PERFORMANCE ASSESSMENT_ D212

OFM3 TASK 2: Dimensionality Reduction Methods

March 7th,2022

Table of Contents

Performance Evaluation of OFM3: Dimensionality Reduction Methods, D212

Name: Rekha Alle
Student ID: 000778673
Course: Masters Data Analytics
Date: 03/07/2022
Program Mentor: Kesselly kamara
Contact: 385-428-4395
Email: Kesselly.kamara@wgu.edu

PARTI. Introduction to Scenario

Understanding customers is one of the most important aspects of customer relationship management that directly influences a company's long-term success. When a corporation has a greater understanding of its consumers' traits, this could good target promotion and advertising campaigns for them, resulting in higher long-term earnings.

You operate as an investigator for a telecoms business that wants to understand more about its customers' characteristics. You've been given with conducting market basket research on customer research to discover critical relationships between consumer purchases, enabling for better operational and organizational selection.

A1. Analytical Question:

Which features of customers suggest a possibility for churn? The principal component analysis (PCA) method will be used to solve this analysis.
To put it another way, we're attempting to better understand the links between customer features in intended to inform investor decisions, even if we're not applying a supervised learning model to generate predictions, such as linear regression.

A2. Goals and Objectives:

Everybody in the organization will profit from recognizing, with some degree of certainty, which customers will be able to churn, since it will give importance to selling enhanced services to consumers with these traits and previous user experiences. This data analysis' purpose is to give numerical information to company stakeholders to assist them better understand their customers.

PARTII. Justification for the method

Do the following to demonstrate why PCA is used:

B1. PCA is explained as follows:

The feature extraction method employed in this study is Principal Component Analysis (PCA). PCA is broken down into linear algebra procedures that modify the dataset into a more tractable format with fewer, more relevant variables. PCA is done in the following manner:

- The Eigenvectors and Eigenvalues can be found in the covariance or correlation matrix.
- Sort the Eigenvalues in ascending order, then pick the k correspond to k Eigenvectors the biggest Eigenvalues, where k is the number of dimensions (columns) in the new feature subspace.
- Ensure that the data is uniform. This is done by subtracting the mean of the pieces of data from the total number of data points and dividing by the standard deviation.
- To get a k-dimensional feature subspace Y, transform the original dataset x via W.
- Construct the projection matrix W using the k Eigenvectors you've chosen. (SuperDataScience)

PCA is expected to provide eigan vectors in decreasing order, such as PC1, PC2, PC3, and so on. As a result, our data will have new axes.

B2. Assumption summary PCA:

This approach assumes that by projecting this d-dimensional churn dataset onto a k-dimensional subspace, we will reduce its dimensions (number of our customers' features). The goal is to find k characteristics that are less than d. (SuperDataScience).

PARTIII. Data Objectives:

C1. Variables in the Dataset:

We might uncover the continuous predictor factors' importance while cleaning the data:

- Bandwidth_GB_Year
- MonthlyCharge
- Tenure (the length of time customer service)
- Contacts
- Email
- Outage_sec_perweek
- Income
- Age
- Children
- Yearly_equip_failure

Dummy variables (1/0) will be used to encode it.

1. Include standard imports all the required references:

In [1]:
```
# Standard data science imports
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
# Visualization libraries
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.axes._axes import _log as matplotlib_axes_logger
matplotlib_axes_logger.setLevel('ERROR')
%matplotlib inline
# Scikit-Learn
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import IncrementalPCA
from sklearn.cluster import KMeans
from sklearn import metrics
# Import Scikit Learn PCA application
from sklearn.decomposition import PCA
# Import Scipy for feature scaling
import scipy
from scipy.cluster.vq import whiten
```

2. Change font and color of the Matplotlib:

In [2]:
```
# Change color of Matplotlib font
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
```

3. Increase display cell-width

In [3]:
```
# Increase Jupyter display cell-width
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:75% !important; }</style>"))
```

4. Ignore warning codes

In [4]:
```
# Ignore Warning Code
import warnings
warnings.filterwarnings('ignore')
```

5. Dataset

In [5]:
```
# Load data set into Pandas dataframe
churn_df = pd.read_csv('C:/Kailash/Rekha/D212/data/churn_clean.csv')
```

6. Data set features

In [7]:
```
# Examine the features of the dataset
churn_df.columns
```

Out[7]:
```
Index(['CaseOrder', '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')
```

7. Data set Size

```
In [7]: # Get an idea of dataset size
        churn_df.shape

Out[7]: (10000, 50)
```

8. Data frame Info

```
In [8]: # View DataFrame info
        churn_df.info

Out[8]: <bound method DataFrame.info of       CaseOrder Customer_id                           Interaction  \
        0             1    K409198  aa90260b-4141-4a24-8e36-b04ce1f4f77b
        1             2    S120509  fb76459f-c047-4a9d-8af9-e0f7d4ac2524
        2             3    K191035  344d114c-3736-4be5-98f7-c72c281e2d35
        3             4     D90850  abfa2b40-2d43-4994-b15a-989b8c79e311
        4             5    K662701  68a861fd-0d20-4e51-a587-8a90407ee574
        ...         ...        ...                                   ...
        9995       9996    M324793  45deb5a2-ae04-4518-bf0b-c82db8dbe4a4
        9996       9997    D861732  6e96b921-0c09-4993-bbda-a1ac6411061a
        9997       9998    I243405  e8307ddf-9a01-4fff-bc59-4742e03fd24f
        9998       9999    I641617  3775ccfc-0052-4107-81ae-9657f81ecdf3
        9999      10000     T38070  9de5fb6e-bd33-4995-aec8-f01d0172a499

                                   UID         City State  \
        0     e885b299883d4f9fb18e39c75155d990  Point Baker    AK
        1     f2de8bef964785f41a2959829830fb8a  West Branch    MI
        2     f1784cfa9f6d92ae816197eb175d3c71      Yamhill    OR
        3     dc8a365077241bb5cd5ccd305136b05e      Del Mar    CA
        4     aabb64a116e83fdc4befc1fbab1663f9    Needville    TX
        ...                                ...          ...   ...
        9995  9499fb4de537af195d16d046b79fd20a  Mount Holly    VT
        9996  c09a841117fa81b5c8e19afec2760104  Clarksville    TN
        9997  9c41f212d1e04dca84445019bbc9b41c     Mobeetie    TX
        9998  3e1f269b40c235a1038863ecf6b7a0df   Carrollton    GA
        9999  0ea683a03a3cd544aefe8388aab16176  Clarkesville    GA

                          County    Zip       Lat        Lng  ...  MonthlyCharge  \
0     Prince of Wales-Hyder  99927  56.25100 -133.37571  ...     172.455519
1                    Ogemaw  48661  44.32893  -84.24080  ...     242.632554
2                   Yamhill  97148  45.35589 -123.24657  ...     159.947583
3                 San Diego  92014  32.96687 -117.24798  ...     119.956840
4                 Fort Bend  77461  29.38012  -95.80673  ...     149.948316
...                     ...    ...       ...        ...  ...            ...
9995                Rutland   5758  43.43391  -72.78734  ...     159.979400
9996             Montgomery  37042  36.56907  -87.41694  ...     207.481100
9997                Wheeler  79061  35.52039 -100.44180  ...     169.974100
9998                Carroll  30117  33.58016  -85.13241  ...     252.624000
9999              Habersham  30523  34.70783  -83.53648  ...     217.484000

      Bandwidth_GB_Year Item1 Item2  Item3  Item4  Item5 Item6 Item7 Item8
0            904.536110     5     5      5      3      4     4     3     4
1            800.982766     3     4      3      3      4     3     4     4
2           2054.706961     4     4      2      4      4     3     3     3
3           2164.579412     4     4      4      2      5     4     3     3
4            271.493436     4     4      4      3      4     4     4     5
...                 ...   ...   ...    ...    ...    ...   ...   ...   ...
9995        6511.252601     3     2      3      3      4     3     2     3
9996        5695.951810     4     5      5      4      4     5     2     5
9997        4159.305799     4     4      4      4      4     4     4     5
9998        6468.456752     4     4      6      4      3     3     5     4
9999        5857.586167     2     2      3      3      3     3     4     1

[10000 rows x 50 columns]>
```

9. Data set

In [9]:
```
# Provide an initial look at extant dataset
churn_df.head()
```

Out[9]:

	CaseOrder	Customer_id	Interaction	UID	City	State	County	Zip	Lat	Lng	...	MonthlyCharge	Bandwidth_GB_Year	Item1	Item2	Item3	Item4
0	1	K409198	aa90260b-4141-4a24-8e36-b04ce1f4f77b	e885b299883d4f9fb18e39c75155d990	Point Baker	AK	Prince of Wales-Hyder	99927	56.25100	-133.37571	...	172.455519	904.536110	5	5	5	3
1	2	S120509	fb76459f-c047-4a9d-8af9-e0f7d4ac2524	f2de8bef964785f41a2959829830fb8a	West Branch	MI	Ogemaw	48661	44.32893	-84.24080	...	242.632554	800.982766	3	4	3	3
2	3	K191035	344d114c-3736-4be5-98f7-c72c281e2d35	f1784cfa9f6d92ae816197eb175d3c71	Yamhill	OR	Yamhill	97148	45.35589	-123.24657	...	159.947583	2054.706961	4	4	2	4
3	4	D90850	abfa2b40-2d43-4994-b15a-989b8c79e311	dc8a365077241bb5cd5ccd305136b05e	Del Mar	CA	San Diego	92014	32.96687	-117.24798	...	119.956840	2164.579412	4	4	4	2
4	5	K662701	68a861fd-0d20-4e51-a587-8a90407ee574	aabb64a116e83fdc4befc1fbab1663f9	Needville	TX	Fort Bend	77461	29.38012	-95.80673	...	149.948316	271.493436	4	4	4	3

5 rows × 50 columns

10. Descriptive statics

In [10]:
```
# Get an overview of descriptive statistics
churn_df.describe()
```

Out[10]:

	CaseOrder	Zip	Lat	Lng	Population	Children	Age	Income	Outage_sec_perweek	Email	...	MonthlyCharge	Bandwidth_GB_Year
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.0000	10000.000000	10000.000000	10000.000000	10000.000000	...	10000.000000	10000.000000
mean	5000.50000	49153.319600	38.757567	-90.782536	9756.562400	2.0877	53.078400	39806.926771	10.001848	12.016000	...	172.624816	3392.341550
std	2886.89568	27532.196108	5.437389	15.156142	14432.698671	2.1472	20.698882	28199.916702	2.976019	3.025898	...	42.943094	2185.294852
min	1.00000	601.000000	17.966120	-171.688150	0.000000	0.0000	18.000000	348.670000	0.099747	1.000000	...	79.978860	155.506715
25%	2500.75000	26292.500000	35.341828	-97.082812	738.000000	0.0000	35.000000	19224.717500	8.018214	10.000000	...	139.979239	1236.470827
50%	5000.50000	48869.500000	39.395800	-87.918800	2910.500000	1.0000	53.000000	33170.605000	10.018560	12.000000	...	167.484700	3279.536903
75%	7500.25000	71866.500000	42.106908	-80.088745	13168.000000	3.0000	71.000000	53246.170000	11.969485	14.000000	...	200.734725	5586.141370
max	10000.00000	99929.000000	70.640660	-65.667850	111850.000000	10.0000	89.000000	258900.700000	21.207230	23.000000	...	290.160419	7158.981530

8 rows × 23 columns

11. Non-numerical data

In [11]:
```
# Use describe method to view non-numerical data
churn_df.describe(exclude='number')
```

Out[11]:

	Customer_id	Interaction	UID	City	State	County	Area	TimeZone	Job	Marital	...	Phone	Multiple	OnlineSecurity	OnlineBackup
count	10000	10000	10000	10000	10000	10000	10000	10000	10000	10000	...	10000	10000	10000	10000
unique	10000	10000	10000	6058	52	1620	3	25	639	5	...	2	2	2	2
top	M376642	cc5bd887-c8a2-4cbf-aef2-ec21d7a52e59	ac45aefd127442fa2570ae44ed546396	Houston	TX	Washington	Suburban	America/New_York	Occupational psychologist	Divorced	...	Yes	No	No	No
freq	1	1	1	34	603	111	3346	4072	30	2092	...	9067	5392	6424	5494

4 rows × 27 columns

12. Data types

In [9]:
```
# Get data types of features
churn_df.dtypes
```

Out[9]:
```
CaseOrder                  int64
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
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
```

13. Dummy Variables

```
In [14]: # Encode binary categorical variable with dummies
churn_df['DummyChurn'] = [1 if v == 'Yes' else 0 for v in churn_df['Churn']] ### If the customer left (churned) they get a'1'
```

14. Drop the features from Data frame

```
In [15]: # Drop original binary categorical feature from dataframe
churn_df = churn_df.drop(columns=['Churn'])
```

15. Remove categorical variables from dataset

```
In [16]: # Remove less meaningful non-numerical categorical variables from dataset to provide fully numerical dataframe
churn_df = churn_df.drop(columns=['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City', 'State',
'County', 'Zip', 'Lat', 'Lng', 'Area', 'TimeZone',
'Job', 'Marital', 'PaymentMethod', 'Gender', 'Techie',
'Contract', 'Port_modem', 'Tablet',
'InternetService', 'Phone', 'Multiple', 'OnlineSecurity',
'OnlineBackup', 'DeviceProtection', 'TechSupport',
'StreamingTV', 'StreamingMovies', 'PaperlessBilling',
'Item2', 'Item3', 'Item4', 'Item5', 'Item6', 'Item7',
'Item8'])
```

16. To set as target Move Dummy Churn to end:

```
In [17]: # Move DummyChurn to end of dataset to set as target
churn_df = churn_df[['Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts',
'Yearly_equip_failure', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year', 'DummyChurn']]
```

17. Churn data frame with values:

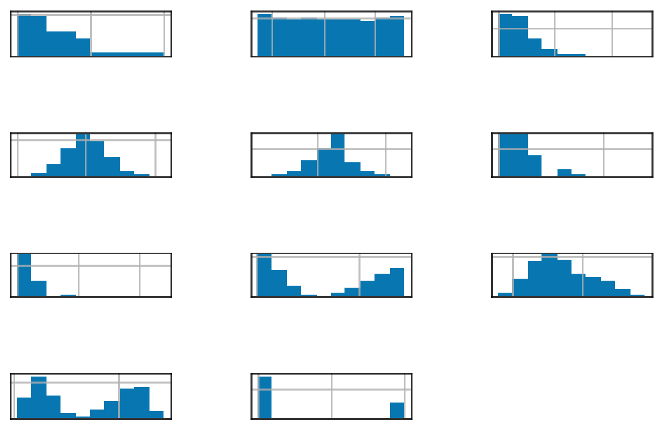
```
In [18]: # Review changes in DataFrame
churn_df.head()
```

Out[18]:

	Children	Age	Income	Outage_sec_perweek	Email	Contacts	Yearly_equip_failure	Tenure	MonthlyCharge	Bandwidth_GB_Year	DummyChurn
0	0	68	28561.99	7.978323	10	0	1	6.795513	172.455519	904.536110	0
1	1	27	21704.77	11.699080	12	0	1	1.156681	242.632554	800.982766	1
2	4	50	9609.57	10.752800	9	0	1	15.754144	159.947583	2054.706961	0
3	1	48	18925.23	14.913540	15	2	0	17.087227	119.956840	2164.579412	0
4	0	83	40074.19	8.147417	16	2	1	1.670972	149.948316	271.493436	1

18. To create histograms:

```
In [20]: # Create histograms of contiuous variables & categorical variables
churn_df[['Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email',
'Contacts', 'Yearly_equip_failure', 'Tenure', 'MonthlyCharge',
'Bandwidth_GB_Year', 'DummyChurn']].hist()
plt.tight_layout()
```

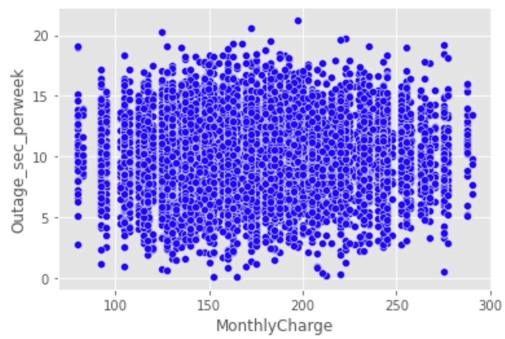


19. To plot style to ggplot:

```
In [21]: # Set plot style to ggplot for aesthetics & R style
plt.style.use('ggplot')
```

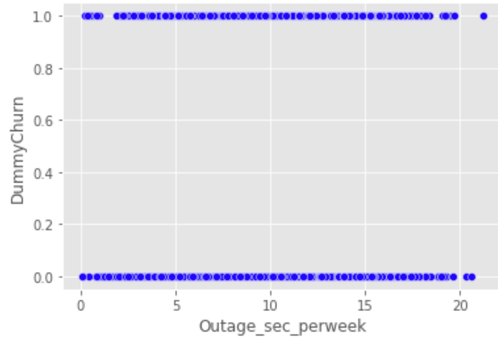
20. scatterplot with Monthly charges:

```
In [22]: # Create a scatterplot to get an idea of correlations between potentially related variables
sns.scatterplot(x=churn_df['MonthlyCharge'], y=churn_df['Outage_sec_perweek'], color='blue')
plt.show();
```



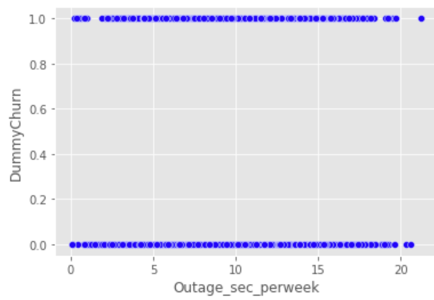
21. Outage per week Scatterplot:

In [23]:
```
# Create a scatterplot to get an idea of correlations between potentially related variables
sns.scatterplot(x=churn_df['Outage_sec_perweek'], y=churn_df['DummyChurn'], color='blue')
plt.show();
```



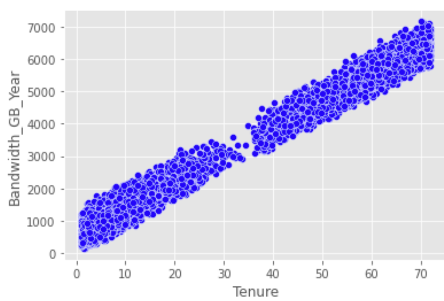
22. Scatterplot with Dummy churn:

In [23]:
```
# Create a scatterplot to get an idea of correlations between potentially related variables
sns.scatterplot(x=churn_df['Outage_sec_perweek'], y=churn_df['DummyChurn'], color='blue')
plt.show();
```



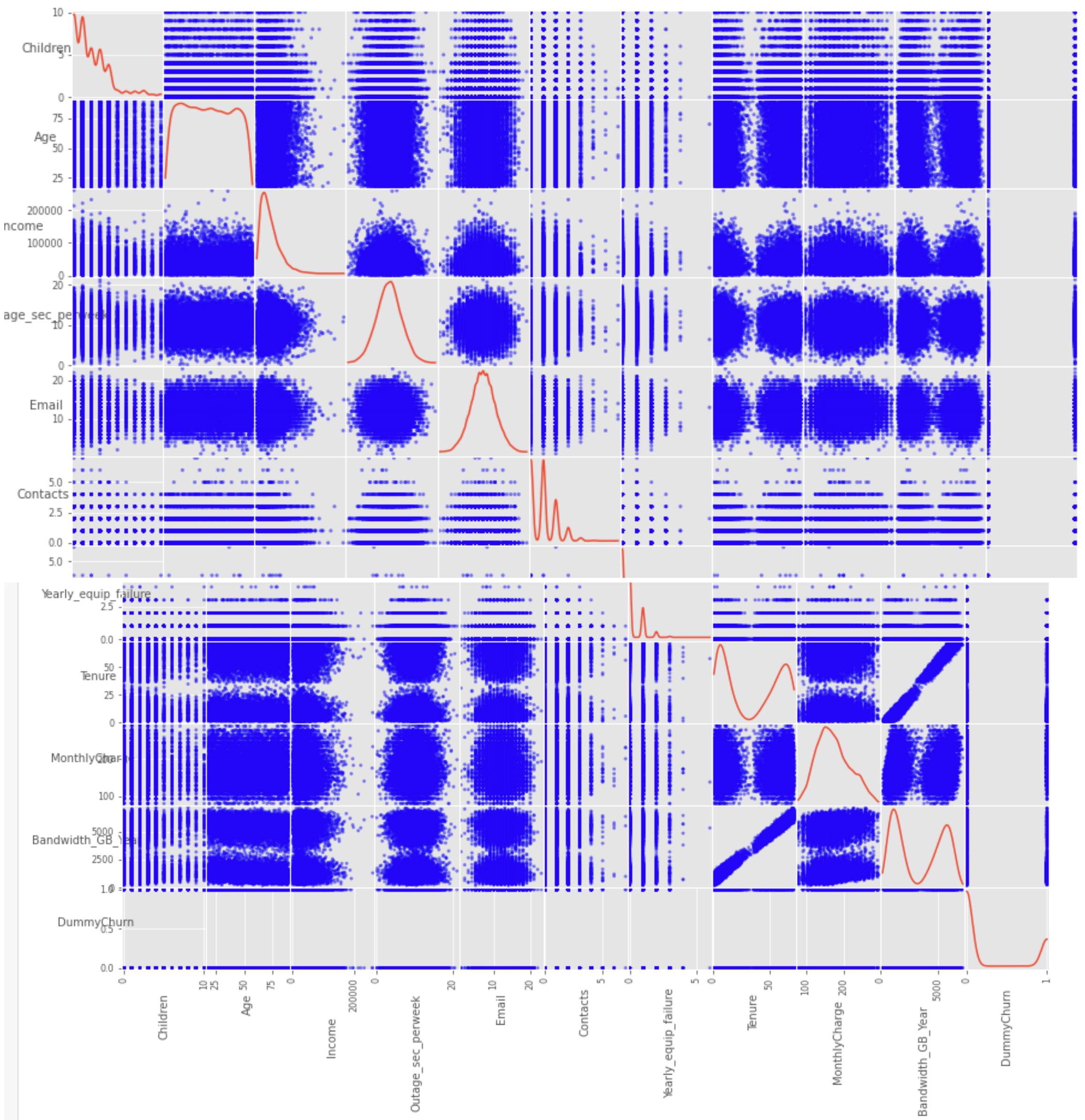
23. scatterplot between Tenure and Band width:

In [24]:
```
# Create a scatterplot to get an idea of correlations between potentially related variables
sns.scatterplot(x=churn_df['Tenure'], y=churn_df['Bandwidth_GB_Year'], color='blue')
plt.show();
```



24. scatter_matrix:

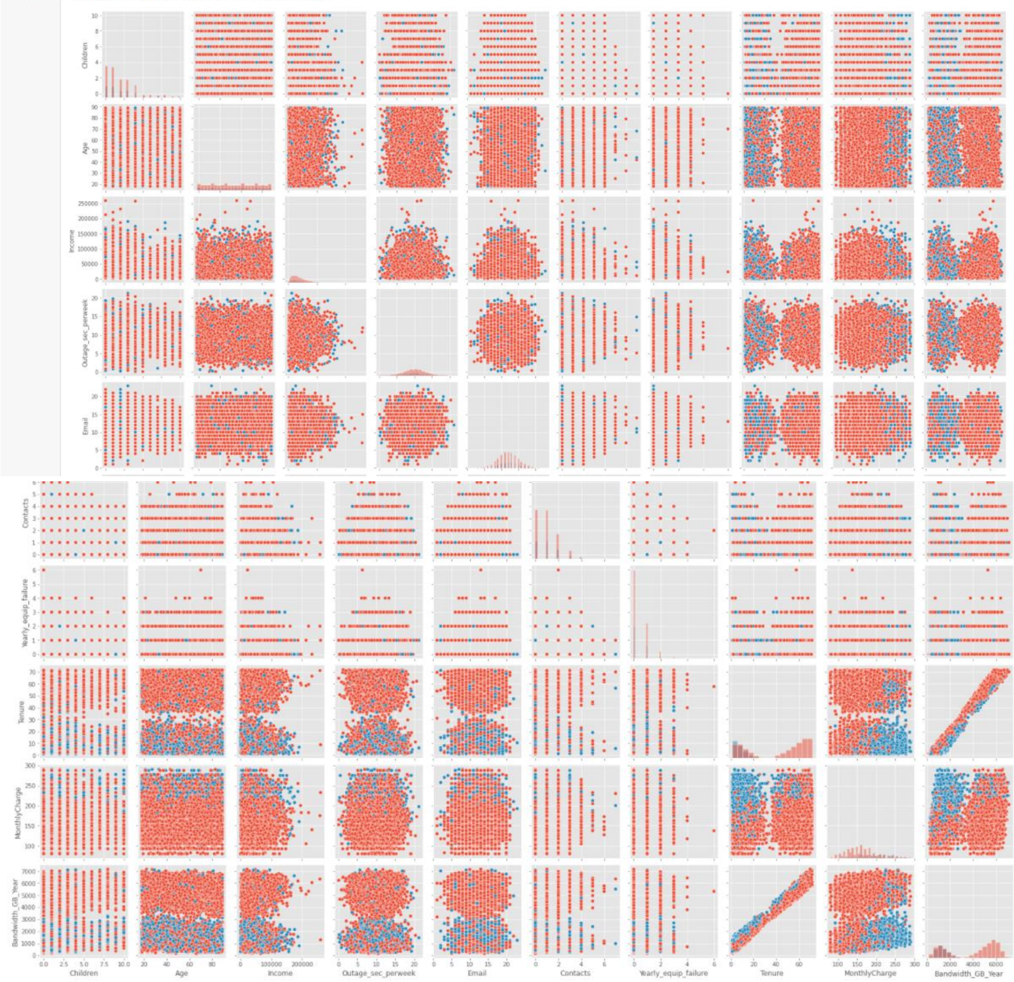
```
In [27]: # Provide a scatter matrix of numeric variables for high level overview of potential relationships & distributions
churn_numeric = churn_df[['Children', 'Age', 'Income', 'Outage_sec_perweek',
'Email', 'Contacts','Yearly_equip_failure', 'Tenure',
'MonthlyCharge', 'Bandwidth_GB_Year', 'DummyChurn']]
scatter_matrix = pd.plotting.scatter_matrix(
churn_numeric,
figsize = [15, 15],
diagonal = "kde",
color="b"
)

for ax in scatter_matrix.ravel():
    ax.set_xlabel(ax.get_xlabel(), fontsize = 10, rotation = 90)
    ax.set_ylabel(ax.get_ylabel(), fontsize = 10, rotation = 0)
```



25. Seaborn pair plot of numeric variables for high level overview

In [28]:
```
# Provide a Seaborn pairplot of numeric variables for high Level overview of potential relationships & distributions
sns.pairplot(churn_df, hue='DummyChurn', diag_kind='hist')
```

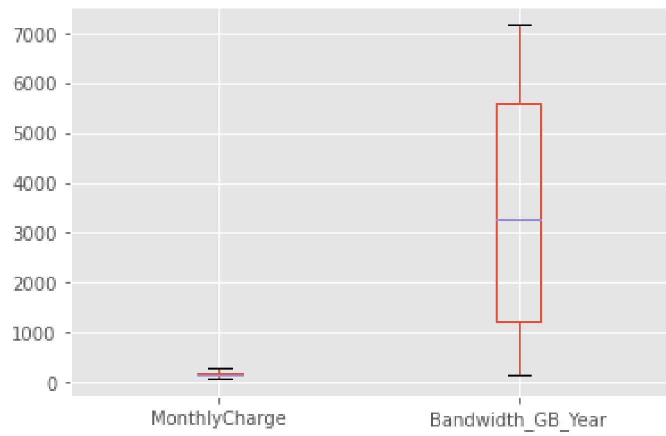
Out[28]:
```
<seaborn.axisgrid.PairGrid at 0x20be3d33940>
```



26. multiple boxplots

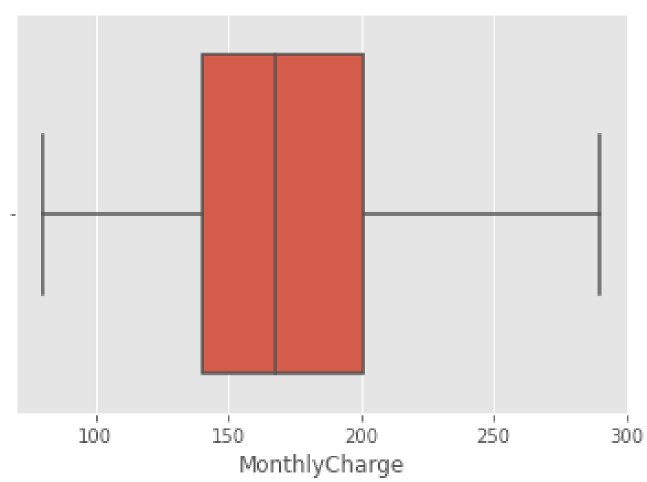
```
In [29]: # Create multiple boxplots for continuous & categorical variables
         churn_df.boxplot(column=['MonthlyCharge','Bandwidth_GB_Year'])
```

```
Out[29]: <AxesSubplot:>
```



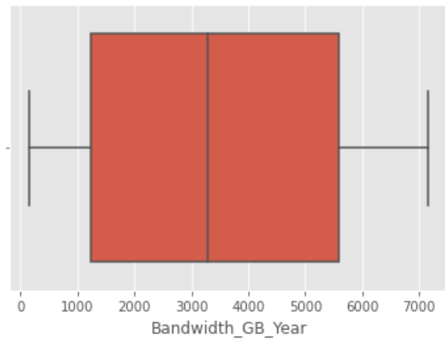
27. Seaborn boxplots for continuous & categorical variables

```
In [30]: # Create Seaborn boxplots for continuous & categorical variables
         sns.boxplot('MonthlyCharge', data = churn_df)
         plt.show()
```



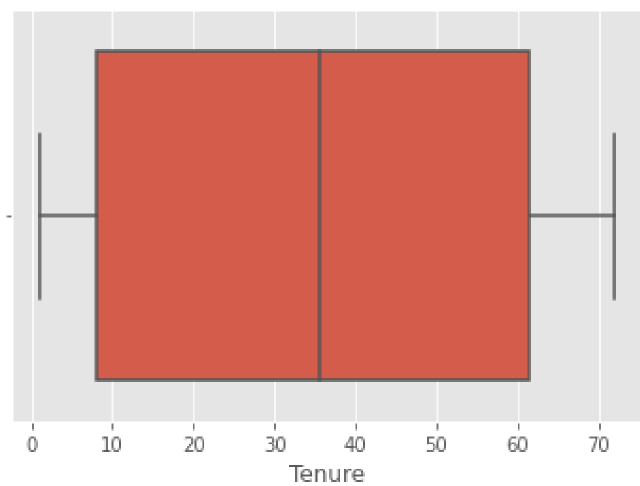
28. Boxplot

```
In [31]: # Create Seaborn boxplots for continuous & categorical variables
         sns.boxplot('Bandwidth_GB_Year', data = churn_df)
         plt.show()
```



29. Boxplot Tenure

```
In [32]: # Create Seaborn boxplots for continuous variables
         sns.boxplot('Tenure', data = churn_df)
         plt.show()
```



Anomalies

It appears that anomolies have been removed from the supplied dataset, churn_clean.csv. There are no remaining outliers.

30. missing data points within dataset

In [33]:
```
# Discover missing data points within dataset
data_nulls = churn_df.isnull().sum()
print(data_nulls)
```

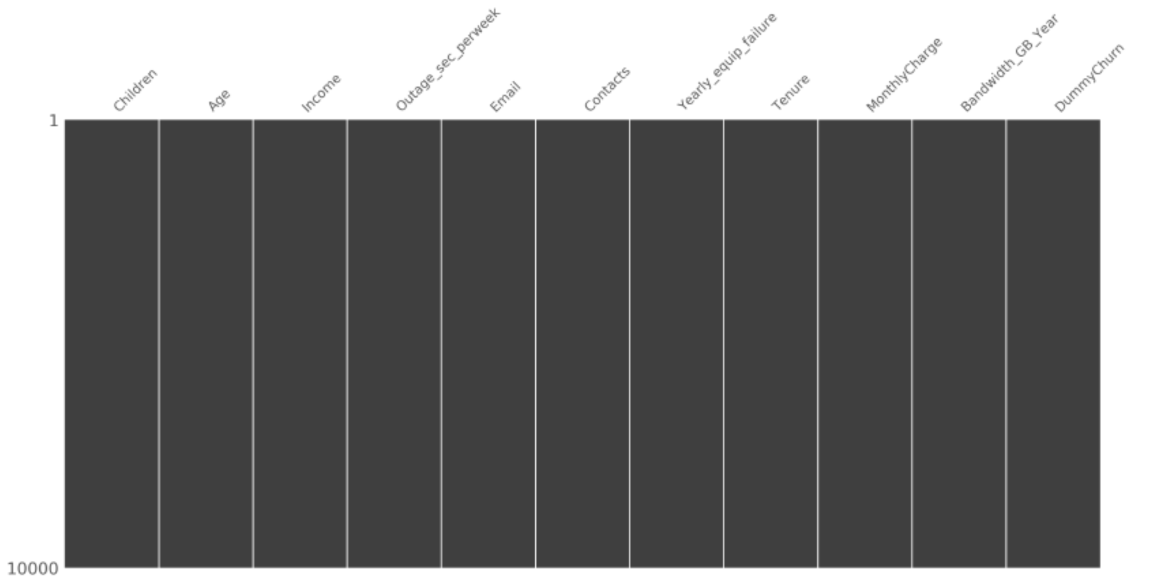
```
Children                0
Age                     0
Income                  0
Outage_sec_perweek      0
Email                   0
Contacts                0
Yearly_equip_failure    0
Tenure                  0
MonthlyCharge           0
Bandwidth_GB_Year       0
DummyChurn              0
dtype: int64
```

In [31]:
```
# Check for missing data & visualize missing values in dataset

# Install appropriate library
!pip install missingno

# Importing the libraries
import missingno as msno

# Visualize missing values as a matrix
msno.matrix(churn_df);
"""(GeeksForGeeks, p. 1)"""
```

```
Requirement already satisfied: missingno in c:\users\vreed\anaconda3\lib\site-packages (0.5.0)
Requirement already satisfied: seaborn in c:\users\vreed\anaconda3\lib\site-packages (from missingno) (0.10.0)
Requirement already satisfied: matplotlib in c:\users\vreed\anaconda3\lib\site-packages (from missingno) (3.1.3)
Requirement already satisfied: numpy in c:\users\vreed\anaconda3\lib\site-packages (from missingno) (1.18.1)
Requirement already satisfied: scipy in c:\users\vreed\anaconda3\lib\site-packages (from missingno) (1.4.1)
Requirement already satisfied: cycler>=0.10 in c:\users\vreed\anaconda3\lib\site-packages (from matplotlib->missingno) (0.10.
0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\vreed\anaconda3\lib\site-packages (from matplotlib->missingn
o) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in c:\users\vreed\anaconda3\lib\site-packages (from m
atplotlib->missingno) (2.4.6)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\vreed\anaconda3\lib\site-packages (from matplotlib->missingno)
(1.1.0)
Requirement already satisfied: six in c:\users\vreed\anaconda3\lib\site-packages (from cycler>=0.10->matplotlib->missingno)
(1.14.0)
Requirement already satisfied: setuptools in c:\users\vreed\anaconda3\lib\site-packages (from kiwisolver>=1.0.1->matplotlib->
missingno) (45.2.0.post20200210)
Requirement already satisfied: pandas>=0.22.0 in c:\users\vreed\anaconda3\lib\site-packages (from seaborn->missingno) (1.0.1)
Requirement already satisfied: pytz>=2017.2 in c:\users\vreed\anaconda3\lib\site-packages (from pandas>=0.22.0->seaborn->miss
ingno) (2019.3)
WARNING: You are using pip version 21.2.4; however, version 21.3 is available.
You should consider upgrading via the 'c:\users\vreed\anaconda3\python.exe -m pip install --upgrade pip' command.
```

Out[31]:
```
'(GeeksForGeeks, p. 1)'
```

```
findfont: Font family ['sans-serif'] not found. Falling back to DejaVu Sans.
findfont: Font family ['sans-serif'] not found. Falling back to DejaVu Sans.
findfont: Font family ['sans-serif'] not found. Falling back to DejaVu Sans.
```



In [35]:
```
churn_df.head()
```

Out[35]:

	Children	Age	Income	Outage_sec_perweek	Email	Contacts	Yearly_equip_failure	Tenure	MonthlyCharge	Bandwidth_GB_Year	DummyChurn
0	0	68	28561.99	7.978323	10	0	1	6.795513	172.455519	904.536110	0
1	1	27	21704.77	11.699080	12	0	1	1.156681	242.632554	800.982766	1
2	4	50	9609.57	10.752800	9	0	1	15.754144	159.947583	2054.706961	0
3	1	48	18925.23	14.913540	15	2	0	17.087227	119.956840	2164.579412	0
4	0	83	40074.19	8.147417	16	2	1	1.670972	149.948316	271.493436	1

In [36]:
```
# List features for analysis
features = (list(churn_df.columns[:-1]))
print('Features for analysis include: \n', features)
```

```
Features for analysis include:
 ['Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts', 'Yearly_equip_failure', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year']
```

In [37]:
```
# Extract Clean dataset
churn_df.to_csv('C:/Kailash/Rekha/D212/data/churn_prepared_pca.csv')
```

C2: Variable Standardization in Datasets:

In [38]:
```
# Load clean, prepared dataset
churn_df = pd.read_csv('C:/Kailash/Rekha/D212/data/churn_prepared_pca.csv')
```

In [40]:
```
# Define matrix of features as X
X = churn_df[['Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts', 'Yearly_equip_failure', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year']]
```

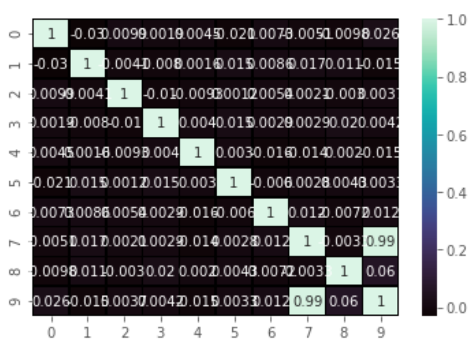
In [41]:
```
# Import StandardScaler class from Scikit Learn
from sklearn.preprocessing import StandardScaler
# Standardize the data
X_standardized = StandardScaler().fit_transform(X)
```

In [42]:
```
# Create covariance matrix
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
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]]
```

Visualization of Feature Scaling

In [43]:
```
# Visualize feature scaling
sns.heatmap(covariance_matrix, annot=True, cmap="mako", linecolor='black', linewidths=0.5)
plt.show()
```



```python
In [44]: # Perform Eigendecomposition on covariance matrix
covariance_matrix = np.cov(X_standardized.T)
eigen_values, eigen_vectors = np.linalg.eig(covariance_matrix)
# Print Eigenvectors and Eigenvalues
print('Eigenvectors: \n%s' %eigen_vectors)
print('Eigenvalues: \n%s' %eigen_values)
```

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

PART IV. Analysis/Research

D1. PCA (Principal Component Analysis):

In [46]:
```
# List descending sorted Eigenvalues
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)
# Print list of Eigenvalues, descending
print('Eigenvalues:')
for i in eigen_pairs:
    print(i[0])
```

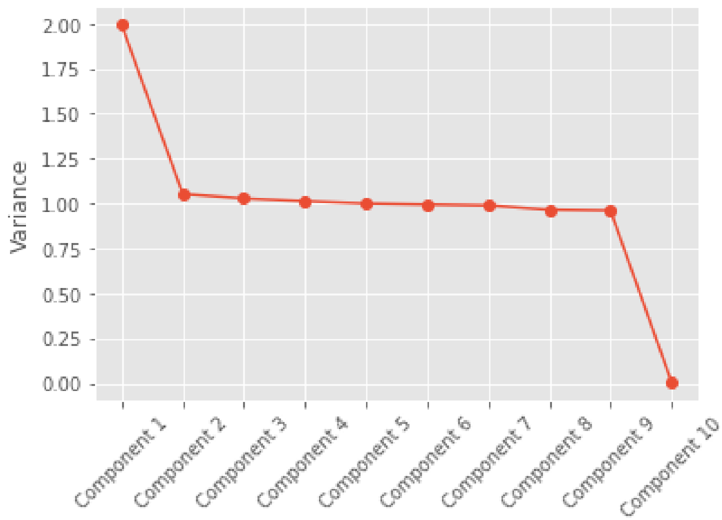
```
Eigenvalues:
1.9943331101364745
1.0533346256283747
1.0275539108057439
1.0125585769497143
0.9997955481739359
0.9937799880491783
0.9890585843412285
0.9647674706143192
0.9603505880457003
0.005467697265331584
```

In [47]:
```
# Fit standardized matrix of features to PCA class
pca = PCA().fit(X_standardized)
# Print 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]
```

D2. Total Number of Components is Determined:

In [49]:
```
# Perform the scree 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()

# Visualize scree plot
screeplot(pca, X_standardized)
```



Examining a scree plot to determine the number of primary components is a regularly used method. By looking for a location on the scree plot where the proportion of variation explained by each consecutive principal component decreases off. In the scree plot, this is referred to as an elbow.

Looking at the plot, we can observe that after the eighth principal component, there is a significant drop (elbow in the scree plot).

The explained variances are concentrated in the 2nd principal component as from 2nd PC till 9th PC the variance is almost consistent i.e. equal to 1.0.

As a result, we would choose the first seven principal components to represent our data set based on the scree plot.

D3. Total Component Variance:

In [44]:
```
# 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, X_standardized)
summary.standard_deviation**2
```

```
Component importance:
```

Out[44]:

	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

Looking at the elbow rule, we can observe that after the eighth principal component, there is a significant drop. Out of the 10 principal components, the first 7 principal components will be used for PCA. The above table also specifies the variance of each of those 7 principal components.

D4. Variance in total Components Captured:

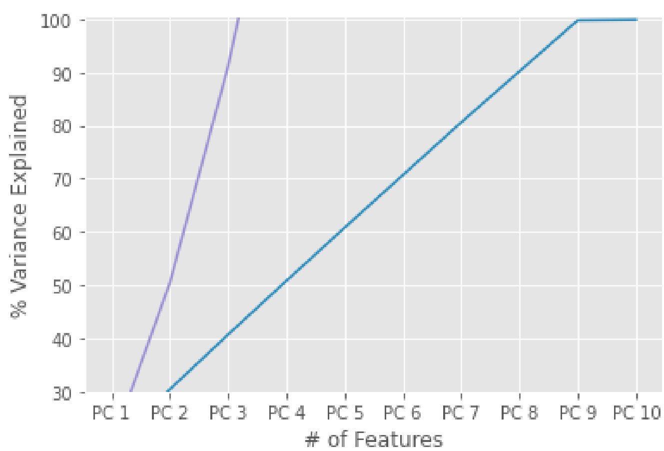
In [51]:
```
# Identify total variance captured by the principal components
np.sum(summary.standard_deviation**2)
```

Out[51]:
```
Standard Deviation    10.0
dtype: float64
```

In [52]:
```
# Visualize total variance captured by components
var = np.cumsum(np.round(summary, decimals=3)*100)
```

In [53]:
```
plt.ylabel('% Variance Explained')
plt.xlabel('# of Features')
plt.title('PCA Analysis')
plt.ylim(30,100.5)

# plt.style.context('seaborn-whitegrid')
plt.plot(var)
```

Out[53]:
```
[<matplotlib.lines.Line2D at 0x20beed8fdc0>,
 <matplotlib.lines.Line2D at 0x20beed8fc70>,
 <matplotlib.lines.Line2D at 0x20beed8fe80>]
```



Looking at the elbow rule, we can observe that after the eighth principal component, there is a significant drop. Out of the 10 principal components, the first 7 principal components will be used for PCA. The variance of 4 principal components is more than or equal to 1.0 from the scree plot above account for 50% of total variation, as shown in the above visualization. We can also see that when we plot the other components with scree plot values less than 1.0 against the explained variance, we get to 100% explained variance after nine components.

D5. Data Analysis Summary:

We performed PCA analysis to compress datasets to a more manageable size with a stronger focus on the main numerical variables that influence customer churn. Because we now have fewer variables and a better understanding of their interactions than before the investigation, we may be able to prevent overfitting.

The steps outlined above are as follows:

- List order of variance
- Scree plot components
- Created a covariance matrix
- Standardized the variables
- Selected continuous variables

Decision-makers and marketers must be aware that after scaling, our predictor factors result in a total of 10.0 variance. There are four components that have a value greater than one. We might not be able to go to more advanced data mining technologies that can extract more meaning from these discovered components.

Finally, we should investigate the characteristics that are common among individuals who have left the company in the past and try to limit their risk of recurrence with any future consumer.

PART VI. Documentary Evidence

E. Panopto recording:

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=72ff5581-75fd-4b94-84f9-ae5a0158f2ba

F. Third Party Evidence:

Title: (Visualize missing values (NaN) values using Missingno Library | Python |), GeeksForGeeks.
URL: https://www.geeksforgeeks.org/python-visualize-missing-values-nan-values-using-missingno-library/
Date: July 4th, 2019

Title: SuperDataScience, (Machine Learning A-Z: Hands-On Python & R in Data Science)
Date: August 15th, 2021
URL: https://www.superdatascience.com/

Title: PCA, (Principal Component Analysis (PCA) in Python)
Author: Sharma A.
URL: https://www.datacamp.com/pal-component-analysis-in-python
Date: 2021

Title: (Python code examples of explained variance in PCA)
Author: Y Zhang.
URL: https://zhang-yang.medium.com/python-code-examples-of-explained-variance-in-pcaa19cfa73a257
Date: 2018

G. Sources:

Title: (Machine Learning A-Z: Hands-On Python & R in Data Science), SuperDataScience
Date: August 15th, 2021
URL: https://www.superdatascience.com/