Matthew Blessing
*Student ID: **000507424***
OFM3 - Task 2, D212 - Data Mining II

OFM 3 - Performance Assessment Task 2: Dimensionality Reduction Methods

Part I: Research Question

A1.
For this analysis, we will be attempting to utilize a principal component analysis (PCA) in order to determine the principal components within the 'Churn' dataset that impact customer churn.

A2.
The goal of the analysis is to successfully implement PCA on a set of continuous feature variables within our dataset in order to specifically identify any principal components that may impact customer churn.

Part II: Method Justification

B1.
PCA works by reducing the dimensionality (the variables/number of features within the dataset) of different variables that are correlated with each other, heavily or lightly, while retaining the variance already present in the dataset. PCA transforms the existing variables into a new set of variables, the principal components, that are orthogonal and ordered so that they retain the variance of the original variables. These principal components are ordered from most variance to least variance, with the first principal component retaining the maximum variation present in the original components of the dataset (Principal Component Analysis Tutorial, 2016).

We expect that our PCA will identify the principal components within our dataset, along with the explained variance for each component.

B2.
One assumption of principal component analysis is that PCA makes the assumption that components with large variance correspond to interesting dynamics, whereas lower variance components correspond to noise and are deemed uninteresting (CSE 564 - Visualization, n.d.).

Part III: Data Preparation

C1.
In preparing our dataset for PCA, we first want to determine the continuous variables we will use to determine the principal components that impact customer churn. We can utilize Pandas .info() function to return the list of features in our dataset, along with their data types and counts:

```
In [26]: #gets the data type and count of all variables in the data set
         df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 50 columns):
CaseOrder               10000 non-null int64
Customer_id             10000 non-null object
Interaction             10000 non-null object
UID                     10000 non-null object
City                    10000 non-null object
State                   10000 non-null object
County                  10000 non-null object
Zip                     10000 non-null int64
Lat                     10000 non-null float64
Lng                     10000 non-null float64
Population              10000 non-null int64
Area                    10000 non-null object
TimeZone                10000 non-null object
Job                     10000 non-null object
Children                10000 non-null int64
Age                     10000 non-null int64
Income                  10000 non-null float64
Marital                 10000 non-null object
Gender                  10000 non-null object
Churn                   10000 non-null object
Outage_sec_perweek      10000 non-null float64
Email                   10000 non-null int64
Contacts                10000 non-null int64
Yearly_equip_failure    10000 non-null int64
Techie                  10000 non-null object
Contract                10000 non-null object
Port_modem              10000 non-null object
Tablet                  10000 non-null object
InternetService         10000 non-null object
Phone                   10000 non-null object
Multiple                10000 non-null object
OnlineSecurity          10000 non-null object
OnlineBackup            10000 non-null object
DeviceProtection        10000 non-null object
TechSupport             10000 non-null object
StreamingTV             10000 non-null object
StreamingMovies         10000 non-null object
PaperlessBilling        10000 non-null object
PaymentMethod           10000 non-null object
Tenure                  10000 non-null float64
MonthlyCharge           10000 non-null float64
Bandwidth_GB_Year       10000 non-null float64
Item1                   10000 non-null int64
Item2                   10000 non-null int64
Item3                   10000 non-null int64
Item4                   10000 non-null int64
Item5                   10000 non-null int64
Item6                   10000 non-null int64
Item7                   10000 non-null int64
Item8                   10000 non-null int64
dtypes: float64(7), int64(16), object(27)
memory usage: 3.8+ MB
```

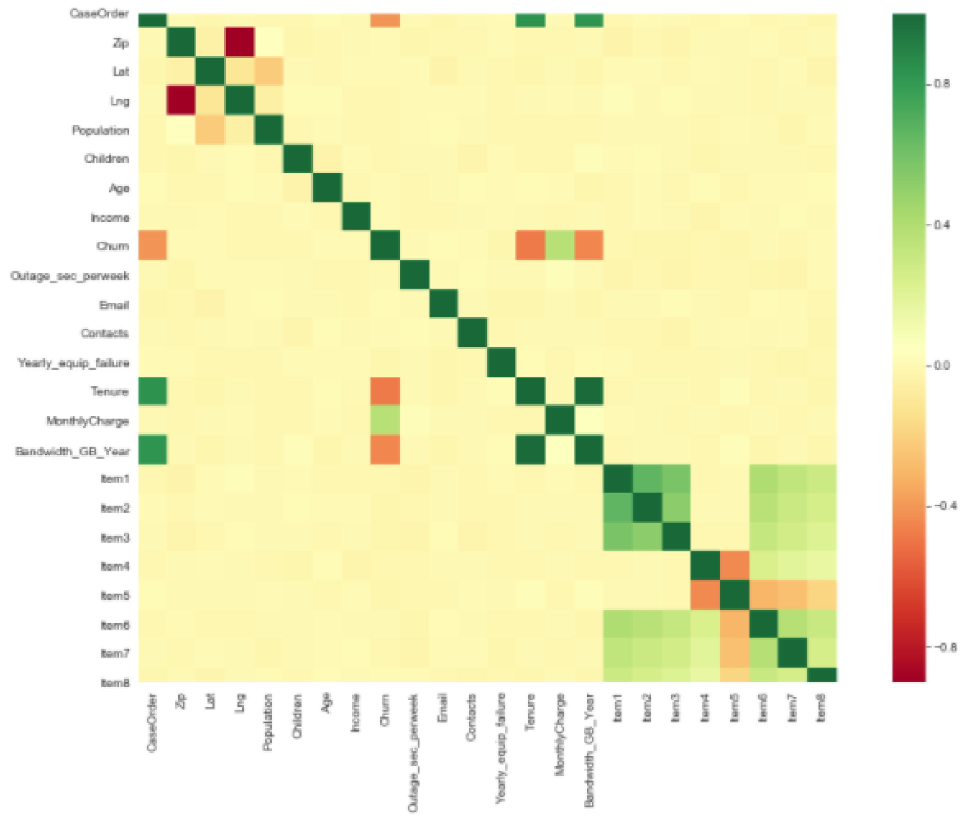
We now have a list of features and their data types from our dataset. We can identify the following continuous variables that we will use as predictor variables for our PCA:

- Age
- Bandwidth_GB_Year
- Children
- Contacts
- Email
- Income
- MonthlyCharge
- Outage_sec_perweek
- Tenure
- Yearly_equip_failure
- Zip

Because we are trying to determine which principal components impact customer churn, our dependent variable will be 'Churn'.

As an additional step to look for multicollinearity within our variables, we will utilize a correlation heatmap to look at the positive and negative correlations within our data:

```
In [40]: plt.figure(figsize=(16, 10))
         sns.heatmap(df.corr(), square=True, cmap='RdYlGn');
```



As we see in the above heatmap, it appears that the only variables that have any correlation with 'Churn' are 'Tenure', 'Bandwidth_GB_Year', and 'MonthlyCharge'. These correlations are not particularly strong, but they do exist. Luckily, it does not appear we have a strong multicollinearity issue with our selected variables.

C2.

PCA requires standardized variables in order to be accurate. Before we can standardize our variables, we need to perform some steps.

First, since 'Churn' is currently a string variable with 'Yes' and 'No' as observations, we need to change these values to be binary values with a 1 = Yes and 0 = No so that the PCA model can read the data. We can do this quickly utilizing Pandas .replace() function:

df.Churn.replace(('Yes', 'No'), (1, 0), inplace=True)

Next, we split the selected continuous variables into their own data frame:

df2 = df[['Age', 'Bandwidth_GB_Year', 'Children','Contacts', 'Churn', 'Email', 'Income', 'MonthlyCharge', 'Outage_sec_perweek', 'Tenure', 'Yearly_equip_failure', 'Zip']]

We then check the top rows of our new data frame:

```
df2.head()
```

We then need to define our features and our target variables:

```
x = df2[['Age','Bandwidth_GB_Year','Children','Contacts','Email', 'Income', 'MonthlyCharge',
'Outage_sec_perweek', 'Tenure', 'Yearly_equip_failure', 'Zip']]

y = df2['Churn']
```

Now we are ready to create our pipeline and standardize our data:

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

scaler = StandardScaler()
pca = PCA()
pipeline = make_pipeline(scaler, pca)
X = pipeline.fit_transform(x)
```

C2.
Please see the attached 'D212_Task2_Dataset.csv'.

Part IV: Analysis

D1.
After creating our pipeline to standardize our data, we print the covariance matrix of our principal components:

```
print('Eigenvectors \n%s' %pca.get_covariance())
print('\nEigenvalues \n%s' %pca.explained_variance_)
```

```
Eigenvectors
[[ 1.00010001 -0.01472512 -0.02973451  0.01506913  0.00158808 -0.00409101
   0.01072958 -0.00804752  0.01698097  0.00857821 -0.0081361 ]
 [-0.01472512  1.00010001  0.02558738  0.00329905 -0.01458061  0.00367392
   0.06041247  0.00417608  0.99159435  0.0120349  -0.00252745]
 [-0.02973451  0.02558738  1.00010001 -0.02077811  0.00447925  0.00994335
  -0.00978238  0.00188944 -0.00509183  0.00732132 -0.01720677]
 [ 0.01506913  0.00329905 -0.02077811  1.00010001  0.00304067  0.00123332
   0.00425907  0.01509319  0.00282037 -0.00603285 -0.00471999]
 [ 0.00158808 -0.01458061  0.00447925  0.00304067  1.00010001 -0.00926842
   0.00199675  0.00399413 -0.01446932 -0.01635598 -0.0078608 ]
```

```
[-0.00409101  0.00367392  0.00994335  0.00123332 -0.00926842  1.00010001
 -0.00301427 -0.01001155  0.00211458  0.00542382  0.00294682]
[ 0.01072958  0.06041247 -0.00978238  0.00425907  0.00199675 -0.00301427
  1.00010001  0.02049812 -0.00333714 -0.00717299 -0.00871765]
[-0.00804752  0.00417608  0.00188944  0.01509319  0.00399413 -0.01001155
  0.02049812  1.00010001  0.00293225  0.00290902 -0.01152155]
[ 0.01698097  0.99159435 -0.00509183  0.00282037 -0.01446932  0.00211458
 -0.00333714  0.00293225  1.00010001  0.01243615 -0.00322746]
[ 0.00857821  0.0120349   0.00732132 -0.00603285 -0.01635598  0.00542382
 -0.00717299  0.00290902  0.01243615  1.00010001  0.01104802]
[-0.0081361  -0.00252745 -0.01720677 -0.00471999 -0.0078608   0.00294682
 -0.00871765 -0.01152155 -0.00322746  0.01104802  1.00010001]]
```

Eigenvalues
**[1.99435251 1.05401359 1.03762842 1.0126684 1.00302824 0.99851856
 0.99369936 0.98525965 0.96305395 0.95341089 0.00546654]**

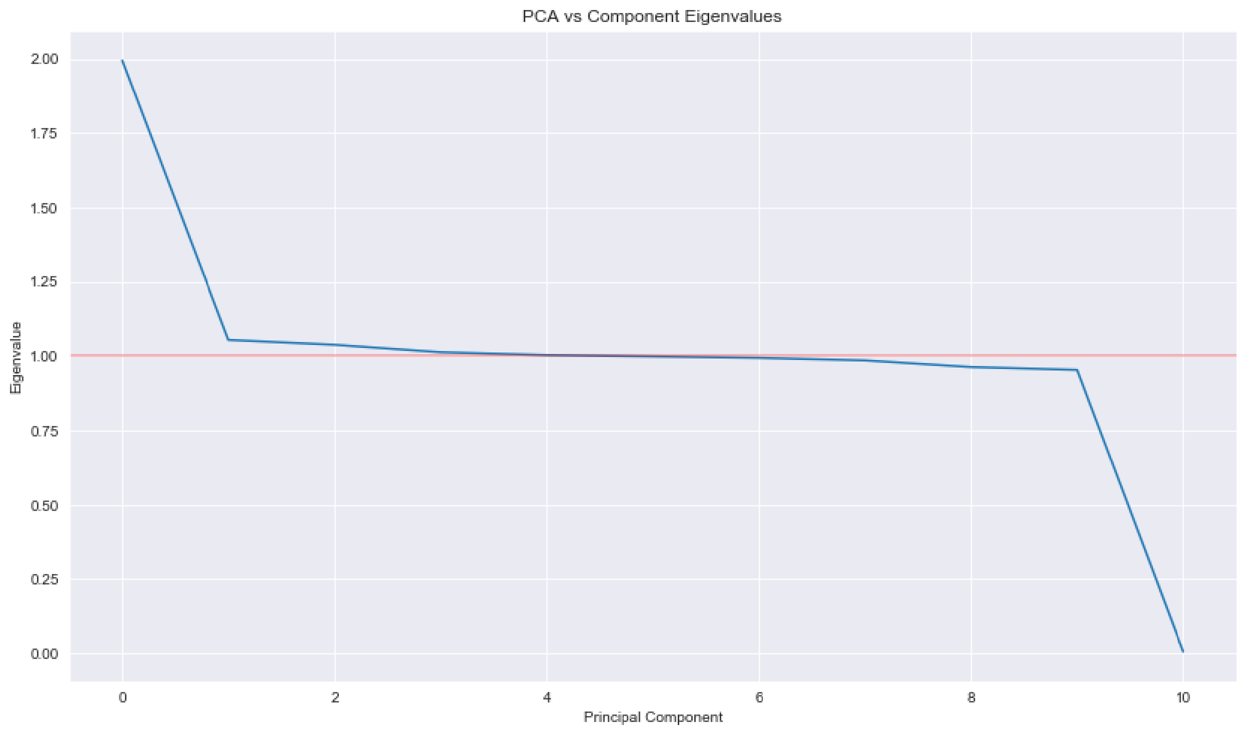
D2.

After retrieving the covariance matrix of all of our principal components, we create a scree plot to plot our principal components compared to their eigenvalues:

```
def scree_plot():
  from matplotlib.pyplot import figure, show
  from matplotlib.ticker import MaxNLocator

  ax = figure(figsize=(14,8)).gca()
  ax.plot(explained_variance)
  ax.xaxis.set_major_locator(MaxNLocator(integer=True))
  plt.xlabel('Principal Component')
  plt.ylabel('Eigenvalue')
  plt.axhline(y=1, linewidth=2, color='r', alpha=0.25)
  plt.title('PCA vs Component Eigenvalues')
  show()

scree_plot()
```



We can see from this scree plot that there is a large difference in variance between component 0 and 1, but then the differences in variance slowly decrease until component 9-10. The Kaiser criterion generally states that a component should not be retained unless its eigenvalue is greater than or equal to one (Kaiser Rule - Displayr, n.d.). Based on this rule and our scree plot, we can determine that components 0, 1, 2, 3, and 4 are significant, while components 5-10 are not.

D3.

After determining our number of components, we can utilize the .explained_variance_ function of SciKit-Learn's PCA model to obtain the explained variance for all of our identified components:

```
explained_variance = pca.explained_variance_
explained_variance
```

```
array([1.99435251, 1.05401359, 1.03762842, 1.0126684 , 1.00302824,
   0.99851856, 0.99369936, 0.98525965, 0.96305395, 0.95341089,
   0.00546654])
```

- Principal component 0 has a cumulative explained variance of 1.99435251
- Principal component 1 has a cumulative explained variance of 1.05401359
- Principal component 2 has a cumulative explained variance of 1.03762842
- Principal component 3 has a cumulative explained variance of 1.0126684
- Principal component 4 has a cumulative explained variance of 1.00302824
- Principal component 5 has a cumulative explained variance of 0.99851856
- Principal component 6 has a cumulative explained variance of 0.99369936
- Principal component 7 has a cumulative explained variance of 0.98525965

- Principal component 8 has a cumulative explained variance of 0.96305395
- Principal component 9 has a cumulative explained variance of 0.95341089
- Principal component 10 has a cumulative explained variance of 0.00546654

If we just want to obtain the explained variance of *only* our five principal components that have a eigenvalue greater than or equal to 1, we can obtain those with the following code that gives us just the first five principal components' explained variances:

```
explained_variance_pca = pca.explained_variance_[:5]
explained_variance_pca
```

```
[1.99435251 1.05401359 1.03762842 1.0126684  1.00302824]
```

- Principal component 0 has a cumulative explained variance of 1.99435251
- Principal component 1 has a cumulative explained variance of 1.05401359
- Principal component 2 has a cumulative explained variance of 1.03762842
- Principal component 3 has a cumulative explained variance of 1.0126684
- Principal component 4 has a cumulative explained variance of 1.00302824

D4.

To determine the total variance for our PCA, with all 10 of our identified principal components, we can simply sum the explained variance:

```
sum(explained_variance)
```

```
11.001100110010995
```

To determine the total variance of only our principal components with eigenvalues greater than or equal to 1, we can sum those by summing explained_variance_pca that we defined in the step above:

```
sum(explained_variance_pca)
```

```
6.101691171613005
```

As we can see, the total explained variance for all 10 principal components is 11.0, while the total explained variance for only our five principal components with eigenvalues greater than or equal to 1 is 6.10.

D5.

Our PCA has determined that there are five principal components with eigenvalues greater than or equal to 1 and are responsible for a large amount of variance within our dataset.

We can utilize a covariance matrix to sum the cumulative explained variance ratio that will inform us of how much variance each principal component accounts for:

```
covar_matrix = PCA(n_components = 5)
```

```
covar_matrix.fit(x)
variance = covar_matrix.explained_variance_ratio_ #calculate variance ratios

total_var = np.cumsum(covar_matrix.explained_variance_ratio_)
print(total_var)

[0.5104989  0.99693305 0.99999852 0.9999997  0.99999998]
```

As we can see above, the first feature explains 51% of the variance, the first two features explain 99.6% of the variance, the first three features explain 99% of the variance, and this continues with the first four and first five features increasingly explaining closer to 100% of the variance. This means that our first two principal components explain most, if not all of the variance. This means we likely could have selected only two principal components earlier, despite having five with an eigenvalue of greater than 1. As a result of our covariance matrix showing that nearly all the variance in the data can be explained with these principal components, we could likely trace back our principal components and explore them further in order to determine the exact causes of the variance, which will allow us to get a more insightful look at what leads to customer churn.

The recommendation to stakeholders within the business would be to explore these primary components in order to determine the differences between them and explore this difference in variance. By exploring these components, the business can gain valuable insight into common factors that cause customers to leave the service.

References

Advantages of Learning Python for Data Science. (2018, March 16). BSD MAG. Retrieved October 14, 2021, from https://bsdmag.org/advantages-of-learning-python-for-data-science/

CSE 564 - Visualization. (n.d.). Stony Brook University. Retrieved October 25, 2021, from https://www3.cs.stonybrook.edu/~mueller/teaching/cse564/

Kaiser Rule - Displayr. (n.d.). Displayr. Retrieved October 25, 2021, from https://docs.displayr.com/wiki/Kaiser_Rule

Principal Component Analysis Tutorial. (2016, March 2). ProjectPro.Io. Retrieved October 25, 2021, from https://www.projectpro.io/data-science-in-python-tutorial/principal-component-analysis-tutorial

Project Jupyter. (n.d.). Project Jupyter. Retrieved October 14, 2021, from https://jupyter.org/

SciKit-Learn: machine learning in Python — scikit-learn 1.0 documentation. (n.d.). SciKit-Learn. Retrieved October 14, 2021, from https://scikit-learn.org/stable/

Tavares, E. (2017, February 10). *Principle Component Analysis (PCA) with Scikit-Learn - Python*. GitHub. Retrieved October 25, 2021, from https://etav.github.io/python/scikit_pca.html