## Part I: Research Question

### A1. Proposal of Question

With the huge amount of data that businesses are bombarded with, they must use those data to help solve their real-world organizational needs. Therefore, an important question to ask is, can we sift through all those data and find what variables or features are important to our business needs?

### A2. Defined Goals

To understand our customers and help further our business needs, we need to be able to find and implement a solution to decrease customer churn. As we know, it costs ten times more to acquire new customers than retain old ones and the industry's annual churn rate hovers around twenty-five percent. Therefore, we need to ensure our stakeholders that we can help mitigate this problem by finding what variables affect customer churn within the dataset.

## Part II: Method Justification

### B1. Explanation of PCA

PCA is a statistical procedure to convert observations of possibly correlated features to principal components such that (Mahapatra, 2021):

- They are uncorrelated with each other.
- They are linear combinations of variables.
- They help in capturing the maximum information in the data set.

For our analysis, the steps to implement PCA Analysis are:

1. Importing the necessary library
2. Load and clean the dataset
3. Standardize the features using the sklearn library
4. Perform PCA Analysis using the sklearn library to:
    a. Obtain Eigenvalues and Eigenvectors

b. Plot a Scree Plot to understand the Number of Components that best fits the analysis

c. Find the smallest Number of Components to best explain the variance within the data

d. Compare that with the most important Number of Components that has an Eigenvalue greater than 1 using the Kaiser Rule

e. Perform machine learning algorithms such as K-Nearest Neighbor etc. using the information obtained from the PCA analysis.

The Principal Component Analysis analyzes the dataset by reducing the dimensionality of large data sets by transforming a large set of variables into smaller ones that still contain most of the information in it (Jaadi, 2021). The expected outcome is that we will have fewer features that explain a greater amount of the variances within the dataset.

**B2. PCA Assumption**

PCA assumes that the principal component with high variance must be paid attention to and the PCs with lower variance are disregarded as noise (Vadapalli, 2022). That is, we want to ensure that we can pick the best feature, using PCA, from a dataset that has many variables while reducing as many unimportant variables as we can.

## Part III: Data Preparation

**C1. Continuous Dataset Variable**

The continuous variables chosen for our dataset are 'Children', 'Age', 'Income', 'Outage_sec_perweek', 'Yearly_equip_failure' , 'Bandwidth_GB_Year', 'Timely_Response', 'Timely_Fixes', 'Timely_Replacement', 'Reliability', 'Options', 'Respectable_Response', 'Respectable_Response', 'Courteous_Exchange', 'Evidence_of_active_listening', 'Sum_services', 'Dummy_Suburban', 'Dummy_Urban', 'Dummy_Part Time', 'Dummy_Retired', 'Dummy_Student', 'Dummy_Unemployed', 'Dummy_One year', and 'Dummy_Two Year'. These are all continuous variables or categorical variables that were changed into dummy continuous variables so that we can perform PCA analysis on the dataset. In total, we have 24 continuous variables. A copy of the variables was provided with the code below.

| | Children | Age | Income | Outage_sec_perweek | Yearly_equip_failure | MonthlyCharge | Bandwidth_GB_Year |
|---|---|---|---|---|---|---|---|
| **0** | 1.0 | 68.0 | 28561.990000 | 6.972566 | 1 | 171.449762 | 904.536110 |
| **1** | 1.0 | 27.0 | 21704.770000 | 12.014541 | 1 | 242.948015 | 800.982766 |
| **2** | 4.0 | 50.0 | 39936.762226 | 10.245616 | 1 | 159.440398 | 2054.706961 |
| **3** | 1.0 | 48.0 | 18925.230000 | 15.206193 | 0 | 120.249493 | 2164.579412 |
| **4** | 0.0 | 83.0 | 40074.190000 | 8.960316 | 1 | 150.761216 | 271.493436 |

| Timely_Response | Timely_Fixes | Timely_Replacement | ... | Evidence_of_active_listening | Sum_services | Dummy_Suburban |
|---|---|---|---|---|---|---|
| 5 | 5 | 5 | ... | 4 | 5.0 | 0 |
| 3 | 4 | 3 | ... | 4 | 6.0 | 0 |
| 4 | 4 | 2 | ... | 3 | 4.0 | 0 |
| 4 | 4 | 4 | ... | 3 | 4.0 | 1 |
| 4 | 4 | 4 | ... | 5 | 3.0 | 1 |

| Dummy_Urban | Dummy_Part Time | Dummy_Retired | Dummy_Student | Dummy_Unemployed | Dummy_One year | Dummy_Two Year |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |

**C2. Standardization of Dataset Variables**

The data was standardized first by creating dummy variables for the categorical variables, Contract, Area, Employment, and Churn. It was then scaled and standardized using sklearn. A clean dataset was later exported once we were done with the analysis. A copy of the code has been provided below.

In [75]:
```
#Create Dummy Variables for Categorical Variables
d_var=pd.get_dummies(df[['Area','Employment','Contract']], prefix="Dummy", drop_first=True)
d_var
```

Out[75]:

| | Dummy_Suburban | Dummy_Urban | Dummy_Part Time | Dummy_Retired | Dummy_Student | Dummy_Unemployed | Dummy_O ye |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 1 | 0 | 0 | 0 | |
| **1** | 0 | 1 | 0 | 1 | 0 | 0 | |
| **2** | 0 | 1 | 0 | 0 | 1 | 0 | |
| **3** | 1 | 0 | 0 | 1 | 0 | 0 | |
| **4** | 1 | 0 | 0 | 0 | 1 | 0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **9995** | 0 | 0 | 0 | 1 | 0 | 0 | |
| **9996** | 0 | 0 | 1 | 0 | 0 | 0 | |
| **9997** | 0 | 0 | 0 | 0 | 0 | 0 | |

In [76]:
```
#Add the dummy variable to a new df. Drop the rows that has null values and create copy of the data fr
df_model=pd.concat([df_cln,d_var], axis=1)
df_model=df_model.dropna()
df_model=df_model.drop(['Area','Employment','Contract','Churn'], axis=1)
df_model
```

Out[76]:

| | Children | Age | Income | Outage_sec_perweek | Yearly_equip_failure | MonthlyCharge | Bandwidth_GB_Year | Timely_Res |
|---|---|---|---|---|---|---|---|---|
| **0** | 1.0 | 68.0 | 28561.990000 | 6.972566 | 1 | 171.449762 | 904.536110 | |
| **1** | 1.0 | 27.0 | 21704.770000 | 12.014541 | 1 | 242.948015 | 800.982766 | |
| **2** | 4.0 | 50.0 | 39936.762226 | 10.245616 | 1 | 159.440398 | 2054.706961 | |
| **3** | 1.0 | 48.0 | 18925.230000 | 15.206193 | 0 | 120.249493 | 2164.579412 | |
| **4** | 0.0 | 83.0 | 40074.190000 | 8.960316 | 1 | 150.761216 | 271.493436 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **9995** | 3.0 | 53.0 | 55723.740000 | 9.265392 | 0 | 159.828800 | 6511.253000 | |
| **9996** | 4.0 | 48.0 | 39936.762226 | 8.115849 | 0 | 208.856400 | 5695.952000 | |
| **9997** | 1.0 | 53.0 | 39936.762226 | 4.837696 | 0 | 168.220900 | 4159.306000 | |
| **9998** | 1.0 | 39.0 | 16667.580000 | 12.076460 | 0 | 252.628600 | 6468.457000 | |
| **9999** | 1.0 | 28.0 | 39936.762226 | 12.641760 | 0 | 218.371000 | 5857.586000 | |

10000 rows × 24 columns

In [77]:
```
#Standardize the data.
scaler = StandardScaler()
scaler.fit(df_model)
```

Out[77]: StandardScaler()

In [78]:
```
#Transform Data
scaled_data = scaler.transform(df_model)
```

```
In [93]: #Export cleaned dataset to a new CSV file
         df_model.to_csv('D212_Cleaned_Dataset.csv')
```

## Part IV: Analysis

### D1. Principal Components

A Principal Component Analysis was performed using the code below. We made sure to get the Eigenvalue, Eigenvector, and Covariance Matrix. We also plotted the Scree Plot to get a better understanding of our principal components.

```
In [81]: #Find Eigen Value
         pca = decomposition.PCA(n_components = 24)
         pca_X = pca.fit(scaled_data)
         variance_retained_ratio = pca_X.explained_variance_ratio_.cumsum()
         variance_retained_ratio

Out[81]: array([0.12285691, 0.19776432, 0.26616659, 0.32876732, 0.38286168,
                0.4297291 , 0.47608935, 0.52229065, 0.56566287, 0.60770592,
                0.64952788, 0.69093682, 0.73177813, 0.77151387, 0.80406181,
                0.83338206, 0.86238428, 0.88983752, 0.91449733, 0.93690914,
                0.95785521, 0.9776251 , 0.991136  , 1.        ])
```

```
In [82]: #Print Individual Explain Variance
         print(pca.explained_variance_ratio_)

         [0.12285691 0.07490741 0.06840226 0.06260074 0.05409436 0.04686742
          0.04636025 0.0462013  0.04337221 0.04204305 0.04182196 0.04140894
          0.04084131 0.03973574 0.03254793 0.02932026 0.02900221 0.02745324
          0.02465982 0.0224118  0.02094607 0.01976989 0.01351091 0.008864  ]
```

```
In [83]: #Print Covariance Matrix
         covariance_matrix_Sigma = pca.get_covariance()
         print (covariance_matrix_Sigma.shape)
         covariance_matrix_Sigma

                [-2.89679113e-03,  5.35821784e-03, -5.55511456e-03,
                 -8.74977769e-04, -9.24050845e-03,  7.76004396e-01,
                  6.36248317e-02,  1.48672942e-02,  1.15432107e-02,
                 -5.29651675e-03,  1.75767542e-03, -2.36438901e-03,
                  4.68385906e-03, -1.81291109e-03,  1.22624667e-02,
                  1.00010001e+00, -7.18552977e-03,  5.75930874e-03,
                 -1.63810871e-03,  1.05554670e-02, -5.59720688e-05,
                 -5.44379298e-03,  1.40431416e-02,  5.78000038e-03],
                [-1.33001846e-03, -6.23405809e-05,  2.59256573e-03,
                  3.91982807e-03,  6.09672817e-03, -2.05195900e-03,
                 -2.74192206e-03, -3.72066329e-03,  2.03543223e-03,
                 -8.35129568e-03,  1.72845132e-03, -5.63475177e-03,
                 -3.06912232e-03,  2.10479180e-03,  7.67049241e-04,
                 -7.18552977e-03,  1.00010001e+00, -5.00761394e-01,
                  6.48429294e-03, -9.00370577e-04, -7.57959475e-03,
                  3.83860639e-03, -8.49432851e-03,  8.34111688e-03],
                [-2.70845700e-04,  1.45096173e-02,  4.48594935e-03,
                 -1.41652487e-02, -8.72686585e-03,  2.43051557e-03,
                 -1.14656704e-02,  4.31239656e-04, -2.35269169e-03,
                 -1.61574787e-02, -8.31468045e-03,  8.10305158e-03,
```

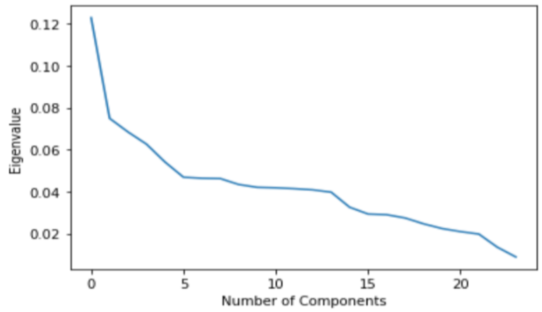
```
In [92]: #Print Eigen Values and Vector
eig_vals, eig_vecs = np.linalg.eig(covariance_matrix_Sigma)
print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)
```

```
Eigenvectors
[[ 2.13323476e-04 -2.94445031e-03  1.04892554e-02 -1.20094324e-03
  -5.46848404e-04  8.25649965e-03 -1.31650942e-03 -1.82630149e-02
   1.06467613e-02  1.39989601e-02 -1.50241864e-02  2.52824213e-02
  -7.47565027e-02 -6.14066392e-03  3.85119901e-02  1.16205691e-01
  -1.11782290e-01  2.00658563e-02  2.68952330e-01 -5.00240510e-01
  -6.02576905e-01  3.88919184e-01 -2.01559966e-01 -2.95349501e-01]
 [ 4.99586893e-03  6.51359389e-03 -1.80207512e-02  2.39664174e-02
   4.09124732e-03 -1.64347951e-02 -1.08297492e-02 -7.85309581e-04
   2.13162409e-02 -1.72722090e-02 -1.29808548e-03  1.60484272e-02
   1.38224195e-02 -1.02574661e-02 -5.85966524e-02 -4.23365871e-02
  -3.35144479e-02  2.77414880e-02  5.35021015e-01  5.37047449e-01
  -4.53010239e-02  2.99502672e-01  4.92709347e-01 -2.84636202e-01]
 [-9.02063364e-04 -8.78445598e-03  2.61558006e-02  1.01163224e-04
  -4.85500350e-04 -4.88757150e-03 -1.92209441e-02 -6.00330758e-03
   1.54115704e-02 -2.54054332e-03 -2.32408078e-03  3.86897947e-02
   2.76666327e-02  6.22729988e-02 -2.41735493e-03  1.07345957e-01
   2.13003074e-02 -6.49582519e-02 -1.57410243e-01 -1.32388154e-01
   3.67929920e-01 -1.88663566e-01 -2.75796792e-02 -8.73031926e-01]
```

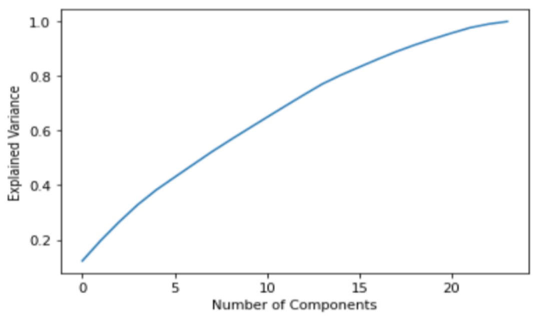
**D2. Identification of Total Number of Components**

The number of components that we started with was 24 which was then reduced to our current total which is 11. These components were chosen using the Kaiser criterion, that is, we only kept components with an Eigenvalue greater than 1. A scree plot was also used to understand the principal components. See the code below.

In [84]:
```
#Scree Plot
plt.plot(pca.explained_variance_ratio_)
plt.xlabel('Number of Components')
plt.ylabel('Eigenvalue')
plt.show()
```




In [85]:
```
#Plot Explained Variances
plt.plot(variance_retained_ratio)
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance')
plt.show()
```




In [90]:
```
#Selecting the fewest components
for pc, var in zip(df_model, (variance_retained_ratio)):
    print(pc, var)
```

```
Children 0.12285691015056698
Age 0.19776432159078824
Income 0.26616658650640324
Outage_sec_perweek 0.32876732211047793
Yearly_equip_failure 0.3828616803137391
MonthlyCharge 0.4297291044944567
Bandwidth_GB_Year 0.4760893500256116
Timely_Response 0.5222906549663214
Timely_Fixes 0.5656628685980641
Timely_Replacement 0.60770592334573
Reliability 0.649527883699912
Options 0.690936824045443
Respectable_Response 0.731778133457208
Courteous_Exchange 0.7715138734106851
Evidence_of_active_listening 0.8040618058411168
Sum_services 0.8333820643817287
Dummy_Suburban 0.8623842776891003
Dummy_Urban 0.8898375155786146
Dummy_Part Time 0.914497333714448
```

In [91]:
```
# List importance of components
def pca_summary(pca, standardized_data, out=True):
 names = ['PC ' + str(i) for i in range(1, len(pca.explained_variance_ratio_) + 1)]
 a = list(np.std(pca.transform(standardized_data), axis = 0))
 b = list(pca.explained_variance_ratio_)
 c = [np.sum(pca.explained_variance_ratio_[:i]) for i in range(1, len(pca.explained_variance_ratio_) + 1)]
 columns = pd.MultiIndex.from_tuples([('standard_deviation', 'Standard Deviation'),
 ('proportion_of_variation', 'Proportion of Variation'),
 ('cumulative_proportion', 'Cumulative Proportion')])
 summary = pd.DataFrame(zip(a, b, c), index=names, columns=columns)
 if out:
     print('Component importance:')
 return summary
# Display summary
summary = pca_summary(pca, scaled_data)
summary.standard_deviation**2
```

```
Component importance:
```

Out[91]:

| | Standard Deviation |
|---|---|
| PC 1 | 2.948566 |
| PC 2 | 1.797778 |
| PC 3 | 1.641654 |
| PC 4 | 1.502418 |
| PC 5 | 1.298265 |
| PC 6 | 1.124818 |
| PC 7 | 1.112646 |
| PC 8 | 1.108831 |
| PC 9 | 1.040933 |
| PC 10 | 1.009033 |
| PC 11 | 1.003727 |
| PC 12 | 0.993815 |
| PC 13 | 0.980191 |
| PC 14 | 0.953658 |
| PC 15 | 0.781150 |
| PC 16 | 0.703686 |
| PC 17 | 0.696053 |
| PC 18 | 0.658878 |
| PC 19 | 0.591836 |
| PC 20 | 0.537883 |
| PC 21 | 0.502706 |
| PC 22 | 0.474477 |
| PC 23 | 0.324262 |
| PC 24 | 0.212736 |

## D3. Total Variance of Components

Each of the component's variance was identified using the code below.

In [81]:
```
#Find Eigen Value
pca = decomposition.PCA(n_components = 24)
pca_X = pca.fit(scaled_data)
variance_retained_ratio = pca_X.explained_variance_ratio_.cumsum()
variance_retained_ratio
```

Out[81]:
```
array([0.12285691, 0.19776432, 0.26616659, 0.32876732, 0.38286168,
       0.4297291 , 0.47608935, 0.52229065, 0.56566287, 0.60770592,
       0.64952788, 0.69093682, 0.73177813, 0.77151387, 0.80406181,
       0.83338206, 0.86238428, 0.88983752, 0.91449733, 0.93690914,
       0.95785521, 0.9776251 , 0.991136  , 1.        ])
```

In [82]:
```
#Print Individual Explain Variance
print(pca.explained_variance_ratio_)
```

```
[0.12285691 0.07490741 0.06840226 0.06260074 0.05409436 0.04686742
 0.04636025 0.0462013  0.04337221 0.04204305 0.04182196 0.04140894
 0.04084131 0.03973574 0.03254793 0.02932026 0.02900221 0.02745324
 0.02465982 0.0224118  0.02094607 0.01976989 0.01351091 0.008864  ]
```

In [91]:
```python
# List importance of components
def pca_summary(pca, standardized_data, out=True):
 names = ['PC ' + str(i) for i in range(1, len(pca.explained_variance_ratio_) + 1)]
 a = list(np.std(pca.transform(standardized_data), axis = 0))
 b = list(pca.explained_variance_ratio_)
 c = [np.sum(pca.explained_variance_ratio_[:i]) for i in range(1, len(pca.explained_variance_ratio_) + 1)]
 columns = pd.MultiIndex.from_tuples([('standard_deviation', 'Standard Deviation'),
 ('proportion_of_variation', 'Proportion of Variation'),
 ('cumulative_proportion', 'Cumulative Proportion')])
 summary = pd.DataFrame(zip(a, b, c), index=names, columns=columns)
 if out:
     print('Component importance:')
 return summary
# Display summary
summary = pca_summary(pca, scaled_data)
summary.standard_deviation**2
```

Component importance:

Out[91]:

| | Standard Deviation |
|---|---|
| PC 1 | 2.948566 |
| PC 2 | 1.797778 |
| PC 3 | 1.641654 |
| PC 4 | 1.502418 |
| PC 5 | 1.298265 |
| PC 6 | 1.124818 |
| PC 7 | 1.112646 |
| PC 8 | 1.108831 |
| PC 9 | 1.040933 |
| PC 10 | 1.009033 |
| PC 11 | 1.003727 |
| PC 12 | 0.993815 |
| PC 13 | 0.980191 |
| PC 14 | 0.953658 |
| PC 15 | 0.781150 |
| PC 16 | 0.703686 |
| PC 17 | 0.696053 |
| PC 18 | 0.658878 |
| PC 19 | 0.591836 |
| PC 20 | 0.537883 |
| PC 21 | 0.502706 |
| PC 22 | 0.474477 |
| PC 23 | 0.324262 |
| PC 24 | 0.212736 |

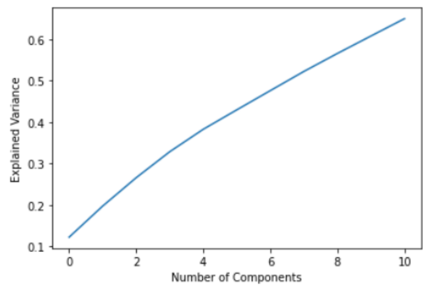
## D4. Total Variance Captured by Components

We reduced the number of principal components to 11 of which the total number of variances captured was about 65%. See the code below for more details.

```
In [132]: pca_b = decomposition.PCA(n_components = 11)
          pca_Z = pca_b.fit(scaled_data)
          variance_retained_ratio1 = pca_Z.explained_variance_ratio_.cumsum()
          variance_retained_ratio1

Out[132]: array([0.12285691, 0.19776419, 0.26616613, 0.32876576, 0.38285997,
                 0.42969598, 0.47605414, 0.52222333, 0.56558915, 0.60762861,
                 0.64936484])
```

```
In [133]: #Plot Total Explained Variance of Reduced principal components
          plt.plot(variance_retained_ratio1)
          plt.xlabel('Number of Components')
          plt.ylabel('Explained Variance')
          plt.show()
```



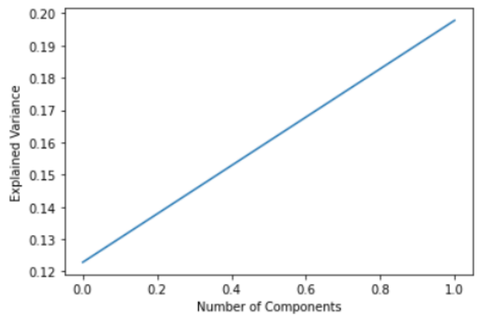

**D5. Summary of Data Analysis**

Using this Principal Component Analysis, we concluded that it requires 11 principal components to explain just 65% of the variances which isn't too great considering that we want to reduce our principal components as much as possible. We did perform another PCA analysis using just 2 components but that performed poorly with it explaining just 20% of the variance (see visual below). We could have imputed other categorical variables into dummy variables which could've increased our explained variances but as it stands, we can only explain 65% of the data using 11 principal components (reduced using the Kaiser rule) and then just 20% when reduced to 2.

```
In [136]: pca_c = decomposition.PCA(n_components = 2)
          pca_Zx = pca_c.fit(scaled_data)
          variance_retained_ratio2 = pca_Zx.explained_variance_ratio_.cumsum()
          variance_retained_ratio2

Out[136]: array([0.12285691, 0.19776386])
```

```
In [137]: #Plot Total Explained Variance of Reduced principal components
          plt.plot(variance_retained_ratio2)
          plt.xlabel('Number of Components')
          plt.ylabel('Explained Variance')
          plt.show()
```




**Part V. Attachments**

## E. Sources for Third-Party Code


*Dimensional Reduction| Principal Component Analysis*. (2019, February 2). [Video]. YouTube. https://www.youtube.com/watch?v=OFyyWcw2cyM

GeeksforGeeks. (2022, July 18). *Reduce Data Dimensionality using PCA - Python*. https://www.geeksforgeeks.org/reduce-data-dimentionality-using-pca-python/

Mangale, S. (2021, December 15). *Scree Plot - SANCHITA MANGALE*. Medium. https://sanchitamangale12.medium.com/scree-plot-733ed72c8608

*Part 10 - Dimensionality Reduction - Principal Component Analysis using Python*. (2020, August 10). [Video]. YouTube. https://www.youtube.com/watch?v=TZv46JQulWw

*Principal Component Analysis with Python | District Data Labs*. (2022). District Data Labs: Data Science Consulting and Corporate Training. https://www.districtdatalabs.com/principal-component-analysis-with-python

Singh, I. (2018, June 22). *How many principal components to take (PCA)?* Linkedin. https://www.linkedin.com/pulse/how-many-principal-components-take-pca-inder-preet-singh/

Z. (2021, September 18). *How to Create a Scree Plot in Python (Step-by-Step)*. Statology. https://www.statology.org/scree-plot-python/

#### F. Sources

Jaadi, Z. (2021, April 1). *A Step-by-Step Explanation of Principal Component Analysis (PCA)*. Built In. https://builtin.com/data-science/step-step-explanation-principal-component-analysis

Mahapatra, A. (2021, December 13). *Interpret Principal Component Analysis (PCA) - Towards Data Science*. Medium. https://towardsdatascience.com/interpret-principal-component-analysis-pca-b8b4a4f22ece

Vadapalli, P. (2022, September 22). *PCA in Machine Learning: Assumptions, Steps to Apply & Applications*. UpGrad Blog. https://www.upgrad.com/blog/pca-in-machine-learning/