FORECASTING HOUSE PRICES WITH REGRESSION MODELS

S.NO.	NAME	EMAIL ID	ROLE
1.	Achyuth Kumar Miryala	AchyuthKumarMiryala@my.unt.edu	Coding and Presentation
2.	Rajkumar Vallepu	RajkumarVallepu@my.unt.edu	Researching and Editing
3.	Bhanu Prakash Gadala	bhanuprakashgadala@my.unt.edu	Training and Testing
4.	Sailendra Chowdary Sabbineni	Sailendrachowdarsabbineni@my.unt.edu	Determining Accuracy and Documentation
5.	Naga Swetha Gollapudi	NagaSwethaGollapudi@my.unt.edu	Data Cleaning and Data Collection

Workflow:

By splitting out the work, we were able to collaborate on the project and complete tasks like data preprocessing, visualization, and forecasting the final result. subsequently talking about the project tasks' updates and progress. This cooperative endeavor has encompassed exchanging study results and cooperating on tasks to create and enhance methods and strategies that enhance performance. We studied missing data handling methodologies. I've included those to the references within this document's section.

Abstract

The Real Estate Agency is interested in building a model that can accurately predict the sale price of a house based on its characteristics. To address this problem, we will be using the Zillow house dataset which contains information on the sale of houses in the USA.

There is an observed increasing demand for accurate and reliable estimates of the sale price of homes. However, predicting the sale price of a house can be a complex and challenging task, as it depends on various factors such as location, size, condition, and amenities. To address this challenge, the Real Estate Agency has decided to build a model that can accurately predict the sale price of a house based on its characteristics.

One of the major challenges in building such a model is identifying which features have the most impact on the sale price of a house. There may also be missing or incorrect data in the dataset, which will need to be handled appropriately. Additionally, some features may be categorical in nature (e.g., zip code) and will need to be converted to numerical values for use in the model.

We propose building a machine learning model using regression techniques to predict the sale price of a house based on its features. We will preprocess the dataset to handle missing or incorrect data and convert any categorical features to numerical values. We will then split the data into training and testing sets and use various regression techniques (such as linear regression, random forest regression, and XG Boost) to train and evaluate the model. We will select the best performing model and use it to predict the sale price of new houses.

By building this model, the Real Estate Agency will be able to provide accurate estimates of house prices to their clients. Additionally, by understanding which features have the most impact on sale price, the Real Estate Agency can provide recommendations to homeowners on how to increase the value of their homes through renovations or improvements. Overall, this project will provide valuable insights into the factors that influence the sale price of a house in the USA.

Data Specification

We'll be using the Zillow dataset.

The dataset we used is a valuable resource for understanding the real estate market in USA. The data can be used to track trends in sale prices, square footage, and other factors over time. It can also be used to compare different neighborhoods and to identify properties that are under- or overpriced.

The data has:

- **21597** rows which are the number of houses sold.
- **21** columns which represent the house's features.

The features are either strings, floats or integers.

The dataset contains:

id - unique identifier for a house

date - date the house was sold.

price - Sale price of the house (in dollars)

bedrooms - number of Bedrooms in a house

bathrooms - number of bathrooms in a house

sqft_living - square footage of the house

sqft_lot - square footage of the lot

floors - total floors (levels) in the house

waterfront - House which has a view to the waterfront.

view - Quality of view from house

condition - How good the condition is (Overall)

grade - overall grade given to the housing unit, based on King County grading system.

sqft_above - square footage of house apart from basement

sqft_basement - square footage of the basement

yr_built - Year the house was built

yr_renovated - Year the house was renovated.

zip code - zip code in which the house is located.

lat - Latitude coordinate

long - Longitude coordinate

sqft_living15 - The square footage of interior housing living space for the nearest 15 neighbors

sqft_lot15 - The square footage of the land lots of the nearest 15 neighbors

Problem Statement

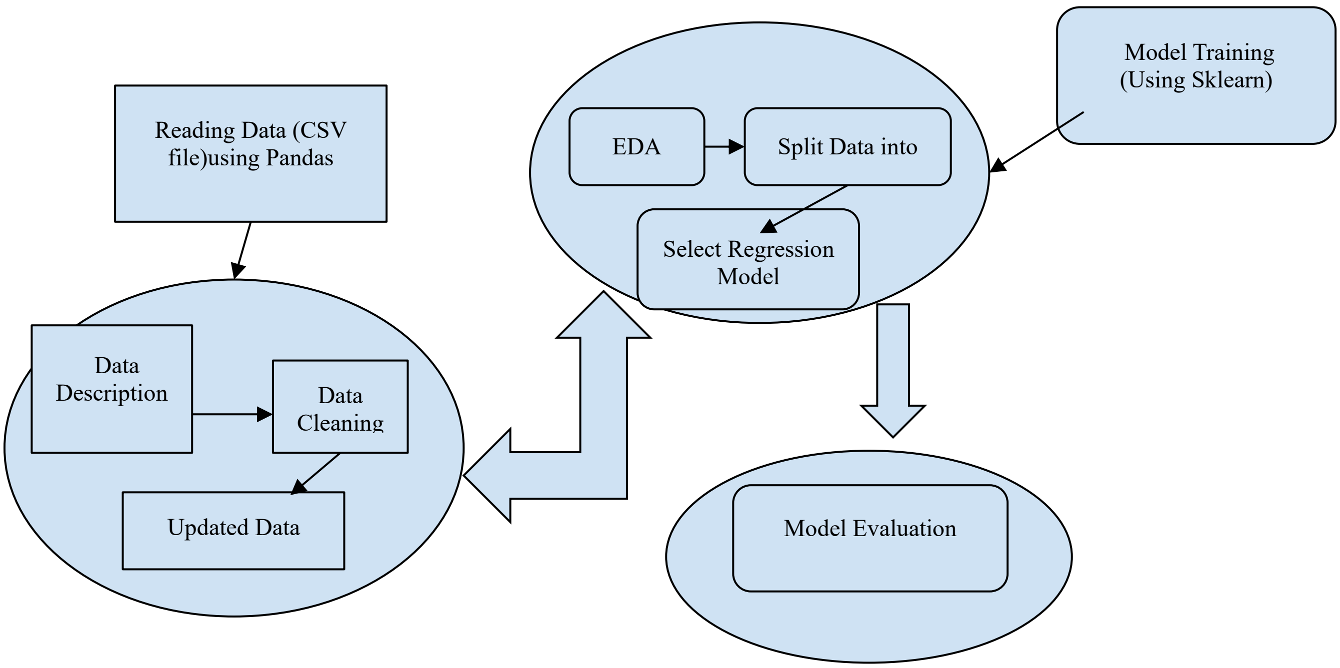
The Real Estate Agency wants to build a model that can accurately predict the sale price of a house based on its characteristics using the Zillow dataset. The agency faces the challenge of identifying which features have the most impact on the sale price of a house, handling missing or incorrect data in the dataset, and converting categorical features to numerical values.

Objectives

- To identify the features that have the most impact on the sale price of a house in USA.
- To preprocess the dataset and handle any missing or incorrect data.
- To convert any categorical features to numerical values for use in the model.
- To build a machine learning model using regression techniques to predict the sale price of a house based on its features.
- To evaluate the performance of various regression techniques and select the best performing model.
- To use the selected model to predict the sale price of new houses with high accuracy.
- To provide insights to the Real Estate Agency on how different features impact the sale price of a house and provide recommendations to homeowners on how to increase the value of their homes through renovations or improvements.

Project Design:

Flowchart of Design



Implementation of data Processing techniques:

Data Preprocessing:

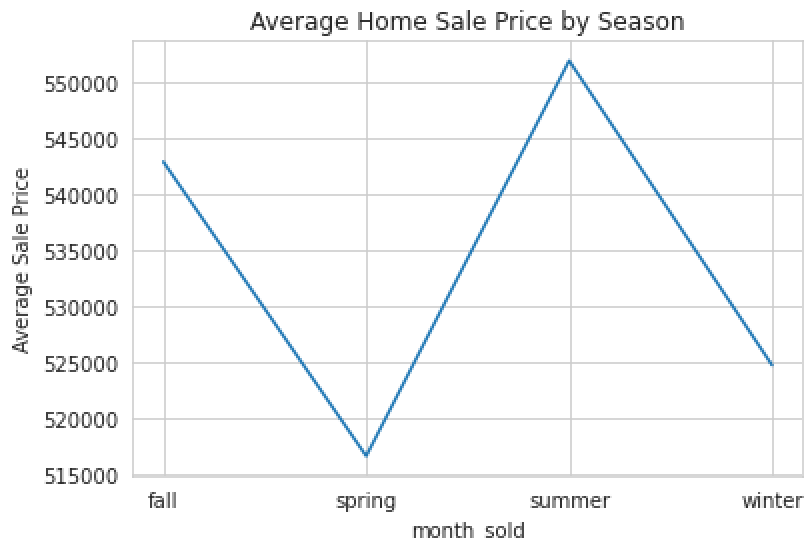
Data Cleaning and EDA

Minimal data cleaning was required, however there was a need to manage missing values, duplicates, and outliers. The exploratory data analysis(EDA) sought answers to the following questions:

How does the price vary with the month sold?

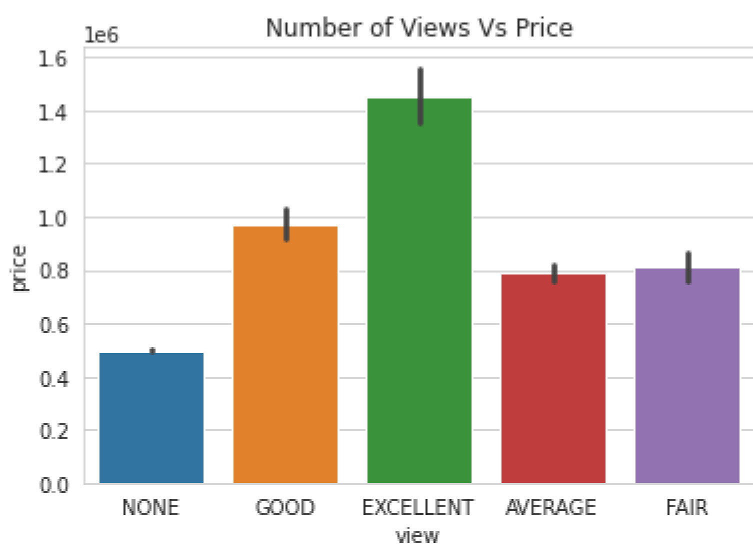
How does the view affect price?

Which factors are most correlated with price?



The above screenshot shows the home sale prices within our first time home buyers criteria by the season the house was sold. The seasons are split between spring , summer , fall and winter.

We see that prices start rising from spring to summer and start dropping during summer to spring.



From the above screenshot houses with better views have higher prices.

Based on the questions we provided, here are some possible answers that the exploratory data analysis (EDA) may have found:

How does the price vary with the month sold?

There is a seasonal effect on the home sale prices, with prices being higher during certain months and lower during others. For example, the analysis may have found that prices tend to be higher in the spring and summer months when the weather is better and people are more likely to be interested in buying a house. On the other hand, prices may be lower in the winter months when there are fewer buyers in the market.

How does the view affect the price?

The better the view the higher the price.

Which factors are most correlated with price?

Certain factors are more strongly correlated with the sale price of a house than others. For example, the size of the house (measured in square feet) may be strongly correlated with the sale price, with larger houses selling for higher prices. Other factors that may be strongly correlated with price include the location of the house (e.g., proximity to schools or public transportation), the number of bedrooms and bathrooms, and the condition of the house.

Modeling and Evaluation

Screenshots for some of our code

XG Boost Regressor

```
[341]: from xgboost import XGBRegressor
xgb = XGBRegressor(n_estimators=1000, learning_rate=0.01)
xgb.fit(X_train, y_train)
predictions = xgb.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(xgb)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "XGBRegressor","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_cross_val}
models = models.append(new_row, ignore_index=True)
```

```
MAE: 153913.37405237267
MSE: 63215152224.8582
RMSE: 251426.23615060182
R2 Score: 0.5145372593211468
------------------------------
RMSE Cross-Validation: 244244.36949240445
```

Random Forest Regressor

```
]: from sklearn.ensemble import RandomForestRegressor
random_forest = RandomForestRegressor(n_estimators=100)
random_forest.fit(X_train, y_train)
predictions = random_forest.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(random_forest)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "RandomForestRegressor","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_cross_val}
models = models.append(new_row, ignore_index=True)
```

```
MAE: 159514.81383475527
MSE: 62146803995.02415
RMSE: 249292.60718084712
R2 Score: 0.5227416730004782
------------------------------
RMSE Cross-Validation: 247600.34588183803
```

Ridge Regression

```
[337]: from sklearn.linear_model import Ridge
ridge = Ridge()
ridge.fit(X_train, y_train)
predictions = ridge.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(ridge)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "Ridge","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_cross_val}
models = models.append(new_row, ignore_index=True)
```

```
MAE: 168826.60557282393
MSE: 65185562261.50544
RMSE: 255314.63385694413
R2 Score: 0.49940543375424207
------------------------------
RMSE Cross-Validation: 257689.51956076143
```

Lasso Regression

[338]:
```
from sklearn.linear_model import Lasso
lasso = Lasso()
lasso.fit(X_train, y_train)
predictions = lasso.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(lasso)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "Lasso","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_cross_val}
models = models.append(new_row, ignore_index=True)
```

```
MAE: 168826.7392325604
MSE: 65185737151.21127
RMSE: 255314.97635511175
R2 Score: 0.49940409068327063
------------------------------
RMSE Cross-Validation: 257689.54636797262
```

Linear Regression

[336]:
```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
predictions = lin_reg.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(lin_reg)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "LinearRegression","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_cross_val}
models = models.append(new_row, ignore_index=True)
```

```
MAE: 168826.95509923415
MSE: 65185818516.93465
RMSE: 255315.1356988744
R2 Score: 0.4994034658326505
------------------------------
RMSE Cross-Validation: 257689.58087194673
```

Support Vector Machines

```
[342]: from sklearn.svm import SVR
svr = SVR(C=100000)
svr.fit(X_train, y_train)
predictions = svr.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(svr)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "SVR","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_cross_val}
models = models.append(new_row, ignore_index=True)
```

```
MAE: 164818.0298561094
MSE: 78277475481.65025
RMSE: 279781.1206669425
R2 Score: 0.39886567629270786
------------------------------
RMSE Cross-Validation: 285141.669698903
```

Project Milestones

There are some achievements that mark significant progress towards project completion. They are listed below:

Milestones:

Milestone 1: Data Preprocessing

Timeline: 1 week

Tasks:

- Import the Zillow dataset.
- Identify and handle missing data.
- Convert categorical features to numerical values.
- Exploratory data analysis (EDA).

Milestone 2: Feature Selection

Timeline: 1 week

Tasks:

- Perform feature selection techniques to identify the most relevant features for predicting house prices.
- Analyze the relationship between features and house prices.
- Evaluate the impact of feature selection on model performance.

Milestone 3: Model Training and Evaluation

Timeline: 2 weeks

Tasks:

- Train various regression models (e.g., linear regression, random forest regression, XG Boost) using the selected features.
- Evaluate the performance of each model using metrics such as mean squared error (MSE) and R-squared.
- Select the best performing model based on evaluation metrics.

Incremental Features:

Incorporate additional data sources:

- Data sources: Real estate market trends, local economic indicators, demographic data.
- Benefits: Improve the accuracy and comprehensiveness of the model.

We will use Linear Regression, Random Forest Regression, Ridge Regression, Lasso Regression, Support Vector Regression and XG Boost Regression for our modeling and select which best suits our data set.

Based on the provided evaluation results, here's a summary of the performance metrics for each model:

Based on the RMSE Cross Validation metric, the XG Boost Regressor and Random Forest Regression model has the lowest value, followed by the Linear Regression and Ridge Regression models. The Linear Regression and SVR Regression models have relatively higher RMSE cross validation values.

Based on the RSME metric, the XG Boost Regression model and Random Forest Regression model has the lowest value, followed by the Random Forest Regression model.Linear Regression, Ridge Regression, and Lasso Regression models have relatively higher RSME values.

Based on the provided evaluation results, the Random Forest Regression model and XGBoost Regression model generally have lower MSE and MAE values compared to the other models. This suggests that they may have better predictive performance.

Regression Models	MAE	MSE	RSME	R2 Score	RSME (Cross-Validation)
XG Boost	153913.374052	6.321515e+10	251426.236151	0.514537	244244.369492
Random Forest	159772.297337	6.265113e+10	250302.083273	0.518869	247275.352598
Ridge	168826.605573	6.518556e+10	255314.633857	0.499405	257689.519561
Lasso	168826.739233	6.518574e+10	255314.976355	0.499404	257689.546368
Linear	168826.955099	6.518582e+10	255315.135699	0.499403	257689.580872
SVR	164818.029856	7.827748e+10	279781.120667	0.398866	285141.669699

Result and Conclusion:

1. Based on the evaluation results, the Random Forest Regression model and XGBoost Regression model show promising performance in terms of MSE and MAE. Further analysis and comparison of these models, considering additional factors, can help determine the most suitable model for your dataset and prediction task.

2. The development of an accurate house price prediction model can provide substantial benefits to a Real Estate Agency and its clients. It can enhance customer satisfaction by providing more precise price estimates and valuable insights into the factors influencing property values. Additionally, the model can support informed decision-making regarding renovations or improvements, leading to increased sales and improved outcomes for homeowners and the agency alike.

Repository /Archive:

https://github.com/Achyuth123456/Group-7_5502.git

Appendix:

Code

```
#import important libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")
%matplotlib inline

from scipy import stats
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.formula.api import ols
import statsmodels.stats.api as sms
from scipy.stats import norm
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings(action= 'ignore', category= UserWarning)
import folium
#code to display all the columns without truncation
pd.set_option('display.max_columns', None)
# Reading the dataset
df = pd.read_csv("zillow_house_data.csv")
#Display the first few rows in the dataframe
df.head()
#Display the last few rows in the dataframe
df.tail()
#Display of the number of rows and columns in the dataframe
df.shape
#Summary of the dataframe
df.info()
#plotting histograms
def histo(df):
    return df.hist(figsize=(20,20))
histo(df);
#summary statistics of the numerical columns in a dataframe
def data_description(df):
    return (df.drop('id', axis = 1)).describe()
data_description(df)
list(df['sqft_basement'][:10])
def clean_basement(col):
    df[col] = df[col].replace('?', '0.0')
    df[col] = df[col].astype(float)
```

```
    print(list(df['sqft_basement'][:10]))
clean_basement('sqft_basement')
def renaming_columns(df,cols_old,cols_new):
    return df.rename(columns={cols_old: cols_new
    }, inplace=True)
renaming_columns(df,'date','date_sold')
#The following function returns all columns with missing values alongside the quantity and percentage

def missing_values(data):
    """A simple function to identify data has missing values"""
    # identify the total missing values per column
    # sort in order
    miss = data.isnull().sum().sort_values(ascending = False)

    # calculate percentage of the missing values
    percentage_miss = (round((data.isnull().sum() / len(data)*100),2)).sort_values(ascending = False)

    # store in a dataframe
    missing = pd.DataFrame({"Missing Values": miss, "Percentage(%)": percentage_miss})
    # remove values that are missing
    missing.drop(missing[missing["Percentage(%)"] == 0].index, inplace = True)

    return missing


missing_data = missing_values(df)
missing_data
df['yr_renovated'].value_counts()
df['yr_renovated'].fillna(value= 0.0, inplace=True)
df['view'].fillna(value='NONE', inplace=True)
#Count of the unique values in the waterfront column
df.waterfront.value_counts(normalize=True)
# replacing the missing values using their probability
def replace_missing_val(df,cols):
    df[cols] = df[cols].fillna(value=(np.random.choice(['YES', 'NO'], p=[0.01, 0.99])))
    # checking if the missing values have been replaced
    print('There are now', df[cols].isnull().sum(), 'missing values in waterfront column')
replace_missing_val(df,'waterfront')
#finding total number of duplicates
# Duplicated entries
def identify_duplicates(df):
    """Simple function to identify any duplicates"""
    # identify the duplicates (dataframename.duplicated() , can add .sum() to get total count)
    # empty list to store Bool results from duplicated
    duplicates = []
    for i in df.duplicated():
        duplicates.append(i)
    # identify if there is any duplicates. (If there is any we expect a True value in the list duplicates)
    duplicates_set = set(duplicates)
```

```
    if (len(duplicates_set) == 1):
        print("The Data has no duplicates")
    else:
        no_true = 0
        for val in duplicates:
            if (val == True):
                no_true += 1
        # percentage of the data represented by duplicates
        duplicates_percentage = np.round(((no_true / len(df)) * 100), 3)
        print(f"The Data has {no_true} duplicated rows.\nThis constitutes {duplicates_percentage}% of the data set.")
identify_duplicates(df)
# checking for duplicates in unique columns
def unique_column_duplicates(df, column):
    df.id.duplicated().sum()
    df = pd.DataFrame(df)
# count number of duplicated values in the 'id' column
num_duplicates = df['id'].duplicated().sum()
print('Number of duplicates in "id" column:', num_duplicates)
#Lets see the rows that have duplicates as per the id column
pd.concat( g for _,g in df.groupby("id") if len(g) > 1).head()
#summary statistics of the numerical columns in a dataframe
df.describe()
# we can also use a box plot to identify outliers
def create_boxplot(dataframe, x_col):
    sns.boxplot(data=dataframe, x=x_col)
create_boxplot(df, "bedrooms");
#let's count the number of occurrences of each unique value in the 'bedrooms' column
df['bedrooms'].value_counts()
#Let's only show the row where the value in the 'bedrooms' column is equal to 33.
df[df['bedrooms'] == 33]
# Let's drop the 33 bedroom row
df.drop(index=15856, inplace=True)

#And verify that it is gone
df['bedrooms'].value_counts()
#Checking for outliers in bathrooms
df['bathrooms'].value_counts()
df.isnull().sum()
#Extracting an individual month column
df['month_sold'] = pd.DatetimeIndex(df['date_sold']).month

# set columns for seasons sold
df.loc[(df['month_sold'] >= 1) & (df['month_sold'] <= 3), 'season'] = 'spring'
df.loc[(df['month_sold'] >= 3) & (df['month_sold'] <= 6), 'season'] = 'summer'
df.loc[(df['month_sold'] >= 6) & (df['month_sold'] <= 9), 'season'] = 'fall'
df.loc[(df['month_sold'] >= 9) & (df['month_sold'] == 12), 'season'] = 'winter'
#Plotting a line graph
plt.plot(df.groupby('season')['price'].mean().round(2))
```

```
plt.title('Average Home Sale Price by Season')
plt.ylabel('Average Sale Price')
plt.xlabel('month_sold');
#Plotting a Scatter plot for age and price
year_sold = pd.DatetimeIndex(df['date_sold']).year
age =year_sold-df['yr_built']
df['age'] = age
plt.scatter(age, df['price'])
plt.xlabel('Age')
plt.ylabel('Price')
plt.title('Relationship between Age and Price');
#Extracting value counts
df['view'].value_counts()
#Plotting a barplot of 'view' against 'price'
sns.barplot(x=df['view'], y=df['price']).set(title='Number of Views Vs Price');
df['condition'].value_counts()
plt.scatter(df['condition'], df['price'], color='brown')
plt.title('Relationship between price and condition')
plt.xlabel('Condition rating')
plt.ylabel('Price')
plt.grid(color='purple',
        alpha=0.1,
        linestyle='-.',
        linewidth=2);
# Lets create a heatmap to check for correlations
sns.pairplot(data=df, x_vars=['sqft_lot','sqft_above','sqft_living','sqft_living15','sqft_lot15'], y_vars=["price"]);

corr = df.corr()
plt.figure(figsize=(7,6))
mask = np.triu(np.ones_like(corr, dtype=bool))
sns.heatmap(df.corr(), cmap= 'coolwarm', mask= mask, linewidth= 1)
plt.title("Correlation between features",weight='bold',fontsize=18)
plt.show()

fig, ax = plt.subplots(figsize=(15,8))
ax= sns.heatmap(df.corr(), annot=True, cmap="viridis")
plt.title("Correlation between features",weight='bold',fontsize=18);
#The following function displays columns that are highly correlated with one other

def check_multiCorr(df):
    num_cols = df.select_dtypes(include=['float64','int64'])
    ncc = list(num_cols.columns)
    for col1 in ncc:
        for col2 in ncc:
            if col1 != col2:
                correlation = df[col1].corr(df[col2])
                if correlation > 0.75:
                    print(col1, " is correlated with ", col2, "by", correlation)
```

```
            ncc.remove(col1)
check_multiCorr(df)
df_num = df.select_dtypes(include=['float64', 'int64'])
df_cat = df.select_dtypes(include=['object'])
df['view'] = df['view'].replace(regex='GOOD', value="AVERAGE")
df['view'] = df['view'].replace(regex='NONE', value="POOR")
df['view'] = df['view'].replace(regex='FAIR', value="AVERAGE")
df['view'].hist();
def bin_column(data, col):
    list_years= list(df[col].unique())
    for x in list_years:
        if x !=0:
            df[col] = df[col].replace(x ,"YES")
        else:
            df[col] = df['yr_renovated'].replace(x,"NO")

bin_column(df, 'yr_renovated')
fig, ax = plt.subplots(figsize=(4,5))
ax.hist(df['yr_renovated'], color='purple');
ax.set_title("Renovated vs Not Renovated");
df_cat.head()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
def rmse_cv(model):
    rmse = np.sqrt(-cross_val_score(model, X, y, scoring="neg_mean_squared_error", cv=5)).mean()
    return rmse


def evaluation(y, predictions):
    mae = mean_absolute_error(y, predictions)
    mse = mean_squared_error(y, predictions)
    rmse = np.sqrt(mean_squared_error(y, predictions))
    r_squared = r2_score(y, predictions)
    return mae, mse, rmse, r_squared
models = pd.DataFrame(columns=["Model","MAE","MSE","RMSE","R2 Score","RMSE (Cross-Validation)"])
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
predictions = lin_reg.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(lin_reg)
print("RMSE Cross-Validation:", rmse_cross_val)
```

```
new_row = {"Model": "LinearRegression","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_cross_val}
models = models.append(new_row, ignore_index=True)
from sklearn.linear_model import Ridge
ridge = Ridge()
ridge.fit(X_train, y_train)
predictions = ridge.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(ridge)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "Ridge","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_cross_val}
models = models.append(new_row, ignore_index=True)
from sklearn.linear_model import Lasso
lasso = Lasso()
lasso.fit(X_train, y_train)
predictions = lasso.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(lasso)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "Lasso","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_cross_val}
models = models.append(new_row, ignore_index=True)
from sklearn.ensemble import RandomForestRegressor
random_forest = RandomForestRegressor(n_estimators=100)
random_forest.fit(X_train, y_train)
predictions = random_forest.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(random_forest)
```

```
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "RandomForestRegressor","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_cross_val}
models = models.append(new_row, ignore_index=True)
from xgboost import XGBRegressor
xgb = XGBRegressor(n_estimators=1000, learning_rate=0.01)
xgb.fit(X_train, y_train)
predictions = xgb.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(xgb)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "XGBRegressor","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_cross_val}
models = models.append(new_row, ignore_index=True)
from sklearn.svm import SVR
svr = SVR(C=100000)
svr.fit(X_train, y_train)
predictions = svr.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(svr)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "SVR","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE (Cross-Validation)": rmse_cross_val}
models = models.append(new_row, ignore_index=True)
models.sort_values(by="RMSE (Cross-Validation)")
plt.figure(figsize=(12,8))
colors = ['aquamarine', 'yellow', 'red', 'pink', 'blue', 'green']
sns.barplot(x=models["Model"], y=models["RMSE (Cross-Validation)"], palette=colors, alpha = 0.8)

plt.title("Models' RMSE Scores (Cross-Validated)", size=15)
plt.xticks(rotation=30, size=12)
plt.show()

```

References

- We took Dataset from 'Zillow'.
  https://www.zillow.com
- We learned Data Cleaning and Preprocessing from 'medium' website.
  https://medium.com/analytics-vidhya/data-cleaning-and-preprocessing-a4b751f4066f
- We referenced our project from 'the clever programmer' website.
  https://thecleverprogrammer.com/2022/08/15/house-rent-prediction-with-machine-learning/
- We referenced the Numpy and Pandas syntax from 'codecademy' website for our project.
  https://www.codecademy.com/article/introduction-to-numpy-and-pandas

Future Work

- Improve accuracy by reducing noise in the data to improve the accuracy of our predictions.
- Look at trends: which house attributes contribute to a higher priced home?
- Simplify model: group low-impact categorical variables. Identify categorical variables that have a minimal impact on the predictions and explore options to group them together or remove them from the model.