D212 Task 2: Dimensionality Reduction Methods

Data Preperation

In [16]:
```python
# import Libraries and packages

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

In [17]:
```python
#load data set
df = pd.read_csv('churn_clean.csv')
df.head()
```

Out[17]:

	CaseOrder	Customer_id	Interaction	UID	City	State	County	Zip	Lat	L
0	1	K409198	aa90260b-4141-4a24-8e36-b04ce1f4f77b	e885b299883d4f9fb18e39c75155d990	Point Baker	AK	Prince of Wales-Hyder	99927	56.25100	-133.375
1	2	S120509	fb76459f-c047-4a9d-8af9-e0f7d4ac2524	f2de8bef964785f41a2959829830fb8a	West Branch	MI	Ogemaw	48661	44.32893	-84.240
2	3	K191035	344d114c-3736-4be5-98f7-c72c281e2d35	f1784cfa9f6d92ae816197eb175d3c71	Yamhill	OR	Yamhill	97148	45.35589	-123.246
3	4	D90850	abfa2b40-2d43-4994-b15a-989b8c79e311	dc8a365077241bb5cd5ccd305136b05e	Del Mar	CA	San Diego	92014	32.96687	-117.247
4	5	K662701	68a861fd-0d20-4e51-a587-8a90407ee574	aabb64a116e83fdc4befc1fbab1663f9	Needville	TX	Fort Bend	77461	29.38012	-95.806

5 rows × 50 columns

In [18]:
```python
# create DataFrame of continuous features
churn_df = df[['Population', 'Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email',\
            'Contacts', 'Yearly_equip_failure', 'Tenure', 'MonthlyCharge',\
            'Bandwidth_GB_Year']]
churn_df.head()
```

Out[18]:

	Population	Children	Age	Income	Outage_sec_perweek	Email	Contacts	Yearly_equip_failure	Tenure	MonthlyCharge	Bandw
0	38	0	68	28561.99	7.978323	10	0	1	6.795513	172.455519	
1	10446	1	27	21704.77	11.699080	12	0	1	1.156681	242.632554	
2	3735	4	50	9609.57	10.752800	9	0	1	15.754144	159.947583	
3	13863	1	48	18925.23	14.913540	15	2	0	17.087227	119.956840	
4	11352	0	83	40074.19	8.147417	16	2	1	1.670972	149.948316	

In [19]:
```python
# Standardize the data set
scaler = StandardScaler()
scaled_data = scaler.fit_transform(churn_df)
scaled_data = pd.DataFrame(scaled_data, columns = churn_df.columns)
scaled_data.head()
```

Out[19]:

	Population	Children	Age	Income	Outage_sec_perweek	Email	Contacts	Yearly_equip_failure	Tenure	MonthlyC
0	-0.673405	-0.972338	0.720925	-0.398778	-0.679978	-0.666282	-1.005852	0.946658	-1.048746	-0.0
1	0.047772	-0.506592	-1.259957	-0.641954	0.570331	-0.005288	-1.005852	0.946658	-1.262001	1.6
2	-0.417238	0.890646	-0.148730	-1.070885	0.252347	-0.996779	-1.005852	0.946658	-0.709940	-0.2
3	0.284537	-0.506592	-0.245359	-0.740525	1.650506	0.986203	1.017588	-0.625864	-0.659524	-1.2
4	0.110549	-0.972338	1.445638	0.009478	-0.623156	1.316700	1.017588	0.946658	-1.242551	-0.5

In [20]:
```python
# create csv of prepared data
scaled_data.to_csv('PCA_Prepared_Data.csv')
```

Analysis

Code from 'All You Need to Know About Principal Component Analysis' https://www.edureka.co/blog/principal-component-analysis/

In [21]:
```python
# compute the covariance matrix
mean_vec = np.mean(scaled_data, axis = 0)
cov_matrix = (scaled_data - mean_vec).T.dot((scaled_data - mean_vec))/(scaled_data.shape[0] - 1)
print('Covariance Matrix: \n%s' % cov_matrix)
```

```
Covariance Matrix: 
                      Population  Children       Age    Income  \
Population              1.000100 -0.005877  0.010539 -0.008639   
Children               -0.005877  1.000100 -0.029735  0.009943   
Age                     0.010539 -0.029735  1.000100 -0.004091   
Income                 -0.008639  0.009943 -0.004091  1.000100   
Outage_sec_perweek      0.005484  0.001889 -0.008048 -0.010012   
Email                   0.017963  0.004479  0.001588 -0.009268   
Contacts                0.004019 -0.020778  0.015069  0.001233   
Yearly_equip_failure   -0.004483  0.007321  0.008578  0.005424   
Tenure                 -0.003560 -0.005092  0.016981  0.002115   
MonthlyCharge          -0.004779 -0.009782  0.010730 -0.003014   
Bandwidth_GB_Year      -0.003902  0.025587 -0.014725  0.003674   

                      Outage_sec_perweek     Email  Contacts  \
Population                      0.005484  0.017963  0.004019   
Children                        0.001889  0.004479 -0.020778   
Age                            -0.008048  0.001588  0.015069   
Income                         -0.010012 -0.009268  0.001233   
Outage_sec_perweek              1.000100  0.003994  0.015093   
Email                           0.003994  1.000100  0.003041   
Contacts                        0.015093  0.003041  1.000100   
Yearly_equip_failure            0.002909 -0.016356 -0.006033   
Tenure                          0.002932 -0.014469  0.002820   
MonthlyCharge                   0.020498  0.001997  0.004259   
Bandwidth_GB_Year               0.004176 -0.014581  0.003299   

                      Yearly_equip_failure    Tenure  MonthlyCharge  \
Population                       -0.004483 -0.003560      -0.004779   
Children                          0.007321 -0.005092      -0.009782   
Age                               0.008578  0.016981       0.010730   
Income                            0.005424  0.002115      -0.003014   
Outage_sec_perweek                0.002909  0.002932       0.020498   
Email                            -0.016356 -0.014469       0.001997   
Contacts                         -0.006033  0.002820       0.004259   
Yearly_equip_failure              1.000100  0.012436      -0.007173   
Tenure                            0.012436  1.000100      -0.003337   
MonthlyCharge                    -0.007173 -0.003337       1.000100   
Bandwidth_GB_Year                 0.012035  0.991594       0.060412   

                      Bandwidth_GB_Year  
Population                    -0.003902  
Children                       0.025587  
Age                           -0.014725  
Income                         0.003674  
Outage_sec_perweek             0.004176  
Email                         -0.014581  
Contacts                       0.003299  
Yearly_equip_failure           0.012035  
Tenure                         0.991594  
MonthlyCharge                  0.060412  
Bandwidth_GB_Year              1.000100  
```

In [22]:
```python
# Calculate eigenvectors and eigenvalues

cov_matrix = np.cov(scaled_data.T)
e_values, e_vectors = np.linalg.eig(cov_matrix)
print('Eigenvectors n%s' %e_vectors)
print('Eigenvalues n%s' %e_values)
```

```
Eigenvectors n[[ 6.00673770e-03  3.46561134e-04 -2.75637177e-01 -2.67635705e-01
   4.22474142e-01  1.77039566e-01 -1.38379385e-01 -6.14035223e-01
   3.94930492e-01  1.75784423e-01  2.43871485e-01]
 [-1.41719844e-02  2.15873620e-02  5.17059144e-01 -3.43247708e-01
  -7.62039747e-02 -1.88740212e-01 -6.72255872e-01  1.84797628e-01
   1.64533239e-01  1.80458617e-01  1.64375971e-01]
 [-1.64304686e-03 -2.23693437e-02 -4.57819130e-01  4.18590404e-01
   1.98972157e-01 -5.63399820e-01 -2.73258508e-01  2.03859075e-01
   1.26613617e-01  3.43080738e-01 -8.77907636e-02]
 [-4.41376460e-03  9.38264094e-04  2.54937535e-01  2.68952295e-01
  -7.71331021e-02 -1.00837059e-01  2.54841612e-01 -1.82476676e-01
  -3.18506370e-01  4.84459329e-01  6.41984729e-01]
 [-5.84991584e-03 -2.82658712e-04 -2.12600279e-01 -3.36482556e-01
  -5.87557904e-01 -4.82900254e-01  2.85041117e-01 -9.83866405e-02
   3.45666884e-01 -1.16979068e-01  1.97563402e-01]
 [ 2.08872409e-02 -2.52220640e-04 -1.95432035e-01 -5.26028754e-01
   3.11467996e-01  9.25560761e-02  3.25796174e-01  5.97954779e-01
  -4.50591138e-02  3.08043642e-01  1.35906274e-01]
 [-4.14789624e-03  9.41483338e-04 -4.27624091e-01  1.07344120e-01
  -1.62242058e-01  2.87738545e-01 -3.66164103e-01  2.45083050e-01
  -1.23928024e-01 -3.76069414e-01  5.87837238e-01]
 [-1.75930701e-02  9.67166778e-05  1.69548640e-01  3.85120068e-01
  -1.35014646e-01  3.72773827e-01  1.59743557e-01  2.77655477e-01
   7.29849851e-01  1.58047870e-01  7.18782549e-02]
 [-7.05404896e-01  7.05262739e-01 -6.55380787e-03  7.86501842e-03
   4.85004295e-02 -2.77736623e-02  2.70414102e-02  7.48150534e-03
   2.91372717e-04 -3.01408363e-02  9.17372945e-03]
 [-4.04475870e-02  4.57563347e-02 -2.94988142e-01 -1.12548153e-01
  -5.30093491e-01  3.75602534e-01 -2.03935256e-01 -9.75204307e-02
  -1.72686461e-01  5.55191858e-01 -2.92147996e-01]
 [-7.06902821e-01 -7.06783125e-01  4.45706718e-03 -2.24263989e-02
   5.44756417e-03  9.21611733e-03  1.47071175e-03 -8.59413682e-06
  -1.02872208e-02  7.06771385e-04 -3.23054624e-04]]
Eigenvalues n[1.99436896 0.00546758 1.05744462 1.02956846 1.01874438 0.9594893
 0.96418654 0.97889558 1.00339009 0.9915275  0.9980171 ]
```

In [23]:
```python
# sort eigenvalues in descending order
e_pairs = [(np.abs(e_values[i]), e_vectors[:,i]) for i in range(len(e_values))]
e_pairs.sort(key = lambda x: x[0], reverse = True)
print('Eigenvalues in decending order:')
for i in e_pairs:
    print(i[0])
```

```
Eigenvalues in decending order:
1.9943689624714998
1.0574446232505101
1.0295684552914954
1.0187443771822249
1.0033900851266027
0.9980170974473459
0.9915275045867098
0.9788955839660184
0.9641865424442037
0.959489300362716
0.005467577881675837
```

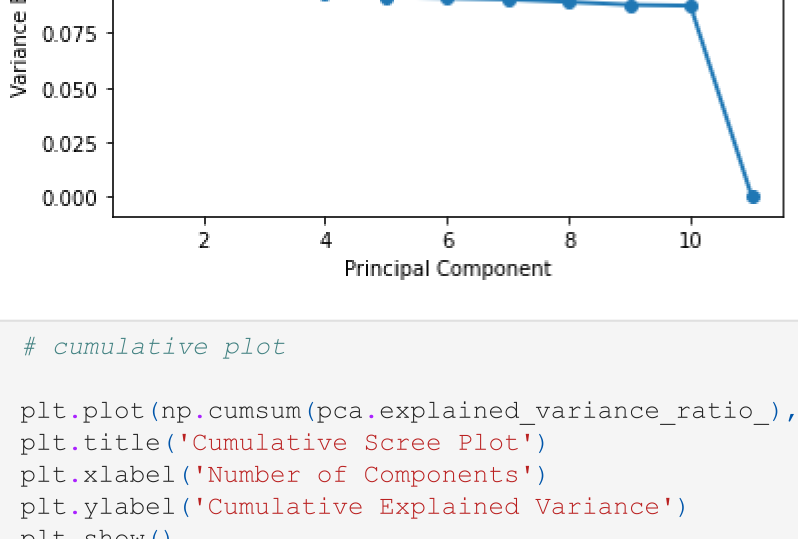
In [25]:
```python
# fit data to PCA class
pca = PCA()
components = pca.fit_transform(scaled_data)

#DataFrame of Principal Components
pc_df = pd.DataFrame(data = components , columns = ['PC 1', 'PC 2', 'PC 3', 'PC 4', 'PC 5', 'PC 6','PC 7', 'PC
print(pc_df)
```

```
          PC 1      PC 2      PC 3      PC 4      PC 5      PC 6      PC 7  \
0    -1.532639  0.119512 -1.562116  0.136206  0.414997 -1.399578  0.191106   
1    -1.659019  0.130539  0.638301 -1.375658  0.723705 -1.271899  0.575596   
2    -0.900522  1.191402 -0.193081 -0.495760  1.308798 -1.158699 -0.434070   
3    -0.942314 -1.138090  1.264619  0.039044  0.394403  0.898011 -1.516688   
4    -1.929748 -1.434578 -0.984405  1.102943  0.459296  0.611698  0.333212   
...        ...       ...       ...       ...       ...       ...       ...   
9995  1.897402  0.789544  0.484892 -0.372859 -1.157516  1.016570 -0.920543   
9996  1.434856 -1.508304  2.101618  2.366782  0.867436  1.420222  1.171667   
9997  0.578813  0.799305 -0.693559  0.471070 -1.131182 -1.009988 -0.023901   
9998  2.002781 -1.589854  1.860081 -0.311399  0.216009 -0.461117  0.605280   
9999  1.551767 -0.898844  2.112172 -0.290109 -0.341160 -0.632862 -0.120582   

          PC 8      PC 9     PC 10     PC 11  
0    -0.130913 -0.527627  0.045657 -0.026622  
1     0.474031 -0.826669  1.302704 -0.038360  
2    -0.004835  0.466449 -0.297649  0.060825  
3    -0.434394 -0.730167 -0.734906  0.130009  
4    -1.448501 -0.347708  0.279139 -0.056541  
...        ...       ...       ...       ...  
9995 -0.197212  0.091115  0.591052  0.081237  
9996  2.015353  1.829908  1.784128 -0.026353  
9997  0.482031 -0.303914  0.032670 -0.086949  
9998  1.201074 -0.039065  1.045767 -0.069400  
9999 -0.414292 -0.790223  0.927587 -0.033821  

[10000 rows x 11 columns]
```

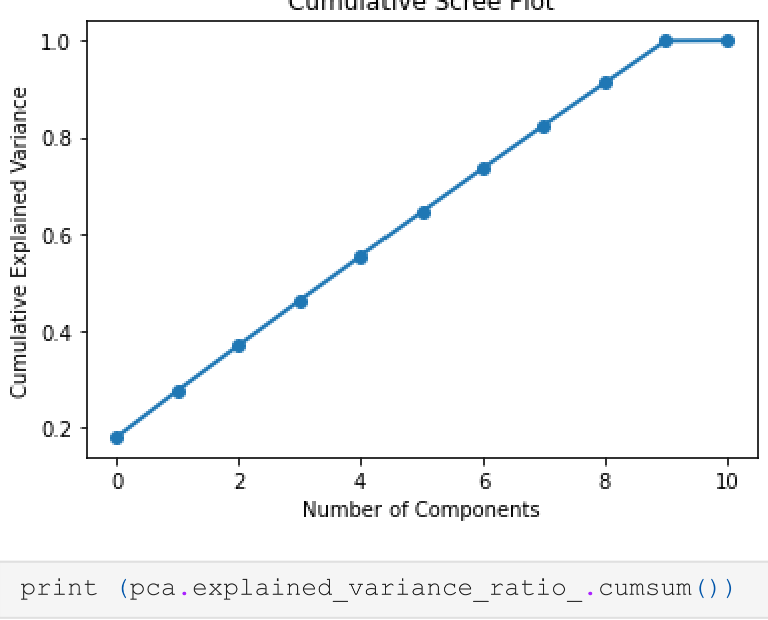
In [26]:
```python
# Variance explained by each principal component
print(pca.explained_variance_ratio_)
```

```
[0.18128814 0.09612172 0.09358777 0.09260386 0.09120816 0.09071975
 0.09012985 0.08898161 0.08764456 0.08721758 0.000497  ]
```

In [31]:
```python
# create scree plot
# Code from: How to Create a Scree Plot in Python
# https://www.statology.org/scree-plot-python/

pc_values = np.arange(pca.n_components_) + 1
plt.plot(pc_values, pca.explained_variance_ratio_, 'o-', linewidth = 2)
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.show()
```



In [32]:
```python
# cumulative plot

plt.plot(np.cumsum(pca.explained_variance_ratio_), 'o-', linewidth = 2)
plt.title('Cumulative Scree Plot')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.show()
```



In [29]:
```python
print (pca.explained_variance_ratio_.cumsum())
```

```
[0.18128814 0.27740985 0.37099763 0.46360149 0.55480965 0.6455294
 0.73565925 0.82464086 0.91228542 0.999503   1.        ]
```

In [ ]: