Real Estate Valuation & Investment

Overview

Background

The real estate market has been the subject of lots of news and political discussion in recent years.

With the introduction of vacation services like Airbnb and Vrbo, a multitude of travel sites like Kayak, TripSavvy, and TripAdvisor, and numerous home improvement shows like Fixer Upper, Property Brothers, and Flip or Flop, there has been an incredible demand for houses of all shapes, sizes, prices, and locales. From large companies like Zillow to the middle class, with so many looking to "get in on the action," the market as a whole has been very hot. Uncoincidentally, both rent and house prices have skyrocketed, well beyond the rate of inflation in many areas.

Some popular tourist cities, like Paris and Miami, have responded to these changes by writing local legislation banning or limiting short-term rentals. Other cities have proposed large increases to property taxes to try and curb the growth in short-term rentals.

In addition to regular market forces, the higher prices have also occasionally encouraged bad behavior. In Denver, CO, one Homeowners Association (HOA) has been foreclosing on homes in order to re-sell them. The public outcry resulted in new legislation aimed at protecting those living in HOA communities. With a spotlight being shone on these undesirable practices of various HOAs, some people are questioning whether they're worth it at all in a market where home prices are doing just fine.

Project Description

Most residential property transactions are generally available as part of public records. These records include a wealth of information about a property - number of bedrooms, number of bathrooms, square footage, lot acreage, year built, number of sales, sale amounts, appraisal value, asessed value, and more. In many cases, counties also provide GIS data downloads that provide supplemental data about the region - school districts, trails and recreation, airports, and homeowners associations.

The city of Denver, CO, and the immediately surrounding counties of Adams, Arapahoe, Broomfield, Douglas, and Jefferson, have seen tremendous population growth between the 2010 and 2020 U.S. Census. Sitting at #19 in both city and metropolitan area population rankings, Denver is a good candidate to study the evolution of the real estate market in recent years.

Focusing on Arapahoe County data, this project will utilize historical sales data to predict the likely `Sale Price` of a property from its publicly available property features.

Performance Metric

The main output of this project will be the predicted `Sale Price` of a property. As such, the focus will be on regression models and their output.

The **Root Mean Squared Error (RMSE)** will be used to evaluate the results, returning the resulting error across the dataset in the same format as the predictions, dollars. Viewed in this manner, the error provides an intuitive look at how far off predicted values are from the actual.

RMSE = √(Σ(ŷi – yi)2 / n)

The actual value is subtracted from the predicted value, and the difference is squared to eliminate negative values. These squared results are summed across all of the predictions and actual values then divided by the total number being evaluated in order to produce the mean. Finally, the root of the mean is taken to reduce the error back to the same scale as the predicted and actual values.

References

Property Data

- Denver Property Search
- Adams County Property Search
- Arapahoe County Property Search
- Broomfield County Property Search
- Douglas County Property Search
- Jefferson County Property Search

GIS Data

- Denver GIS Data
- Adams GIS Data
- Arapahoe GIS Data
- Broomfield GIS Data
- Douglas GIS Data
- Jefferson GIS Data

Load Libraries

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import requests
import seaborn as sns
```

Load Data

```
sample_sales_url = "https://drive.google.com/uc?export=download&id=17y-WWKfGXuocM1vdalZXn2-gBJlWvdur"
```

```python
df = pd.read_csv(sample_sales_url)

shape = df.shape
row_count = shape[0]
column_count = shape[1]
address_count = str(df['Address'].nunique())
qualified_count = str(len(df[df['Qualified'] == 'Qualified Sale']))

print(f"Row Count: {row_count}")
print(f"Column Count: {column_count}")
print(f"Unique Addresses: {address_count}")
print(f"Qualified Sales: {qualified_count}")
print("Column Counts: ")
print(df.count())

df.sample(5)
```

```
Row Count: 90640
Column Count: 23
Unique Addresses: 17692
Qualified Sales: 37814
Column Counts: 
AIN                    90640
PIN                    90640
Address                90636
Sale Date              90640
Sale Price             90640
Book Page              90640
Vacant or Improved     90640
Qualified              84349
Building               90640
Year Built             90640
Land Use               90640
Architecture           90640
Living Area            90640
Basement Area          64322
Basement Finish        41501
Acreage                90510
Appraised Value        90510
Assessed Value         90510
Bathrooms              90510
Bedrooms               90510
Fireplaces             90510
Neighborhood Code      90510
Quality Grade          90510
dtype: int64
```

	AIN	PIN	Address	Sale Date	\
58544	2077-34-1-08-031	32221127	7886 S Logan Way	7/30/1986	
12388	1975-19-2-01-035	31374090	14289 E Arizona Ave	11/24/1998	

```
3766   1975-27-1-13-009  33795253  19643 E Caspian Cir   7/19/2003
86594  2077-22-2-03-001  32105453   499 W Aberdeen Ave   9/16/1991
13973  2073-17-3-02-037  32356511  5725 S Kittredge Ct   7/24/2008

       Sale Price  Book Page Vacant or Improved  \
58544    109000.0  5011 0652           Improved
12388         0.0  A820 0442           Improved
3766          0.0  B318 8322           Improved
86594         0.0  6253 0552           Improved
13973         0.0  B808 6201           Improved

                                               Qualified  Building  \
58544                                     Qualified Sale         1
12388  Disqualified Sale. Non-arms length or non-market.         1
3766   Disqualified Sale. Non-arms length or non-market.         1
86594  Disqualified Sale. Non-arms length or non-market.         1
13973  Disqualified Sale. Non-arms length or non-market.         1

       Year Built  ... Basement Area Basement Finish  Acreage  \
58544        1980  ...         614.0             NaN    0.245
12388        1974  ...           NaN             NaN    0.021
3766         1999  ...         406.0             NaN    0.154
86594        1956  ...         731.0           674.0    0.502
13973        1984  ...        1414.0           515.0    0.256

       Appraised Value  Assessed Value  Bathrooms Bedrooms Fireplaces
\
58544          447,600          31,109        3.0      3.0        1.0

12388          226,300          15,728        2.0      3.0        1.0

3766           359,400          24,979        2.0      2.0        0.0

86594          653,700          45,433        2.0      3.0        1.0

13973          538,200          37,405        3.0      4.0        1.0


       Neighborhood Code  Quality Grade
58544             1654.0        Average
12388               91.0        Average
3766                40.0        Average
86594                9.0           Good
13973               46.0           Good

[5 rows x 23 columns]
```

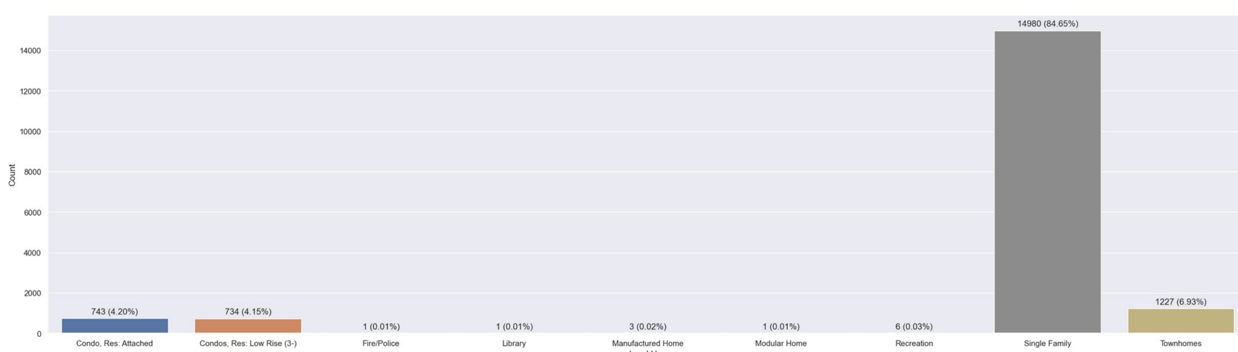
Exploratory Data Analysis (EDA)

Search for insights on the following:

- What classifications exist for `Land Use`, and what is their distribution?
- What classifications exist for `Architecture`, and what is their distribution?
- What classifications exist for `Quality Grade`, and what is their distribution?
- What classifications exist for `Qualified`, and what is their distribution?
- How has the average `Sale Price` evolved over the years?

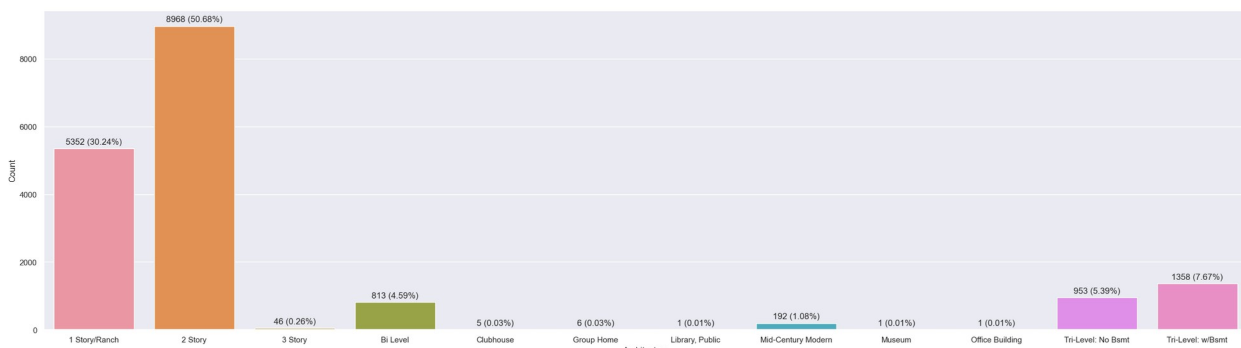
Land Use

```
# Set seaborn theme and figure size
sns.set_theme(style="whitegrid")
sns.set(rc = {'figure.figsize':(30,8)})

# Setup the figure data
# Use the unique nature of 1 AIN / parcel to count properties uniquely
seaborn_data = df[['AIN', 'Land Use']]
unique_count = seaborn_data.groupby('Land Use')['AIN'].nunique()
unique_count = unique_count.reset_index()
unique_count.columns = ['Land Use', 'Count']

# Draw a barplot
barplot = sns.barplot(x='Land Use', y='Count',  data=unique_count)

# Determine the unique count sum
unique_sum = unique_count['Count'].sum()

# Label the bars with their values
# Easier with bar_label, but requires newer versions of seaborn and
matplotlib
for g in barplot.patches:
    raw_count = format(g.get_height(), '.0f')
    percentage = format((g.get_height()/unique_sum)*100, '.2f') + '%'
    barplot.annotate(f"{raw_count} ({percentage})",
                     (g.get_x() + g.get_width() / 2., g.get_height()),
                     ha = 'center', va = 'center',
                     xytext = (0, 9),
                     textcoords = 'offset points')
```



Architecture

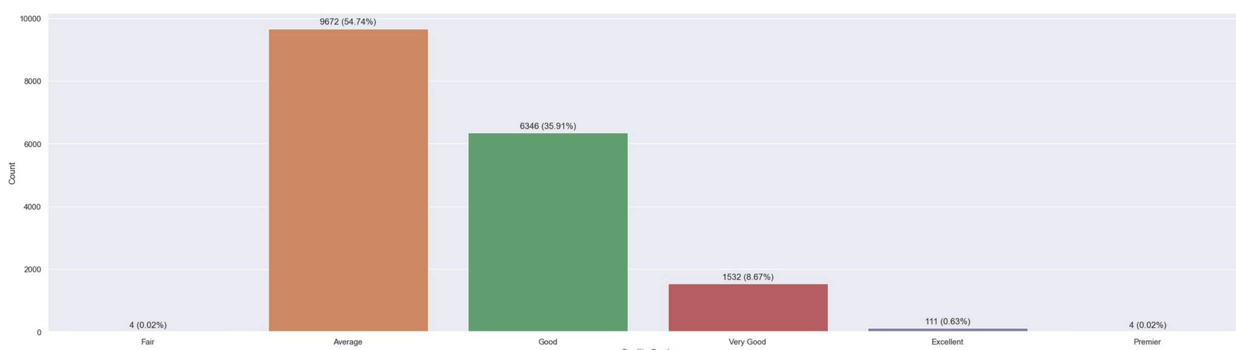
```
# Set seaborn theme and figure size
sns.set_theme(style="whitegrid")
sns.set(rc = {'figure.figsize':(30,8)})

# Setup the figure data
# Use the unique nature of 1 AIN / parcel to count properties uniquely
seaborn_data = df[['AIN', 'Architecture']]
unique_count = seaborn_data.groupby('Architecture')['AIN'].nunique()
unique_count = unique_count.reset_index()
unique_count.columns = ['Architecture', 'Count']

# Draw a barplot
barplot = sns.barplot(x='Architecture', y='Count',  data=unique_count)

# Determine the unique count sum
unique_sum = unique_count['Count'].sum()

# Label the bars with their values
# Easier with bar_label, but requires newer versions of seaborn and matplotlib
for g in barplot.patches:
    raw_count = format(g.get_height(), '.0f')
    percentage = format((g.get_height()/unique_sum)*100, '.2f') + '%'
    barplot.annotate(f"{raw_count} ({percentage})",
                    (g.get_x() + g.get_width() / 2., g.get_height()),
                    ha = 'center', va = 'center',
                    xytext = (0, 9),
                    textcoords = 'offset points')
```



Quality Grade

```
# Set seaborn theme and figure size
sns.set_theme(style="whitegrid")
sns.set(rc = {'figure.figsize':(30,8)})

# Setup the figure data
# Use the unique nature of 1 AIN / parcel to count properties uniquely
```

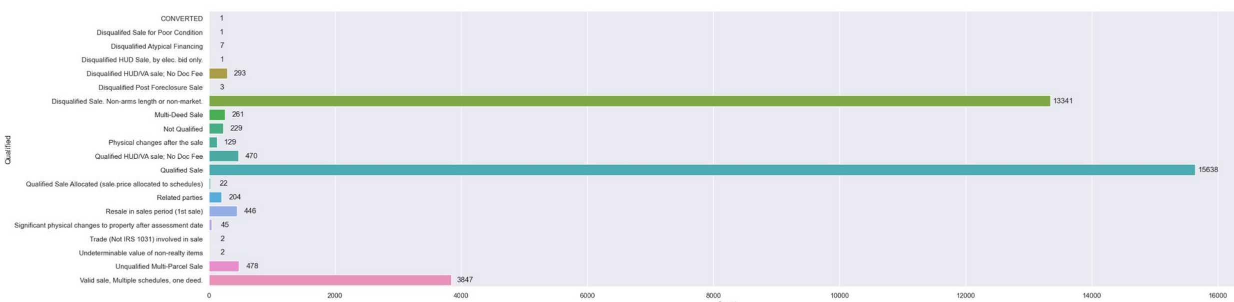
```
seaborn_data = df[['AIN', 'Quality Grade']]
unique_count = seaborn_data.groupby('Quality Grade')['AIN'].nunique()
unique_count = unique_count.reset_index()
unique_count.columns = ['Quality Grade', 'Count']

# After initial look, define explicit order to express quality from
low to high
bar_order = ['Fair', 'Average', 'Good', 'Very Good', 'Excellent',
'Premier']

# Draw a barplot
barplot = sns.barplot(x='Quality Grade', y='Count',
data=unique_count, order=bar_order)

# Determine the unique count sum
unique_sum = unique_count['Count'].sum()

# Label the bars with their values
# Easier with bar_label, but requires newer versions of seaborn and
matplotlib
for g in barplot.patches:
    raw_count = format(g.get_height(), '.0f')
    percentage = format((g.get_height()/unique_sum)*100, '.2f') + '%'
    barplot.annotate(f"{raw_count} ({percentage})",
                     (g.get_x() + g.get_width() / 2., g.get_height()),
                     ha = 'center', va = 'center',
                     xytext = (0, 9),
                     textcoords = 'offset points')
```



Qualified

```
# Set seaborn theme and figure size
sns.set_theme(style="whitegrid")
sns.set(rc = {'figure.figsize':(30,8)})

# Setup the figure data
# Use the unique nature of 1 AIN / parcel to count properties uniquely
seaborn_data = df[['AIN', 'Qualified']]
unique_count = seaborn_data.groupby('Qualified')['AIN'].nunique()
```

```python
unique_count = unique_count.reset_index()
unique_count.columns = ['Qualified', 'Count']

# Draw a barplot
# Display horizontally to make categories easier to read
barplot = sns.barplot(y='Qualified', x='Count',  data=unique_count)

# Label the bars with their values
# Easier with bar_label, but requires newer versions of seaborn and
matplotlib
# Have to manipulate the position of the labels differently for
horizontal
for g in barplot.patches:
    barplot.annotate(format(g.get_width(), '.0f'),
                       (g.get_y() + g.get_width() + 200, g.get_y() +
g.get_height()),
                       ha = 'center', va = 'center',
                       xytext = (0, 9),
                       textcoords = 'offset points')
```



Yearly Sales

```python
# Limit results to only Qualified Sale sales
yearly_sales = df[df['Qualified'] == 'Qualified Sale'].copy()

# Add a Sale Year column to the DataFrame
yearly_sales['Sale Year'] = pd.to_datetime(yearly_sales['Sale
Date']).dt.year

# Only look at Sale Year and Sale Price, ignore other features
yearly_sales = yearly_sales[['Sale Year', 'Sale Price']]
count_2022 = yearly_sales[yearly_sales['Sale Year'] == 2022].count()
print(str(count_2022))

# Eliminate sales for 0 and 1 dollar
# These are quit claim deeds, typically re-finance or bulk purchases
non_zero_yearly_sales = yearly_sales[yearly_sales['Sale Price'] > 1]

# Iterate over data year by year to calculate summary stats
sales_data = []
```

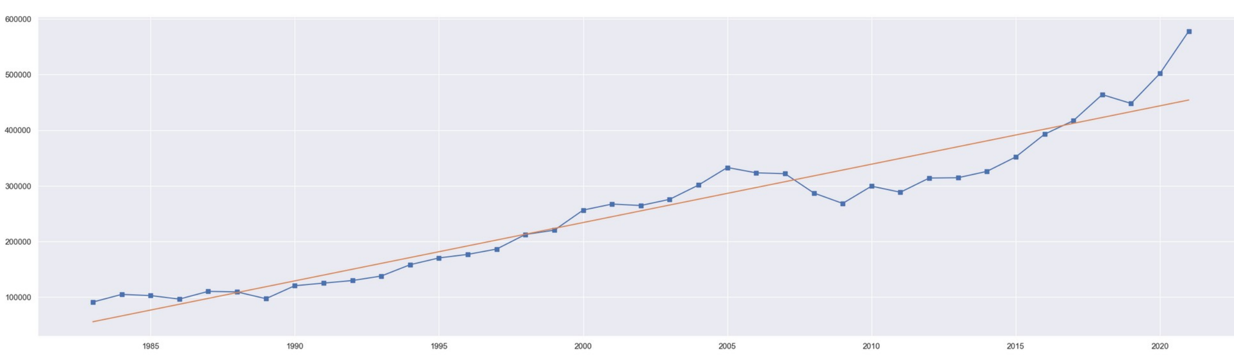
```python
for year in range(1983, 2022, 1):
  sales = non_zero_yearly_sales[non_zero_yearly_sales['Sale Year'] ==
year]

  sale_price = sales['Sale Price']
  count = sale_price.count()
  min = sale_price.min()
  max = sale_price.max()
  mean = sale_price.mean()
  median = sale_price.median()
  mode = sale_price.mode().mean()

  sales_data.append([year, count, min, max, mean, median, mode])

# Build a new report DataFrame from the summary stats
columns=['Year', 'Sales', 'Min', 'Max', 'Mean', 'Median', 'Mode']
report_data=[*sales_data]
yearly_sales_report = pd.DataFrame(report_data, columns=columns)

# Display the summary stats
yearly_sales_report.round(2)
```

```
Sale Year     569
Sale Price    569
dtype: int64
```

	Year	Sales	Min	Max	Mean	Median	Mode
0	1983	239	26000.0	290000.0	90919.25	77500.0	76000.00
1	1984	188	30000.0	475000.0	104653.72	83450.0	75000.00
2	1985	133	22500.0	515000.0	102510.53	87000.0	87133.33
3	1986	239	22000.0	354000.0	96235.56	84000.0	74500.00
4	1987	418	20900.0	449000.0	110127.99	88500.0	75000.00
5	1988	212	19000.0	329000.0	109122.17	89300.0	76666.67
6	1989	264	15000.0	383500.0	96939.39	85450.0	65800.00
7	1990	318	13700.0	349800.0	120261.79	96950.0	75000.00
8	1991	362	18000.0	563200.0	124985.55	107750.0	60000.00
9	1992	775	17600.0	750000.0	129579.07	96500.0	55000.00
10	1993	1263	14000.0	788700.0	137688.51	109000.0	78750.00
11	1994	1367	20000.0	1020000.0	158044.44	129000.0	105000.00
12	1995	1678	15000.0	1330000.0	170240.36	141250.0	118750.00
13	1996	1933	20400.0	1348800.0	176620.96	148100.0	100000.00
14	1997	1720	22500.0	1150000.0	186190.02	152350.0	132500.00
15	1998	1960	5000.0	1650000.0	212467.89	170900.0	120000.00
16	1999	1731	25000.0	1895500.0	220147.00	176500.0	140000.00
17	2000	1806	30000.0	2050000.0	256273.23	199950.0	160000.00
18	2001	1355	63500.0	3000000.0	267096.83	205000.0	200000.00
19	2002	1132	63000.0	2743800.0	264507.95	206900.0	210000.00
20	2003	1053	67000.0	3400000.0	275638.46	214900.0	185000.00
21	2004	1074	8500.0	5089000.0	301296.93	222200.0	210000.00
22	2005	1008	39000.0	3800400.0	332882.84	230000.0	215000.00

```
23  2006    927   22000.0  3250000.0  323287.70  236000.0  200000.00
24  2007    826   23000.0  3249500.0  321715.38  228200.0  185000.00
25  2008    696   28000.0  3175000.0  286980.32  199950.0  235000.00
26  2009    586   30000.0  2495000.0  268261.26  206500.0  200000.00
27  2010    582   27500.0  1925000.0  299350.69  208700.0  195000.00
28  2011    599   33000.0  3650000.0  288365.11  204100.0  230000.00
29  2012    785   21000.0  4625000.0  313910.70  227000.0  185000.00
30  2013   1080    1500.0  3010000.0  314498.84  240500.0  250000.00
31  2014   1113    9700.0  4300000.0  325877.11  270000.0  225000.00
32  2015   1048   36000.0  4150000.0  351818.89  279750.0  255000.00
33  2016   1027   35000.0  4700000.0  392680.53  310000.0  250000.00
34  2017   1027   78000.0  3182000.0  417159.90  340000.0  305000.00
35  2018    991   75000.0  4500000.0  463967.99  370000.0  325000.00
36  2019   1160  105000.0  3300000.0  448050.38  383950.0  350000.00
37  2020   1146  160000.0  3700000.0  501902.43  412563.5  370000.00
38  2021   1264   56500.0  7428000.0  577999.59  475000.0  450000.00
```

```python
# Grab Year and Mean to plot
x = yearly_sales_report['Year'].to_numpy()
y = yearly_sales_report['Mean'].to_numpy()

# Create a line to map the trend
z = np.polyfit(x, y, 1)
p = np.poly1d(z)

# Add averages to a plot
plt.plot(x, y, marker='s')

# Add the trend line to a plot
plt.plot(x, p(x))

# Show the plot
plt.show()
```



EDA Insights

```python
# Helper function to list unique values
def print_unique_column_values(column_name, df, as_type='string'):
  column_unique_values = df[column_name].astype(as_type).unique()
```

```
  column_unique_values = column_unique_values.dropna()
  for value in np.sort(column_unique_values):
    print(f"* {value}")
```

Land Use

The following classifications exist within `Land Use`:

```
print_unique_column_values('Land Use', df)
```

```
* Condo, Res: Attached
* Condos, Res: Low Rise (3-)
* Fire/Police
* Library
* Manufactured Home
* Modular Home
* Recreation
* Single Family
* Townhomes
```

The data primarily consists of `Single Family` sales, ~85%. Future feature engineering may include dropping classifications that aren't residential, like `Fire/Police`, `Library`, and `Recreation`.

Architecture

The following classifications exist within `Architecture`:

```
print_unique_column_values('Architecture', df)
```

```
* 1 Story/Ranch
* 2 Story
* 3 Story
* Bi Level
* Clubhouse
* Group Home
* Library, Public
* Mid-Century Modern
* Museum
* Office Building
* Tri-Level: No Bsmt
* Tri-Level: w/Bsmt
```

The `2 Story` and `1 Story/Ranch` make up the majority of the data, ~30% and ~51%, respectively. Similar to `Land Use`, there are several classifications that could be eliminated in future feature engineering, like `Clubhouse`, `Group Home`, `Library`, `Public`, `Museum`, and `Office Building`.

Quality Grade

The following classifications exist within Quality Grade:

```
print_unique_column_values('Quality Grade', df)
```

```
* Average
* Economy
* Excellent
* Fair
* Good
* Premier
* Very Good
```

For `Quality Grade` the majority of the properties are classified as `Average` or `Good`, with ~55% and ~36%, respectively. Unlike Land Use and Architecture, these classifications may have significant bearing on the model's final output. The hypothesis here is that a higher grade will result in a higher `Sale Price`.

Qualified

The breakdown in `Qualified` classifications shows that there are many different ways that the county assessor views types of sales. Here, it's useful to note that the unique properties possessing `Qualified` sales, 15,638, covers the majority of total unique properties, 17,692, roughly 88%. This will allow for eliminting the more nuanced circumstances surrounding other types of sales from the model fitting.

Yearly Sales

Looking at the mean `Sale Price` from 1983 until 2021, a linear trend can be seen. The trend line is not a perfect fit, as it doesn't account for what happened with the market around the 2008 financial crisis. Additionally, a sharp increase in home pricing can be seen starting in 2019 and extending through 2021. These circumstances notwithstanding, the overall trend line is a potentially promising sign for the ability of the model to successfully predict `Sale Price`.

Feature Engineering

Drops & Filters

The `AIN`, `PIN`, `Address`, and `Book Page` columns are useful identifiers to research a single property and understand its history, but they could potentially allow the model to "cheat" or overfit by just performing a direct lookup. Additionally, `AIN`, `PIN`, and `Book Page` would be unavailable to a potential buyer.

The `Appraised Value` and `Assessed Value` are unlikely to impart useful information to the model, as historical data for these values is not available, only the values for the current year.

The `Qualified` status is a real estate industry term that indicates how a sale took place. A *Qualified* sale is one that is meant to represent fair market value. A *Disqualified* or *Not Qualified* sale can be indicative of various arragements and deals where the property or properties in question are not sold at market value. This includes things like foreclosures and short sales. Since this has the potential to significantly skew the `Sale Price`, the dataset will be filtered to only *Qualified* sales prior to training and testing.

Future filtering may examine the results of EDA to narrow the classifications included from `Land Use` and `Architecture`.

Transformations

The exact date contained in `Sale Date` is too granular to mean much to the model. However, the year and the season of the sale are likely to have some bearing on the resulting price. A `Sale Year` will be extracted from `Sale Date`, and `Sale Season` will be created by examing the sale month and mapping it to *SRPING*, *SUMMER*, *FALL*, and *WINTER*. These labels will be OneHot encoded.

Final Feature List

- Acreage
- Architecture
- Basement State
- Bathrooms
- Bedrooms
- Fireplaces
- Land Use
- Living Area
- Quality Grade
- Sale Year
- Sale Season
- Year Built

Model Fitting & Evaluation

Overall Assumptions

- `Quality Grade` will have a noticeable impact on `Sale Price`
- `Year Built` will have a noticeable impact on `Sale Price`
- `Living Area` will have a noticeable impact on `Sale Price`
- `Sale Year` will have a noticeable impact on `Sale Price`
- `Fireplaces` will have a minimal impact on `Sale Price`
- Model performance will beat 70% accuracy.
- Model performance will have **RMSE** <= $75k

To anyone well-versed in real estate, it would be expected that `Quality Grade` and `Year Built` have a significant impact on a property's `Sale Price`. Similarly, the `Living Area`, or the square footage of a property, is one of the first things considered when buying.

`Sale Year` is unique to the interpretation of this dataset. Rather than only looking at current sales and treating them equally, the model is being fitted with data across several decades. The

when of a sale will have significant impact on the price, as many of the properties will have multiple sale records throughout the years.

`Fireplaces` is a bit of an odd one. The expectation here is that they won't have much of an impact, and that idea is based on them being more present than not in Colorado's colder climate.

The hope is that the model can achieve at least 70% accuracy in its predictions. This feels achievable based on early plots of the linear trend for mean `Sale Price`. Additionally, while there is a pretty wide spread in ultimate `Sale Price` within Arapahoe County, the target is to have a **RMSE** less than $75k. This takes into account the upper end of several homes in the million dollar price range as well as decades old sales with price tags of just a couple hundred thousand.

Linear Regression Baseline

Train / Test Split

With the overall data limited to `Qualified == 'Qualified Sale'`, the dataset drops to ~38k rows. Training will be performed on 85% of the data, ~32.3k rows. Testing will be run on 15% of the data, ~5.7k rows. If a "Golden Holdout" is used, the plan is to use sales from 2022 for this purpose. The count of that data is indeterminate at this time, as it's still being scraped from sources.

```python
from sklearn.model_selection import train_test_split

# Filter data before proceeding
filtered_data = df[df['Qualified'] == 'Qualified Sale']

# Drop NA across specific columns
filtered_data = filtered_data.dropna(subset=[
    'Acreage',
    'Bathrooms',
    'Bedrooms',
    'Fireplaces',
    'Quality Grade',
])

# Now that we've filtered, create the split
X_train, X_test, y_train, y_test = train_test_split(
    filtered_data.drop(columns=['Sale Price']),
    filtered_data['Sale Price'],
    test_size=0.15, random_state=42)
```

ML Pipeline

```python
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
```

```python
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder

categorical_features = [
    'Land Use',
    'Architecture',
]

drop_features = [
    'Sale Date',
    'AIN',
    'PIN',
    'Address',
    'Book Page',
    'Vacant or Improved',
    'Qualified',
    'Building',
    'Appraised Value',
    'Assessed Value',
    'Basement Area',
    'Basement Finish',
    'Neighborhood Code',
]

qualities = [['Fair', 'Average', 'Good', 'Very Good', 'Excellent',
'Premier']]

cat_pipe = Pipeline([('cat_pipe', OneHotEncoder(handle_unknown =
'ignore'))])
qulty_ord_pipe = Pipeline([('qulty_ord_pipe',
OrdinalEncoder(categories=qualities))])

preproc = ColumnTransformer([
    ('cat_transform', cat_pipe, categorical_features),
    ('qulty_transform', qulty_ord_pipe, ['Quality Grade']),
    ('drop_transform', 'drop', drop_features),
  ],
  remainder='passthrough'
)


print(X_train.iloc[0])
result = preproc.fit_transform(X_train)
print(result[0])
```

```
AIN                      2075-24-1-03-015
PIN                              33559886
Address                  6129 S Potomac Way
Sale Date                       8/13/1998
Book Page                       A813 1701
Vacant or Improved               Improved
```

```
Qualified                   Qualified Sale
Building                                 1
Year Built                            1998
Land Use                     Single Family
Architecture                       2 Story
Living Area                           3440
Basement Area                       1867.0
Basement Finish                     1309.0
Acreage                              0.277
Appraised Value                    951,800
Assessed Value                      66,150
Bathrooms                              5.0
Bedrooms                               4.0
Fireplaces                             1.0
Neighborhood Code                     13.0
Quality Grade                    Very Good
Name: 78765, dtype: object
[0.000e+00 0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00
 1.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
 3.000e+00 1.998e+03 3.440e+03 2.770e-01 5.000e+00 4.000e+00
1.000e+00]
```

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import math

# Here we build the final pipeline that combines transformations with
a model
# Try running a LinearRegression first
lr_model = LinearRegression()
lr_pipeline = Pipeline(steps=[
    ('preprocess', preproc),
    ('model', lr_model)
])

# Fit the training data to the model
lr_pipeline.fit(X_train, y_train)

# Take a look at how the model performs against the test data
print(f"Property Data:\n{X_test.iloc[0]}")
print(f"\nSale Price: {round(y_test.iloc[0],2)}")
print(f"Predicted Price: {round(lr_pipeline.predict(X_test)[0],2)}")

# See how the LinearRegression model determined coefficients
# These are not 1-to-1 with features, as OneHotEncoder adds columns
# We end up with n-columns for n-categories within a feature
print("\nCoefficients:")
print(lr_model.coef_)

# Evaluate how the model performed by examing RMSE and model score
```

```
# This is done for both the train set and the test set
rmse_train = math.sqrt(mean_squared_error(y_train,
lr_pipeline.predict(X_train)))
rmse_test = math.sqrt(mean_squared_error(y_test,
lr_pipeline.predict(X_test)))
score_train = lr_pipeline.score(X_train, y_train)
score_test = lr_pipeline.score(X_test, y_test)
print(f"\nRMSE Train: {round(rmse_train,2)}")
print(f"RMSE Test: {round(rmse_test,2)}")
print(f"Score Train: {score_train}")
print(f"Score Test: {score_test}")

# Performance isn't very good right away, so this is going to need
refinement!
```

```
Property Data:
AIN                          2075-19-4-19-006
PIN                                  33215583
Address                      6462 S Forest St
Sale Date                           9/28/1990
Book Page                           6021 0179
Vacant or Improved                   Improved
Qualified                      Qualified Sale
Building                                    1
Year Built                               1990
Land Use                        Single Family
Architecture                          2 Story
Living Area                              2295
Basement Area                           881.0
Basement Finish                         654.0
Acreage                                 0.063
Appraised Value                       456,600
Assessed Value                         31,734
Bathrooms                                 3.0
Bedrooms                                  4.0
Fireplaces                                1.0
Neighborhood Code                        83.0
Quality Grade                            Good
Name: 104, dtype: object

Sale Price: 156000.0
Predicted Price: 277702.87

Coefficients:
[ 3.37601201e+04  1.07296722e+04 -5.38894929e+04 -6.78580225e+04
  4.18607592e+04  3.53969639e+04  1.68995320e+04 -7.52335439e+04
  7.06327378e+03  1.97096886e+04 -2.93816862e+04  1.04079648e+05
  1.39341669e+01 -4.31508461e+04  3.01148640e+04  6.11320732e+02
  1.16648032e+02  9.64886042e+03  4.41561461e+04 -2.80951263e+04
  4.09545895e+04]
```

```
RMSE Train: 212508.36
RMSE Test: 243903.19
Score Train: 0.4008534442412257
Score Test: 0.42724349676860796
```

Multi-Model Grid Search Baseline

ML Pipeline

```
from sklearn import config_context
from sklearn.ensemble import GradientBoostingRegressor,
RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, GridSearchCV


# Set up modeling pipelines
lr_pipe = Pipeline(steps=[('preproc', preproc),
                       ('mdl', LinearRegression())])


rf_pipe = Pipeline(steps=[('preproc', preproc),
                       ('mdl', RandomForestRegressor())])


gb_pipe = Pipeline(steps=[('preproc', preproc),
                       ('mdl', GradientBoostingRegressor())])




# Visualize the overall pielines
with config_context(display='diagram'):
    display(lr_pipe)
    display(rf_pipe)
    display(gb_pipe)

Pipeline(steps=[('preproc',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('cat_transform',

Pipeline(steps=[('cat_pipe',

OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['Land Use',
'Architecture']),
                                                 ('qulty_transform',

Pipeline(steps=[('qulty_ord_pipe',
```

```
OrdinalEncoder(categories=[['Fair',

'Average',

'Good',

'Very '

'Good',

'Excellent',

'Premier']]))]),
                                                  ['Quality Grade']),
                                                 ('drop_transform',
'drop',
                                                  ['Sale Date', 'AIN',
'PIN',
                                                   'Address', 'Book
Page',
                                                   'Vacant or
Improved',
                                                   'Qualified',
'Building',
                                                   'Appraised Value',
                                                   'Assessed Value',
                                                   'Basement Area',
                                                   'Basement Finish',
                                                   'Neighborhood
Code'])])),
                ('mdl', LinearRegression())])

Pipeline(steps=[('preproc',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('cat_transform',

Pipeline(steps=[('cat_pipe',

OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['Land Use',
'Architecture']),
                                                 ('qulty_transform',

Pipeline(steps=[('qulty_ord_pipe',

OrdinalEncoder(categories=[['Fair',

'Average',

'Good',
```

```
'Very '

'Good',

'Excellent',

'Premier']]))]),
                                                  ['Quality Grade']),
                                                 ('drop_transform',
'drop',
                                                  ['Sale Date', 'AIN',
'PIN',
                                                   'Address', 'Book
Page',
                                                   'Vacant or
Improved',
                                                   'Qualified',
'Building',
                                                   'Appraised Value',
                                                   'Assessed Value',
                                                   'Basement Area',
                                                   'Basement Finish',
                                                   'Neighborhood
Code'])])),
                ('mdl', RandomForestRegressor())])

Pipeline(steps=[('preproc',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('cat_transform',

Pipeline(steps=[('cat_pipe',

OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['Land Use',
'Architecture']),
                                                 ('qulty_transform',

Pipeline(steps=[('qulty_ord_pipe',

OrdinalEncoder(categories=[['Fair',

'Average',

'Good',

'Very '

'Good',
```

```
'Excellent',

'Premier']]))]),
                                                            ['Quality Grade']),
                                                           ('drop_transform',
'drop',
                                                            ['Sale Date', 'AIN',
'PIN',
                                                             'Address', 'Book
Page',
                                                             'Vacant or
Improved',
                                                             'Qualified',
'Building',
                                                             'Appraised Value',
                                                             'Assessed Value',
                                                             'Basement Area',
                                                             'Basement Finish',
                                                             'Neighborhood
Code'])])),
                    ('mdl', GradientBoostingRegressor())])

# # Logistic Regression Tuning Grid
lr_tuning_grid = {}

# # Random Forest Tuning Grid
rf_tuning_grid = {'mdl__n_estimators' : [100, 200, 500],
                  'mdl__max_depth': [10, 15, 20] }

# # Gradient Boosting Tuning Grid
gb_tuning_grid = {'mdl__n_estimators' : [100, 200, 500],
                      'mdl__learning_rate' : [0.1, 0.2, 0.3]}

searches = [
    ('LinearRegression', 'lr', lr_pipe, lr_tuning_grid),
    ('RandomForestRegressor', 'rf', rf_pipe, rf_tuning_grid),
    ('GradientBoostingRegressor', 'gb', gb_pipe, gb_tuning_grid),
]

grid_searches = {}
for search in searches:
    name, prefix, pipe, tuning_grid = search
    grid_search_name = f"{prefix}_grid_search"
    grid_searches[grid_search_name] = GridSearchCV(pipe, param_grid =
tuning_grid,
                                                    cv = 5,
return_train_score=True, n_jobs=-2)

# Fit the models
for grid_search_name, grid_search in grid_searches.items():
```

```
    print(f"Running Grid Search - {grid_search_name}")
    grid_search.fit(X_train, y_train)
```

```
Running Grid Search - lr_grid_search
Running Grid Search - rf_grid_search
Running Grid Search - gb_grid_search
```

Multi-Model Evaluation

```
# Print the scores and best params for each model
for grid_search_name, grid_search in grid_searches.items():
    print(f"Grid Search - {grid_search_name}")
    print(f"Best Score - {grid_search.best_score_}")
    print(f"Best Params - {grid_search.best_params_}")
```

```
Grid Search - lr_grid_search
Best Score - 0.3982886281196901
Best Params - {}
Grid Search - rf_grid_search
Best Score - 0.37088633495060347
Best Params - {'mdl__max_depth': 10, 'mdl__n_estimators': 200}
Grid Search - gb_grid_search
Best Score - 0.4042377644473909
Best Params - {'mdl__learning_rate': 0.1, 'mdl__n_estimators': 100}
```

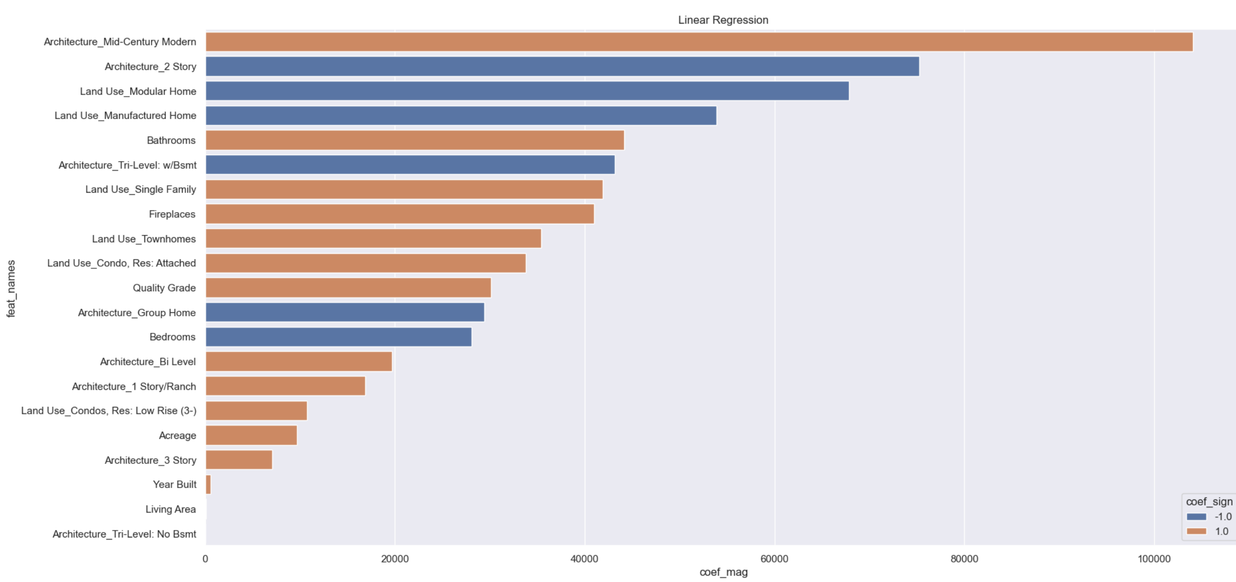
```
# Take a look at feature importances
print("\nLinear Regression Coefficients:")
lr_vip = grid_searches['lr_grid_search'].best_estimator_['mdl'].coef_
print(lr_vip)

print("\nRandom Forest Feature Importances:")
rf_vip =
grid_searches['rf_grid_search'].best_estimator_['mdl'].feature_importa
nces_
print(rf_vip)

print("\nGradient Boosting Feature Importances:")
gb_vip =
grid_searches['gb_grid_search'].best_estimator_['mdl'].feature_importa
nces_
print(gb_vip)
```

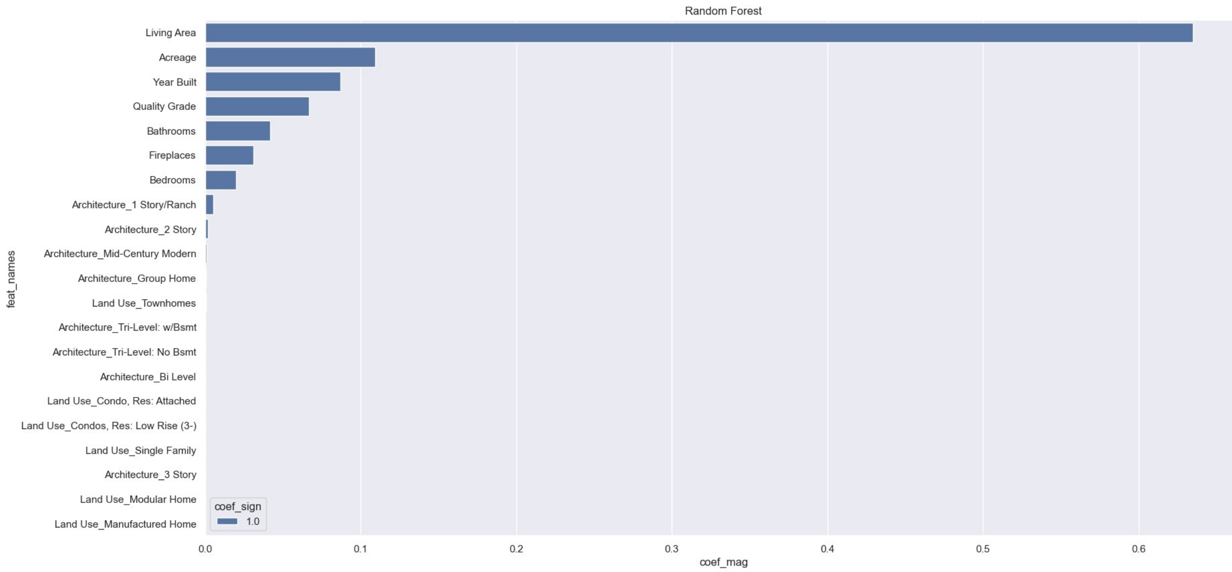
```
Linear Regression Coefficients:
[ 3.37601201e+04  1.07296722e+04 -5.38894929e+04 -6.78580225e+04
  4.18607592e+04  3.53969639e+04  1.68995320e+04 -7.52335439e+04
  7.06327378e+03  1.97096886e+04 -2.93816862e+04  1.04079648e+05
  1.39341669e+01 -4.31508461e+04  3.01148640e+04  6.11320732e+02
  1.16648032e+02  9.64886042e+03  4.41561461e+04 -2.80951263e+04
  4.09545895e+04]
```

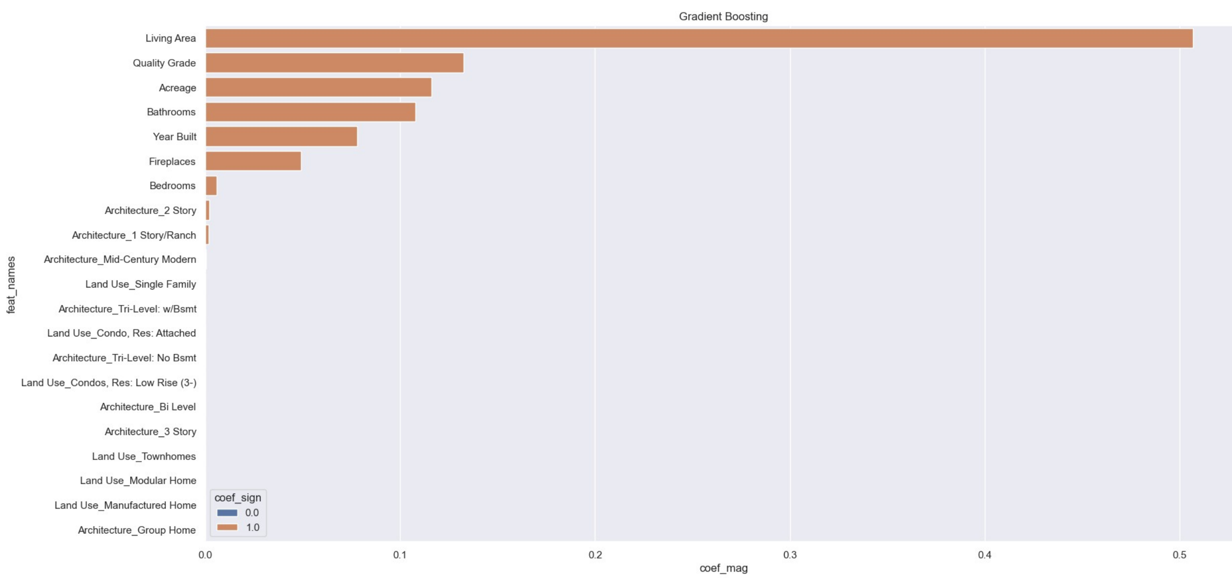
```
Random Forest Feature Importances:
[1.02625576e-04 4.36813188e-05 1.82396606e-06 6.39376736e-06
 3.58839740e-05 3.63621547e-04 5.30128669e-03 1.67738987e-03
 3.20148625e-05 1.31501919e-04 3.93364276e-04 8.29041241e-04
 1.68745635e-04 2.81627701e-04 6.66368251e-02 8.69199630e-02
 6.35003441e-01 1.09293303e-01 4.17732383e-02 1.99603351e-02
 3.10438920e-02]

Gradient Boosting Feature Importances:
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 1.00579126e-04 0.00000000e+00 1.68947160e-03 1.97983093e-03
 0.00000000e+00 0.00000000e+00 0.00000000e+00 4.18043553e-04
 0.00000000e+00 0.00000000e+00 1.32474521e-01 7.78927655e-02
 5.06698455e-01 1.15957068e-01 1.07823752e-01 5.78387968e-03
 4.91816339e-02]
```

```
# Investigate feature importance by plotting each model's results
searches = [
    ('Linear Regression', grid_searches['lr_grid_search'], lr_vip),
    ('Random Forest', grid_searches['rf_grid_search'], rf_vip),
    ('Gradient Boosting', grid_searches['gb_grid_search'], gb_vip)
]

for search in searches:
    name, grid_search, vip = search

    #get names in correct preproc order
    # This code is going to be VERY specific to how you formulate the
pipeline
    # It is NOT portable to all use cases in its current state
    cat_names = grid_search.best_estimator_.named_steps['preproc']\
              .transformers_[0]
[1].named_steps['cat_pipe'].get_feature_names_out()

    qlty_ord_names =
grid_search.best_estimator_.named_steps['preproc']\
              .transformers_[1]
[1].named_steps['qulty_ord_pipe'].get_feature_names_out()

    # bsmt_ord_names =
grid_search.best_estimator_.named_steps['preproc']\
    #           .transformers_[2]
[1].named_steps['bsmt_ord_pipe'].get_feature_names_out()


    num_columns = grid_search.best_estimator_.named_steps['preproc']\
              .transformers_[3][2]

    remainders = []
    for col in num_columns:
```

```
        remainders.append(X_train.axes[1][col])

    #create df with vip info
    coef_info = pd.DataFrame({
        'feat_names':np.hstack([cat_names, qlty_ord_names, 
remainders]),
        'vip': vip
    })

    #get sign and magnitude information
    coef_info = coef_info.assign(coef_mag = abs(coef_info['vip']),
                                 coef_sign = np.sign(coef_info['vip']))

    #sort and plot
    coef_info = coef_info.set_index('feat_names')\
                .sort_values(by='coef_mag', ascending=False)
    plt.figure(figsize = (20,10))
    sns.barplot(y=coef_info.index, x='coef_mag', hue='coef_sign',
                data=coef_info, orient='h', 
dodge=False).set(title=name)
```








```
# Investigate feature importance by plotting each model's results
searches = [
    ('Linear Regression', grid_searches['lr_grid_search']),
    ('Random Forest', grid_searches['rf_grid_search']),
    ('Gradient Boosting', grid_searches['gb_grid_search'])
]

for search in searches:
    name, grid_search = search
    best_estimator = grid_search.best_estimator_
    # Evaluate how the model performed by examing RMSE and model score
    # This is done for both the train set and the test set
    rmse_train = math.sqrt(mean_squared_error(y_train,
best_estimator.predict(X_train)))
```

```
    rmse_test = math.sqrt(mean_squared_error(y_test,
best_estimator.predict(X_test)))
    score_train = best_estimator.score(X_train, y_train)
    score_test = best_estimator.score(X_test, y_test)
    print(f"\n**** {name} Results ****")
    print(f"RMSE Train: {round(rmse_train,2)}")
    print(f"RMSE Test: {round(rmse_test,2)}")
    print(f"Score Train: {score_train}")
    print(f"Score Test: {score_test}")


**** Linear Regression Results ****
RMSE Train: 212508.36
RMSE Test: 243903.19
Score Train: 0.4008534442412257
Score Test: 0.42724349676860796

**** Random Forest Results ****
RMSE Train: 180333.08
RMSE Test: 238459.89
Score Train: 0.5685486314107351
Score Test: 0.45252321069821444

**** Gradient Boosting Results ****
RMSE Train: 193749.61
RMSE Test: 237851.53
Score Train: 0.5019617342494209
Score Test: 0.4553130613369285
```

Model Testing Summary

The baseline for the multi-model grid search shows some moderate improvement for the `RandomForestRegressor` and `GradientBoostingRegressor` over `LinearRegression`. With columns added or modified as part of feature engineering, it will be interesting to see how performance is impacted across all three models.

Multi-Model w/ Features

Train / Test Split

For features, here we add `Basement State`, `Sale Season`, and `Sale Year`

```
from sklearn.model_selection import train_test_split

def get_season(month):
  if month in [3,4,5]:
    return 'SPRING'
  elif month in [6,7,8]:
    return 'SUMMER'
  elif month in [9,10,11]:
```

```
    return 'FALL'
  elif month in [12,1,2]:
    return 'WINTER'
  else:
    return 'UNKNOWN'

def get_basement_state(series):
  a = series['Basement Area']
  f = series['Basement Finish']

  if pd.isna(a) and pd.isna(f):
    return 'NONE'
  elif a > 0 and pd.isna(f):
    return 'UNFINISHED'
  elif a > 0 and f > 0:
    return 'FINISHED'
  else:
    return 'NONE'


# Filter data before proceeding
filtered_data = df[df['Qualified'] == 'Qualified Sale']

# Drop NA across specific columns
filtered_data = filtered_data.dropna(subset=[
    'Acreage',
    'Bathrooms',
    'Bedrooms',
    'Fireplaces',
    'Quality Grade',
])

# Trying to add columns and then apply onehot to them is problematic
# We also want to add these values to the entire dataset ahead of
split
# Add a Sale Year & Sale Month column to the overall dataframe
filtered_data['Sale Year'] = pd.to_datetime(filtered_data['Sale
Date']).dt.year
filtered_data_month = pd.to_datetime(filtered_data['Sale
Date']).dt.month
# Add a Sale Season based on month
filtered_data['Sale Season'] =
pd.Categorical(filtered_data_month.apply(get_season, 0))
# Add a Basement State based on basement area and finish
filtered_data['Basement State'] =
pd.Categorical(filtered_data[['Basement Area', 'Basement Finish']]
                                      .apply(get_basement_state, 1))

# Now that we've filtered, create the split
X_train, X_test, y_train, y_test = train_test_split(
```

```
    filtered_data.drop(columns=['Sale Price']),
    filtered_data['Sale Price'],
    test_size=0.15, random_state=42)
```

Data Pipeline

```
# from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder

categorical_features = [
    'Land Use',
    'Architecture',
    'Sale Season',
]

drop_features = [
    'Sale Date',
    'AIN',
    'PIN',
    'Address',
    'Book Page',
    'Vacant or Improved',
    'Qualified',
    'Building',
    'Appraised Value',
    'Assessed Value',
    'Basement Area',
    'Basement Finish',
    'Neighborhood Code',
]

qualities = [['Fair', 'Average', 'Good', 'Very Good', 'Excellent',
'Premier']]
basements = [['NONE', 'UNFINISHED', 'FINISHED']]

cat_pipe = Pipeline([('cat_pipe', OneHotEncoder(handle_unknown =
'ignore'))])
qulty_ord_pipe = Pipeline([('qulty_ord_pipe',
OrdinalEncoder(categories=qualities))])
bsmt_ord_pipe = Pipeline([('bsmt_ord_pipe',
OrdinalEncoder(categories=basements))])

preproc = ColumnTransformer([
    ('cat_transform', cat_pipe, categorical_features),
    ('qulty_transform', qulty_ord_pipe, ['Quality Grade']),
    ('bsmt_transform', bsmt_ord_pipe, ['Basement State']),
    ('drop_transform', 'drop', drop_features),
  ],
```

```
  remainder='passthrough'
)


print(X_train.iloc[0])
result = preproc.fit_transform(X_train)
print(result[0])
```

```
AIN                         2075-24-1-03-015
PIN                                 33559886
Address                    6129 S Potomac Way
Sale Date                          8/13/1998
Book Page                          A813 1701
Vacant or Improved                  Improved
Qualified                     Qualified Sale
Building                                   1
Year Built                              1998
Land Use                       Single Family
Architecture                         2 Story
Living Area                             3440
Basement Area                         1867.0
Basement Finish                       1309.0
Acreage                                0.277
Appraised Value                      951,800
Assessed Value                        66,150
Bathrooms                                5.0
Bedrooms                                 4.0
Fireplaces                               1.0
Neighborhood Code                       13.0
Quality Grade                      Very Good
Sale Year                               1998
Sale Season                           SUMMER
Basement State                      FINISHED
Name: 78765, dtype: object
[0.000e+00 0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00
 1.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
 0.000e+00 0.000e+00 1.000e+00 0.000e+00 3.000e+00 2.000e+00 1.998e+03
 3.440e+03 2.770e-01 5.000e+00 4.000e+00 1.000e+00 1.998e+03]
```

ML Pipeline

```
from sklearn import config_context
from sklearn.ensemble import GradientBoostingRegressor,
RandomForestRegressor
from sklearn.linear_model import LinearRegression

# Set up modeling pipelines
lr_pipe = Pipeline(steps=[('preproc', preproc),
                       ('mdl', LinearRegression())])
```

```
rf_pipe = Pipeline(steps=[('preproc', preproc),
                          ('mdl', RandomForestRegressor())])


gb_pipe = Pipeline(steps=[('preproc', preproc),
                          ('mdl', GradientBoostingRegressor())])




# Visualize the overall pielines
with config_context(display='diagram'):
    display(lr_pipe)
    display(rf_pipe)
    display(gb_pipe)
```

```
Pipeline(steps=[('preproc',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('cat_transform',

Pipeline(steps=[('cat_pipe',

OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['Land Use',
'Architecture',
                                                   'Sale Season']),
                                                 ('qulty_transform',

Pipeline(steps=[('qulty_ord_pipe',

OrdinalEncoder(categories=[['Fair',

'Average',

'Good',

'Very '

'Good',

'Excellent',

'Premier'...
                                                 ('bsmt_transform',

Pipeline(steps=[('bsmt_ord_pipe',

OrdinalEncoder(categories=[['NONE',

'UNFINISHED',
```

```
'FINISHED']]))]),
                                                                                               ['Basement State']),
                                                                                              ('drop_transform',
'drop',
                                                                                               ['Sale Date', 'AIN',
'PIN',
                                                                                                'Address', 'Book
Page',
                                                                                                'Vacant or
Improved',
                                                                                                'Qualified',
'Building',
                                                                                                'Appraised Value',
                                                                                                'Assessed Value',
                                                                                                'Basement Area',
                                                                                                'Basement Finish',
                                                                                                'Neighborhood
Code'])]))),
                ('mdl', LinearRegression())])

Pipeline(steps=[('preproc',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('cat_transform',

Pipeline(steps=[('cat_pipe',

OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['Land Use',
'Architecture',
                                                   'Sale Season']),
                                                 ('qulty_transform',

Pipeline(steps=[('qulty_ord_pipe',

OrdinalEncoder(categories=[['Fair',

'Average',

'Good',

'Very '

'Good',

'Excellent',

'Premier'...

Pipeline(steps=[('bsmt_ord_pipe',
```

```
OrdinalEncoder(categories=[['NONE',

'UNFINISHED',

'FINISHED']]))]),
                                                  ['Basement State']),
                                                 ('drop_transform',
'drop',
                                                  ['Sale Date', 'AIN',
'PIN',
                                                   'Address', 'Book
Page',
                                                   'Vacant or
Improved',
                                                   'Qualified',
'Building',
                                                   'Appraised Value',
                                                   'Assessed Value',
                                                   'Basement Area',
                                                   'Basement Finish',
                                                   'Neighborhood
Code'])])),
                ('mdl', RandomForestRegressor())])

Pipeline(steps=[('preproc',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('cat_transform',

Pipeline(steps=[('cat_pipe',

OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['Land Use',
'Architecture',
                                                   'Sale Season']),
                                                 ('qulty_transform',

Pipeline(steps=[('qulty_ord_pipe',

OrdinalEncoder(categories=[['Fair',

'Average',

'Good',

'Very '

'Good',

'Excellent',
```

```
'Premier'...

Pipeline(steps=[('bsmt_ord_pipe',

OrdinalEncoder(categories=[['NONE',

'UNFINISHED',

'FINISHED']]))]),
                                                  ['Basement State']),
                                                 ('drop_transform',
'drop',
                                                  ['Sale Date', 'AIN',
'PIN',
                                                   'Address', 'Book
Page',
                                                   'Vacant or
Improved',
                                                   'Qualified',
'Building',
                                                   'Appraised Value',
                                                   'Assessed Value',
                                                   'Basement Area',
                                                   'Basement Finish',
                                                   'Neighborhood
Code'])])),
                ('mdl', GradientBoostingRegressor())])
```

```
from sklearn.model_selection import GridSearchCV

# Linear Regression Tuning Grid
lr_tuning_grid = {}

# Random Forest Tuning Grid
rf_tuning_grid = {'mdl__n_estimators' : [100, 200, 500],
               'mdl__max_depth': [10, 15, 20] }

# Gradient Boosting Tuning Grid
gb_tuning_grid = {'mdl__n_estimators' : [100, 200, 500],
                  'mdl__learning_rate' : [0.1, 0.2, 0.3]}

searches = [
    ('LinearRegression', 'lr', lr_pipe, lr_tuning_grid),
    ('RandomForestRegressor', 'rf', rf_pipe, rf_tuning_grid),
    ('GradientBoostingRegressor', 'gb', gb_pipe, gb_tuning_grid),
]

grid_searches = {}
for search in searches:
```

```
    name, prefix, pipe, tuning_grid = search
    grid_search_name = f"{prefix}_grid_search"
    grid_searches[grid_search_name] = GridSearchCV(pipe, param_grid = 
tuning_grid,
                                                     cv = 5, 
return_train_score=True, n_jobs=-2)

# Fit the models
for grid_search_name, grid_search in grid_searches.items():
    print(f"Running Grid Search - {grid_search_name}")
    grid_search.fit(X_train, y_train)
```

```
Running Grid Search - lr_grid_search
Running Grid Search - rf_grid_search
Running Grid Search - gb_grid_search
```

Multi-Model Evaluation

```
# Print the scores and best params for each model
for grid_search_name, grid_search in grid_searches.items():
    print(f"Grid Search - {grid_search_name}")
    print(f"Best Score - {grid_search.best_score_}")
    print(f"Best Params - {grid_search.best_params_}")
```

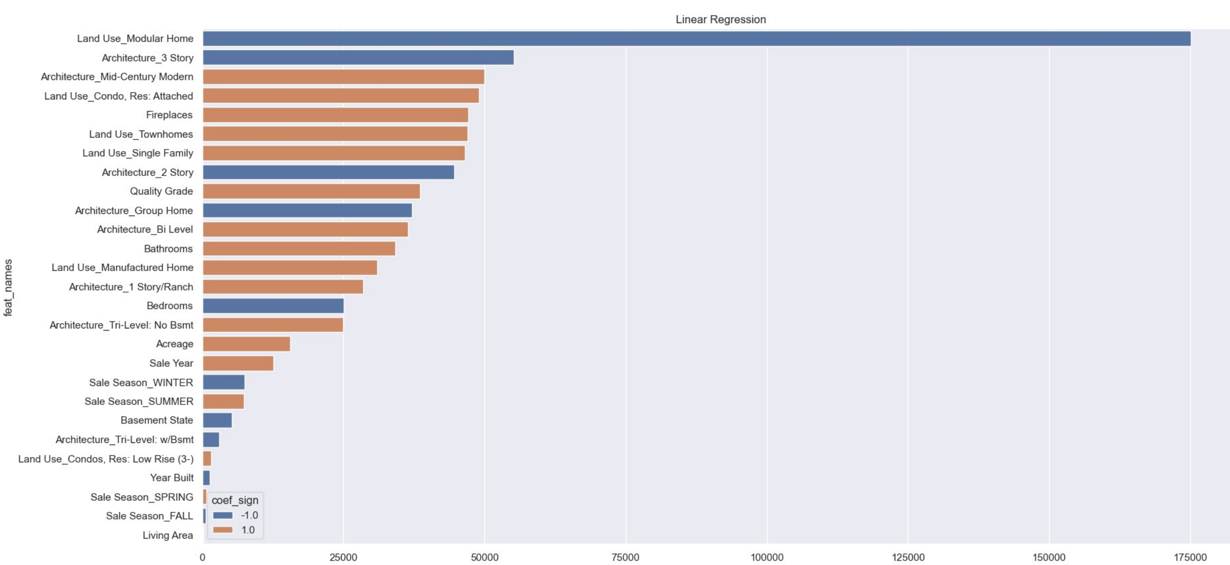
```
Grid Search - lr_grid_search
Best Score - 0.6006043818375766
Best Params - {}
Grid Search - rf_grid_search
Best Score - 0.8732815506988751
Best Params - {'mdl__max_depth': 20, 'mdl__n_estimators': 500}
Grid Search - gb_grid_search
Best Score - 0.8435163358710044
Best Params - {'mdl__learning_rate': 0.2, 'mdl__n_estimators': 500}
```

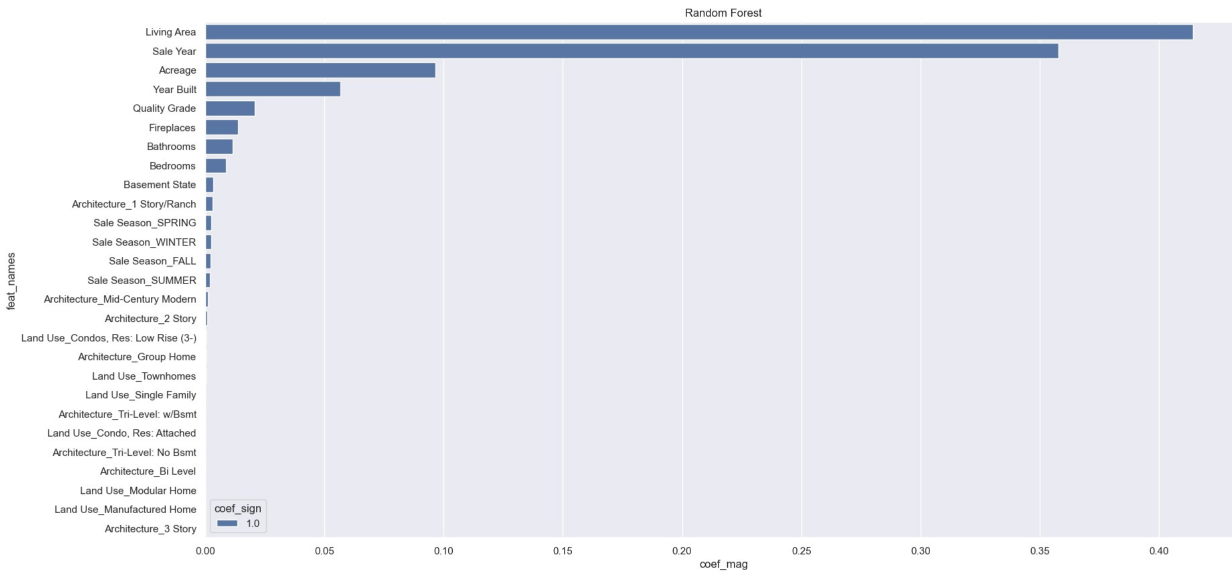
```
# Take a look at feature importances
print("\nLinear Regression Coefficients:")
lr_vip = grid_searches['lr_grid_search'].best_estimator_['mdl'].coef_
print(lr_vip)

print("\nRandom Forest Feature Importances:")
rf_vip = 
grid_searches['rf_grid_search'].best_estimator_['mdl'].feature_importa
nces_
print(rf_vip)

print("\nGradient Boosting Feature Importances:")
gb_vip = 
grid_searches['gb_grid_search'].best_estimator_['mdl'].feature_importa
nces_
print(gb_vip)
```

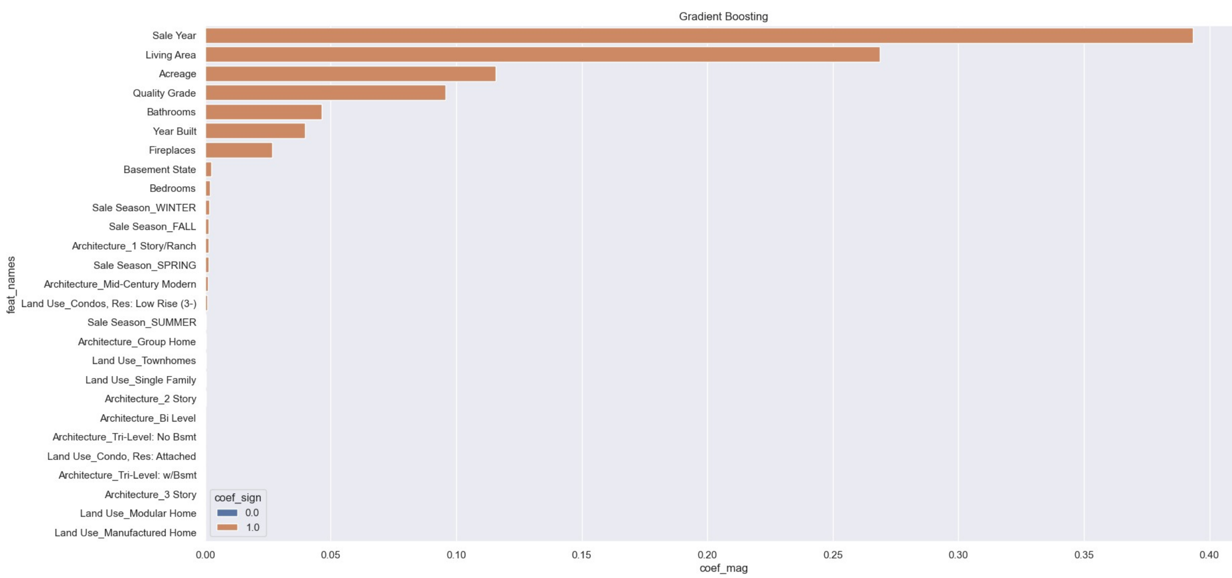
```
Linear Regression Coefficients:
[ 4.89906967e+04  1.61316182e+03  3.09635827e+04 -1.75139586e+05
  4.65381459e+04  4.70339994e+04  2.85508750e+04 -4.46489123e+04
 -5.52054381e+04  3.64958738e+04 -3.71685068e+04  5.00270226e+04
  2.49282954e+04 -2.97920959e+03 -6.37782802e+02  7.60158352e+02
  7.41633023e+03 -7.53870578e+03  3.86256921e+04 -5.20284547e+03
 -1.29934210e+03  1.32664010e+02  1.55146107e+04  3.41847630e+04
 -2.50627860e+04  4.70990524e+04  1.25428295e+04]

Random Forest Feature Importances:
[1.12913816e-04 2.87163669e-04 4.85429177e-06 5.69946881e-06
 1.62899172e-04 2.39413845e-04 3.28510814e-03 8.97106731e-04
 3.82054076e-06 3.80574834e-05 2.79905899e-04 1.10274172e-03
 8.19094722e-05 1.34895363e-04 2.37601586e-03 2.60840341e-03
 2.09981116e-03 2.56391022e-03 2.07546253e-02 3.42715598e-03
 5.68039603e-02 4.14202047e-01 9.64857462e-02 1.16254226e-02
 8.75534203e-03 1.39501029e-02 3.57710968e-01]

Gradient Boosting Feature Importances:
[5.04554993e-05 8.87835499e-04 0.00000000e+00 1.25063194e-06
 2.18064567e-04 2.23730478e-04 1.42279517e-03 1.93897819e-04
 5.35276782e-06 1.05325147e-04 2.43622487e-04 1.02947910e-03
 6.70657394e-05 1.06815441e-05 1.50470113e-03 1.35526715e-03
 4.37375176e-04 1.59531121e-03 9.58473553e-02 2.35763884e-03
 3.96374637e-02 2.68642969e-01 1.15646197e-01 4.64831361e-02
 2.02950501e-03 2.65819903e-02 3.93421534e-01]
```

```python
# Investigate feature importance by plotting each model's results
searches = [
    ('Linear Regression', grid_searches['lr_grid_search'], lr_vip),
    ('Random Forest', grid_searches['rf_grid_search'], rf_vip),
    ('Gradient Boosting', grid_searches['gb_grid_search'], gb_vip)
]

for search in searches:
    name, grid_search, vip = search

    #get names in correct preproc order
    # This code is going to be VERY specific to how you formulate the
pipeline
    # It is NOT portable to all use cases in its current state
    cat_names = grid_search.best_estimator_.named_steps['preproc']\
              .transformers_[0]
[1].named_steps['cat_pipe'].get_feature_names_out()

    qlty_ord_names =
grid_search.best_estimator_.named_steps['preproc']\
              .transformers_[1]
[1].named_steps['qulty_ord_pipe'].get_feature_names_out()
```

```
    bsmt_ord_names =
grid_search.best_estimator_.named_steps['preproc']\
              .transformers_[2]
[1].named_steps['bsmt_ord_pipe'].get_feature_names_out()


    num_columns = grid_search.best_estimator_.named_steps['preproc']\
              .transformers_[4][2]

    remainders = []
    for col in num_columns:
        remainders.append(X_train.axes[1][col])

    #create df with vip info
    coef_info = pd.DataFrame({
        'feat_names':np.hstack([cat_names, qlty_ord_names,
bsmt_ord_names, remainders]),
        'vip': vip
    })

    #get sign and magnitude information
    coef_info = coef_info.assign(coef_mag = abs(coef_info['vip']),
                                coef_sign = np.sign(coef_info['vip']))

    #sort and plot
    coef_info = coef_info.set_index('feat_names')\
                .sort_values(by='coef_mag', ascending=False)
    plt.figure(figsize = (20,10))
    sns.barplot(y=coef_info.index, x='coef_mag', hue='coef_sign',
                data=coef_info, orient='h',
dodge=False).set(title=name)
```








```python
# Investigate feature importance by plotting each model's results
searches = [
    ('Linear Regression', grid_searches['lr_grid_search']),
    ('Random Forest', grid_searches['rf_grid_search']),
    ('Gradient Boosting', grid_searches['gb_grid_search'])
]

for search in searches:
    name, grid_search = search
    best_estimator = grid_search.best_estimator_
    # Evaluate how the model performed by examing RMSE and model score
    # This is done for both the train set and the test set
    rmse_train = math.sqrt(mean_squared_error(y_train, 
best_estimator.predict(X_train)))
```

```
    rmse_test = math.sqrt(mean_squared_error(y_test,
best_estimator.predict(X_test)))
    score_train = best_estimator.score(X_train, y_train)
    score_test = best_estimator.score(X_test, y_test)
    print(f"\n**** {name} Results ****")
    print(f"RMSE Train: {round(rmse_train,2)}")
    print(f"RMSE Test: {round(rmse_test,2)}")
    print(f"Score Train: {score_train}")
    print(f"Score Test: {score_test}")
```

```
**** Linear Regression Results ****
RMSE Train: 173237.53
RMSE Test: 207373.9
Score Train: 0.6018332542559669
Score Test: 0.5859590993791106

**** Random Forest Results ****
RMSE Train: 35843.26
RMSE Test: 121719.57
Score Train: 0.9829550391204521
Score Test: 0.8573552216029929

**** Gradient Boosting Results ****
RMSE Train: 62792.09
RMSE Test: 137268.4
Score Train: 0.9476892523839205
Score Test: 0.8185837532730114
```

Model Testing Summary

Out of the 3 models tried, `LinearRegression`, `RandomForestRegressor`, and `GradientBoostingRegressor`, the `RandomForestRegressor` had the best overall performance. I expected `LinearRegression` to have the worst performance, and while that was indeed true, it was surprising to see how big the gap really was between it and the other two models, with its test model score of 58.6% versus 81.6% for `GradientBoostingRegressor` and 85.5% for `RandomForestRegressor`.