Non-Parametric Analysis of Sonoma County House Prices

Brian McMurray

Western Governors University

WGU Student ID XXXXXXXXXXX

Abstract

The features used to predict house prices can include categorical independent variables (IVs) with high cardinality. This paper focuses on comparing different approaches for encoding high cardinality IVs, and their impact on predictive accuracy across multiple different regression models. This paper will compare Multiple Correspondence Analysis (MCA), K-Modes clustering, James Stein target encoding, M-Measure encoding, CatBoost encoding, and a baseline of removing the categorical features from the data entirely, using multiple predictive models.

Contents

Abstract 2

A1. Introduction 4

Context 4

Research Question 4

Hypothesis 5

B. Data Collection 5

Challenges 11

C. Data Extraction and Preparation 11

D. Analysis 16

Exploratory Data Analysis 16

Categorical Encoders 20

Predictive Models 24

Research Analysis 25

Tools Justification 39

Data Summary and Implications 40

References 42

A1. Introduction

Context

Before a house can be listed for sale, a price needs to be set. There are many methods for determining the house price, including using the sale prices from recently sold comparable homes, the seller determining the minimum amount that they would be willing to accept, or using machine learning methods to predict the value.

There are many factors that can be used for predicting the price of a house. Features available through the Multiple Listing Service (MLS) have been used for predicting house prices by other studies to some success (Park & Bae, 2015). There are many features available for a house that have a high level of cardinality (Calhoun, 2001), which can be difficult to utilize when building a predictive model. This paper utilizes different methods of encoding these high cardinality categorical features and compares their impact on the accuracy of predicting house prices.

Research Question

When building a predictive model, the dataset needs to be in a form that is compatible with the model. This typically includes converting categorical features to some numerical representation. There are a multitude of different methods for achieving this, each with its own effectiveness. The effectiveness of any individual categorical encoding method depends on the data. This paper will address the research question: Do different categorical encoding methods impact the predictive accuracy of house prices?

For this paper, three years of historical house sales data from Sonoma County will be used as the dataset to compare M-Measure Encoding, James Stein Target Encoding, Multiple Correspondence Analysis, K-Mode Clustering, and CatBoost Encoding. The aforementioned encoding methods will be used to encode their own version of the dataset, which will be used to test their impact on the accuracy of CatBoost, RandomForest, Gradient Boost, and XGBoost regression models.

Hypothesis

The root mean squared error, mean absolute error, R^2, and $R^2(Adj)$, for all predictive models will be recorded for each of the datasets created by the various categorical encoding methods which will be tested. Since the splitting of the dataset for training and testing validation, as well as the initial parameters of the models, can have an impact on the performance metrics, twenty tests for each combination of categorical encoding method and model will be performed. The performance metrics from the tests will be used to test the following hypothesis:

Hypothesis $\mathbf{H_0}$: Study factors do not impact house price.

Hypothesis $\mathbf{H_1}$: Study factors do impact house price.

B. Data Collection

In order to predict the price of a house, adequate data is needed. Typically this data includes features of the house, as well as local recent market statistics for the housing market. The data used for this paper comes from two different sources. The first data source comes from scraping a Sonoma County Multiple Listing Service (MLS) website, REMOVED COMPANY at https://REMOVED/. The second data source comes from RedFin's data center at https://www.redfin.com/blog/data-center/, and includes monthly historical statistics for the Sonoma County housing market.

The data collected from REMOVED website was programmatically scraped. The website included an advanced search feature, which was used to filter the results to all house sales for the past three years in the major cities within Sonoma County. Smaller towns and rural areas were excluded to avoid more anomalous records. The results were also limited to Single Family Homes and Condos/Townhouses. Only the last three years of historical sales records were available.

To collect the data, the Python packages requests and bs4 were imported into a Jupyter notebook that would be used for the web scraping process. The same web session needed to be maintained throughout the scraping process, to ensure that the results were from the same search transaction. The initial search was submitted, using a dictionary for the payload, which specified the parameters of the search.

```
            payload = {
                     "from": "advanced", "SelectSearchBy": "city",
                     "SEARCH_state[CA]": "CA",
                     "SEARCH_county[Sonoma County]": "Sonoma County",
                     "SEARCH_city[Rohnert Park]": "Rohnert Park",
                     "SEARCH_city[Cotati]": "Cotati",
                     "SEARCH_city[Santa Rosa]": "Santa Rosa",
                     "SEARCH_city[Windsor]": "Windsor",
                     "SEARCH_city[Healdsburg]": "Healdsburg",
                     "SEARCH_city[Petaluma]": "Petaluma",
                     "SEARCH_status[S]": "S",
                     "SEARCH_sold_date_range": "3 yr",
                     "SEARCH_type[Single Family Home]": "Single Family Home",
                     "SEARCH_type[Condo/Townhouse]": "Condo/Townhouse",
                     "SEARCH_bedrooms": "0",
                     "SEARCH_baths": "0",
                     "SEARCH_minprice": "0",
                     "SEARCH_maxprice": "0",
                     "SEARCH_lotsize": "0",
                     "SEARCH_rooms": "0",
                     "SEARCH_sqft": "0",
                     "SEARCH_units": "0",
                     "SEARCH_maxyear_built": "0",
                     "SEARCH_minyear_built": "0"
                    }
```

Listing 1: Parameters used for initial search.

After the initial search was submitted, the first results page was given as the response to the web POST event. The first result address was manually copied into a variable to pass as the first target for the code that looped through the website, using the next_listing link embedded on each page. The code below was used to scrape the sales record pages.

```
breakNext = False
i = 0
start_time = time.time()

```

```
while True:
    page = session.get(targetURL, headers=headers)
    soup = BeautifulSoup(page.content, 'html.parser')

    if len(soup.findAll('a', {'class': 'next'})) == 0:
        breakNext = True

    targetURL = (baseURL + soup.findAll('a', {'class': 'next'})[0]['href'])
    features = soup.findAll('li', {'class': 'listing-feature'})

    if len(soup.findAll('a', {'class': 'listing-street-address'})):
        data.loc[i, 'StreetAddress'] = \
            soup.findAll('a', {'class': 'listing-street-address'})[0].text

    if len(soup.findAll('a', {'class': 'listing-address-city'})):
        data.loc[i, 'City'] = \
            soup.findAll('a', {'class': 'listing-address-city'})[0].text

    if len(soup.findAll('span', {'class': 'listing-mls-number'})):
        data.loc[i, 'MLS'] = \
            soup.findAll('span',
                         {'class': 'listing-mls-number'})[0].text.split("#")[1]

    for feature in features:
        data.loc[i, feature.text.split(':')[0]] = feature.text.split(':')[1]

    if i % 100 == 0:
        data.to_csv('housingData.csv')
        elapsed_time = time.time() - start_time
        print("Iteration: ", i,
              "100 Interval Duration: ",
              elapsed_time, " Data: ",
              data.shape)
        start_time = time.time()

    i+=1
    if breakNext:
        break
```

Listing 2: Code used to scrape MLS site.

All records returned from the search were scraped, for a total of 10,263 records. Each record contained 116 columns. Not all of the columns were explicitly defined. Each result page contained a section of list items that were loaded into an array. Each list item consisted of a key string, a colon, and the value for that key. This data was treated like key-value pair data, and the column in the pandas dataframe that each record was being saved to was specified by the key in the list item. The advantage of this method is that it allowed new columns to be created as the website introduced new features or details on particular pages. The resulting columns, with the first two records, are displayed in the table below.

Table 1

Sample data and structure from the MLS website data scrape.

	0	1
StreetAddress	2321 Sophia Drive	180 Courtyards E
MLS	21918929	21913013
Property Type	Single Family Home	Condo/Townhouse
Sub Type	Single Family Residence	Condo/Coop
Listing Status	Sold	Sold
Listing Price	$535,000	$325,000
County/Area	Sonoma County	Sonoma County
Zip Code	95403	95492
Year Built	2006	1987
Sq.Ft.	1,919 sq ft	980 sq ft
Style	Contemporary	NaN
Stories	2 Story	2 Story
Total Rooms	8	4
Bedrooms	4	2
Full Baths	3	2
Garage	2-Car	NaN
Fireplace	Yes, 1 Fireplace, Gas Burning	Yes, 1 Fireplace
Lot Size	4356 sq ft lot	871 sq ft lot
Baths	3	2
Area	Santa Rosa-Northwest	Windsor
#Off Street Spaces	1	NaN
Attach/Detach Home	Attached	Attached
Construct/Condition	Completed	Completed
Construction Type	Wood Frame	Wood Frame
Exterior	Wood Siding	Wood Siding
Heat/Cool	Central Air, Central Heat	Fireplace(s), Wall Furnace
Laundry/Appliance	220 V, Gas, Hookups only	220 V, Hookups only
Lot Description	Level	Level
Lot Measurement	Acres	Acres
Lot Size [SqFt]	4217	1080
Roof	Composition	Composition
Sale/Lease-Rent	Sale	Sale
Sewer/Septic	Sewer Public	Sewer Public
Square Footage Source	Realist Public Rec	Realist Public Rec
Status Date	2019-08-23	2019-08-16
Utilities	PG&E	Cable TV Available, Elec. Water Heater, Elect...
Water Source	Water Public	Water District

Year Built Source	Realist Public Rec	Realist Public Rec
Sold Date	2019-08-23 00:00:00	2019-08-16 00:00:00
Sold Price	540000	330000
School District	NaN	Windsor Unified
Pool or Spa	NaN	Yes, Comm Facility
Bath Type	NaN	Shower and Tub
Comm/Rec	NaN	Greenbelt, Pool
Common Int Dev	NaN	Yes
Dining Room	NaN	LR/DR Combo
Drive/Sidewalk	NaN	Paved Drive
Fee Includes	NaN	External Bldg. Maint, Garbage, Grounds Maint,...
Fencing	NaN	Wood Board
Floors	NaN	Laminate
HOA	NaN	Yes
HOA Amount	NaN	370
HOA Name	NaN	Courtyards East
HOA Paid	NaN	Monthly
Kitchen	NaN	220 V Wiring, Electric Range Incl., Refrigera...
Living Room	NaN	Cathedral Ceiling, Fireplace(s)
Location of Unit	NaN	End Unit
Main Level	NaN	Bath(s), Bedroom(s), Dining Room, Living Room...
Planned Unit Develop	NaN	Yes
Upper Level	NaN	Bath(s), Master Suite(s)
Yard/Grounds	NaN	Landscaped- Rear
Elementary School	NaN	NaN
Middle School	NaN	NaN
High School	NaN	NaN
Half Baths	NaN	NaN
Foundation	NaN	NaN
Garage/Parking	NaN	NaN
Miscellaneous	NaN	NaN
Pub Transportation	NaN	NaN
Spa/HotTub Y/N	NaN	NaN
Other Rooms	NaN	NaN
Senior	NaN	NaN
Upgrade New Con Only	NaN	NaN
Family Room	NaN	NaN
Other Structures	NaN	NaN
Lower Level	NaN	NaN
2nd Unit Bdrms	NaN	NaN
2nd Unit Full Baths	NaN	NaN
2nd Unit Kitchen	NaN	NaN
2nd Unit Occupied	NaN	NaN
2nd Unit Rents for	NaN	NaN
2nd Unit Type	NaN	NaN
2nd Unit on Lot	NaN	NaN
View	NaN	NaN
Energy Conservation	NaN	NaN
Irrigation	NaN	NaN
Drainage	NaN	NaN
Accessibility	NaN	NaN
New Construction	NaN	NaN
Restrictions	NaN	NaN
Zoning	NaN	NaN
Soil	NaN	NaN
Subdivision/Neighborhood	NaN	NaN
Waterfront	NaN	NaN
Height Limit	NaN	NaN
Home Protection Plan	NaN	NaN
2nd Unit Approx SqFt	NaN	NaN
Furnished	NaN	NaN

Builder/Architect	NaN	NaN
Model Name	NaN	NaN
Total Units in Subd	NaN	NaN
Well GPM	NaN	NaN
Units	NaN	NaN
# Different Models	NaN	NaN
Green Rated by Whom?	NaN	NaN
Green Rating	NaN	NaN
Year Rated Green	NaN	NaN
Fireplace Details	NaN	NaN
View Details	NaN	NaN
Location	NaN	NaN
Subdiv Developer	NaN	NaN
Unit/Blk/Lot	NaN	NaN
Well Depth	NaN	NaN
Open House	NaN	NaN
2nd Unit Half Baths	NaN	NaN
Pool Details	NaN	NaN

Next the RedFin data was downloaded from the RedFin datacenter page. The data was restricted to the County of Sonoma, and the data was seasonally adjusted using the X-12-Arima method. The seasonality adjustment is used to smooth data over periodic seasonal swings and reduce noise in the data (Findley & Martin, 2006). The data was downloaded from RedFin as a comma-separated values file (CSV). The table below shows the columns and first three records of the RedFin dataset.

Table 2

Sample data and structure from the RedFin dataset.

	0	1	2
AvgSaleToList	0.979693	0.983069	0.981798
AvgSaleToListMom	0.7%	0.3%	-0.1%
AvgSaleToListYoy	0.8%	0.8%	0.3%
HomesSold	401	397	535
HomesSoldMom	-17.1%	-1.0%	34.8%
HomesSoldYoy	25.7%	2.6%	7.2%
Inventory	1313	1310	1261
InventoryMom	-4.6%	-0.2%	-3.7%
InventoryYoy	-26.0%	-27.2%	-33.0%
SeasonallyAdjusted	False	False	False
MedianDom	57	50	50
MedianDomMom	10	-7	0
MedianDomYoy	-2	-13	0
MedianSalePrice	$300K	$299K	$300K
MedianSalePriceMom	2.7%	-0.3%	0.3%
MedianSalePriceYoy	0.0%	-0.3%	3.4%
NewListings	524	590	628

NewListingsMom	53.2%	12.6%	6.4%
NewListingsYoy	-11.3%	-5.3%	-6.0%
PeriodEnd	2012-01-31 00:00:00	2012-02-29 00:00:00	2012-03-31 00:00:00
PropertyType	All Residential	All Residential	All Residential
Region	Sonoma County, CA	Sonoma County, CA	Sonoma County, CA
RegionType	County	County	County
StateCode	CA	CA	CA

Challenges

Scraping the MLS website imposed several challenges. The first being that the search results were session based, so the same session had to be maintained throughout the process of scraping the website. In addition, response time of the website fluctuated considerably throughout the day, and the website had a crawl-delay in their robots.txt file. The process took several hours, and crashed twice due to timeouts. However, the data was successfully collected, and was the only source found that contained 3 years of historical house sales records that allowed web scraping.

C. Data Extraction and Preparation

Jupyter Notebook was used to perform the data extraction and preparation. There were many columns in the MLS data that contained a large number of empty cells, which will require analysis and cleaning. The MLS data also contained many categorical variables that were actually a list of comma-delineated properties in various combinations. This lead to some columns having over a thousand unique categories.

First the data was loaded into the new Jupyter Notebook. The Sold Date column was defined as a date field when the data file was loaded into a dataframe. To make the columns easier to work with, spaces, forward slashes, pound signs, and periods were removed from the columns. Leading spaces were almost removed from the column names. The leading spaces were a result of the web scraping method. Several columns were identified as being critical to either the analysis, or ensuring the integrity of the data. Any records that were missing values in these columns were removed, and a list of columns that contained more than 70% missing values was created. The code for this is displayed below.

```
original_data = pd.read_csv('housingData.csv',
                            index_col=0, parse_dates=['Sold_Date'])
original_data.columns = original_data.columns.str.replace(\
                        ' ', '').str.replace(\
                        '/','').str.replace(\
                        '.','').str.replace(\
                        '#','')

original_data = original_data[~(original_data.ListingStatus.isnull()) &
                              ~(original_data.ZipCode.isnull()) &
                              ~(original_data.PropertyType.isnull()) &
                              ~(original_data.SqFt.isnull())]

original_data.Senior.replace({'Yes':1, np.nan: 0}, inplace=True)
original_data.SpaHotTubYN.replace({'Yes':1, np.nan: 0}, inplace=True)
original_data.HOA.replace({'Yes':1, np.nan: 0}, inplace=True)
original_data['ZipCode'] = original_data.ZipCode.astype(int)

original_data['ListingPrice'] = original_data.ListingPrice.replace(\
                                    '[\$,]', '', regex=True).astype(float)
original_data['SoldPrice'] = original_data.SoldPrice.replace(\
                                    '[\$,]', '', regex=True).astype(float)
original_data['SqFt'] = original_data.SqFt.replace(\
                                    '[sqft.,]', '', regex=True).astype(int)
original_data['Garage'] = original_data.Garage.replace(\
                                    '[\-Car]', '', regex=True).astype(float)
original_data['OffStreetSpaces'] = original_data.OffStreetSpaces.replace(\
                                    '[\+]', '', regex=True).astype(float)

null_counts = original_data.isna().mean()
null_counts = list(null_counts[null_counts > 0.7].index)
```

Listing 3: Code used to clean and prepare the data.

Fifty-nine columns (listing 5) were identified as containing more than 70% missing values. Of the original fifty-nine columns, five were identified as being worth saving, as they would inherently be rare occurrences. One of the fields retained was the LocationofUnit field, which would only apply to condos and town homes, which account for only 15% of the data. The columns were removed (listing 4), and a table displaying the new data structure is presented below.

```python
print(len(null_counts))
print(null_counts)
keep_list = list(['SchoolDistrict', 'PoolorSpa',
                  'CommRec', 'LocationofUnit', 'EnergyConservation'])
null_counts = list(set(null_counts) - set(keep_list))
original_data = original_data.drop(null_counts, axis=1)
original_data.head()
```

Listing 4: Code used to remove the columns with a high number of null values.

```
['SchoolDistrict', 'PoolorSpa', 'CommRec', 'CommonIntDev', 'FeeIncludes',
 'LocationofUnit', 'PlannedUnitDevelop', 'ElementarySchool', 'MiddleSchool',
 'HighSchool', 'Miscellaneous', 'PubTransportation', 'UpgradeNewConOnly',
 'FamilyRoom', 'LowerLevel', '2ndUnitBdrms', '2ndUnitFullBaths', '2ndUnitKitchen',
 '2ndUnitOccupied', '2ndUnitRentsfor', '2ndUnitType', '2ndUnitonLot',
 'EnergyConservation', 'Irrigation', 'Drainage', 'Accessibility', 'NewConstruction',
 'Restrictions', 'Zoning', 'Soil', 'SubdivisionNeighborhood', 'Waterfront',
 'HeightLimit', 'HomeProtectionPlan', '2ndUnitApproxSqFt', 'Furnished',
 'BuilderArchitect', 'ModelName', 'TotalUnitsinSubd', 'WellGPM', 'Units',
 'DifferentModels', 'GreenRatedbyWhom?', 'GreenRating', 'YearRatedGreen',
 'FireplaceDetails', 'ViewDetails', 'Location', 'SubdivDeveloper', 'UnitBlkLot',
 'WellDepth', 'OpenHouse', '2ndUnitHalfBaths', 'PoolDetails']
```

Listing 5: The list of columns to be removed.

Table 3

Data structure after the high-null columns were removed.

	0	1
StreetAddress	2321 Sophia Drive	180 Courtyards E
MLS	21918929	21913013
PropertyType	Single Family Home	Condo/Townhouse
SubType	Single Family Residence	Condo/Coop
ListingStatus	Sold	Sold
ListingPrice	535000	325000
CountyArea	Sonoma County	Sonoma County
ZipCode	95403	95492
YearBuilt	2006	1987
SqFt	1919	980
Style	Contemporary	NaN
Stories	2 Story	2 Story
TotalRooms	8	4
Bedrooms	4	2

FullBaths	3	2
Garage	2	NaN
Fireplace	Yes, 1 Fireplace, Gas Burning	Yes, 1 Fireplace
LotSize	4356 sq ft lot	871 sq ft lot
Baths	3	2
Area	Santa Rosa-Northwest	Windsor
OffStreetSpaces	1	NaN
AttachDetachHome	Attached	Attached
ConstructCondition	Completed	Completed
ConstructionType	Wood Frame	Wood Frame
Exterior	Wood Siding	Wood Siding
HeatCool	Central Air, Central Heat	Fireplace(s), Wall Furnace
LaundryAppliance	220 V, Gas, Hookups only	220 V, Hookups only
LotDescription	Level	Level
LotMeasurement	Acres	Acres
LotSize[SqFt]	4217	1080
Roof	Composition	Composition
SaleLease-Rent	Sale	Sale
SewerSeptic	Sewer Public	Sewer Public
SquareFootageSource	Realist Public Rec	Realist Public Rec
StatusDate	2019-08-23	2019-08-16
Utilities	PG&E	Cable TV Available, Elec. Water Heater, Electr...
WaterSource	Water Public	Water District
YearBuiltSource	Realist Public Rec	Realist Public Rec
SoldDate	2019-08-23 00:00:00	2019-08-16 00:00:00
SoldPrice	540000	330000
SchoolDistrict	NaN	Windsor Unified
PoolorSpa	NaN	Yes, Comm Facility
BathType	NaN	Shower and Tub
CommRec	NaN	Greenbelt, Pool
DiningRoom	NaN	LR/DR Combo
DriveSidewalk	NaN	Paved Drive
Fencing	NaN	Wood Board
Floors	NaN	Laminate
HOA	0	1
HOAAmount	NaN	370
HOAName	NaN	Courtyards East
HOAPaid	NaN	Monthly
Kitchen	NaN	220 V Wiring, Electric Range Incl., Refrigerat...
LivingRoom	NaN	Cathedral Ceiling, Fireplace(s)
LocationofUnit	NaN	End Unit
MainLevel	NaN	Bath(s), Bedroom(s), Dining Room, Living Room,...
UpperLevel	NaN	Bath(s), Master Suite(s)
YardGrounds	NaN	Landscaped- Rear
HalfBaths	NaN	NaN
Foundation	NaN	NaN
GarageParking	NaN	NaN
SpaHotTubYN	0	0
OtherRooms	NaN	NaN
Senior	0	0
OtherStructures	NaN	NaN
View	NaN	NaN
EnergyConservation	NaN	NaN

Next the RedFin data was imported into the Jupyter Notebook session. There were several columns that stored large dollar amounts as a string with a leading dollar sign, and rounding up to the nearest thousands with a K suffix to indicate one thousand. These columns were converted to float data types, and any column containing a percent sign was also converted from a string to the actual floating point value. The data was then combined with the MLS dataset, matching on the previous month value from the RedFin data. Columns not needed for the analysis or predictive model were removed. Any potentially influential records for the SalePrice were removed as well. There were 122 outliers identified that fell outside of three standard deviations, see equation 1, of the SalePrice.

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{1}$$

```
redfin_data = pd.read_csv('data_data.csv', parse_dates=['Period End'])
redfin_data.columns = redfin_data.columns.str.replace(' ', '')

string_cols = redfin_data.select_dtypes(include='O')
percent = []
for column in string_cols.columns:
    if string_cols[column].str.contains("%").any():
        percent+=[column]
percent

for percol in percent:
    redfin_data[percol] = redfin_data[percol].str.strip('%').astype('float')/100

redfin_data['MedianSalePrice'] = redfin_data[ \
                                'MedianSalePrice'].str.replace(\
                                '[\$K]', '', regex=True).astype('float') * 1000

combined_data = pd.merge(original_data.assign(grouper=original_data[\
                         'SoldDate'].dt.to_period('M')),
                         redfin_data.assign(grouper=redfin_data[\
                         'PeriodEnd'].dt.to_period('M')+1),
                         how='left', on='grouper')

combined_data = combined_data.drop(['PeriodEnd', 'grouper', 'StateCode',
                                    'SeasonallyAdjusted', 'StatusDate',
                                    'SaleLease-Rent', 'StreetAddress',
                                    'MLS', 'SubType', 'HOAName', 'RegionType',
                                    'Region', 'PropertyType_y', 'ListingStatus',
                                    'CountyArea', 'YearBuiltSource',
                                          'SquareFootageSource',
                                          'LotMeasurement'], axis=1)
combined_data.drop('LotSize', axis=1, inplace=True)
combined_data.rename(columns={'PropertyType_x':'PropertyType',
                                   '#OffStreetSpaces': 'OffStreetSpaces',
                                   'LotSize[SqFt]':'LotSize'}, inplace=True)

cleaned_data = combined_data[stats.zscore(combined_data.SoldPrice) <= 3]
```

Listing 6: Code to load RedFin data, clean it and merge it with MLS dataset.

D. Analysis

Exploratory Data Analysis

With the two datasets combined, the data cleaned, and the unnecessary columns removed, the analysis phase could begin. Within the data, a fire occurred in Sonoma County in 2017 that wiped out roughly 5% of the housing market (Nauslar, Abatzoglou, & Marsh, 2018). This could lead to greater variability in the data. To determine how significant of an impact this could be, an Analysis of Variance (ANOVA) test was performed on the SalesPrice from before and after the fire incident. Before this can be performed, the assumptions of ANOVA must be validated. First, the distribution of the SoldPrice column is tested. Figures 1 and 2 show the quantile of quantile (QQ) plots to visually review the distributions of the data.

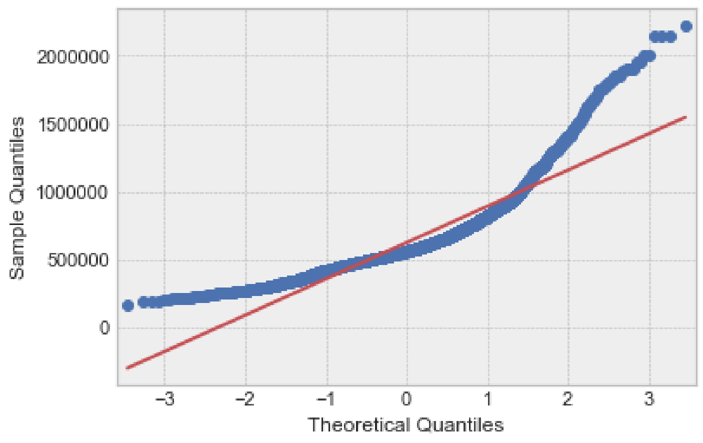

Figure 1. Quantile of Quantiles graph for SoldPrice from before the fires.

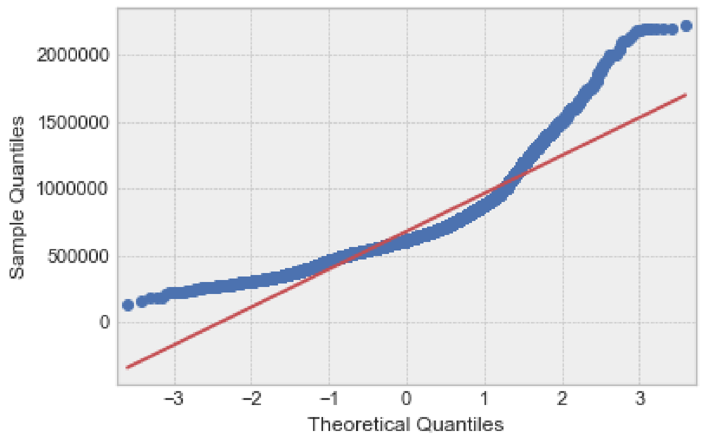

Figure 2. Quantile of Quantiles graph for SoldPrice from after the fires.

After reviewing the QQ plots, it is clear they do not follow a normal distribution. To validate this, a Shapiro-Wilks test of normality, equation 2, was performed as well. This test calculates a W statistic, which is a measure of the goodness of fit to the normal distribution. The Shapiro-Wilks test has the null hypothesis that the data is normally distributed (Mendes & Pala, 2003). With an alpha of 0.05, the test on both distributions is performed.

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{2}$$

The p-value from both tests, equations 3 and 4, are well below the alpha level of 0.05, which means the null hypothesis is rejected in favor of the alternative hypothesis. The alternative hypothesis of the Shapiro-Wilks test is that distributions are not normal. This means a non-parametric test must be used to compare the two sample groups. The non-parametric Kruskal-Wallis test will be used. The Kruskal-Wallis test can be used to test the distribution across two or more populations when the data is not normally distributed (Kruskal & Wallis, 1952). The null hypothesis for the Kruskal-Wallis test is that the population medians are equal. The formula for the test is depicted in equation 5.

$$Before : W = 0.847, p - value = 0.00000 \tag{3}$$

$$After : W = 0.851, p - value = 0.00000 \tag{4}$$

$$H = (N-1)\frac{\sum_{i=1}^{g} n_i(\bar{r}_{i\cdot} - \bar{r})^2}{\sum_{i=1}^{g}\sum_{j=1}^{n_i}(r_{ij} - \bar{r})^2} \tag{5}$$

where:

n_i = number of observations for group i

r_{ij} = group-wide rank of observation j from group i

N = total number of observations

$\bar{r}_i$ = mean rank of group i

$\bar{r}$ = mean of all r_{ij}

The results from the Kruskal-Wallis test are shown in equation 6 The p-value is less than the alpha of 0.05, so the null hypothesis is rejected in favor of the alternative hypothesis that there is a statistically significant difference in the medians of the two populations. The graph in figure 3 visually depicts the distribution of house prices between the market from before the 2017 fires, and after. The difference in the graph appears minor, but as the tests indicated, there is a statistically significant difference between these two samples.

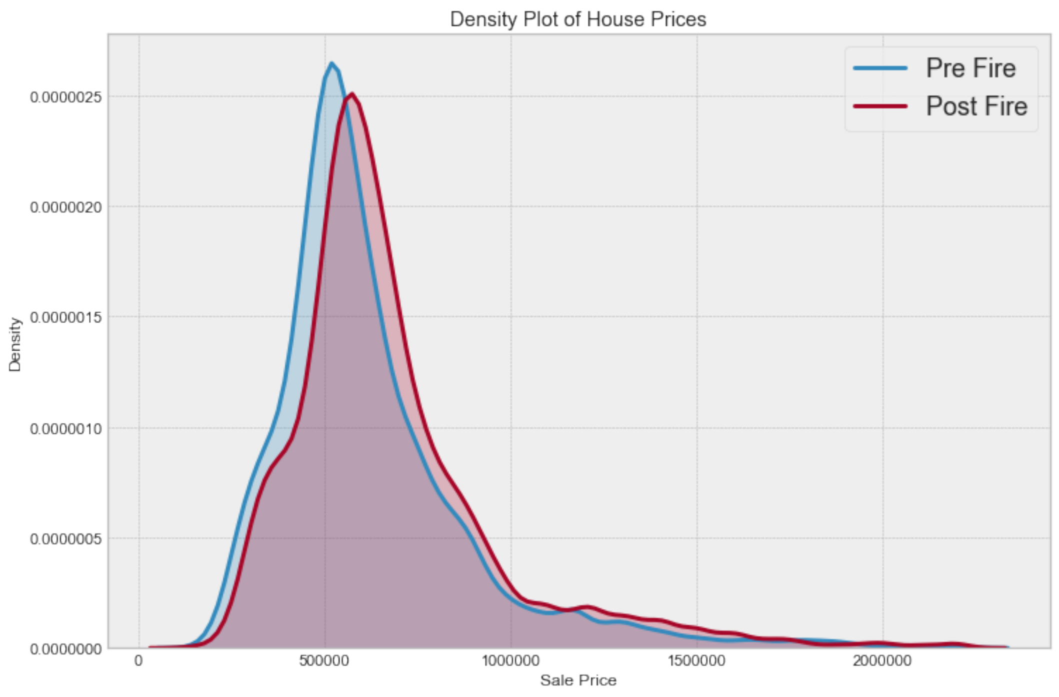

Figure 3. Density graph of the distribution for house prices from before and after the fire.

$$H = 154.230, \; p - value = 0.00000 \tag{6}$$

The difference between the list price and sale price, or listing error, was calculated for each sale record. The average listing error for each month across the three year time-span of the data was graphed, with two black vertical lines indicating the start end end of the fires in figure 4. There appears to be an increasing trend in houses selling for above listing before the fires, with a brief dip the month before the fires. The RedFin data detailing the previous month median sale prices and inventory levels may help balance the increased variance around this time period.

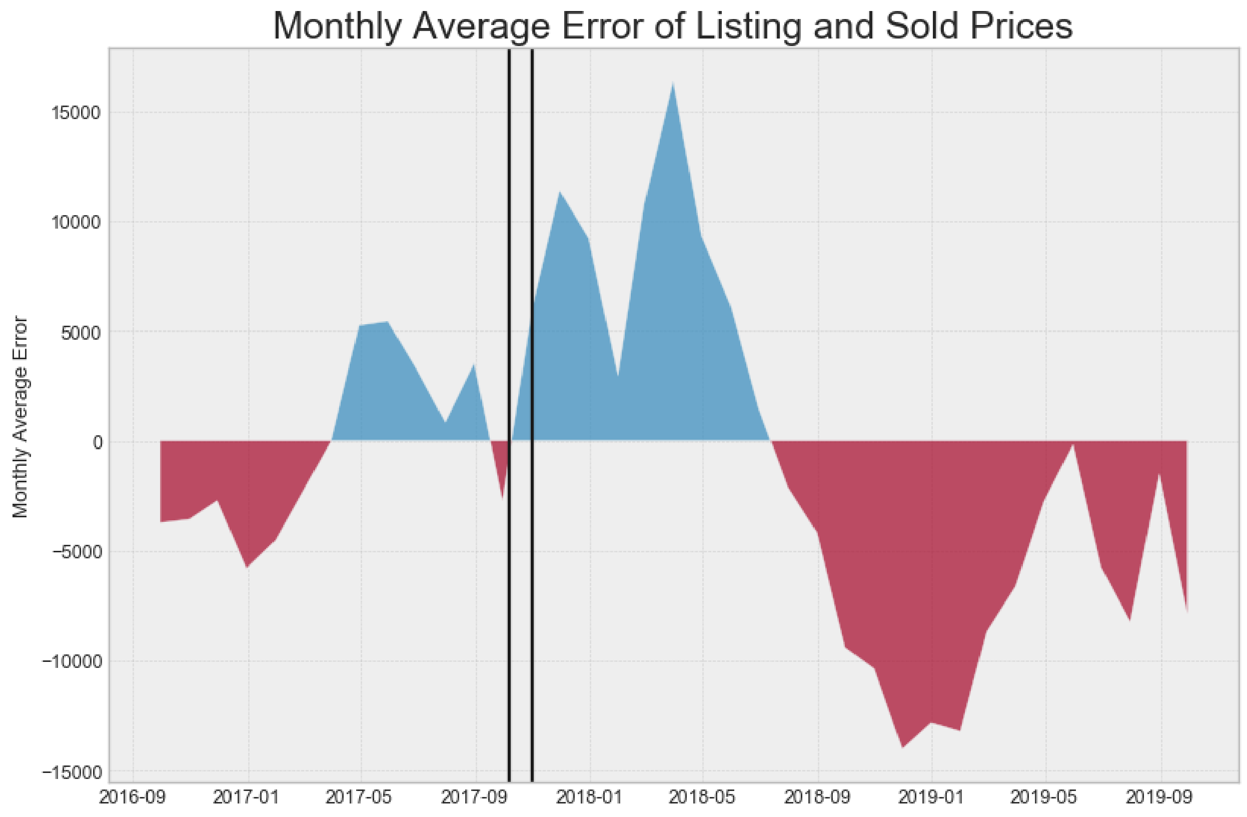

Figure 4. Listing error graph depicting average monthly listing errors.

Categorical Encoders

The next step in the analysis is comparing the five different methods of encoding high cardinality categorical variables. The dataset contains thirty-four columns that will have these five different methods applied. First the details for each method will be described. The encoding methods that will be tested include M-Measure Encoding, James Stein Target Encoding, Multiple Correspondence Analysis, K-Mode Clustering, and CatBoost Encoding.

M-Measure Encoding is a Bayesian target encoding method that is simplistic and includes additive smoothing (Micci-Barreca, 2001). This method requires a hyperparameter, a setting in which you must manually determine, of m. The hyperparameter m controls the amount of regularization that is imposed on the computed encoded values. For this study, the m value was set to 10, the most commonly seen value in research. The smoothing is to accommodate rare categories, preventing them from being too influential (Micci-Barreca, 2001). The equation for the M-Measure encoding method is depicted in equation 7, where μ is the encoded mean, n is the number of values, $\bar{x}$ is the estimated mean, m is the weight, and w is the overall mean. The code for this encoding method was manually written, refer to listing 9.

$$\mu = \frac{n \times \bar{x} + m \times w}{n + m} \tag{7}$$

James Stein Target Encoding is another Bayesian target encoder. Named after Charles Stein and Willard James, this method shrinks the estimations for each category class towards the central average (Hausser & Strimmer, 2009). The equation for this method is depicted in equation 8, where Bi is the estimated value for a group, pi is the groups proportion, pall is the total population, and B is the weight of the population mean.

$$\mathrm{B}i = (1 - B)p_i + Bp_{\mathrm{all}} \tag{8}$$

Multiple Correspondence Analysis (MCA) is a dimensionality reduction method that is similar to Principal Component Analysis (PCA), but for qualitative variables instead of quantitative. It works by performing generalized singular value decomposition (SVD) Greenacre of the data, column weights, and row weights (Nenadic & Greenacre, 2005). In the PRINCE package for Python, this is accomplished by converting the categorical variables to one-hot encoded values (Halford, 2019). The optimal number of components needs to be determined. To attempt to determine this, given the high dimensionality of the categorical columns, MCA was calculated for 1,000 components. Figure 5 is a scree plot for the amount of explained variance for each principal component. Figure 6 plots the cumulative explained variance. Based on the results, even after 1,000 components the scree plot had not leveled out. The cumulative explained variance plot shows that even with 1,000 components, only 35% variance is accounted for. Instead, 3 components will be calculated for individual categorical column as a standard baseline. The same will be done for K-Mode clustering, as the amount of variance in the data does not allow for clustering that many columns together at once.

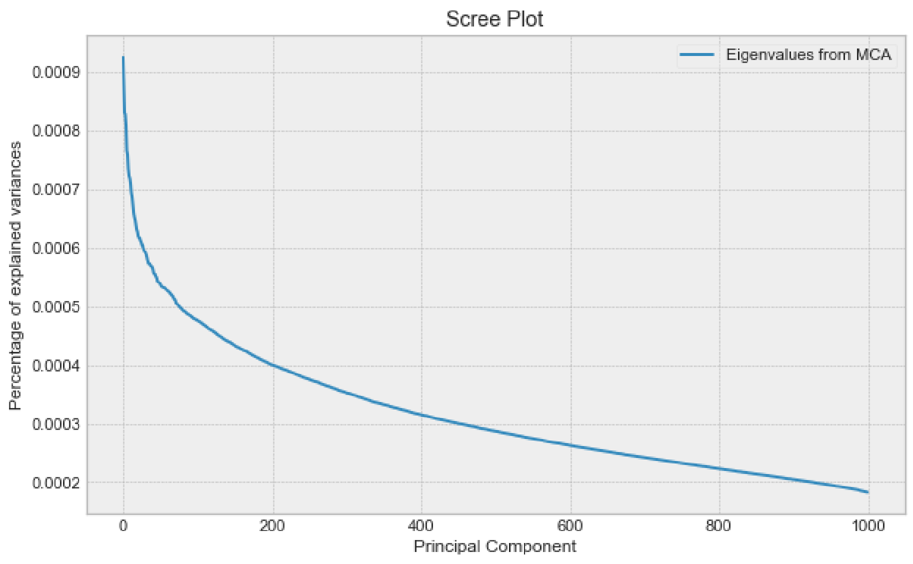

Figure 5. Scree plot of MCA components.

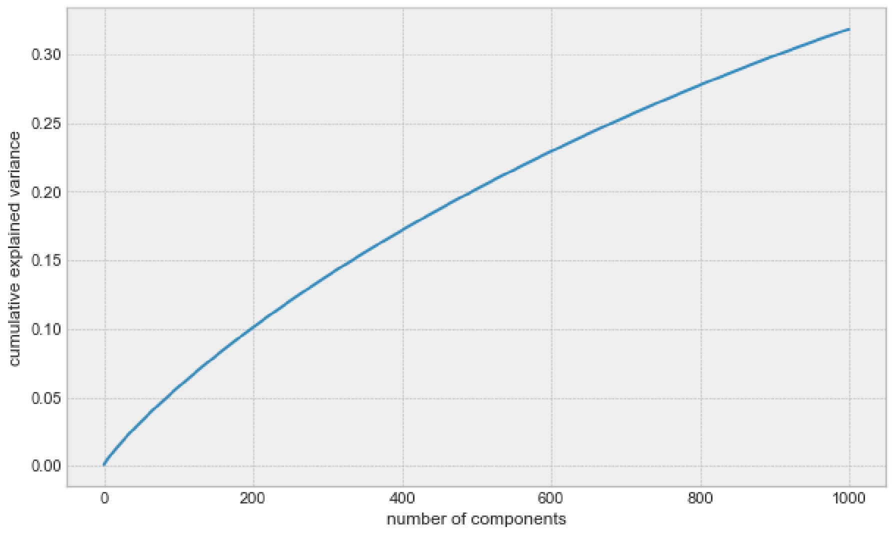

Figure 6. The cumulative explained variance for each component from MCA.

K-Mode clustering is the categorical equivalent of the numerical clustering method K-Means clustering. Equation 9 demonstrates the dissimilarity matching measure, the Hamming Distance (Huang, 1998). Equation 10 represents the cost function which K-Modes attempts to minimize (Huang, 1998). Here $w_{i,k} \in \{0, 1\}$ and $\sum_{k=1}^{K} w_{i,k} = 1 \forall i = 1 \cdots N$. For testing K-Mode clustering, each categorical variable was clustered into 3 different clusters.

$$Dis(x, y) \equiv d_H(x, y) = \sum_{j=1}^{M} (1 - \delta(x_j, y_j)) \tag{9}$$

where:

x_j	$= jth$ feature of x
$\delta(x_j, y_j)$	$= 1$ if $x_j = y_j$ or $\delta(x_j, yj) = 0$
M	$=$ number of categorical attributes

$$min(W, Z) = \sum_{k=1}^{K} \sum_{i=1}^{N} w_{ik} d_c(Z_i, X_k) \tag{10}$$

where:

W	$=$ memberships matrix
Z	$=$ centroid matrix
d_c	$=$ dissimilarity measurement
n	$=$ number of strings
c	$=$ number of target clusters

Last, a baseline method will be included, where the high dimensionality categorical variables will be excluded from the data entirely, and run against the predictive models. This can be used to establish a baseline of the accuracy for the models without the categorical variables, and help provide a baseline for how much variance can be explained for the house prices without that data.

Predictive Models

To thoroughly test the different categorical encoding methods, five different predictive models were chosen. GardientBoost, CatBoost, XGBoost, RandomForest, and Multiple Linear Regression. To account for variance in the train and test data distributions, each model and method pairing will be run for a total of twenty times. The mean performance metrics will be reviewed.

GradientBoost is a gradient boosted decision tree ensemble. It works by building an ensemble of weak decision tree models using functional gradient descent. Gradient Boosting seek to approximate the function $\hat{F}(x)$, using the formula depicted in equation 11, where $h_m(x)$ the class functions of the ensembled weak learners (Friedman, 2001). It accomplishes this objective by attempting to minimize the loss function by starting with a base constant function $F_0(x)$, and expands upon it with each iteration. The base learning functions are represented as $h_m \in \mathcal{H}$, depicted in equations 12 and 13. The GradientBoostingRegressor method from the sklearn Python library will be used, due to its compatibility with the other methods being used (Pedregosa et al., 2011).

$$F(x) = \sum_{m=1}^{M} \gamma_m h_m(x) \tag{11}$$

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma) \tag{12}$$

$$F_m(x) = F_{m-1}(x) + \arg\min_{h_m \in \mathcal{H}} \left[\sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h_m(x_i)) \right] \tag{13}$$

CatBoost is a predictive regression model that implements gradient boosting over decision trees (GBDT), similar to GradientBoost. It includes a method of handling categorical variables, which is tested separately and is not utilized in the model itself. This model was selected for its ability to produce state-of-the-art results without significant tuning (Prokhorenkova, Gusev, Vorobev, Dorogush, & Gulin, 2018). The CatBoostRegressor method from the Python catboost library will be used, due to its compatibility with the sklearn library (Prokhorenkova et al., 2018).

XGBoost is another gradient boosting decision tree ensembling method. It was selected for providing state-of-the-art results at little computational cost. XGBoost improves performance by parallelization of training the weak learner trees, and also uses L1 and L2 regularization (Chen & Guestrin, 2016). The XGBRegressor method from the XGBoost Python library will be used, as it is also compatible with the other libraries being utilized.

A generic random forest regression model using the RandomForestRegressor from the sklearn Python library will also be used. As will a generic multiple linear regression model, using the LinearRegression method from the sklearn Python library. These models were used for a baseline comparison due to their extensive usage.

Research Analysis

With the exploratory data analysis completed, the final stages of data preparation for training and analysis must be performed, as demonstrated in listing 7. First, the categorical columns with high dimensionality need to be identified. After this, additional unneeded columns were dropped. Columns with null values were imputed using median imputation. Two categorical columns with low dimensionality were one-hot encoded as well.

```
string_cols = cleaned_data.select_dtypes(include='O')

comma_cols = []
for column in string_cols.columns:
    if string_cols[column].str.contains(",").any():
        comma_cols+=[column]

prep_data = cleaned_data.copy()

prep_data.drop('HOAPaid', axis=1, inplace=True)
prep_data = prep_data.reset_index(drop=True)
prep_data.HOAAmount.fillna(0, inplace=True)
null_counts = len(prep_data) - prep_data.count()

target_cols = comma_cols + ['ZipCode', 'Area']
null_cols = null_counts[null_counts > 0].index
```

```python
null_cols = set(null_cols) - set(target_cols)

for column in null_cols:
    prep_data[column].fillna(prep_data[column].median(),
                              inplace=True)

propClus = prep_data.PropertyType.str.get_dummies()
propClus.reset_index(drop=True, inplace=True)
prep_data.drop('PropertyType', axis=1, inplace=True)
prep_data.reset_index(drop=True, inplace=True)
prep_data = prep_data.join(propClus)

ADHClus = prep_data.AttachDetachHome.str.get_dummies()
ADHClus.reset_index(drop=True, inplace=True)
prep_data.drop('AttachDetachHome', axis=1, inplace=True)
prep_data.reset_index(drop=True, inplace=True)
prep_data = prep_data.join(ADHClus)

prep_data.drop(['ListingPrice',
                'SoldDate',
                'ConstructCondition'], axis=1, inplace=True)
```

Listing 7: Code to prepare the data for predictive modeling and evaluation.

A function was defined to take in the various different models and perform the same performance calculations on all of them. The function takes the model, the model type, the current iteration, the categorical encoding method, and training data as input parameters. The data is then split into a training and validation set, with 80% for training, and 20% for validation.

```python
def test_model(model, model_type, i, method, independant, dependant):
    train_features, \
    test_features, \
    train_labels, \
    test_labels = train_test_split(independant, dependant, test_size = 0.2)

    model.fit(train_features, train_labels)
    predictions = model.predict(test_features)
    errors = abs(test_labels - predictions)
    rmse = np.sqrt(mean_squared_error(test_labels, predictions))
    results = pd.DataFrame({"method": method,
                                "iteration": i,
                                "model": model_type,
                                "rmse": rmse,
                                "score": model.score(test_features, test_labels),
                                "mae": np.mean(errors)
                               }, index=[0])
    print(results)
    return results
```

Listing 8: Function to test the models and return their performance metrics.

Two functions were created to handle some of the additional logic and calculations required for the M-Measure and K-Mode encoding methods. The code in listing 9 then iterates through each encoding method from an explicitly defined list, and encodes the high dimensionality categorical variables to a copy of the dataset using that encoding method. It then iterates through the five regression models twenty times, and records the scores for each one into a dataframe. During each time a new model is created, and the data is redistributed to a new training and validation sets in the test_model function. A total of 600 iterations were performed.

```
from category_encoders.cat_boost import CatBoostEncoder
from category_encoders import JamesSteinEncoder
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
import xgboost as xgb
from kmodes.kmodes import KModes
from catboost import CatBoostRegressor
import prince

def m_measure(data, category, target, weight):
    t_mean = data[target].mean()
    grouped = data[[category, target]].groupby(category)[target]
    grouped = grouped.agg(['count', 'mean'])
    c_means = grouped['mean']
    counts = grouped['count']
    smooth = (c_means * counts + t_mean * weight) / (counts + weight)
    data[category] = data[category].map(smooth)

def kmodes_cats(source_cat, categories, cluster_count=5, prefix='cluster'):
    km = KModes(n_clusters=cluster_count,
              init='Huang',
              n_init=cluster_count,
              verbose=0)
    clusters = km.fit_predict(categories)
    cluster_dict = {}
    for i in range(0,cluster_count):
        cluster_dict[str(prefix+str(i))] = source_cat[clusters == i].unique()
    clusters = pd.get_dummies(clusters, prefix=prefix)
    return cluster_dict, clusters

methods = ['MCA', 'Baseline', 'JamesStein', 'CatBoostEnc', 'M-Measure', 'KMode']
target_cols = comma_cols + ['ZipCode', 'Area']
train_hist = pd.DataFrame()
k_dicts = []
iterations = 10

for method in methods:
    iter_data = prep_data.copy()
    iter_data.reset_index(drop=True, inplace=True)
    if method == 'M-Measure':
        for col in target_cols:
            iter_data[col].fillna('Unknown', inplace=True)
            m_measure(iter_data, col, 'SoldPrice', 10)
    elif method == 'KMode':
        k_cols_dict = {}
        for col in target_cols:
            iter_data[col].fillna('Unknown', inplace=True)
            dummies = iter_data[col].astype('str').str.get_dummies(sep=',')
            clus_dist, new_cols = kmodes_cats(iter_data[col], dummies, 3, col)
            iter_data = iter_data.drop(col, axis=1)
            new_cols.reset_index(drop=True, inplace=True)
            iter_data.reset_index(drop=True, inplace=True)
            iter_data = iter_data.join(new_cols)
            k_cols_dict.update({col: clus_dist})
        k_dicts += k_cols_dict
    elif method == 'CatBoostEnc':
        for col in target_cols:
            encoder = CatBoostEncoder(handle_missing='value')
            iter_data[col] = encoder.fit_transform(iter_data[col],
                                                       iter_data.SoldPrice)
    elif method == 'JamesStein':
        for col in target_cols:
            encoder = JamesSteinEncoder(return_df=False, model='independent')
            iter_data[col] = encoder.fit_transform(iter_data[col],
                                                       iter_data.SoldPrice)
    elif method == 'MCA':
        iter_data[target_cols] = iter_data[target_cols].fillna('unknown')
        new_cols = pd.DataFrame()
        for col in target_cols:
            mca = prince.MCA(n_components=3, n_iter=10, check_input=True)
            mca = mca.fit(iter_data[[col]])
            clusters = mca.transform(iter_data[[col]])
            new_cols = pd.concat([new_cols, clusters], ignore_index=True)
        iter_data = iter_data.drop(target_cols, axis=1)
        new_cols.reset_index(drop=True, inplace=True)
        iter_data.reset_index(drop=True, inplace=True)
        iter_data = iter_data.join(new_cols)
    elif method == 'Baseline':
        iter_data.drop(target_cols, axis=1, inplace=True)

    dependant = iter_data['SoldPrice']
    independant = iter_data.drop('SoldPrice', axis=1)
    col_names = independant.columns

    sc = StandardScaler()
    independant = sc.fit_transform(independant)

    for i in range(iterations):
        lr = LinearRegression()
        results = test_model(lr, 'MLR', i, method, independant, dependant)
        train_hist = train_hist.append(results, ignore_index=True)

        cb = CatBoostRegressor(iterations=2200,
                               learning_rate=0.01,
                               depth=10,
                               eval_metric='RMSE',
                               verbose = False,
                               task_type="GPU",
                               devices='0:1')
        results = test_model(cb, 'CatBoost', i, method, independant, dependant)
        train_hist = train_hist.append(results, ignore_index=True)

        rf = RandomForestRegressor(n_estimators=1000, n_jobs = -1,
                                   max_features = "auto",
                                   max_leaf_nodes = 200)
        results = test_model(rf, 'RandomForest', i, method, independant, dependant)
        train_hist = train_hist.append(results, ignore_index=True)

        gbr = GradientBoostingRegressor(n_estimators=100,
                                        max_features = "auto",
                                        max_leaf_nodes = 200)
        results = test_model(gbr, 'GradientBoost',
                             i, method, independant, dependant)
        train_hist = train_hist.append(results, ignore_index=True)

        xgboost = xgb.XGBRegressor(learning_rate=0.01, n_estimators=2800,
                                   max_depth=8, min_child_weight=0,
                                   gamma=0, subsample=0.7,
                                   colsample_bytree=0.7,
                                   objective='reg:squarederror', nthread=-1,
                                   scale_pos_weight=1,
                                   reg_alpha=0.00006)

        results = test_model(xgboost, 'XGBoost', i, method, independant, dependant)
        train_hist = train_hist.append(results, ignore_index=True)
```

Listing 9: Code to test the six encoding methods against the five regression models.

The models are all scored based on four metrics: root mean square error (RMSE), R^2, $R^2(Adj)$, and mean absolute error (MAE). R^2, or the coefficient of determination, represents the proportion of the dependent variable's variance explained by the model, representing the goodness of fit (Cameron & Windmeijer, 1997). There is also the Adjusted R^2 metric ($R^2(Adj)$), which negatively penalizes the R^2 score based on the model complexity. Since the different encoding methods being tested also impact the complexity of the model, the $R^2(Adj)$ metric will be used in this study as well. Equation 14 is the formula for MAE, equation 15 depicts RMSE, and equations 16, 17, and 18 depict how R^2 is calculated.

MAE is calculated by the sum of the absolute difference between the y_i predicted and x_i actual values, divided by the total number of observations:

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{14}$$

RMSE is calculated as the square root of the average squared differences between the Y_i predicted and $\hat{Y}_i$ actual values of Y:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2} \tag{15}$$

Total sum of squares (TSS) is calculated as the sum of the squared differences between the dependent variable y_i, and it's mean $\bar{y}$:

$$TSS = \sum_i (y_i - \bar{y})^2 \tag{16}$$

Residual sum of squares (RSS), where y_i is the i^{th} dependant variable, x_i is the i^{th} independent variable, and $f(x_i$ represents the function to predict $\hat{y}_i$:

$$RSS = \sum_i (y_i - f_i)^2 = \sum_i e_i^2 \tag{17}$$

R^2 is calculated as 1 minus the difference between RSS and TSS:

$$R^2 \equiv 1 - \frac{RSS}{TSS} \tag{18}$$

$R^2(Adj)$ is very similar to R^2, with a slight modification to the last step of the equation that penalizes the score based on the number of input variables. Where the newly added term df_t represents degrees of freedom for the estimated population variance of the dependent variable, and df_e represents the degrees of freedom for the estimated underlying population error variance (Miles, 2014).

$$R^2 \equiv 1 - \frac{RSS/\text{df}_e}{TSS/\text{df}_t} \tag{19}$$

Upon reviewing the score results, it was clear that the multiple linear regression (MLR) model performed very poorly. There were several scores present in the results for the MLR model that were significant outliers. To ensure that the plots are readable, the results from that model were removed from the following graphs and tables. Figure 7 depicts the boxplots of the $R^2(Adj)$ score for each method, across all models other than the MLR model. Table 4 shows the average RMSE, R^2, $R^2(Adj)$, and MAE for all models tested.

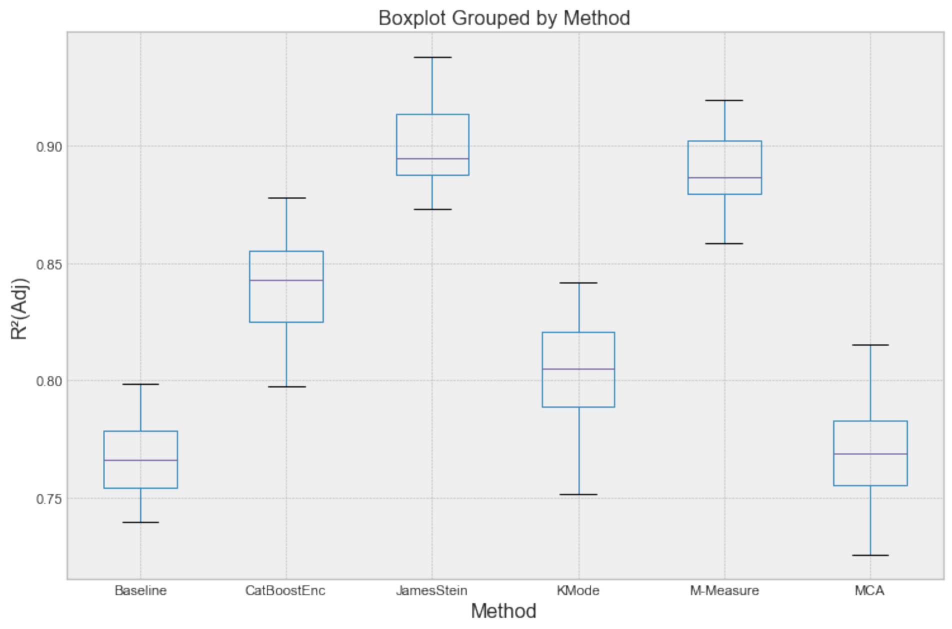

Figure 7. Boxplots of the $R^2(Adj)$ scores across all models for each encoding method.

Table 4

Mean performance metrics for each encoding method across all models, except MLR.

	RMSE	R^2	$R^2(Adj)$	MAE
Baseline	135547.813448	0.811898	0.766115	81784.721618
CatBoostEnc	118162.038530	0.867583	0.841530	68629.732551
JamesStein	86901.334525	0.912764	0.899770	49976.451239
KMode	127361.617687	0.840964	0.802525	75718.095129
MCA	135102.815988	0.815404	0.769450	81354.028821
M-Measure	90951.676758	0.904086	0.889790	53428.676875

To test the hypothesis of this paper, the assumptions of ANOVA need to be validated. The Shapiro-Wilks test was performed against the MAE values, grouped by the encoding method. The scores and p-values are display in table 5. The population p-value for both the JamesStein method and M-Measure method for the Shapiro-Wilks test for normality were less than the alpha level of 0.05. Since two of the populations do not have a normal distribution, the non-parametric Kruskal-Wallis test will be used to test this paper's hypothesis.

Table 5

Shapiro-Wilks test results of the MAE scores for all models across all encoding methods.

Method	Statistic	p-value
Baseline	0.985969	0.532834
MCA	0.974791	0.114788
JamesStein	0.955449	0.007161
CatBoostEnc	0.984403	0.440611
M-Measure	0.951929	0.004452
KMode	0.989911	0.789638

The desnity plots for the MAE scores of each encoding method are displayed in figure 8. With a p-value of < 0.05 for the Kruskal-Wallis test, equation 20, this paper's null hypothesis can be rejected in favor of the alternative hypothesis, indicating that the study factors do have an impact on house prices.

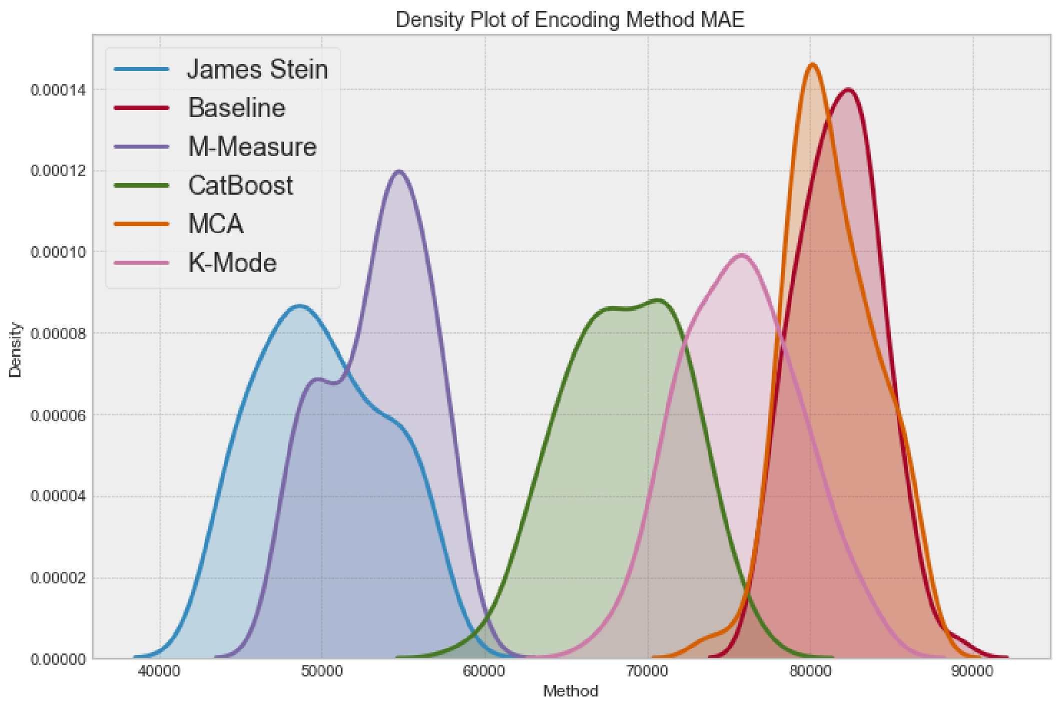

Figure 8. Density plots of the MAE scores, grouped by the encoding method.

$$H = 402.649,\ p - value = 0.00000 \tag{20}$$

An objective outcome of this paper is to produce a predictive model. Since the scale of the root mean squared error of a house price predictive model is likely to be dependent on the median house price for a region, $R^2(Adj)$ will be used primarily. To determine the best model and categorical encoding combination to use for the final predictive model, a clustered box plot was created, grouped together by the method. In the box plot in figure 9, XGBoost shows the highest peak and median $R^2(Adj)$ when combined the JamesStein encoding method. It also shows less variance than most of the other models and methods. A single predictive model will be created using JamesStein encoding and XGBoost.

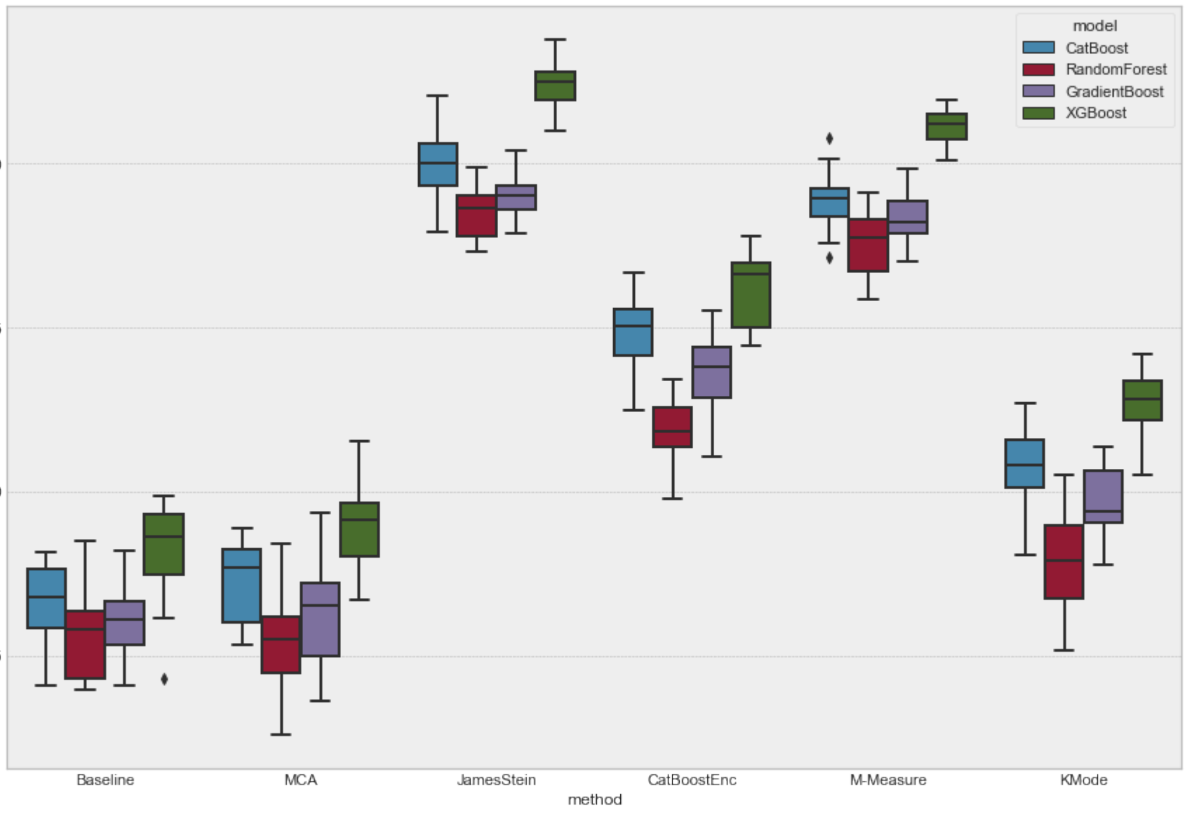

Figure 9. Clustered box plot of the $R^2(Adj)$ value of each model for each encoding method.

In order to produce the best model possible, the model needs to be fine tuned. There are various hyperparameters that can be adjusted in a wide array of combinations. To automate the process of comparing the different combinations of hyper-parameters, Bayesian optimization of hyperparameters will be utilized. Bayesian optimization of hyper parameters attempts to model the probability for the objective function and then use that model to evaluate and select the best hyperparameters (Snoek, Larochelle, & Adams, 2012). To perform this, the BayesSearchCV method from the Scikit-Optimize Python module will be used. This module was used because it supports all of the models that were tested in the analysis, and supports Bayesian hyperparameter optimization for regression models (Head et al., 2018). Listing 10 shows the code, including parameter ranges that the BayesSearchCV method explored. The optimization ran for 200 iterations, and the parameters from the best model were collected, refer to table 6.

```
from skopt import BayesSearchCV

iters = 200

bayes_optimizer = BayesSearchCV(
    xgb.XGBRegressor(
        n_jobs = 1,
        objective='reg:squarederror',
        eval_metric = 'mae',
        silent=1,
        n_estimators=2400
    ),
    search_spaces = {
        'learning_rate': (0.0001, 1.0, 'log-uniform'),
        'min_child_weight': (0.01, 10),
        'max_depth': (6, 20),
        'max_delta_step': (0.001, 20),
        'subsample': (0.1, 1.0, 'uniform'),
        'colsample_bytree': (0.1, 1.0, 'uniform'),
        'colsample_bylevel': (0.1, 1.0, 'uniform'),
        'reg_lambda': (1e-9, 100, 'log-uniform'),
        'reg_alpha': (1e-9, 1.0, 'log-uniform'),
        'gamma': (1e-9, 0.9, 'log-uniform'),
        'min_child_weight': (0.001, 5),
        'scale_pos_weight': (1e-6, 500, 'log-uniform')
    },
    scoring = 'neg_mean_absolute_error',
    n_jobs = 4,
    n_iter = iters,
    verbose = 0,
    refit = True,
)

def status_print(optim_result):
    all_models = pd.DataFrame(bayes_cv_tuner.cv_results_)

    best_params = pd.Series(bayes_cv_tuner.best_params_)
    print('Model #{}\nBest -MAE: {}\nBest params: {}\n'.format(
        len(all_models),
        np.round(bayes_cv_tuner.best_score_, 4),
        bayes_cv_tuner.best_params_
    ))
```

```

result = bayes_optimizer.fit(independant, dependant, callback=status_print)
```

Listing 10: Code used to perform the Bayesian optimization process.

Table 6

Hyperparameters selected using BayesSearchCV.

Hyperparameter	Value
learning_rate	0.007728507610359386
max_depth	10
max_delta_step	6
min_child_weight	3
reg_alpha	5.336538271717769e-05
reg_lamba	0.08231349349678034
subsample	0.7423750009164934
scale_pos_weight	0.00017127543086869703
gamma	6.889043966799518e-06
colsample_bytree	0.6781974358044769
colsample_bylevel	0.8556205847499845

A new XGBoost model was created using the parameters identified by the hyperparameter optimization. This new model was then trained with the same dataset used in the original comparison, with the categorical variables encoded using the James Stein method. The train and test datasets were split 80%-20% respectively. The performance metrics for the final model are listed in table 7. Table 8 shows the first 20 predictions from the test dataset, generated by the final model, next to the true values.

Table 7

Performance metrics for the final XGBoost model using James Stein encoding.

Method	Statistic
MAE	41,256.36
RMSE	70,273.70
$\mathbf{R^2}$	0.940022

R^2(Adj)	0.935269

Table 8

Sample records comparing final model predicted house price to the actual value.

Predicted	**Actual**	**Predicted**	**Actual**
$546,308.69	$541,335.00	$1,184,985.50	$1,140,000.00
$540,785.56	$550,000.00	$641,866.44	$599,000.00
$906,319.12	$915,000.00	$782,819.50	$808,000.00
$636,141.12	$700,000.00	$857,006.12	$840,000.00
$962,434.81	$925,000.00	$417,794.41	$420,000.00
$1,756,102.12	$1,770,000.00	$595,049.75	$650,000.00
$760,420.31	$815,000.00	$583,253.31	$569,000.00
$400,987.72	$426,000.00	$765,118.06	$772,000.00
$569,343.56	$599,000.00	$473,540.62	$497,000.00
$463,974.22	$467,500.00	$491,345.62	$490,000.00

The model acheived a Mean Absolute Error of 41,256.36. This indicates that across the 20% of the dataset the model was tested on, using data it had never seen before, it was either over or under an average absolute value of $41,256.36 from the actual price the house sold for. To determine which features were the most critical, the feautre_importances_ method from the XGBoost was run. Figure 10 contains the top 10 most important features the model used to determine the house price.

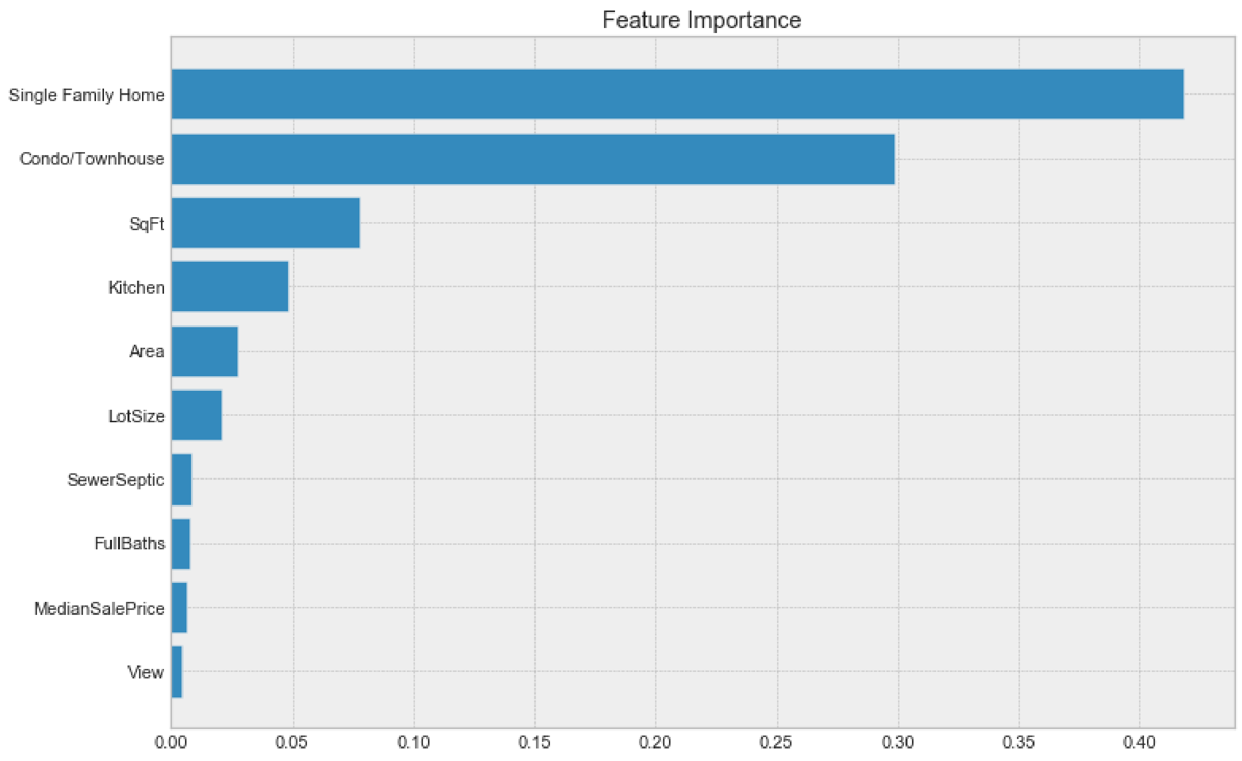

Figure 10. Top 10 most important features from the final model.

Tools Justification

The GradientBoostingRegressor, LinearRegression, RandomForestRegressor from the sklearn Python library were used due to their optimal default settings, ease of use, and compatibility with the other tools and models used during the analysis. Sklearn is the project created by Scikit-learn, and is a simple and efficient library for data mining and data analysis (Pedregosa et al., 2011). The BayesSearchCV method from the Scikit-Optimize Python library was designed to work with the sklearn library, as were the XGBoost, and catboost libraries. The kmodes method from the kmodes Python library was also selected for its inherent compatibility with sklearn, and is built on top of numpy (Vos, 2019).

Data Summary and Implications

The 2017 wildfire event identified in the data in the exploratory data analysis did prove to have an impact of the variability of the house prices. This event was a potential impact and limitation within the data. However, the previous month housing market data from the RedFin dataset, which included the MedianSalePrice feature, which represents the median house sale price for the previous month for the county the data was sourced from, was the ninth most important feature.

Comparing multiple encoding methods did impose some limitations. The CatBoost and James Stein encoding methods did not require any hyper-parameters, and M-Measure only had one hyper-parameter, which had a limited impact, but MCA and K-Mode both required a hyper-parameter that has a significant impact on the variance accounted for in the data (Nenadic & Greenacre, 2005). Tests were performed to look for an effective number of clusters for MCA, which revealed that the high cardinality in the 54 categorical variables was too severe to allow for a manageable number of clusters without losing most of the explained variance within the data. For both K-Modes and MCA, a base number of clusters was set. Future studies could explore the fine-tuning of these methods in greater depth.

The final predictive model achieved an R^2 of 0.940022, meaning that it was able to account for 94.0% of the variance within the data for the house prices. The categorical encoding methods tested for their impact on house price accuracy did show that the encoding method does have a significant impact on house price prediction accuracy. Four of the top ten most important features were categorical features that were encoded using the final selected method, James Stein, indicating that after the encoding method was implied, some of the features had an impact on the predictions the model produced. Additional encoding methods could be included in future studies. The Bayesian target encoding methods did perform the best, however there are other methods that were not included in this study.

The final model could be utilized by a real estate or listing company in the County of Sonoma to help determine the and listing house price for a new property listing, utilizing data available in MLS and RedFin. Based on the results of the study, James Stein can be an effective method compared to removing the categorical features, or using MCA or K-Modes. M-Measure tested the closest to James Stein, and could also be used as a secondary method to consider for using in house price predictions.

References

Calhoun, C. A. (2001). Property valuation methods and data in the united states. *Housing Finance International*, *16*(2), 12–23.

Cameron, A. C., & Windmeijer, F. A. (1997). An r-squared measure of goodness of fit for some common nonlinear regression models. *Journal of econometrics*, *77*(2), 329–342.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785–794).

Findley, D. F., & Martin, D. E. (2006). Frequency domain analyses of seats and x-11/12-arima seasonal adjustment filters for short and moderate-length time series. *Journal of Official Statistics-Stockholm-*, *22*(1), 1.

Friedman, J. H. (2001, 10). Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, *29*(5), 1189–1232. Retrieved from `https://doi.org/10.1214/aos/1013203451` doi: 10.1214/aos/1013203451

Halford, M. (2019). *Maxhalford/prince.* Retrieved from `https://github.com/MaxHalford/prince`

Hausser, J., & Strimmer, K. (2009). Entropy inference and the james-stein estimator, with application to nonlinear gene association networks. *Journal of Machine Learning Research*, *10*(Jul), 1469–1484.

Head, T., MechCoder, Louppe, G., Shcherbatyi, I., fcharras, Vinícius, Z., … Fabisch, A. (2018, March). *scikit-optimize/scikit-optimize: v0.5.2.* Zenodo. Retrieved from `https://doi.org/10.5281/zenodo.1207017` doi: 10.5281/zenodo.1207017

Huang, Z. (1998). Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data mining and knowledge discovery*, *2*(3), 283–304.

Kruskal, W. H., & Wallis, W. A. (1952). Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, *47*(260), 583–621.

Mendes, M., & Pala, A. (2003). Type i error rate and power of three normality tests.

Pakistan Journal of Information and Technology, *2*(2), 135–139.

Micci-Barreca, D. (2001). A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD Explorations Newsletter*, *3*(1), 27–32.

Miles, J. (2014). R squared, adjusted r squared. *Wiley StatsRef: Statistics Reference Online*.

Nauslar, N., Abatzoglou, J., & Marsh, P. (2018). The 2017 north bay and southern california fires: a case study. *Fire*, *1*(1), 18.

Nenadic, O., & Greenacre, M. (2005). Computation of multiple correspondence analysis, with code in r. *UPF Working Paper*(887).

Park, B., & Bae, J. K. (2015). Using machine learning algorithms for housing price prediction: The case of fairfax county, virginia housing data. *Expert Systems with Applications*, *42*(6), 2928-2934. doi: 10.1016/j.eswa.2014.11.040

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). Catboost: unbiased boosting with categorical features. In *Advances in neural information processing systems* (pp. 6638–6648).

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems* (pp. 2951–2959).

Vos, N. d. (2019, Jul). *nicodv/kmodes.* Retrieved from `https://github.com/nicodv/kmodes`