D600 Task 2
By Eric Williams

A:GITLAB REPOSITORY

Link provided for the gitlab repository was provided in the link submission.

B1:PROPOSAL OF QUESTION

My research question is: Are Square Footage and Number of Bedrooms accurate predictors of whether or not a house is luxury? If so, how accurately can they predict luxury housing status, and how much do incremental additions in square footage and number of bedrooms increase probability of a house being categorized as luxury?

B2:DEFINED GOAL

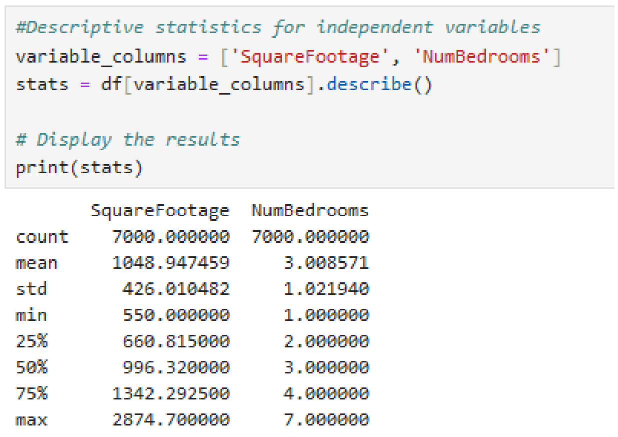
The goal of this data analysis is to quantify how well square footage and number of bedrooms predict the status of a house being classified as luxury. If I were doing this analysis for a building or real estate company, our multiple logistic regression could help us determine the value of unlisted homes or homes that have not been built yet. If we can determine which houses are luxury and are thus worth more in commission by training a computer model, it could make the business a lot of money and save a lot of time by letting the statistics make the predictions.

C1:VARIABLE IDENTIFICATION

In my regression, the independent variables are square footage and crime number of bedrooms. The dependent variable is the luxury status of the houses in the dataset. In this analysis, I expect that when we change square footage and number of bedrooms, luxury status will be affected (meaning it is dependent on the other two variables). We can test this using logistic regression. That is my justification for choosing these variables.

C2:DESCRIPTIVE STATISTICS

Here are the descriptive statistics for the independent variables:



```
#Descriptive statistics for independent variables
variable_columns = ['SquareFootage', 'NumBedrooms']
stats = df[variable_columns].describe()

# Display the results
print(stats)
```

```
       SquareFootage  NumBedrooms
count    7000.000000  7000.000000
mean     1048.947459     3.008571
std       426.010482     1.021940
min       550.000000     1.000000
25%       660.815000     2.000000
50%       996.320000     3.000000
75%      1342.292500     4.000000
max      2874.700000     7.000000
```

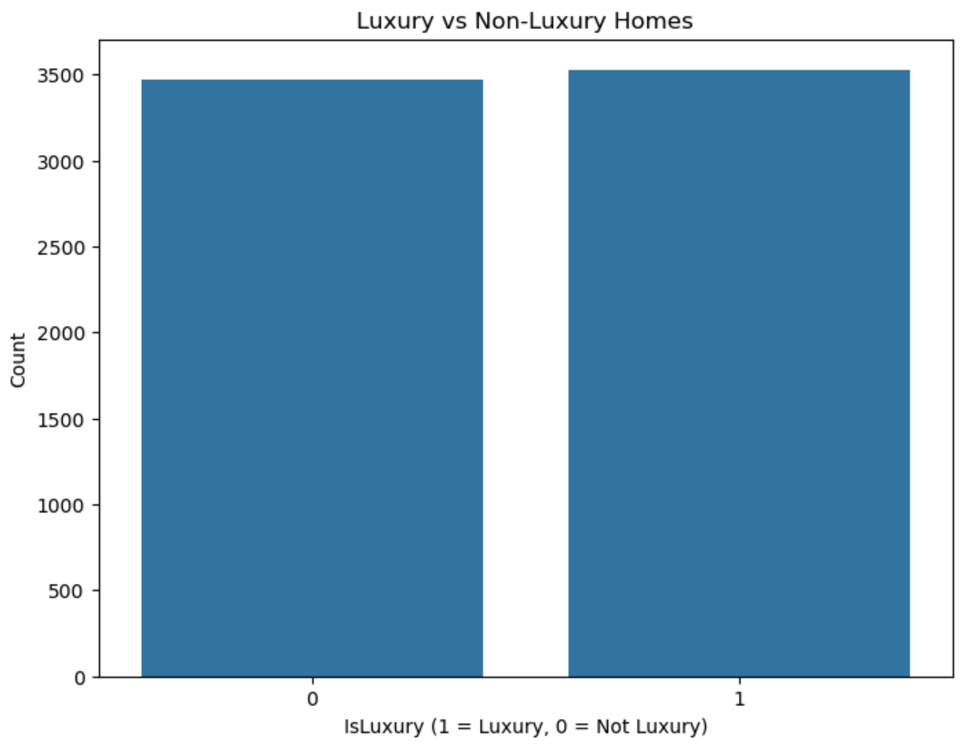
However, since the dependent variable is categorical, we need a different approach to look at the distribution of the data:

```python
#Descriptive statistics for the dependent variable
descriptive_stats = df['IsLuxury'].value_counts()
proportions = df['IsLuxury'].value_counts(normalize=True)

print("Frequency of each category:")
print(descriptive_stats)

print("\nProportion of each category:")
print(proportions)
```

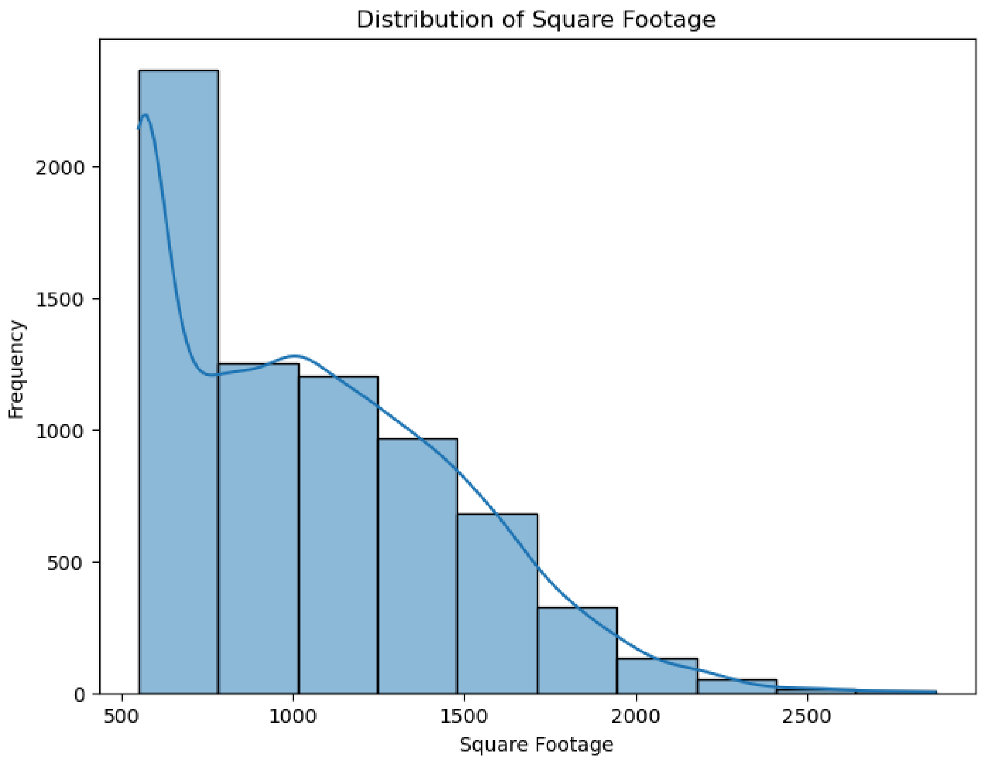
```
Frequency of each category:
IsLuxury
1    3528
0    3472
Name: count, dtype: int64

Proportion of each category:
IsLuxury
1    0.504
0    0.496
```

<u>C3:VISUALIZATIONS</u>

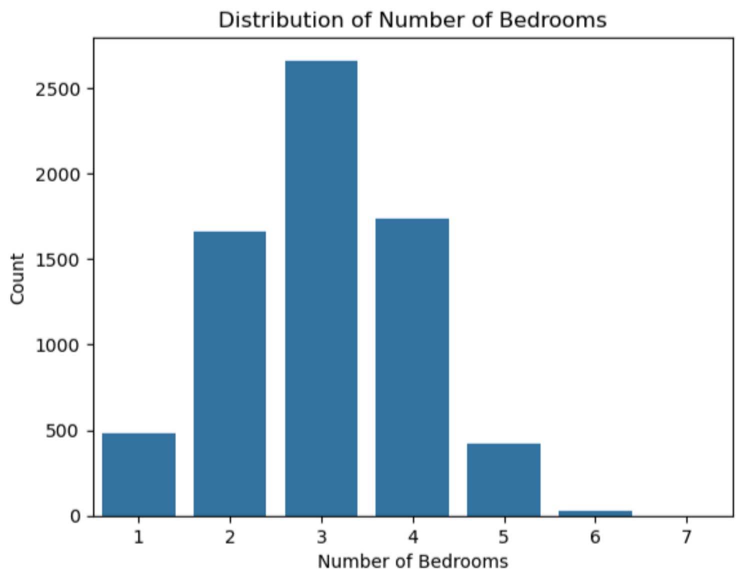
Here are the univariate visualizations of each variable (IsLuxury, SquareFootage, and NumBedrooms):



As you can see, our distribution of luxury and not luxury is roughly even.

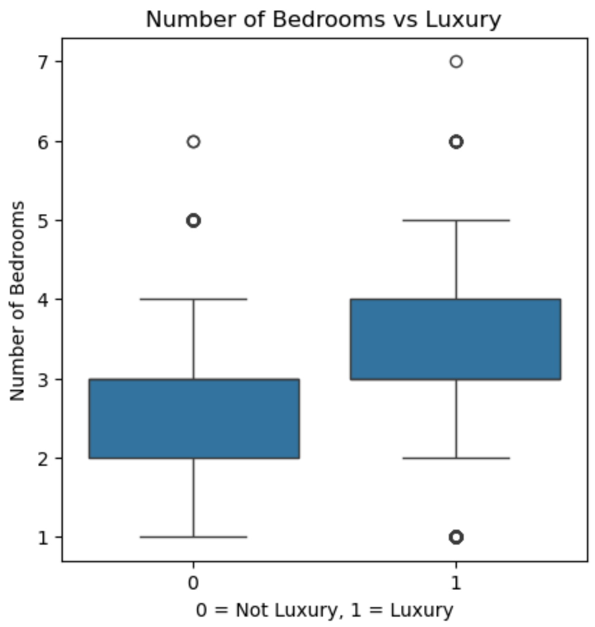


The distribution of Square Footage is skewed right. Our dataset heavily favors low square footage houses.

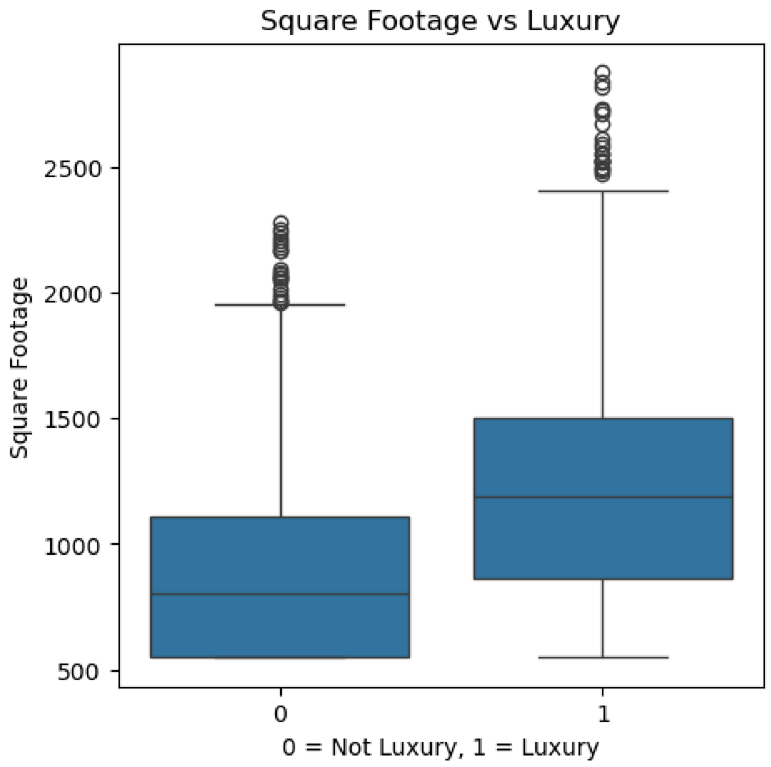


The distribution of the number of bedrooms is normal with very little skew. The most common number of bedrooms being 3.

Here are the bivariate visualizations variable of IsLuxury (the dependent variable) against SquareFootage and NumBedrooms:

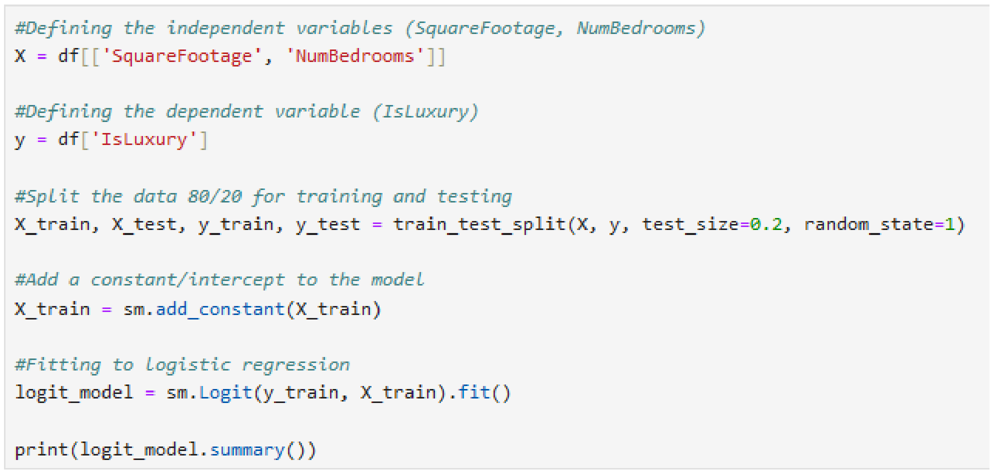


As you can see above, at a glance, luxury houses seem to have more bedrooms. Below, you can see how generally when square footage increases, luxury status increases.



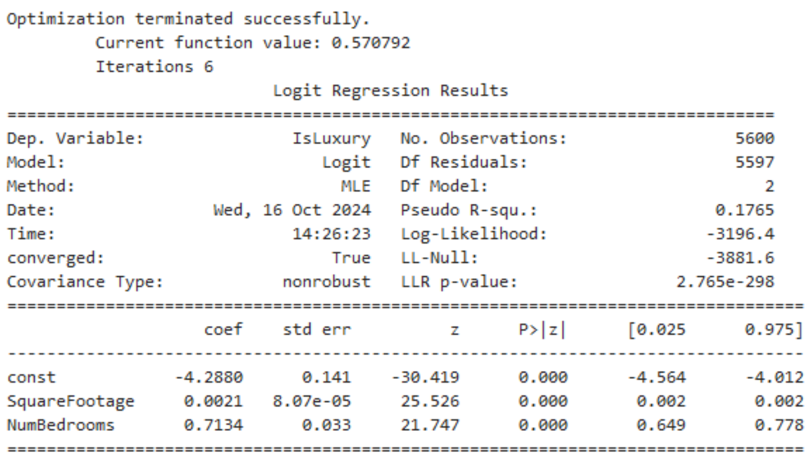
D1:SPLITTING THE DATA

This is the code I used to split the data. Note the 80/20 split, with the larger percentage assigned to training and the smaller percentage assigned to test the data.



```python
#Defining the independent variables (SquareFootage, NumBedrooms)
X = df[['SquareFootage', 'NumBedrooms']]

#Defining the dependent variable (IsLuxury)
y = df['IsLuxury']

#Split the data 80/20 for training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

#Add a constant/intercept to the model
X_train = sm.add_constant(X_train)

#Fitting to logistic regression
logit_model = sm.Logit(y_train, X_train).fit()

print(logit_model.summary())
```

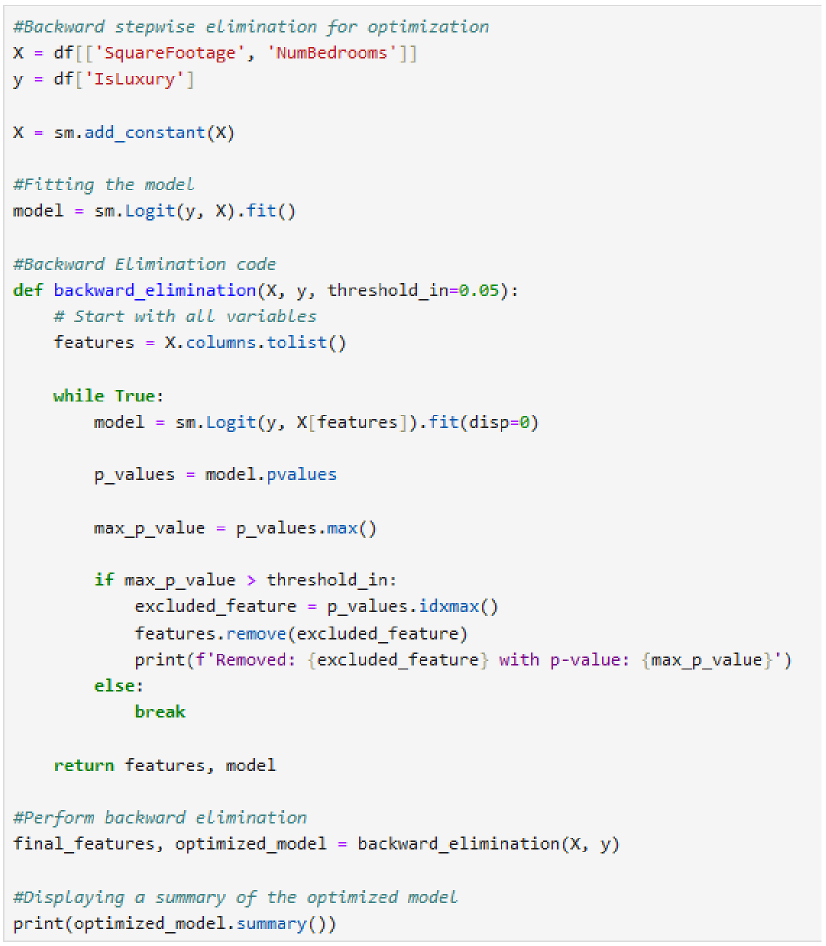
Here is the output for the code:



```
Optimization terminated successfully.
         Current function value: 0.570792
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:               IsLuxury   No. Observations:                 5600
Model:                          Logit   Df Residuals:                     5597
Method:                           MLE   Df Model:                            2
Date:                Wed, 16 Oct 2024   Pseudo R-squ.:                  0.1765
Time:                        14:26:23   Log-Likelihood:                -3196.4
converged:                       True   LL-Null:                       -3881.6
Covariance Type:            nonrobust   LLR p-value:                2.765e-298
=================================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------
const            -4.2880      0.141    -30.419      0.000      -4.564      -4.012
SquareFootage     0.0021   8.07e-05     25.526      0.000       0.002       0.002
NumBedrooms       0.7134      0.033     21.747      0.000       0.649       0.778
=================================================================================
```

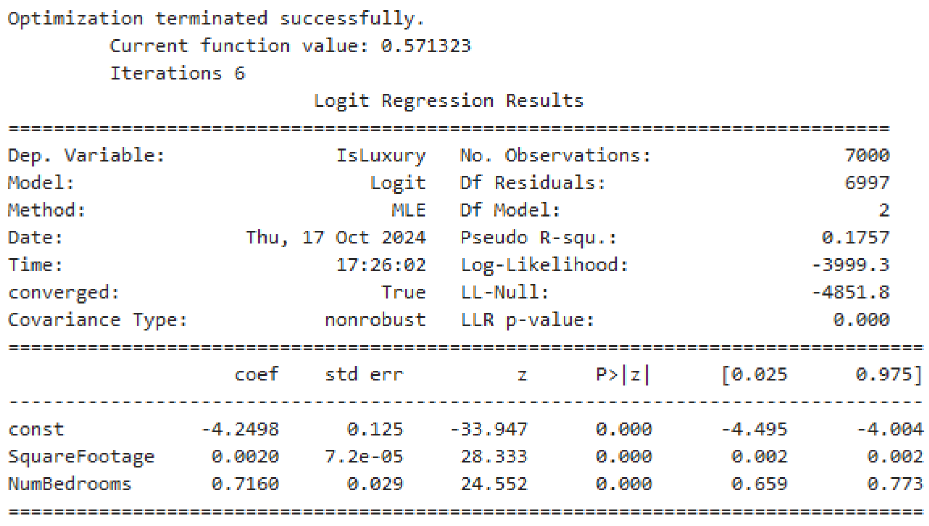
D2:MODEL OPTIMIZATION

Here is the code I used to optimize my model using the backward elimination method.



```python
#Backward stepwise elimination for optimization
X = df[['SquareFootage', 'NumBedrooms']]
y = df['IsLuxury']

X = sm.add_constant(X)

#Fitting the model
model = sm.Logit(y, X).fit()

#Backward Elimination code
def backward_elimination(X, y, threshold_in=0.05):
    # Start with all variables
    features = X.columns.tolist()

    while True:
        model = sm.Logit(y, X[features]).fit(disp=0)

        p_values = model.pvalues

        max_p_value = p_values.max()

        if max_p_value > threshold_in:
            excluded_feature = p_values.idxmax()
            features.remove(excluded_feature)
            print(f'Removed: {excluded_feature} with p-value: {max_p_value}')
        else:
            break

    return features, model

#Perform backward elimination
final_features, optimized_model = backward_elimination(X, y)

#Displaying a summary of the optimized model
print(optimized_model.summary())
```

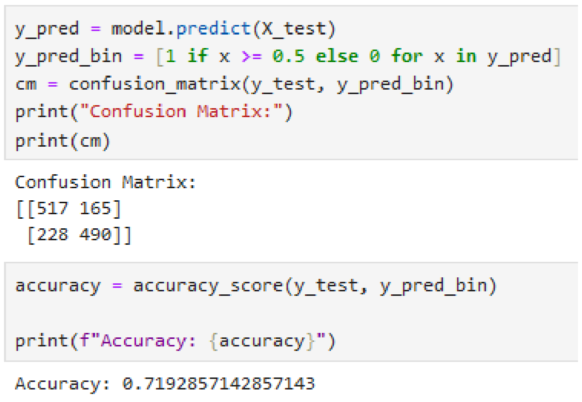
Here are the results after the optimization:



```
Optimization terminated successfully.
         Current function value: 0.571323
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:               IsLuxury   No. Observations:                 7000
Model:                          Logit   Df Residuals:                     6997
Method:                           MLE   Df Model:                            2
Date:                Thu, 17 Oct 2024   Pseudo R-squ.:                  0.1757
Time:                        17:26:02   Log-Likelihood:                -3999.3
converged:                       True   LL-Null:                       -4851.8
Covariance Type:            nonrobust   LLR p-value:                     0.000
=================================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------
const            -4.2498      0.125    -33.947      0.000      -4.495      -4.004
SquareFootage     0.0020    7.2e-05     28.333      0.000       0.002       0.002
NumBedrooms       0.7160      0.029     24.552      0.000       0.659       0.773
=================================================================================
```

D3:CONFUSION MATRIX AND ACCURACY

Here is the code and results of the confusion matrix on the optimized model, as well as the accuracy of the model:



```
y_pred = model.predict(X_test)
y_pred_bin = [1 if x >= 0.5 else 0 for x in y_pred]
cm = confusion_matrix(y_test, y_pred_bin)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[517 165]
 [228 490]]
```

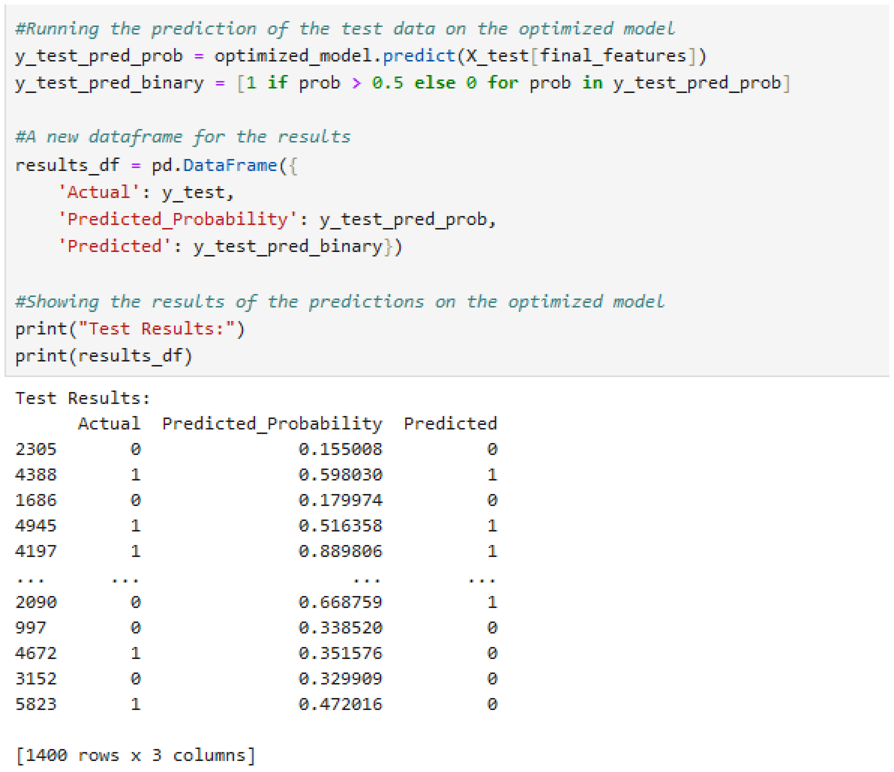
```
accuracy = accuracy_score(y_test, y_pred_bin)

print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.7192857142857143
```

An accuracy score of about 72% means our model is more effective than not. But it also shows that there is more to Luxury Homes than *just* the number of bedrooms and square footage.

D4:PREDICTION

Here is my code and the results of the prediction being run on the test data using the optimized model (with confusion matrix and accuracy shown above):



```
#Running the prediction of the test data on the optimized model
y_test_pred_prob = optimized_model.predict(X_test[final_features])
y_test_pred_binary = [1 if prob > 0.5 else 0 for prob in y_test_pred_prob]

#A new dataframe for the results
results_df = pd.DataFrame({
    'Actual': y_test,
    'Predicted_Probability': y_test_pred_prob,
    'Predicted': y_test_pred_binary})

#Showing the results of the predictions on the optimized model
print("Test Results:")
print(results_df)
```

```
Test Results:
      Actual  Predicted_Probability  Predicted
2305       0               0.155008          0
4388       1               0.598030          1
1686       0               0.179974          0
4945       1               0.516358          1
4197       1               0.889806          1
...      ...                    ...        ...
2090       0               0.668759          1
997        0               0.338520          0
4672       1               0.351576          0
3152       0               0.329909          0
5823       1               0.472016          0

[1400 rows x 3 columns]
```

E1:PACKAGES OR LIBRARIES LIST

Here is a list of packages I imported and why they were essential:

- Pandas: useful for making dataframes to store the data
- Seaborn: helpful for data visualizations, such as the scatterplots i made above
- Matplotlib: essential for visually plotting the univariate and bivariate statistics
- NumPy: Needed for running the difference between test and train set splits
- Statsmodels.api: needed to run the logistic regression
    - Specifically variance_inflation_factor for checking for multicollinearity
- Sklearn: essential for the specific tools I imported, namely train_test_split, classification_report, confusion_matrix, and accuracy_score

E2:OPTIMIZATION METHOD and **E3:METHOD JUSTIFICATION**

For optimization, I chose backward elimination. The goal of backward elimination is to remove the less significant variables in a series of steps. Eventually, this will leave us with just the variables that are the most important since the less helpful predictors are eliminated. I found that the optimization raised the accuracy of the model by about 1%.

However, backward elimination can have some drawbacks and in order to use it, we do have to make a few assumptions. Backward elimination assumes that the covariance matrix of the errors is correctly specified and that the multicollinearity will not affect the model. While using this approach, we are taking for granted that the model size is not too small and that it is a relatively simple model. If a model is too small or complex, the analysis may result in overfitting. Also, backward elimination does not do a great job of analyzing how variables are interrelated. Because sometimes variables can be collinear, working together to contribute to a result, taking away one of the contributing but less significant variables can hurt the model. In essence, we are assuming the model is not too complex and that the variables each contribute to the result and aren't necessarily collinear. This is not necessarily the case because when you build more bedrooms in your house, that adds to the square footage. We are assuming that this will not affect the model which may not be exactly correct and that by using backward elimination to eliminate less valuable variables, our model will be improved.

E4:ASSUMPTION SUMMARY

Here are four assumptions when running a logistic regression:

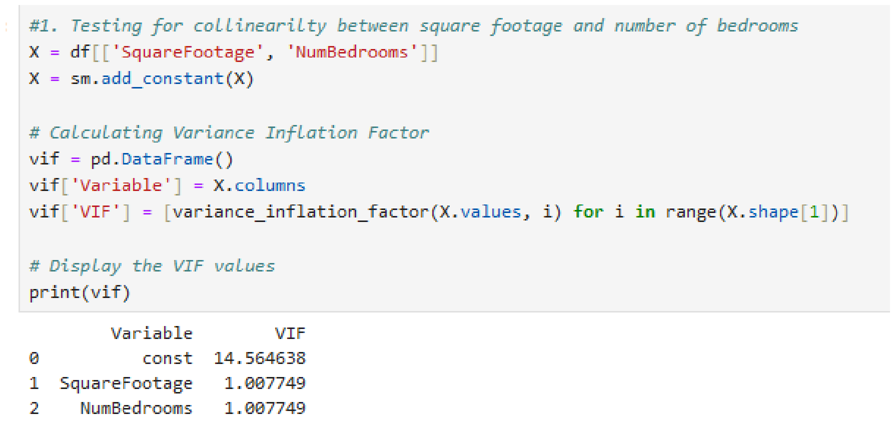
1. As mentioned before in backward elimination, we are assuming **no multicollinearity**. While it seems probable that square footage would correlate with the number of bedrooms, below I will provide verification otherwise on why this is not an issue for the model. In essence, adding square footage does not *necessarily* mean more bedrooms.
2. The **dependent variable needs to be categorical**. In this experiment, IsLuxury is our dependent variable and the data is entirely 1 or 0 entries. Even so, I will provide a test for showing that all IsLuxury entries are categorized into 1 (yes) or 0 (no).
3. **The observations need to be independent of each other,** meaning one row cannot be the same as, related to, or dependent on the other rows.

4. It is assumed that the data has a **sufficient sample size**. Generally, datasets under 500 can be more problematic when used to draw conclusions from the analysis. Below I will provide verification for the size of the data.

E5:VERIFICATION OF ASSUMPTIONS

Here is how I tested and verified each of the assumptions:

1. **No Multicollinearity**.

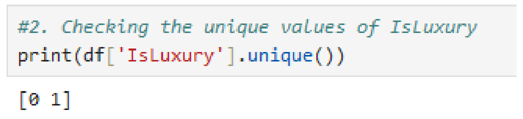


```
#1. Testing for collinearilty between square footage and number of bedrooms
X = df[['SquareFootage', 'NumBedrooms']]
X = sm.add_constant(X)

# Calculating Variance Inflation Factor
vif = pd.DataFrame()
vif['Variable'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Display the VIF values
print(vif)
```

```
        Variable        VIF
0          const  14.564638
1  SquareFootage   1.007749
2    NumBedrooms   1.007749
```

A VIF score of close to 1 means there is no multicollinearity between the two variables. According to this analysis, the square footage does not necessarily depend on the number of bedrooms, and vice versa.

2. **Dependent variable needs to be categorical**.

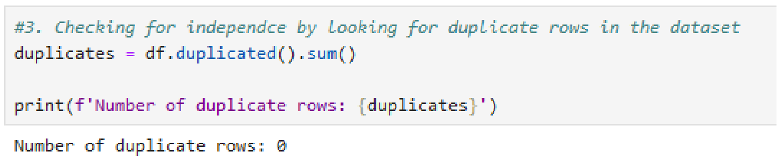


```
#2. Checking the unique values of IsLuxury
print(df['IsLuxury'].unique())
```

```
[0 1]
```

Because every datapoint in IsLuxury is either a 0 or 1, the entire column must be categorical--as in, every datapoint fits into the category of 0 (not luxury) or 1 (luxury).

3. **The observations need to be independent of each other,** meaning one row cannot be the same or related to the other rows.

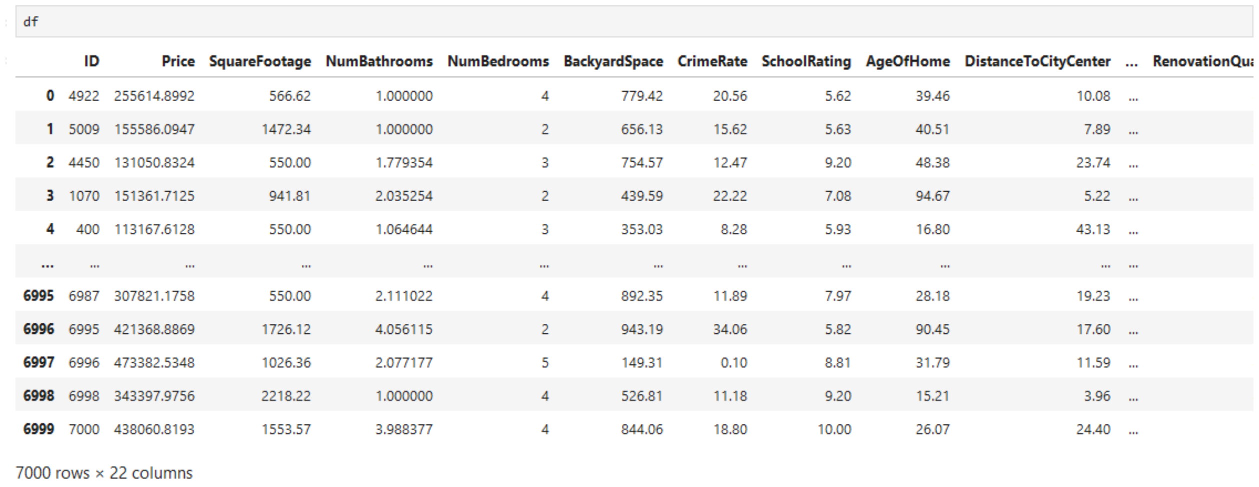


```
#3. Checking for independce by looking for duplicate rows in the dataset
duplicates = df.duplicated().sum()

print(f'Number of duplicate rows: {duplicates}')
```

```
Number of duplicate rows: 0
```

Since there are no duplicate rows, the rows are independent. We can assume the data was collected in a sensible way and that the data collected on individual houses was done correctly, meaning that there is no reason to believe the rows might be dependent on each other.

4. **Sufficient sample size**



```
df
```

```
        ID        Price  SquareFootage  NumBathrooms  NumBedrooms  BackyardSpace  CrimeRate  SchoolRating  AgeOfHome  DistanceToCityCenter  ...  RenovationQu
   0  4922  255614.8992         566.62      1.000000            4         779.42      20.56          5.62      39.46                 10.08  ...
   1  5009  155586.0947        1472.34      1.000000            2         656.13      15.62          5.63      40.51                  7.89  ...
   2  4450  131050.8324         550.00      1.779354            3         754.57      12.47          9.20      48.38                 23.74  ...
   3  1070  151361.7125         941.81      2.035254            2         439.59      22.22          7.08      94.67                  5.22  ...
   4   400  113167.6128         550.00      1.064644            3         353.03       8.28          5.93      16.80                 43.13  ...
 ...   ...          ...            ...           ...          ...            ...        ...           ...        ...                   ...  ...
6995  6987  307821.1758         550.00      2.111022            4         892.35      11.89          7.97      28.18                 19.23  ...
6996  6995  421368.8869        1726.12      4.056115            2         943.19      34.06          5.82      90.45                 17.60  ...
6997  6996  473382.5348        1026.36      2.077177            5         149.31       0.10          8.81      31.79                 11.59  ...
6998  6998  343397.9756        2218.22      1.000000            4         526.81      11.18          9.20      15.21                  3.96  ...
6999  7000  438060.8193        1553.57      3.988377            4         844.06      18.80         10.00      26.07                 24.40  ...

7000 rows × 22 columns
```

This was done at the beginning of the program. There are 7,000 rows of data for individual houses. Since our data is much more than 500 rows of data, our sample size is reasonably large.

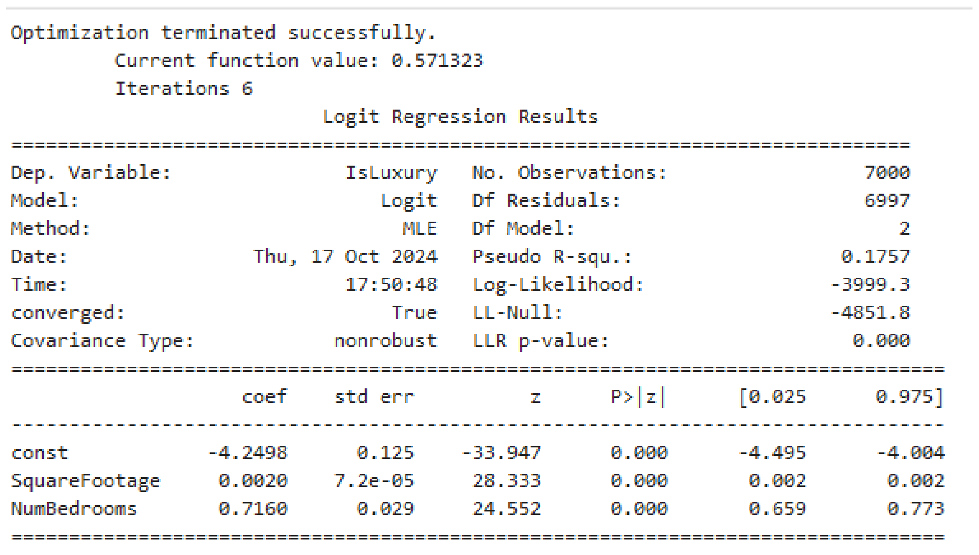
E6:EQUATION

Here is the logistic regression equation for two variables:

$$Logit(Y(p)) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

In this case, the outcome value Y is the Luxury status.
β_0 is the intercept when the independent variables are 0.
β_1 is the coefficient for square footage.
β_2 is the coefficient for the number of bedrooms.

Here are the results given by the model:



```
Optimization terminated successfully.
         Current function value: 0.571323
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:               IsLuxury   No. Observations:                 7000
Model:                          Logit   Df Residuals:                     6997
Method:                           MLE   Df Model:                            2
Date:                Thu, 17 Oct 2024   Pseudo R-squ.:                  0.1757
Time:                        17:50:48   Log-Likelihood:                -3999.3
converged:                       True   LL-Null:                       -4851.8
Covariance Type:            nonrobust   LLR p-value:                     0.000
=================================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------
const            -4.2498      0.125    -33.947      0.000      -4.495      -4.004
SquareFootage     0.0020    7.2e-05     28.333      0.000       0.002       0.002
NumBedrooms       0.7160      0.029     24.552      0.000       0.659       0.773
=================================================================================
```

The constant coefficient β_0= -4.24, which means the probability of being luxury is -4.24 when square footage and number of bedrooms is 0. Although because a house cannot have no square footage and wouldn't have no bedrooms, this number isn't particularly telling.

The square footage coefficient is $\beta_1 = 0.0020$, which means when a square foot is added to a home, the log-odds of the house being luxury increases by 0.002.

The NumBedrooms coefficient $\beta_2 = 0.716$ means that for every bedroom in a house, the log-odds of the house being luxury increases by 0.716

Thus our final equation is
$logit(IsLuxury(p)) = -4.24 + 0.002\, x\, (Square\ Footage) + 0.716\, x\, (Crime\ Rate)$

E7:MODEL METRICS

Here I will discuss the model metrics by addressing each of the following:

1. **The accuracy for the test set**

The accuracy for the test set is 0.7142857142857143 (rounded to 71.43%)

2. **The comparison of the accuracy of the training set to the accuracy of the test set**

The accuracy for the test set is 0.7142857142857143 (rounded to 71.43%)
The accuracy of the training set is 0.7141071428571428 (rounded to 71.41%)

To compare the two, note that the above numbers are very similar, but have a difference in accuracy of 0.02%.

3. **The comparison of the confusion matrix for the training set to the confusion matrix of the test set**

Here is the confusion matrix for the training set:

```
[[2035 755]
[ 846 1964]]
```

Here is the confusion matrix for the test set:

```
[[521 161]
[239 479]]
```

To compare the two confusion matrices, we must scale up the test set matrix by 4 because of the 20/80 split. Here is the scaled up version of the confusion matrix for the test set:

```
[[2084 644]
[956  1916]]
```

By comparing the two matrices, we can find that the training set:

Produced 49 less true negatives, proportionally
Produced 111 more false positives, proportionally
Produced 110 less false negatives, proportionally
Produced 48 more true positives, proportionally

The difference in accuracy score being 0.02% is reflected in these numbers.

E8:RESULTS AND IMPLICATIONS and **E9:COURSE OF ACTION**

Our results tell us that increased square footage and number of bedrooms is a good predictor of being a luxury home. However, our model was only 72% accurate so other factors should be taken into account when predicting luxury home status. My first recommendation would be to do an analysis involving ALL the most important factors because that would result in a stronger, more accurate prediction model. Including renovations, crime rate, price, number of bathrooms, yard size, and more will produce a more accurate predicting model. However, the model I created will still be helpful on a general level of predicting whether or not a home is luxury. Thus, when trying to predict luxury house status, this model should be used until a better model becomes available. To be more specific, I recommend using this model to predict whether or not a house will be luxury or not in the following circumstances:

A. Houses are being built to certain specifications to be luxury and thus be sold for more profit
B. Renovations and improvements are made when trying to sell a house with enough square footage and bedrooms to be luxury, thus upgrading to make it luxury and increase profits
C. Predicting whether a potential house to sell will be luxury, resulting in greater commissions for a real estate company.

The answer to our question “Are Square Footage and Number of Bedrooms accurate predictors of whether or not a house is luxury?” is yes, and we can predict whether or not a house is luxury based on those two factors using logistic regression with about 72% accuracy.

Sources

Because of the similarity of the rubric for this task to task 1, I used the same layout as the previous task but updated all the code, variables, and visualizations.

No other sources were used except for official WGU course materials.