D600 Task 3

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: How well do ALL the non-categorical, non-binary variables predict the price of a home? What are the principal components in predicting the price of a house and how well do they predict price?

## B2:DEFINED GOAL

The goal of this data analysis is to quantify how the non-categorical, non-binary variables affect the cost of home prices. If I were doing this analysis for a real estate company, a multiple linear regression on the PCA values would help provide an estimate for the value of currently unlisted houses or for future houses based on several variables. Being able to estimate a house's value before building it will help the company make better investment decisions and optimize use of land.

## C1:PCA USE

PCA, or Principal Component Analysis, is a highly specialized technique that will allow us to analyze many different variables in our model to create predictions. PCA takes many variables (or "dimensions") and combines and reduces them into just a few components, simultaneously using every variable for data but simplifying the results. This will be useful because we have many different variables that can help determine the price of a home. We have previously run predictions using just 3 variables for logistic and linear regression, effectively ignoring the rest of the data. Using PCA, we will be able to model using all variables by reducing them into three variables.

## C2:PCA ASSUMPTION

One key assumption for PCA is that our variables are linear and continuous. Because we will be doing a linear regression at the end, if we have variables that are exponential or grow in other non-linear ways, our linear model will not analyze the data properly. Also because of the calculations we will be doing, the input data for the PCA needs to be purely numerical, which means we are assuming the numerical variables have enough predicting power to do a proper analysis.

## D1:VARIABLE IDENTIFICATION

As previously mentioned, we can only use numerical data for our inputs. Thus, I will be including: Price, Square Footage, Number of Bathrooms, Number of Bedrooms, Backyard Space, Crime Rate, School Rating, Age of Home, Distance to City Center, Employment Rate, Property Tax Rate, Renovation Quality, Local Amenities, Transport Access, Previous Sales Price, and Windows. ID will not be included because it is an arbitrary label. Fireplace, House Color,

## D2:STANDARDIZED DATA

Here is my code standardizing the data, applying the PCA, displaying variance, and then converting the standardized data to an dataframe to save and attach to this project:

```
#Defining the continuous variables for PCA
X = df[['SquareFootage', 'NumBathrooms', 'NumBedrooms', 'BackyardSpace',
        'CrimeRate', 'SchoolRating', 'AgeOfHome', 'DistanceToCityCenter',
        'EmploymentRate', 'PropertyTaxRate', 'RenovationQuality', 'LocalAmenities',
        'TransportAccess', 'PreviousSalePrice', 'Windows']]


#Standardizing
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

#Applying PCA
pca = PCA(n_components=None)
X_pca_all = pca.fit_transform(X_scaled)
```

## D3:DESCRIPTIVE STATISTICS

First I'll describe price, my dependent variable

```
#Describing Price, the dependent variable
dependent_variable = df['Price']
descriptive_stats_dependent = dependent_variable.describe()

print(descriptive_stats_dependent)
```

```
count    7.000000e+03
mean     3.072820e+05
std      1.501734e+05
min      8.500000e+04
25%      1.921075e+05
50%      2.793230e+05
75%      3.918781e+05
max      1.046676e+06
Name: Price, dtype: float64
```

Then the independent variables:

```
#Describing independent variables
descriptive_stats_independent = X.describe()

# Display the statistics
print(descriptive_stats_independent)
```

```
       SquareFootage  NumBathrooms  NumBedrooms  BackyardSpace    CrimeRate  \
count    7000.000000   7000.000000  7000.000000    7000.000000  7000.000000   
mean     1048.947459      2.131397     3.008571     511.507029    31.226194   
std       426.010482      0.952561     1.021940     279.926549    18.025327   
min       550.000000      1.000000     1.000000       0.390000     0.030000   
25%       660.815000      1.290539     2.000000     300.995000    17.390000   
50%       996.320000      1.997774     3.000000     495.965000    30.385000   
75%      1342.292500      2.763997     4.000000     704.012500    43.670000   
max      2874.700000      5.807239     7.000000    1631.360000    99.730000   

       SchoolRating    AgeOfHome  DistanceToCityCenter  EmploymentRate  \
count   7000.000000  7000.000000           7000.000000     7000.000000   
mean       6.942923    46.797046             17.475337       93.711349   
std        1.888148    31.779701             12.024985        4.505359   
min        0.220000     0.010000              0.000000       72.050000   
25%        5.650000    20.755000              7.827500       90.620000   
50%        7.010000    42.620000             15.625000       94.010000   
75%        8.360000    67.232500             25.222500       97.410000   
max       10.000000   178.680000             65.200000       99.900000   

       PropertyTaxRate  RenovationQuality  LocalAmenities  TransportAccess  \
count      7000.000000        7000.000000     7000.000000      7000.000000   
mean          1.500437           5.003357        5.934579         5.983860   
std           0.498591           1.970428        2.657930         1.953974   
min           0.010000           0.010000        0.000000         0.010000   
25%           1.160000           3.660000        4.000000         4.680000   
50%           1.490000           5.020000        6.040000         6.000000   
75%           1.840000           6.350000        8.050000         7.350000   
max           3.360000          10.000000       10.000000        10.000000   

       PreviousSalePrice      Windows  
count       7.000000e+03  7000.000000  
mean        2.845094e+05    16.248857  
std         1.857340e+05     8.926479  
min        -8.356902e+03    -6.000000  
25%         1.420140e+05    11.000000  
50%         2.621831e+05    15.000000  
75%         3.961212e+05    20.000000  
max         1.296607e+06    63.000000  
```

## E1:MATRIX DETERMINATION

Here is my code and the result of creating a matrix of the principal components:

```
#Creating a matrix of all principal components
pca_matrix = pd.DataFrame(X_pca_all, columns=[f'PC{i+1}' for i in range(X_pca_all.shape[1])])

print(pca_matrix.head())
```

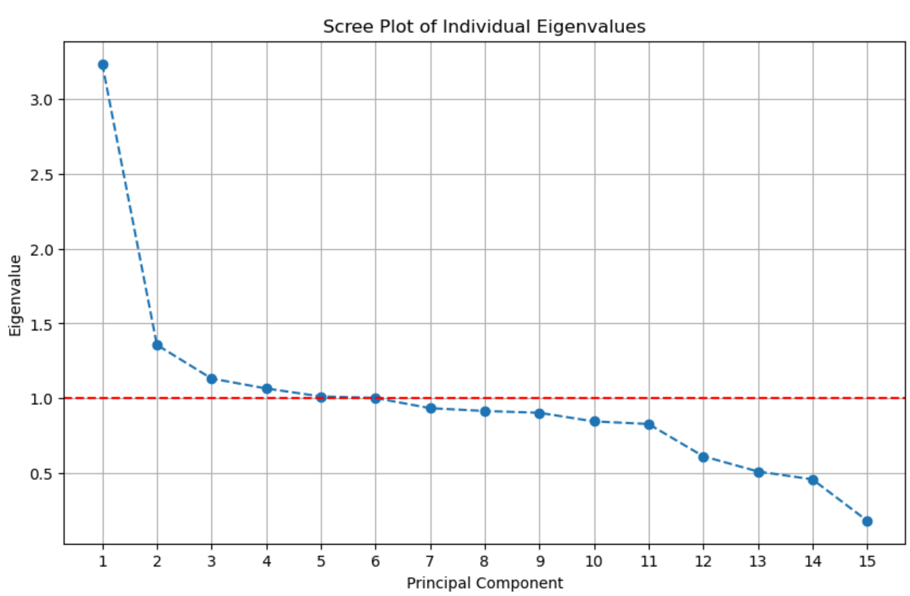
```
        PC1       PC2       PC3       PC4       PC5       PC6       PC7  \
0 -0.902343  0.333013 -1.569851  0.094842 -0.268075 -0.284527  1.616614
1 -0.804669 -0.665626 -0.878430 -1.677154  0.441913 -0.022723 -0.506909
2 -0.282699 -1.931889 -1.579538  0.933843  1.210100  1.383809  1.112252
3 -1.055752 -0.075489 -0.216879 -0.200526 -0.015122 -0.576201 -0.545630
4 -2.170138 -1.071303 -0.556838 -0.098562 -0.655281  0.985124  0.078494

        PC8       PC9      PC10      PC11      PC12      PC13      PC14  \
0 -0.610619 -0.246474  0.500900 -0.589744  0.497684 -0.165569  0.855532
1  0.755633 -0.336947  1.481971  0.266378 -0.051247  0.448011  0.150985
2  0.422386 -0.321725 -0.674668 -0.122049 -0.812482  0.337412 -0.801456
3  1.414695 -0.347068  0.297748 -1.228864 -0.789727  0.365840 -0.130236
4  0.234006 -1.448756 -1.008390  2.246380 -0.369723  0.467352  0.343461

       PC15
0  0.090008
1 -0.707817
2 -0.192023
3 -0.229669
4 -0.283988
```

## E2:TOTAL PRINCIPAL COMPONENTS

Choosing the number of principal components is not an exact science. If we use a percentage threshold (such as 70%), our results would be very different from a subjective test like the elbow test. Here is the scree plot I created to analyze this, including an eigenvalue line at y=1:



The Kaiser test (where the eigenvalue is 1 or greater) indicates that we should keep the first six components:

```
eigenvalues
```

```
array([3.23394031, 1.35730882, 1.13251244, 1.06606718, 1.0129399 ,
       1.00251743, 0.9345122 , 0.91564176, 0.90364539, 0.8460953 ,
       0.82888336, 0.61347521, 0.51192504, 0.45915282, 0.183526  ])
```

However, visually, the elbow test indicates a significant bend after the 2nd principal component, meaning the data levels off after the 3rd component. But because the explained variance of these three components only sums to 38%, in this case, I would favor the Kaiser rule that sums to about 59%. That is why I have chosen the top six components moving forward.

## E3:VARIANCE

I included the variance of every variable below, however, the six components that will be included moving forward are the first six on the list:

```
#Displaying explained variance
explained_variance = pca.explained_variance_ratio_
print(f'Explained Variance Ratio: {explained_variance}')
```

```
Explained Variance Ratio: [0.21556522 0.09047433 0.07549004 0.07106099 0.06751968 0.06682495
 0.06229191 0.06103406 0.06023442 0.0563983  0.055251   0.04089251
 0.03412346 0.03060581 0.01223332]
```

## E4:PCA SUMMARY

The PCA used the input of 15 variables to define the principal components of the dataset. Given the Kaiser test from above, I have 6 principal components that will help represent the dataset. It is much easier to work with six variables/dimensions than 15 and they also account for 59% of the variance. Next, I will use this data on a Multiple Linear Regression to see how accurately it can predict the price of a house.  This will show how effective our PCA is at prediction. These results will be displayed below.

## F1:SPLITTING THE DATA

Here is the code I used to split the data using an 80/20 split (the smaller split being for testing):

```
#Train Test Split our dependent variable (price) and our 6 principal component variables
y = df['Price']

X = final_df.drop(columns=['Price'])
y = final_df['Price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)


#Fitting the linear regression
model = sm.OLS(y_train, sm.add_constant(X_train)).fit()

print(model.summary())
```

## F2:MODEL OPTIMIZATION

For optimizing the model, I used the backward elimination method:

```
import statsmodels.api as sm

#Backward elimination
def backward_elimination(X, y, significance_level=0.05):
    X = sm.add_constant(X)
    model = sm.OLS(y, X).fit()

    print(model.summary())

    p_values = model.pvalues

    while p_values.max() > significance_level:
        remove_var = p_values.idxmax()
        X = X.drop(columns=remove_var)
        model = sm.OLS(y, X).fit()
        p_values = model.pvalues
        print(model.summary())

    return model

final_model = backward_elimination(X_train, y_train)
```

Here were the results. Note that the results come in three sets because two variables are dropped as a result of the process:

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.679
Model:                            OLS   Adj. R-squared:                  0.679
Method:                 Least Squares   F-statistic:                     1976.
Date:                Wed, 23 Oct 2024   Prob (F-statistic):               0.00
Time:                        03:10:34   Log-Likelihood:                -71558.
No. Observations:                5600   AIC:                         1.431e+05
Df Residuals:                    5593   BIC:                         1.432e+05
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       3.074e+05   1146.784    268.021      0.000    3.05e+05     3.1e+05
PC1            0.6655      0.006    108.067      0.000       0.653       0.678
PC2           38.4532      3.317     11.592      0.000      31.950      44.956
PC3           -5.7134      4.140     -1.380      0.168     -13.828       2.402
PC4         -104.4691     36.330     -2.876      0.004    -175.689     -33.249
PC5            4.4498     63.496      0.070      0.944    -120.028     128.927
PC6         -433.0883     98.698     -4.388      0.000    -626.574    -239.603
==============================================================================
Omnibus:                      338.901   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              836.343
Skew:                           0.359   Prob(JB):                    2.46e-182
Kurtosis:                       4.752   Cond. No.                     1.86e+05
==============================================================================
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.679
Model:                            OLS   Adj. R-squared:                  0.679
Method:                 Least Squares   F-statistic:                     2372.
Date:                Wed, 23 Oct 2024   Prob (F-statistic):               0.00
Time:                        03:10:34   Log-Likelihood:                -71558.
No. Observations:                5600   AIC:                         1.431e+05
Df Residuals:                    5594   BIC:                         1.432e+05
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       3.074e+05   1146.642    268.055      0.000    3.05e+05     3.1e+05
PC1            0.6655      0.006    108.077      0.000       0.653       0.678
PC2           38.4530      3.317     11.593      0.000      31.951      44.955
PC3           -5.7140      4.139     -1.380      0.167     -13.828       2.400
PC4         -104.4580     36.326     -2.876      0.004    -175.671     -33.245
PC6         -433.0526     98.687     -4.388      0.000    -626.518    -239.587
==============================================================================
Omnibus:                      338.937   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              836.228
Skew:                           0.359   Prob(JB):                    2.60e-182
Kurtosis:                       4.751   Cond. No.                     1.86e+05
==============================================================================
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.679
Model:                            OLS   Adj. R-squared:                  0.679
Method:                 Least Squares   F-statistic:                     2964.
Date:                Wed, 23 Oct 2024   Prob (F-statistic):               0.00
Time:                        03:10:34   Log-Likelihood:                -71559.
No. Observations:                5600   AIC:                         1.431e+05
Df Residuals:                    5595   BIC:                         1.432e+05
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       3.074e+05   1146.728    268.039      0.000    3.05e+05     3.1e+05
PC1            0.6655      0.006    108.067      0.000       0.653       0.678
PC2           38.4313      3.317     11.586      0.000      31.928      44.934
PC4         -104.4481     36.329     -2.875      0.004    -175.667     -33.229
PC6         -432.4779     98.695     -4.382      0.000    -625.957    -238.998
==============================================================================
Omnibus:                      339.429   Durbin-Watson:                   1.994
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              836.898
Skew:                           0.360   Prob(JB):                    1.86e-182
Kurtosis:                       4.752   Cond. No.                     1.86e+05
==============================================================================
```

## F3:MEAN SQUARED ERROR

I calculated the mean squared error of the optimized model on the training set. The error dropped from 6,629,445,434 to 6,626,095,870, or by roughly 3 million, as a result of the backward elimination method.

## F4:MODEL ACCURACY

Here is the code I used to run the prediction on the test dataset using the optimized regression model from part F2 to give the accuracy of the prediction model based on the MSE

```
#Predictors after backward elimination
#Run the prediction on the test dataset using the optimized regression model from part F2 to give the accuracy of the prediction
#model based on the mean squared error
X_train_optimized = X_train[final_model.model.exog_names[1:]]
X_test_optimized = X_test[final_model.model.exog_names[1:]]

#Predictions on the training set
y_train_pred = final_model.predict(sm.add_constant(X_train_optimized))

#Predictions on the test set
y_test_pred = final_model.predict(sm.add_constant(X_test_optimized))

#MSE on the test set
mse_test = mean_squared_error(y_test, y_test_pred)
print(f'Mean Squared Error on test set: {mse_test:.2f}')

#MSE on the training set
mse_train = mean_squared_error(y_train, y_train_pred)
print(f'Mean Squared Error on train set: {mse_train:.2f}')
```

```
Mean Squared Error on test set: 6626095870.32
Mean Squared Error on train set: 7356194870.71
```

As you can see the output of the Mean Squared Error on the test set was 6626095870.32 while the Mean Squared Error on the train set was 7356194870.71.

As far as gauging the accuracy of the optimized model, the overall mean squared error of $6,626,095,870, predicts the accuracy of home costs after PCA and Multiple Linear Regression within an average of $81,407.

## G1: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 regression
- Sklearn: essential for the specific tools I imported, namely the PCA, StandardScaler to scale the PCA properly, test_train_split as necessary for the regression, and mean_squared_error to calculate the mean squared error.
- Lastly we need statsmodels.stats.outliers_influence import variance_inflation_factor to prove (below) that the variables in our model are not multicollinear.

## G2: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 from a mean squared error of $6,629,445,434 to $6,626,095,870. This difference in mean squared error was $3,349,564, meaning the average accuracy rose by $1,830 per house. Backward elimination works as long as the dataset is relatively large and there is no multicollinearity between the independent variables.

## G3:VERIFICATION OF ASSUMPTIONS

Essentially, to ensure the validity of backward elimination, we need to prove that there is no significant multicollinearity and that our sample size is not too small. Since the data has well over 500 data points, the latter requirement is met. To ensure there is no significant multicollinearity we can calculate the Variance Inflation Factor between the remaining variables. Since the values below are very close to 1, there is very little multicollinearity.

```
#Calculating Variance Inflation Factor (VIF) for the predictor variables
vif_data = pd.DataFrame()
vif_data["Feature"] = X_train_optimized.columns
vif_data["VIF"] = [variance_inflation_factor(X_train_optimized.values, i) for i in range(X_train_optimized.shape[1])]

print(vif_data)
```

```
  Feature       VIF
0     PC1  1.000082
1     PC2  1.000066
2     PC4  1.000115
3     PC6  1.000156
```

## G4:EQUATION

Here is the multiple predictor model equation for four variables is:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \epsilon$$

Y is the outcome value, which is the price.
$\beta_0$ = 307,400 is the intercept/constant, or the price when all our other variables are 0.
$\beta_1 = 0.6655$ is the coefficient on PC1 ($x_1$).
$\beta_2 = 38.4313$ is the coefficient on PC2 ($x_2$).
$\beta_3 = -104.4481$ is the coefficient on PC4 ($x_3$).
$\beta_4 = -432.4779$ is the coefficient on PC6 ($x_4$).
$\epsilon$ = $81,407 which is the average value of error on our model.

This means the final equation for our model is:

$$Y = 307,400 + 0.6655x_1 + 38.4313x_2 - 104.4481x_3 - 432.4779x_4 + 81,407.$$

Where $x_1$, $x_2$ , $x_3$, and $x_4$ are the PC1, PC2, PC4, and PC6 respectively.

## G5:MODEL METRICS

Here is my discuss of the model metrics by addressing:

1. The $R^2$ and adjusted $R^2$ of the training set

Both the $R^2$ value and adjusted $R^2$ value of my optimized model were 0.679. This means that roughly 68% of the variance in Price was due to the variables from the PCA. While this means the model did fairly well, that still leaves about 32% of the variance unexplained by the model.

2. The comparison of the MSE for the training set to the MSE of the test set

Next, I used the following code to compare the MSE for the training and test set:

```python
#MSE for training set
mse_train = mean_squared_error(y_train, y_train_pred)

#MSE for test set
mse_test = mean_squared_error(y_test, y_test_pred)

print(f'Mean Squared Error for Training Set: {mse_train}')
print(f'Mean Squared Error for Test Set: {mse_test}')
```

```
Mean Squared Error for Training Set: 7356194870.711278
Mean Squared Error for Test Set: 6626095870.316331
```

As expected, the MSE for training is higher than on the test set.

Mean Squared Error for Training Set: 7356194870
Mean Squared Error for Test Set: 6626095870

This means our model is not overfitted. However, these values are relatively high. That does mean the difference in prediction was $4,367 between the test set and training set.

## G6:RESULTS AND IMPLICATIONS and G7:COURSE OF ACTION

Our results tell us that the PCA can predict the cost of houses within $81,407 if we are given the 15 variables we used. This is a big improvement over the other models and should be used as the golden standard moving forward. However, as previously mentioned, our $R^2$ value was 0.679, meaning about only 68% of the variance in Price was due to the variables from the PCA. This means there is still some room for improvement and we might be able to come up with a better predicting tool in the future.

If a real estate company is trying to predict the value of houses before they are built, they can use this model to predict the value of the house within $81,000 before it's even built. Similarly, if they are looking for what houses sell to maximize their profits, I recommend using this model to predict home values (with the assumption that the value will be, on average, within $81,000).

The answer to our question "How well do ALL the non-categorical, non-binary variables predict the price of a home?" The answer is that a combination of the 15 variables, after undergoing PCA, MLR, and optimization can predict the value of a house (whether it exists or not) within $81,000.

## Sources

Because of the similarity of the rubric for this task to task 1 and 2, I used the same layout as the previous tasks but updated all the code, variables, and visualizations.

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