Final Project Submission

Please fill out:

- Student name: GROUP 9
- Student pace: part time
- Scheduled project review date/time:
- Instructor name: William Okomba
- Blog post URL:

Project Overview

XYZ real estate agency, nestled in the heart of a vibrant northwestern county, serves as the gateway to turning homeownership dreams into reality. With unwavering commitment to excellence and data-driven strategies, the agency seeks to be a pioneering force in optimal pricing and successful real estate journeys. It's ultimate goal being to transcend conventional boundaries by harnessing the power of technology and analytical insights to reimagine real estate as we know it.

Business Problem

XYZ real estate agency need help determining the key factors that influence house prices in the area and use this information to guide the agency's pricing of houses. The goal of this project is therefore to develop the best model that will help the agency determine the best housing prices and ultimately increasing their annual sales.

Project Objectives

- To identify key features that significantly influence house prices in the northwestern county.
- To develop an optimal pricing strategy using a robust multiple linear regression model.
- To identify overpriced or underpriced houses by comparing predicted and actual prices of the houses.
- To help improve the agency's annual revenue by leveraging the analytical insights and pricing strategy developed through this project.

About the Dataset

This dataset contains house sale prices for King County,USA which includes Seattle. It includes homes sold between May 2014 and May 2015.

Data description report

- The dataset contains 21 columns and 21,597 rows. This means there are 21 different variables each with 21,597 records.
- The dataset consists of three main data types; float, integer and object.
- Three columns have missing values:
  - The waterfront column has 2376 missing values, which is about 11% of the data.
  - The view column has 63 missing values, which is about 0.3% of the data.

    - The yr_renovated column has 3842 missing values, which is about 18% of the data.
- There are no duplicated rows in this dataset.
- Finally, we used a histograms to help visualize the distribution of our numerical data and provide insights into its shape, central tendency, and spread.

Getting Started...

In [1]:

```
#Importing the relevant libraries

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from scipy.stats import pearsonr
from sklearn.preprocessing import MinMaxScaler
from statsmodels.formula.api import ols
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
import sklearn.metrics as metrics
from random import gauss
from mpl_toolkits.mplot3d import Axes3D
from scipy import stats as stats
%matplotlib inline
import statsmodels.formula.api as smf
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.model_selection import train_test_split
from math import sqrt
import sklearn.metrics as metrics
from sklearn.metrics import mean_squared_error, make_scorer
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
#Loading the data and viewing a random sample of 5 rows
df = pd.read_csv('data/kc_house_data.csv')
df.sample(10)
```

```
---------------------------------------------------------------------------
-
FileNotFoundError                         Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_3684\1085223007.py in <module>
      1 #Loading the data and viewing a random sample of 5 rows
----> 2 df = pd.read_csv('data/kc_house_data.csv')
      3 df.sample(10)

~\anaconda3\lib\site-packages\pandas\util\_decorators.py in wrapper(*args,
**kwargs)
    309                     stacklevel=stacklevel,
    310                 )
--> 311             return func(*args, **kwargs)
    312
    313         return wrapper

~\anaconda3\lib\site-packages\pandas\io\parsers\readers.py in read_csv(fil
epath_or_buffer, sep, delimiter, header, names, index_col, usecols, squeez
e, prefix, mangle_dupe_cols, dtype, engine, converters, true_values, false
_values, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_de
fault_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetim
e_format, keep_date_col, date_parser, dayfirst, cache_dates, iterator, chu
nksize, compression, thousands, decimal, lineterminator, quotechar, quotin
g, doublequote, escapechar, comment, encoding, encoding_errors, dialect, e
rror_bad_lines, warn_bad_lines, on_bad_lines, delim_whitespace, low_memor
y, memory_map, float_precision, storage_options)
    676     kwds.update(kwds_defaults)
    677
--> 678     return _read(filepath_or_buffer, kwds)
    679
    680

~\anaconda3\lib\site-packages\pandas\io\parsers\readers.py in _read(filepa
th_or_buffer, kwds)
    573
    574     # Create the parser.
--> 575     parser = TextFileReader(filepath_or_buffer, **kwds)
    576
    577     if chunksize or iterator:

~\anaconda3\lib\site-packages\pandas\io\parsers\readers.py in __init__(sel
f, f, engine, **kwds)
    930
    931         self.handles: IOHandles | None = None
--> 932         self._engine = self._make_engine(f, self.engine)
    933
    934     def close(self):

~\anaconda3\lib\site-packages\pandas\io\parsers\readers.py in _make_engine
(self, f, engine)
   1214             # "Union[str, PathLike[str], ReadCsvBuffer[bytes], Rea
dCsvBuffer[str]]"
   1215             # , "str", "bool", "Any", "Any", "Any", "Any", "Any"
-> 1216             self.handles = get_handle(  # type: ignore[call-overlo
ad]
   1217                 f,
   1218                 mode,

~\anaconda3\lib\site-packages\pandas\io\common.py in get_handle(path_or_bu
f, mode, encoding, compression, memory_map, is_text, errors, storage_optio
```

```
ns)
    784         if ioargs.encoding and "b" not in ioargs.mode:
    785             # Encoding
--> 786             handle = open(
    787                 handle,
    788                 ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory: 'data/kc_house_dat
a.csv'
```

In [ ]:

```
#Viewing the last 5 rows
df.tail(10)
```

Data Understanding

Exploring the data to get a glimpse of:

- Shape of the data
- The column names
- The data types
- Statistical summary of the data
- The null values
- The duplicates

In [ ]:

```
# An overview of the data
df.info()
```

In [ ]:

```
# Shape of the data
df.shape
```

In [ ]:

```
# A view of the columns
df.columns
```

In [ ]:

```
# A statistical summary of the data
df.describe()
```

In [ ]:

```
#Checking for missing values in our data set
df.isna().sum()
```

In [ ]:

```
# Calculating the percentage of missing values in each column
missing_percentage = (df.isna().sum() / len(df)) * 100
print(missing_percentage)
```

In [ ]:

```
# Checking for duplicates
df.duplicated().sum()
```

In [ ]:

```
# PLOT DATA HISTOGRAM to get an overview of the distribution
df.hist(figsize=(30,15), edgecolor = 'black');
```

Data Cleaning

- Dropping missing values. All 3 columns with missing values had null values below 20% and so we decided to drop the rows with missing values as this would not have a large effect on the dataset.
- Converting data types:
  - Converted the 'grade' column from object type to integer by splitting and creating a new column 'numerical_grade'.
  - Converted the 'waterfront' column from object type to integer.
  - Converted the 'date' column from object type to integer by defining a lambda function and then created a new column 'year' using pd.DatetimeIndex function with only the year component from the 'date' column
- Finally, created a subset of the main dataset consisting of features to use during the analysis.

In [ ]:

```
#Dropping the missing values in waterfront, yr_renovated and view columns
df.dropna(subset = ["waterfront"], inplace = True)
df.dropna(subset = ["yr_renovated"], inplace = True)
df.dropna(subset = ["view"], inplace = True)
```

In [ ]:

```
# Rechecking for missing values
df.isna().sum()
```

In [ ]:

```
# Reindexing after dropping and removing the missing values
df.reset_index(drop=True, inplace=True)
#Checking for the columns of the dataframe
df.columns
```

In [ ]:

```
#Spliting grade and creating a new column 'rating' so that we can see its numeric correla
df[["numerical_grade", "rating"]] = df["grade"].str.split(n=1, expand=True)
df["numerical_grade"] = df["numerical_grade"].astype(int)
df.drop('grade', axis=1, inplace=True)
```

In [ ]:

```
# Checking correlation against price to determine the features to use in our analysis
df.corr()['price']
```

In [ ]:

```
# A copy of the data frame with the features that we are working with
df_subset = df[[ 'price', 'bedrooms', 'bathrooms', 'sqft_living',  'waterfront','sqft_abo
df_subset
```

In [ ]:

```
pd.options.display.float_format = lambda x : '{:.0f}'.format(x) if int(x) == x else '{:,.
df_subset['year'] = pd.DatetimeIndex(df_subset['date']).year
df_subset.drop("date",axis = 1,inplace =True)
```

In [ ]:

```
#convert waterfront to numerical values
df_subset["waterfront"] = df_subset["waterfront"].map({'NO': 0, 'YES': 1})
df_subset
```

In [ ]:

```
df_subset.corr()['price']
```

Exploratory Data Analysis

Univariate Analysis

- **Numerical Variables**

In [ ]:

```
#Plotting box plots to check wether we have outliers in the data set
#Columns for our box plots
columns = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_above', 'sqft_living15'

# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=len(columns), figsize=(12, 4))

# Generate box plots for each column
for i, column in enumerate(columns):
    sns.boxplot(data=df_subset[column], ax=axes[i])
    axes[i].set_title(f'Box plot - {column}', fontsize=10)
    axes[i].set_xlabel(column, fontsize=8)

# Adjust the layout and spacing
plt.tight_layout()

# Display the plots
plt.show()
```

In [ ]:

```
# Statistical view to confirm the outliers plotted above
#df.describe()
min_max_summary = df_subset.agg(['min', 'max'])
min_max_summary
```

In [ ]:

```
# Defining function to remove outliers from the subset
def remove_outliers(df_subset):
    '''removes entries with z-score above 3 for specific columns'''
    variables = ['price','bedrooms', 'bathrooms', 'sqft_living', 'sqft_above', 'sqft_livi

    for variable in variables:
        df_subset= df_subset[np.abs(df_subset[variable]-df_subset[variable].mean()) <= (3

    return df_subset
#Removing the outliers which have a standard deviation greater than 3
df_subset = remove_outliers(df_subset)
```

In [ ]:

```
# Statistical view to recheck for outliers after cleaning
#df_subset.describe()
min_max_summary = df_subset.agg(['min', 'max'])
min_max_summary
```

In [ ]:

```
# Plotting to recheck for outliers in the dataset after cleaning
# Define the columns for box plots
columns = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_above', 'sqft_living15'

# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=len(columns), figsize=(12, 4))

# Generate box plots for each column
for i, column in enumerate(columns):
    sns.boxplot(data= df_subset[column], ax=axes[i])
    axes[i].set_title(f'Box plot - {column}', fontsize=10)
    axes[i].set_xlabel(column, fontsize=8)

# Adjust the layout and spacing
plt.tight_layout()

# Display the plots
plt.show()
```

In [ ]:

```
# Plotting histograms to check for the frequencies of the dataset after cleaning
# Define the columns for histograms
columns = ['price', 'bedrooms', 'bathrooms', 'sqft_living','numerical_grade', 'sqft_above

# Calculate the number of rows and columns for the subplot grid
n_cols = 3
n_rows = (len(columns) - 1) // n_cols + 1

# Create subplots
fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(24, 4*n_rows))

# Flatten the axes array
axes = axes.flatten()

# Generate histograms for each column
for i, column in enumerate(columns):
    sns.histplot(data=df_subset[column], ax=axes[i])
    axes[i].set_title(f'Histogram - {column}', fontsize=15)
    axes[i].set_xlabel(column, fontsize=15)

# Remove any unused subplots
for j in range(len(columns), n_rows * n_cols):
    fig.delaxes(axes[j])

# Adjust the layout and spacing
plt.tight_layout()

# Display the plots
plt.show()
```

In [ ]:

```
#A statistical view of skewness
df_subset.skew()
```

- **Categorical Variables**

In [ ]:

```
# Define the columns
columns = ['waterfront','numerical_grade','year']

# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=len(columns), figsize=(12, 3))

# Perform univariate analysis for each column
for i, column in enumerate(columns):
    counts = df_subset[column].value_counts().sort_values(ascending=False)
    sns.barplot(x=counts, y=counts.index, ax=axes[i], orient='h')
    axes[i].set_title(f'{column}', fontsize=10)
    axes[i].set_xlabel('Count', fontsize=8)
    axes[i].set_ylabel(column, fontsize=8)

# Adjust the layout and spacing
plt.tight_layout()

# Display the plots
plt.show()
```

- **For our univariate analysis**, first we **plotted box plots to check for outliers** in our features in which we found all our features to have extreme values that could potentially violate the normalcy distribution of our data and result in an inaccurate predictive model. We therefore decided to remove these outliers.
- We also **plotted histograms to provide insights into the frequency distribution of our features** and to provide some visual indications of skewness in our data. While this is not the precise measure of skewness, we notice that squarefoot living has a somewhat normal distribution while price, sqft_above, sqft_living15 are positively skewed.
- **For the categorical variables**, we plotted bar plots to get a sense of the distribution of our categorical data. The bar plot suggest majority of houses have no waterfronts with only a few houses having waterfronts. It will be interesting to see what impact this has on pricing further into our analysis.
- The numerical_grade bar plot indicates majority of the houses are grade 7 which is average rating while very few are above 10. We will also look at how grading potentially affects house pricing.
- With the year variable we also noticed that more houses were sold in 2014 compared to 2015

Bivariate Analysis

- **Numerical Variables**

In [ ]:

```
#for numeric values
# Select the desired features and the target variable
features = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_above', 'sqft_living15','numeri
target = 'price'

# Calculate the number of rows and columns for the subplot grid
n_rows = 3
n_cols = 3

# Create subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 10))

# Flatten the axes array
axes = axes.flatten()

# Plotting bivariate relationships
for i, feature in enumerate(features):
    axes[i].scatter(df_subset[feature], df_subset[target], alpha=0.5)
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel(target)
    axes[i].set_title(f'{feature} vs {target}')

# Remove any unused subplots
if len(features) < n_rows * n_cols:
    for j in range(len(features), n_rows * n_cols):
        fig.delaxes(axes[j])

# Adjust the layout and spacing
plt.tight_layout()

# Display the plots
plt.show()
```

- **Categorical variables**

In [ ]:

```
#categorical values against price
# Define the columns
columns = ['waterfront','year','zipcode']

# Create bar plots
fig, axes = plt.subplots(nrows=1, ncols=len(columns), figsize=(12, 3))

# Generate bar plots for each column
for i, column in enumerate(columns):
    grouped_data = df_subset.groupby(column)['price'].mean().sort_values(ascending=False)
    axes[i].barh(grouped_data.index, grouped_data.values)
    axes[i].set_xlabel('Average Price')
    axes[i].set_ylabel(column)
    axes[i].set_title(f'Average Price by {column}')

# Adjust the layout and spacing
plt.tight_layout()

# Display the plots
plt.show()
```

- For our **Bivariate analyisis** we plotted scatter plots and bar plots for our numerical and categorical variables.
- The scatter plots were a representation of the relationship between the **numerical values**(bedrooms,bathrooms,sqft_living,sqft_above, sqft_living15,numerical_grade,floors,year,zipcode) and the **price**. According to our plots the numerical variables each have a unique effect on the house pricing.
- The bar plots represent the relationship between our **categorical variables**(waterfront, year built and zipcode) and the **price**. These plots show how the categorical values individually affect the prices of houses

Multivariate Analysis

In [ ]:

```
# Correlation graph
plt.figure(figsize=(20,10))
corr_matrix = df[['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long',
       'sqft_living15', 'sqft_lot15', 'numerical_grade', 'rating']].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')

plt.show()
```

- Our **Multivariate** analysis evaluates the correlation of multiple variables.The heatmap above displays the correlation of all the columns in our data frame

Modelling

Model 1

Simple linear regression

For our baseline model we chose square foot living as the predictor variable because it has the highst correlation at 0.71.

In [ ]:

```
y = df_subset["price"]
X_baseline = df_subset[["sqft_living"]]
```

In [ ]:

```
baseline_model = sm.OLS(y, sm.add_constant(X_baseline))
baseline_results = baseline_model.fit()

print(baseline_results.summary())
```

- The model is statistically significant overall, with an F-statistic p-value well below 0.05
- R-squared value is 0.372, indicating that approximately 37.2% of the variance in the price can be explained by the sqft_living variable
- The model coefficients (price and sqft_living) are both statistically significant, with t-statistic p-values well below 0.05
- The coefficient for "const" is 9.71e+04, and the coefficient for "sqft_living" is 199.9715. These values indicate that, on average, for each additional square foot of living space, the predicted price increases by approximately 199.9715 units.

In [ ]:

```
#Plotting the fit of the baseline model on the "sqft_living" feature
sm.graphics.plot_fit(baseline_results, "sqft_living")
plt.show()
```

In [ ]:

```
# Line of best fit
fig, ax = plt.subplots()
df_subset.plot.scatter(x="sqft_living", y="price", label="Data points", ax=ax)
sm.graphics.abline_plot(model_results=baseline_results, label="Regression line", ax=ax, c
ax.legend();
```

In [ ]:

```
# Checking for residuals
fig, ax = plt.subplots()
ax.scatter(df_subset["price"], baseline_results.resid)
ax.axhline(y=0,color = "black")
ax.set_xlabel("sqrt frt living")
ax.set_ylabel("residuals")
```

In [ ]:

```
resids_1= baseline_results.resid
```

In [ ]:

```
# a QQ plot to see if an S curve, therefore forms normal distribution
plt.figure(figsize=(4,4))
fig = sm.graphics.qqplot(resids_1, dist=stats.norm, line='45', fit=True)
fig.show()
```

Our model showed right skeweness which means there are extreme values that impact the significance of the model

Multilinear Regression

MODEL 2

Data Preprocessing before modelling

Onehot encoding

In [ ]:

```
#Onehotencoding to change waterfront to a numeric value
waterfront = df_subset[['waterfront']]
ohe = OneHotEncoder(categories="auto", sparse=False, handle_unknown="ignore")
ohe.fit(waterfront)
waterfront_enc = ohe.transform(waterfront)
waterfront_enc = pd.DataFrame(waterfront_enc,
                              columns=['waterfront_NO','waterfront_YES'],
                              index=df_subset.index)
df_subset.drop('waterfront', axis=1, inplace=True)
df_subset= pd.concat([df_subset, waterfront_enc], axis=1)
```

This code performs one-hot encoding on the 'waterfront' column. One-hot encoding is a process of converting categorical variables into a binary representation suitable for machine learning algorithms.

In [ ]:

```
#checking for the columns
df_subset.columns
```

In [ ]:

```
# PREPARATION OF BATHROOMS COLUMNS
df_subset =df_subset[['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_above',
       'sqft_living15', 'numerical_grade', 'floors', 'zipcode',
       'year', 'waterfront_NO', 'waterfront_YES']]  # Keep selected columns

df_subset.loc[df_subset['bathrooms'] <= 1, 'bathrooms'] = 1
df_subset.loc[(df_subset['bathrooms'] > 1) & (df_subset['bathrooms'] <= 2), 'bathrooms']
df_subset.loc[(df_subset['bathrooms'] > 2) & (df_subset['bathrooms'] <= 3), 'bathrooms']
df_subset.loc[(df_subset['bathrooms'] > 3) & (df_subset['bathrooms'] <= 4), 'bathrooms']
df_subset.loc[(df_subset['bathrooms'] > 4) & (df_subset['bathrooms'] <= 5), 'bathrooms']
df_subset.loc[(df_subset['bathrooms'] > 5) & (df_subset['bathrooms'] <= 6), 'bathrooms']
df_subset.loc[(df_subset['bathrooms'] > 6) & (df_subset['bathrooms'] <= 7), 'bathrooms']
df_subset.loc[(df_subset['bathrooms'] > 7) & (df_subset['bathrooms'] <= 8), 'bathrooms']
```

In [ ]:

```
# CHANGING 'BATHROOMS' TO INT64
df_subset['bathrooms'] = df_subset['bathrooms'].astype('Int64')
df_subset["floors"] = df_subset["floors"].round().astype(int)
# ADDING DUMMIES FOR "GRADE", "BEDROOMS" AND "BATHROOMS"
```

In [ ]:

```
#creating dummies for our categorical and continuous  data features
df_dummies = pd.get_dummies(df_subset, columns=['bedrooms', 'bathrooms', 'numerical_grade
df_dummies.columns = [column.replace('.0', '') if column.endswith('.0') else column for c
```

In [ ]:

```
# checking for the new columns created
df_dummies.columns
```

In [ ]:

```
df_dummies.head()
```

Let's now proceed to loading our data in the new model

In [ ]:

```
y = 'price'
X_1 = df_dummies[['sqft_living', 'sqft_above', 'sqft_living15', 'waterfront_NO',
       'waterfront_YES', 'bedrooms_1', 'bedrooms_2', 'bedrooms_3',
       'bedrooms_4', 'bedrooms_5', 'bedrooms_6', 'bathrooms_1', 'bathrooms_2',
       'bathrooms_3', 'bathrooms_4', 'bathrooms_5', 'numerical_grade_5',
       'numerical_grade_6', 'numerical_grade_7', 'numerical_grade_8',
       'numerical_grade_9', 'numerical_grade_10', 'year_2014', 'year_2015',
       'floors_1', 'floors_2', 'floors_3', 'zipcode_98001', 'zipcode_98002',
       'zipcode_98003', 'zipcode_98004', 'zipcode_98005', 'zipcode_98006',
       'zipcode_98007', 'zipcode_98008', 'zipcode_98010', 'zipcode_98011',
       'zipcode_98014', 'zipcode_98019', 'zipcode_98022', 'zipcode_98023',
       'zipcode_98024', 'zipcode_98027', 'zipcode_98028', 'zipcode_98029',
       'zipcode_98030', 'zipcode_98031', 'zipcode_98032', 'zipcode_98033',
       'zipcode_98034', 'zipcode_98038', 'zipcode_98039', 'zipcode_98040',
       'zipcode_98042', 'zipcode_98045', 'zipcode_98052', 'zipcode_98053',
       'zipcode_98055', 'zipcode_98056', 'zipcode_98058', 'zipcode_98059',
       'zipcode_98065', 'zipcode_98070', 'zipcode_98072', 'zipcode_98074',
       'zipcode_98075', 'zipcode_98077', 'zipcode_98092', 'zipcode_98102',
       'zipcode_98103', 'zipcode_98105', 'zipcode_98106', 'zipcode_98107',
       'zipcode_98108', 'zipcode_98109', 'zipcode_98112', 'zipcode_98115',
       'zipcode_98116', 'zipcode_98117', 'zipcode_98118', 'zipcode_98119',
       'zipcode_98122', 'zipcode_98125', 'zipcode_98126', 'zipcode_98133',
       'zipcode_98136', 'zipcode_98144', 'zipcode_98146', 'zipcode_98148',
       'zipcode_98155', 'zipcode_98166', 'zipcode_98168', 'zipcode_98177',
       'zipcode_98178', 'zipcode_98188', 'zipcode_98198', 'zipcode_98199']]
all_columns = '+'.join(X_1.columns)
multi_formula_1 = y + '~' + all_columns
```

In [ ]:

```
model_1 = ols(formula=multi_formula_1, data=df_dummies).fit()
print(model_1.summary())
```

Interpretation

- R-squared: The R-squared value of 0.808 indicates that approximately 80.8% of the variance in the dependent variable (price) can be explained by the independent variables included in the model.
- Model significance: The F-statistic of 692.2 with a probability (p-value) of 0.00 suggests that the overall model is statistically significant, meaning that at least one of the independent variables has a significant effect on the dependent variable.
- The coefficient represents the estimated effect of the features on the price.
- The independent variables are represented by different features, such as "sqft_living," "waterfront," "bedrooms," "bathrooms," "numerical_grade," "year," "floors," and "zipcode." Each feature has multiple categories, and the coefficients represent the estimated effect of each category compared to a reference category for example the coefficient for "sqft_living" is 112.0845, indicating that, on average, a one-unit increase in the square footage of living space is associated with an increase of approximately $112,084.50 in the predicted price, holding other variables constant.
- The intercept coefficient is 1.441e+05 (144,100), indicating the estimated price when all other independent variables are zero.

In [ ]:

```
# get MAE to see how much error is in our model
y_predic = model_1.resid
y = np.log(df_dummies['price'])
mae_resid_2 = np.mean(np.abs(y - y_predic))
mae_resid_2
```

- The absolute difference between the predicted values and the actual values is 78325.75392200101. It indicates that the model's predictions have a larger average deviation from the actual values.

In [ ]:

```
# the difference between the actual target values and the predicted target
resids_2 = model_1.resid
resids_2
```

- In regression analysis, the residuals represent the differences between the observed values and the predicted values of the dependent variable.

In [ ]:

```
# check residuals for linearity
plt.figure(figsize=(4,4))
sns.scatterplot(y=y_predic,x=resids_2 );
```

- If the residuals form a straight line with no distinct pattern or curvature, it indicates that the model is performing well and making unbiased predictions.
- This suggests that the model is capturing the underlying relationship between the independent and dependent variables accurately.

In [ ]:

```
# and normality of residuals
plt.figure(figsize=(4,4))
sns.histplot(data=resids_2,bins=30, kde=True);
```

This pattern suggests that, on average, the model predictions are unbiased and do not display any systematic or significant deviation from the true values. In other words, the errors are equally likely to be positive or negative, and they tend to cluster around zero.

In [ ]:

```
# a QQ plot to see if an S curve, therefore forms normal distribution
plt.figure(figsize=(4,4))
fig = sm.graphics.qqplot(resids_2, dist=stats.norm, line='45', fit=True)
fig.show()
```

Our model has improved but still shows right skeweness.

In [ ]:

```
# Checking for residuals to show homoscedasticity
model_1 = sm.OLS(y, X_1)

# Obtain the model results
results = model_1.fit()

# Calculate the residuals
residuals = results.resid
fig, ax = plt.subplots()
ax.scatter(df_dummies["price"], resids_2)
ax.axhline(y=0,color = "black")
ax.set_xlabel("X_1")
ax.set_ylabel("residuals")
```

This model is still skewed and the residuals are not evenly spread along the x-axis this warrants for an improvement of the model.

To attempt to correct this and further improve the model we will do data log transformation in the next model.

MODEL 3

Data Log Transformation

In [ ]:

```
# Log transformation and creation of a new dataframe
columns_to_transform = ['price', 'sqft_living', 'sqft_living15', 'sqft_above']
transformed_data = df_dummies.copy()

transformed_data[columns_to_transform] = np.log(transformed_data[columns_to_transform])
transformed_data.head()
```

In [ ]:

```
#Checking for the frequencies of the dataset
# Define the columns for histograms
columns = ['price', 'sqft_living', 'sqft_living15', 'sqft_above']

# Calculate the number of rows and columns for the subplot grid
n_cols = 3
n_rows = (len(columns) - 1) // n_cols + 1

# Create subplots
fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(24, 4*n_rows))

# Flatten the axes array
axes = axes.flatten()

# Generate histograms for each column
for i, column in enumerate(columns):
    sns.histplot(data=transformed_data[column], ax=axes[i])
    axes[i].set_title(f'Histogram - {column}', fontsize=10)
    axes[i].set_xlabel(column, fontsize=8)

# Remove any unused subplots
for j in range(len(columns), n_rows * n_cols):
    fig.delaxes(axes[j])

# Adjust the layout and spacing
plt.tight_layout()

# Display the plots
plt.show()
```

In [ ]:

```
transformed_data.columns
```

In [ ]:

```
y = 'price'
X_2 = transformed_data[['sqft_living', 'sqft_above', 'sqft_living15', 'waterfront_NO',
       'waterfront_YES', 'bedrooms_1', 'bedrooms_2', 'bedrooms_3',
       'bedrooms_4', 'bedrooms_5', 'bedrooms_6', 'bathrooms_1', 'bathrooms_2',
       'bathrooms_3', 'bathrooms_4', 'bathrooms_5', 'numerical_grade_5',
       'numerical_grade_6', 'numerical_grade_7', 'numerical_grade_8',
       'numerical_grade_9', 'numerical_grade_10', 'year_2014', 'year_2015',
       'floors_1', 'floors_2', 'floors_3', 'zipcode_98001', 'zipcode_98002',
       'zipcode_98003', 'zipcode_98004', 'zipcode_98005', 'zipcode_98006',
       'zipcode_98007', 'zipcode_98008', 'zipcode_98010', 'zipcode_98011',
       'zipcode_98014', 'zipcode_98019', 'zipcode_98022', 'zipcode_98023',
       'zipcode_98024', 'zipcode_98027', 'zipcode_98028', 'zipcode_98029',
       'zipcode_98030', 'zipcode_98031', 'zipcode_98032', 'zipcode_98033',
       'zipcode_98034', 'zipcode_98038', 'zipcode_98039', 'zipcode_98040',
       'zipcode_98042', 'zipcode_98045', 'zipcode_98052', 'zipcode_98053',
       'zipcode_98055', 'zipcode_98056', 'zipcode_98058', 'zipcode_98059',
       'zipcode_98065', 'zipcode_98070', 'zipcode_98072', 'zipcode_98074',
       'zipcode_98075', 'zipcode_98077', 'zipcode_98092', 'zipcode_98102',
       'zipcode_98103', 'zipcode_98105', 'zipcode_98106', 'zipcode_98107',
       'zipcode_98108', 'zipcode_98109', 'zipcode_98112', 'zipcode_98115',
       'zipcode_98116', 'zipcode_98117', 'zipcode_98118', 'zipcode_98119',
       'zipcode_98122', 'zipcode_98125', 'zipcode_98126', 'zipcode_98133',
       'zipcode_98136', 'zipcode_98144', 'zipcode_98146', 'zipcode_98148',
       'zipcode_98155', 'zipcode_98166', 'zipcode_98168', 'zipcode_98177',
       'zipcode_98178', 'zipcode_98188', 'zipcode_98198', 'zipcode_98199']]
all_columns = '+'.join(X_2.columns)
multi_formula_2 = y + '~' + all_columns
```

In [ ]:

```
model_2 = ols(formula=multi_formula_2, data=transformed_data).fit()
print(model_2.summary())
```

Interpretation

1. The R-squared value is 0.843, indicating that approximately 84.3% of the variation in the price can be explained by the independent variables included in the model.
2. The F-statistic is 881.6, and the p-value is 0.00 which indicates that the model is statistically significant.
3. The coefficient represents the estimated effect of the features on the price. For example, the coefficient of "sqft_living" is 0.3747, suggesting that a one-unit increase in the square footage of living space is associated with a 0.3747 increase in the predicted price, holding other variables constant.
4. The Mean Absolute Error between the predicted values and the actual values is 2.56458455201137,indicating that, on average, the model's predictions deviate by approximately 2.56 from the actual values.
5. The residuals consistently remain close to zero.

In [ ]:

```
# get MAE to see how much error is in our model
y_predic = model_2.resid
y = np.log(transformed_data['price'])
mae_resid_2 = np.mean(np.abs(y - y_predic))
mae_resid_2
```

In [ ]:

```
# the difference between the actual target values and the predicted target
resids_3 = model_2.resid
```

In [ ]:

```
# check residuals for linearity
plt.figure(figsize=(4,4))
sns.scatterplot(y=y_predic,x=resids_3 );
```

In [ ]:

```
# and normality of residuals
plt.figure(figsize=(4,4))
sns.histplot(data=resids_3,bins=30, kde=True);
```

This pattern suggests that, on average, the model predictions are unbiased and do not display any systematic or significant deviation from the true values. In other words, the errors are equally likely to be positive or negative, and they tend to cluster around zero.

In [ ]:

```
# a QQ plot to see if an S curve, therefore forms normal distribution
plt.figure(figsize=(4,4))
fig = sm.graphics.qqplot(resids_3, dist=stats.norm, line='45', fit=True)
fig.show()
```

In [ ]:

```
# Checking for residuals
model_2 = sm.OLS(y, X_1)

# Obtain the model results
results = model_2.fit()

# Calculate the residuals
residuals = results.resid
fig, ax = plt.subplots()
ax.scatter(transformed_data["price"], resids_3)
ax.axhline(y=0,color = "black")
ax.set_xlabel("X_1")
ax.set_ylabel("residuals")
```

To further improve the model we will:

1. Drop independent variables that are not statistically significant (['zipcode_98155','zipcode_98077','zipcode_98024','zipcode_98011')

2. Drop predictors that are highly corelated to others eg. 'sqft_above'& 'sqft_living15' for example : ('sqft_above' vs +
3. Drop predictors that are highly corelated to others eg. 'sqft_above'& 'sqft_living15' for example :

Model 4

In [ ]:

```
#Dropping colums that are not statistically significant
#Dropping columns that are highly correlated
transformed_data.drop(['zipcode_98155','zipcode_98077','zipcode_98024','zipcode_98011','s
```

In [ ]:

```
transformed_data.head()
```

In [ ]:

```
#cheking for the correlation of the features
correlation_table = transformed_data[['price','sqft_living','waterfront_NO',
       'waterfront_YES', 'bedrooms_1', 'bedrooms_2', 'bedrooms_3',
       'bedrooms_4', 'bedrooms_5', 'bedrooms_6', 'bathrooms_1', 'bathrooms_2',
       'bathrooms_3', 'bathrooms_4', 'bathrooms_5', 'numerical_grade_5',
       'numerical_grade_6', 'numerical_grade_7', 'numerical_grade_8',
       'numerical_grade_9', 'numerical_grade_10', 'year_2014', 'year_2015',
       'floors_1', 'floors_2', 'floors_3', 'zipcode_98001', 'zipcode_98002',
       'zipcode_98003', 'zipcode_98004', 'zipcode_98005', 'zipcode_98006',
       'zipcode_98007', 'zipcode_98008', 'zipcode_98010',
       'zipcode_98014', 'zipcode_98019', 'zipcode_98022', 'zipcode_98023',
       'zipcode_98027', 'zipcode_98028', 'zipcode_98029',
       'zipcode_98030', 'zipcode_98031', 'zipcode_98032', 'zipcode_98033',
       'zipcode_98034', 'zipcode_98038', 'zipcode_98039', 'zipcode_98040',
       'zipcode_98042', 'zipcode_98045', 'zipcode_98052', 'zipcode_98053',
       'zipcode_98055', 'zipcode_98056', 'zipcode_98058', 'zipcode_98059',
       'zipcode_98065', 'zipcode_98070', 'zipcode_98072', 'zipcode_98074',
       'zipcode_98075', 'zipcode_98092', 'zipcode_98102',
       'zipcode_98103', 'zipcode_98105', 'zipcode_98106', 'zipcode_98107',
       'zipcode_98108', 'zipcode_98109', 'zipcode_98112', 'zipcode_98115',
       'zipcode_98116', 'zipcode_98117', 'zipcode_98118', 'zipcode_98119',
       'zipcode_98122', 'zipcode_98125', 'zipcode_98126', 'zipcode_98133',
       'zipcode_98136', 'zipcode_98144', 'zipcode_98146', 'zipcode_98148',
       'zipcode_98166', 'zipcode_98168', 'zipcode_98177',
       'zipcode_98178', 'zipcode_98188', 'zipcode_98198', 'zipcode_98199']].corr()
correlation_table
```

In [ ]:

```
y = 'price'
X_3 = transformed_data[['sqft_living','waterfront_NO',
       'waterfront_YES', 'bedrooms_1', 'bedrooms_2', 'bedrooms_3',
       'bedrooms_4', 'bedrooms_5', 'bedrooms_6', 'bathrooms_1', 'bathrooms_2',
       'bathrooms_3', 'bathrooms_4', 'bathrooms_5', 'numerical_grade_5',
       'numerical_grade_6', 'numerical_grade_7', 'numerical_grade_8',
       'numerical_grade_9', 'numerical_grade_10', 'year_2014', 'year_2015',
       'floors_1', 'floors_2', 'floors_3', 'zipcode_98001', 'zipcode_98002',
       'zipcode_98003', 'zipcode_98004', 'zipcode_98005', 'zipcode_98006',
       'zipcode_98007', 'zipcode_98008', 'zipcode_98010',
       'zipcode_98014', 'zipcode_98019', 'zipcode_98022', 'zipcode_98023',
       'zipcode_98027', 'zipcode_98028', 'zipcode_98029',
       'zipcode_98030', 'zipcode_98031', 'zipcode_98032', 'zipcode_98033',
       'zipcode_98034', 'zipcode_98038', 'zipcode_98039', 'zipcode_98040',
       'zipcode_98042', 'zipcode_98045', 'zipcode_98052', 'zipcode_98053',
       'zipcode_98055', 'zipcode_98056', 'zipcode_98058', 'zipcode_98059',
       'zipcode_98065', 'zipcode_98070', 'zipcode_98072', 'zipcode_98074',
       'zipcode_98075', 'zipcode_98092', 'zipcode_98102',
       'zipcode_98103', 'zipcode_98105', 'zipcode_98106', 'zipcode_98107',
       'zipcode_98108', 'zipcode_98109', 'zipcode_98112', 'zipcode_98115',
       'zipcode_98116', 'zipcode_98117', 'zipcode_98118', 'zipcode_98119',
       'zipcode_98122', 'zipcode_98125', 'zipcode_98126', 'zipcode_98133',
       'zipcode_98136', 'zipcode_98144', 'zipcode_98146', 'zipcode_98148',
       'zipcode_98166', 'zipcode_98168', 'zipcode_98177',
       'zipcode_98178', 'zipcode_98188', 'zipcode_98198', 'zipcode_98199']]
all_columns = '+'.join(X_3.columns)
multi_formula_3 = y + '~' + all_columns
```

In [ ]:

```
model_3 = ols(formula=multi_formula_3, data=transformed_data).fit()
print(model_3.summary())
```

Interpretation

1. The R-squared value of 0.835 on our last model means that approximately 83.5% of the variation in the price can be explained by the independent variables. There is a slight drop from our previous model because we dropped parameters that were not statistically significant.
2. The F-statistic is 880.6, and the p-value is 0.00 which indicates that the model is statistically significant.
3. The coefficient represents the estimated effect of the features on the price. For example the coefficient for "sqft_living" is 0.5261 , indicating that a one-unit increase in square footage of living space is associated with an estimated increase of 0.5261 in the price, holding other variables constant.

In [ ]:

```
# get MAE to see how much error is in our model
y_predic = model_3.resid
y = np.log(transformed_data['price'])
mae_resid_3 = np.mean(np.abs(y - y_predic))
mae_resid_3
```

The absolute difference between the predicted values and the actual values is 2.5637814367187293.

In [ ]:

```
# the difference between the actual target values and the predicted target
resids_4 = model_3.resid
```

In [ ]:

```
# check residuals for linearity
plt.figure(figsize=(4,4))
sns.scatterplot(y=y_predic,x=resids_4 );
```

In [ ]:

```
# and normality of residuals
plt.figure(figsize=(4,4))
sns.histplot(data=resids_4,bins=30, kde=True);
```

In [ ]:

```
# Checking for residuals/homescendascity
model_3 = sm.OLS(y, X_1)

# Obtain the model results
results = model_3.fit()

# Calculate the residuals
residuals = results.resid
fig, ax = plt.subplots()
ax.scatter(transformed_data["price"], resids_4)
ax.axhline(y=0,color = "black")
ax.set_xlabel("X_1")
ax.set_ylabel("residuals")
```

Residuals consistently remain close to zero

In [ ]:

```
# a QQ plot to see if an S curve, therefore forms normal distribution
plt.figure(figsize=(4,4))
fig = sm.graphics.qqplot(resids_4, dist=stats.norm, line='45', fit=True)
fig.show()
```

After running several models, the above is the model with the highest R2 value, and the best fit of residuals to a normal distribution.

Conclusion

Based on the OLS regression results, the multiple linear regression model developed to identify key factors influencing house prices in the northwestern county has an adjusted R-squared value of 0.8350, indicating that 84% of the variation in house prices can be explained by the selected features. The F-statistic of 880.6 with a corresponding p-value of 0.00 suggests that the overall model is statistically significant.

Several features have a significant influence on house prices in the area. The square footage of the living area (sqft_living), whether the property has a waterfront view (waterfront), the number of bedrooms and bathrooms, the grade of the property(which is basically its design), the year of sale, and various zip codes are among the significant factors affecting house prices.

The developed model can be used to determine optimal pricing strategies for the agency, as it provides coefficients for each feature. By considering these coefficients and the corresponding features of a property, the agency can estimate its price more accurately. Furthermore, by comparing predicted prices with actual prices, the agency can identify overpriced or underpriced houses and make necessary adjustments to maximize sales potential.

The analytical insights and pricing strategy developed through this project can significantly contribute to improving the agency's annual revenue. By leveraging the model's findings and implementing the recommended pricing strategies, the agency can enhance its decision-making process, attract potential buyers, and increase the number of homes sold.

In conclusion, the research successfully achieved its objectives by identifying key features influencing house prices, developing an optimal pricing strategy through multiple linear regression, identifying overpriced or underpriced houses, and providing insights to enhance the agency's annual revenue. By implementing the findings of this research, the XYZ real estate agency can make informed pricing decisions, resulting in increased sales and improved overall performance in the northwestern county housing market.

Recommendation

Based on the analysis conducted and the results obtained from the multiple linear regression model, the following recommendations can be made to XYZ real estate agency:

Focus on key features: The analysis reveals that several key features significantly influence house prices in the northwestern county. These features include square footage of living space, waterfront location, number of bedrooms and bathrooms, property grade, year of construction, and specific zip codes. It is recommended that the agency pays close attention to these factors when determining the pricing of houses.

Optimize pricing strategy: Utilize the developed robust multiple linear regression model to create an optimal pricing strategy. The coefficients obtained from the model provide insights into the impact of each feature on house prices. By considering these coefficients and incorporating market trends, XYZ real estate agency can set competitive and attractive prices for their listed properties. This approach will help maximize the chances of selling homes at desirable price points.

Identify overpriced and underpriced houses: By comparing predicted prices from the model with actual prices of houses, the agency can identify overpriced or underpriced properties in their inventory. This information can guide them in adjusting the prices accordingly to improve sales and ensure competitive pricing in the market.

Leverage analytical insights: The analytical insights derived from this research project can significantly contribute to improving the agency's annual revenue. By incorporating the findings into their decision-making processes, XYZ real estate agency can gain a competitive advantage, attract more buyers, and increase their overall sales volume. It is crucial to regularly update and refine the model as new data becomes available to maintain its accuracy and relevance.

In conclusion, by implementing the recommendations mentioned above and utilizing the developed multiple linear regression model, XYZ real estate agency can enhance their pricing strategy, identify lucrative opportunities, and ultimately increase their annual sales. The insights obtained from this research provide a solid foundation for making data-driven decisions and gaining a competitive edge in the real estate market.

Next Steps

1. Incorporate temporal dynamics: Consider incorporating temporal dynamics by analyzing how the housing market changes over time. Explore time series analysis techniques to capture seasonal trends,

price fluctuations, or long-term market trends. This can help provide a more accurate representation of house price dynamics.

2. Collaborate with domain experts: Engage in discussions and collaborations with real estate experts or domain specialists to gain deeper insights into the market and the factors that influence house prices. Their expertise can provide valuable guidance and help validate the analysis conducted.
3. Refine the model: While the multiple linear regression model provided valuable insights, we could explore other regression techniques to improve the accuracy and predictive power of the model such as handling multicollinearity and over fitting and under under fitting.
4. Stay updated and iterate: Keep track of new data and industry trends to ensure that the model remains up-to-date and relevant. Regularly iterate and refine the analysis as new data becomes available or market dynamics change.