Data Mining II

Jason Willis

College of Information Technology,

Western Governors University

Dr. Kesselly Kamara

June 20th, 2022

Table of Contents for Each Rubric

Hospital Readmission Problem

For our chain of hospitals to lower readmission concerns, we need to identify patients who have increased risk of rehospitalization within a month of their release. According to Schuller (2020), non-obese adults were 21% less likely to be readmitted than obese adults. A readmission study by Gert, et. al. (2002) showed a correlation between longer initial hospital stays and readmission. Within the provided dataset, I'm leveraging these studies to help create my hypothetical question and shape my approach in finding potential patient groups with a statistically significant chance for readmission outcomes.

After viewing the provided medical_clean.csv data set and accompanying data dictionary, there seems to be some patient groupings which are aligned with the research mentioned above. For instance, the following patient data fields: Initial patient admin days, Total Charges, and Initial Says (inpatient) both caught my attention and were underscored by the research mentioned above. While my initial feelings towards these variables might make them feel related, are they?

A1 – Proposal of Question

From information about previous patients who were readmitted, can we predict which patients are likely to be readmitted in the future?

A2 – Defined Goal

The goal of our analysis is to logically investigate the provided data set and, with evidence, support or reject the hypothesis. Some data will need to be converted from categorical to numerical data types prior to processing. Our objective is to see how, if at all, any patient's data correlate with potential for readmission.

B1 – Explanation of Clustering Technique

According to Larose (2019) "A *cluster* is a collection of records that are similar to one another and dissimilar to records in other clusters." Unlike KNN, where the method tries to classify a record, clustering attempts to segment data into uniform subdivisions of the whole data set.

B2 – Summary of Technique Assumption:

From the DataCamp lesson "Cluster Analysis in Python" the following K-Means assumptions to consider (Daityari, 2022):

- Bias of analysis towards uniformly sized clusters

B3 – Packages or Libraries List

The following Python libraries were used followed by their corresponding reason for use:

- Pandas – Used to import dataset and data analysis tasks.
- Numpy – Used for describing the data set and computing distances in KNN.
- Matplotlib – Used for viewing the testing and actual data as a scatter plot.
- Seaborn – Used for creating a heatmap when looking for null values in the original dataset and ggplot style graph matrix to help visualize univariate data.
- Scipy – Used for statistical techniques
- Sklearn – Used for preprocessing, PCA, model splitting and K-Means tasks.

C1 – Data Preprocessing

A preprocessing goal achieved for this model was standardization, as seen below in Figure 1 using sklearn. Standardization helps by formatting variables on similar scales

(subtracting the mean and dividing the standard deviation) as to reduce variables from being a dominant influence to the model.

K-Means Clustering

```
# http://mlreference.com/k-means-standardization-sklearn
# Preprocessing – Standardize Data

# Create the scalar.
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(pca_df_binary)

# Standardize the columns.
pca_df_binary_standardized = pca_df_binary.copy()
standardized_data = scaler.transform(pca_df_binary)
pca_df_binary_standardized[pca_df_binary_standardized.columns] = standardized_data
pca_df_binary_standardized.head()
```

	Lat	Lng	Population	Children	Age	Income	VitD_levels	Doc_visits	Full_meals_eaten	vitD_supp	...
0	-0.814668	0.297134	-0.473168	-0.507129	-0.024795	1.615914	0.583603	0.944647	-0.993387	-0.634713	...
1	-1.463305	0.395522	0.090242	0.417277	-0.121706	0.221443	0.483901	-0.967981	0.990609	0.956445	...
2	0.886966	-0.354788	0.482983	0.417277	-0.024795	-0.915870	0.046227	-0.967981	-0.001389	-0.634713	...
3	0.952530	-0.149403	-0.526393	-0.969332	1.186592	-0.026263	-0.687811	-0.967981	-0.001389	-0.634713	...
4	-0.213252	0.943984	-0.315586	-0.507129	-1.526914	-1.377325	-0.260366	-0.011667	-0.993387	2.547602	...

Figure 1 - Preprocessing Goal: Scaling

C2 – Data Set Variables

Once preprocessing was completed, to include creating dummy variables (Figure 3), the data as shown in Figure 4, had all numerical data types:

```
[159]: 1 pca_df_binary_standardized.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 40 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Lat                                   10000 non-null  float64
 1   Lng                                   10000 non-null  float64
 2   Population                            10000 non-null  float64
 3   Children                              10000 non-null  float64
 4   Age                                   10000 non-null  float64
 5   Income                                10000 non-null  float64
 6   VitD_levels                           10000 non-null  float64
 7   Doc_visits                            10000 non-null  float64
 8   Full_meals_eaten                      10000 non-null  float64
 9   vitD_supp                             10000 non-null  float64
 10  Initial_days                          10000 non-null  float64
 11  TotalCharge                           10000 non-null  float64
 12  Additional_charges                    10000 non-null  float64
 13  Area__Suburban                        10000 non-null  float64
 14  Area__Urban                           10000 non-null  float64
 15  Marital__Married                      10000 non-null  float64
 16  Marital__Never Married                10000 non-null  float64
 17  Marital__Separated                    10000 non-null  float64
 18  Marital__Widowed                      10000 non-null  float64
 19  Gender__Male                          10000 non-null  float64
 20  Gender__Nonbinary                     10000 non-null  float64
 21  Soft_drink__Yes                       10000 non-null  float64
 22  Initial_admin__Emergency Admission    10000 non-null  float64
 23  Initial_admin__Observation Admission  10000 non-null  float64
 24  HighBlood__Yes                        10000 non-null  float64
 25  Stroke__Yes                           10000 non-null  float64
 26  Complication_risk__Low                10000 non-null  float64
 27  Complication_risk__Medium             10000 non-null  float64
 28  Overweight__Yes                       10000 non-null  float64
 29  Arthritis__Yes                        10000 non-null  float64
 30  Diabetes__Yes                         10000 non-null  float64
 31  Hyperlipidemia__Yes                   10000 non-null  float64
 32  BackPain__Yes                         10000 non-null  float64
 33  Anxiety__Yes                          10000 non-null  float64
 34  Allergic_rhinitis__Yes                10000 non-null  float64
 35  Reflux_esophagitis__Yes               10000 non-null  float64
 36  Asthma__Yes                           10000 non-null  float64
 37  Services__CT Scan                     10000 non-null  float64
 38  Services__Intravenous                 10000 non-null  float64
 39  Services__MRI                         10000 non-null  float64
dtypes: float64(40)
memory usage: 3.1 MB
```

Figure 2 - Data Types

Create Dummies Function for specific datatypes, reduce multicoliniarity

```
def dummify(df, max_cols=10):
    # Get list of orig df cols
    df_cols = df.columns
    # Make copy of df
    df_dummy=df.copy()
    # ForEach Col, check if numeric.  If no, convert to binary
    for t in df_cols:
        if str(df_dummy[t].dtypes) not in ['float64', 'int64']:  # if numeric var dissapears, check dtypes and add new ones as needed.
            # take non numerics, set (unique) list, then sort
            val_list = sorted(list(set(df_dummy[t])))
            if len(val_list) > 1 and len(val_list) <= max_cols:
                for v in val_list[1:]: # make 'no' the dummy var
                    df_dummy[t+"__"+str(v)]=df_dummy[t].apply(lambda x: 1 if x == v else 0)
            df_dummy.drop([t],axis=1, inplace=True)
    return df_dummy
```

Figure 3 - Create Dummy Data

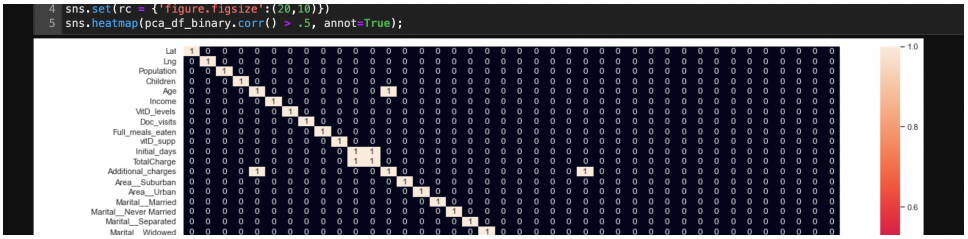


Figure 4 - Correlation Matrix Showing Results Above 0.5

C3 Steps for Analysis

Initially, the dataset was loaded using pd.read_csv('medical_clean.csv') and a data frame was created. Some exploratory data analysis was performed to familiarize myself to the data, look for missing values and view data statistics using df.describe(). Since the target and many predictor variable were initially a categorical data series, I converted this column to integers (0,1) using dummy data. A correlation matrix identified data greater than 0.5 with our target "ReAdmis_Yes" for classification. Using PCA and experimenting with various *n* components, 3 was used with a shape of 10,000 rows by 3 components and 40 columns.

Next, the pca_df_binary data was standardized (Figure 1) and fit to the K-Means clustering technique.

K-Means Clustering

```
# http://mlreference.com/k-means-standardization-sklearn
# Preprocessing - Standardize Data

# Create the scalar.
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(pca_df_binary)

# Standardize the columns.
pca_df_binary_standardized = pca_df_binary.copy()
standardized_data = scaler.transform(pca_df_binary)
pca_df_binary_standardized[pca_df_binary_standardized.columns] = standardized_data
pca_df_binary_standardized.head()
```

```
# Clusters --> ReAdmin vs Not ReAdmitted
kmeans = KMeans(n_clusters=2).fit(pca_df_binary_standardized)
```

```
pca_df_binary_standardized.info()
```

Figure 5 - K-Means Clustering – and Data Standardization

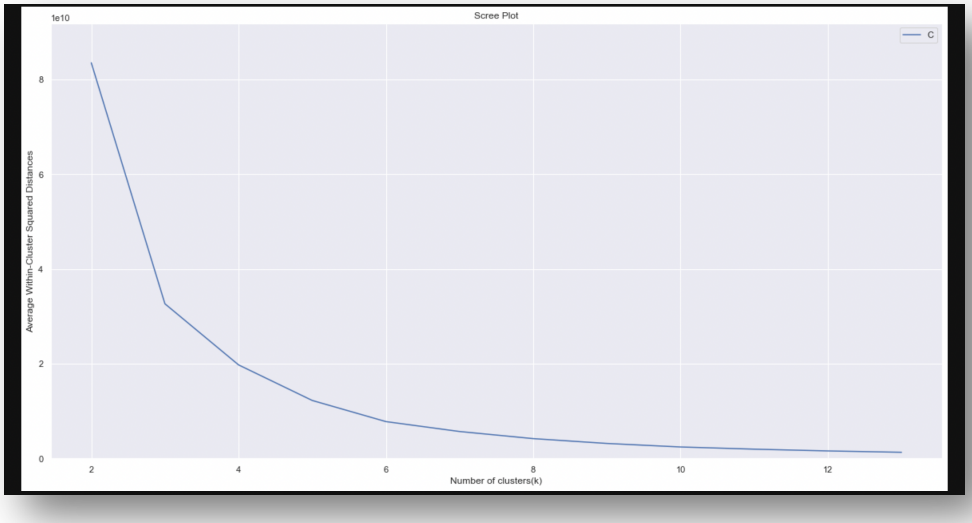


Figure 6 - Scree Diagram

First, the algorithm iterates through a number of cluster means by finding the closest squared distance of each point to a user-defined K(s). Each row is assigned to the nearest cluster mean and the algorithm continues to converge around the K-mean (cluster group) until data stops adjusting. A Scree diagram helped figure out an optimal number of clusters (Figure 6). Using sklearn's train-test-split method the standardized data set was run to create training and testing models (Figure 7).

```
# X = predictor, y = response
X_train, X_test, y_train, y_test = train_test_split(pca_df_binary_standardized, df_dummies['ReAdmis_Yes'], test_size=0.33, random_state=42)
```

```
print("PCA_Standardized Shape: " + str(pca_df_binary_standardized.shape))
print("X_Train Shape: " + str(X_train.shape))
print("y_Train Shape: " + str(y_train.shape))
print("X_Test Shape: " + str(X_test.shape))
print("y_Test Shape: " + str(y_test.shape))
```

```
PCA_Standardized Shape: (10000, 40)
X_Train Shape: (6700, 40)
y_Train Shape: (6700,)
X_Test Shape: (3300, 40)
y_Test Shape: (3300,)
```

```
print("***** Train_Set *****")
print(X_train.head())
print("\n")
print("***** Test_Set *****")
print(X_test.head())
```

```
***** Train_Set *****
            Lat       Lng  Population  Children       Age    Income  \
8371 -0.313238  0.627401   -0.426690 -0.969332  0.750493 -0.700692
5027 -2.210094  0.609127    0.097190  0.417277  0.605126  0.187865
9234 -0.518676  0.442590    0.465106 -0.969332 -1.478459  0.977979
3944  0.792480  1.002280    0.174362 -0.507129 -0.896993  0.218487
6862 -0.788833 -1.757797    0.348741  1.803886 -0.896993  0.177451

      VitD_levels  Doc_visits  Full_meals_eaten  VitD_supp  ...  \
8371    -1.027936   -0.967981          0.990609  -0.634713  ...
5027     0.135836   -0.967981          1.982607  -0.634713  ...
9234    -0.157318   -0.967981         -0.001389   0.956445  ...
3944    -0.380859   -0.011667         -0.993387   0.956445  ...
6862     0.792105   -0.011667         -0.993387  -0.634713  ...

      Diabetes_Yes  Hyperlipidemia_Yes  BackPain_Yes  Anxiety_Yes  \
8371      1.628589           -0.713268      1.196129    -0.688360
```

Figure 7 - Create Training and Testing Models

C4 – Cleaned Data Set

The cleaned data set was saved to "cleaned_pca_df_binary_standardized_plus.csv".

D1 – Output and Intermediate Calculations

Data is standardized to compare variables with a similar scale (Figure 5) and fit to the model, using predicted and actual data from the split (Figure 7). As seen in Figure 8 the predicted and actual cluster data is almost identical.

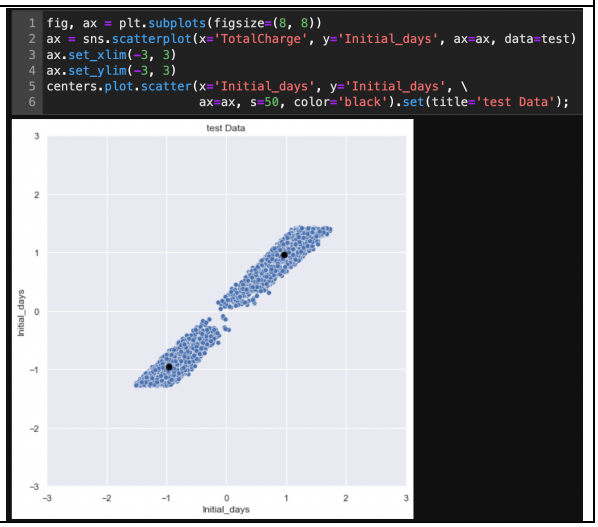
Predicted	Actual
	

Figure 8 - Plotting Predicted and Actual Data

D2 – Code Execution

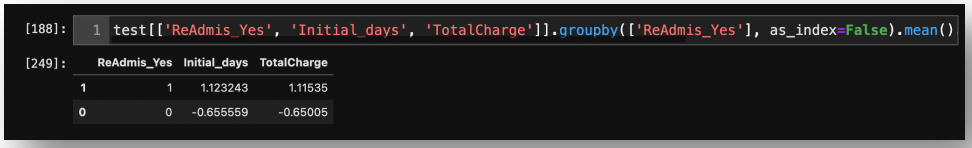
Code is located in the "D212 - DataMiningII_PA_jw.ipynb" document.

E1 – Accuracy of Clustering Technique

As seen in Figure 8 the predicted and actual cluster data is almost identical. The cluster centroids are well placed and there is a clear separation between clusters.

E2 – Results and Implications

The model is highly accurate. Looking at Figure 5, you can see that each plotted point is a patient from the data set. As the patient's initial stay is longer and their total charge is higher, the chance that they are readmitted is closer to "Yes".



```
[188]: 1 test[['ReAdmis_Yes', 'Initial_days', 'TotalCharge']].groupby(['ReAdmis_Yes'], as_index=False).mean()
```

[249]:

	ReAdmis_Yes	Initial_days	TotalCharge
1	1	1.123243	1.11535
0	0	-0.655559	-0.65005

Figure 9 – Model Findings

E3 – Limitation

One limitation is finding the best number of *k* clusters. Using a scree diagram helps (Figure 6) though this can feel a bit loose and won't always provide the best number of clusters to input into your model. Meaning, sometimes, you will need to play with the number of +/- *k provided.*

E4 – Course of Action

Our model is highly predictive of patient readmissions rates and follows previously analyzed performance assessment outcomes. My recommendation would be to focus on researching and identifying patient's threshold criteria's for initial days and total charges as these predictor variables helped predict readmission with a high accuracy rate. By focusing on these predictors, we should be able to predict patients that will have a higher likelihood of readmission within 30 days.

F – Panopto Demonstration

Panopto video Will be uploaded once report is finalized.

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=6906d4d4-ad26-4449-8cd6-aeba01419d20

G – Sources for Third-Party Code

- Help using Markdown: https://www.markdownguide.org/basic-syntax/
- Help to see ALL columns: https://stackoverflow.com/questions/24524104/pandas-describe-is-not-returning-summary-of-all-columns
- Help to create a better histogram design: https://mode.com/example-gallery/python_histogram/
- Matplotlib Help: https://matplotlib.org/2.1.2/api/_as_gen/matplotlib.pyplot.plot.html
- Multiple ways to conduct ANOVA: https://www.marsja.se/four-ways-to-conduct-one-way-anovas-using-python/
- Numpy Help: https://numpy.org/doc/stable/
- Pandas Help: https://pandas.pydata.org/docs/user_guide/index.html#user-guide
- Python Help: https://docs.python.org/3.9/library/index.html
- Scipy.stats Help: https://docs.scipy.org/doc/scipy/reference/tutorial/stats.html

References

Daityari, S., et. al., Cluster Analysis in Python, (2022). Found At: https://app.datacamp.com/learn/courses/cluster-analysis-in-python

Gert P Westert, Ronald J Lagoe, Ilmo Keskimäki, Alastair Leyland, Mark Murphy, An international study of hospital readmissions and related utilization in Europe and the USA, Health Policy, Volume 61, Issue 3, 2002, Pages 269-278, ISSN 0168-8510, https://doi.org/10.1016/S0168-8510(01)00236-6. (https://www.sciencedirect.com/science/article/pii/S0168851001002366)

Larose, D., C., & Larose, D., T. (2019). Data Science Using Python and R. Wiley. https://www.wiley.com/en-us/Data+Science+Using+Python+and+R-p-9781119526810

Paul, S. (2018). K-Means Clustering in Python with Scikit-Learn. *DataCamp*.com. Retrieved From: https://www.datacamp.com/tutorial/k-means-clustering-python

Schuller K. A. (2020). Is obesity a risk factor for readmission after acute myocardial infarction? *Journal of healthcare quality research*, *35*(1), 4–11. https://doi.org/10.1016/j.jhqr.2019.09.002

Urbonas K. (2022). Customer Segmentation in Python. Data Preprocessing for K-Means Clustering. *DataCamp.* https://campus.datacamp.com/courses/customer-segmentation-in-python/data-pre-processing-for-clustering?ex=1