Project Purpose

Organizations keep detailed records with many attributes to describe each customer. Characteristics like the population where the customer lives and the number of children they have could factor into their decision to terminate services with the company. The question I would like to answer with this report is: "How can I optimize the customer data to best predict whether a customer will terminate their services?"

One goal of this analysis project is to create a list of components that contribute to at least 80% of the variability in the data. Ultimately, I would like to use this information to go back to some of the prior course tasks to see if using the transformations created using this PCA can create statistical models with higher accuracy.

Explanation of Method

In large datasets that contain many variables or "features", models can be affected by extra information that does not add value to the analysis technique (Wilson, n.d.). This can happen in the case where the features are insignificant to the analysis or are highly correlated with other features. One way to eliminate these issues is to drop those features from the dataset before creating a model, but this can result in the loss of data that might be helpful in prediction.

Principal Component Analysis (PCA) is a method that uses feature extraction. The algorithm performs a linear transformation on correlated variables and combines them into a new set of uncorrelated features (Loukas, 2020). The feature combinations, or components, are ordered based on how much variability they contribute to the dataset. The most useful components that provide the majority of the variability can be kept to create models for regression and classification, which will be free from the influence of multicollinearity (Brems, 2019). The transformed components keep the majority of the information of the original dataset. This can be seen in the case of image compression using PCA, where an almost exact replica of the original pictures can be seen after applying the inverse of the transformation (Boeye, n.d.).

One downside of using PCA is that the transformed components are no longer the clear distinct variables of the original dataset. Therefore, PCA should not be used when there is a desire to keep the independent variables available for interpretation (Brems, 2019).

Assumptions

One assumption of PCA is that the data has been standardized. While the variables being used in the PCA analysis are all numeric, many of them have large variations in the range of values. A variable like population can range into the thousands, while 'Yearly_equip_failure' has a maximum value of 6. Standardizing the data ensures that all features are using a similar range. This is important for PCA because it groups components to maximize the variance (Pedregosa, 2011). If the features are not scaled, PCA might incorrectly group the data causing it to underperform (Boeye, n.d.).

Preprocessing

In [1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

import warnings
warnings.filterwarnings('ignore') # Ignore warning messages for readability
```

In [2]:
```
# Read in data set and view head
df = pd.read_csv('churn_clean.csv')
pd.options.display.max_columns = None
df.head()
```

Out[2]:

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

In [4]:
```
# View column names
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 50 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   CaseOrder             10000 non-null  int64  
 1   Customer_id           10000 non-null  object 
 2   Interaction           10000 non-null  object 
 3   UID                   10000 non-null  object 
 4   City                  10000 non-null  object 
 5   State                 10000 non-null  object 
 6   County                10000 non-null  object 
 7   Zip                   10000 non-null  int64  
 8   Lat                   10000 non-null  float64
 9   Lng                   10000 non-null  float64
 10  Population            10000 non-null  int64  
 11  Area                  10000 non-null  object 
 12  TimeZone              10000 non-null  object 
 13  Job                   10000 non-null  object 
 14  Children              10000 non-null  int64  
 15  Age                   10000 non-null  int64  
 16  Income                10000 non-null  float64
 17  Marital               10000 non-null  object 
 18  Gender                10000 non-null  object 
 19  Churn                 10000 non-null  object 
 20  Outage_sec_perweek    10000 non-null  float64
 21  Email                 10000 non-null  int64  
 22  Contacts              10000 non-null  int64  
 23  Yearly_equip_failure  10000 non-null  int64  
 24  Techie                10000 non-null  object 
 25  Contract              10000 non-null  object 
 26  Port_modem            10000 non-null  object 
 27  Tablet                10000 non-null  object 
 28  InternetService       10000 non-null  object 
 29  Phone                 10000 non-null  object 
 30  Multiple              10000 non-null  object 
 31  OnlineSecurity        10000 non-null  object 
 32  OnlineBackup          10000 non-null  object 
 33  DeviceProtection      10000 non-null  object 
 34  TechSupport           10000 non-null  object 
 35  StreamingTV           10000 non-null  object 
 36  StreamingMovies       10000 non-null  object 
 37  PaperlessBilling      10000 non-null  object 
 38  PaymentMethod         10000 non-null  object 
 39  Tenure                10000 non-null  float64
 40  MonthlyCharge         10000 non-null  float64
 41  Bandwidth_GB_Year     10000 non-null  float64
 42  Item1                 10000 non-null  int64  
 43  Item2                 10000 non-null  int64  
 44  Item3                 10000 non-null  int64  
 45  Item4                 10000 non-null  int64  
 46  Item5                 10000 non-null  int64  
 47  Item6                 10000 non-null  int64  
 48  Item7                 10000 non-null  int64  
 49  Item8                 10000 non-null  int64  
dtypes: float64(7), int64(16), object(27)
memory usage: 3.8+ MB
```

- Numeric variables: Lat, Lng, Population, Children, Age, Income, Outage_sec_perweek, Email, Contacts, Yearly_equip_failure, Tenure, MonthlyCharge, Bandwidth_GB_Year, Item1, Item2, Item3, Item4, Item5, Item6, Item7, Item8

In [5]:
```
# Create new dataframe for variables to be used in PCA and view head
df_PCA = df[['Lat', 'Lng', 'Population', 'Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contact
s',
            'Yearly_equip_failure', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year', 'Item1', 'Item2', 'Item3',
            'Item4', 'Item5', 'Item6', 'Item7', 'Item8']]
df_PCA.head()
```

Out[5]:

	Lat	Lng	Population	Children	Age	Income	Outage_sec_perweek	Email	Contacts	Yearly_equip_failure	Tenure	Mc
0	56.25100	-133.37571	38	0	68	28561.99	7.978323	10	0	1	6.795513	
1	44.32893	-84.24080	10446	1	27	21704.77	11.699080	12	0	1	1.156681	
2	45.35589	-123.24657	3735	4	50	9609.57	10.752800	9	0	1	15.754144	
3	32.96687	-117.24798	13863	1	48	18925.23	14.913540	15	2	0	17.087227	
4	29.38012	-95.80673	11352	0	83	40074.19	8.147417	16	2	1	1.670972	

In [6]:
```
# Scale the data
scaler = StandardScaler()
PCA_std = scaler.fit_transform(df_PCA)
```

In [7]:
```
# Convert standardized numpy array to dataframe and export to Excel

# Create an array of column names for dataframe
columns = df_PCA.columns

# Convert numpy array to dataframe
df_PCA_std = pd.DataFrame(PCA_std, columns=columns)

# Save standardized dataframe to Excel file (ref E3)
df_PCA_std.to_excel('df_PCA_std.xlsx', index = False, encoding = 'utf-8')
```

In [8]:
```
# View standardized data
df_PCA_std.head()
```

Out[8]:

	Lat	Lng	Population	Children	Age	Income	Outage_sec_perweek	Email	Contacts	Yearly_equip_failure	T
0	3.217410	-2.810432	-0.673405	-0.972338	0.720925	-0.398778	-0.679978	-0.666282	-1.005852	0.946658	-1.0
1	1.024691	0.431644	0.047772	-0.506592	-1.259957	-0.641954	0.570331	-0.005288	-1.005852	0.946658	-1.2
2	1.213570	-2.142079	-0.417238	0.890646	-0.148730	-1.070885	0.252347	-0.996779	-1.005852	0.946658	-0.7
3	-1.065031	-1.746273	0.284537	-0.506592	-0.245359	-0.740525	1.650506	0.986203	1.017588	-0.625864	-0.6
4	-1.724710	-0.331512	0.110549	-0.972338	1.445638	0.009478	-0.623156	1.316700	1.017588	0.946658	-1.2

Principal Component Analysis

In [9]:
```
# Create a PCA model and fit to the data
pca = PCA()
pca.fit(PCA_std)
```

Out[9]: PCA()

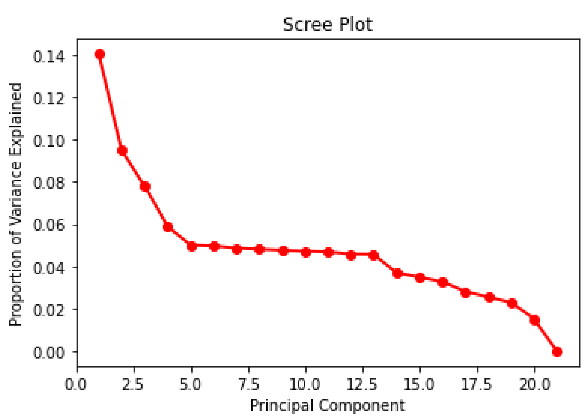
In [10]:
```
# Display matrix of all principal components
print(pca.components_)
```

```
[[-1.11224657e-03  8.05819399e-03 -2.18128395e-03  4.12840237e-03
   6.50872306e-03  1.02244405e-03 -1.74936656e-02  8.79175823e-03
  -8.72526329e-03 -7.70545082e-03 -1.62662110e-02  9.79819920e-04
  -1.67899389e-02  4.58718711e-01  4.33833886e-01  4.00518325e-01
   1.45752051e-01 -1.75652341e-01  4.05012316e-01  3.58210511e-01
   3.08715988e-01]
 [-2.31207748e-02  9.44743876e-03 -7.70736959e-04  1.59568291e-02
   5.21341098e-04  5.80777653e-03  3.90861691e-03 -1.97411527e-02
   3.45914531e-03  1.76711247e-02  7.02098205e-01  3.98835005e-02
   7.03617405e-01  3.13345268e-02  3.86172310e-02  3.55977192e-02
  -3.98140691e-02  5.65295150e-02 -6.73644002e-03  1.73660383e-03
  -1.33498148e-02]
 [-7.37976389e-03  2.24453811e-02  1.56158359e-02  2.87838361e-02
  -2.88357024e-02  2.56219223e-02 -1.41664595e-02 -2.77271616e-03
  -1.15239627e-02  8.04308341e-03 -6.36930267e-02 -9.13849129e-03
  -6.27244201e-02  2.80923749e-01  2.81971006e-01  2.80414776e-01
  -5.68295206e-01  5.86829296e-01 -1.83774772e-01 -1.81488127e-01
  -1.31542558e-01]
 [-7.13406868e-01  1.77971762e-01  6.52679273e-01 -1.68847906e-02
   5.52936602e-02 -5.59379729e-02  1.39369925e-02  1.49799462e-01
   2.93057182e-02 -7.24362705e-03 -7.69567233e-03 -2.96406218e-03
  -9.17706459e-03 -1.11986107e-02 -1.89806979e-02 -3.38103667e-03
  -5.33908607e-03 -8.55354310e-03  1.25649703e-02 -2.02500673e-02
   4.52830776e-02]
 [-2.50422725e-02 -3.38391754e-01  1.73323338e-01  4.13387893e-01
  -4.26834116e-01  1.86963619e-01 -2.59856119e-01 -8.84086612e-02
  -4.38742106e-01  1.50265224e-01  9.77048768e-03 -4.16994472e-01
   9.11580056e-03 -1.73778396e-02 -2.03354652e-02  3.04114392e-04
   9.23781856e-03 -2.89680424e-02  1.20137988e-02  1.99274162e-02
  -1.14267566e-02]
 [ 1.12068892e-01 -7.10966944e-01  3.07611644e-01 -4.93891165e-01
   2.63319214e-01 -3.54400442e-02 -1.15987601e-01 -1.46478819e-01
   1.41564089e-01  5.21736509e-02  2.51265282e-02 -1.07632173e-01
  -4.36326212e-03 -2.03268982e-03  1.82057944e-02  3.29976535e-03
  -1.35905018e-02  4.26021458e-02  1.58857556e-02 -6.08817001e-03
   1.66704201e-02]
 [-9.85948738e-02  3.54154145e-01 -1.22629594e-01 -9.71389067e-02
   4.23545497e-01  3.24461248e-01 -4.57488302e-01 -3.45697021e-01
   2.03151340e-02  4.15508498e-01  9.25337425e-03 -2.28323849e-01
  -2.13631613e-02 -2.23932307e-03 -1.65173771e-02 -1.29572657e-02
   5.82711524e-03  3.13718529e-03 -4.96750891e-03  2.53284511e-02
  -4.74373080e-03]
 [-2.88079133e-02 -9.22076938e-02  9.75077176e-02  1.36314473e-01
  -7.54783772e-02  9.23387314e-02  5.84093492e-01 -4.26344910e-01
   2.09262664e-02  5.81381558e-01 -3.63611528e-02  2.80071617e-01
  -1.11505845e-02  1.52280025e-02  1.41407935e-02 -2.61799500e-02
  -1.25441246e-02 -1.40659062e-02  7.99869747e-03 -2.69090109e-02
   6.95004389e-02]
 [-1.03318844e-02 -6.43241650e-02  5.45989450e-02  6.65117982e-02
  -1.78439360e-01  7.79760095e-01  9.03400635e-02  3.64364677e-02
   5.15860109e-01 -2.54131398e-01 -4.25299151e-03 -2.03439595e-02
   3.87559094e-03 -2.20079172e-02  5.43848953e-04 -3.59073719e-02
  -2.86553375e-02 -2.50514979e-03  1.91492148e-02  6.98936718e-02
  -9.08692087e-04]
 [-2.22917130e-02 -6.62068245e-02  6.78289760e-02 -7.60095868e-02
   9.67579436e-02  3.32630907e-01 -2.10242990e-01 -1.35628288e-01
  -5.25188716e-01 -2.48980730e-01 -3.50376091e-02  6.79170154e-01
   3.70404896e-03 -1.09375233e-02 -9.91412277e-03 -1.15181876e-02
  -1.06132083e-02 -3.01280645e-03 -3.23140169e-03 -1.24822905e-02
   3.42394890e-02]
 [ 8.75203783e-02 -1.73572053e-01 -2.56816670e-02  1.87103577e-01
   3.45448861e-01  2.05332053e-01  3.45537564e-02  7.51639906e-01
  -8.44668676e-02  4.20133201e-01  4.50767008e-04  1.11057380e-01
   2.31531449e-03 -4.49985583e-03 -2.17888972e-03 -4.23020662e-03
  -2.17178814e-02 -7.60864271e-03  2.17686659e-02  1.38713577e-02
  -4.08454284e-02]
 [-1.07902901e-02 -9.49724962e-02  2.72184099e-02  1.76812131e-01
  -3.23264116e-01 -2.38138329e-01 -5.51538463e-01  5.45273921e-03
   4.54008038e-01  2.66143714e-01 -3.88475202e-02  4.52563487e-01
   6.58142243e-03  2.48504112e-02 -8.82372677e-04 -7.59021706e-03
   2.08184841e-02 -1.38705792e-02  1.75926137e-02  1.47727127e-02
  -9.09673819e-02]
 [ 5.77190111e-02 -1.58148818e-01  1.08331171e-01  6.90935074e-01
   5.38840567e-01 -1.46506229e-01 -4.92638284e-03 -2.36984791e-01
   1.60662993e-01 -2.94976651e-01 -8.38045154e-03  1.32186401e-02
  -3.17965068e-03 -7.65851876e-03  1.82777814e-02 -2.00472341e-02
```

```
 -1.09989619e-02  2.46584846e-03 -1.53517803e-03 -1.04835977e-02
  2.04552541e-02]
[ 9.52240805e-02  7.13979256e-02  1.67885112e-01 -1.94795561e-02
  3.59718838e-02  2.41107310e-02  8.15768416e-02 -5.73434939e-02
 -4.56563347e-02 -1.30807842e-02 -4.23542432e-03  4.38985819e-03
 -8.82985685e-03  7.19721927e-02  1.09221868e-01  1.75057593e-01
  1.80289612e-01 -1.36958914e-01  5.35177579e-02  1.59746947e-01
 -9.03150487e-01]
[ 6.60205270e-01  3.60598447e-01  6.06032780e-01 -4.98712927e-03
 -4.38143030e-02  1.81817067e-02 -4.96284544e-02  4.10136768e-02
  7.68905081e-04  3.99887120e-02  1.18819299e-02 -9.06310512e-03
  1.17600213e-02  2.15359686e-02 -6.48109248e-03 -5.38746059e-03
  6.13641194e-02  1.51247611e-02 -6.13727662e-02 -1.24996785e-01
  1.85797003e-01]
[ 8.78447550e-02  5.92195712e-02  9.03099007e-02 -1.35765344e-02
 -2.09263016e-03 -7.73278169e-02  1.22223710e-02 -1.27514194e-02
 -3.59952759e-02  1.09390407e-02 -2.08893769e-03  1.30840012e-02
 -2.07739580e-03 -1.13274152e-01 -1.71007401e-01 -2.49519832e-01
 -4.72789242e-01  5.92861507e-02  5.07315023e-02  7.99106629e-01
 -4.54718627e-03]
[-4.40669539e-02 -3.85424669e-02 -1.22118763e-02  1.50981506e-02
  4.17127748e-03  7.59527391e-03  1.02829376e-02  1.47967288e-02
  4.01187094e-03  1.41880197e-02 -7.21997947e-03  1.74025526e-02
 -6.10191339e-03  4.46571358e-02  6.84034726e-02  1.49957766e-01
  4.45426487e-01  2.08306807e-01 -7.56382559e-01  3.74344137e-01
  1.09457331e-01]
[-5.20352731e-03  1.78369989e-02  5.93123188e-04  1.38944120e-02
 -9.87774262e-03 -2.39279053e-03  1.34324136e-02  5.77169106e-03
 -2.68193490e-02 -1.25075312e-03 -7.82606880e-03 -5.05629418e-04
 -6.22391855e-03  2.54459782e-02  7.21717516e-02 -3.95794121e-01
  4.30804712e-01  6.93579204e-01  4.02498949e-01  7.09063019e-02
 -4.62176696e-02]
[ 1.58050205e-02  4.16185760e-04  1.05329710e-03  2.09485290e-02
  5.71162872e-03  5.19945832e-03  1.79767735e-02 -1.65556068e-02
  2.02969480e-02  7.76280584e-03 -4.39119096e-03  2.14656673e-02
 -1.99178935e-03 -2.40333897e-01 -5.91233618e-01  6.73666275e-01
  8.71877611e-02  2.63929042e-01  2.29704517e-01  6.63312593e-02
  4.61394328e-02]
[-1.16815358e-02 -2.52672075e-02 -7.96386073e-03 -4.64806530e-04
  1.42105843e-02  1.34036913e-02  1.38466979e-02  8.68553403e-04
 -5.00885004e-04 -2.17911124e-02  7.35951599e-03 -1.15779935e-02
  1.78995746e-03  7.92983429e-01 -5.72810412e-01 -1.76094924e-01
  1.90611221e-02 -4.20832780e-02 -6.52033299e-02 -4.11938884e-02
 -4.35226151e-02]
[ 1.01059859e-03  7.11108946e-04 -6.35373656e-05 -2.16229296e-02
  2.24117679e-02 -9.12774017e-04  3.49895746e-04  2.46531593e-04
 -9.52783322e-04 -1.31288846e-04 -7.05242755e-01 -4.57863922e-02
  7.06787076e-01  2.93139262e-03 -1.13550991e-03  7.84370619e-05
  8.87506118e-05 -8.08921582e-04 -5.64004222e-04  4.81491217e-04
 -1.97005842e-03]]
```

In [11]:

```python
# Display scree plot to implement elbow rule
PC_values = np.arange(pca.n_components_) + 1
plt.plot(PC_values, pca.explained_variance_ratio_, 'ro-', linewidth=2)
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Proportion of Variance Explained')
plt.show()
```



- According to the scree plot above, there is an elbow at the 6th component. This means that the ideal number of components to select is 6.

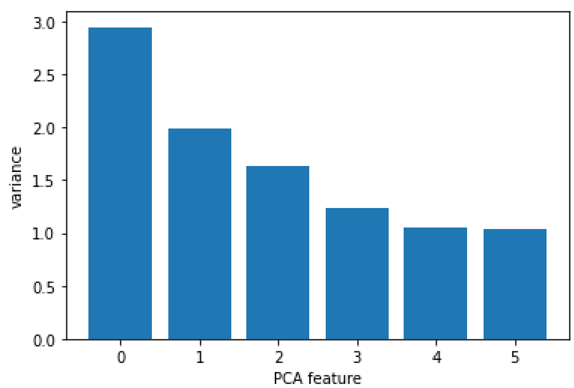
In [12]:
```
# Create a PCA model with 6 components and fit to the data
pca2 = PCA(n_components = 6)
pca2.fit(PCA_std)
```

Out[12]:
```
PCA(n_components=6)
```

In [13]:
```
# Print the variance of each component
for i in range(pca2.n_components_):
    print("Component", i, "has a variance of", round(pca2.explained_variance_[i],2))
    print("Component", i, "explains",round(pca2.explained_variance_ratio_[i]*100,2), 'percent of the variance
 in the data.')
```

```
Component 0 has a variance of 2.95
Component 0 explains 14.04 percent of the variance in the data.
Component 1 has a variance of 2.0
Component 1 explains 9.51 percent of the variance in the data.
Component 2 has a variance of 1.64
Component 2 explains 7.79 percent of the variance in the data.
Component 3 has a variance of 1.24
Component 3 explains 5.88 percent of the variance in the data.
Component 4 has a variance of 1.05
Component 4 explains 5.01 percent of the variance in the data.
Component 5 has a variance of 1.04
Component 5 explains 4.96 percent of the variance in the data.
```

In [14]:
```
# Plot the explained variances
features = range(pca2.n_components_)
plt.bar(features, pca2.explained_variance_)
plt.xlabel('PCA feature')
plt.ylabel('variance')
plt.xticks(features)
plt.show()
```



In [15]:
```
# Calculate and print the total variance captured by components
count = 0
for i in range(pca2.n_components_):
    count = count + pca2.explained_variance_ratio_[i]
print('The total variance explained by the 6 components is', round(count *100, 2), 'percent.')
```

```
The total variance explained by the 6 components is 47.19 percent.
```

Findings

In the PCA analysis described in the assignment's directions, we were to choose the number of components based on the "elbow rule". This uses a scree plot on the PCA transformed data and looks for an "elbow" or bend where the data variability appears to level out. In the case of my scree plot above, this occurs at component 6. When I then performed PCA using six components, only 47% of the variance in the data could be described by the 6 components.

Since my goal was to describe at least 80% of the variance in the data, I decided to run another instance of PCA below specifying that. Using that parameter to optimize my results left me with 13 components that describe a little over 80% of the variance. Looking at the above scree plot, there is another elbow near there so I should have chosen a number closer to that for my initial PCA analysis.

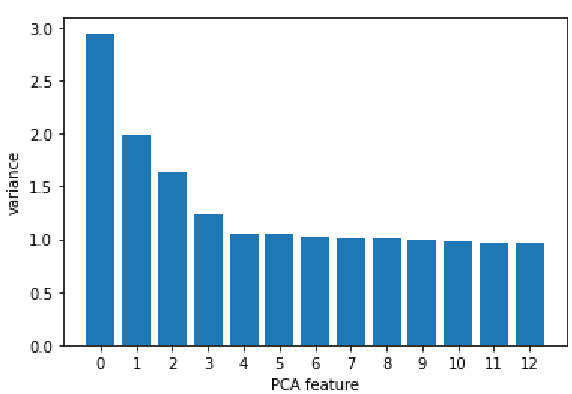
In [16]:
```
# Create a comparison PCA that will keep 80% of the variance
pca3 = PCA(n_components = 0.8)
pca3.fit(PCA_std)
```

Out[16]:
```
PCA(n_components=0.8)
```

In [17]:
```
# Print the variance of each component
for i in range(pca3.n_components_):
    print("Component", i, "has a variance of", round(pca3.explained_variance_[i],2))
    print("Component", i, "explains",round(pca3.explained_variance_ratio_[i]*100,2), 'percent of the variance
 in the data.')
```

```
Component 0 has a variance of 2.95
Component 0 explains 14.04 percent of the variance in the data.
Component 1 has a variance of 2.0
Component 1 explains 9.51 percent of the variance in the data.
Component 2 has a variance of 1.64
Component 2 explains 7.79 percent of the variance in the data.
Component 3 has a variance of 1.24
Component 3 explains 5.88 percent of the variance in the data.
Component 4 has a variance of 1.05
Component 4 explains 5.02 percent of the variance in the data.
Component 5 has a variance of 1.04
Component 5 explains 4.97 percent of the variance in the data.
Component 6 has a variance of 1.02
Component 6 explains 4.87 percent of the variance in the data.
Component 7 has a variance of 1.01
Component 7 explains 4.82 percent of the variance in the data.
Component 8 has a variance of 1.0
Component 8 explains 4.77 percent of the variance in the data.
Component 9 has a variance of 0.99
Component 9 explains 4.73 percent of the variance in the data.
Component 10 has a variance of 0.99
Component 10 explains 4.69 percent of the variance in the data.
Component 11 has a variance of 0.96
Component 11 explains 4.59 percent of the variance in the data.
Component 12 has a variance of 0.96
Component 12 explains 4.58 percent of the variance in the data.
```

In [19]:
```
# Plot the explained variances
features = range(pca3.n_components_)
plt.bar(features, pca3.explained_variance_)
plt.xlabel('PCA feature')
plt.ylabel('variance')
plt.xticks(features)
plt.show()
```



In [20]:
```
count = 0
increment = 0
for i in range(pca3.n_components_):
    count = count + pca3.explained_variance_ratio_[i]
    increment = increment + 1
print('The total variance explained by the', increment, 'components is', round(count *100, 2), 'percent.')
```

```
The total variance explained by the 13 components is 80.27 percent.
```

Sources

- Boeye, J. (n.d.). Dimensionality Reduction in Python. Retrieved February 16, 2021, from https://learn.datacamp.com/courses/dimensionality-reduction-in-python (https://learn.datacamp.com/courses/dimensionality-reduction-in-python)
- Brems, M. (2019, June 10). A one-stop shop for principal component analysis. Retrieved February 18, 2021, from https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c (https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c)
- Loukas, S. (2020, July 28). PCA clearly explained - how, when, why to use it and feature importance: A guide in Python. Retrieved February 18, 2021, from https://towardsdatascience.com/pca-clearly-explained-how-when-why-to-use-it-and-feature-importance-a-guide-in-python-7c274582c37e (https://towardsdatascience.com/pca-clearly-explained-how-when-why-to-use-it-and-feature-importance-a-guide-in-python-7c274582c37e)
- Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), 2825–2830.
- Wilson, B. (n.d.). Unsupervised Learning in Python. Retrieved February 16, 2021, from https://learn.datacamp.com/courses/unsupervised-learning-in-python (https://learn.datacamp.com/courses/unsupervised-learning-in-python)

Helpful Sites Used in Coding Project

1. https://campus.datacamp.com/courses/dimensionality-reduction-in-python (https://campus.datacamp.com/courses/dimensionality-reduction-in-python)
2. https://campus.datacamp.com/courses/unsupervised-learning-in-python/ (https://campus.datacamp.com/courses/unsupervised-learning-in-python/)
3. https://stackoverflow.com/questions/51904126/write-a-numpy-ndarray-to-an-xlsx-spreadsheet (https://stackoverflow.com/questions/51904126/write-a-numpy-ndarray-to-an-xlsx-spreadsheet)
4. https://www.datasklr.com/principal-component-analysis-and-factor-analysis/principal-component-analysis (https://www.datasklr.com/principal-component-analysis-and-factor-analysis/principal-component-analysis)