D212 – DATA MINING II

Task 2: Dimensionality Reduction Methods

WGU - MSDA

Advanced Data Mining Principal Component Analysis with Churn Dataset

Richard Flores

Rflo147@wgu.edu

Student ID: 006771163

Table of Contents

Part I: Research Question

A1. Proposal of Question

As the telecommunications market becomes increasingly competitive with new and improved technologies including free applications like META (Facebook) messenger, Telegram, and TikTok the need for customer retention is becoming critically important.
The question answered in this research project is:

How do we identify customers at risk of churn and what telecom services or features are correlated?

We will be using Principal Component Analysis to analyze the churn customer dataset and identify the principal variables of our customers.

A2. Defined Goal

The goal of the research question is to provide stakeholders direct and actionable insight to create a plan for operations personnel, officers, and managers to increase customer satisfaction through targeted services observed from insights in the dataset and to reduce customer churn and protect long-term profits. In this analysis we will provide numerical calculations to stakeholders sharing an overview of principal components that correlate to customer churn.

Part II: Method Justification

B1. Explanation of PCA

Having a lot of data to analyze is great, and having more data is even better! But when your dataset contains thousands or possibly hundreds of thousands of features, the analysis can quickly become overwhelming and actionable insights can become hard to see. Principal Component Analysis or PCA helps us to focus on the important information by transforming a plethora of records into the idealized set of features (Cheng 2022).

To obtain our goal of trimming the dataset to its idealized set of features we use PCA to create a principal set of components rank ordered by variance. The component with the highest variance is ranked first, the second highest variance is ranked second and so on. We will also focus on selecting components that are uncorrelated, if we choose to include correlated features our analysis will quickly become redundant and the insights less meaningful. Finally, the PCA analysis will focus on selecting only the most crucial components and minimizing the total amount of features selected as selecting too many components cause the model to become overfit.

In order to mathematically calculate PCA we use linear regression to obtain the values of variance (Yiu 2021). Once we collect information about variance, we then rank the components by highest to lowest amount of variance. Once the features are analyzed and ranked, we then choose components with the strongest underlying trends. Finally, we choose components for PCA that contain trends and data which are orthogonal as this represents features that are not correlated.

B2. PCA Assumption

The PCA model is built based on the assumption that we will reduce the total amount of features by following three guidelines. The first guideline is that features selected contain a high amount of variance or features that possess a high amount of potential signal. Second, we select features that are uncorrelated to reduce the chance of multicollinearity and redundancy in our analysis. Third, we select the lowest number of features possible to provide meaningful observation of the target variable and prevent overfitting the model (Yiu 2021).

Part III: Data Preparation

C1. Continuous Dataset Variables

The churn dataset contains many features, but the continuous variables of essence chosen for this analysis are:

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

For the Principal Component Analysis these variables have been chosen for their high variance and uncorrelated nature.

```
# Standard library imports, and Visualization, Statistics, SciKit libraries
import numpy as np
import pandas as pd
from pandas import Series, DataFrame

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import sklearn
from sklearn import datasets
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

# Ignore Warning messages
import warnings
warnings.filterwarnings('ignore')

import matplotlib as mpl
COLOR = 'white'
mpl.rcParams['text.color'] = COLOR
mpl.rcParams['axes.labelcolor'] = COLOR
mpl.rcParams['xtick.color'] = COLOR
mpl.rcParams['ytick.color'] = COLOR
```

```
# Load churn dataset into a Pandas dataframe
churn_df = pd.read_csv('churn_clean.csv', index_col=0)
```

```
# List columns in the dataframe
churn_df.columns
```

```
Index(['Customer_id', 'Interaction', 'UID', 'City', 'State', 'County', 'Zip',
       'Lat', 'Lng', 'Population', 'Area', 'TimeZone', 'Job', 'Children',
       'Age', 'Income', 'Marital', 'Gender', 'Churn', 'Outage_sec_perweek',
       'Email', 'Contacts', 'Yearly_equip_failure', 'Techie', 'Contract',
       'Port_modem', 'Tablet', 'InternetService', 'Phone', 'Multiple',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'PaperlessBilling', 'PaymentMethod',
       'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year', 'Item1', 'Item2',
       'Item3', 'Item4', 'Item5', 'Item6', 'Item7', 'Item8'],
      dtype='object')
```

```python
# Verify the number of records and columns in the dataset
churn_df.shape
```

```
(10000, 49)
```

```python
# Verify headers of imported dataset
churn_df.head()
```

	Customer_id	Interaction	UID	City	State	County	Zip	Lat	Lng	Population	...	MonthlyChar
CaseOrder												
1	K409198	aa90260b-4141-4a24-8e36-b04ce1f4f77b	e885b299883d4f9fb18e39c75155d990	Point Baker	AK	Prince of Wales-Hyder	99927	56.25100	-133.37571	38	...	172.4555
2	S120509	fb76459f-c047-4a9d-8af9-e0f7d4ac2524	f2de8bef964785f41a2959829830fb8a	West Branch	MI	Ogemaw	48661	44.32893	-84.24080	10446	...	242.6325
3	K191035	344d114c-3736-4be5-98f7-c72c281e2d35	f1784cfa9f6d92ae816197eb175d3c71	Yamhill	OR	Yamhill	97148	45.35589	-123.24657	3735	...	159.9475
4	D90850	abfa2b40-2d43-4994-b15a-989b8c79e311	dc8a365077241bb5cd5ccd305136b05e	Del Mar	CA	San Diego	92014	32.96687	-117.24798	13863	...	119.9568
5	K662701	68a861fd-0d20-4e51-a587-8a90407ee574	aabb64a116e83fdc4befc1fbab1663f9	Needville	TX	Fort Bend	77461	29.38012	-95.80673	11352	...	149.9483

5 rows × 49 columns

```python
# Verify dataset info
churn_df.info
```

```
<bound method DataFrame.info of           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
...               ...                                   ...
9996          M324793  45deb5a2-ae04-4518-bf0b-c82db8dbe4a4
9997          D861732  6e96b921-0c09-4993-bbda-a1ac6411061a
9998          I243405  e8307ddf-9a01-4fff-bc59-4742e03fd24f
9999          I641617  3775ccfc-0052-4107-81ae-9657f81ecdf3
10000          T38070  9de5fb6e-bd33-4995-aec8-f01d0172a499

                                        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
...                                     ...          ...   ...
9996       9499fb4de537af195d16d046b79fd20a  Mount Holly    VT
9997       c09a841117fa81b5c8e19afec2760104  Clarksville    TN
9998       9c41f212d1e04dca84445019bbc9b41c     Mobeetie    TX
9999       3e1f269b40c235a1038863ecf6b7a0df   Carrollton    GA
10000      0ea683a03a3cd544aefe8388aab16176  Clarkesville   GA

                          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  ...
...                          ...    ...       ...        ...         ...  ...
9996                     Rutland   5758  43.43391  -72.78734         640  ...
9997                  Montgomery  37042  36.56907  -87.41694       77168  ...
9998                     Wheeler  79061  35.52039 -100.44180         406  ...
9999                     Carroll  30117  33.58016  -85.13241       35575  ...
10000                  Habersham  30523  34.70783  -83.53648       12230  ...
```

```
         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
...                ...               ...   ...    ...    ...    ...   ...
9996        159.979400       6511.252601     3      2      3      3     4
9997        207.481100       5695.951810     4      5      5      4     4
9998        169.974100       4159.305799     4      4      4      4     4
9999        252.624000       6468.456752     4      4      6      4     3
10000       217.484000       5857.586167     2      2      3      3     3

         Item6 Item7  Item8
CaseOrder
1            4     3      4
2            3     4      4
3            3     3      3
4            4     3      3
5            4     4      5
...        ...   ...    ...
9996         3     2      3
9997         5     2      5
9998         4     4      5
9999         3     5      4
10000        3     4      1

[10000 rows x 49 columns]>
```

```
# Describe Churn dataset
churn_df.describe()
```

	Zip	Lat	Lng	Population	Children	Age	Income	Outage_sec_perweek	Email	Contacts	...
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.0000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	...
mean	49153.319600	38.757567	-90.782536	9756.562400	2.0877	53.078400	39806.926771	10.001848	12.016000	0.994200	...
std	27532.196108	5.437389	15.156142	14432.698671	2.1472	20.698882	28199.916702	2.976019	3.025898	0.988466	...
min	601.000000	17.966120	-171.688150	0.000000	0.0000	18.000000	348.670000	0.099747	1.000000	0.000000	...
25%	26292.500000	35.341828	-97.082812	738.000000	0.0000	35.000000	19224.717500	8.018214	10.000000	0.000000	...
50%	48869.500000	39.395800	-87.918800	2910.500000	1.0000	53.000000	33170.605000	10.018560	12.000000	1.000000	...
75%	71866.500000	42.106908	-80.088745	13168.000000	3.0000	71.000000	53246.170000	11.969485	14.000000	2.000000	...
max	99929.000000	70.640660	-65.667850	111850.000000	10.0000	89.000000	258900.700000	21.207230	23.000000	7.000000	...

8 rows × 22 columns

```
# List features available in the dataset
churn_df.dtypes
```

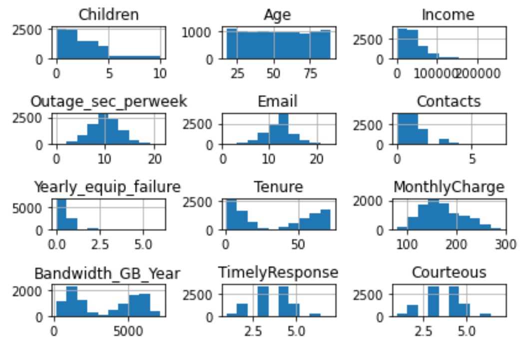
```
Customer_id           object
Interaction           object
UID                   object
City                  object
State                 object
County                object
Zip                    int64
Lat                  float64
Lng                  float64
Population             int64
Area                  object
TimeZone              object
Job                   object
Children               int64
Age                    int64
Income               float64
Marital               object
Gender                object
Churn                 object
```

(DataCamp 2022)

```
Outage_sec_perweek      float64
Email                     int64
Contacts                  int64
Yearly_equip_failure      int64
Techie                   object
Contract                 object
Port_modem               object
Tablet                   object
InternetService          object
Phone                    object
Multiple                 object
OnlineSecurity           object
OnlineBackup             object
DeviceProtection         object
TechSupport              object
StreamingTV              object
StreamingMovies          object
PaperlessBilling         object
PaymentMethod            object
Tenure                  float64
MonthlyCharge           float64
Bandwidth_GB_Year       float64
Item1                     int64
Item2                     int64
Item3                     int64
Item4                     int64
Item5                     int64
Item6                     int64
Item7                     int64
Item8                     int64
dtype: object
```

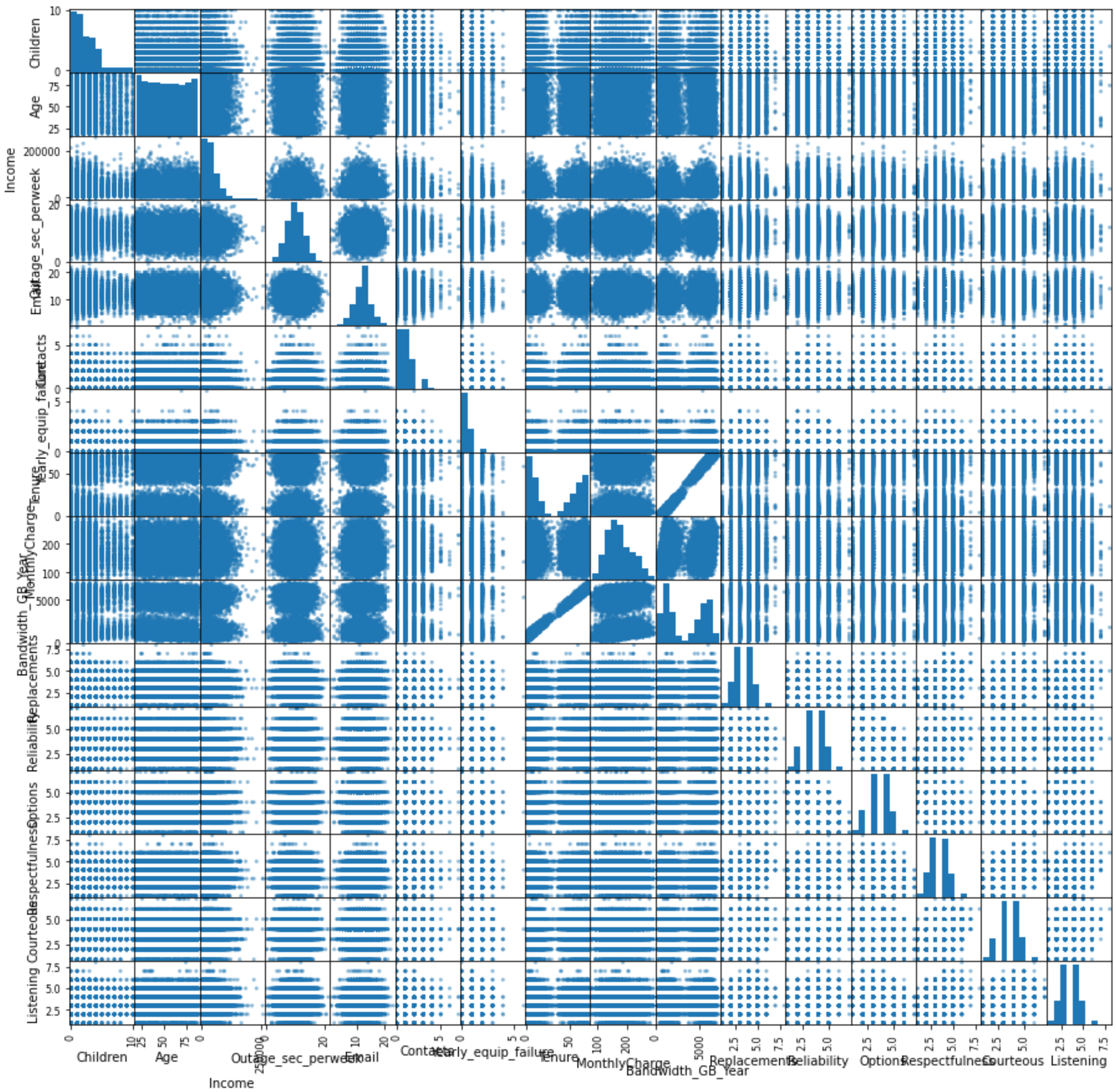
```
# Rename 8 customer survey features to represent descriptions for clarity
churn_df.rename(columns = {'Item1':'TimelyResponse',
                   'Item2':'Fixes',
                    'Item3':'Replacements',
                    'Item4':'Reliability',
                    'Item5':'Options',
                    'Item6':'Respectfulness',
                    'Item7':'Courteous',
                    'Item8':'Listening'},
          inplace=True)
```

```
# Display histograms of continuous & categorical variables
churn_df[['Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts', 'Yearly_equip_failure', 'Tenure',
          'MonthlyCharge', 'Bandwidth_GB_Year', 'TimelyResponse', 'Courteous']].hist()
plt.savefig('classification_pyplot.jpg')
plt.tight_layout()
```



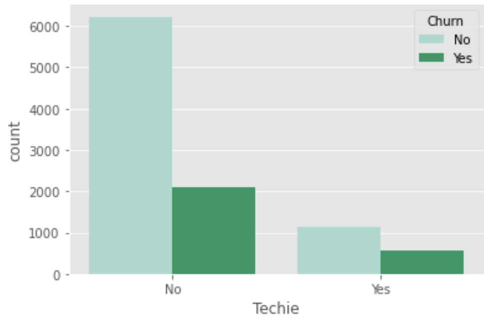
(DataCamp 2022)

```python
# Scatter matrixes of numeric variables for a broad overview of possible relationships.
churn_numeric = churn_df[['Children', 'Age', 'Income', 'Outage_sec_perweek',
                          'Email', 'Contacts','Yearly_equip_failure', 'Tenure',
                          'MonthlyCharge', 'Bandwidth_GB_Year', 'Replacements',
                          'Reliability', 'Options', 'Respectfulness', 'Courteous',
                          'Listening']]

pd.plotting.scatter_matrix(churn_numeric, figsize = [15, 15]);
```

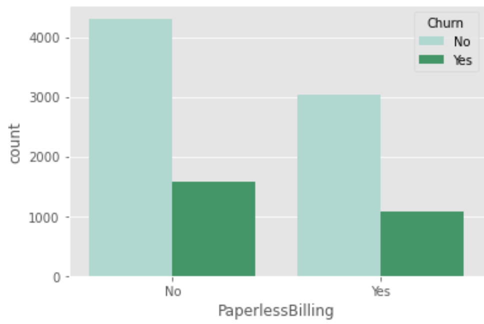


(DataCamp 2022)

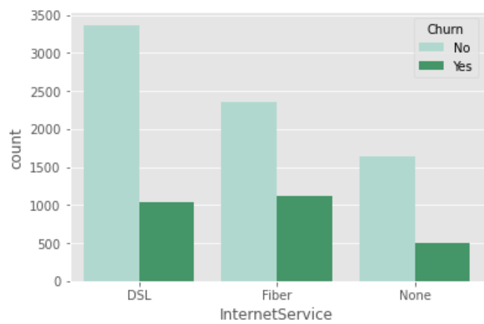
```
# Enable ggplot
plt.style.use('ggplot')

# Countplot to show relationship of binary feature techie and churn
plt.figure()
sns.countplot(x='Techie', hue='Churn', data=churn_df, palette='BuGn')
plt.xticks([0,1], ['No', 'Yes'])
plt.show()
```




```
# Countplot to show relationship of binary feature PaperlessBilling and churn
plt.figure()
sns.countplot(x='PaperlessBilling', hue='Churn', data=churn_df, palette='BuGn')
plt.xticks([0,1], ['No', 'Yes'])
plt.show()
```




```
# Countplot to show relationship of binary feature InternetService and churn
plt.figure()
sns.countplot(x='InternetService', hue='Churn', data=churn_df, palette='BuGn')
plt.xticks([0,1,2], ['DSL', 'Fiber', 'None'])
plt.show()
```



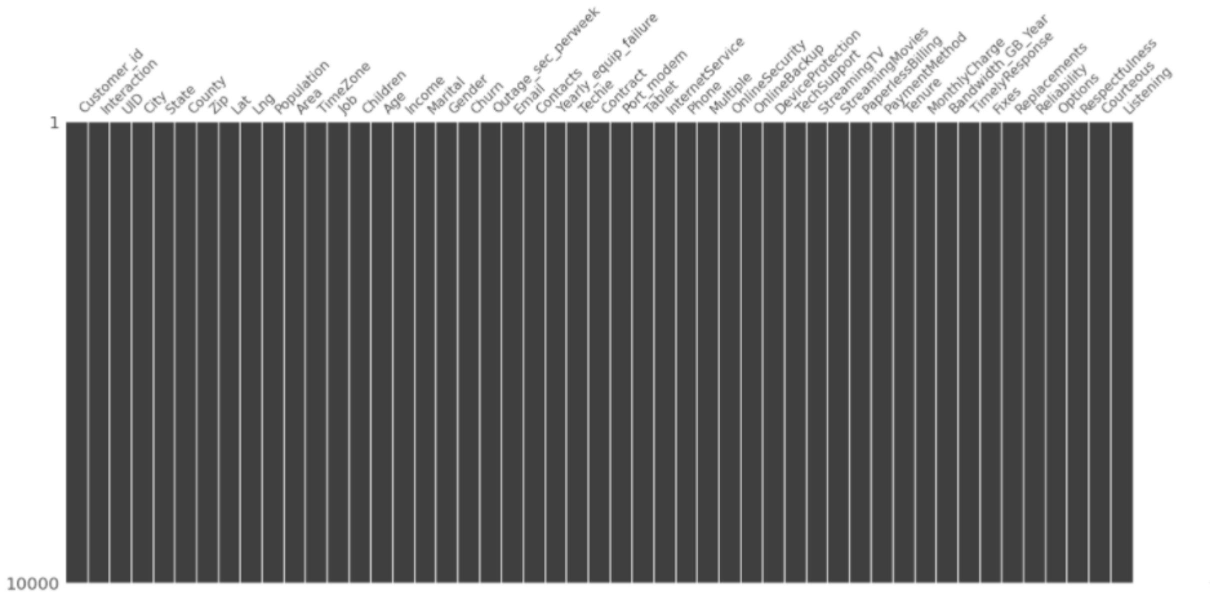
(DataCamp 2022)

```
# Verify missing data points
data_nulls = churn_df.isnull().sum()
print(data_nulls)
```

```
Customer_id             0
Interaction             0
UID                     0
City                    0
State                   0
County                  0
Zip                     0
Lat                     0
Lng                     0
Population              0
Area                    0
TimeZone                0
Job                     0
Children                0
Age                     0
Income                  0
Marital                 0
Gender                  0
Churn                   0
Outage_sec_perweek      0
Email                   0
Contacts                0
Yearly_equip_failure    0
Techie                  0
Contract                0
Port_modem              0
Tablet                  0
InternetService         0
Phone                   0
Multiple                0
OnlineSecurity          0
OnlineBackup            0
DeviceProtection        0
TechSupport             0
StreamingTV             0
StreamingMovies         0
PaperlessBilling        0
PaymentMethod           0
Tenure                  0
MonthlyCharge           0
Bandwidth_GB_Year       0
TimelyResponse          0
Fixes                   0
Replacements            0
Reliability             0
Options                 0
Respectfulness          0
Courteous               0
Listening               0
dtype: int64
```

(DataCamp 2022)

```python
# Visualize missing values in dataset using missingno

!pip install missingno
import missingno as msno

# Display matrix to visualize any missing values
msno.matrix(churn_df);
```




```python
# Convert all "Yes/No" data into binary "1/0" representation
churn_df['DummyChurn'] = [1 if v == 'Yes' else 0 for v in churn_df['Churn']]
churn_df['DummyContract'] = [1 if v == 'Two Year' else 0 for v in churn_df['Contract']]
churn_df['DummyDeviceProtection'] = [1 if v == 'Yes' else 0 for v in churn_df['DeviceProtection']]
churn_df['DummyGender'] = [1 if v == 'Male' else 0 for v in churn_df['Gender']]
churn_df['DummyInternetService'] = [1 if v == 'Fiber Optic' else 0 for v in churn_df['InternetService']]
churn_df['DummyMultiple'] = [1 if v == 'Yes' else 0 for v in churn_df['Multiple']]
churn_df['DummyOnlineBackup'] = [1 if v == 'Yes' else 0 for v in churn_df['OnlineBackup']]
churn_df['DummyOnlineSecurity'] = [1 if v == 'Yes' else 0 for v in churn_df['OnlineSecurity']]
churn_df['DummyPaperlessBilling'] = [1 if v == 'Yes' else 0 for v in churn_df['PaperlessBilling']]
churn_df['DummyPhone'] = [1 if v == 'Yes' else 0 for v in churn_df['Phone']]
churn_df['DummyPort_modem'] = [1 if v == 'Yes' else 0 for v in churn_df['Port_modem']]
churn_df['DummyStreamingTV'] = [1 if v == 'Yes' else 0 for v in churn_df['StreamingTV']]
churn_df['DummyTablet'] = [1 if v == 'Yes' else 0 for v in churn_df['Tablet']]
churn_df['DummyTechSupport'] = [1 if v == 'Yes' else 0 for v in churn_df['TechSupport']]
churn_df['DummyTechie'] = [1 if v == 'Yes' else 0 for v in churn_df['Techie']]
churn_df['StreamingMovies'] = [1 if v == 'Yes' else 0 for v in churn_df['StreamingMovies']]
```

```python
# Remove redundant 'yes/no' features from dataframe
churn_df = churn_df.drop(columns=['Gender', 'Churn', 'Techie', 'Contract', 'Port_modem', 'Tablet',
                                  'InternetService', 'Phone', 'Multiple', 'OnlineSecurity',
                                  'OnlineBackup', 'DeviceProtection', 'TechSupport',
                                  'StreamingTV', 'StreamingMovies', 'PaperlessBilling'])
```

(DataCamp 2022)

```
#Remove features not relevant to the proposed analysis question
churn_df = churn_df.drop(columns=[
    'TimelyResponse', 'Fixes Replacements', 'Reliability', 'Options', 'Respectfulness', 'Courteous', 'Listening',
    'DummyGender', 'DummyTechie', 'DummyContract', 'DummyPort_modem', 'DummyTablet', 'DummyInternetService', 'DummyPhone',
    'DummyMultiple', 'DummyOnlineSecurity', 'DummyOnlineBackup', 'DummyDeviceProtection', 'DummyTechSupport',
    'DummyStreamingTV', 'DummyPaperlessBilling', 'DummyChurn'
])
```

```
# Display 10 selected features for PCA
features = (list(churn_df.columns[:-1]))
print('Selected PCA Features: \n', features)
```

```
Selected PCA Features:
 ['Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts', 'Yearly_equip_failure', 'Tenure', 'MonthlyCharge', 'B
andwidth_GB_Year']
```

```
# Extract prepared dataset for submission
churn_df.to_csv('pca_prepared.csv')
```

(DataCamp 2022)

C2. Standardization of Dataset Variables

```
# Standard library imports, and Visualization, Statistics, SciKit libraries
import numpy as np
import pandas as pd
from pandas import Series, DataFrame

import seaborn as sns

import matplotlib.pyplot as plt
from matplotlib.axes._axes import _log as matplotlib_axes_logger
matplotlib_axes_logger.setLevel('ERROR')
%matplotlib inline

import sklearn
from sklearn.decomposition import IncrementalPCA
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler # Transform dataset mean value and standard deviation
from sklearn import metrics

# Import Scipy Cluster
import scipy
from scipy.cluster.vq import whiten

# Import matplotlib for graphing plots and style to ggplot
import matplotlib as mpl

COLOR = 'black'
mpl.rcParams['text.color'] = COLOR
mpl.rcParams['axes.labelcolor'] = COLOR
mpl.rcParams['xtick.color'] = COLOR
mpl.rcParams['ytick.color'] = COLOR

plt.style.use('ggplot')

import warnings
warnings.filterwarnings('ignore')
```

```
# Load churn dataset into a Pandas dataframe
churn_df = pd.read_csv('pca_prepared.csv')
```

```
# Display 10 selected features for PCA
features = (list(churn_df.columns[:-1]))
print('Selected PCA Features: \n', features)
```

```
Selected PCA Features:
 ['Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts', 'Yearly_equip_failure', 'Tenure', 'MonthlyCharge', 'B
andwidth_GB_Year']
```

```
# Create matrix X
X = churn_df[['Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts', 'Yearly_equip_failure', 'Tenure', 'Monthly(
```

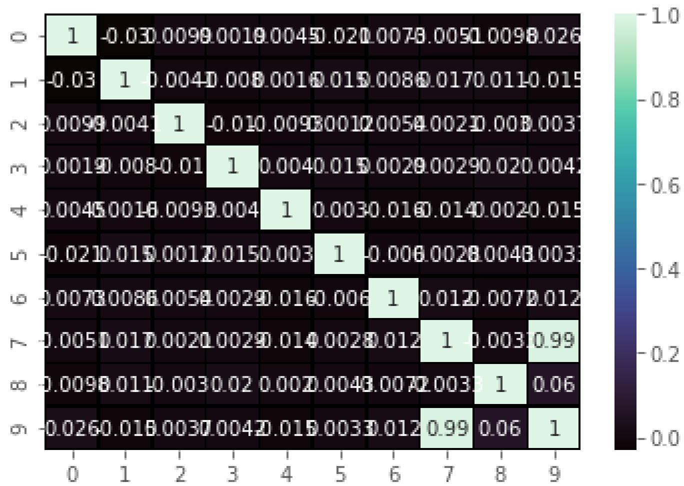
```
# Apply fit transform to StandardScaler
X_standardized = StandardScaler().fit_transform(X)
```

(DataCamp 2022)

```python
# Initialize covariance matrix and display data
mean_vec = np.mean(X_standardized, axis=0)
covariance_matrix = (X_standardized - mean_vec).T.dot((X_standardized - mean_vec)) / (X_standardized.shape[0] - 1)

print('Covariance_Matrix: \n%s' %covariance_matrix)
```

```
Covariance_Matrix:
[[ 1.00010001 -0.02973451  0.00994335  0.00188944  0.00447925 -0.02077811
   0.00732132 -0.00509183 -0.00978238  0.02558738]
 [-0.02973451  1.00010001 -0.00409101 -0.00804752  0.00158808  0.01506913
   0.00857821  0.01698097  0.01072958 -0.01472512]
 [ 0.00994335 -0.00409101  1.00010001 -0.01001155 -0.00926842  0.00123332
   0.00542382  0.00211458 -0.00301427  0.00367392]
 [ 0.00188944 -0.00804752 -0.01001155  1.00010001  0.00399413  0.01509319
   0.00290902  0.00293225  0.02049812  0.00417608]
 [ 0.00447925  0.00158808 -0.00926842  0.00399413  1.00010001  0.00304067
  -0.01635598 -0.01446932  0.00199675 -0.01458061]
 [-0.02077811  0.01506913  0.00123332  0.01509319  0.00304067  1.00010001
  -0.00603285  0.00282037  0.00425907  0.00329905]
 [ 0.00732132  0.00857821  0.00542382  0.00290902 -0.01635598 -0.00603285
   1.00010001  0.01243615 -0.00717299  0.0120349 ]
 [-0.00509183  0.01698097  0.00211458  0.00293225 -0.01446932  0.00282037
   0.01243615  1.00010001 -0.00333714  0.99159435]
 [-0.00978238  0.01072958 -0.00301427  0.02049812  0.00199675  0.00425907
  -0.00717299 -0.00333714  1.00010001  0.06041247]
 [ 0.02558738 -0.01472512  0.00367392  0.00417608 -0.01458061  0.00329905
   0.0120349   0.99159435  0.06041247  1.00010001]]
```

```python
# Plot heatmap of covariance matrix
sns.heatmap(covariance_matrix, annot=True, cmap="mako", linecolor='black', linewidths=0.5)
plt.show()
```



(DataCamp 2022)

```
# Calculate Eigen values and vectors on the covariance matrix
covariance_matrix = np.cov(X_standardized.T)
eigen_values, eigen_vectors = np.linalg.eig(covariance_matrix)

# Display Calculations of Eigencomposition
print('Vectors: \n%s' %eigen_vectors)
print('Values: \n%s' %eigen_values)
```

```
Vectors:
[[ 2.15854596e-02  1.41347924e-02 -5.59467157e-01 -2.82399326e-01
  -6.46748662e-01 -2.85318727e-01  1.41418217e-01 -2.87326245e-01
   5.77211536e-02  3.16792374e-02]
 [-2.23657297e-02  1.70801624e-03  4.79835590e-01 -5.78528649e-01
  -2.07964687e-01  4.21944284e-01 -8.98051752e-02 -4.05096045e-01
  -1.25005511e-01 -1.59620872e-01]
 [ 9.35369421e-04  4.35978315e-03 -2.23932319e-01 -9.07206677e-02
   3.02723086e-01  2.67257143e-01  1.66467676e-01 -2.94875246e-01
  -2.10454046e-01  7.87135785e-01]
 [-2.80743720e-04  5.88358241e-03  2.12259615e-01 -4.42194433e-01
   3.67329262e-01 -4.79537437e-01  5.78437841e-01  1.69773973e-03
   2.43383022e-01 -2.56863653e-02]
 [-2.46034405e-04 -2.07788587e-02  1.07066510e-01  2.05475213e-01
   2.29615135e-01 -4.38464782e-01 -4.54311812e-01 -6.86127907e-01
   1.53996990e-01 -4.96007075e-03]
 [ 9.42747188e-04  4.17502587e-03  4.58770120e-01  2.54312989e-01
  -4.38267152e-01  1.38442926e-02  1.04530277e-01  4.31843019e-02
   5.50932285e-01  4.65025757e-01]
 [ 9.52581748e-05  1.75653215e-02 -1.43554702e-01  4.08175882e-01
   7.89968226e-02  3.95130635e-01  5.30963217e-01 -4.24544209e-01
   2.27787102e-01 -3.68863854e-01]
 [ 7.05262361e-01  7.05422257e-01  1.85082253e-03 -2.22443617e-02
   2.97190659e-02  2.10784846e-02 -4.17351931e-02  4.47130889e-03
   3.70435709e-02 -4.96324517e-03]
 [ 4.57545853e-02  4.04234226e-02  3.44887052e-01  3.28189805e-01
  -2.44887367e-01 -2.99619131e-01  3.29363949e-01 -1.16153637e-01
  -7.04988074e-01  2.99153688e-02]
 [-7.06783878e-01  7.06916770e-01 -7.92224048e-03  9.11019719e-03
   2.31786341e-04 -1.96605152e-02 -1.28030621e-02  8.34585697e-04
  -2.61919415e-03  4.62684898e-03]]
Values:
[0.0054677  1.99433311 1.05333463 0.96035059 0.96476747 1.02755391
 1.01255858 0.98905858 0.99377999 0.99979555]
```

(DataCamp 2022)

Part IV: Analysis

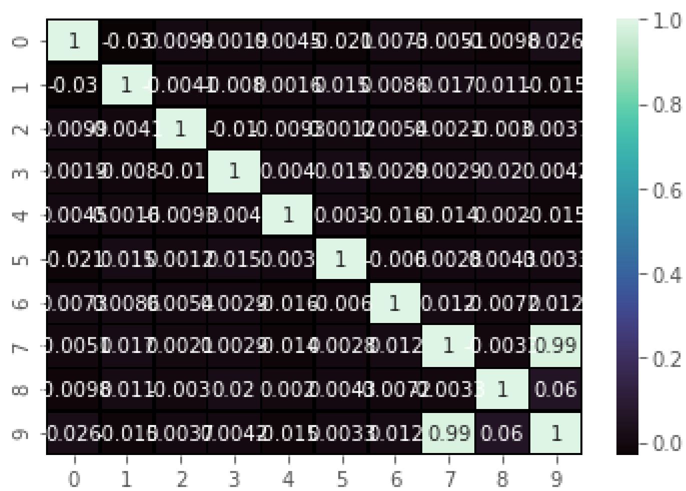
D1. Principal Components

The variables under consideration for PCA are: Age, Bandwidth_GB_Year, Children, Contacts, Email, Income, MonthlyCharge, Outage_sec_perweek, Tenure, and Yearly_equip_failure.

We can create a Covariance Matrix of the variables using NumPy to calculate the mean vector then take the standardize values ratio minus the mean divided by the shape of the values. From this manipulation we calculate our covariance matrix as follows:

```
Covariance_Matrix:
[[ 1.00010001 -0.02973451  0.00994335  0.00188944  0.00447925 -0.02077811
   0.00732132 -0.00509183 -0.00978238  0.02558738]
 [-0.02973451  1.00010001 -0.00409101 -0.00804752  0.00158808  0.01506913
   0.00857821  0.01698097  0.01072958 -0.01472512]
 [ 0.00994335 -0.00409101  1.00010001 -0.01001155 -0.00926842  0.00123332
   0.00542382  0.00211458 -0.00301427  0.00367392]
 [ 0.00188944 -0.00804752 -0.01001155  1.00010001  0.00399413  0.01509319
   0.00290902  0.00293225  0.02049812  0.00417608]
 [ 0.00447925  0.00158808 -0.00926842  0.00399413  1.00010001  0.00304067
  -0.01635598 -0.01446932  0.00199675 -0.01458061]
 [-0.02077811  0.01506913  0.00123332  0.01509319  0.00304067  1.00010001
  -0.00603285  0.00282037  0.00425907  0.00329905]
 [ 0.00732132  0.00857821  0.00542382  0.00290902 -0.01635598 -0.00603285
   1.00010001  0.01243615 -0.00717299  0.0120349 ]
 [-0.00509183  0.01698097  0.00211458  0.00293225 -0.01446932  0.00282037
   0.01243615  1.00010001 -0.00333714  0.99159435]
 [-0.00978238  0.01072958 -0.00301427  0.02049812  0.00199675  0.00425907
  -0.00717299 -0.00333714  1.00010001  0.06041247]
 [ 0.02558738 -0.01472512  0.00367392  0.00417608 -0.01458061  0.00329905
   0.0120349   0.99159435  0.06041247  1.00010001]]
```

We can plot a visualization of the Matrix to illustrate the values:



We can use our Matrix to calculate the Eigenvectors using NumPy and the Linalg function in the library. We determine the Eigenvectors as follows:

Eigenvectors

```
[[ 2.15854596e-02  1.41347924e-02 -5.59467157e-01 -2.82399326e-01
  -6.46748662e-01 -2.85318727e-01  1.41418217e-01 -2.87326245e-01
   5.77211536e-02  3.16792374e-02]
 [-2.23657297e-02  1.70801624e-03  4.79835590e-01 -5.78528649e-01
  -2.07964687e-01  4.21944284e-01 -8.98051752e-02 -4.05096045e-01
  -1.25005511e-01 -1.59620872e-01]
 [ 9.35369421e-04  4.35978315e-03 -2.23932319e-01 -9.07206677e-02
   3.02723086e-01  2.67257143e-01  1.66467676e-01 -2.94875246e-01
  -2.10454046e-01  7.87135785e-01]
 [-2.80743720e-04  5.88358241e-03  2.12259615e-01 -4.42194433e-01
   3.67329262e-01 -4.79537437e-01  5.78437841e-01  1.69773973e-03
   2.43383022e-01 -2.56863653e-02]
 [-2.46034405e-04 -2.07788587e-02  1.07066510e-01  2.05475213e-01
   2.29615135e-01 -4.38464782e-01 -4.54311812e-01 -6.86127907e-01
   1.53996990e-01 -4.96007075e-03]
 [ 9.42747188e-04  4.17502587e-03  4.58770120e-01  2.54312989e-01
  -4.38267152e-01  1.38442926e-02  1.04530277e-01  4.31843019e-02
   5.50932285e-01  4.65025757e-01]
 [ 9.52581748e-05  1.75653215e-02 -1.43554702e-01  4.08175882e-01
   7.89968226e-02  3.95130635e-01  5.30963217e-01 -4.24544209e-01
   2.27787102e-01 -3.68863854e-01]
 [ 7.05262361e-01  7.05422257e-01  1.85082253e-03 -2.22443617e-02
   2.97190659e-02  2.10784846e-02 -4.17351931e-02  4.47130889e-03
   3.70435709e-02 -4.96324517e-03]
 [ 4.57545853e-02  4.04234226e-02  3.44887052e-01  3.28189805e-01
  -2.44887367e-01 -2.99619131e-01  3.29363949e-01 -1.16153637e-01
  -7.04988074e-01  2.99153688e-02]
 [-7.06783878e-01  7.06916770e-01 -7.92224048e-03  9.11019719e-03
   2.31786341e-04 -1.96605152e-02 -1.28030621e-02  8.34585697e-04
  -2.61919415e-03  4.62684898e-03]]
```

Using the same NumPy Linalg function we calculate our Eigenvalues as follows:

EigenValues

```
[0.0054677  1.99433311 1.05333463 0.96035059 0.96476747 1.02755391
 1.01255858 0.98905858 0.99377999 0.99979555]
```

Finally, to gain better insight from our PCA we sort the Eigenvalues by decreasing value and then calculate the variance ratio of the eigenvectors and eigenvalues.

```
# Sort Eigenvalues in descending value
eigen_pairs = [(np.abs(eigen_values[i]), eigen_vectors[:,i]) for i in range(len(eigen_values))]
eigen_pairs.sort(key=lambda x: x[0], reverse=True)

# Display Eigenvalues
print('Values:')
for i in eigen_pairs:
    print(i[0])
```

```
Values:
1.9943331101364756
1.0533346256283744
1.0275539108057439
1.0125585769497123
0.9997955481739358
0.9937799880491779
0.9890585843412278
0.9647674706143197
0.9603505880456993
0.005467697265331362
```

```
# Fit features into standardized matrix using PCA library
pca = PCA().fit(X_standardized)

# Display PCA explained variance ratio
print(pca.explained_variance_ratio_)
```

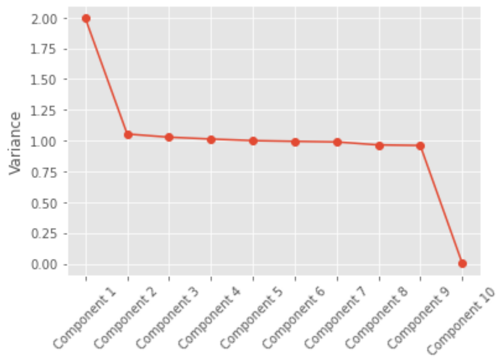
```
[0.19941337 0.10532293 0.10274512 0.10124573 0.09996956 0.09936806
 0.09889597 0.0964671  0.09602546 0.00054672]
```

The resulting Matrix of *all* principal components, variables, and eigenvalues reveals the following:

	Principal Component									
Variable	1	2	3	4	5	6	7	8	9	10
Age	0.021	0.014	**-0.559**	-0.282	**-0.646**	-0.285	0.141	-0.287	0.057	0.031
Bandwidth	-0.022	0.002	**0.480**	**-0.579**	-0.208	**0.422**	-0.090	**-0.405**	-0.125	-0.160
Children	0.001	0.004	-0.224	-0.091	0.303	0.267	0.166	-0.295	-0.210	**0.787**
Contacts	0.000	0.006	0.212	-0.442	0.367	-0.480	0.578	0.002	0.243	-0.026
Email	0.000	-0.021	0.107	0.205	0.230	**-0.438**	**-0.454**	**-0.686**	0.154	-0.005
Income	0.001	0.004	**0.459**	0.254	**-0.438**	0.014	0.105	0.043	0.551	**0.465**
MonthlyCharge	0.000	0.018	-0.144	**0.408**	0.079	0.395	0.531	**-0.425**	0.228	-0.369
Outages	**0.705**	**0.705**	0.002	-0.022	0.030	0.021	-0.042	0.004	0.037	-0.005
Tenure	0.046	0.040	0.345	0.328	-0.245	-0.300	0.329	-0.116	-0.705	0.030
YearlyFailures	**-0.707**	**0.707**	-0.008	0.009	0.000	-0.020	-0.013	0.001	-0.003	0.005

D2. Identification of Total Number of Components

```python
# Create Scree Plot of standardized values and display plot
def screeplot(pca, standardized_values):
    y = np.std(pca.transform(standardized_values), axis=0)**2
    x = np.arange(len(y)) + 1
    plt.plot(x, y, "o-")
    plt.xticks(x, ['Component ' + str(i) for i in x], rotation=45)
    plt.ylabel('Variance')
    plt.show()

screeplot(pca, X_standardized)
```



(DataCamp 2022)

D3. Total Variance of Components

```python
# Sort components in order of importance
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

# Print summary calculations
summary = pca_summary(pca, X_standardized)
summary.standard_deviation**2
```

```
Component importance:
```

	Standard Deviation
PC 1	1.994134
PC 2	1.053229
PC 3	1.027451
PC 4	1.012457
PC 5	0.999696
PC 6	0.993681
PC 7	0.988960
PC 8	0.964671
PC 9	0.960255
PC 10	0.005467

D4. Total Variance Captured by Components

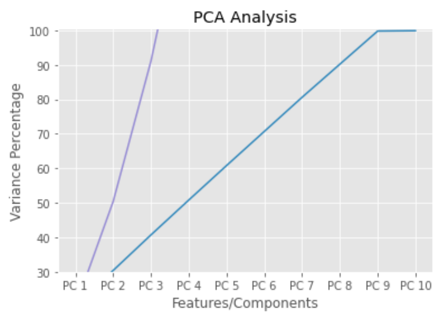
```
# Show Standard Deviation of total variance calculated by PCA
np.sum(summary.standard_deviation**2)
```

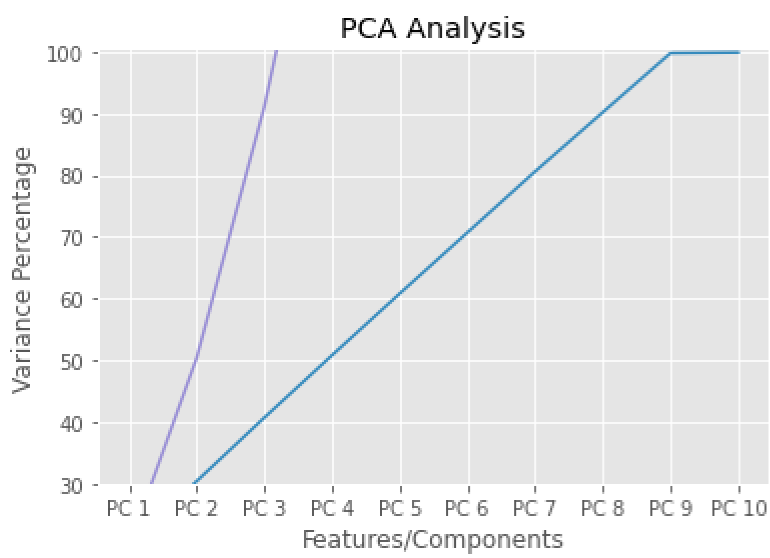
```
Standard Deviation    10.0
dtype: float64
```

```
# Show calculated variance captured by features/components
var = np.cumsum(np.round(summary, decimals=3)*100)
```

```
# Plot PCA Analysis
plt.ylabel('Variance Percentage')
plt.xlabel('Features/Components')
plt.title('PCA Analysis')
plt.ylim(30,100.5)

plt.plot(var)
```

```
[<matplotlib.lines.Line2D at 0x276227b51c0>,
 <matplotlib.lines.Line2D at 0x276227b50d0>,
 <matplotlib.lines.Line2D at 0x276227b59a0>]
```





The calculated Variance Totals can be demonstrated in the PCA plot above. The plot displays the Variance Percentage on the y-axis and Features or Components on the x-axis. From the plot we can observe that half of all calculated variance is determined by 4 components showing a value greater or equal to 1. The remaining six components have a variance score less than 1 and are plotted in contrast to determined variance. In total we account for 100% variance of all components.

D5. Summary of Data Analysis

The goal of the Principal Component Analysis is to reduce the number of features to the smallest amount possible while preventing overfitting and providing meaningful insight. The features or components chosen were selected based on three criteria, the first criterion is to select components with a high amount of variance, the second criterion is selecting uncorrelated components to reduce multicollinearity, and finally we selected the fewest and only the most crucial components for analysis (Cheng 2022).

The information from this analysis we should share with stakeholders and executives are the predictor variables that show a determined total variance score of 10. From our calculated components, 4 show a total variance score higher than 1.

These components determined by the Principal Component Analysis to be of great importance as predictors of churn should be further studied and analyzed. Based on the results of the PCA analysis these components indicate key predictors of customer churn. By analyzing historical data for these four components, we will be able to better predict the likelihood of churn given customer characteristics and ultimately reduce customer disconnections and increase annual profits.

Part V: Attachments

E. Sources for Third-Party Code

Dimensionality reduction in python course. DataCamp. (n.d.). Retrieved March 25, 2022, from https://www.datacamp.com/courses/dimensionality-reduction-in-python

Python PCA tutorial: Principal component analysis with Sklearn. DataCamp Community. (n.d.). Retrieved March 25, 2022, from https://www.datacamp.com/community/tutorials/principal-component-analysis-in-python

Principal component analysis: Python. campus.datacamp.com. (n.d.). Retrieved March 25, 2022, from https://campus.datacamp.com/courses/ai-fundamentals/unsupervised-learning?ex=2

F. Sources

Cheng, C. (2022, March 22). *Principal Component Analysis (PCA) explained visually with Zero math*. Medium. Retrieved March 25, 2022, from https://towardsdatascience.com/principal-component-analysis-pca-explained-visually-with-zero-math-1cbf392b9e7d

Yiu, T. (2021, September 29). *Understanding PCA*. Medium. Retrieved March 25, 2022, from https://towardsdatascience.com/understanding-pca-fae3e243731d

Yiu, T. (2021, September 29). *The curse of dimensionality*. Medium. Retrieved March 25, 2022, from https://towardsdatascience.com/the-curse-of-dimensionality-50dc6e49aa1e