.

By following this iterative process and constantly improving methods, we can deepen our knowledge of housing market dynamics and provide important insights to the real estate research community.

**Further research:**

It would be beneficial for future research to delve into the distinct market trends seen in different regions, explore how various factors impact housing prices, and use advanced modeling techniques to enhance the accuracy of predictions. Sharing findings from these research efforts could greatly add to the overall understanding and knowledge in the real estate analytics field.

# References:

1. Residential data, December 2023, Apr 13, 2024, <[https://www.realtor.com/research/data/>.](https://www.realtor.com/research/data/%3e.)

2. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

# Appendix:

#Import Packages

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

from sklearn import metrics

# import the necessary libraries

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean\_squared\_error,mean\_absolute\_error, r2\_score, accuracy\_score, confusion\_matrix, classification\_report

#Remove warnings

import warnings

warnings.filterwarnings("ignore")

from sklearn.ensemble import AdaBoostClassifier

from sklearn.naive\_bayes import GaussianNB

base = GaussianNB()

ada\_clf = AdaBoostClassifier(base\_estimator=base, n\_estimators=50, learning\_rate=1, random\_state=123)

#Load the Dataset \*\*\*Note: You must change the path

Homes = pd.read\_csv("RDC\_Inventory\_Core\_Metrics\_County\_History 2.csv")

#Preview Dataset in Python

display(Homes.head())

display(Homes.tail())

#Check the shape of the dataset

Homes.shape

#Categorical Variables Clean: Split County Name and State Column

Homes[['county', 'state']] = Homes['county\_name'].str.split(',', expand=True)

#Categorical Variables Clean: Split Year and Month

Homes['month\_date\_yyyymm'] = pd.Series(Homes['month\_date\_yyyymm'], dtype="string")

Homes['year'] = Homes['month\_date\_yyyymm'].str[0:4]

Homes['month'] = Homes['month\_date\_yyyymm'].str[-2:]

display(Homes.head())

#For the purpose of the proect we only need some of the columns.   We can select them and rename the data.

HomesCleaned = Homes[['year', 'month', 'county', 'state', 'median\_listing\_price', 'active\_listing\_count', 'median\_days\_on\_market', 'new\_listing\_count',

               'price\_increased\_count', 'price\_reduced\_count', 'pending\_listing\_count', 'median\_square\_feet', 'average\_listing\_price',

               'total\_listing\_count']]

display(HomesCleaned.head())

display(HomesCleaned.tail())

#Add new listing Prop

Homes['new\_listing\_Prop'] = Homes['new\_listing\_count'] / Homes['total\_listing\_count']

Homes.new\_listing\_Prop.head()

#Add price reduced Prop

Homes['price\_reduced\_Prop'] = Homes['price\_reduced\_count'] / Homes['total\_listing\_count']

Homes.price\_reduced\_Prop.head()

#Add price increased Prop

Homes['price\_increased\_Prop'] = Homes['price\_increased\_count'] / Homes['total\_listing\_count']

Homes.price\_increased\_Prop.tail()

#Add pending\_listing\_Prop

Homes['pending\_listing\_Prop'] = Homes['pending\_listing\_count'] / Homes['total\_listing\_count']

Homes.pending\_listing\_Prop.head()

#Add active\_listing\_Prop

Homes['active\_listing\_Prop'] = Homes['active\_listing\_count'] / Homes['total\_listing\_count']

Homes.active\_listing\_Prop.head()

HomesEnriched = Homes[['year', 'month', 'county', 'state', 'median\_listing\_price', 'active\_listing\_Prop', 'median\_days\_on\_market', 'new\_listing\_Prop',

               'price\_increased\_Prop', 'price\_reduced\_Prop', 'pending\_listing\_Prop', 'median\_square\_feet', 'average\_listing\_price',

               'total\_listing\_count']]

HomesEnriched.head()

HomesEnriched.tail()

#Find missing Values

mean = HomesEnriched.mean(numeric\_only=True)

HomesEnriched.fillna(value=mean, inplace=True)

HomesEnriched['median\_days\_on\_market'] = HomesEnriched['median\_days\_on\_market'].astype(int)

print(type('median\_days\_on\_market'))

print("Unique MDOM Count: " +str(HomesEnriched.median\_days\_on\_market.nunique()))

print(HomesEnriched.median\_days\_on\_market.value\_counts())

#Create a new column for the median\_market\_time

def conditions(s):

    if s['median\_days\_on\_market'] <= 45:

        return 'Rapid'

    if 46 <= s['median\_days\_on\_market'] <= 90:

        return 'Normal'

    if s['median\_days\_on\_market'] in [91, 145]:

        return 'Long'

    else:

        return 'Stale'

HomesEnriched['median\_market\_time'] = HomesEnriched.apply(conditions, axis=1)

HomesEnriched.head()

#change the create new column from month using the as four seasons

def conditions(row):

    month = int(row['month'])  # Convert to int here

    if month in [12, 1, 2]:

        return 'Winter'

    elif month in [3, 4, 5]:

        return 'Spring'

    elif month in [6, 7, 8]:

        return 'Summer'

    else:

        return 'Fall'

# Convert the entire column before applying the function

HomesEnriched['month'] = HomesEnriched['month'].str.replace('o', '').astype(int)  # Convert the entire column before applying the function

HomesEnriched['season'] = HomesEnriched.apply(conditions, axis=1)

HomesEnriched.season.shape

HomesEnriched.tail()

#Prepare the data

#dummy variables

HomesEnriched\_dummies = pd.get\_dummies(HomesEnriched, columns=['season','median\_market\_time'])

HomesEnriched\_dummies.head()

#find where Input contains infinity or a value too large for dtype('float64').

HomesEnriched\_dummies.replace([np.inf, -np.inf], np.nan, inplace=True)

HomesEnriched\_dummies.fillna(0, inplace=True)

# scale the data

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

HomesEnriched\_dummies[['median\_listing\_price', 'active\_listing\_Prop', 'median\_days\_on\_market', 'new\_listing\_Prop', 'price\_increased\_Prop',

               'price\_reduced\_Prop', 'pending\_listing\_Prop', 'median\_square\_feet', 'average\_listing\_price', 'total\_listing\_count']] = scaler.fit\_transform(HomesEnriched\_dummies[['median\_listing\_price', 'active\_listing\_Prop', 'median\_days\_on\_market', 'new\_listing\_Prop', 'price\_increased\_Prop',

               'price\_reduced\_Prop', 'pending\_listing\_Prop', 'median\_square\_feet', 'average\_listing\_price', 'total\_listing\_count']])

#change string to numeric

HomesEnriched\_dummies.columns

#find the best features

from sklearn.feature\_selection import SelectKBest, f\_regression

from sklearn.feature\_selection import chi2

# split the data

X = HomesEnriched\_dummies.drop(['year', 'month', 'county', 'state', 'median\_listing\_price'], axis=1)

y = HomesEnriched\_dummies['median\_listing\_price']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# find the best features

selector\_f = SelectKBest(f\_regression, k=5)

X\_new = selector\_f.fit\_transform(X\_train, y\_train)

# get back the features we've kept, zero out all other features

selected\_features = pd.DataFrame(selector\_f.inverse\_transform(X\_new),

                                 index=X\_train.index,

                                 columns=X\_train.columns)

selected\_columns = selected\_features.columns[selected\_features.var() != 0]

selected\_columns

# split the data

X = HomesEnriched\_dummies[selected\_columns]

y = HomesEnriched\_dummies['median\_listing\_price']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

#Use a fitted multiple regression model to make predictions.

# Create a linear regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Make predictions

y\_pred = model.predict(X\_test)

# Evaluate the model

mse = metrics.mean\_squared\_error(y\_test, y\_pred)

rmse = metrics.mean\_squared\_error(y\_test, y\_pred, squared=False)

mae = metrics.mean\_absolute\_error(y\_test, y\_pred)

r2 = metrics.r2\_score(y\_test, y\_pred)

print(f'MSE: {mse}')

print(f'RMSE: {rmse}')

print(f'MAE: {mae}')

print(f'R2: {r2}')

# Create a polynomial regression model

poly = PolynomialFeatures(degree=2)

X\_poly = poly.fit\_transform(X\_train)

poly.fit(X\_poly, y\_train)

model = LinearRegression()

model.fit(X\_poly, y\_train)

# Make predictions

y\_pred = model.predict(poly.fit\_transform(X\_test))

# Evaluate the model

poly\_mse = metrics.mean\_squared\_error(y\_test, y\_pred)

poly\_rmse = metrics.mean\_squared\_error(y\_test, y\_pred, squared=False)

poly\_mae = metrics.mean\_absolute\_error(y\_test, y\_pred)

poly\_r2 = metrics.r2\_score(y\_test, y\_pred)

print(f'MSE: {poly\_mse}')

print(f'RMSE: {poly\_rmse}')

print(f'MAE: {poly\_mae}')

print(f'R2: {poly\_r2}')

# create a table with the results include the result using the pipeline

results = pd.DataFrame({'Model': ['Linear Regression', 'Polynomial Regression', 'Pipeline'],

                        'MSE': [mse, poly\_mse, pipeline\_mse],

                        'RMSE': [rmse, poly\_rmse, pipeline\_rmse],

                        'MAE': [mae, poly\_mae, pipeline\_mae],

                        'R2': [r2, poly\_r2, pipeline\_r2]})

results