```
#Jane Martha
#James Njoroge
#Leon Maina
#Magdalene Ondimu
#Daniel Wahome
#Faith Mwenda


# import standard packages

import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings ('ignore')
import matplotlib.pyplot as plt
plt.style.use('ggplot')
from scipy import stats
import statsmodels.api as sm
import statsmodels.graphics as smg
import statsmodels.stats.api as sms
from statsmodels.stats.outliers_influence import variance_inflation_factor
import datetime as dt

#Loading data set for Analysis
kc = pd.read_csv(r"C:\Users\HP\Documents\Flatiron\Assignments\Phase 2
Project\kc_house_data.csv")

kc.head(5)

#Data understanding and exploration
kc.info()

kc.describe()


#changing the selling date to  and updating the column name to yr_sold
kc['date']=pd.to_datetime(kc['date'])
kc['date'] = kc['date'].dt.year


kc.rename (columns={'date': 'yr_sold'}, inplace=True)

kc['yr_sold']= kc['yr_sold'].astype(int)

kc.head(5)
```

```
kc.isna().sum()


#Filling missing values in the 'view' and 'waterfront' columns
kc['waterfront'].fillna('NO', inplace=True)
kc['view'].fillna('NONE', inplace=True)
kc.isna().sum()


#Counting the occurences of unique values in 'yr_revovated'
kc['yr_renovated'].value_counts()


kc['yr_renovated'].fillna(0.0, inplace=True)

kc['yr_renovated'].astype(int)

kc.isna().sum()


yr_renovated = kc['yr_renovated']
kc['renovated_last_10'] = (yr_renovated >= (kc['yr_sold'] - 10))
kc['renovated_last_10'] = kc['renovated_last_10'].map({True: 'Yes', False: 'No'})


kc.head(5)


kc['Age'] = kc['yr_sold']-kc['yr_built']


kc.head(5)

#Dropping columns that are not needed
columns_to_drop =
['id','zipcode','lat','long','sqft_living15','sqft_lot15','sqft_basement','yr_sold',
'yr_renovated']
kc.drop(columns_to_drop, axis = 1, inplace = True)

#Confirming the new dataframe
kc.head(5)


#VISUALIZING
#Creating a plotting function for ease or resizing in different plots
def resizeplot(l,a):
    plt.figure(figsize=(l,a));
```

```
#Creating a plot that will visualize the relationship between price and sqft_lot
resizeplot(6,4)
sns.relplot(x='sqft_lot',y='price',data=kc,palette='terrain');
plt.show()


#Finding out the occurences of bedroom values
bedrooms_counts = kc['bedrooms'].value_counts()
bedrooms_counts


#Confirming how the conditon of the house over the year affects the price
#We can see that if the conditon of the house- is improved then the price of the
house is higher
#Poor conditons leads to lower price
resizeplot(10,5)
sns.lineplot(x='condition',y='price',data=kc,palette='terrain')
plt.show()

#Using histogram to visualize the distribution of the price
#As the count is high then the price increases
resizeplot(6,4)
sns.histplot(kc['price'],kde=True,bins=50, color = 'red');


#Using linear to analyse price increament over the year
#At the begginging of the yrs the price is low as the yrs increases then the price
has a higher peak

resizeplot(12,6)
sns.lineplot(x='Age',y='price',data=kc);


#Price trend over the years
#The price from 2000 to 2015 was good meaning the owners of real estaste were
getting good profit
#It seems that the houses had been reinovated

resizeplot(12,6)
sns.lineplot(x='yr_built',y='price',data=kc,palette = 'deep');


#Showing the price changes in relation to unit grade
# poor grade causes the price to decrease and vice versa
resizeplot(12,4)# changed the size to give a better view of the grade.
sns.lineplot(x='grade',y='price',data=kc,palette='terrain');
plt.xticks(rotation=45)
plt.show()
```

```
#CATEGORICAL VARIABLE
# Plotting histogram for columns within the dataset
#The histogram shows us how different features in each column affect each other

fig, axes = plt.subplots(nrows=(5), ncols=3, figsize=(20,10))
df_cols = kc.columns

# Using sns color pallets for each plot

color = sns.color_palette("Blues", n_colors=1)[0]

# creation of a function for plotting the hustogram for the given columns

for col, ax in zip(df_cols, axes.flatten()):
    ax.hist(kc[col].dropna(), bins='auto', color=color )
    ax.set_title(col)


# automatically adjusting subplot params so that the subplot(s) fits in to the
figure area

fig.tight_layout()


#From the above plots we can see that
floors,waterfront,condition,grade,renovated_last_10,bedrooms,bathrooms are
categorical

#Creating a category of values to work with
categories = ['bedrooms', 'bathrooms', 'floors', 'waterfront', 'view', 'condition',
'grade', 'renovated_last_10']

for cate in categories:

    # getting the  value counts

    counts = kc[cate].value_counts()

    # Isolate offending categories for each variable

    bad_cate = counts[counts < 50].index

    # Isolate indices in the dataset where offending categories are found

    to_drop = kc[kc[cate].isin(bad_cate)].index
    # Dropping unnecessary data within the category in dataset
    kc.drop(to_drop, inplace=True)
```

```
    # converting view into binary (binarzing)
#for consistent and easier to compare in the view.
view_dict = {
    'FAIR': 1,
    'AVERAGE': 1,
    'GOOD': 1,
    'EXCELLENT': 1,
    'NONE': 0
}

kc['view'] = kc['view'].map(view_dict)

# Binarizing waterfront
waterfront_dict = {
    'YES': 1,
    'NO': 0
}

kc['waterfront'] = kc['waterfront'].map(waterfront_dict)

# Binarizing renovated_last_10
renovated_dict = {
    'Yes': 1,
    'No': 0
}

kc['renovated_last_10'] = kc['renovated_last_10'].map(renovated_dict)
```

```
#ONEHOTENCODING
#Creating a copy of the dataset so as not to alter the original copy incase
kc2= kc.copy()

#GENERATING DUMMIES
#Converting the 3columns: bedrooms,bathrooms,floors to string for pandas to be able
to dumify them
col = ['bedrooms', 'bathrooms', 'floors']
kc[col] =  kc[col].astype(str)


#Checking if the conversion was successful
kc.dtypes

# Creating variable from the already cleaned original dataset
kc_binary = kc[['waterfront', 'view', 'renovated_last_10']]
kc_num = kc[['price', 'sqft_living', 'sqft_lot', 'Age']]
```

```
kc_cate = kc[['floors', 'bedrooms', 'bathrooms', 'condition', 'grade']]


# Applying one-hot encoding to the categorical features
kc_cate_dummies = pd.get_dummies(kc_cate, dtype=int)

#Creating a list of dummies to be dropped

dummies_to_drop = [
    'floors_1.0',
    'bedrooms_1',
    'bathrooms_0.75',
    'condition_Fair',
    'grade_5 Fair'
    ]
#Dropping the specified dummies

kc_cate_dummies.drop(
    dummies_to_drop,
     axis = 1,
     inplace=True)



    #Combining the variable into a single variable
kc = pd.concat([kc_num, kc_binary, kc_cate_dummies], axis=1)

#Confirming if the event was successful
kc.head()

#Confirming the columns
kc.columns


#Setting the columns in an asceding order for easy analysis
grade_columns = [
    'grade_6 Low Average',
    'grade_7 Average',
    'grade_8 Good',
    'grade_9 Better',
    'grade_10 Very Good',
    'grade_11 Excellent',
    'grade_12 Luxury'
]

# Extracting other columns not related to 'grade'

other_columns = [col for col in kc.columns if col not in grade_columns]

# Reordering columns
```

```
reordered_columns = other_columns + grade_columns
kc = kc[reordered_columns]


#Accessing the columns
kc.columns


#Accessing the columns and rows
kc.head()

#MODELING
#Confirming the linear relationship between variable and price
kc_num.head()

#Plotting scatter plot for clear visulaization of different features
pd.plotting.scatter_matrix(kc_num, figsize=(20,15), alpha=.3);


#Displaying the correlation matrix in responce to  price in descending order
kc_num.corr()['price'].sort_values(ascending=False)


#Ensuring proper visualization using heatmap
#the lighter the color the stronger the correlation

fig, ax = plt.subplots(figsize = (8,10))

sns.heatmap(
    kc_num.corr().abs(),
#     mask=np.triu(np.ones_like(data.corr(), dtype=bool)),
    ax=ax,
    annot=True,
    cbar_kws={"label": "Correlation", "orientation": "horizontal", "pad": .2,
"extend": "both"}
);
```

Assessing Multicollinearity Across Predictor Combinations

```
# Create the correlation matrix directly and then reshape it for visualizatio
# Dropping the 'price' column
df = kc_num.drop('price', axis=1)

# Creating a correlation matrix
corr_matrix = df.corr().abs()
```

```
# Reshaping the correlation matrix for visualization
df_predictor = corr_matrix.stack().reset_index()
df_predictor.columns = ['Variable 1', 'Variable 2', 'Coefficient']

# Dropping duplicate rows where variables are the same
df_predictor = df_predictor[df_predictor['Variable 1'] != df_predictor['Variable
2']]

# Sorting by coefficient in descending order
df_predictor.sort_values(by='Coefficient', ascending=False, inplace=True)

df_predictor.head()

#sqft_lot and Age lack linear relationship
kc_num.drop(['sqft_lot', 'Age'], axis=1, inplace=True)



def reg_qq_sced(y, X, add_constant=True, qq=True, sced=True):
    """
    Fits a linear regression model, display its summary, and output plots to check
linear regression assumptions.

    Parameters:
    - y: Target variable.
    - X: Predictor variables.
    - add_constant: Whether to add a constant term to the predictors (default:
True).
    - qq: Whether to display a QQ plot for residual normality check (default: True).
    - sced: Whether to display a plot of predicted values vs. residuals for
homoscedasticity check (default: True).
    """
    # Add a constant to the predictors if required
    X_sm = sm.add_constant(X, has_constant='add') if add_constant else X

    # Run a linear regression and display the summary
    model = sm.OLS(y, X_sm).fit()
    display(print(model.summary()))

    # Display a QQ plot for residual normality check
    if qq:
        sm.qqplot(model.resid, line='45', fit=True)
        plt.title('QQ plot for residual normality check')
        plt.show()
    else:
        pass

    # Display a plot of predicted values vs. residuals for homoscedasticity check
    if sced:
        preds = model.predict(X_sm)
```

```
        residuals = model.resid
        fig_resid, ax = plt.subplots(figsize=(10, 5))
        fig_resid.suptitle('Predicted vs. residual plot for homoscedasticity check')
        ax.scatter(preds, residuals, alpha=0.2, color= "blue")
        ax.plot(preds, [0 for _ in range(len(X_sm))])
        ax.set_xlabel("Predicted Value")
        ax.set_ylabel("Actual - Predicted Value")
    else:
        pass

    # Output additional model performance metrics
    print(f'Model adjusted R-squared: {model.rsquared_adj}')
    print(f'Model RMSE: {np.sqrt(model.mse_resid)}')



    # Set baseline predictor as 'sqft_living'

baseline = 'sqft_living'

# Define target variable and predictor

y = kc.price
X = kc[baseline]

# Feed these inputs into our function

reg_qq_sced(y, X)



#The linear regression results indicate that the model's R-squared value is 0.466,
suggesting that approximately 46.6% of the variance in the target variable (price)
is explained by the predictor variable (sqft_living).

#The coefficient for the constant term (intercept) is -9704.3054, indicating the
predicted price when sqft_living is zero. The coefficient for sqft_living is
262.8635, indicating that for every unit increase in sqft_living, the price is
expected to increase by approximately $262.86. The p-value for sqft_living is less
than 0.05, indicating that the predictor variable is statistically significant. The
confidence interval for the coefficient of sqft_living ranges from 259.088 to
266.639. The residual plots indicate that there might be some heteroscedasticity
present, as the spread of residuals increases with predicted values. The QQ plot
suggests that the residuals are approximately normally distributed, but there might
be some deviations, especially in the tails.

#Overall, the model performs reasonably well, but there might be room for
improvement, especially in addressing heteroscedasticity.

# Isolating columns to be transformed
```

```
log_trans_cols = ['price', 'sqft_living']
kc_logged = kc.copy()[log_trans_cols]

# Log transforming and renaming columns

kc_logged = np.log(kc_logged)
kc_logged.columns = kc_logged.columns.map(lambda x: 'log_' + x)

# Merge it with the rest of the dataset

kc_transformed = kc_logged.join(kc.drop(log_trans_cols, axis=1))


kc_transformed.head()



# visualizing linearity between the transformed predictor and target variable
# The plot helps visualize the linearity between these two variables, which is
essential for linear regression modeling.

fig, ax = plt.subplots(figsize= (10,6))
ax = plt.gca()
plt.plot(kc_transformed['log_sqft_living'], kc_transformed['log_price'], 'o',
alpha=0.3)

# Set the background color of the axis

ax.set_facecolor('yellow')
plt.gcf().set_facecolor('green')
plt.xlabel('log of sqft_living')
plt.ylabel('log of price')
plt.title('Scatter Plot of log(sqft_living) vs. log(price)')
plt.show()

baseline = 'log_sqft_living'

y = kc_transformed.log_price
X = kc_transformed.log_sqft_living

# Feeding these inputs into our function

model = reg_qq_sced(y, X)



[ ]
# import standard packages
```

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings ('ignore')
import matplotlib.pyplot as plt
plt.style.use('ggplot')
from scipy import stats
import statsmodels.api as sm
import statsmodels.graphics as smg
import statsmodels.stats.api as sms
from statsmodels.stats.outliers_influence import variance_inflation_factor
import datetime as dt
[ ]
#Loading data set for Analysis
kc = pd.read_csv(r"C:\Users\HP\Documents\Flatiron\Assignments\Phase 2
Project\kc_house_data.csv")

kc.head(5)

[ ]
#Data understanding and exploration
kc.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21597 non-null  int64
 1   date           21597 non-null  object
 2   price          21597 non-null  int64
 3   bedrooms       21597 non-null  int64
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64
 6   sqft_lot       21597 non-null  int64
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  object
 9   view           21534 non-null  object
 10  condition      21597 non-null  object
 11  grade          21597 non-null  object
 12  sqft_above     21597 non-null  int64
 13  sqft_basement  21597 non-null  object
 14  yr_built       21597 non-null  int64
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64
```

```
 20  sqft_lot15     21597 non-null  int64
dtypes: float64(5), int64(10), object(6)
memory usage: 3.5+ MB
[ ]
kc.describe()

[ ]
#changing the selling date to  and updating the column name to yr_sold
kc['date']=pd.to_datetime(kc['date'])
kc['date'] = kc['date'].dt.year

[ ]
kc.rename (columns={'date': 'yr_sold'}, inplace=True)

[ ]
kc['yr_sold']= kc['yr_sold'].astype(int)
[ ]
kc.head(5)

[ ]
kc.isna().sum()
id                  0
yr_sold             0
price               0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront       2376
view               63
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated     3842
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
dtype: int64
[ ]
#Filling missing values in the 'view' and 'waterfront' columns
kc['waterfront'].fillna('NO', inplace=True)
kc['view'].fillna('NONE', inplace=True)
kc.isna().sum()
id                  0
yr_sold             0
```

```
price              0
bedrooms           0
bathrooms          0
sqft_living        0
sqft_lot           0
floors             0
waterfront         0
view               0
condition          0
grade              0
sqft_above         0
sqft_basement      0
yr_built           0
yr_renovated    3842
zipcode            0
lat                0
long               0
sqft_living15      0
sqft_lot15         0
dtype: int64
```

[ ]

```
#Counting the occurences of unique values in 'yr_revovated'
kc['yr_renovated'].value_counts()
```

```
0.0       17011
2014.0       73
2003.0       31
2013.0       31
2007.0       30
          ...
1946.0        1
1959.0        1
1971.0        1
1951.0        1
1954.0        1
Name: yr_renovated, Length: 70, dtype: int64
```

[ ]

```
kc['yr_renovated'].fillna(0.0, inplace=True)
```

[ ]

```
kc['yr_renovated'].astype(int)
```

```
0           0
1        1991
2           0
3           0
4           0
         ...
21592       0
21593       0
21594       0
21595       0
21596       0
```

```
Name: yr_renovated, Length: 21597, dtype: int32
[ ]
kc.isna().sum()
id               0
yr_sold          0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
[ ]
yr_renovated = kc['yr_renovated']
kc['renovated_last_10'] = (yr_renovated >= (kc['yr_sold'] - 10))
kc['renovated_last_10'] = kc['renovated_last_10'].map({True: 'Yes', False: 'No'})
[ ]
kc.head(5)

[ ]
kc['Age'] = kc['yr_sold']-kc['yr_built']
[ ]
kc.head(5)

[ ]
#Dropping columns that are not needed
columns_to_drop =
['id','zipcode','lat','long','sqft_living15','sqft_lot15','sqft_basement','yr_sold',
'yr_renovated']
kc.drop(columns_to_drop, axis = 1, inplace = True)
[ ]
#Confirming the new dataframe
kc.head(5)

Visualizing

[ ]
```

```
#Creating a plotting function for ease or resizing in different plots
def resizeplot(l,a):
    plt.figure(figsize=(l,a));

[ ]
#Creating a plot that will visualize the relationship between price and sqft_lot
resizeplot(6,4)
sns.relplot(x='sqft_lot',y='price',data=kc,palette='terrain');
plt.show()

[ ]
#Finding out the occurences of bedroom values
bedrooms_counts = kc['bedrooms'].value_counts()
bedrooms_counts
3     9824
4     6882
2     2760
5     1601
6      272
1      196
7       38
8       13
9        6
10       3
11       1
33       1
Name: bedrooms, dtype: int64
[ ]
#Confirming how the conditon of the house over the year affects the price
#We can see that if the conditon of the house- is improved then the price of the
house is higher
#Poor conditons leads to lower price
resizeplot(10,5)
sns.lineplot(x='condition',y='price',data=kc,palette='terrain')
plt.show()

[ ]
#Using histogram to visualize the distribution of the price
#As the count is high then the price increases
resizeplot(6,4)
sns.histplot(kc['price'],kde=True,bins=50, color = 'red');

[ ]
#Using linear to analyse price increament over the year
#At the begginging of the yrs the price is low as the yrs increases then the price
has a higher peak

resizeplot(12,6)
sns.lineplot(x='Age',y='price',data=kc);
```

```
[ ]
#Price trend over the years
#The price from 2000 to 2015 was good meaning the owners of real estaste were
getting good profit
#It seems that the houses had been reinovated

resizeplot(12,6)
sns.lineplot(x='yr_built',y='price',data=kc,palette = 'deep');

[ ]
#Showing the price changes in relation to unit grade
# poor grade causes the price to decrease and vice versa
resizeplot(12,4)# changed the size to give a better view of the grade.
sns.lineplot(x='grade',y='price',data=kc,palette='terrain');
plt.xticks(rotation=45)
plt.show()

Categorical Variables
[ ]
# creation of a function for plotting the hustogram for the given columns

for col, ax in zip(df_cols, axes.flatten()):
    ax.hist(kc[col].dropna(), bins='auto', color=color )
    ax.set_title(col)


# automatically adjusting subplot params so that the subplot(s) fits in to the
figure area

fig.tight_layout()

[ ]
#From the above plots we can see that
floors,waterfront,condition,grade,renovated_last_10,bedrooms,bathrooms are
categorical

#Creating a category of values to work with
categories = ['bedrooms', 'bathrooms', 'floors', 'waterfront', 'view', 'condition',
'grade', 'renovated_last_10']

for cate in categories:

    # getting the  value counts

    counts = kc[cate].value_counts()

    # Isolate offending categories for each variable

    bad_cate = counts[counts < 50].index
```

```
    # Isolate indices in the dataset where offending categories are found

    to_drop = kc[kc[cate].isin(bad_cate)].index
    # Dropping unnecessary data within the category in dataset
    kc.drop(to_drop, inplace=True)
[ ]
# converting view into binary (binarzing)
#for consistent and easier to compare in the view.
view_dict = {
    'FAIR': 1,
    'AVERAGE': 1,
    'GOOD': 1,
    'EXCELLENT': 1,
    'NONE': 0
}

kc['view'] = kc['view'].map(view_dict)

# Binarizing waterfront
waterfront_dict = {
    'YES': 1,
    'NO': 0
}

kc['waterfront'] = kc['waterfront'].map(waterfront_dict)

# Binarizing renovated_last_10
renovated_dict = {
    'Yes': 1,
    'No': 0
}

kc['renovated_last_10'] = kc['renovated_last_10'].map(renovated_dict)
OneHotEncoding
[ ]
#Creating a copy of the dataset so as not to alter the original copy incase
kc2= kc.copy()
Generating dummies
[ ]
#Converting the 3columns: bedrooms,bathrooms,floors to string for pandas to be able
to dumify them
col = ['bedrooms', 'bathrooms', 'floors']
kc[col] =  kc[col].astype(str)
[ ]
#Checking if the conversion was successful
kc.dtypes
price                 int64
bedrooms             object
bathrooms            object
sqft_living           int64
```

```
sqft_lot              int64
floors               object
waterfront            int64
view                  int64
condition            object
grade                object
sqft_above            int64
yr_built              int64
renovated_last_10     int64
Age                   int64
dtype: object
[ ]
# Creating variable from the already cleaned original dataset
kc_binary = kc[['waterfront', 'view', 'renovated_last_10']]
kc_num = kc[['price', 'sqft_living', 'sqft_lot', 'Age']]
kc_cate = kc[['floors', 'bedrooms', 'bathrooms', 'condition', 'grade']]
[ ]
# Applying one-hot encoding to the categorical features
kc_cate_dummies = pd.get_dummies(kc_cate, dtype=int)
[ ]
#Creating a list of dummies to be dropped

dummies_to_drop = [
    'floors_1.0',
    'bedrooms_1',
    'bathrooms_0.75',
    'condition_Fair',
    'grade_5 Fair'
    ]
#Dropping the specified dummies

kc_cate_dummies.drop(
    dummies_to_drop,
     axis = 1,
     inplace=True)
[ ]
#Combining the variable into a single variable
kc = pd.concat([kc_num, kc_binary, kc_cate_dummies], axis=1)
[ ]
#Confirming if the event was successful
kc.head()

[ ]
#Confirming the columns
kc.columns
Index(['price', 'sqft_living', 'sqft_lot', 'Age', 'waterfront', 'view',
       'renovated_last_10', 'floors_1.5', 'floors_2.0', 'floors_2.5',
       'floors_3.0', 'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5',
       'bedrooms_6', 'bathrooms_1.0', 'bathrooms_1.5', 'bathrooms_1.75',
       'bathrooms_2.0', 'bathrooms_2.25', 'bathrooms_2.5', 'bathrooms_2.75',
```

```
       'bathrooms_3.0', 'bathrooms_3.25', 'bathrooms_3.5', 'bathrooms_3.75',
       'bathrooms_4.0', 'bathrooms_4.25', 'bathrooms_4.5', 'condition_Average',
       'condition_Good', 'condition_Very Good', 'grade_10 Very Good',
       'grade_11 Excellent', 'grade_12 Luxury', 'grade_6 Low Average',
       'grade_7 Average', 'grade_8 Good', 'grade_9 Better'],
      dtype='object')
[ ]
#Setting the columns in an asceding order for easy analysis
grade_columns = [
    'grade_6 Low Average',
    'grade_7 Average',
    'grade_8 Good',
    'grade_9 Better',
    'grade_10 Very Good',
    'grade_11 Excellent',
    'grade_12 Luxury'
]

# Extracting other columns not related to 'grade'

other_columns = [col for col in kc.columns if col not in grade_columns]

# Reordering columns

reordered_columns = other_columns + grade_columns
kc = kc[reordered_columns]
[ ]
#Accessing the columns
kc.columns
Index(['price', 'sqft_living', 'sqft_lot', 'Age', 'waterfront', 'view',
       'renovated_last_10', 'floors_1.5', 'floors_2.0', 'floors_2.5',
       'floors_3.0', 'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5',
       'bedrooms_6', 'bathrooms_1.0', 'bathrooms_1.5', 'bathrooms_1.75',
       'bathrooms_2.0', 'bathrooms_2.25', 'bathrooms_2.5', 'bathrooms_2.75',
       'bathrooms_3.0', 'bathrooms_3.25', 'bathrooms_3.5', 'bathrooms_3.75',
       'bathrooms_4.0', 'bathrooms_4.25', 'bathrooms_4.5', 'condition_Average',
       'condition_Good', 'condition_Very Good', 'grade_6 Low Average',
       'grade_7 Average', 'grade_8 Good', 'grade_9 Better',
       'grade_10 Very Good', 'grade_11 Excellent', 'grade_12 Luxury'],
      dtype='object')
[ ]
#Accessing the columns and rows
kc.head()

Modeling
[ ]
#Confirming the linear relationship between variable and price
kc_num.head()

[ ]
```

```
#Plotting scatter plot for clear visulaization of different features
pd.plotting.scatter_matrix(kc_num, figsize=(20,15), alpha=.3);

[ ]
#Displaying the correlation matrix in responce to  price in descending order
kc_num.corr()['price'].sort_values(ascending=False)
price          1.000000
sqft_living    0.682342
sqft_lot       0.084739
Age           -0.048693
Name: price, dtype: float64
[ ]
#Ensuring proper visualization using heatmap
#the lighter the color the stronger the correlation

fig, ax = plt.subplots(figsize = (8,10))

sns.heatmap(
    kc_num.corr().abs(),
#     mask=np.triu(np.ones_like(data.corr(), dtype=bool)),
    ax=ax,
    annot=True,
    cbar_kws={"label": "Correlation", "orientation": "horizontal", "pad": .2,
"extend": "both"}
);

**sqft_living: 0.682342 (Strong positive correlation) sqft_lot: 0.084739 (Weak
positive correlation) Age: -0.048722 (Weak negative correlation)

Assessing Multicollinearity Across Predictor Combinations
[ ]
# Create the correlation matrix directly and then reshape it for visualizatio
# Dropping the 'price' column
df = kc_num.drop('price', axis=1)

# Creating a correlation matrix
corr_matrix = df.corr().abs()

# Reshaping the correlation matrix for visualization
df_predictor = corr_matrix.stack().reset_index()
df_predictor.columns = ['Variable 1', 'Variable 2', 'Coefficient']

# Dropping duplicate rows where variables are the same
df_predictor = df_predictor[df_predictor['Variable 1'] != df_predictor['Variable
2']]

# Sorting by coefficient in descending order
df_predictor.sort_values(by='Coefficient', ascending=False, inplace=True)

df_predictor.head()
```

```
[ ]
#sqft_lot and Age lack linear relationship
kc_num.drop(['sqft_lot', 'Age'], axis=1, inplace=True)
[ ]
def reg_qq_sced(y, X, add_constant=True, qq=True, sced=True):
    """
    Fits a linear regression model, display its summary, and output plots to check
linear regression assumptions.

    Parameters:
    - y: Target variable.
    - X: Predictor variables.
    - add_constant: Whether to add a constant term to the predictors (default:
True).
    - qq: Whether to display a QQ plot for residual normality check (default: True).
    - sced: Whether to display a plot of predicted values vs. residuals for
homoscedasticity check (default: True).
    """
    # Add a constant to the predictors if required
    X_sm = sm.add_constant(X, has_constant='add') if add_constant else X

    # Run a linear regression and display the summary
    model = sm.OLS(y, X_sm).fit()
    display(print(model.summary()))

    # Display a QQ plot for residual normality check
    if qq:
        sm.qqplot(model.resid, line='45', fit=True)
        plt.title('QQ plot for residual normality check')
        plt.show()
    else:
        pass

    # Display a plot of predicted values vs. residuals for homoscedasticity check
    if sced:
        preds = model.predict(X_sm)
        residuals = model.resid
        fig_resid, ax = plt.subplots(figsize=(10, 5))
        fig_resid.suptitle('Predicted vs. residual plot for homoscedasticity check')
        ax.scatter(preds, residuals, alpha=0.2, color= "blue")
        ax.plot(preds, [0 for _ in range(len(X_sm))])
        ax.set_xlabel("Predicted Value")
        ax.set_ylabel("Actual - Predicted Value")
    else:
        pass

    # Output additional model performance metrics
    print(f'Model adjusted R-squared: {model.rsquared_adj}')
```

```
    print(f'Model RMSE: {np.sqrt(model.mse_resid)}')
```

[ ]

```
# Set baseline predictor as 'sqft_living'

baseline = 'sqft_living'

# Define target variable and predictor

y = kc.price
X = kc[baseline]

# Feed these inputs into our function

reg_qq_sced(y, X)
```

The linear regression results indicate that the model's R-squared value is 0.466, suggesting that approximately 46.6% of the variance in the target variable (price) is explained by the predictor variable (sqft_living).

The coefficient for the constant term (intercept) is -9704.3054, indicating the predicted price when sqft_living is zero. The coefficient for sqft_living is 262.8635, indicating that for every unit increase in sqft_living, the price is expected to increase by approximately $262.86. The p-value for sqft_living is less than 0.05, indicating that the predictor variable is statistically significant. The confidence interval for the coefficient of sqft_living ranges from 259.088 to 266.639. The residual plots indicate that there might be some heteroscedasticity present, as the spread of residuals increases with predicted values. The QQ plot suggests that the residuals are approximately normally distributed, but there might be some deviations, especially in the tails.

Overall, the model performs reasonably well, but there might be room for improvement, especially in addressing heteroscedasticity.

[ ]

```
# Isolating columns to be transformed

log_trans_cols = ['price', 'sqft_living']
kc_logged = kc.copy()[log_trans_cols]

# Log transforming and renaming columns

kc_logged = np.log(kc_logged)
kc_logged.columns = kc_logged.columns.map(lambda x: 'log_' + x)

# Merge it with the rest of the dataset

kc_transformed = kc_logged.join(kc.drop(log_trans_cols, axis=1))
```

```
kc_transformed.head()

[ ]
# visualizing linearity between the transformed predictor and target variable
# The plot helps visualize the linearity between these two variables, which is
essential for linear regression modeling.

fig, ax = plt.subplots(figsize= (10,6))
ax = plt.gca()
plt.plot(kc_transformed['log_sqft_living'], kc_transformed['log_price'], 'o',
alpha=0.3)

# Set the background color of the axis

ax.set_facecolor('yellow')
plt.gcf().set_facecolor('green')
plt.xlabel('log of sqft_living')
plt.ylabel('log of price')
plt.title('Scatter Plot of log(sqft_living) vs. log(price)')
plt.show()

[ ]
baseline = 'log_sqft_living'

y = kc_transformed.log_price
X = kc_transformed.log_sqft_living

# Feeding these inputs into our function

model = reg_qq_sced(y, X)


#The regression results indicate that the logarithm of square footage of living
space (log_sqft_living) is a significant predictor of the logarithm of price
(log_price). Here's a summary of the regression results:

#R-squared: The coefficient of determination indicates that approximately 44.1% of
the variance in the logarithm of price can be explained by the logarithm of square
footage of living space.

#Coefficient Estimates:

#The coefficient for log_sqft_living is approximately 0.8241, indicating that for
every one-unit increase in the logarithm of square footage of living space, the
logarithm of price is expected to increase by approximately 0.8241 units. The
intercept (constant) term is approximately 6.8236, which represents the estimated
logarithm of price when the logarithm of square footage of living space is zero.
Statistical Significance: Both coefficients are statistically significant with
p-values < 0.05, suggesting that they are unlikely to be zero.
```

#Model Fit: The model's goodness of fit is indicated by the adjusted R-squared value of approximately 0.441, which is a measure of how well the independent variable explains the variation in the dependent variable.

#Overall, based on these results, we can conclude that there is a strong linear relationship between the logarithm of square footage of living space and the logarithm of price.

```
baseline = 'log_sqft_living'


# Define target variable and predictors

y = kc_transformed.log_price
X = kc_transformed[[baseline, 'waterfront', 'view', 'renovated_last_10']]

model = reg_qq_sced(y, X)
```

#The updated regression results indicate that the model now includes additional predictors: waterfront, view, and renovated_last_10, in addition to log_sqft_living. Here's a summary of the updated regression results:

#R-squared: The coefficient of determination has increased to approximately 0.480, suggesting that the additional predictors have improved the model's ability to explain the variance in the logarithm of price.

#Coefficient Estimates:

#The coefficient for log_sqft_living remains significant and has a value of approximately 0.7696. The coefficients for the additional predictors (waterfront, view, and renovated_last_10) are also significant: waterfront: Coefficient is approximately 0.5078, indicating that waterfront properties tend to have higher prices. view: Coefficient is approximately 0.2787, suggesting that properties with better views tend to have higher prices. renovated_last_10: Coefficient is approximately 0.2415, indicating that recently renovated properties tend to have higher prices. Statistical Significance: All coefficients are statistically significant with p-values < 0.05.

#Model Fit: The adjusted R-squared value of approximately 0.480 indicates that the model with the additional predictors provides a better fit to the data compared to the previous model.

```
# Grouping the dummies together into lists form

dummies = ['floors', 'bedrooms', 'bathrooms', 'condition', 'grade']
floors_dummies = []
```

```
bedrooms_dummies = []
bathrooms_dummies = []
condition_dummies = []
grade_dummies = []

for col in list(kc.columns):
    for cate in dummies:
        if col.startswith(cate):
            eval(cate + '_dummies').append(col)



    # Defining the target variable and predictors

y = kc_transformed.log_price
X = kc_transformed[floors_dummies +
                   bedrooms_dummies +
                   bathrooms_dummies +
                   condition_dummies +
                   grade_dummies +
                   ['log_sqft_living']]

model = reg_qq_sced(y, X)


[ ]
# import standard packages

import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings ('ignore')
import matplotlib.pyplot as plt
plt.style.use('ggplot')
from scipy import stats
import statsmodels.api as sm
import statsmodels.graphics as smg
import statsmodels.stats.api as sms
from statsmodels.stats.outliers_influence import variance_inflation_factor
import datetime as dt
[ ]
#Loading data set for Analysis
kc = pd.read_csv(r"C:\Users\HP\Documents\Flatiron\Assignments\Phase 2
Project\kc_house_data.csv")

kc.head(5)
```

```
[ ]
#Data understanding and exploration
kc.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21597 non-null  int64
 1   date           21597 non-null  object
 2   price          21597 non-null  int64
 3   bedrooms       21597 non-null  int64
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64
 6   sqft_lot       21597 non-null  int64
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  object
 9   view           21534 non-null  object
 10  condition      21597 non-null  object
 11  grade          21597 non-null  object
 12  sqft_above     21597 non-null  int64
 13  sqft_basement  21597 non-null  object
 14  yr_built       21597 non-null  int64
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64
 20  sqft_lot15     21597 non-null  int64
dtypes: float64(5), int64(10), object(6)
memory usage: 3.5+ MB
[ ]
kc.describe()

[ ]
#changing the selling date to  and updating the column name to yr_sold
kc['date']=pd.to_datetime(kc['date'])
kc['date'] = kc['date'].dt.year

[ ]
kc.rename (columns={'date': 'yr_sold'}, inplace=True)

[ ]
kc['yr_sold']= kc['yr_sold'].astype(int)
[ ]
kc.head(5)

[ ]
kc.isna().sum()
id                  0
```

```
yr_sold             0
price               0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront       2376
view               63
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated     3842
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
dtype: int64
```

[ ]

```
#Filling missing values in the 'view' and 'waterfront' columns
kc['waterfront'].fillna('NO', inplace=True)
kc['view'].fillna('NONE', inplace=True)
kc.isna().sum()
```

```
id                  0
yr_sold             0
price               0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront          0
view                0
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated     3842
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
dtype: int64
```

[ ]

```
#Counting the occurences of unique values in 'yr_revovated'
```

```
kc['yr_renovated'].value_counts()
0.0       17011
2014.0       73
2003.0       31
2013.0       31
2007.0       30
          ...
1946.0        1
1959.0        1
1971.0        1
1951.0        1
1954.0        1
Name: yr_renovated, Length: 70, dtype: int64
[ ]
kc['yr_renovated'].fillna(0.0, inplace=True)
[ ]
kc['yr_renovated'].astype(int)
0           0
1        1991
2           0
3           0
4           0
         ...
21592       0
21593       0
21594       0
21595       0
21596       0
Name: yr_renovated, Length: 21597, dtype: int32
[ ]
kc.isna().sum()
id               0
yr_sold          0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
```

```
sqft_living15    0
sqft_lot15       0
dtype: int64
[ ]
yr_renovated = kc['yr_renovated']
kc['renovated_last_10'] = (yr_renovated >= (kc['yr_sold'] - 10))
kc['renovated_last_10'] = kc['renovated_last_10'].map({True: 'Yes', False: 'No'})
[ ]
kc.head(5)

[ ]
kc['Age'] = kc['yr_sold']-kc['yr_built']
[ ]
kc.head(5)

[ ]
#Dropping columns that are not needed
columns_to_drop =
['id','zipcode','lat','long','sqft_living15','sqft_lot15','sqft_basement','yr_sold',
'yr_renovated']
kc.drop(columns_to_drop, axis = 1, inplace = True)
[ ]
#Confirming the new dataframe
kc.head(5)

Visualizing

[ ]
#Creating a plotting function for ease or resizing in different plots
def resizeplot(l,a):
    plt.figure(figsize=(l,a));

[ ]
#Creating a plot that will visualize the relationship between price and sqft_lot
resizeplot(6,4)
sns.relplot(x='sqft_lot',y='price',data=kc,palette='terrain');
plt.show()

[ ]
#Finding out the occurences of bedroom values
bedrooms_counts = kc['bedrooms'].value_counts()
bedrooms_counts
3     9824
4     6882
2     2760
5     1601
6      272
1      196
7       38
8       13
```

```
9        6
10       3
11       1
33       1
Name: bedrooms, dtype: int64
[ ]
#Confirming how the conditon of the house over the year affects the price
#We can see that if the conditon of the house- is improved then the price of the
house is higher
#Poor conditons leads to lower price
resizeplot(10,5)
sns.lineplot(x='condition',y='price',data=kc,palette='terrain')
plt.show()

[ ]
#Using histogram to visualize the distribution of the price
#As the count is high then the price increases
resizeplot(6,4)
sns.histplot(kc['price'],kde=True,bins=50, color = 'red');

[ ]
#Using linear to analyse price increament over the year
#At the begginging of the yrs the price is low as the yrs increases then the price
has a higher peak

resizeplot(12,6)
sns.lineplot(x='Age',y='price',data=kc);

[ ]
#Price trend over the years
#The price from 2000 to 2015 was good meaning the owners of real estaste were
getting good profit
#It seems that the houses had been reinovated

resizeplot(12,6)
sns.lineplot(x='yr_built',y='price',data=kc,palette = 'deep');

[ ]
#Showing the price changes in relation to unit grade
# poor grade causes the price to decrease and vice versa
resizeplot(12,4)# changed the size to give a better view of the grade.
sns.lineplot(x='grade',y='price',data=kc,palette='terrain');
plt.xticks(rotation=45)
plt.show()

Categorical Variables
[ ]
# Plotting histogram for columns within the dataset
#The histogram shows us how different features in each column affect each other
```

```
fig, axes = plt.subplots(nrows=(5), ncols=3, figsize=(20,10))
df_cols = kc.columns

# Using sns color pallets for each plot

color = sns.color_palette("Blues", n_colors=1)[0]

# creation of a function for plotting the hustogram for the given columns

for col, ax in zip(df_cols, axes.flatten()):
    ax.hist(kc[col].dropna(), bins='auto', color=color )
    ax.set_title(col)


# automatically adjusting subplot params so that the subplot(s) fits in to the
figure area

fig.tight_layout()

[ ]
#From the above plots we can see that
floors,waterfront,condition,grade,renovated_last_10,bedrooms,bathrooms are
categorical

#Creating a category of values to work with
categories = ['bedrooms', 'bathrooms', 'floors', 'waterfront', 'view', 'condition',
'grade', 'renovated_last_10']

for cate in categories:

    # getting the  value counts

    counts = kc[cate].value_counts()

    # Isolate offending categories for each variable

    bad_cate = counts[counts < 50].index

    # Isolate indices in the dataset where offending categories are found

    to_drop = kc[kc[cate].isin(bad_cate)].index
    # Dropping unnecessary data within the category in dataset
    kc.drop(to_drop, inplace=True)
[ ]
# converting view into binary (binarzing)
#for consistent and easier to compare in the view.
view_dict = {
    'FAIR': 1,
    'AVERAGE': 1,
    'GOOD': 1,
```

```
    'EXCELLENT': 1,
    'NONE': 0
}

kc['view'] = kc['view'].map(view_dict)

# Binarizing waterfront
waterfront_dict = {
    'YES': 1,
    'NO': 0
}

kc['waterfront'] = kc['waterfront'].map(waterfront_dict)

# Binarizing renovated_last_10
renovated_dict = {
    'Yes': 1,
    'No': 0
}

kc['renovated_last_10'] = kc['renovated_last_10'].map(renovated_dict)
```

OneHotEncoding

[ ]

```
#Creating a copy of the dataset so as not to alter the original copy incase
kc2= kc.copy()
```

Generating dummies

[ ]

```
#Converting the 3columns: bedrooms,bathrooms,floors to string for pandas to be able
to dumify them
col = ['bedrooms', 'bathrooms', 'floors']
kc[col] =  kc[col].astype(str)
```

[ ]

```
#Checking if the conversion was successful
kc.dtypes
price                 int64
bedrooms             object
bathrooms            object
sqft_living           int64
sqft_lot              int64
floors               object
waterfront            int64
view                  int64
condition            object
grade                object
sqft_above            int64
yr_built              int64
renovated_last_10     int64
Age                   int64
dtype: object
```

[ ]

```
# Creating variable from the already cleaned original dataset
kc_binary = kc[['waterfront', 'view', 'renovated_last_10']]
kc_num = kc[['price', 'sqft_living', 'sqft_lot', 'Age']]
kc_cate = kc[['floors', 'bedrooms', 'bathrooms', 'condition', 'grade']]
[ ]
# Applying one-hot encoding to the categorical features
kc_cate_dummies = pd.get_dummies(kc_cate, dtype=int)
[ ]
#Creating a list of dummies to be dropped

dummies_to_drop = [
    'floors_1.0',
    'bedrooms_1',
    'bathrooms_0.75',
    'condition_Fair',
    'grade_5 Fair'
    ]
#Dropping the specified dummies

kc_cate_dummies.drop(
    dummies_to_drop,
     axis = 1,
     inplace=True)
[ ]
#Combining the variable into a single variable
kc = pd.concat([kc_num, kc_binary, kc_cate_dummies], axis=1)
[ ]
#Confirming if the event was successful
kc.head()

[ ]
#Confirming the columns
kc.columns
Index(['price', 'sqft_living', 'sqft_lot', 'Age', 'waterfront', 'view',
       'renovated_last_10', 'floors_1.5', 'floors_2.0', 'floors_2.5',
       'floors_3.0', 'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5',
       'bedrooms_6', 'bathrooms_1.0', 'bathrooms_1.5', 'bathrooms_1.75',
       'bathrooms_2.0', 'bathrooms_2.25', 'bathrooms_2.5', 'bathrooms_2.75',
       'bathrooms_3.0', 'bathrooms_3.25', 'bathrooms_3.5', 'bathrooms_3.75',
       'bathrooms_4.0', 'bathrooms_4.25', 'bathrooms_4.5', 'condition_Average',
       'condition_Good', 'condition_Very Good', 'grade_10 Very Good',
       'grade_11 Excellent', 'grade_12 Luxury', 'grade_6 Low Average',
       'grade_7 Average', 'grade_8 Good', 'grade_9 Better'],
      dtype='object')
[ ]
#Setting the columns in an asceding order for easy analysis
grade_columns = [
    'grade_6 Low Average',
    'grade_7 Average',
    'grade_8 Good',
```

```
    'grade_9 Better',
    'grade_10 Very Good',
    'grade_11 Excellent',
    'grade_12 Luxury'
]

# Extracting other columns not related to 'grade'

other_columns = [col for col in kc.columns if col not in grade_columns]

# Reordering columns

reordered_columns = other_columns + grade_columns
kc = kc[reordered_columns]
[ ]
#Accessing the columns
kc.columns
Index(['price', 'sqft_living', 'sqft_lot', 'Age', 'waterfront', 'view',
       'renovated_last_10', 'floors_1.5', 'floors_2.0', 'floors_2.5',
       'floors_3.0', 'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5',
       'bedrooms_6', 'bathrooms_1.0', 'bathrooms_1.5', 'bathrooms_1.75',
       'bathrooms_2.0', 'bathrooms_2.25', 'bathrooms_2.5', 'bathrooms_2.75',
       'bathrooms_3.0', 'bathrooms_3.25', 'bathrooms_3.5', 'bathrooms_3.75',
       'bathrooms_4.0', 'bathrooms_4.25', 'bathrooms_4.5', 'condition_Average',
       'condition_Good', 'condition_Very Good', 'grade_6 Low Average',
       'grade_7 Average', 'grade_8 Good', 'grade_9 Better',
       'grade_10 Very Good', 'grade_11 Excellent', 'grade_12 Luxury'],
      dtype='object')
[ ]
#Accessing the columns and rows
kc.head()

Modeling
[ ]
#Confirming the linear relationship between variable and price
kc_num.head()

[ ]
#Plotting scatter plot for clear visulaization of different features
pd.plotting.scatter_matrix(kc_num, figsize=(20,15), alpha=.3);

[ ]
#Displaying the correlation matrix in responce to  price in descending order
kc_num.corr()['price'].sort_values(ascending=False)
price          1.000000
sqft_living    0.682342
sqft_lot       0.084739
Age           -0.048693
Name: price, dtype: float64
[ ]
```

```
#Ensuring proper visualization using heatmap
#the lighter the color the stronger the correlation

fig, ax = plt.subplots(figsize = (8,10))

sns.heatmap(
    kc_num.corr().abs(),
#     mask=np.triu(np.ones_like(data.corr(), dtype=bool)),
    ax=ax,
    annot=True,
    cbar_kws={"label": "Correlation", "orientation": "horizontal", "pad": .2,
"extend": "both"}
);

**sqft_living: 0.682342 (Strong positive correlation) sqft_lot: 0.084739 (Weak
positive correlation) Age: -0.048722 (Weak negative correlation)

Assessing Multicollinearity Across Predictor Combinations
[ ]
# Create the correlation matrix directly and then reshape it for visualizatio
# Dropping the 'price' column
df = kc_num.drop('price', axis=1)

# Creating a correlation matrix
corr_matrix = df.corr().abs()

# Reshaping the correlation matrix for visualization
df_predictor = corr_matrix.stack().reset_index()
df_predictor.columns = ['Variable 1', 'Variable 2', 'Coefficient']

# Dropping duplicate rows where variables are the same
df_predictor = df_predictor[df_predictor['Variable 1'] != df_predictor['Variable
2']]

# Sorting by coefficient in descending order
df_predictor.sort_values(by='Coefficient', ascending=False, inplace=True)

df_predictor.head()


[ ]
#sqft_lot and Age lack linear relationship
kc_num.drop(['sqft_lot', 'Age'], axis=1, inplace=True)
[ ]
def reg_qq_sced(y, X, add_constant=True, qq=True, sced=True):
    """
    Fits a linear regression model, display its summary, and output plots to check
linear regression assumptions.

    Parameters:
```

```
    - y: Target variable.
    - X: Predictor variables.
    - add_constant: Whether to add a constant term to the predictors (default:
True).
    - qq: Whether to display a QQ plot for residual normality check (default: True).
    - sced: Whether to display a plot of predicted values vs. residuals for
homoscedasticity check (default: True).
    """
    # Add a constant to the predictors if required
    X_sm = sm.add_constant(X, has_constant='add') if add_constant else X

    # Run a linear regression and display the summary
    model = sm.OLS(y, X_sm).fit()
    display(print(model.summary()))

    # Display a QQ plot for residual normality check
    if qq:
        sm.qqplot(model.resid, line='45', fit=True)
        plt.title('QQ plot for residual normality check')
        plt.show()
    else:
        pass

    # Display a plot of predicted values vs. residuals for homoscedasticity check
    if sced:
        preds = model.predict(X_sm)
        residuals = model.resid
        fig_resid, ax = plt.subplots(figsize=(10, 5))
        fig_resid.suptitle('Predicted vs. residual plot for homoscedasticity check')
        ax.scatter(preds, residuals, alpha=0.2, color= "blue")
        ax.plot(preds, [0 for _ in range(len(X_sm))])
        ax.set_xlabel("Predicted Value")
        ax.set_ylabel("Actual - Predicted Value")
    else:
        pass

    # Output additional model performance metrics
    print(f'Model adjusted R-squared: {model.rsquared_adj}')
    print(f'Model RMSE: {np.sqrt(model.mse_resid)}')

[ ]
# Set baseline predictor as 'sqft_living'

baseline = 'sqft_living'

# Define target variable and predictor

y = kc.price
X = kc[baseline]
```

```
# Feed these inputs into our function

reg_qq_sced(y, X)
```

The linear regression results indicate that the model's R-squared value is 0.466, suggesting that approximately 46.6% of the variance in the target variable (price) is explained by the predictor variable (sqft_living).

The coefficient for the constant term (intercept) is -9704.3054, indicating the predicted price when sqft_living is zero. The coefficient for sqft_living is 262.8635, indicating that for every unit increase in sqft_living, the price is expected to increase by approximately $262.86. The p-value for sqft_living is less than 0.05, indicating that the predictor variable is statistically significant. The confidence interval for the coefficient of sqft_living ranges from 259.088 to 266.639. The residual plots indicate that there might be some heteroscedasticity present, as the spread of residuals increases with predicted values. The QQ plot suggests that the residuals are approximately normally distributed, but there might be some deviations, especially in the tails.

Overall, the model performs reasonably well, but there might be room for improvement, especially in addressing heteroscedasticity.

[ ]

```
# Isolating columns to be transformed

log_trans_cols = ['price', 'sqft_living']
kc_logged = kc.copy()[log_trans_cols]

# Log transforming and renaming columns

kc_logged = np.log(kc_logged)
kc_logged.columns = kc_logged.columns.map(lambda x: 'log_' + x)

# Merge it with the rest of the dataset

kc_transformed = kc_logged.join(kc.drop(log_trans_cols, axis=1))


kc_transformed.head()
```

[ ]

```
# visualizing linearity between the transformed predictor and target variable
# The plot helps visualize the linearity between these two variables, which is
essential for linear regression modeling.

fig, ax = plt.subplots(figsize= (10,6))
ax = plt.gca()
plt.plot(kc_transformed['log_sqft_living'], kc_transformed['log_price'], 'o',
alpha=0.3)
```

```
# Set the background color of the axis

ax.set_facecolor('yellow')
plt.gcf().set_facecolor('green')
plt.xlabel('log of sqft_living')
plt.ylabel('log of price')
plt.title('Scatter Plot of log(sqft_living) vs. log(price)')
plt.show()

[ ]
baseline = 'log_sqft_living'

y = kc_transformed.log_price
X = kc_transformed.log_sqft_living

# Feeding these inputs into our function

model = reg_qq_sced(y, X)
```

The regression results indicate that the logarithm of square footage of living space (log_sqft_living) is a significant predictor of the logarithm of price (log_price). Here's a summary of the regression results:

R-squared: The coefficient of determination indicates that approximately 44.1% of the variance in the logarithm of price can be explained by the logarithm of square footage of living space.

Coefficient Estimates:

The coefficient for log_sqft_living is approximately 0.8241, indicating that for every one-unit increase in the logarithm of square footage of living space, the logarithm of price is expected to increase by approximately 0.8241 units. The intercept (constant) term is approximately 6.8236, which represents the estimated logarithm of price when the logarithm of square footage of living space is zero. Statistical Significance: Both coefficients are statistically significant with p-values < 0.05, suggesting that they are unlikely to be zero.

Model Fit: The model's goodness of fit is indicated by the adjusted R-squared value of approximately 0.441, which is a measure of how well the independent variable explains the variation in the dependent variable.

Overall, based on these results, we can conclude that there is a strong linear relationship between the logarithm of square footage of living space and the logarithm of price.

```
[ ]
baseline = 'log_sqft_living'


# Define target variable and predictors
```

```
y = kc_transformed.log_price
X = kc_transformed[[baseline, 'waterfront', 'view', 'renovated_last_10']]

model = reg_qq_sced(y, X)
```

The updated regression results indicate that the model now includes additional predictors: waterfront, view, and renovated_last_10, in addition to log_sqft_living. Here's a summary of the updated regression results:

R-squared: The coefficient of determination has increased to approximately 0.480, suggesting that the additional predictors have improved the model's ability to explain the variance in the logarithm of price.

Coefficient Estimates:

The coefficient for log_sqft_living remains significant and has a value of approximately 0.7696. The coefficients for the additional predictors (waterfront, view, and renovated_last_10) are also significant: waterfront: Coefficient is approximately 0.5078, indicating that waterfront properties tend to have higher prices. view: Coefficient is approximately 0.2787, suggesting that properties with better views tend to have higher prices. renovated_last_10: Coefficient is approximately 0.2415, indicating that recently renovated properties tend to have higher prices. Statistical Significance: All coefficients are statistically significant with p-values < 0.05.

Model Fit: The adjusted R-squared value of approximately 0.480 indicates that the model with the additional predictors provides a better fit to the data compared to the previous model.

Overall, based on these results, we can conclude that the model including log_sqft_living, waterfront, view, and renovated_last_10 as predictors explains a significant portion of the variance in the logarithm of price and provides valuable insights into the factors influencing house prices.

[ ]

```
# Grouping the dummies together into lists form

dummies = ['floors', 'bedrooms', 'bathrooms', 'condition', 'grade']
floors_dummies = []
bedrooms_dummies = []
bathrooms_dummies = []
condition_dummies = []
grade_dummies = []

for col in list(kc.columns):
    for cate in dummies:
        if col.startswith(cate):
            eval(cate + '_dummies').append(col)
```

```
[ ]
# Defining the target variable and predictors

y = kc_transformed.log_price
X = kc_transformed[floors_dummies +
                     bedrooms_dummies +
                     bathrooms_dummies +
                     condition_dummies +
                     grade_dummies +
                     ['log_sqft_living']]

model = reg_qq_sced(y, X)

#The regression results show the coefficients, standard errors, t-values, and
p-values for each predictor in the model. Here's a summary of the results:

#R-squared: The coefficient of determination, indicating the proportion of the
variance in the dependent variable that is predictable from the independent
variables. In this case, the R-squared value is 0.583, which means that
approximately 58.3% of the variance in the logarithm of price can be explained by
the predictors in the model.

#Adjusted R-squared: A version of R-squared that adjusts for the number of
predictors in the model. It penalizes excessive complexity. The adjusted R-squared
value is also 0.583.

#F-statistic: A measure of the overall significance of the regression model. It
tests whether at least one of the predictors has a non-zero coefficient. Here, the
F-statistic is 878.4, with a very low p-value, indicating that the overall model is
statistically significant.

#Coefficients: The estimated coefficients for each predictor variable. These
represent the expected change in the dependent variable for a one-unit change in the
predictor, holding all other predictors constant.

#P-values: The p-values associated with each coefficient estimate. They indicate the
statistical significance of each predictor. In this context, a p-value less than
0.05 suggests that the predictor is statistically significant.

#Overall, the regression model appears to be statistically significant, with several
predictors showing significant associations with the logarithm of price.

# Defining target variable and predictors

y = kc_transformed.log_price
X = kc_transformed[floors_dummies +
                     condition_dummies +
                     grade_dummies +
                     ['view']+
                     ['waterfront']+
```

```
                    ['renovated_last_10']+
                    ['log_sqft_living']]

model = reg_qq_sced(y, X)



#The regression results indicate the following:

#R-squared: The coefficient of determination is 0.590, suggesting that approximately 59.0% of the variance in the logarithm of price can be explained by the predictors in the model.

#Adjusted R-squared: The adjusted R-squared value is also 0.590, indicating that the model's explanatory power is not compromised by the inclusion of additional predictors.

#F-statistic: The F-statistic is 1709.0 with a very low p-value, indicating that the overall model is statistically significant.

#Coefficients: The coefficients represent the estimated change in the logarithm of price for a one-unit change in the corresponding predictor variable, holding all other predictors constant. For example, a one-unit increase in log_sqft_living is associated with an increase of approximately 0.3822 in the logarithm of price.

#P-values: The p-values associated with each coefficient estimate indicate the statistical significance of the predictors. All predictors have p-values less than 0.05, suggesting that they are statistically significant in predicting the logarithm of price.

#Overall, the model appears to be statistically significant, with several predictors showing significant associations with the logarithm of price.
```