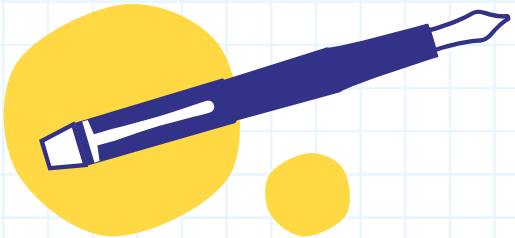
CA Housing Loan Data Analysis

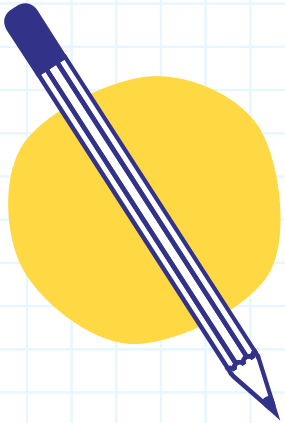
Xinyu Zou

Table of contents



01

Introduction



Datasets Information

- **Name :** California Housing Data
- **Description:** This dataset contains over 45K rows and 29 columns that introduces basic information about houses posted on Redfin.com that are located in California.
- **Important Columns**
  - **CITY**: It holds the name of the city where the property is located.
  - **PRICE**: This column should represent the price at which the property was sold or listed
  - **SQUARE FEET**: It's likely the size of the property in square feet.
- **Dataset features**
  - It describes the several important information about the property which is currently active on the market.
- **Dataset Target**
  - The target value in the dataset is ‘PRICE”

Dataset Overview

	Unnamed: 0	index	SALE TYPE	SOLD DATE	PROPERTY TYPE	ADDRESS	CITY	STATE OR PROVINCE	ZIP OR POSTAL CODE	PRICE	...	STATUS	NEXT OPEN HOUSE START TIME	NEXT OPEN HOUSE END TIME	URL (SEE https://www.redfin.com/buy-a-home/comparative-market-analysis FOR INFO ON PRICING)	SOURCE	MLS#	FAVORITE	INTERESTED	LATITUDE	LONGITUDE
0	0	0	MLS Listing	NaN	Single Family Residential	898 Saint Jean Ct	Kenwood	CA	95452	1895000.0	...	Active	NaN	NaN	https://www.redfin.com/CA/Kenwood/898-St-Jean-...	BAREIS	323008219	N	Y	38.423663	-122.546226
1	1	1	MLS Listing	NaN	Vacant Land	1890 Lawndale Rd	Kenwood	CA	95452	749000.0	...	Active	NaN	NaN	https://www.redfin.com/CA/Kenwood/1890-Lawndal...	BAREIS	323008952	N	Y	38.407264	-122.572197
2	2	2	MLS Listing	NaN	Single Family Residential	9305 Clyde Ave	Kenwood	CA	95452	1299000.0	...	Active	NaN	NaN	https://www.redfin.com/CA/Kenwood/9305-Clyde-A...	BAREIS	323008201	N	Y	38.415200	-122.549272
3	3	3	MLS Listing	NaN	Single Family Residential	1335 Kinnybrook Dr	Kenwood	CA	95452	4350000.0	...	Active	NaN	NaN	https://www.redfin.com/CA/Kenwood/1335-Kinnybr...	BAREIS	322101941	N	Y	38.426317	-122.535020
4	4	4	MLS Listing	NaN	Single Family Residential	2700 Nelligan Rd	Glen Ellen	CA	95452	2995000.0	...	Active	NaN	NaN	https://www.redfin.com/CA/Kenwood/2700-Nelliga...	BAREIS	322073908	N	Y	38.413720	-122.545828

5 rows × 29 columns

Dataset Overview

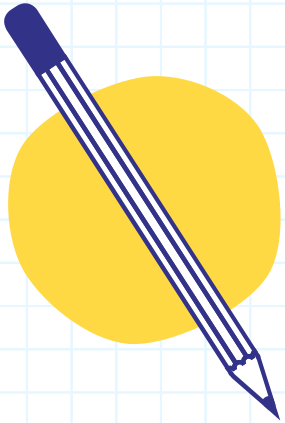
	Unnamed: 0	index	SOLD DATE	PRICE	BEDS	BATHS	SQUARE FEET	LOT SIZE	YEAR BUILT	DAYS ON MARKET	$/SQUARE FEET	HOA/MONTH	LATITUDE	LONGITUDE
count	47116.000000	47116.000000	0.0	4.710300e+04	33983.000000	33347.000000	33086.000000	4.110400e+04	31642.000000	45451.000000	3.308600e+04	12339.000000	47085.000000	47085.000000
mean	23557.500000	59.250934	NaN	1.459580e+06	3.610452	3.510541	2425.450795	1.325999e+07	1977.756969	119.505973	1.212938e+03	514.054461	34.751742	-118.510022
std	13601.361978	70.669886	NaN	6.116573e+06	3.223399	96.889531	3439.643191	2.522417e+09	31.091486	200.771422	2.711093e+04	798.932007	1.518264	1.938378
min	0.000000	0.000000	NaN	1.000000e+00	0.000000	0.500000	1.000000	1.000000e+00	1776.000000	1.000000	3.000000e+00	0.000000	32.555579	-123.426265
25%	11778.750000	12.000000	NaN	2.980000e+05	2.000000	2.000000	1305.250000	6.098000e+03	1958.000000	18.000000	3.060000e+02	180.000000	33.808946	-118.808435
50%	23557.500000	31.000000	NaN	6.550000e+05	3.000000	2.500000	1824.000000	1.066400e+04	1980.000000	53.000000	4.800000e+02	337.000000	34.131992	-118.007671
75%	35336.250000	77.000000	NaN	1.300000e+06	4.000000	3.000000	2682.000000	8.363425e+04	2003.000000	149.000000	7.340000e+02	565.000000	35.051729	-117.198825
max	47115.000000	349.000000	NaN	1.000000e+09	100.000000	17680.000000	364162.000000	5.112162e+11	2024.000000	4590.000000	2.000000e+06	35315.000000	38.888113	-114.256344

```
ca_housing.shape
```

```
(47116, 29)
```

02

Data Cleaning



Step by Step

- **Check for Duplicated rows**

```
# Check for Duplicate
duplicate_rows = ca_housing.duplicated()
print(ca_housing[duplicate_rows])
```

```
Empty DataFrame
Columns: [Unnamed: 0, index, SALE TYPE, SOLD DATE, PROPERTY TYPE, ADDRESS, CITY, STATE OR PROVINCE,
Index: []

[0 rows x 29 columns]
```

- **Check Outlier and Drop Outlier**

```
# Check Outlier with IQR Method
Q1 = ca_housing.quantile(0.25)
Q3 = ca_housing.quantile(0.75)
IQR = Q3 - Q1

# Drop Outlier with IQR Method
outliers = ((ca_housing < (Q1 - 1.5 * IQR)) | (ca_housing > (Q3 + 1.5 * IQR)))
ca_housing = ca_housing[~outliers.any(axis=1)]
```

```
ca_housing.shape
```

```
(24200, 29)
```

- **Find focus group**

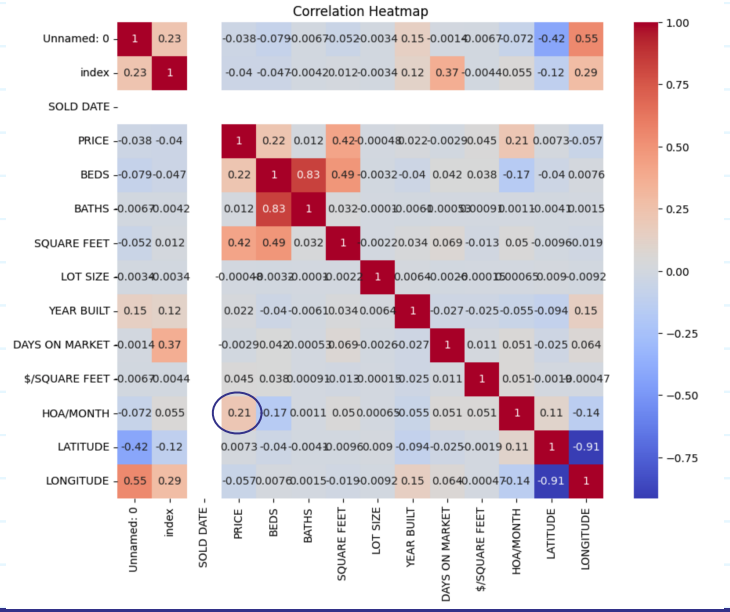
```
ca_housing = ca_housing[ca_housing['SALE TYPE'] == 'MLS Listing']
ca_housing.shape
```

SALE TYPE	
MLS Listing	95%
New Construction P...	3%
Other (574)	1%

Step by Step(cont)

- **Define Feature and Target**

```
# Define Features and Target
features = ca_housing[['BEDS', 'BATHS', 'CITY', 'SQUARE FEET', 'YEAR BUILT', 'DAYS ON MARKET', 'PROPERTY TYPE', 'LOT SIZE']]
target = ca_housing['PRICE']
```



Step by Step(cont)

- **Drop the Nan Value**

```
features = features.dropna()
features.shape
```

```
index = features.index
target = target.loc[index]
```

- **Mak target and features same length**

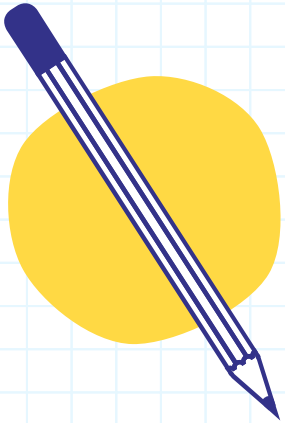
```
target = target.dropna()
target.shape
```

```
features.shape
```

```
(15204, 8)
```

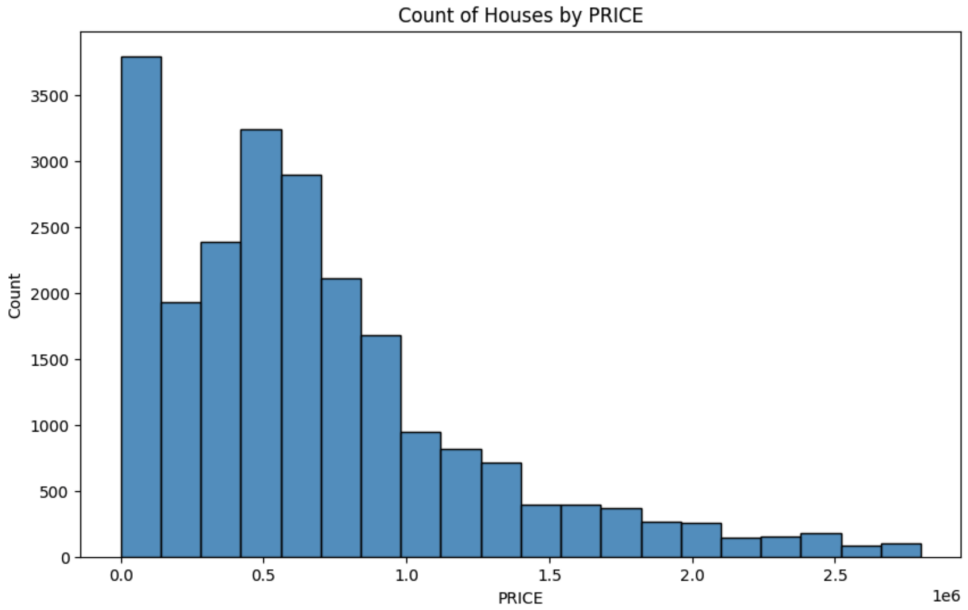
03

Data Visualization



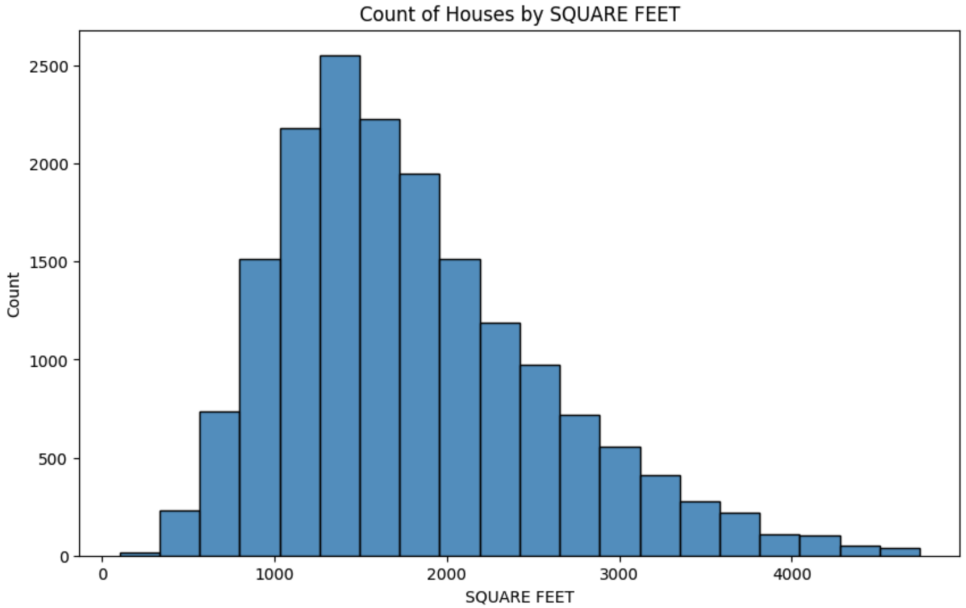
Price

```
plt.figure(figsize=(10, 6))
sns.histplot(data=ca_housing, x='PRICE', bins=20)
plt.xlabel("PRICE")
plt.ylabel("Count")
plt.title("Count of Houses by PRICE")
plt.show()
```



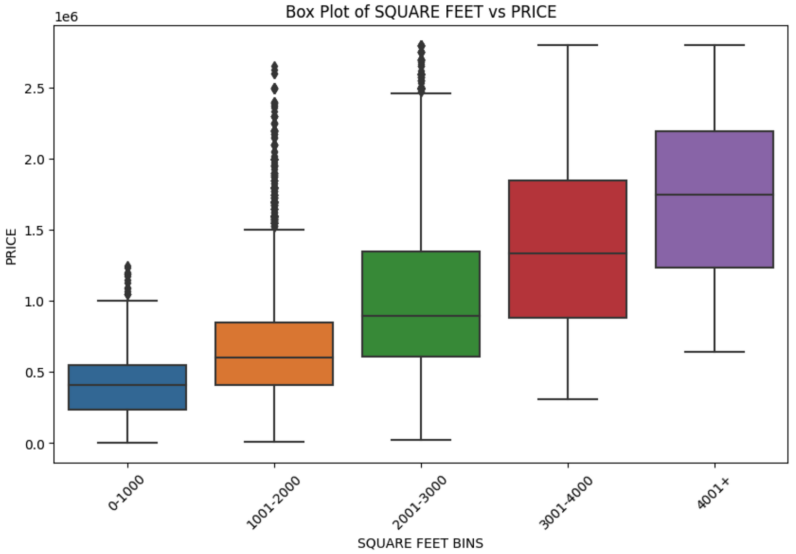
Square Feet

```
plt.figure(figsize=(10, 6))
sns.histplot(data=ca_housing, x='SQUARE FEET', bins=20)
plt.xlabel("SQUARE FEET")
plt.ylabel("Count")
plt.title("Count of Houses by SQUARE FEET")
plt.show()
```



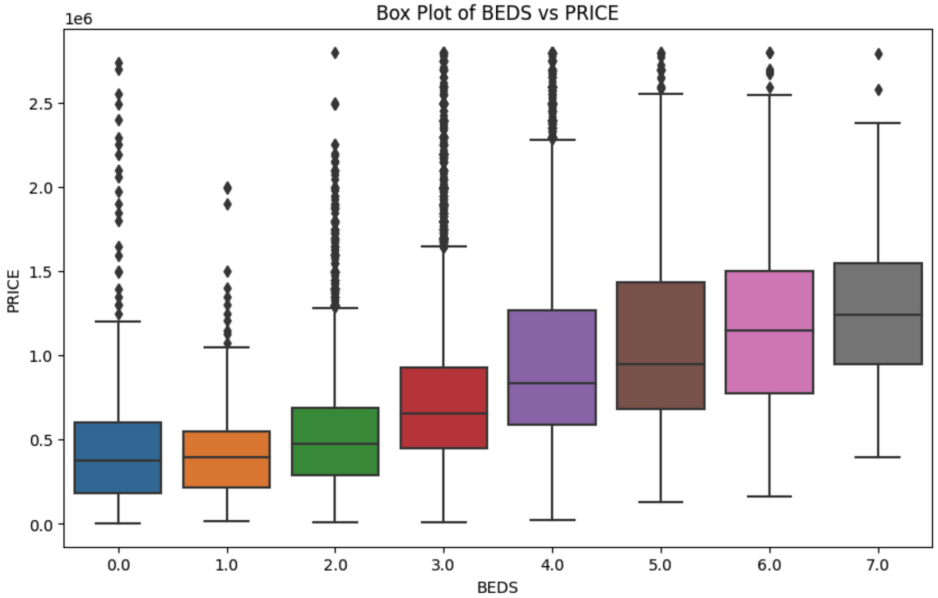
Price & Square Feet

```
bins = [0, 1000, 2000, 3000, 4000, float('inf')]
labels = ['0-1000', '1001-2000', '2001-3000', '3001-4000', '4001+']
ca_housing['SQUARE FEET BINS'] = pd.cut(ca_housing['SQUARE FEET'], bins=bins, labels=labels)

plt.figure(figsize=(10, 6))
sns.boxplot(data=ca_housing, x='SQUARE FEET BINS', y='PRICE')
plt.title("Box Plot of SQUARE FEET vs PRICE")
plt.xlabel("SQUARE FEET BINS")
plt.ylabel("PRICE")
plt.xticks(rotation=45)
plt.show()
```



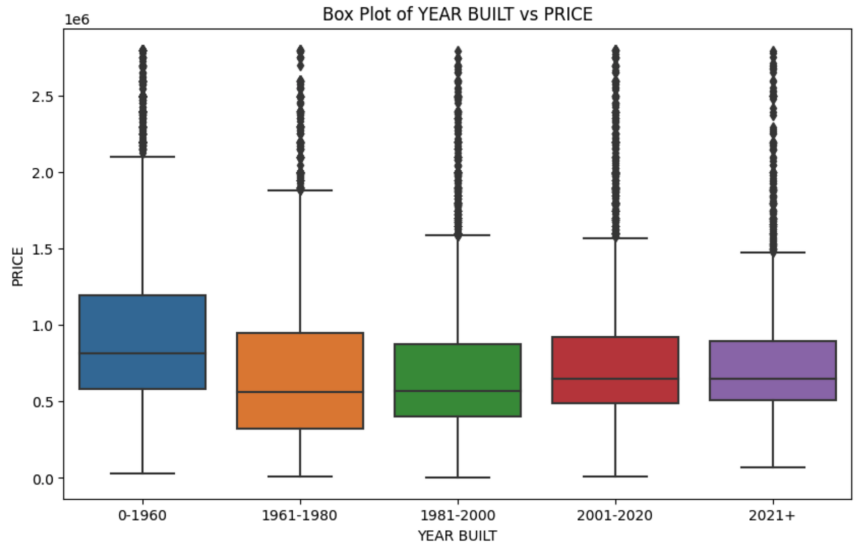
Price & Beds

```
# price & bed
plt.figure(figsize=(10, 6))
sns.boxplot(data=ca_housing, x='BEDS', y='PRICE')
plt.title("Box Plot of BEDS vs PRICE")
plt.xlabel("BEDS")
plt.ylabel("PRICE")
plt.show()
```



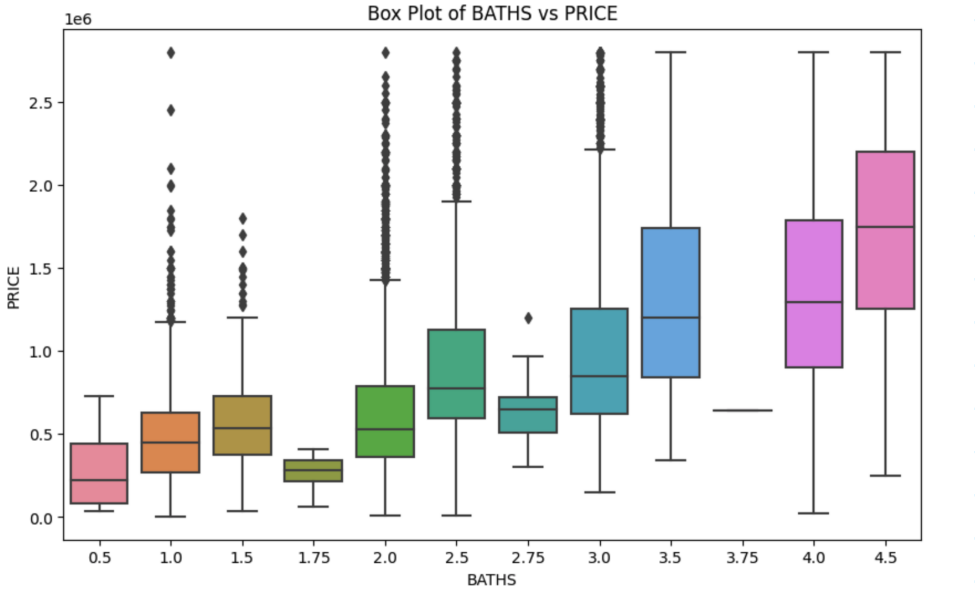
Price & Year Built

```
# price & year built
bins = [0, 1960, 1980, 2000, 2020, float('inf')]
labels = ['0-1960', '1961-1980', '1981-2000', '2001-2020', '2021+']
ca_housing['YEAR BUILT'] = pd.cut(ca_housing['YEAR BUILT'], bins=bins, labels=labels)
plt.figure(figsize=(10, 6))
sns.boxplot(data=ca_housing, x='YEAR BUILT', y='PRICE')
plt.title("Box Plot of YEAR BUILT vs PRICE")
plt.xlabel("YEAR BUILT")
plt.ylabel("PRICE")
plt.show()
```



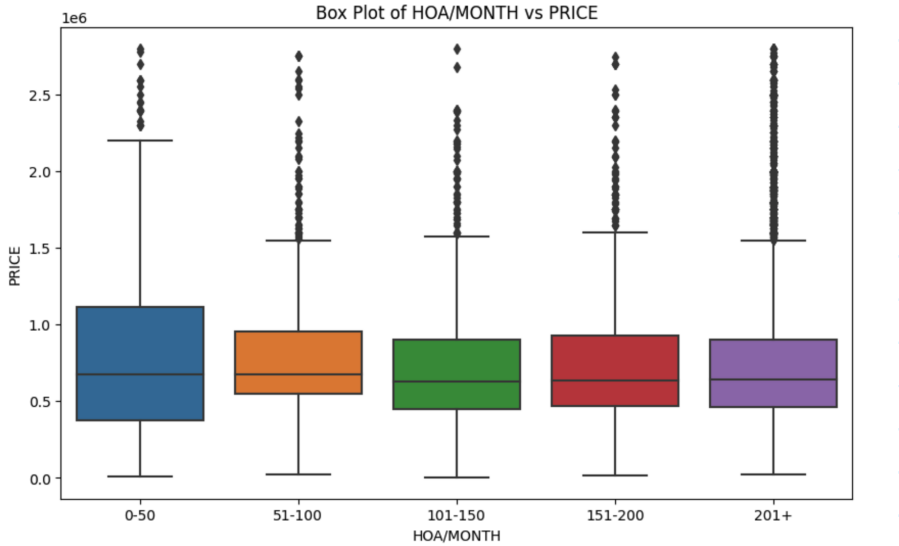
Price & BATHS

```
# price & bath
plt.figure(figsize=(10, 6))
sns.boxplot(data=ca_housing, x='BATHS', y='PRICE')
plt.title("Box Plot of BATHS vs PRICE")
plt.xlabel("BATHS")
plt.ylabel("PRICE")
plt.show()
```



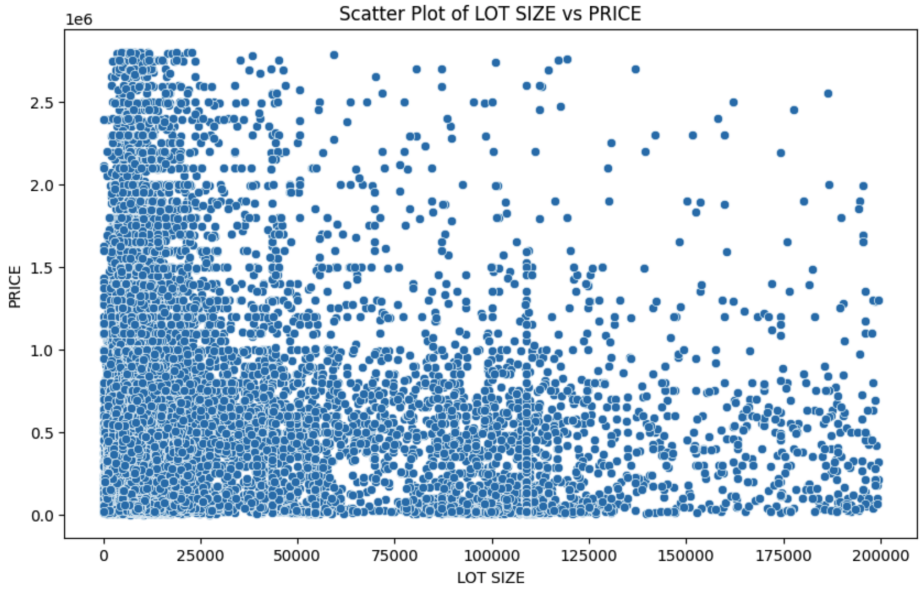
Price & HOA/Month

```
# price & hoa
bins = [0, 50, 100, 150, 200, float('inf')]
labels = ['0-50', '51-100', '101-150', '151-200', '201+']
ca_housing['HOA/MONTH'] = pd.cut(ca_housing['HOA/MONTH'], bins=bins, labels=labels)
plt.figure(figsize=(10, 6))
sns.boxplot(data=ca_housing, x='HOA/MONTH', y='PRICE')
plt.title("Box Plot of HOA/MONTH vs PRICE")
plt.xlabel("HOA/MONTH")
plt.ylabel("PRICE")
plt.show()
```



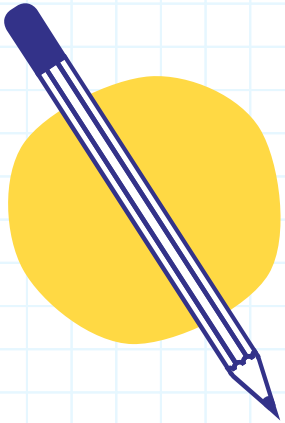
Price & LOT SIZE

```
# price & lot size
plt.figure(figsize=(10, 6))
sns.scatterplot(data=ca_housing, x='LOT SIZE', y='PRICE')
plt.title("Scatter Plot of LOT SIZE vs PRICE")
plt.xlabel("LOT SIZE")
plt.ylabel("PRICE")
plt.show()
```



04

Machine Learning



Models

Model 1	Linear Regression
Model 2	Logistic Regression
Model 3	Gradient Boosting Regression
Model 4	Lasso Regression
Model 5	Neural Network

Linear Regression

```
X1 = pd.get_dummies(features, columns=['PROPERTY TYPE', 'CITY'])
y1 = target
```

```
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2, random_state=42)
```

```
model1 = LinearRegression()
model1.fit(X1_train, y1_train)
```

```
] score1 = r2_score(y1_test, y1_pred)
 print("The score is:", score1)
```

```
The score is: 0.7464689530130446
```

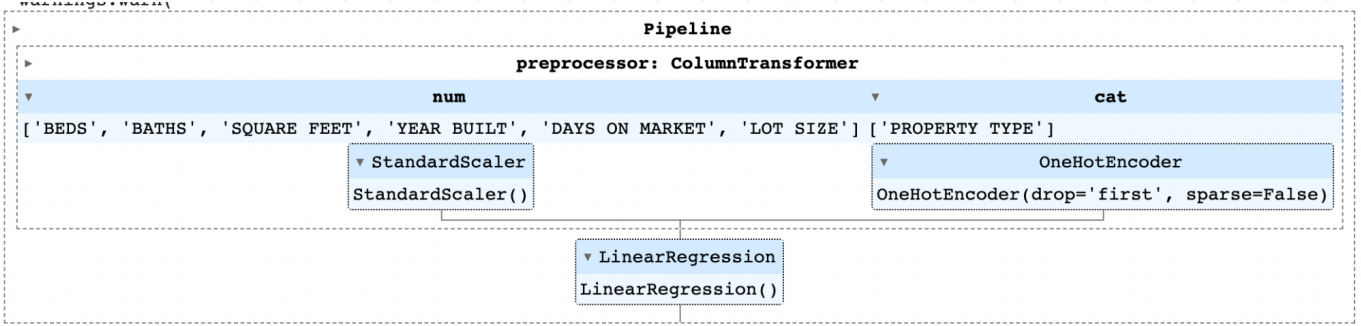
Logistic Regression

```
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2, random_state=42)
```

```
numeric_features = ['BEDS', 'BATHS', 'SQUARE FEET', 'YEAR BUILT', 'DAYS ON MARKET','LOT SIZE']
categorical_features = ['PROPERTY TYPE']

numeric_imputer = SimpleImputer(strategy='mean')
X2_train[numeric_features] = numeric_imputer.fit_transform(X2_train[numeric_features])


categorical_imputer = SimpleImputer(strategy='most_frequent')
X2_train[categorical_features] = categorical_imputer.fit_transform(X2_train[categorical_features])
```

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Create a pipeline with preprocessor and regressor
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

# Fit the pipeline to the data
pipeline.fit(X2_train, y2_train)
```

Logistic Regression




```
mae = mean_absolute_error(y2_test, y2_pred)
mse = mean_squared_error(y2_test, y2_pred)
r2 = r2_score(y2_test, y2_pred) #calculates the R-squared (R2) score, which measures the goodness of fit.

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2): {r2}")
```

```
Mean Absolute Error (MAE): 288724.0296255128
Mean Squared Error (MSE): 149157846278.70868
R-squared (R2): 0.44624406264789906
```

Gradient Boosting Regression

```
gb_regressor = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
gb_regressor.fit(Xg_train_scaled, yg_train)
```

```
GradientBoostingRegressor
GradientBoostingRegressor(random_state=42)
```

```
yg_pred = gb_regressor.predict(Xg_test_scaled)

r2 = r2_score(yg_test, yg_pred)
print("R-squared (R2) Score:", r2)
```

```
R-squared (R2) Score: 0.7229034813442556
```

Lasso Regression

```
[51] pipe_lasso = make_pipeline(StandardScaler(), Lasso(alpha = 100,max_iter = 30000))
```

```
[53] lasso = pipe_lasso.fit(Xl_train, yl_train)
```

```
[55] print("Test set score: {:.2f}".format(lasso.score(Xl_test, yl_test)))

     Test set score: 0.74
```

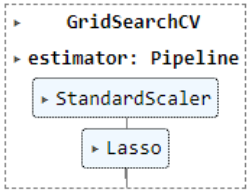
Lasso Regression

Grid search

```
param_grid = {'lasso__alpha': np.logspace(1, 7, num = 13),
              'lasso__max_iter': [10000]}
```

```
grid = GridSearchCV(pipe_lasso, param_grid = param_grid, cv = 10, n_jobs = -1)
```

```
grid.fit(Xl_train, yl_train)
```




```
] print("Score for the best model: {:.2f}".format(grid.score(Xl_test, yl_test)))

  Score for the best model: 0.74
```

Neural Network

Neural Network can capture the complex relationship between data, and we have experimented with different architectures

```
modeln = keras.Sequential()
modeln.add(layers.Input(shape=(Xn_train_scaled.shape[1],)))
modeln.add(layers.Dense(64, activation='relu'))
modeln.add(layers.Dense(32, activation='relu'))
modeln.add(layers.Dense(1))  # Output layer for regression
modeln.compile(optimizer='adam', loss='mean_squared_error')
modeln2 = keras.Sequential()
modeln2.add(layers.Input(shape=(Xn_train_scaled.shape[1],)))
modeln2.add(layers.Dense(128, activation='relu'))
modeln2.add(layers.Dense(128, activation='relu'))
modeln2.add(layers.Dense(64, activation='relu'))
modeln2.add(layers.Dense(1))  # Output layer for regression
modeln2.compile(optimizer='adam', loss='mean_squared_error')
modeln3 = keras.Sequential()
modeln3.add(layers.Input(shape=(Xn_train_scaled.shape[1],)))
modeln3.add(layers.Dense(256, activation='relu'))
modeln3.add(layers.Dense(128, activation='relu'))
modeln3.add(layers.Dense(64, activation='relu'))
modeln3.add(layers.Dense(1))  # Output layer for regression
modeln3.compile(optimizer='adam', loss='mean_squared_error')
```

Neural Network

After comparing the result, we found that our best performance reached a 0.799 R2 score

```
] score = modeln.evaluate(Xn_test, yn_test, verbose=0)
  print("Test loss:", score)

  Test loss: 4.933050177482129e+18
```

```
] from sklearn.metrics import r2_score

  yn_pred = modeln.predict(Xn_test_scaled)
  r2 = r2_score(yn_test, yn_pred)

  print("R-squared (R2) Score:", r2)

  96/96 [==============================] - 0s 2ms/step
  R-squared (R2) Score: 0.6414794924787358
```

```
score = modeln2.evaluate(Xn_test, yn_test, verbose=0)
print("Test loss:", score)
yn2_pred = modeln2.predict(Xn_test_scaled)
r2 = r2_score(yn_test, yn2_pred)

print("R-squared (R2) Score:", r2)

Test loss: 2.6895307858656625e+19
96/96 [==============================] - 0s 1ms/step
R-squared (R2) Score: 0.7901178580051719
```

```
score = modeln3.evaluate(Xn_test, yn_test, verbose=0)
print("Test loss:", score)
yn3_pred = modeln3.predict(Xn_test_scaled)
r2 = r2_score(yn_test, yn3_pred)

print("R-squared (R2) Score:", r2)

Test loss: 2.5739109809399005e+19
96/96 [==============================] - 0s 3ms/step
R-squared (R2) Score: 0.7993165635101627
```

79.9%

Best Model for CA Housing Loan

Deep Neural Network

Key Takeaways

- Our model performance greatly improved aftering changing zip code to city because the model takes the zip code column as values instead of string
- Despite that HOA column has a higher correlation, we still choose to not include it into our feature columns because it has many null values. Include HOA column would decrease the sample size and increase the risk of overfitting
- Model tuning is very important in building machine learning models, there is not a universal parameters that fits all the datasets

Thanks!

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**