Exploring the Real Estate Market

Data Analytics - Group Project

Group 4
Aishwarya Khinvasara
Nihal Jayan
Rahul Chada
Ruchika Balani
Swetha Srinivasan

Project Goals

- Collect real estate data from different cities in SF Bay Area
- Clean the data by using data cleaning techniques
- Propose research questions on the real estate data
- Perform analyses on the data to deduce answers to those questions

Data Facts

- Realtor.com
- Cities: San Francisco, Sunnyvale, San Mateo, Hayward, Fremont, Berkeley, Dublin, Pleasanton, San Ramon, Union City, San Jose
- Average 100 properties per city
- Property Attributes - 20 - property url, state, city, street address, zipcode, bed, bath, sqft, lot size, price, property type, monthly HOA fees, price per sqft, parking space, year of construction, median home price, median selling price, median home price per sqft, school district, broker

Data Scraping

Website throws *"403 Forbidden Error"* for scraping more than 50 properties

- Scraped 50 properties at a time, city by city.
- Exported property data to one csv file.
- Changed cookies for each iteration.
- Combined each 50-property csv to get a single csv file with raw data.

```python
base_url = "https://www.realtor.com"
all_dataframes = []
for city,urls in listing_urls.items():

    listing_data = []
    school_data = []
    historic_data = []

    for i in range(0,50):
        listing_details = {}
        school_details = {}
        historic_details = {}

        url = base_url+urls[i]


        # prints a random value from the list
        list1 = [1]
        time.sleep(random.choice(list1))

        req = urllib.request.Request(url,headers=headers)
        htmlfile = urlopen(req)
        soup = BeautifulSoup(htmlfile,"html.parser")

        #Url
        listing_details["url"] = urls[i]

        #City
        listing_details["city"] = city

        #State
        listing_details["state"] = "CA"
```

Raw Data

- Shape: 1835 rows x 20 columns
- Duplicate rows: 17
- Data types: float, int, object

Column Name	Null Value Count
Street address	2
Sqft	21
Lot size	388
Price	2
Property type	22
Monthly HOA fees	1,045
Price per sqft	21
Parking space	458
Year of construction	77
Median home price	307
Median selling price	430
Median home price per sqft	307

```
-----------Shape of data--------------
(1835, 22)
-----------Datatypes for each column------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1835 entries, 0 to 1834
Data columns (total 22 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   url                              1835 non-null   object
 1   city                             1835 non-null   object
 2   state                            1835 non-null   object
 3   street-address                   1833 non-null   object
 4   zipcode                          1835 non-null   int64
 5   beds                             1835 non-null   object
 6   baths                            1835 non-null   object
 7   sqft                             1835 non-null   object
 8   lotsize                          1835 non-null   object
 9   price                            1835 non-null   object
 10  property-type                    1813 non-null   object
 11  time-on-realtor                  1833 non-null   object
 12  hoa                              780 non-null    object
 13  price/sqft                       1814 non-null   object
 14  garage                           1364 non-null   object
 15  year                             1758 non-null   float64
 16  median_listing_home_price        1528 non-null   object
 17  median_sold_home_price           1404 non-null   object
 18  median_days_on_market            1404 non-null   float64
 19  median_listing_home_price_persqft 1528 non-null  object
 20  school_district                  1835 non-null   object
 21  broker                           1835 non-null   object
dtypes: float64(2), int64(1), object(19)
memory usage: 179.3+ KB
None
```

Data Cleaning

1. Cleaned text/chars values from numerical value columns like sqft
2. Converted data type for columns with numerical values
3. Renamed column names to a single string with underscores
4. Dropped duplicate rows to get rid of redundant data
5. Dropped rows with null values in certain important columns
6. Removed outliers from numerical variables

Result - Clean Data

- Shape: 1409 rows x 20 columns
- Duplicate rows: 0
- Data types: float, int, object

Column Name	Null Value Count
Street address	0
Sqft	0
Lot size	0
Price	0
Property type	0
Monthly HOA fees	893
Price per sqft	0
Parking space	319
Year of construction	8
Median home price	209
Median selling price	316
Median home price per sqft	209

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1409 entries, 0 to 1834
Data columns (total 22 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   url                  1409 non-null   object
 1   city                 1409 non-null   object
 2   state                1409 non-null   object
 3   street_address       1409 non-null   object
 4   zipcode              1409 non-null   int64
 5   bed                  1409 non-null   float64
 6   bath                 1409 non-null   float64
 7   sqft                 1409 non-null   float64
 8   lotsize              1409 non-null   float64
 9   price                1409 non-null   float64
 10  property_type        1409 non-null   object
 11  time-on-realtor      1409 non-null   object
 12  hoa_monthly          516 non-null    float64
 13  price_per_sqft       1409 non-null   float64
 14  parking_space        1090 non-null   object
 15  year                 1401 non-null   float64
 16  median_home_price    1200 non-null   float64
 17  median_selling_price 1093 non-null   float64
 18  median_days_on_market 1093 non-null  float64
 19  mhp_per_sqft         1200 non-null   float64
 20  school_district      1409 non-null   object
 21  broker               1409 non-null   object
dtypes: float64(12), int64(1), object(9)
memory usage: 203.6+ KB
```

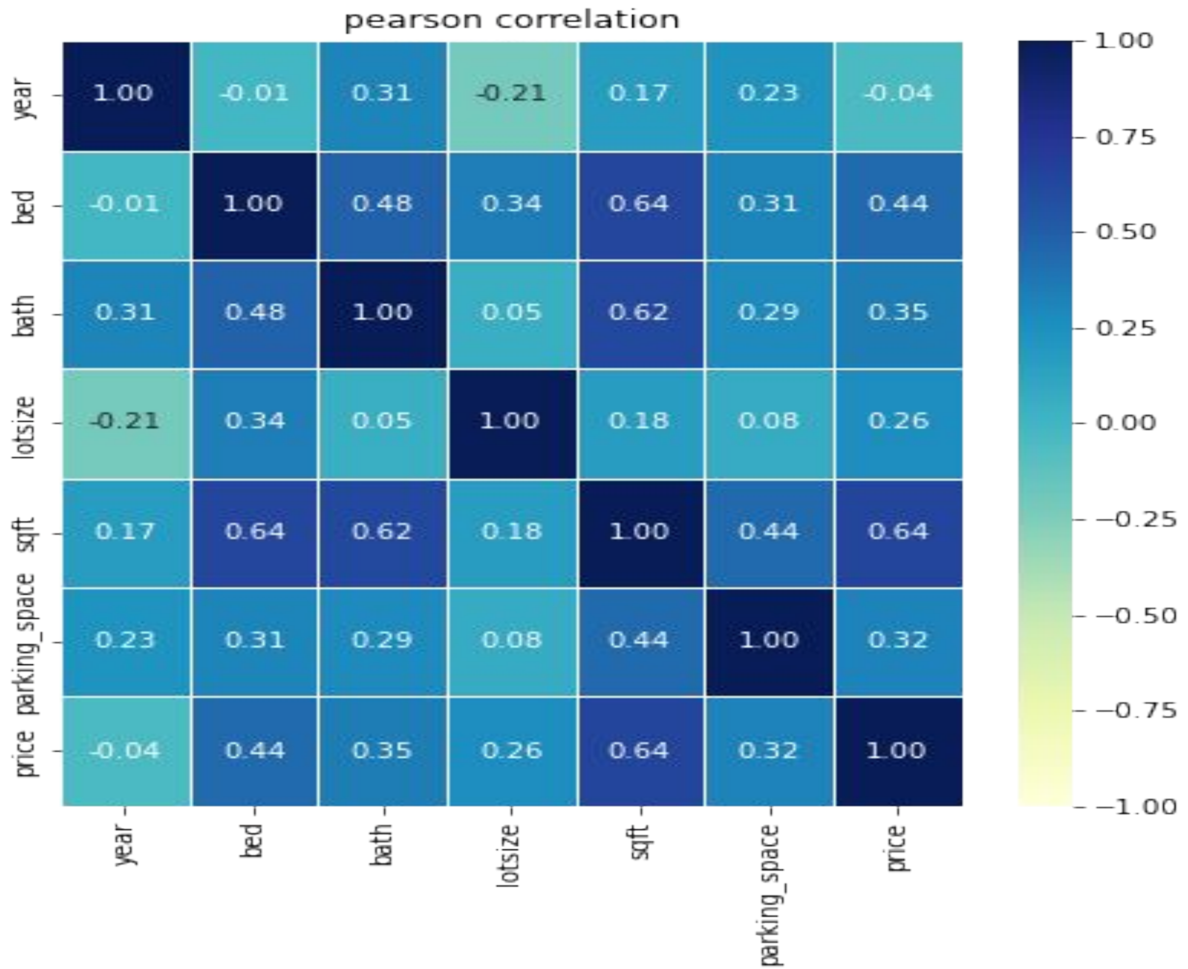
General Overview of the Analysis

Analyzing Heat Maps

1. Checking variables that influence price.
2. Making decisions for next steps of data preparation and analysis.

What we found

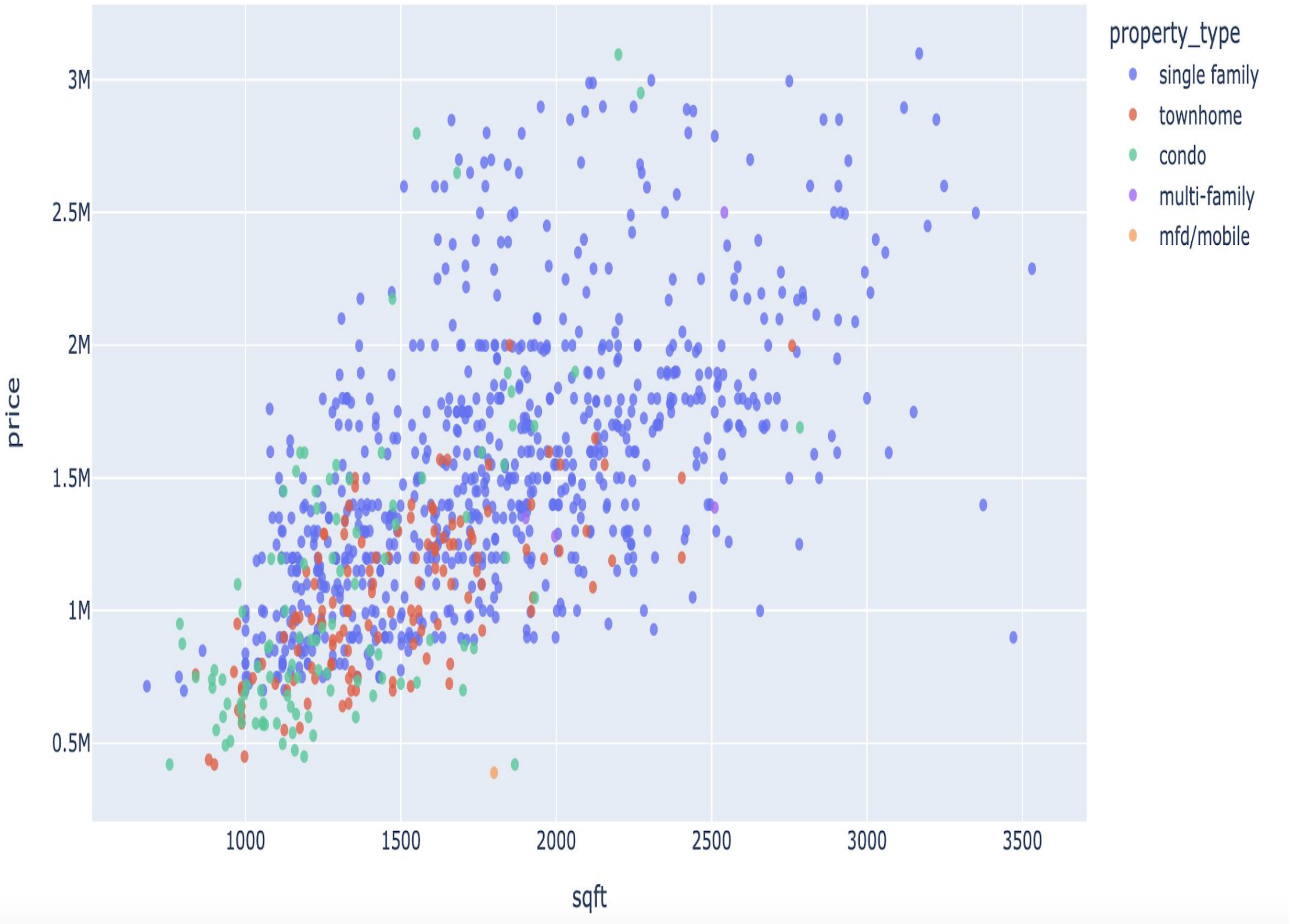
1. Moderate positive relationship between price and number of beds, price and number of bath, price and lot size, price and square feet.
2. Year the property is built in has no influence on price as it has no relationship.



Price and Square Feet

What we found

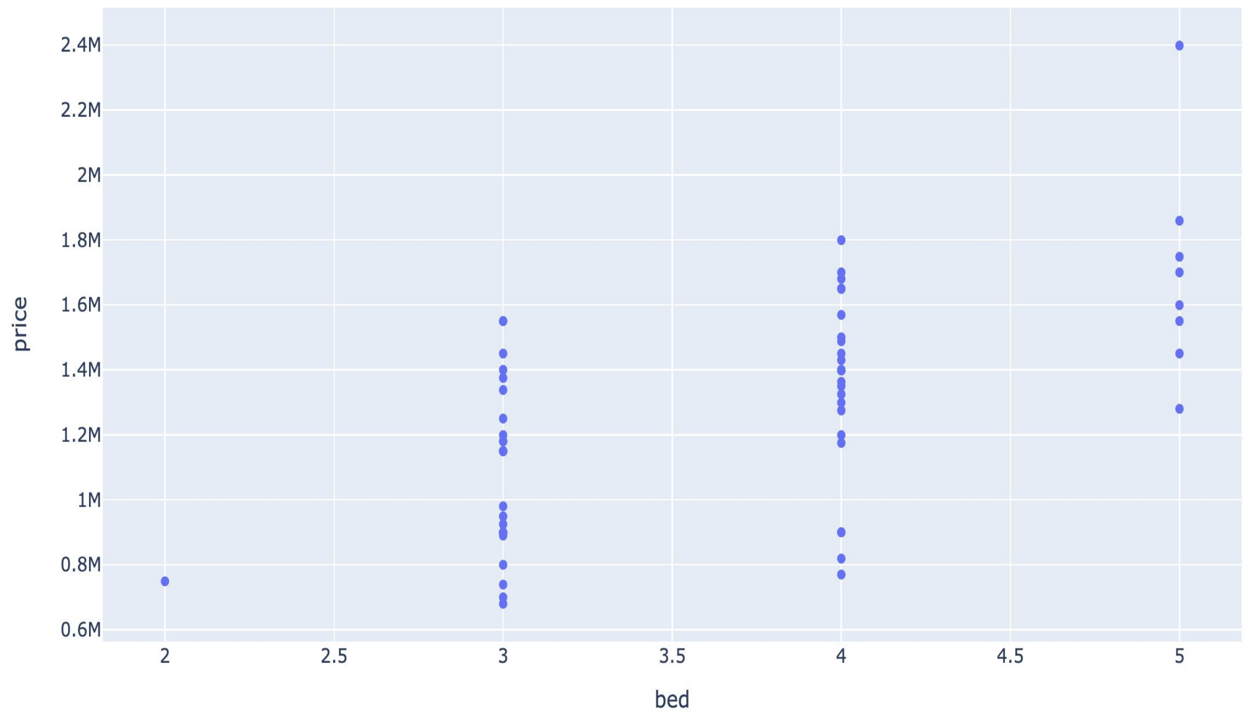
1. Positive relationship between square feet and price.
2. Most listed properties are clustered under the price range of 2M and under 2000 square feet.



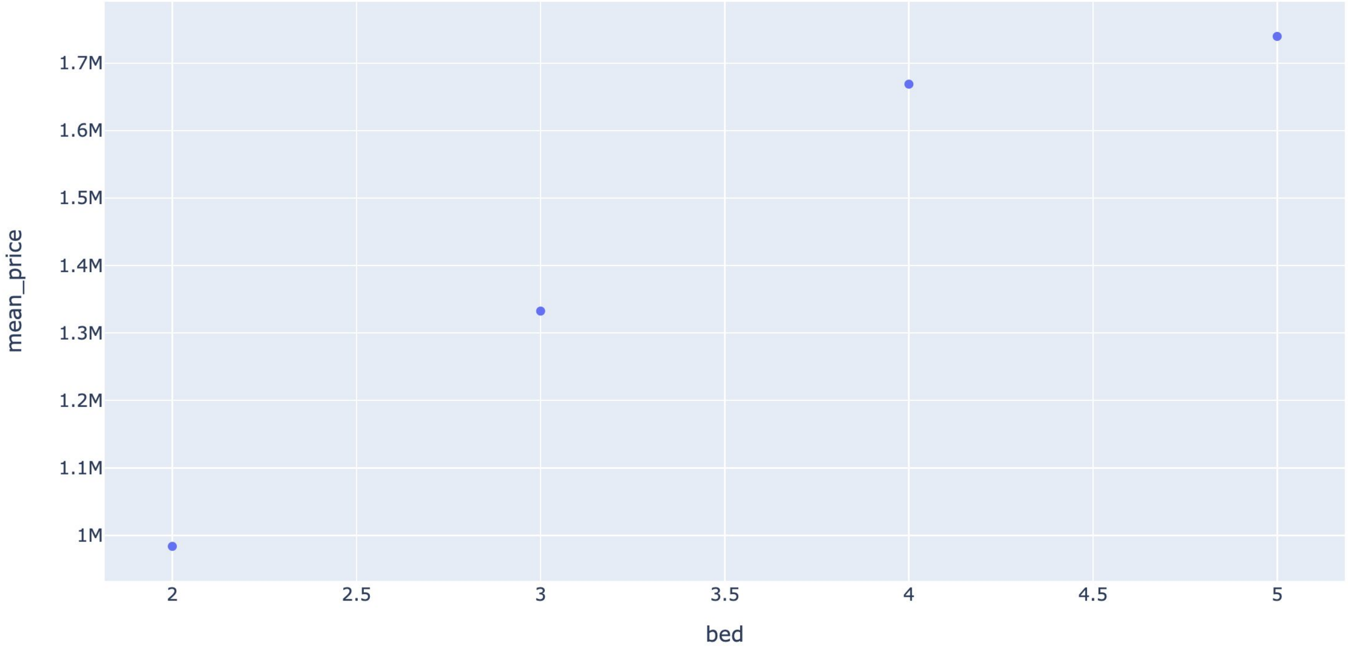
Exploring Union City by Zipcode

What we found

1. Not many options for 2 or 5 beds.
2. Lot of properties with 3 and 4 beds.
3. Wide overlap in price range among properties with 3 and 4 beds.
4. Properties listed with 3 beds appear to be skewed on both sides of the median.
5. Properties with 4 beds appears to be right skewed.



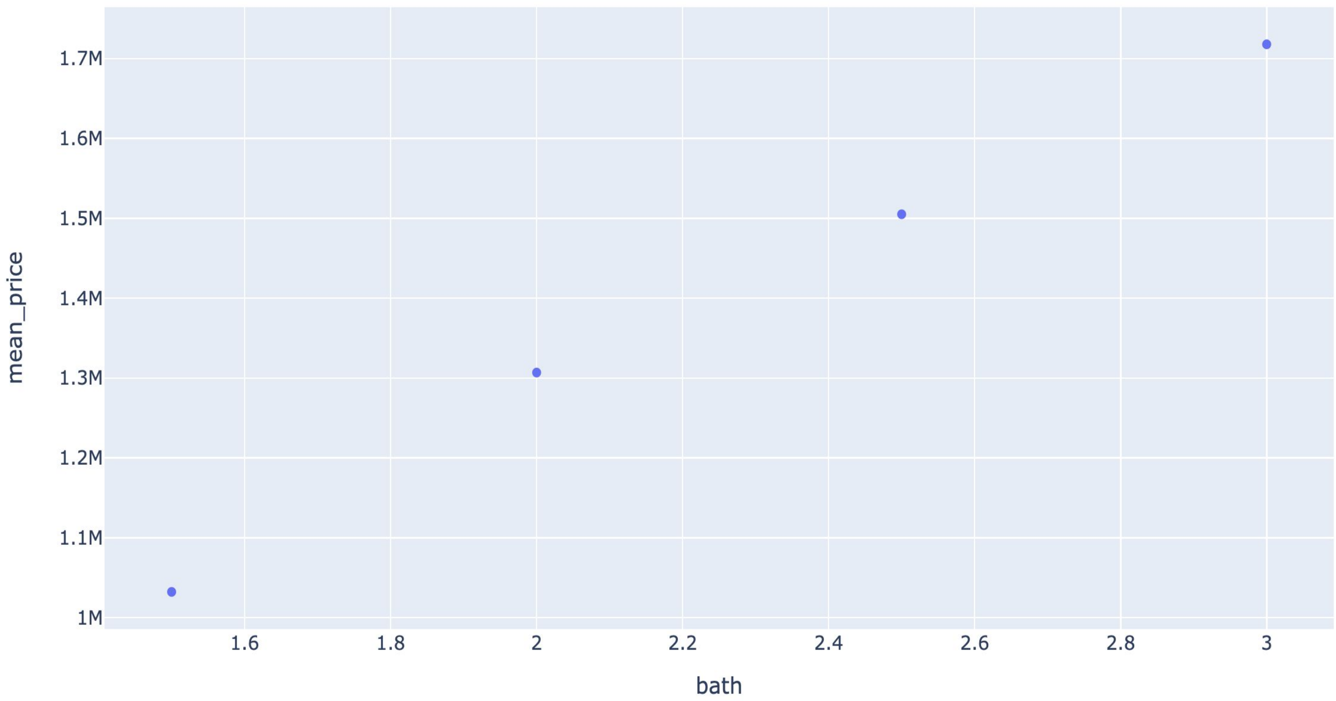
How do prices vary by bed?



	bed	mean_price
0	4.00	1668768.88
1	5.00	1739541.31
2	2.00	983762.32
3	3.00	1332438.75

Positive relationship between bed and mean price of the properties.

How do prices vary by bath?



	bath	mean_price
0	2.50	1505147.89
1	2.00	1306918.64
2	1.50	1032251.86
3	3.00	1717964.01

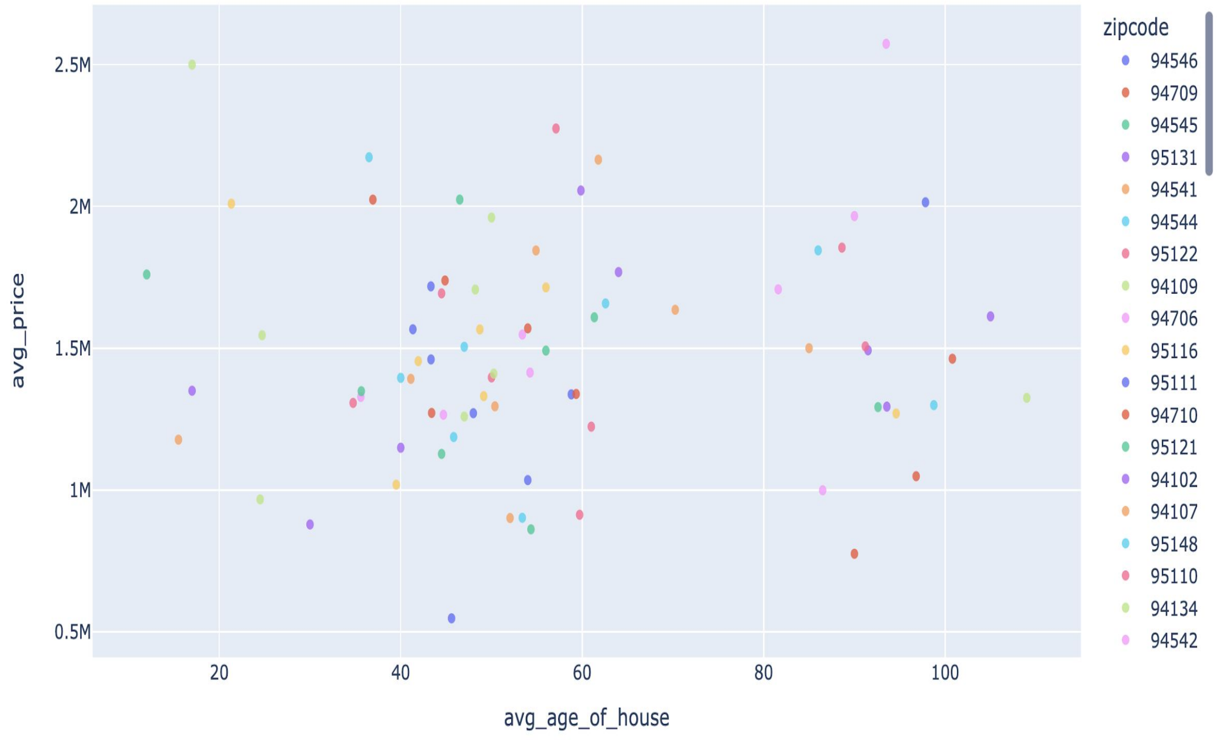
Positive relationship between bath and mean price of the properties.

Relationship between year built and price

No relationship between the price and the year the property is built in.

Fun Fact:

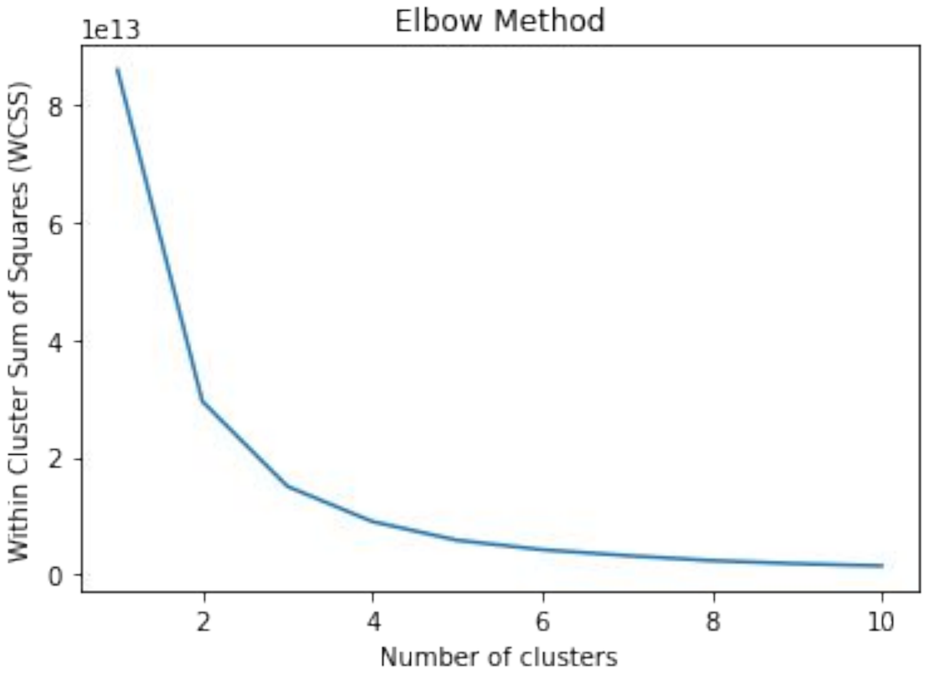
Not many properties were built around 1950s - specifically between age 60 and 80. This could be because of war that took place in 1950 or any other factor.



Hotspots in East Bay

K-Means Clustering

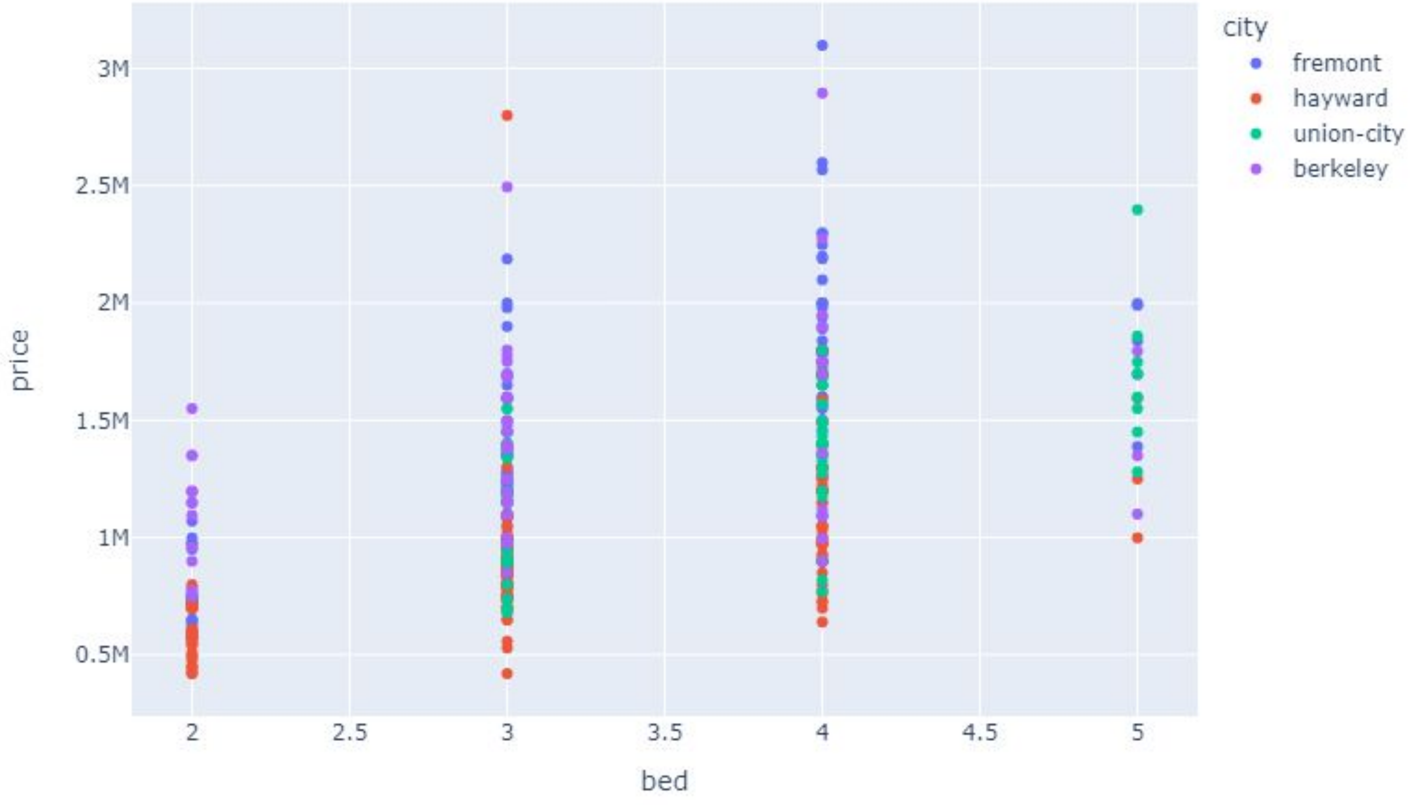
- Filtered only those listings that fall under East Bay - **Fremont, Hayward, Union City and Berkeley**
- k-means based on **price**, number of **bedrooms** and number of **bathrooms**
- Identified the ideal number of clusters using the elbow method
- Properties by cluster:



0B, 1B, 2B, 3B, 4B, 5B, 6B

{0: [0, 0, 7, 62, 51, 5, 0], 1: [0, 0, 51, 39, 8, 0, 0], 2: [0, 0, 0, 3, 10, 1, 0], 3: [0, 0, 1, 19, 47, 11, 0], 4: [0, 0, 8, 78, 28, 2, 0]}

So, what are the hotspots in East Bay?



Comparing Properties in San Francisco Bay Area and Other Cities

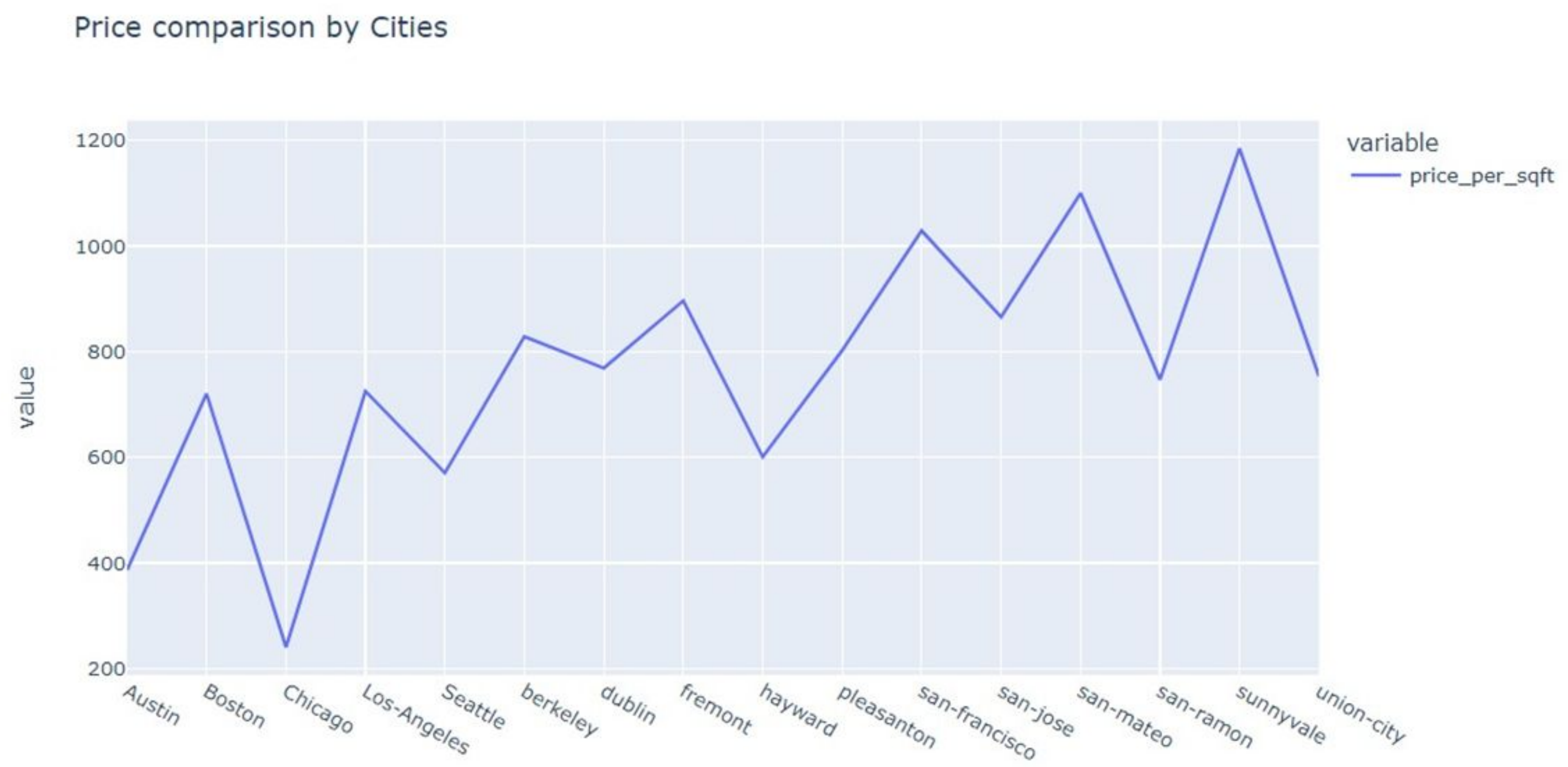
Getting new data

- Cities: **Boston**, **Chicago**, **Austin**, **Seattle**, **Los Angeles**
- Scraped data (price related columns only) to compare pricing
- Performed data cleaning steps(remove duplicates, null, outliers)

Data Analysis

- Calculated average price per sqft for each city
- Compared price per sqft for all cities

SF Bay Area Cities vs Other Big Cities




```
city
Austin           386.764706
Boston           720.463768
Chicago          240.642857
Los-Angeles      725.125000
Seattle          570.483146
berkeley         828.629630
dublin           768.869565
fremont          896.212121
hayward          600.803279
pleasanton       802.303030
san-francisco   1029.000000
san-jose         865.589041
san-mateo       1100.102941
san-ramon        747.168317
sunnyvale       1184.666667
union-city       753.612903
Name: price_per_sqft, dtype: float64
```

What we found

- Sunnyvale, San Mateo and San Francisco have the highest property prices
- Property prices in other bay area cities are comparable with prices in Boston and Los Angeles
- Hayward properties are cheapest in bay area and comparable to Seattle prices
- Austin and Chicago have lowest property prices

Regression Analysis

Missing Values

- Replaced missing values with **mean** (parking space & year) and **median** (median home price, hoa monthly, median selling price, and mhp per sqft) as the prediction model will not perform well with missing values.

Column	Non-Null Count	Dtype
state	981 non-null	object
street_address	981 non-null	object
zipcode	981 non-null	int64
bed	981 non-null	float64
bath	981 non-null	float64
sqft	981 non-null	float64
lotsize	981 non-null	float64
price	981 non-null	float64
hoa_monthly	981 non-null	float64
price_per_sqft	981 non-null	float64
parking_space	981 non-null	float64
year	981 non-null	float64
median_home_price	981 non-null	float64
median_selling_price	981 non-null	float64
mhp_per_sqft	981 non-null	float64
school_district	981 non-null	object
broker	981 non-null	object
city_berkeley	981 non-null	int64
city_dublin	981 non-null	int64
city_fremont	981 non-null	int64
city_hayward	981 non-null	int64
city_pleasanton	981 non-null	int64
city_san-francisco	981 non-null	int64
city_san-jose	981 non-null	int64
city_san-mateo	981 non-null	int64
city_san-ramon	981 non-null	int64
city_sunnyvale	981 non-null	int64
city_union-city	981 non-null	int64
property_type_condo	981 non-null	int64
property_type_mfd/mobile	981 non-null	int64
property_type_multi-family	981 non-null	int64
property_type_single_family	981 non-null	int64
property_type_townhome	981 non-null	int64

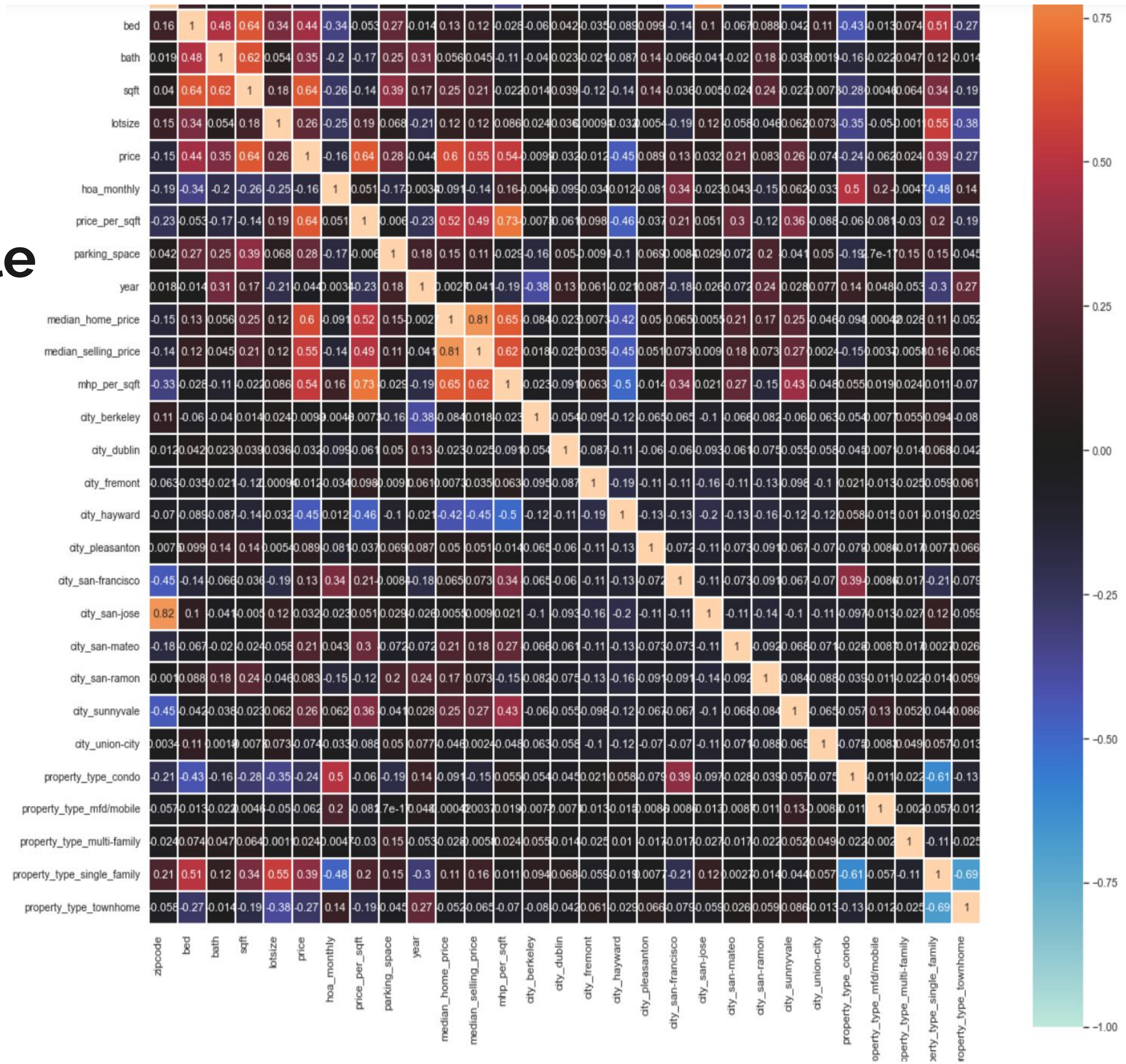
Categorical Data: One-Hot Encoding

Property type contains categorical data which should be encoded using one-hot encoding method using dummy values and concatenated to the main dataframe.

```
property_type_condo          981 non-null    uint8
property_type_mfd/mobile     981 non-null    uint8
property_type_multi-family   981 non-null    uint8
property_type_single family  981 non-null    uint8
property_type_townhome       981 non-null    uint8
```

Predictors & Outcome Variable

- Using the correlation heatmap, identified and eliminated those variables dependent partially or completely on price.
- The predictor variables are:
  - zip code
  - lot size
  - parking_space
  - year
  - property_type_condo
  - property_type_single-family
  - property_type_multi-family



Prediction: Train-Test Split

- Data Division:
- Training Data: 75 %
- Test Data: 25 %

```
train_X, valid_X, train_y, valid_y = train_test_split(X, y, random_state=3,test_size=0.25)
```

Analysis of different models using Lazy Predict

- Lazy Predict contains 41 different regression vanilla models which predicts the outcome with metrics such as R-squared value, RMSE value and the time taken for each model to run.

```
reg = LazyRegressor(ignore_warnings=False, custom_metric=None)
models, predictions = reg.fit(train_X, valid_X, train_y, valid_y)
print(models)
```

```
100%|██████████████████████████████████████████████████| 41/41 [00:01<00:00, 27.27it/s]
```

Model	Adjusted R-Squared	R-Squared	RMSE	Time Taken
LGBMRegressor	0.60	0.61	346549.87	0.06
HistGradientBoostingRegressor	0.59	0.60	351616.39	0.28
ExtraTreesRegressor	0.58	0.59	354512.17	0.12
GradientBoostingRegressor	0.58	0.59	354810.15	0.05
RandomForestRegressor	0.57	0.59	358376.49	0.16
XGBRegressor	0.54	0.56	370127.10	0.06
BaggingRegressor	0.51	0.52	384116.68	0.02
KNeighborsRegressor	0.49	0.50	392404.54	0.01
AdaBoostRegressor	0.34	0.36	444359.47	0.04
ExtraTreeRegressor	0.34	0.36	444583.89	0.00
PoissonRegressor	0.28	0.30	466624.89	0.00
LassoLarsIC	0.28	0.30	466983.69	0.00
LassoLarsCV	0.27	0.29	467323.20	0.01
LarsCV	0.27	0.29	467323.20	0.01
LassoCV	0.27	0.29	467397.29	0.04
SGDRegressor	0.27	0.29	467449.02	0.01
RidgeCV	0.27	0.29	467794.50	0.00
Ridge	0.27	0.29	467914.31	0.00
LassoLars	0.27	0.29	467924.87	0.01
Lasso	0.27	0.29	467931.23	0.01
LinearRegression	0.27	0.29	467931.47	0.00
TransformedTargetRegressor	0.27	0.29	467931.47	0.00
Lars	0.27	0.29	467931.47	0.01
OrthogonalMatchingPursuitCV	0.27	0.29	468558.04	0.01
HuberRegressor	0.26	0.28	471601.50	0.01
ElasticNet	0.24	0.26	477141.21	0.00
DecisionTreeRegressor	0.23	0.25	482059.88	0.01
GammaRegressor	0.21	0.23	487572.92	0.01

Model 1: Linear Regression

- R-Squared Value = 0.29
- MSE = 370450
- MAE = 361301

```
##Linear Regression:
LR_price = LinearRegression()
LR_price.fit(train_X, train_y)
LR_price_pred = LR_price.predict(valid_X)
print('Actual, Prediction, and Residual Prices for Validation Set\n\n')
result = round(pd.DataFrame({'Actual': valid_y,'Predicted': LR_price_pred,
                    'Residual': valid_y - LR_price_pred}), 2)
print(result.head(15))
print(rmse(LR_price,train_X, valid_X, train_y, valid_y))
mae(valid_y, LR_price_pred)
```

```
Actual, Prediction, and Residual Prices for Validation Set


         Actual   Predicted   Residual
971 1338000.00 1572321.92 -234321.92
194 1999888.00 1634663.37  365224.63
620 1798000.00 1284527.44  513472.56
58  1649000.00 1599835.78   49164.22
415  965000.00  779070.87  185929.13
825 1698000.00 1425847.02  272152.98
525 1150777.00 1339779.10 -189002.10
201 1799000.00 1610410.92  188589.08
157 1199000.00 1580640.87 -381640.87
639 1188000.00 1030691.98  157308.02
846 2349000.00 1663618.57  685381.43
676 2199000.00 1921017.50  277982.50
103  950000.00 1034399.35  -84399.35
680 2170000.00 1628433.22  541566.78
749 1338000.00 1099643.80  238356.20
R-squared: 0.293073338958433
Mean Squared Error: 370450.0452414826
None

361301.5627240737
```

Model 2: LGBM Regressor

- R-Squared Value = 0.61
- MSE = 348655
- MAE = 244612

```
##LGBM Regressor:
import lightgbm as ltb
LGBM_price = ltb.LGBMRegressor()
LGBM_price.fit(train_X, train_y)
LGBM_price_pred = LGBM_price.predict(valid_X)
print('Actual, Prediction, and Residual Prices for Validation Set\n\n')
result = round(pd.DataFrame({'Actual': valid_y,'Predicted': LGBM_price_pred,
                        'Residual': valid_y - LGBM_price_pred}), 2)
print(result.head(15))
print(rmse(LGBM_price,train_X, valid_X, train_y, valid_y))
mae(valid_y, LGBM_price_pred)
```

```
Actual, Prediction, and Residual Prices for Validation Set


        Actual  Predicted   Residual
971 1338000.00 1408745.73  -70745.73
194 1999888.00 2227334.18 -227446.18
620 1798000.00 1663084.64  134915.36
58  1649000.00 1554768.42   94231.58
415  965000.00  803352.68  161647.32
825 1698000.00 2115272.60 -417272.60
525 1150777.00 1556516.98 -405739.98
201 1799000.00 1729419.56   69580.44
157 1199000.00 1436020.70 -237020.70
639 1188000.00  934599.73  253400.27
846 2349000.00 1594359.14  754640.86
676 2199000.00 2140271.52   58728.48
103  950000.00 1231240.29 -281240.29
680 2170000.00 1817334.00  352666.00
749 1338000.00 1426044.20  -88044.20
R-squared: 0.6075343069342829
Mean Squared Error: 348655.0167650648
None

244612.63722140118
```

Model 3: Extra Trees Regressor

- R-Squared Value = 0.59
- MSE = 355689
- MAE = 255340

```
##ExtraTreesRegressor:
from sklearn.ensemble import ExtraTreesRegressor
ETR_price = ExtraTreesRegressor()
ETR_price.fit(train_X, train_y)
ETR_price_pred = ETR_price.predict(valid_X)
print('Actual, Prediction, and Residual Prices for Validation Set\n\n')
result = round(pd.DataFrame({'Actual': valid_y,'Predicted': ETR_price_pred,
                'Residual': valid_y - ETR_price_pred}), 2)
print(result.head(15))
print(rmse(ETR_price,train_X, valid_X, train_y, valid_y))
mae(valid_y, ETR_price_pred)
```

```
Actual, Prediction, and Residual Prices for Validation Set


        Actual   Predicted   Residual
971 1338000.00 1529230.00 -191230.00
194 1999888.00 1699743.74  300144.26
620 1798000.00 1697680.80  100319.20
58  1649000.00 1478799.45  170200.55
415  965000.00  788170.80  176829.20
825 1698000.00 1525663.88  172336.12
525 1150777.00 1740758.42 -589981.42
201 1799000.00 1840557.00  -41557.00
157 1199000.00 1373162.94 -174162.94
639 1188000.00 1082327.61  105672.39
846 2349000.00 1390659.97  958340.03
676 2199000.00 1811917.42  387082.58
103  950000.00 1022041.09  -72041.09
680 2170000.00 1855420.00  314580.00
749 1338000.00 1188531.73  149468.27
R-squared: 0.5915373985066905
Mean Squared Error: 355689.64208599715
None

258352.19796747967
```

Model 4: Random Forest Regressor

- R-Squared Value = 0.58
- MSE = 357892
- MAE = 253929

```
##RandomForestRegressor:
from sklearn.ensemble import RandomForestRegressor
RFR_price = RandomForestRegressor()
RFR_price.fit(train_X, train_y)
RFR_price_pred = RFR_price.predict(valid_X)
print('Actual, Prediction, and Residual Prices for Validation Set\n\n')
result = round(pd.DataFrame({'Actual': valid_y,'Predicted': RFR_price_pred,
                            'Residual': valid_y - RFR_price_pred}), 2)
print(result.head(15))
print(rmse(RFR_price,train_X, valid_X, train_y, valid_y))
mae(valid_y, RFR_price_pred)
```

```
Actual, Prediction, and Residual Prices for Validation Set


        Actual   Predicted   Residual
971 1338000.00 1383379.00  -45379.00
194 1999888.00 2006292.16   -6404.16
620 1798000.00 1753399.76   44600.24
58  1649000.00 1470032.68  178967.32
415  965000.00  784800.21  180199.79
825 1698000.00 1947510.00 -249510.00
525 1150777.00 1543281.72 -392504.72
201 1799000.00 1687571.00  111429.00
157 1199000.00 1515472.56 -316472.56
639 1188000.00  945398.62  242601.38
846 2349000.00 1536587.75  812412.25
676 2199000.00 1906602.94  292397.06
103  950000.00  932310.75   17689.25
680 2170000.00 1850918.00  319082.00
749 1338000.00 1395056.96  -57056.96
R-squared: 0.5864631847793949
Mean Squared Error: 357892.13814994076
None

254120.5703348174
```

Model 5:
Bagging Regressor

- R-Squared Value = 0.59
- MSE = 373999
- MAE = 252078

```
##BaggingRegressor:
from sklearn.ensemble import BaggingRegressor
BR_price = BaggingRegressor()
BR_price.fit(train_X, train_y)
BR_price_pred = BR_price.predict(valid_X)
print('Actual, Prediction, and Residual Prices for Validation Set\n\n')
result = round(pd.DataFrame({'Actual': valid_y,'Predicted': BR_price_pred,
                             'Residual': valid_y - BR_price_pred}), 2)
print(result.head(15))
print(rmse(BR_price,train_X, valid_X, train_y, valid_y))
mae(valid_y, BR_price_pred)
```

```
Actual, Prediction, and Residual Prices for Validation Set


         Actual   Predicted   Residual
971 1338000.00 1199900.00  138100.00
194 1999888.00 2018700.00  -18812.00
620 1798000.00 1847888.80  -49888.80
58  1649000.00 1594800.00   54200.00
415  965000.00  723790.00  241210.00
825 1698000.00 1599400.00   98600.00
525 1150777.00 1539100.00 -388323.00
201 1799000.00 1677260.00  121740.00
157 1199000.00 1670183.80 -471183.80
639 1188000.00  833688.00  354312.00
846 2349000.00 1577100.00  771900.00
676 2199000.00 1957377.60  241622.40
103  950000.00  939179.40   10820.60
680 2170000.00 1972600.00  197400.00
749 1338000.00 1201980.00  136020.00
R-squared: 0.5484022813085616
Mean Squared Error: 373999.42852427653
None

266350.88732446416
```

Model 6: Gradient Boosting Regressor

- R-Squared Value = 0.59
- MSE = 355775
- MAE = 251751

```
#GradientBoosting Regressor
GBR_price = GradientBoostingRegressor()
GBR_price.fit(train_X, train_y)
GBR_price_pred = GBR_price.predict(valid_X)
print('Actual, Prediction, and Residual Prices for Validation Set\n\n')
result = round(pd.DataFrame({'Actual': valid_y,'Predicted': GBR_price_pred,
                          'Residual': valid_y - GBR_price_pred}), 2)
print(result.head(15))
print(rmse(GBR_price,train_X, valid_X, train_y, valid_y))
mae(valid_y, GBR_price_pred)
```

```
Actual, Prediction, and Residual Prices for Validation Set


         Actual  Predicted   Residual
971 1338000.00 1353776.34  -15776.34
194 1999888.00 2046797.79  -46909.79
620 1798000.00 1592814.20  205185.80
58  1649000.00 1482671.19  166328.81
415  965000.00  857637.86  107362.14
825 1698000.00 2027559.55 -329559.55
525 1150777.00 1452504.51 -301727.51
201 1799000.00 1762775.82   36224.18
157 1199000.00 1510678.71 -311678.71
639 1188000.00  968645.84  219354.16
846 2349000.00 1779762.51  569237.49
676 2199000.00 2151611.20   47388.80
103  950000.00 1456749.16 -506749.16
680 2170000.00 1685454.22  484545.78
749 1338000.00 1366101.45  -28101.45
R-squared: 0.5913392251722225
Mean Squared Error: 355775.91639183747
None

251600.49409073155
```

Model 7: Histogram-based Gradient Boosting Regressor

- R-Squared Value = 0.60
- MSE = 351617
- MAE = 246976

```
#HistGradientBoosting Regressor
from sklearn.ensemble import HistGradientBoostingRegressor
HGBR_price = HistGradientBoostingRegressor()
HGBR_price.fit(train_X, train_y)
HGBR_price_pred = HGBR_price.predict(valid_X)
print('Actual, Prediction, and Residual Prices for Validation Set\n\n')
result = round(pd.DataFrame({'Actual': valid_y,'Predicted': HGBR_price_pred,
                      'Residual': valid_y - HGBR_price_pred}), 2)
print(result.head(15))
print(rmse(HGBR_price,train_X, valid_X, train_y, valid_y))
mae(valid_y, HGBR_price_pred)
```

```
Actual, Prediction, and Residual Prices for Validation Set


        Actual   Predicted   Residual
971 1338000.00 1373783.37  -35783.37
194 1999888.00 2259231.39 -259343.39
620 1798000.00 1628077.02  169922.98
58  1649000.00 1564005.76   84994.24
415  965000.00  833698.10  131301.90
825 1698000.00 2175019.54 -477019.54
525 1150777.00 1537484.50 -386707.50
201 1799000.00 1770094.97   28905.03
157 1199000.00 1482469.64 -283469.64
639 1188000.00  911904.48  276095.52
846 2349000.00 1644941.22  704058.78
676 2199000.00 2121951.63   77048.37
103  950000.00 1176338.26 -226338.26
680 2170000.00 1809173.72  360826.28
749 1338000.00 1418940.54  -80940.54
R-squared: 0.6008358330302545
Mean Squared Error: 351617.79226172494
None

246976.10336539647
```

Model 8: XGB Regressor

- R-Squared Value = 0.55
- MSE = 370450
- MAE = 256924

```
##XGBRegressor:
from xgboost import XGBRegressor
XGB_price = XGBRegressor()
XGB_price.fit(train_X, train_y)
XGB_price_pred = XGB_price.predict(valid_X)
print('Actual, Prediction, and Residual Prices for Validation Set\n\n')
result = round(pd.DataFrame({'Actual': valid_y,'Predicted': XGB_price_pred,
                             'Residual': valid_y - XGB_price_pred}), 2)
print(result.head(15))
print(rmse(XGB_price,train_X, valid_X, train_y, valid_y))
mae(valid_y, XGB_price_pred)
```

```
Actual, Prediction, and Residual Prices for Validation Set


         Actual   Predicted   Residual
971 1338000.00 1328038.88    9961.12
194 1999888.00 2343275.25 -343387.25
620 1798000.00 1789482.50    8517.38
58  1649000.00 1287320.75  361679.25
415  965000.00  663786.31  301213.69
825 1698000.00 1894996.75 -196996.88
525 1150777.00 1486556.38 -335779.25
201 1799000.00 1813741.50  -14741.38
157 1199000.00 1315660.25 -116660.25
639 1188000.00  968920.94  219079.00
846 2349000.00 1464048.00  884952.00
676 2199000.00 2253135.75  -54135.75
103  950000.00  920715.38   29284.62
680 2170000.00 1909839.25  260160.75
749 1338000.00 1267977.75   70022.25
R-squared: 0.5569332429445695
Mean Squared Error: 370450.0452414826
None

256924.881351626
```

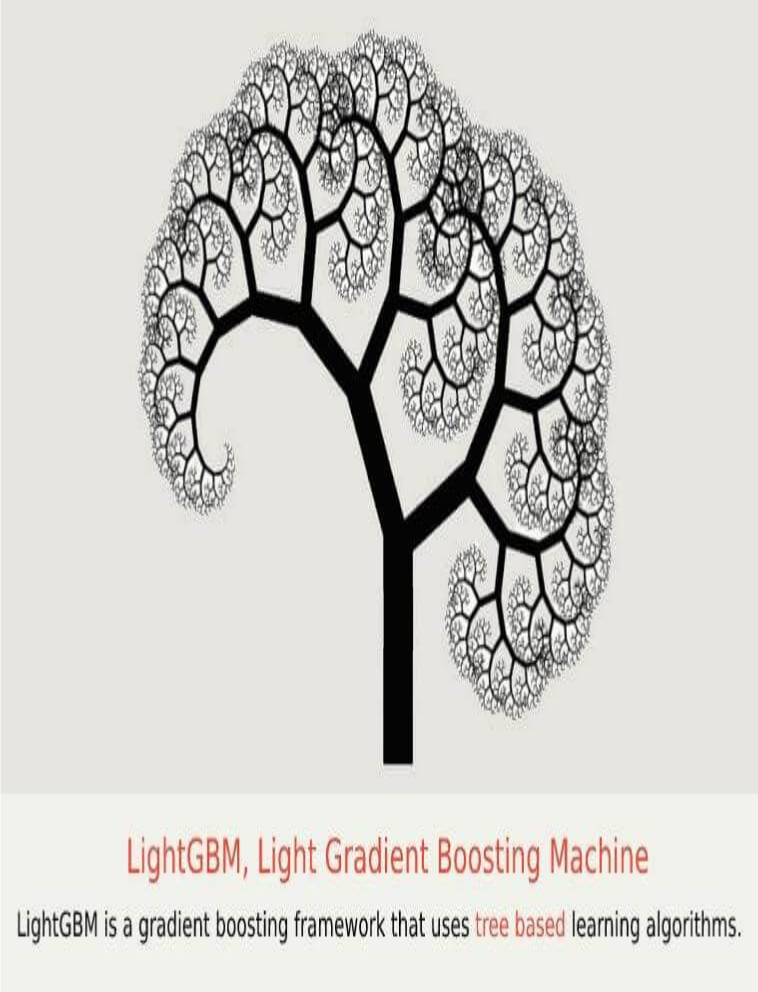
Factors considered when choosing the best model

- R-squared value: Explains the variance of the dependent variable (higher the better).
- Mean Squared Error: Lower the mean squared error, better the model.
- Mean Absolute Error: Lower the mean absolute error, better the model
- Time taken: Time taken for the model to predict the outcome.

Conclusion: Best Model

LGBM Regressor as it has:

- R-squared value of 0.61 which means the independent variable can explain 61 % of the variance of the dependent variables.
- Mean Absolute Error is 244612 which the least value compared to the other models.
- Mean Squared Error is 348655 which is the least value compared to the other models.
- Time taken is 6 seconds which helps to save computational costs.



Questions ?

Thank You!

Group 4
Aishwarya Khinvasara
Nihal Jayan
Rahul Chada
Ruchika Balani
Swetha Srinivasan