D600 UGN1 Task 3

Principal Component Analysis

Shanay Murdock
smurd32@wgu.edu
011377935

D600 Statistical Data Mining
MS, Data Analytics

A. GitLab

Link to GitLab repository (working_branch_3):

https://gitlab.com/wgu-gitlab-environment/student-repos/smurd32/d600-statistical-data-mining/-/tree/working_branch_3?ref_type=heads

The link is also provided in the “Comments to Evaluator” upon submission.

B. Purpose of Analysis

B1. Proposal of Question

How do various continuous factors contribute to predicting the price of a house, and can we reduce dimensionality through PCA to identify significant components that have the greatest impact on house pricing? Can the data available be used to help a home buyer predict what the cost of a home will be based on the characteristics they wish to attain?

B2. Defined Goal

This question aims to leverage PCA to determine the key components that can predict a house's `Price` from variables like `Square Footage`, `NumBathrooms`, `NumBedrooms`, `BackyardSpace`, `CrimeRate`, `SchoolRating`, `AgeOfHome`, `DistanceToCityCenter`, `EmploymentRate`, `PropertyTaxRate`, `RenovationQuality`, `LocalAmenities`, and `TransportAccess`, and `PreviousSalePrice` to simplify the model without losing predictive power.

C. PCA Reasoning

C1. PCA Use

Principal Component Analysis (PCA) is a powerful unsupervised learning technique that helps simplify datasets by focusing on their most important aspects. As a form of dimensionality reduction, PCA makes the subsequent regression analysis more efficient and interpretable as it reduces the number of factors needed for testing, ensures the data is uncorrelated (a requirement for linear regression), and retains the most influential factors needed to make accurate predictions.

Principal components are new variables that are derived out of linear combinations of the original variables. They are uncorrelated, so they work as a foundational step in preparing for linear regression while reducing the potential for overfitting errors. The first few principal components typically capture the most significant features of the data, meaning you can retain fewer components instead of the entire dataset. The features deemed most significant maintain the highest variance for explanation after normalizing the data to ensure that each variable contributes equally to the analysis.

By using PCA, we can expect the following outcomes: reduced overfitting, improved model interpretability, efficiency, and elimination of multicollinearity. By eliminating less significant features, PCA helps to reduce overfitting, particularly when dealing with high-dimensional datasets. PCA helps to improve model interpretability by focusing on the principal components that capture the most variance. PCA provides additional efficiency when running the data through a model by reducing the number of components that need to be analyzed. And PCA eliminates multicollinearity by scaling the data so all data is originally viewed as equally significant, it combines variables in a way to capture the maximum variance in the data, and effectively discards components that contribute little to the information of the dataset (Ahmad, 2023).

C2. PCA Assumption

One assumption of Principal Component Analysis (PCA) is that the relationships among the selected variables are all linear. PCA works by transforming the original dataset of continuous variables into a new set of uncorrelated principal components. Non-linear relationships might not be captured effectively. If the data is non-linear, other dimensionality reduction techniques are more appropriate.

In addition, all variables need to be numeric (continuous), and therefore any string (object), binary, and categorical data has been excluded from the analysis.

D. Data Preparation Summary

D1. Variable Identification

The following variables will be used as they satisfy the rule for being numerical, continuous data: `Square Footage`, `NumBathrooms`, `NumBedrooms`, `BackyardSpace`, `CrimeRate`, `SchoolRating`, `AgeOfHome`, `DistanceToCityCenter`, `EmploymentRate`, `PropertyTaxRate`, `RenovationQuality`, `LocalAmenities`, and `TransportAccess`. Any data types that were qualitative, discrete, or binary were dropped from the analysis as PCA performs best with numeric, non-categorical data.

```
# Identify the continuous variables needed to answer pricing predictions
X = df[['SquareFootage', 'NumBathrooms', 'NumBedrooms',
        'BackyardSpace', 'CrimeRate', 'SchoolRating', 'AgeOfHome',
        'DistanceToCityCenter', 'EmploymentRate', 'PropertyTaxRate',
        'RenovationQuality', 'LocalAmenities', 'TransportAccess',
        'PreviousSalePrice']]
```

D2. Standardized Data

I used the below code to standardize the data and save it to a CSV. We standardize the data in order to convert all of the data to the same unit, as the continuous data was originally measured in various units.

See the attached CSV file: "D600 Task 3 Dataset 1 Housing Information - Standardized.csv"

```
# Standardize the data to adjust for varying units
scaler = StandardScaler()

# Fit and transform the data to standardize it
X_scaled = scaler.fit_transform(X)

# Place the data in a DataFrame
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
X_scaled_df.head().T
```

	0	1	2	3	4
SquareFootage	-1.132277	0.993926	-1.171293	-0.251508	-1.171293
NumBathrooms	-1.187828	-1.187828	-0.369602	-0.100938	-1.119960
NumBedrooms	0.970213	-0.986989	-0.008388	-0.986989	-0.008388
BackyardSpace	0.957151	0.516683	0.868372	-0.256932	-0.566178
CrimeRate	-0.591776	-0.865854	-1.040621	-0.499677	-1.273088
SchoolRating	-0.700696	-0.695399	1.195478	0.072604	-0.536502
AgeOfHome	-0.230889	-0.197846	0.049814	1.506508	-0.943973
DistanceToCityCenter	-0.615042	-0.797175	0.521008	-1.019229	2.133599
EmploymentRate	0.794367	-0.109066	0.641205	-0.501960	-1.600730
PropertyTaxRate	0.681094	-1.104064	0.741268	-0.101167	-0.482268
RenovationQuality	-0.037232	-0.468641	-0.377284	-0.280851	-0.834070
LocalAmenities	-0.562349	-0.140939	0.803473	-0.351644	-0.178565
TransportAccess	-0.733870	0.433066	1.277560	0.146450	0.514957
PreviousSalePrice	-0.552700	-1.262469	-1.271312	-1.076429	-1.408430

```
# Save the standardized dataset to a new CSV file
X_scaled_df.to_csv('D600 Task 3 Dataset 1 Housing Information - Standardized.csv', index=False)
```

D3. Descriptive Statistics

I used the code below to gather descriptive statistics about the `Price` data, the independent variables prior to standardized scaling, and the independent variables with standardized scaling.

```
# Descriptive statistics of dependent variable
y = df['Price']
y.describe().T
```

```
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
```

```
# Descriptive statistics of independent variables
X.describe().T
```

	count	mean	std	min	25%	50%	75%	max
SquareFootage	7000.0	1048.947459	426.010482	550.000000	660.815000	996.320000	1342.292500	2.874700e+03
NumBathrooms	7000.0	2.131397	0.952561	1.000000	1.290539	1.997774	2.763997	5.807239e+00
NumBedrooms	7000.0	3.008571	1.021940	1.000000	2.000000	3.000000	4.000000	7.000000e+00
BackyardSpace	7000.0	511.507029	279.926549	0.390000	300.995000	495.965000	704.012500	1.631360e+03
CrimeRate	7000.0	31.226194	18.025327	0.030000	17.390000	30.385000	43.670000	9.973000e+01
SchoolRating	7000.0	6.942923	1.888148	0.220000	5.650000	7.010000	8.360000	1.000000e+01
AgeOfHome	7000.0	46.797046	31.779701	0.010000	20.755000	42.620000	67.232500	1.786800e+02
DistanceToCityCenter	7000.0	17.475337	12.024985	0.000000	7.827500	15.625000	25.222500	6.520000e+01
EmploymentRate	7000.0	93.711349	4.505359	72.050000	90.620000	94.010000	97.410000	9.990000e+01
PropertyTaxRate	7000.0	1.500437	0.498591	0.010000	1.160000	1.490000	1.840000	3.360000e+00
RenovationQuality	7000.0	5.003357	1.970428	0.010000	3.660000	5.020000	6.350000	1.000000e+01
LocalAmenities	7000.0	5.934579	2.657930	0.000000	4.000000	6.040000	8.050000	1.000000e+01
TransportAccess	7000.0	5.983860	1.953974	0.010000	4.680000	6.000000	7.350000	1.000000e+01
PreviousSalePrice	7000.0	284509.353128	185734.016770	-8356.902464	142013.977225	262183.133450	396121.169350	1.296607e+06

```
# Descriptive statistics of scaled independent variables
X_scaled_df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
SquareFootage	7000.0	0.000000e+00	1.000071	-1.171293	-0.911152	-0.123544	0.688636	4.286005
NumBathrooms	7000.0	1.299278e-16	1.000071	-1.187828	-0.882798	-0.140288	0.664152	3.859180
NumBedrooms	7000.0	1.624098e-16	1.000071	-1.965590	-0.986989	-0.008388	0.970213	3.906016
BackyardSpace	7000.0	-1.624098e-17	1.000071	-1.826027	-0.752080	-0.055526	0.687749	4.000810
CrimeRate	7000.0	2.030122e-16	1.000071	-1.730810	-0.767652	-0.046671	0.690400	3.800691
SchoolRating	7000.0	-6.171571e-16	1.000071	-3.560846	-0.684806	0.035528	0.750565	1.619204
AgeOfHome	7000.0	-4.060244e-17	1.000071	-1.472336	-0.819514	-0.131447	0.643081	4.150208
DistanceToCityCenter	7000.0	3.248195e-16	1.000071	-1.453356	-0.802373	-0.153885	0.644302	3.969075
EmploymentRate	7000.0	2.931496e-15	1.000071	-4.808251	-0.686198	0.066293	0.821004	1.373718
PropertyTaxRate	7000.0	-3.491810e-16	1.000071	-2.989512	-0.682847	-0.020935	0.681094	3.729903
RenovationQuality	7000.0	-2.273737e-16	1.000071	-2.534329	-0.681808	0.008447	0.683475	2.535997
LocalAmenities	7000.0	-2.760966e-16	1.000071	-2.232942	-0.727904	0.039666	0.795947	1.529653
TransportAccess	7000.0	1.624098e-17	1.000071	-3.057506	-0.667334	0.008261	0.699210	2.055517
PreviousSalePrice	7000.0	-2.598556e-16	1.000071	-1.576917	-0.767256	-0.120214	0.600966	5.449565

E. Perform PCA

E1. Matrix Determination

I used the code below to grab the standardized data and run the principal component analysis, resulting in the principal component matrix.

```
# Be sure to work with the standardized data
df = pd.read_csv('D600 Task 3 Dataset 1 Housing Information -
Standardized.csv')

# List of independent variables
independent_vars = [
    'SquareFootage', 'NumBathrooms', 'NumBedrooms', 'BackyardSpace',
    'CrimeRate', 'SchoolRating', 'AgeOfHome', 'DistanceToCityCenter',
    'EmploymentRate', 'PropertyTaxRate', 'RenovationQuality',
    'LocalAmenities', 'TransportAccess', 'PreviousSalePrice'
]
```

```
# Initialize PCA with the number of components set to the number of
# variables
pca = PCA(n_components = len(independent_vars))

# Fit PCA to the standardized data and transform it
```

```
pca_components = pca.fit_transform(df[independent_vars])

# Create a DataFrame for the principal components
pca_df = pd.DataFrame(data=pca_components,
                      columns=[f'PC{i+1}' for i in
                               range(len(independent_vars))])
```

```
# Output the matrix of principal components
pca_df.head(10)
```

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14
0	-0.899945	-0.321092	-1.554966	0.110730	-0.126251	1.589749	0.711681	-0.246433	-0.482176	-0.615057	-0.497830	-0.166842	0.855549	0.087778
1	-0.805220	0.664521	-0.877927	-1.689104	0.397888	-0.445135	-0.769102	-0.337006	-1.493831	0.257016	0.051287	0.448326	0.150981	-0.707251
2	-0.295618	1.875894	-1.665436	0.817241	0.487000	0.905630	0.049423	-0.321857	0.615564	-0.000373	0.813414	0.345321	-0.801561	-0.178838
3	-1.054050	0.084872	-0.199952	-0.177118	0.222900	-0.325076	-1.557940	-0.347143	-0.280309	-1.234242	0.789672	0.365198	-0.130226	-0.231017
4	-2.173687	1.055700	-0.579576	-0.130284	-1.017059	0.004817	-0.081007	-1.448811	0.955163	2.296010	0.369945	0.469296	0.343436	-0.280579
5	-1.235976	-2.412471	-0.490006	-0.667366	1.535381	0.045972	-0.366593	0.443276	-0.887939	0.491971	0.128089	-0.181225	1.000261	-0.465026
6	-1.447139	-0.318394	1.486552	-1.531248	-0.284258	1.503192	-0.523143	-1.084751	-0.893111	-0.027733	1.011282	0.086020	1.381223	-0.519727
7	-1.733073	0.584193	-0.120449	-1.477425	0.789963	-1.391373	0.528003	-0.092555	-0.608396	-0.962142	0.668784	0.103039	1.391509	1.014189
8	-2.009489	1.091419	-1.490719	2.958190	0.866654	0.985568	-0.269875	-0.969163	0.164839	0.362314	-0.862254	-0.177022	0.174861	-0.379520
9	-0.564468	2.710514	0.141519	1.114606	-0.041316	-0.688885	-0.668448	0.730248	-0.335990	1.015043	0.288209	0.316710	1.376069	0.191353

E2. Total Principal Components

Choosing the right number of principal components to work with is a delicate balance of reducing dimensionality while retaining meaningful variance in the selected components. Two of the most commonly used criteria for selecting the number of principal components to retain are the elbow rule and the Kaiser rule.

First, the PCA is initialized with the total number of independent variables. This dataset features 14 components that qualify based on data type. This allows the model to look at all potential predictive factors.

The PCA is then fitted to the standardized model, obtaining eigenvalues (or explained variance) for each component, and explained variance ratios, which add up to 100% of the explained variance.

A scree plot is a useful visualization for eigenvalues, where a horizontal line can be placed at the point denoting an eigenvalue of 1, which is the relevant criteria for the Kaiser rule. The Kaiser rule recommends retaining only the components with eigenvalues greater than 1, suggesting that the individual component explains more variance than an individual variable.

The elbow rule uses a visual examination of a scree plot to determine where the percentage of the variance explained decreases at a significant rate, creating an "elbow" shape to indicate where the optimal cutoff point is. This is generally where the "curve" or line starts to flatten. Because the elbow rule uses a much more subjective approach (and only retains 2-3 principal

components in the scree plot below as compared to the Kaiser rule's retention of 5 principal components), the Kaiser rule fares better in this analysis.

Below is the code used to determine the explained variance, the explained variance ratios, the cumulative sum of the explained variance, and scree plots for both the cumulative explained variance and the Kaiser rule as applied to eigenvalues. Here we retain about 56% of the variance explained by the data, condensed down to 5 principal components as opposed to 14 variables.

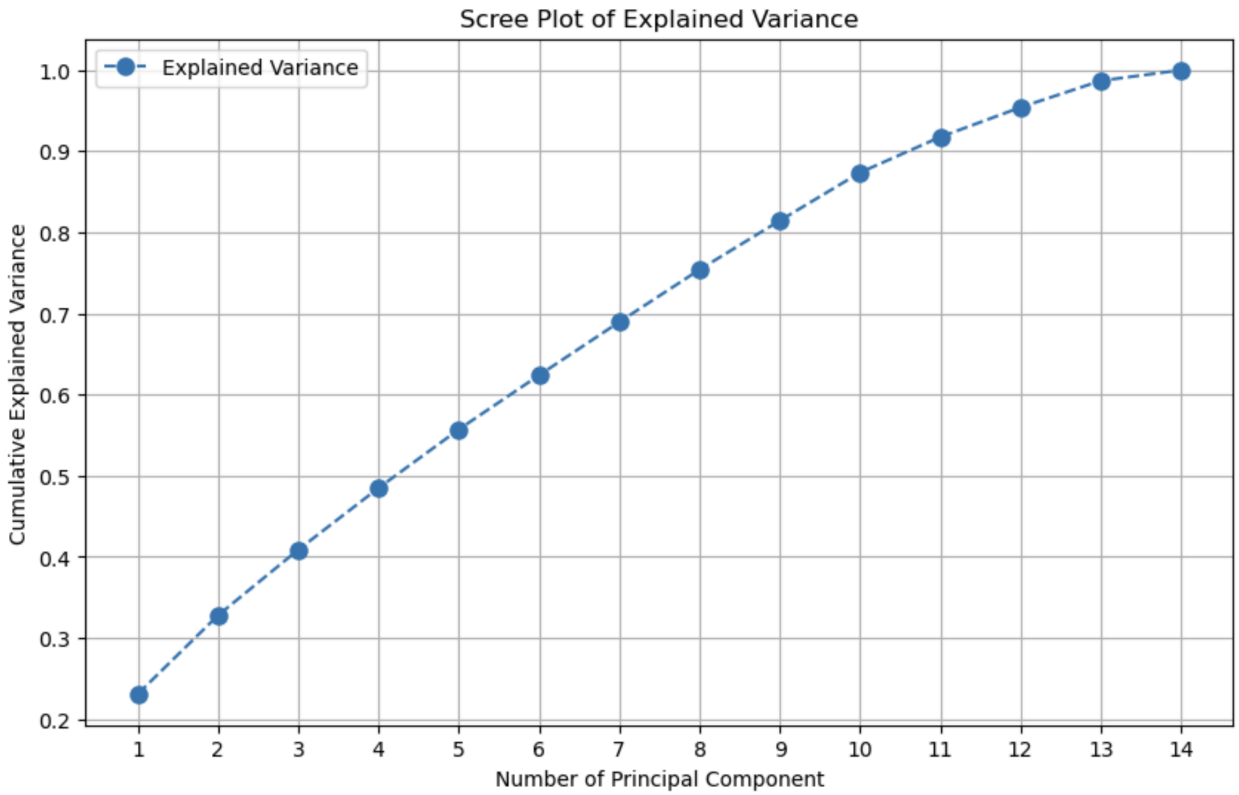
```
# Identify the variance of each of the principal components
explained_variance = pca.explained_variance_ratio_

# Print each principal component's explained variance
for i, var in enumerate(explained_variance):
    print(f"Principal Component {i+1}: {var:.4f}")
```

```
Principal Component 1: 0.2310
Principal Component 2: 0.0969
Principal Component 3: 0.0809
Principal Component 4: 0.0761
Principal Component 5: 0.0722
Principal Component 6: 0.0668
Principal Component 7: 0.0656
Principal Component 8: 0.0645
Principal Component 9: 0.0604
Principal Component 10: 0.0592
Principal Component 11: 0.0438
Principal Component 12: 0.0366
Principal Component 13: 0.0328
Principal Component 14: 0.0131
```

```
# Scree plot for visualizing variance
plt.figure(figsize=(10, 6))
plt.plot(np.arange(1, len(explained_variance) + 1),
explained_variance.cumsum(),
         marker='o', markersize=8, label="Explained Variance",
linestyle='--')
plt.title('Scree Plot of Explained Variance')
plt.xlabel('Number of Principal Component')
plt.ylabel('Cumulative Explained Variance')
```

```
plt.xticks(range(1, len(explained_variance) + 1))
plt.legend()
plt.grid(True)
plt.show()
```

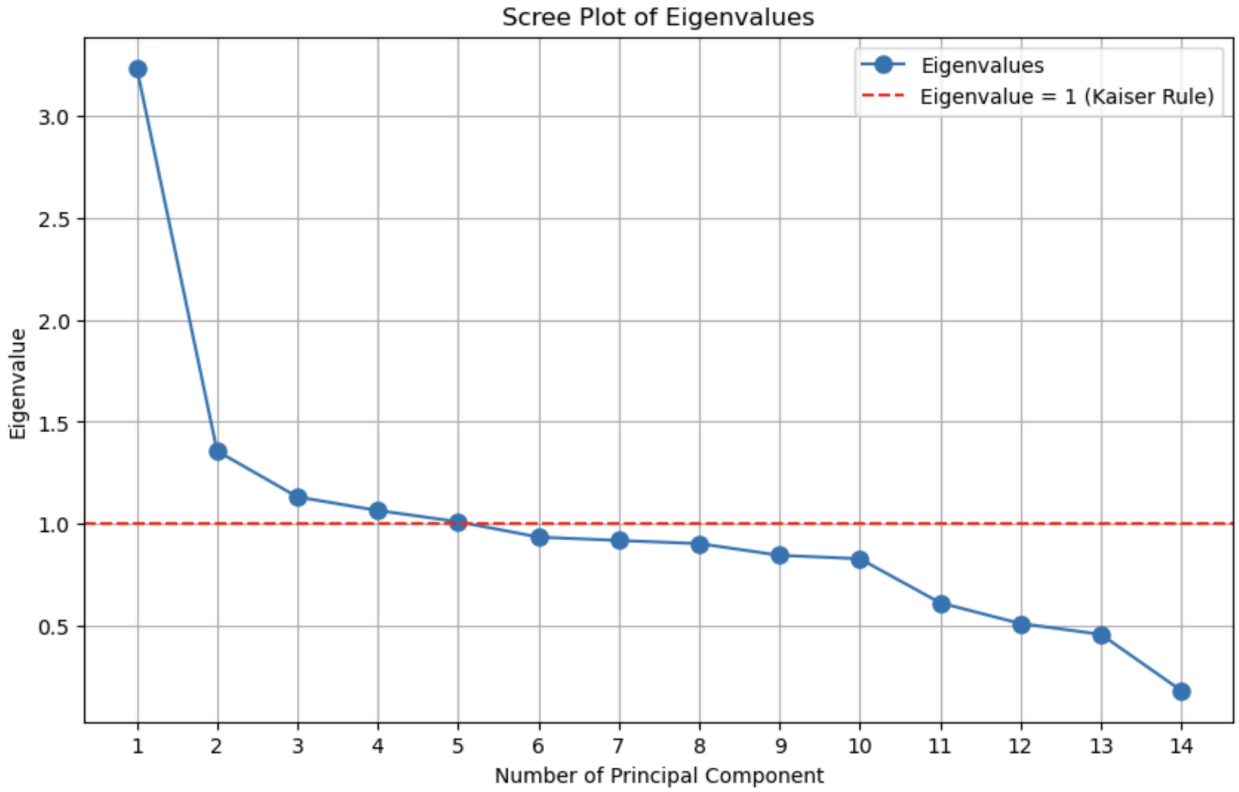



```
# Identify Eigenvalues
eigenvalues = pca.explained_variance_
print(f"Eigenvalues:\n {eigenvalues}")
```

```
Eigenvalues:
 [3.23384649 1.35700378 1.13225984 1.06585231 1.01099575 0.93557991
 0.91907318 0.90364539 0.84622548 0.82939552 0.6134753  0.51193268
 0.45915282 0.18356185]
```

```
# Scree plot using the Kaiser Rule
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(eigenvalues) + 1), eigenvalues, marker='o',
```

```
         markersize=8, label="Eigenvalues")
plt.axhline(y=1, color='r', linestyle='--',
            label='Eigenvalue = 1 (Kaiser Rule)')
plt.xticks(range(1, len(independent_vars) + 1))
plt.title('Scree Plot of Eigenvalues')
plt.xlabel('Number of Principal Component')
plt.ylabel('Eigenvalue')
plt.legend()
plt.grid(True)
plt.show()
```




```
# Find the Principal Components that should be retained based on the Kaiser Rule
pca = PCA(n_components = 5)
X_pca = pca.fit_transform(X_scaled)
X_pca_df = pd.DataFrame(X_pca, columns=[f'PC{i+1}' for i in range(5)])
X_pca_df
```

	PC1	PC2	PC3	PC4	PC5
0	-0.899945	-0.321092	-1.554966	0.110730	-0.126251
1	-0.805220	0.664521	-0.877927	-1.689104	0.397888
2	-0.295618	1.875894	-1.665436	0.817241	0.487000
3	-1.054050	0.084872	-0.199952	-0.177118	0.222900
4	-2.173687	1.055700	-0.579576	-0.130284	-1.017059
...	...	...	...	...	...
6995	0.305374	-0.022458	-1.816575	-0.109783	0.282700
6996	1.669242	1.632416	2.030405	-0.190591	2.735052
6997	1.225342	-0.972373	-1.380511	0.674188	-2.063607
6998	2.434477	0.008076	-1.293639	-0.168930	-0.519733
6999	3.444527	-1.071282	-0.327111	0.093358	1.112313

7000 rows × 5 columns

E3. Variance

Below is the code used to include the variance for each of the 5 principal components that were retained for further analysis.

```
# Extract the variance and explained variance ratio for each principal component
explained_variance = pca.explained_variance_
explained_variance_ratio = pca.explained_variance_ratio_

# Display the variance and the explained variance ratio for each component
print("Explained Variance of Each Principal Component:")
for i, variance in enumerate(explained_variance, start=1):
    print(f"Principal Component {i}: Variance = {variance:.4f}")

# Print proportions
print("\nProportion of Variance Explained by Each Principal Component:")
for i, variance_ratio in enumerate(explained_variance_ratio, start=1):
    print(f"Principal Component {i}: Explained Variance Ratio =
```

```
{variance_ratio:.4%}")
```

```
Explained Variance of Each Principal Component:
Principal Component 1: Variance = 3.2338
Principal Component 2: Variance = 1.3570
Principal Component 3: Variance = 1.1323
Principal Component 4: Variance = 1.0659
Principal Component 5: Variance = 1.0110

Proportion of Variance Explained by Each Principal Component:
Principal Component 1: Explained Variance Ratio = 23.0956%
Principal Component 2: Explained Variance Ratio = 9.6915%
Principal Component 3: Explained Variance Ratio = 8.0864%
Principal Component 4: Explained Variance Ratio = 7.6121%
Principal Component 5: Explained Variance Ratio = 7.2204%
```

```
# Print the cumulative explained variance for selecting components
# via elbow rule
cumulative_explained_variance = pca.explained_variance_ratio_.cumsum()
print("Cumulative explained variance:", cumulative_explained_variance)
```

```
Cumulative explained variance: [0.23095604 0.32787103 0.40873518 0.48485661 0.55706028]
```

E4. PCA Summary

Section E1. Matrix Determination shows the Principal Component Analysis (PCA) started with the input of 14 independent variables from the original dataset, given that no data was missing from any column. This suggests that all 14 principal components provide 100% of the variance, but retaining all 14 fails to simplify the data or enhance model performance. To keep all 14 principal components would also likely result in overfitting the model, resulting in inaccurate predictions.

I used the Kaiser rule (keeping any components with an eigenvalue of 1.00 or greater) to retain the first 5 principal components as they explain more variance than any individual variable. With the Kaiser rule, we maintain approximately 56% of the explained variance. Given the dimensionality reduction, working with 5 principal components is significantly easier to work with

than 14 variables and encapsulates the most crucial information while discarding less important details.

We can use the data remaining to run a multiple linear regression analysis to verify how accurately the model can predict pricing given the new metrics (Ahmad, 2023).

F. Data Analysis (Linear Regression)

F1. Splitting the Data

I used the below code to split the PCA data using a 70/30 split, with 70% going to the training dataset and 30% being designated for the testing dataset. The code also includes saving the datasets as separate CSV files.

The following CSV files are attached as a result:

"D600 Task 3 Dataset 1 Housing Information - Train.csv"

"D600 Task 3 Dataset 1 Housing Information - Test.csv"

```
# Create the dataset from the principal componenents remaining
X_pca = pd.DataFrame(X_pca, columns=[f'PC{i+1}' for i in
range(X_pca.shape[1])])

# Add the dependent variable 'Price' back to the PCA DataFrame
housing_df = pd.concat([X_pca, y], axis=1)
housing_df
```

	PC1	PC2	PC3	PC4	PC5	Price
0	-0.899945	-0.321092	-1.554966	0.110730	-0.126251	255614.8992
1	-0.805220	0.664521	-0.877927	-1.689104	0.397888	155586.0947
2	-0.295618	1.875894	-1.665436	0.817241	0.487000	131050.8324
3	-1.054050	0.084872	-0.199952	-0.177118	0.222900	151361.7125
4	-2.173687	1.055700	-0.579576	-0.130284	-1.017059	113167.6128
...	...	...	...	...	...	...
6995	0.305374	-0.022458	-1.816575	-0.109783	0.282700	307821.1758
6996	1.669242	1.632416	2.030405	-0.190591	2.735052	421368.8869
6997	1.225342	-0.972373	-1.380511	0.674188	-2.063607	473382.5348
6998	2.434477	0.008076	-1.293639	-0.168930	-0.519733	343397.9756
6999	3.444527	-1.071282	-0.327111	0.093358	1.112313	438060.8193

7000 rows × 6 columns

```
# Split the data into a train_df and test_df to start
train_df, test_df = train_test_split(housing_df, test_size=0.3,
random_state=42)
```

```
# Dataset shapes
print(f"Training dataset shape: {train_df.shape}")
print(f"Test dataset shape: {test_df.shape}")
```

```
Training dataset shape: (4900, 6)
Test dataset shape: (2100, 6)
```

```
# Export the data to CSV files
train_df.to_csv('D600 Task 3 Dataset 1 Housing Information - Train.csv',
index=False)
test_df.to_csv('D600 Task 3 Dataset 1 Housing Information - Test.csv',
index=False)
```

```
# Establish the dependent and independent variables to create the linear
# regression model
y = housing_df['Price']
```

```
X = housing_df.drop('Price', axis=1)

# Resplit the data for testing and training data given variable splits
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

F2. Model Optimization

The following code was used to fit the initial linear regression model, create a function to run backward elimination as an optimization technique, and refit the model using the optimization. Backward elimination is helpful as an optimization technique as it uses an iterative approach to remove statistically insignificant coefficients, from least significant to most significant, until a threshold has been reached. This retains only the coefficients that retain a statistical significance of the industry threshold of having a p-value of 0.05 or greater. This threshold can be modified based on the industry or needs. In this case, only one of the retained principal components was not deemed statistically significant, retaining four of the original five found via the Kaiser rule.

```
# Add a constant to the training set
X_train = sm.add_constant(X_train)

# Fit the linear regression model
model = sm.OLS(y_train, X_train).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.638
Model:                            OLS   Adj. R-squared:                  0.638
Method:                 Least Squares   F-statistic:                     1726.
Date:                Fri, 21 Feb 2025   Prob (F-statistic):               0.00
Time:                        21:33:18   Log-Likelihood:                -62897.
No. Observations:                4900   AIC:                         1.258e+05
Df Residuals:                    4894   BIC:                         1.258e+05
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       3.082e+05   1299.326    237.181      0.000    3.06e+05    3.11e+05
PC1         6.187e+04    713.672     86.699      0.000    6.05e+04    6.33e+04
PC2        -2.408e+04   1108.649    -21.719      0.000   -2.63e+04   -2.19e+04
PC3         3.021e+04   1228.321     24.599      0.000    2.78e+04    3.26e+04
PC4         4869.4430   1262.708      3.856      0.000    2393.968    7344.917
PC5        -1452.3826   1291.699     -1.124      0.261   -3984.693    1079.927
==============================================================================
Omnibus:                      286.943   Durbin-Watson:                   1.977
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              362.960
Skew:                           0.567   Prob(JB):                     1.53e-79
Kurtosis:                       3.703   Cond. No.                         1.83
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```python
# Chosen Method: Backward Elimination
def backward_elimination(X, y, significance_level=0.05):
    X = sm.add_constant(X)
    model = sm.OLS(y, X).fit()

    while True:
        max_p_value = model.pvalues.max()
        if max_p_value >= significance_level:
            excluded_feature = model.pvalues.idxmax()
            X = X.drop(columns=[excluded_feature])
            model = sm.OLS(y, X).fit()
        else:
            break
    return model
```

```
# Perform backward elimination
optimized_model = backward_elimination(X_train, y_train)

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

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.638
Model:                            OLS   Adj. R-squared:                  0.638
Method:                 Least Squares   F-statistic:                     2157.
Date:                Fri, 21 Feb 2025   Prob (F-statistic):               0.00
Time:                        21:33:18   Log-Likelihood:                -62898.
No. Observations:                4900   AIC:                         1.258e+05
Df Residuals:                    4895   BIC:                         1.258e+05
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       3.082e+05   1299.265    237.179      0.000    3.06e+05    3.11e+05
PC1         6.187e+04    713.684     86.693      0.000    6.05e+04    6.33e+04
PC2        -2.409e+04   1108.655    -21.726      0.000   -2.63e+04   -2.19e+04
PC3         3.021e+04   1228.354     24.597      0.000    2.78e+04    3.26e+04
PC4         4878.2607   1262.718      3.863      0.000    2402.767    7353.754
==============================================================================
Omnibus:                      285.634   Durbin-Watson:                   1.976
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              360.910
Skew:                           0.565   Prob(JB):                     4.26e-79
Kurtosis:                       3.700   Cond. No.                         1.82
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

F3. Mean Squared Error (MSE)

I used the `mean_squared_error` method from the Scikit-Learn package to calculate the mean squared error (MSE). MSE measures how well a model's predictions match the actual data and acts as a baseline metric to compare different models or the same model with different parameters. A lower MSE indicates a better-fitting model regarding its ability to predict unseen data with accuracy.

The MSE on the training dataset was 8,260,974,658.34, meaning the price predictions were within an average of $90,889.90 per prediction.

```
# Calculate Mean Squared Error (MSE) on the training set
# Use the features selected by the optimized model for prediction
y_train_pred = optimized_model.predict(optimized_model.model.exog)

mse_train = mean_squared_error(y_train, y_train_pred)
print(f"Mean Squared Error (MSE) on Training Set: {mse_train:.2f}")
```

```
Mean Squared Error (MSE) on Training Set: 8260974658.34
```

F4. Model Accuracy

I used the following code to run the prediction on the test dataset using the optimized regression model to give the accuracy of the prediction model based on the MSE.

The output of the MSE on the test dataset was 8,116,119,380.71. With regards to the accuracy of the model, the MSE predicts the accuracy of the test dataset within an average of $90,089.51, an improvement of approximately $800 per prediction.

```
# Calculate Mean Squared Error (MSE) on the test set
# Use the features selected by the optimized model for prediction

# Add a constant to the test set (similar to what was done for the training
set)
X_test = sm.add_constant(X_test)

# Use X_test instead of optimized_model.model.exog for prediction
y_test_pred =
optimized_model.predict(X_test[optimized_model.model.exog_names])

# Calculate the Mean Squared Error (MSE) on the test dataset
mse_test = mean_squared_error(y_test, y_test_pred)

# Print results for comparison
print(f"Mean Squared Error (MSE) on Training Set: {mse_train:.2f}")
print(f"Mean Squared Error (MSE) on Test Set: {mse_test:.2f}")
```

```
Mean Squared Error (MSE) on Training Set: 8260974658.34
Mean Squared Error (MSE) on Test Set: 8116119380.71
```

G. Data Analysis Summary

G1. Packages or Libraries List

- The Pandas package is useful for storing data in DataFrames and making the data tabular and easy to work with.
- The NumPy package is useful for statistical analysis on data.
- The Matplotlib package is used as a base package for data visualization and for explicitly stating sizes of figures and attributes such as plot labels.
- The Scikit-Learn package is used for splitting the data into training and test sets, scaling the data to address differing units using `StandardScaler`, performing Principal Component Analysis using `PCA`, and calculating mean squared error (MSE).
- The Statsmodels package is used to perform linear regression and calculate the variance inflation factor to check for multicollinearity.

G2. Method Justification

As mentioned in F2. Model Optimization, I chose backward elimination as my optimization method. Backward elimination uses an iterative approach to remove the least statistically significant coefficients using the coefficient's p-value, until all remaining coefficients have a p-value of 0.05 or less. This indicates that each of the remaining coefficients have statistical significance for predictive value. On each iteration, the function refits the model based on the remaining coefficients and helps to simplify the model further.

Based on the mean squared errors used on the optimized training and test sets, the accuracy after using backward elimination improved the model by an average of $800 per house. As will be shown in G3. Verification of Assumptions, there is no multicollinearity to deteriorate the performative power of predicting house prices and at 7,000 independent observations, there is enough data to satisfy size requirements.

G3. Verification of Assumptions

I used statsmodels' `variance_inflation_factor` method to verify that there is no multicollinearity between the remaining coefficients. As all the values are right around 1, we can be assured there is no multicollinearity.

```
# Calculate the Variance Inflation Factor (VIF) of the remaining
coefficients
vif = pd.DataFrame()
vif["Features"] = optimized_model.model.exog_names
vif["VIF"] = [variance_inflation_factor(optimized_model.model.exog, i) for
i in range(optimized_model.model.exog.shape[1])]
vif
```

	Features	VIF
0	const	1.000269
1	PC1	1.000016
2	PC2	1.000154
3	PC3	1.000166
4	PC4	1.000033

G4. Equation

The multiple linear regression equation to include 4 predictor variables is:

$$Y = \beta_0 + \beta_1 x_1 + \beta_{2x} 2 + \beta_3 x_3 + \beta_4 x_4 + \epsilon \, Y = \beta_0 + \beta_1 x_1 + \beta_{2x} 2 + \beta_3 x_3 + \beta_4 x_4 + \epsilon$$

- Y is the price / dependent variable.
- β_0 is the intercept when the independent variable is 0: 308,200
- $\beta_1 x_1$ is the coefficient for PC1: 61,870
- $\beta_2 x_2$ is the coefficient for PC2: -24,090
- $\beta_3 x_3$ is the coefficient for PC3: 30,210
- $\beta_4 x_4$ is the coefficient for PC4: 4,878.26
- ϵ is the average value of error on the model: $90,089.51

The final equation with the coefficients is as follows:

$Y = 308,200 + 81,870(x_1) - 24,090(x_2) + 30,210(x_3) + 4,878.26(x_4) + 90,089.51$, where x_1 is PC1, x_2 is PC2, x_3 is PC3, and x_4 is PC4.

G5. Model Metrics

R-squared is a measurement of the proportion of variance in a dependent variable that can be explained by the independent variable in a regression model.

Adjusted R-squared measures the proportion of variance in a regression model, taking into account the number of independent variables acting as predictors.

The R-squared value in this model is 0.638, meaning that nearly 64% of the variance in housing value can be explained through the predictor variables that were chosen for this model. This is a moderately strong score, so while they can be used with relative confidence, there is 36% of the value that would be explained by other variables.

The adjusted R-squared value matches the R-squared value because a small number of predictor variables were used in the regression model.

The Mean Squared Error (MSE) differed between the training and test set. In the training set, the model was off by an average of $90,889.90. In the test set, the model was off by an average of $90,089.51, improving by $800.39. The model is still off by quite a bit, but this can be expected as only a few variables were chosen as predictor variables and account for approximately 64% of the factors necessary to predict a value with total confidence.

```
# Use the features selected by the optimized model for prediction
y_train_pred = optimized_model.predict(optimized_model.model.exog)

# Calculate the Mean Squared Error (MSE) on the training dataset
mse_train = mean_squared_error(y_train, y_train_pred)

# Add a constant to the test set (similar to what was done for the training
set)
X_test = sm.add_constant(X_test)

# Use X_test instead of optimized_model.model.exog for prediction
y_test_pred =
optimized_model.predict(X_test[optimized_model.model.exog_names])

# Calculate the Mean Squared Error (MSE) on the test dataset
mse_test = mean_squared_error(y_test, y_test_pred)

# Print results for comparison
```

```
print(f"Mean Squared Error (MSE) on Training Set: {mse_train:.2f}")
print(f"Mean Squared Error (MSE) on Test Set: {mse_test:.2f}")
```

```
Mean Squared Error (MSE) on Training Set: 8260974658.34
Mean Squared Error (MSE) on Test Set: 8116119380.71
```

G6. Results & Implications

Principal Component Analysis (PCA) proved to be an effective model for dimensionality reduction in this study. In E4. PCA Summary, I determined that 5 principal components retained approximately 56% of the original data's variance, suggesting that these components effectively distill the most informative aspects of the 14 continuous variables initially considered.

The combined effect of PCA and multiple linear regression allowed for the prediction of house prices with a moderate degree of accuracy. The optimized linear regression model further reduced the dimensionality to the 4 most statistically significant principal components as identified by a combination of PCA and backward elimination. The model achieved a mean squared error (MSE) that indicates predictions of prices are within, on average, approximately $90,000 of actual house prices. This level of variance indicates a moderate model fit, capturing nearly 64% of the variance in house prices as measured by the R-squared value.

The implications of these results are twofold:

Stakeholders: The model offers an insightful starting point for stakeholders including potential home buyers, real estate agents, and developers. This simplified model can be used to gauge potential pricing based on key predictors as they are condensed into the principal components, encompassing Square Footage, Crime Rate, School Rating, and more, as they are captured within the components themselves.

Future Model Tuning: While the current model offers a practical prediction framework, the moderate accuracy indicates room for improvement. Integrating additional variables, especially non-continuous variables, categorical data, and newly acquired data, could enhance the model's robustness.

G7. Course of Action

The original exploratory question posed in B1 asked how various continuous features contribute to predicting the price of a house, aiming to use Principal Component Analysis (PCA) and multiple linear regression to reduce dimensionality while preserving the factors that have the most statistical significance to impact housing prices. The intended goal is to assist potential home buyers in estimating a home's cost based on desirable characteristics.

The analysis demonstrated that by leverage PCA, dimensionality reduction was possible, reducing the complexity of the model from 14 continuous variables to 5 principal components (4 after the use of multiple linear regression) to retain 56% of the dataset's variance. This allows for the prediction of house prices with a mean square error suggesting up to a $90,000 deviation on average from actual prices.

Recommend Course of Action:

Utilize predictive insights in decision making: Stakeholders such as potential home buyers, real estate agents, and developers can employ this model to assess potential properties. The principal components offer a focused view of critical aspects influencing house prices, which may inform investment and development strategies.

Enhance predictive capabilities with additional variables: To enhance the model's predictive accuracy and reduce the margin of error, additional variables could be incorporated at the linear regression stage that cannot be assessed by PCA. This includes binary, categorical, and qualitative insights.

Ongoing model enhancement: As new data becomes available, update the model to include current and ongoing metrics that could impact house prices. Context is important (economic changes, changes to local policies, etc.) and will help prevent the model from overfitting.

Engage stakeholders and domain experts for model feedback: Engage with stakeholders to gather qualitative feedback and identify other critical local factors influencing house prices.

H. Panopto Recording

I. Sources

Ahmad, I. (2023). *50 Algorithms Every Programmer Should Know* (2nd ed.). Packt Publishing.

Safjan, K. (2023, February 20). *Checks and Data Preprocessing Steps Before Applying PCA*. Krystian Safjan's Blog. Retrieved February 17, 2025, from https://safjan.com/before-pca/#:~:text=Before%20applying%20PCA%20to%20a%20dataset%20for%20dimensionality%20reduction%2C%20we,most%20out%20of%20the%20data.

Shmueli, G., Bruce, P. C., Gedeck, P., & Patel, N. R. (2020). Data Mining for Business Analytics: Concepts, Techniques and Applications in Python. Wiley.

Due to the similarities between Task 3 and Tasks 1 and 2, some of my own work and explanations have been re-used for this task but all code, choice of variables, and visualizations have been updated.