Statistical Data Mining

Task 3

Nathan Hefner

B. Describe the purpose of this data analysis by doing the following:

1. Propose one research question that is relevant to a real-world organizational situation captured in the provided dataset that you will answer using linear regression in the initial model.

Using linear regression with principal component analysis, which variables affect the price of a home the most amount and least amount.

2. Define one goal of the data analysis. Ensure that your goal is reasonable within the scope of the scenario and is represented in the available data.

The goal of our data analysis will be to see if we can accurately predict the price of a home with low error, along with pinpointing the biggest reasons for a home's price change.

C. Explain the reasons for using PCA by doing the following:

1. Explain how PCA can be used to prepare the selected dataset for regression analysis. Include expected outcomes.

Principal component analysis is used in linear regression to take massive amounts of highly correlated data and break it into smaller sets while retaining the variance. This will improve the model's performance while helping the model generalize better and, most importantly, help remove multicollinearity.

2. Summarize one assumption of PCA.

PCA assumes that all variables are numeric and continuous and share a linear relationship. This means the model cannot perform as efficiently or optimally if any variables have a more complex relationship.

D. Summarize the data preparation process for linear regression analysis by doing the following:

1. Identify the continuous dataset variables that you will need to answer the research question proposed in part B1.

The continuous dataset variables we will be using are as follows: 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, and Previous Sale Price

2. Standardize the continuous dataset variables identified in part D1. Include a copy of the cleaned dataset.

Below is the code I used to standardize my dataset.

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

columns = ['SquareFootage','NumBathrooms', 'NumBedrooms', 'BackyardSpace',
           'CrimeRate', 'SchoolRating', 'AgeOfHome', 'DistanceToCityCenter',
           'EmploymentRate', 'PropertyTaxRate', 'RenovationQuality', 'LocalAmenities',
           'TransportAccess', 'PreviousSalePrice']

X = df[columns]

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

pca = PCA(n_components=None)
X_pca_all = pca.fit_transform(X_scaled)

df_pca = pd.DataFrame(X_pca_all, columns=columns)

df_pca.to_csv('pca_transformed_data.csv', index=False)
```

3. Describe the dependent variable and all independent variables from part D1 using descriptive statistics (counts, means, modes, ranges, min/max), including a screenshot of the descriptive statistics output for each of these variables.

Below is my code for finding the descriptive statistics for our independent and dependent variables.

```
variables = df[['Price', 'SquareFootage','NumBathrooms', 'NumBedrooms', 'BackyardSpace',
           'CrimeRate', 'SchoolRating', 'AgeOfHome', 'DistanceToCityCenter',
           'EmploymentRate', 'PropertyTaxRate', 'RenovationQuality', 'LocalAmenities',
           'TransportAccess', 'PreviousSalePrice']]
variable_stats = variables.describe()
print(variable_stats)
```

```
              Price  SquareFootage  NumBathrooms  NumBedrooms  BackyardSpace  \
count  7.000000e+03    7000.000000   7000.000000  7000.000000    7000.000000
mean   3.072820e+05    1048.947459      2.131397     3.008571     511.507029
std    1.501734e+05     426.010482      0.952561     1.021940     279.926549
min    8.500000e+04     550.000000      1.000000     1.000000       0.390000
25%    1.921075e+05     660.815000      1.290539     2.000000     300.995000
50%    2.793230e+05     996.320000      1.997774     3.000000     495.965000
75%    3.918781e+05    1342.292500      2.763997     4.000000     704.012500
max    1.046676e+06    2874.700000      5.807239     7.000000    1631.360000

         CrimeRate  SchoolRating    AgeOfHome  DistanceToCityCenter  \
count  7000.000000   7000.000000  7000.000000           7000.000000
mean     31.226194      6.942923    46.797046             17.475337
std      18.025327      1.888148    31.779701             12.024985
min       0.030000      0.220000     0.010000              0.000000
25%      17.390000      5.650000    20.755000              7.827500
50%      30.385000      7.010000    42.620000             15.625000
75%      43.670000      8.360000    67.232500             25.222500
max      99.730000     10.000000   178.680000             65.200000

       EmploymentRate  PropertyTaxRate  RenovationQuality  LocalAmenities  \
count     7000.000000      7000.000000        7000.000000     7000.000000
mean        93.711349         1.500437           5.003357        5.934579
std          4.505359         0.498591           1.970428        2.657930
min         72.050000         0.010000           0.010000        0.000000
25%         90.620000         1.160000           3.660000        4.000000
50%         94.010000         1.490000           5.020000        6.040000
75%         97.410000         1.840000           6.350000        8.050000
max         99.900000         3.360000          10.000000       10.000000

       TransportAccess  PreviousSalePrice
count      7000.000000       7.000000e+03
mean          5.983860       2.845094e+05
std           1.953974       1.857340e+05
min           0.010000      -8.356902e+03
25%           4.680000       1.420140e+05
50%           6.000000       2.621831e+05
75%           7.350000       3.961212e+05
max          10.000000       1.296607e+06
```

E. Perform PCA by doing the following:

1. Determine the matrix of all the principal components.

Below is my code for the 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 -0.899945 -0.321092 -1.554966  0.110730 -0.126251  1.589749  0.711681
1 -0.805220  0.664521 -0.877927 -1.689104  0.397888 -0.445135 -0.769102
2 -0.295618  1.875894 -1.665436  0.817241  0.487000  0.905630  0.049423
3 -1.054050  0.084872 -0.199952 -0.177118  0.222900 -0.325076 -1.557940
4 -2.173687  1.055700 -0.579576 -0.130284 -1.017059  0.004817 -0.081007

        PC8       PC9      PC10      PC11      PC12      PC13      PC14
0 -0.246433 -0.482176 -0.615057 -0.497830 -0.166842  0.855549  0.087778
1 -0.337006 -1.493831  0.257016  0.051287  0.448326  0.150981 -0.707251
2 -0.321857  0.615564 -0.000373  0.813414  0.345321 -0.801561 -0.178838
3 -0.347143 -0.280309 -1.234242  0.789672  0.365198 -0.130226 -0.231017
4 -1.448811  0.955163  2.296010  0.369945  0.469296  0.343436 -0.280579
```

2. Identify the total number of principal components (that should be retained), using the elbow rule or the Kaiser rule. Include a screenshot of the scree plot.

When using the Kaiser rule, we want to keep principal components with eigenvalues greater than 1. In the scree plot, we look for the elbow rule where the drop in explained variance begins to flatten out. Our results state we should keep the first five components. Below is my code and screenshot.

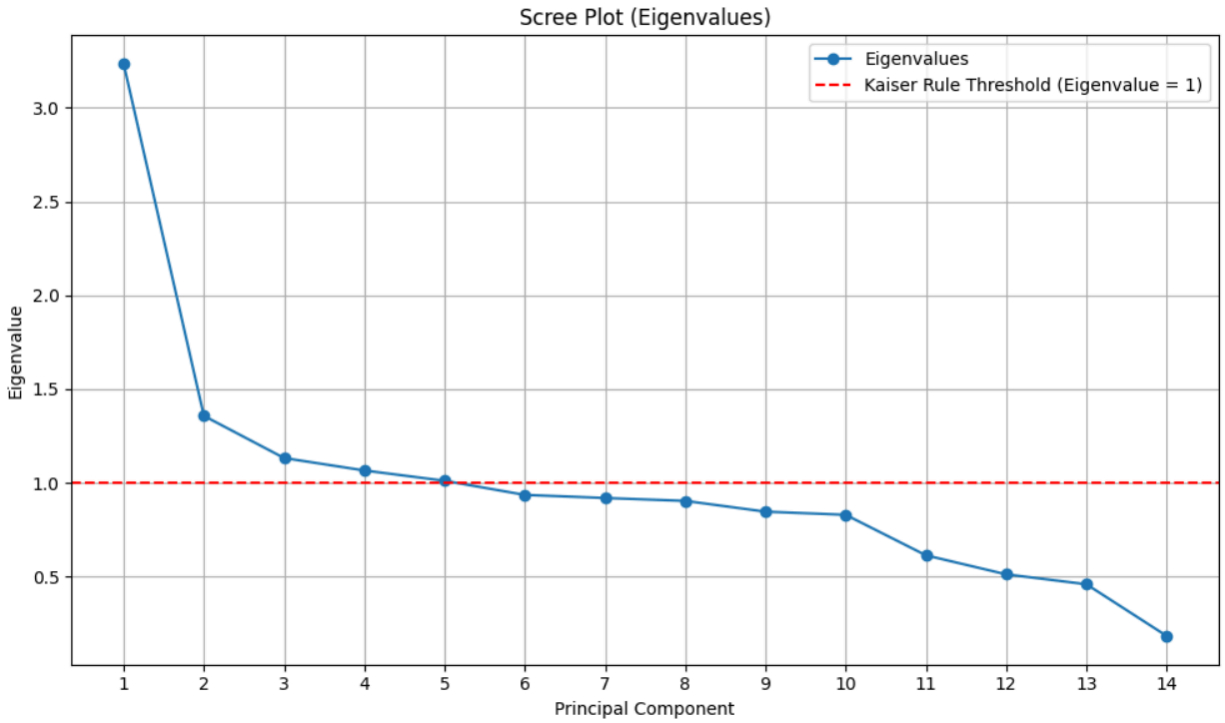
```
import matplotlib.pyplot as plt
import numpy as np

explained_variance = pca.explained_variance_
explained_variance_ratio = pca.explained_variance_ratio_

kaiser_rule_count = np.sum(explained_variance > 1)
print(f"Number of components to retain (Kaiser Rule): {kaiser_rule_count}")

plt.figure(figsize=(10, 6))
plt.plot(range(1, len(explained_variance) + 1), explained_variance, marker='o', linestyle='-', label='Eigenvalues')
plt.axhline(y=1, color='r', linestyle='--', label='Kaiser Rule Threshold (Eigenvalue = 1)')
plt.title('Scree Plot (Eigenvalues)')
plt.xlabel('Principal Component')
plt.ylabel('Eigenvalue')
plt.xticks(range(1, len(explained_variance) + 1))
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Number of components to retain (Kaiser Rule): 5
```



3. Identify the variance of each of the principal components identified in part E2.

Below is my code identifying the variance of each principal component.

```
explained_variance = pca.explained_variance_
explained_variance_ratio = pca.explained_variance_ratio_

variance_df = pd.DataFrame({
    'Principal Component': [f'PC{i+1}' for i in range(len(explained_variance))],
    'Variance (Eigenvalue)': explained_variance,
    'Explained Variance Ratio': explained_variance_ratio
})

print(variance_df)
```

	Principal Component	Variance (Eigenvalue)	Explained Variance Ratio
0	PC1	3.233846	0.230956
1	PC2	1.357004	0.096915
2	PC3	1.132260	0.080864
3	PC4	1.065852	0.076121
4	PC5	1.010996	0.072204
5	PC6	0.935580	0.066818
6	PC7	0.919073	0.065639
7	PC8	0.903645	0.064537
8	PC9	0.846225	0.060436
9	PC10	0.829396	0.059234
10	PC11	0.613475	0.043813
11	PC12	0.511933	0.036561
12	PC13	0.459153	0.032792
13	PC14	0.183562	0.013110

4. Summarize the results of your PCA.

After principal component analysis, only five values will help the model, as they are the most significant because their eigenvalue is greater than 1. With these five components, we will perform linear regression and build a model to predict a house's price accurately.

F. Perform the data analysis and report on the results by doing the following:

1. Split the data into two datasets, with a larger percentage assigned to the training dataset and a smaller percentage assigned to the test dataset.

Below is the code that split my data. We are doing a 70/30 split.

```
import statsmodels.api as sm
pca = PCA(n_components=5)
X_pca = pca.fit_transform(X)
X_pca = pd.DataFrame(X_pca, columns=[f'PC{i+1}' for i in range(X_pca.shape[1])])
final_df = pd.concat([X_pca, df['Price']], axis=1)


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=0.3, random_state=42)

X_train.to_csv('pca_train.csv', index=False)
X_test.to_csv('pca_test.csv', index=False)

print("Data split and saved successfully")
```

```
Data split and saved successfully
```

2. Use the training dataset to create and perform a regression model using regression as a statistical method. Optimize the regression model using a process of your selection.

Below is the code and the model results before and after optimization. For optimization, I chose backward stepwise elimination.

```
X_train_sm = sm.add_constant(X_train)
X_test_sm = sm.add_constant(X_test)

model = sm.OLS(y_train, sm.add_constant(X_train)).fit()

print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.679
Model:                            OLS   Adj. R-squared:                  0.678
Method:                 Least Squares   F-statistic:                     2068.
Date:                Sun, 06 Apr 2025   Prob (F-statistic):               0.00
Time:                        17:45:17   Log-Likelihood:                -62606.
No. Observations:                4900   AIC:                         1.252e+05
Df Residuals:                    4894   BIC:                         1.253e+05
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       3.075e+05   1224.191    251.217      0.000    3.05e+05     3.1e+05
PC1            0.6637      0.007    101.219      0.000       0.651       0.677
PC2           33.6871      3.529      9.545      0.000      26.768      40.606
PC3           -3.8991      4.418     -0.882      0.378     -12.561       4.763
PC4          -97.2965     39.248     -2.479      0.013    -174.240     -20.352
PC5           -8.4491     68.345     -0.124      0.902    -142.435     125.537
==============================================================================
Omnibus:                      326.840   Durbin-Watson:                   2.016
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              934.068
Skew:                           0.349   Prob(JB):                    1.48e-203
Kurtosis:                       5.022   Cond. No.                     1.87e+05
==============================================================================
```

```
X_be = X_train.copy()
y_be = y_train.copy()

X_be = sm.add_constant(X_be)

def backward_elimination(X, y, significance_level=0.05):
    iteration = 1
    while True:
        model = sm.OLS(y, X).fit()
        p_values = model.pvalues
        max_p_value = p_values.max()
        if max_p_value > significance_level:
            excluded_feature = p_values.idxmax()
            print(f"Iteration {iteration}: Dropping '{excluded_feature}' (p = {max_p_value:.4f})")
            X = X.drop(columns=excluded_feature)
            iteration += 1
        else:
            break
    return model, X

optimized_model, X_train_optimized = backward_elimination(X_be, y_be)

print("Final Model Summary")
print(optimized_model.summary())
```

```
Iteration 1: Dropping 'PC5' (p = 0.9016)
Iteration 2: Dropping 'PC3' (p = 0.3774)
```

```
Final Model Summary
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.679
Model:                            OLS   Adj. R-squared:                  0.678
Method:                 Least Squares   F-statistic:                     3447.
Date:                Sun, 06 Apr 2025   Prob (F-statistic):               0.00
Time:                        18:44:41   Log-Likelihood:                -62606.
No. Observations:                4900   AIC:                         1.252e+05
Df Residuals:                    4896   BIC:                         1.252e+05
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       3.075e+05   1223.983    251.251      0.000    3.05e+05     3.1e+05
PC1            0.6637      0.007    101.228      0.000       0.651       0.677
PC2           33.6408      3.528      9.534      0.000      26.724      40.558
PC4          -96.9494     39.239     -2.471      0.014    -173.876     -20.023
==============================================================================
Omnibus:                      327.030   Durbin-Watson:                   2.015
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              936.447
Skew:                           0.348   Prob(JB):                    4.50e-204
Kurtosis:                       5.025   Cond. No.                     1.87e+05
==============================================================================
```

After optimization, the model dropped PC3 and PC5 because they were larger than 0.05.

3. Give the mean squared error (MSE) of the optimized model used on the training set.

Below is my code calculating the mean squared error on the Training Set.

```python
from sklearn.metrics import mean_squared_error

y_train_pred = optimized_model.predict(X_train_optimized)

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

```
Mean Squared Error on Training Set: 7333759601.89
```

4. 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 (MSE).

Below is the code to calculate the mean squared error on the Test Set.

```
X_test_optimized = sm.add_constant(X_test)[X_train_optimized.columns]

y_test_pred = optimized_model.predict(X_test_optimized)

mse_test = mean_squared_error(y_test, y_test_pred)
print(f"Mean Squared Error on Test Set: {mse_test:.2f}")
```

```
Mean Squared Error on Test Set: 6990302579.66
```

G. Summarize your data analysis by doing the following:

1. List the packages or libraries you have chosen for Python or R and justify how each item on the list supports the analysis.

Here are all the Python packages I used and the reason for using them.

- Pandas - Allows us to import, read, and change datasets
- Sklearn - Allows us to do multiple crucial steps in the process such as splitting the data into separate training and test sets, importing PCA which allows us to run principal component analysis and standard scaler which scales the PCA properly
- Statsmodels.api - Allows us to perform regression and extract the model parameters
- Statsmodels.stats.outliers - Allows us to import the variance inflation factor to help verify the assumptions
- Numpy - Allows us to calculate the values for our graphs
- Seaborn/Matplotlib.pyplot - Allows us to use graph and visual different data relationships

2. Discuss the method used to optimize the model and justification for the approach.

We used the Backward Stepwise Elimination method to optimize the dataset, which helps remove the least important variables. This helps improve the model performance and prevent overfitting by eliminating unnecessary predictors. It does so by relying on p-values to determine which insignificant predictors to remove. In our case any p-value greater than 0.05 will be dropped, thus simplifying the model by keeping the most significant features. We used BSE because our dataset is large enough, and variables can be removed without significantly harming the integrity of the dataset.

3. Discuss the verification of assumptions used to create the optimized model.

When building a linear regression model, we assume certain conditions are met so that the model may perform adequately and hold the results valid. If these assumptions are not met, the values produced by the model may be biased or skewed. To verify our model, we will check the dataset for multicollinearity.

To do so, I will first check for multicollinearity with the code below. We do this because the independent variables should not be highly correlated with each other.

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
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   const  1.000151
1     PC1  1.000078
2     PC2  1.000083
3     PC4  1.000130
```

With the VIF value being close to 1 and under the safe threshold of 5, we can safely determine that none of the variables are multicollinear. Therefore, we can safely verify the assumptions are met.

4. Provide the regression equation and discuss the coefficient estimates

Below is the linear regression equation.

$$y = \beta_0 + \beta_n x_n$$

Where:

- y = Dependent variable (Predicted Price in this case)
- x_n=Independent variable (For our case we will have 3)
- β_0 = The intercept when $x = 0$ (Constant Coefficient)
- β_n=The slope

With our values calculated from the optimized dataset, we can plug the values in and find the equation for this dataset.

The constant coefficient, β_0, equals 307,500. This value will be our intercept.

PC1 coefficient, β_1, equals 0.6637.

PC2 coefficient, β_2, equals 33.6408.

PC4 coefficient, β_3, equals -96.9494

Subbing in variables where needed this will be our final equation

$$\textit{Predicted Price} = 307{,}5000 + (0.6637 \times \textit{PC1}) + (33.6408 \times \textit{PC2}) + (-96.9494 \times \textit{PC4})$$

5. Discuss the model metrics by addressing each of the following:

- **the R2 and adjusted R2 of the training set**
- **the comparison of the MSE for the training set to the MSE of the test set**

The R squared and adjusted R squared value was 67.9%. This means that the model can accurately predict the variability in home prices 67.9% of the time. Which for real estate prices and how unpredictable they can be is considered a good model.

The mean squared value for the training set is 7333759601.89, and the test set is 6990302579.66. Since the Mean Squared Error on the Test Set is lower than the training set, it is a sign that the model is performing well and that there's not much overfitting, with the squared difference being $18,532.59.

6. Discuss the results and implications of your prediction analysis.

With such a high accuracy score, we can confidently say that PCA should be standard procedure along with testing many different variables. After running PCA, the model combined numerous variables into new components to help reduce overfitting, thus producing an accuracy score of 67.9%, which means the model was only inaccurate 32.1% of the time, and for real estate prices, this is deemed as a well-fitting model. With more tweaks and fine-tuning, the model can be optimized even further.

7. Recommend a course of action for the real-world organizational situation from part B1 based on your results and implications discussed in part E6.

My recommended course of action to builders and benefactors would be, if you want the best and most accurate model for predicting the price of a home, I would use as many non-categorical variables as we reasonably can. This means that we must collect many data points and have lots of information before building the home to find the best possible return on price with our model.

Sources

No Sources were used except WGU Material