William Stults - Clustering Techniques (D212 Task 1)

January 29, 2023

---

1 Part I: Research Question

1.1 Research Question

My dataset for this data mining exercise includes data on a telco company's current and former subscribers, with an emphasis on customer churn (whether customers are maintaining or discontinuing their subscription to service). Data analysis performed on the dataset will be aimed with this research question in mind: which features of the data set exhibit the greatest amount of variance among data points, thereby having the greatest impact on determining which cluster each data point belongs to? Continuous numerical data will include numerical data which includes a measurable variable, rather than numerical data used as a label.

1.2 Objectives and Goals

Conclusions gleaned from the analysis of this data can benefit stakeholders by revealing information on how customers can be grouped based on their characteristics. Such information may be used to predict future customer events based on another variable the telco company may be interested in. My goal will be to first determine an optimal value for "k" by evaluating inertia values, then utilize k-means clustering to group customers into "k" groupings. Once done, I will analyze the results to determine which features of the data set had the greatest impact on determining which data point belongs to which cluster.

---

2 Part II: Technique Justification

2.1 K-means Clustering

K-means clustering is an unsupervised learning algorithm. It helps to group similar data points (rows in a data set) together, while attempting to maintain distance between each cluster to eliminate overlap. Before beginning the clustering process there must be a known value for "k". This value reflects the number of clusters to be generated by the algorithm (Nagar, 2020).

The algorithm takes a numerical value as input, the "k" variable, and initializes k cluster centers. The algorithm then analyzes each data point in the data set, measuring the distance between each data point and each cluster center, or "centroid". Each data point is grouped with the centroid it is closest to. Once every data point has been grouped, the algorithm re-calculates the centroid values by taking the sum of all the data points belonging to a centroid and dividing that value by the number of data points grouped with that centroid. The process repeats until there is no change in a centroid's value after re-calculation, meaning each centroid has centered itself in the middle of its group of data points, or "cluster" (Al-Masri, 2019).

The expected outcome will be "k" clusters, in this case groups of customers, with a similar number of customers in each cluster. The clusters should group together customers with similar characteristics, and analysis of the clusters should reveal which characteristics are most prevalent in each cluster.

One assumption of k-means clustering is that the resulting clusters will be similar in size. This assumption helps in determing where the boundaries of each cluster should be and how many data points should make up its members (Perceptive Analytics, 2017). K-means also assumes the clusters will be of a generally spherical shape, as the data points within a cluster would only fall within a specified maximum distance from its center (Nagar, 2020).

2.2 Tool Selection

All code execution was carried out via Jupyter Lab, using Python 3. I used Python as my selected programming language due to prior familiarity and broader applications when considering programming in general. R is a very strong and robust language tool for data analysis and statistics but finds itself somewhat limited to that niche role (Insights for Professionals, 2019). I utilized the NumPy, Pandas, and Matplotlib libraries to perform many of my data analysis tasks, as they are among the most popular Python libraries employed for this purpose and see widespread use. Seaborn is included primarily for its versatility and pleasing aesthetics when created visulizations.

Beyond these libraries, I relied upon the scikit-learn library. Scikit-learn supports k-means clustering, variable scaling, accuracy scoring and principal component analysis (KMeans, StandardScaler, silhouette_score and PCA functions, respectively), and the course material relied upon its use. I also used the parallel_coordinates function from pandas' plotting module for use in creating visualizations for the purpose of analyzing k-means clustering results.

[1]:
```
# Imports and housekeeping
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
from pandas.plotting import parallel_coordinates
```

---

3 Part III: Data Preparation

3.1 Data Preparation Goals and Data Manipulations

I would like my data to include only variables relevant to my research question, and to be clean and free of missing values and duplicate rows. K-means clustering can only operate on continuous variables, so my first goal in data preparation is to make sure the data I will be working with contains no categorical data.

A list of the variables I will be using for my analysis is included below, along with their variable types and a brief description of each.

- Population - **continuous** - *Population within a mile radius of customer*
- Children - **continuous** - *Number of children in customer's household*
- Age - **continuous** - *Age of customer*
- Income - **continuous** - *Annual income of customer*
- Outage_sec_perweek - **continuous** - *Average number of seconds per week of system outages in the customer's neighborhood*
- Email - **continuous** - *Number of emails sent to the customer in the last year*
- Contacts - **continuous** - *Number of times customer contacted technical support*
- Yearly_equip_failure - **continuous** - *The number of times customer's equipment failed and had to be reset/replaced in the past year*
- Tenure - **continuous** - *Number of months the customer has stayed with the provider*
- MonthlyCharge - **continuous** - *The amount charged to the customer monthly*
- Bandwidth_GB_Year - **continuous** - *The average amount of data used, in GB, in a year by the customer*
- Item1: Timely response - **continuous** - *survey response - scale of 1 to 8 (1 = most important, 8 = least important)*
- Item2: Timely fixes - **continuous** - *survey response - scale of 1 to 8 (1 = most important, 8 = least important)*
- Item3: Timely replacements - **continuous** - *survey response - scale of 1 to 8 (1 = most important, 8 = least important)*
- Item4: Reliability - **continuous** - *survey response - scale of 1 to 8 (1 = most important, 8 = least important)*
- Item5: Options - **continuous** - *survey response - scale of 1 to 8 (1 = most important, 8 = least important)*
- Item6: Respectful response - **continuous** - *survey response - scale of 1 to 8 (1 = most important, 8 = least important)*
- Item7: Courteous exchange - **continuous** - *survey response - scale of 1 to 8 (1 = most important, 8 = least important)*
- Item8: Evidence of active listening - **continuous** - *survey response - scale of 1 to 8 (1 = most important, 8 = least important)*

My first steps will be to import the complete data set and execute functions that will give me information on its size and the data types of its variables. I will then narrow the data set to a new dataframe containing only the variables I am concerned with, and then utilize functions to determine if any null values or duplicate rows exist. By using the index_col parameter in my import I utilize CaseOrder, the data set's natural index column, as the index column in my pandas dataframe.

[2]:
```
# Import the main dataset
df = pd.read_csv('churn_clean.csv', dtype={'locationid':np.int64},␣
↪index_col=[0])
```

[3]:
```
# Display dataset info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 1 to 10000
Data columns (total 49 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Customer_id           10000 non-null  object
 1   Interaction           10000 non-null  object
 2   UID                   10000 non-null  object
 3   City                  10000 non-null  object
 4   State                 10000 non-null  object
 5   County                10000 non-null  object
 6   Zip                   10000 non-null  int64
 7   Lat                   10000 non-null  float64
 8   Lng                   10000 non-null  float64
 9   Population            10000 non-null  int64
 10  Area                  10000 non-null  object
 11  TimeZone              10000 non-null  object
 12  Job                   10000 non-null  object
 13  Children              10000 non-null  int64
 14  Age                   10000 non-null  int64
 15  Income                10000 non-null  float64
 16  Marital               10000 non-null  object
 17  Gender                10000 non-null  object
 18  Churn                 10000 non-null  object
 19  Outage_sec_perweek    10000 non-null  float64
 20  Email                 10000 non-null  int64
 21  Contacts              10000 non-null  int64
 22  Yearly_equip_failure  10000 non-null  int64
 23  Techie                10000 non-null  object
 24  Contract              10000 non-null  object
 25  Port_modem            10000 non-null  object
 26  Tablet                10000 non-null  object
 27  InternetService       10000 non-null  object
 28  Phone                 10000 non-null  object
 29  Multiple              10000 non-null  object
 30  OnlineSecurity        10000 non-null  object
 31  OnlineBackup          10000 non-null  object
 32  DeviceProtection      10000 non-null  object
 33  TechSupport           10000 non-null  object
```

```
 34  StreamingTV           10000 non-null  object
 35  StreamingMovies       10000 non-null  object
 36  PaperlessBilling      10000 non-null  object
 37  PaymentMethod         10000 non-null  object
 38  Tenure                10000 non-null  float64
 39  MonthlyCharge         10000 non-null  float64
 40  Bandwidth_GB_Year     10000 non-null  float64
 41  Item1                 10000 non-null  int64
 42  Item2                 10000 non-null  int64
 43  Item3                 10000 non-null  int64
 44  Item4                 10000 non-null  int64
 45  Item5                 10000 non-null  int64
 46  Item6                 10000 non-null  int64
 47  Item7                 10000 non-null  int64
 48  Item8                 10000 non-null  int64
dtypes: float64(7), int64(15), object(27)
memory usage: 3.8+ MB
```

[4]:
```
# Display dataset top 5 rows
df.head()
```

[4]:
```
          Customer_id                           Interaction  \
CaseOrder
1             K409198  aa90260b-4141-4a24-8e36-b04ce1f4f77b
2             S120509  fb76459f-c047-4a9d-8af9-e0f7d4ac2524
3             K191035  344d114c-3736-4be5-98f7-c72c281e2d35
4              D90850  abfa2b40-2d43-4994-b15a-989b8c79e311
5             K662701  68a861fd-0d20-4e51-a587-8a90407ee574

                                        UID         City State  \
CaseOrder
1          e885b299883d4f9fb18e39c75155d990  Point Baker    AK
2          f2de8bef964785f41a2959829830fb8a  West Branch    MI
3          f1784cfa9f6d92ae816197eb175d3c71      Yamhill    OR
4          dc8a365077241bb5cd5ccd305136b05e      Del Mar    CA
5          aabb64a116e83fdc4befc1fbab1663f9   Needville    TX

                          County    Zip       Lat        Lng  Population  …  \
CaseOrder                                                                 …
1          Prince of Wales-Hyder  99927  56.25100 -133.37571          38  …
2                        Ogemaw  48661  44.32893  -84.24080       10446  …
3                       Yamhill  97148  45.35589 -123.24657        3735  …
4                     San Diego  92014  32.96687 -117.24798       13863  …
5                     Fort Bend  77461  29.38012  -95.80673       11352  …

          MonthlyCharge Bandwidth_GB_Year Item1  Item2  Item3  Item4 Item5  \
CaseOrder
```

```
1             172.455519        904.536110     5       5       5       3     4
2             242.632554        800.982766     3       4       3       3     4
3             159.947583       2054.706961     4       4       2       4     4
4             119.956840       2164.579412     4       4       4       2     5
5             149.948316        271.493436     4       4       4       3     4

           Item6  Item7  Item8
CaseOrder
1              4      3      4
2              3      4      4
3              3      3      3
4              4      3      3
5              4      4      5

[5 rows x 49 columns]
```

[5]:
```
# Trim data set to variables relevant to research question
columns = ['Population', 'Children', 'Age', 'Income', 'Outage_sec_perweek',␣
 ↪'Email', 'Contacts',
           'Yearly_equip_failure', 'Tenure', 'MonthlyCharge',␣
 ↪'Bandwidth_GB_Year', 'Item1', 'Item2',
           'Item3', 'Item4', 'Item5', 'Item6', 'Item7', 'Item8']
df_data = pd.DataFrame(df[columns])
# Store the data set in variable 'X'
X = df_data
```

[6]:
```
# Check data for null or missing values
df_data.isna().any()
```

[6]:
```
Population              False
Children                False
Age                     False
Income                  False
Outage_sec_perweek      False
Email                   False
Contacts                False
Yearly_equip_failure    False
Tenure                  False
MonthlyCharge           False
Bandwidth_GB_Year       False
Item1                   False
Item2                   False
Item3                   False
Item4                   False
Item5                   False
Item6                   False
Item7                   False
```

```
Item8                    False
dtype: bool
```

[7]:
```
# Check data for duplicated rows
df_data.duplicated().sum()
```

[7]:
```
0
```

[8]:
```
df_data.head()
```

[8]:
```
           Population  Children  Age    Income  Outage_sec_perweek  Email  \
CaseOrder
1                  38         0   68  28561.99            7.978323     10
2               10446         1   27  21704.77           11.699080     12
3                3735         4   50   9609.57           10.752800      9
4               13863         1   48  18925.23           14.913540     15
5               11352         0   83  40074.19            8.147417     16

           Contacts  Yearly_equip_failure     Tenure  MonthlyCharge  \
CaseOrder
1                 0                     1   6.795513     172.455519
2                 0                     1   1.156681     242.632554
3                 0                     1  15.754144     159.947583
4                 2                     0  17.087227     119.956840
5                 2                     1   1.670972     149.948316

           Bandwidth_GB_Year  Item1  Item2  Item3  Item4  Item5  Item6  Item7  \
CaseOrder
1                 904.536110      5      5      5      3      4      4      3
2                 800.982766      3      4      3      3      4      3      4
3                2054.706961      4      4      2      4      4      3      3
4                2164.579412      4      4      4      2      5      4      3
5                 271.493436      4      4      4      3      4      4      4

           Item8
CaseOrder
1              4
2              4
3              3
4              3
5              5
```

3.2 Summary Statistics

I can use the describe() function to display the summary statistics for the entire dataframe, as well as each variable I'll be evaluating for inclusion in the k-means clustering exercise.

[9]:
```
# Display summary statistics for entire dataset - continuous variables
df_data.describe()
```

[9]:
```
         Population    Children           Age         Income  \
count  10000.000000  10000.0000  10000.000000   10000.000000
mean    9756.562400      2.0877     53.078400   39806.926771
std    14432.698671      2.1472     20.698882   28199.916702
min        0.000000      0.0000     18.000000     348.670000
25%      738.000000      0.0000     35.000000   19224.717500
50%     2910.500000      1.0000     53.000000   33170.605000
75%    13168.000000      3.0000     71.000000   53246.170000
max   111850.000000     10.0000     89.000000  258900.700000

       Outage_sec_perweek         Email      Contacts  Yearly_equip_failure  \
count        10000.000000  10000.000000  10000.000000          10000.000000
mean            10.001848     12.016000      0.994200              0.398000
std              2.976019      3.025898      0.988466              0.635953
min              0.099747      1.000000      0.000000              0.000000
25%              8.018214     10.000000      0.000000              0.000000
50%             10.018560     12.000000      1.000000              0.000000
75%             11.969485     14.000000      2.000000              1.000000
max             21.207230     23.000000      7.000000              6.000000

             Tenure  MonthlyCharge  Bandwidth_GB_Year         Item1  \
count  10000.000000   10000.000000       10000.000000  10000.000000
mean      34.526188     172.624816        3392.341550      3.490800
std       26.443063      42.943094        2185.294852      1.037797
min        1.000259      79.978860         155.506715      1.000000
25%        7.917694     139.979239        1236.470827      3.000000
50%       35.430507     167.484700        3279.536903      3.000000
75%       61.479795     200.734725        5586.141370      4.000000
max       71.999280     290.160419        7158.981530      7.000000

              Item2         Item3         Item4         Item5         Item6  \
count  10000.000000  10000.000000  10000.000000  10000.000000  10000.000000
mean       3.505100      3.487000      3.497500      3.492900      3.497300
std        1.034641      1.027977      1.025816      1.024819      1.033586
min        1.000000      1.000000      1.000000      1.000000      1.000000
25%        3.000000      3.000000      3.000000      3.000000      3.000000
50%        4.000000      3.000000      3.000000      3.000000      3.000000
75%        4.000000      4.000000      4.000000      4.000000      4.000000
max        7.000000      8.000000      7.000000      7.000000      8.000000

              Item7         Item8
count  10000.000000  10000.000000
mean       3.509500      3.495600
std        1.028502      1.028633
```

```
min          1.000000        1.000000
25%          3.000000        3.000000
50%          4.000000        3.000000
75%          4.000000        4.000000
max          7.000000        8.000000
```

3.3 Further Preparation Steps

I will use the StandardScaler function to scale my variables for more accurate attribute weighting. StandardScaler transforms each variable value to have a mean of 0 and a variance of 1. Once done, every variable value will fall between -1 and 1, and the data set values can be considered "standardized". The standardized data set is then assigned to variable "X_scaled".

[10]:
```
# Scaling continuous variables with StandardScaler
scaler = StandardScaler()
scaler.fit(X)
StandardScaler(copy=True, with_mean=True, with_std=True)
X_scaled = scaler.transform(X)
```

3.4 Copy of Prepared Data Set

Below is the code used to export the prepared data set to CSV format.

[11]:
```
df_prepared = pd.DataFrame(X_scaled, columns=df_data.columns)
# Export prepared dataframe to csv
df_prepared.to_csv(r'C:\Users\wstul\d212\churn_clean_prepared.csv')
```

---

4 Part IV: Analysis

4.1 Determining the Optimal Value for "k"

Using the best "k" value, or number of clusters, is critical in order to receive good results from the clustering. With 19 features, this data frame is more likely to benefit from a lower "k" value. I will use an iterative loop to help ultimately determine which value is best, but before doing that I will need to initialize a couple of arrays. The first array contains the numbers 1-10, and will represent the k values used in the iterative loop. The second array will be empty, intended to store the results from each iteration.

[12]:
```
# Initializing the kvalues and inertia arrays
kvalues = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
inertia = np.array([])
```

4.2 Iterative Loop and Inertia Values

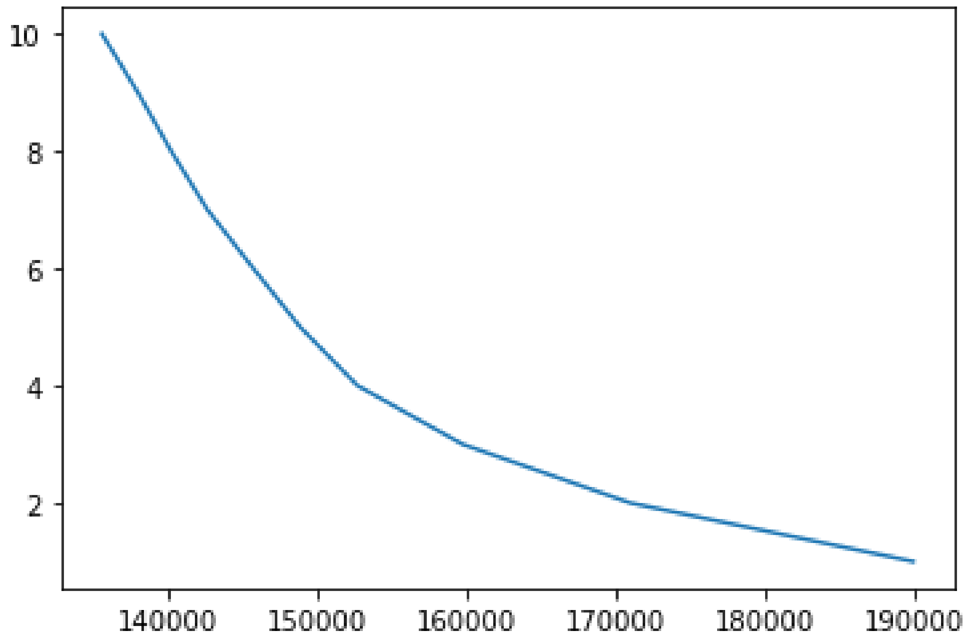
The iterative loop will run the k-means algorithm for "i" number of clusters, "i" being each number in the "kvalues" array. It will then fit the model to X_scaled, our standardized data set, print the resulting inertia value, and then add that inertia value to the "inertia" array.

[13]:
```
# Iterative loop to determine best k value
for i in kvalues:
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(X_scaled)
    print(kmeans.inertia_)
    # At the end of each loop the inertia observed during clustering is added␣
 ↪to the inertia array
    inertia = np.append(inertia, kmeans.inertia_)
```

```
189999.99999999942
170985.80514446239
159747.48793044547
152672.85862704748
148825.728746196
145682.52962918975
142597.18724832847
140161.5498601546
137931.01317082296
135507.27653759837
```

Being able to see the inertia values is good, but generating a visualization based on the inertia values and their related "k" values will be more helpful in spotting the optimal value for "k". The below code will generate that visualization.

[14]:
```
# Create a figure containing a single axes.
fig, ax = plt.subplots()
# Plot the kvalue and inertia data on the axes.
ax.plot(inertia, kvalues);
```



I can see that the "elbow" in the visualization falls at the "k" value 4, indicating 4 is the optimal value of "k" for this data set. I can then run the k-means algorithm specifying a "k" value of 4, and use the results to create clusters using KMeans.predict().

[15]:
```
# k-means for 4 clusters
kmeans = KMeans(n_clusters=4)
kmeans.fit(X_scaled)
print(kmeans.inertia_)
clusters = kmeans.predict(X_scaled)
```

```
152673.16370739162
```

4.3 Data Analysis Process

I will use the series of functions documented below to create visualizations based on the k-means clusters, which will allow me to better analyze the results.

- display_factorial_planes - utilizes matplotlib to generate a scatter plot on a factorial plane, one for each factorial plane, and highlight cluster centroids
- display_parallel_coordinates - utilizes matplotlib to display a parallel coordinates plot for the clusters
- display_parallel_coordinates_centroids - utilizes matplotlib to display a parallel coordinates plot for the centroids
- addAlpha - used to manipulate color and opacity

[16]:
```python
def display_factorial_planes(X_projected, n_comp, pca, axis_ranks, labels=None,␣
↪alpha=1, illustrative_var=None):
    # Display a scatter plot on a factorial plane, one for each factorial plane

    # For each factorial plane
    for d1,d2 in axis_ranks:
        if d2 < n_comp:

            # Initialise the matplotlib figure
            fig = plt.figure(figsize=(7,6))

            # Display the points
            if illustrative_var is None:
                plt.scatter(X_projected[:, d1], X_projected[:, d2], alpha=alpha)
            else:
                illustrative_var = np.array(illustrative_var)
                for value in np.unique(illustrative_var):
                    selected = np.where(illustrative_var == value)
                    plt.scatter(X_projected[selected, d1],␣
↪X_projected[selected, d2], alpha=alpha, label=value)
                plt.legend()

            # Display the labels on the points
            if labels is not None:
                for i,(x,y) in enumerate(X_projected[:,[d1,d2]]):
                    plt.text(x, y, labels[i],
                              fontsize='14', ha='center',va='center')

            # Define the limits of the chart
            boundary = np.max(np.abs(X_projected[:, [d1,d2]])) * 1.1
            plt.xlim([-boundary,boundary])
            plt.ylim([-boundary,boundary])

            # Display grid lines
            plt.plot([-100, 100], [0, 0], color='grey', ls='--')
            plt.plot([0, 0], [-100, 100], color='grey', ls='--')

            # Label the axes, with the percentage of variance explained
            plt.xlabel('PC{} ({}%)'.format(d1+1, round(100*pca.
↪explained_variance_ratio_[d1],1)))
            plt.ylabel('PC{} ({}%)'.format(d2+1, round(100*pca.
↪explained_variance_ratio_[d2],1)))

            plt.title("Projection of points (on PC{} and PC{})".format(d1+1,␣
↪d2+1))
            #plt.show(block=False)
```

[17]:
```python
def display_parallel_coordinates(df, num_clusters):
    # Display a parallel coordinates plot for the clusters

    # Select data points for individual clusters
    cluster_points = []
    for i in range(num_clusters):
        cluster_points.append(df[df.cluster==i])

    # Create the plot
    fig = plt.figure(figsize=(16, 15))
    title = fig.suptitle("Parallel Coordinates Plot for the Clusters",␣
↪fontsize=18)
    fig.subplots_adjust(top=0.95, wspace=0)

    # Display one plot for each cluster, with the lines for the main cluster␣
↪appearing over the lines for the other clusters
    for i in range(num_clusters):
        plt.subplot(num_clusters, 1, i+1)
        for j,c in enumerate(cluster_points):
            if i!= j:
                pc = parallel_coordinates(c, 'cluster',␣
↪color=[addAlpha(palette[j],0.2)])
        pc = parallel_coordinates(cluster_points[i], 'cluster',␣
↪color=[addAlpha(palette[i],0.5)])

        # Stagger the axes
        ax=plt.gca()
        for tick in ax.xaxis.get_major_ticks()[1::2]:
            tick.set_pad(20)
```

[18]:
```python
def display_parallel_coordinates_centroids(df, num_clusters):
    # Display a parallel coordinates plot for the centroids

    # Create the plot
    fig = plt.figure(figsize=(16, 5))
    title = fig.suptitle("Parallel Coordinates plot for the Centroids",␣
↪fontsize=18)
    fig.subplots_adjust(top=0.9, wspace=0)

    # Draw the chart
    parallel_coordinates(df, 'cluster', color=palette)

    # Stagger the axes
    ax=plt.gca()
    for tick in ax.xaxis.get_major_ticks()[1::2]:
        tick.set_pad(20)
```

[19]:
```python
def addAlpha(colour, alpha):
    # Add an alpha to the RGB colour

    return (colour[0],colour[1],colour[2],alpha)
```

My feature set currently has 19 dimensions, too many to visualize. Using principal component analysis I can narrow this down to 2 and create a new data frame with the PCA results, adding my cluster labels as an additional column.

[20]:
```python
# Create a PCA model to reduce our data to 2 dimensions for visualisation
pca = PCA(n_components=2)
pca.fit(X_scaled)

# Transform the scaled data to the new PCA space
X_reduced = pca.transform(X_scaled)

# Convert to a data frame
X_reduceddf = pd.DataFrame(X_reduced, index=X.index, columns=['PC1','PC2'])
X_reduceddf['cluster'] = clusters

X_reduceddf.head()
```

[20]:
```
                PC1       PC2  cluster
CaseOrder
1          1.943688 -1.343161        3
2         -0.206792 -1.620005        0
3         -0.670625 -0.897178        0
4          0.019301 -0.741511        3
5          1.348349 -1.844241        3
```

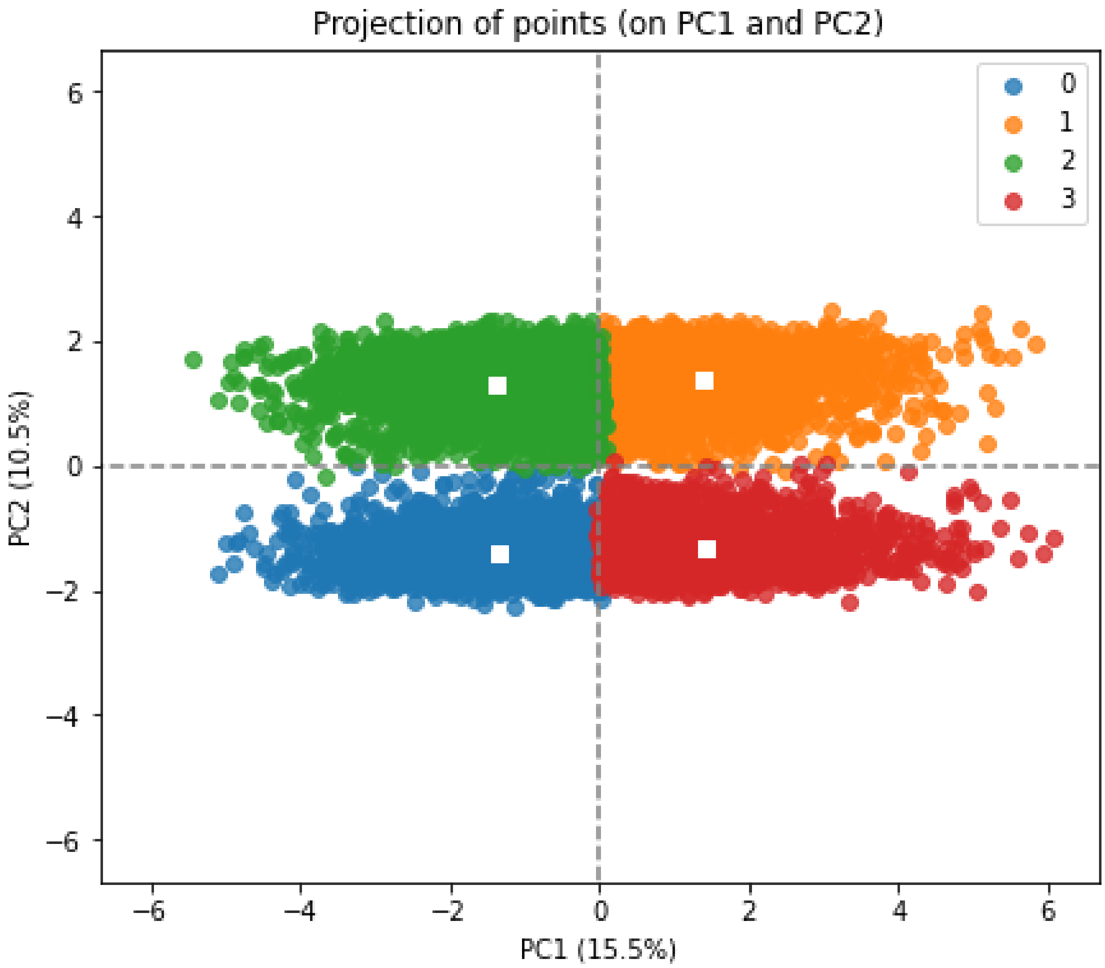
I will use PCA again on my cluster centers so I can have them appear as part of the visualized clusters as well.

[21]:
```python
centers_reduced = pca.transform(kmeans.cluster_centers_)
```

Using the display_factorial_planes function I can view the clusters and their centroids in a scatter plot and observe if any significant overlap has occurred.

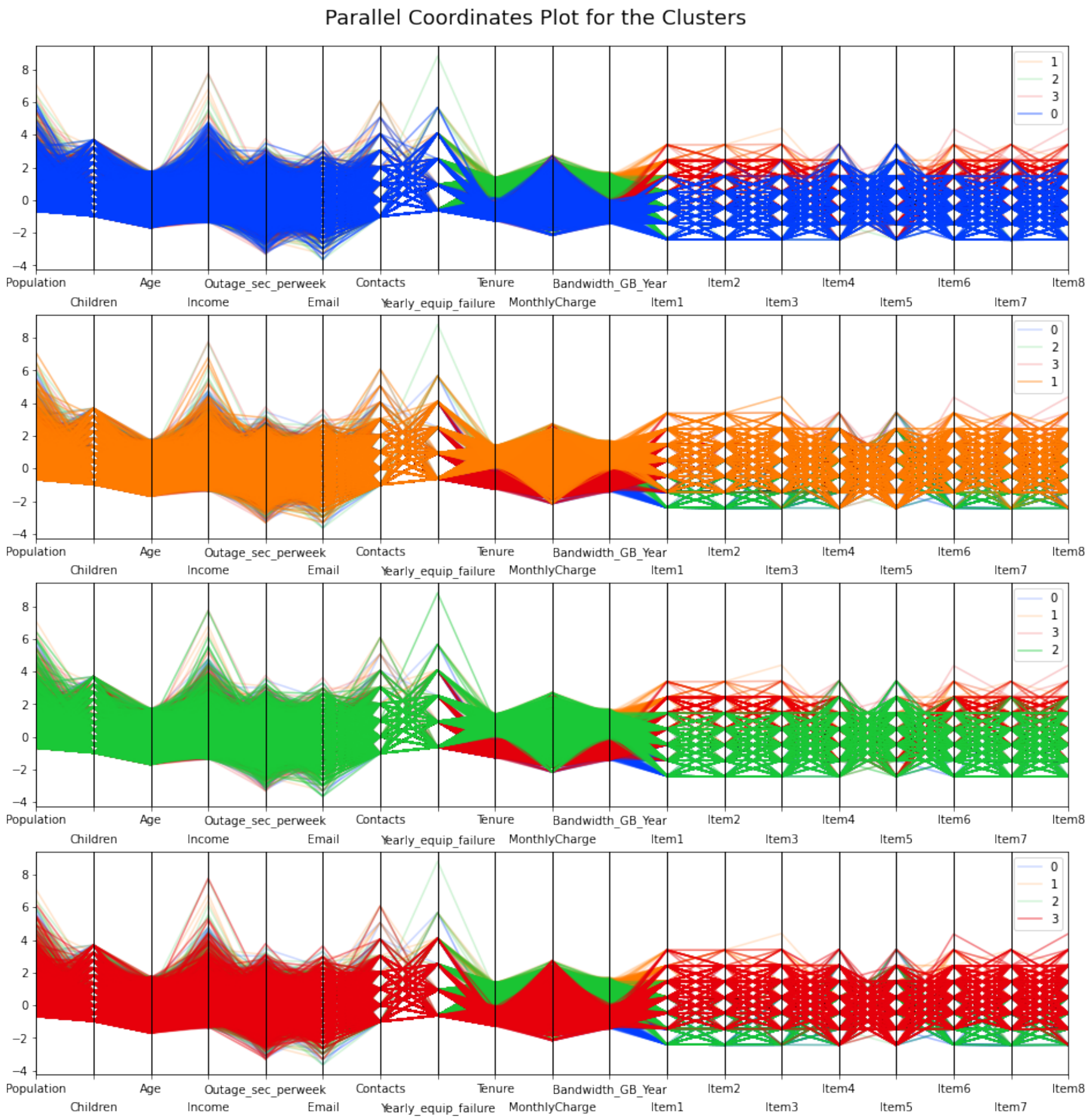
[22]:
```python
display_factorial_planes(X_reduced, 2, pca, [(0,1)], illustrative_var =␣
 ↪clusters, alpha = 0.8)
plt.scatter(centers_reduced[:, 0], centers_reduced[:, 1],
            marker='s', color='w', zorder=10)
```

[22]:
```
<matplotlib.collections.PathCollection at 0x1acfcc06bb0>
```



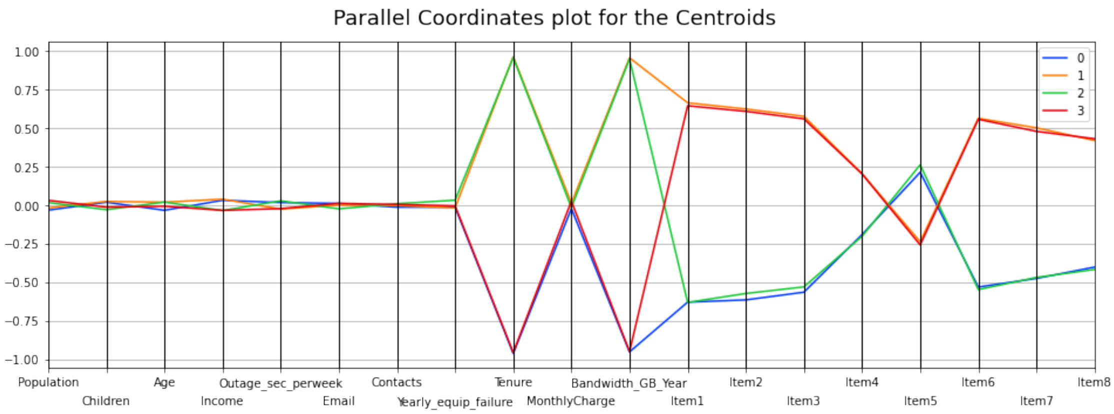
I'll add the cluster labels, the numbers 0-3, to my standardized data set in a new data frame, "X_clustered". I can then see the distribution of variables in each cluster by using parallel coordinates plots.

[23]:
```
# Add the cluster number to the original scaled data
X_clustered = pd.DataFrame(X_scaled, index=X.index, columns=X.columns)
X_clustered["cluster"] = clusters

# Display parallel coordinates plots, one for each cluster
palette = sns.color_palette("bright", 10)
display_parallel_coordinates(X_clustered, 4)
```



The parallel coordinates plots reveal a great deal about which features are represented in each cluster, and to what degree. Drilling down further, I can use the same type of plot to view the centroids for each cluster.

[24]:
```python
# Create a data frame containing our centroids
centroids = pd.DataFrame(kmeans.cluster_centers_, columns=X.columns)
centroids['cluster'] = centroids.index

display_parallel_coordinates_centroids(centroids, 10)
```



---

5 Part V: Data Summary and Implications

5.1 Clustering Accuracy

I used the silhouette_score function to judge the accuracy of the k-means clustering.

[25]:
```
# compute an average silhouette score for each point and print the score
silhouette_score_average = silhouette_score(X_scaled, kmeans.predict(X_scaled))
print(silhouette_score_average)
```

```
0.07593951280861816
```

The score was .076 rounded up. The score is above 0, which indicates above average accuracy, but may also improve with adjustments to the “k” value used in the algorithm.

5.2 Summary of Findings

The k-means algorithm identified 4 clusters. Clusters exhibit uniformity across the Population, Children, Age, Income, Outage_sec_perweek, Email, Contacts, and Yearly_equip_failure variables. For the Tenure, Bandwidth_GB_Year, and Item1 - Item8 variables, we begin to see significant divergence from the mean. Characteristics of the clusters are listed below.

- Cluster 0 - Low Tenure, Bandwidth_GB_Year, Items 1, 2, 3, and 6
- Cluster 1 - High Tenure and Bandwidth_GB_Year, High Items 1, 2, 3, 6 and 7
- Cluster 2 - High Tenure and Bandwidth_GB_Year, Low Items 1, 2, 3 and 6
- Cluster 3 - Low Tenure and Bandwidth_GB_Year, High Items 1, 2, 3 and 6

5.3 Limitations

The most significant limitation when using k-means clustering is its restriction to numerical data. While I am able to use the results of the clustering to compare against categorical variables, I cannot use those variables in the process of creating the clusters unless they are re-expressed as numerical.

5.4 Recommended Course of Action

My recommendation to the business team would be to explore further data analysis of a more predictive nature using the cluster data.

One example of this might be to see if one or more customer clusters are more likely to churn. For example, the cluster labels, in this case the numbers 0-3, can be added to the original data set.

[26]:
```
df_clusters = pd.read_csv('churn_clean.csv', dtype={'locationid':np.int64},␣
 ↪index_col=[0])
df_clusters['cluster'] = clusters
```

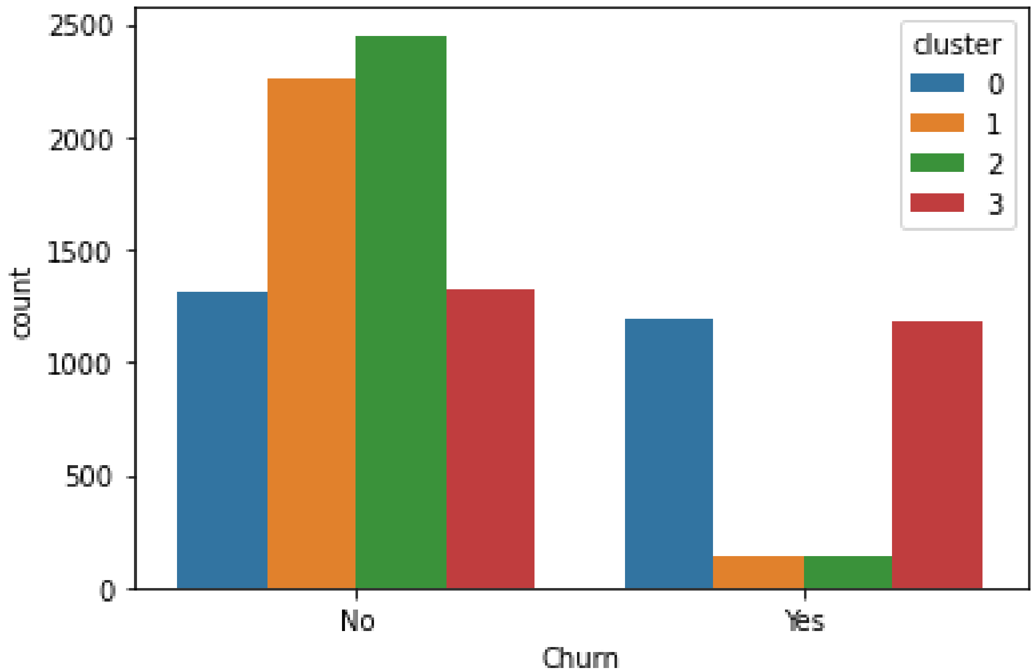
[27]:
```
df_clusters[['Customer_id','cluster']]
```

[27]:
```
          Customer_id  cluster
CaseOrder
1             K409198        3
2             S120509        0
3             K191035        0
4              D90850        3
5             K662701        3
…                 …      …
9996          M324793        2
9997          D861732        1
9998          I243405        1
9999          I641617        1
10000          T38070        2

[10000 rows x 2 columns]
```

Once available with the original set of features, a countplot reveals that customers in clusters 1 and 2 appear far less likely to churn, while customers in clusters 0 and 3 are equally likely to churn or not churn.

[28]:
```
sns.countplot(data=df_clusters, x="Churn", hue="cluster")
```

[28]:
```
<AxesSubplot:xlabel='Churn', ylabel='count'>
```



5.5 Conclusion

With an above average accuracy score and four distinct clusters, the exercise answers the research question "which features of the data set exhibit the greatest amount of variance among data points, thereby having the greatest impact on determining which cluster each data point belongs to?" by identifying Tenure, Badwidth_GB_Year, Item1, Item2, Item3, and Item6.

---

6 Part VI: Demonstration

Panopto Video Recording

A link for the Panopto video has been provided separately. The demonstration includes the following:

- Demonstration of the functionality of the code used for the analysis
- Identification of the version of the programming environment

---

7 Web Sources

https://openclassrooms.com/en/courses/5869986-perform-an-exploratory-data-analysis/6177861-analyze-the-results-of-a-k-means-clustering

https://enjoymachinelearning.com/blog/k-means-accuracy-python-silhouette/

https://stackoverflow.com/questions/36519086/how-to-get-rid-of-unnamed-0-column-in-a-pandas-dataframe-read-in-from-csv-fil

https://matplotlib.org/stable/api/markers_api.html#module-matplotlib.markers

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html

https://scikit-learn.org/stable/modules/clustering.html#k-means

---

8 References

Insights for Professionals. (2019, February 26). *5 Niche Programming Languages (And Why They're Underrated).* https://www.insightsforprofessionals.com/it/software/niche-programming-languages

Nagar, Akanksha. (2020, January 26). *K-means Clustering — Everything you need to know.* Medium. https://medium.com/analytics-vidhya/k-means-clustering-everything-you-need-to-know-175dd01766d5#5ac5

Perceptive Analytics. (2017, August 7). *Exploring Assumptions of K-means Clustering using R.* R-bloggers. https://www.r-bloggers.com/2017/08/exploring-assumptions-of-k-means-clustering-using-r/

Al-Masri, Anas. (2019, May 14). *How Does k-Means Clustering in Machine Learning Work?* Towards Data Science. https://towardsdatascience.com/how-does-k-means-clustering-in-machine-learning-work-fdaaaf5acfa0