Final Project

DSCI 508: Machine Learning

Daniel Carter

The final project for this class was attempt to predict whether or not a data scientist was going to stay in a fictional union. There were many features in the data such as whether or not the individual was in management, how long (in months) the individual was in the union, the financial obligations to the union per individual, the individual’s gender, and many more. As stated the goal was to predict whether a given person was going to stay in the union based on all the factors in the dataset. In order to determine this, several things needed to take place. I chose to do this in Python using the following modules:

* Pandas
* Numpy
* Sklearn
  + Label Encoder
  + Standard Scaler
  + Decision Tree Classifier
  + Random Forest Classifier
  + Neural Networks
    - MLP Classifier
* Matplotlib
* Seaborn

I chose Python and the above modules because I am more comfortable with coding in Python than I am in R. The modules I chose are the industry standard when it comes to processing data and creating visualizations and models.

The first thing that I did was load the dataset into a pandas dataframe so that I could get more familiar with the data I was going to be working with. My first step was to take a count of any ‘NaN’ or missing values that there were within the data.

A screenshot of a computer

Description automatically generated

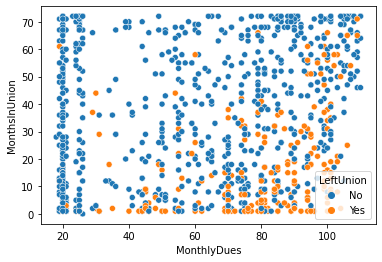
As you can see in the picture above, there where no ‘NaN’ values in the dataset. The next check I had was to see if there were any outliers in the data. In order to do this, I checked the numeric columns using box plots.

A black and white background

Description automatically generatedA screen shot of a computer

Description automatically generated

By looking above, you can see that I was able to spot outliers in the data. Before addressing this, I wanted to visualize the data more. I filtered out the outliers and created two more visualizations. The first one that I created was a scatter plot showing the relationship between the monthly dues and the months spent in the union color-coded by whether or not they were still in the union.



The second visual was a bivariate comparison of the same thing, only comparing the monthly dues and months in the union regarding the type of connectivity the individuals had instead. I used box plots for this as well.

A screenshot of a computer

Description automatically generated

Once I was done visualizing the data, I went to do some data cleanup, standardization, and label encoding to take care of the outliers and categorical features. Upon visual inspection of the data, I noticed that in the TotalDues column, there was a non-numeric value. Since I did not know what that value was supposed to be, I replaced the string value with the value of 0. I then moved on to scaling the data. I used the StandardScaler() function from sklearn. From the box plots, I knew that the outliers were in the numeric columns. I applied the standard scaler function to all the numeric (not the numeric categorical columns) columns in the dataset. Once I did this, it was time to encode the categorical columns that contained string values. I considered several options for this but ultimately went with sklearn’s LabelEncoder() function. I encoded all my string categorical columns using this. The last bit of cleanup that I did was to check for the difference between the unnamed column and the ID column as both looked like unique IDs to me. Once confirming that each column contained unique IDs, I removed the one that contained string values so that I would not have to encode it. As per the assignment recommendations, I started running principal component analysis to see if my data was suffering from any collinearity. I created a scree plot to visualize the results.

A blue line with dots on a white background

Description automatically generated

It was now time to start building and comparing the models. The first model that I tested was a decision tree classifier. I split the data into train and test sets on a 70/30 split. To find the optimal parameters for my tree, I created a range from 2-30 and created temporary trees in a for loop where the max\_depth was whichever number I was on in the range saving the depth and the accuracy to separate lists. I then used numpy to find the depth that had the highest accuracy and used that value for my final model. I then tested the model on my test data because I was worried that my model may have been overfitting due to having a 100% accuracy on the training data. My suspicions were confirmed when the score for the test data ended up being about 74% accuracy compared to the 100% accuracy of the training data.

A number of numbers on a white background

Description automatically generated

The next model that I tested out was a random forest classification model. I took a similar approach to how I created the decision tree but looping through the same range to find the optimal model. Once the optimal model was found, I then did the same test on my test data. As occurred previously, the model overfit the training data. The accuracy of the model on the trained data was 100%. The accuracy of the model on the test data was 78%.

A screenshot of a computer screen

Description automatically generated

The last model that I created was a neural network classification model. This model overfit the data, but nowhere near as bad as it did for the previous two models. The accuracy score for the training data was 76% and the accuracy score for the test data was 73%.

A screenshot of a computer screen

Description automatically generated

As per the requirements please see my answers to the following questions below:

* Comparing your results, to that of a blind guess, explain why you think the results differed?
  + I think the results will be vastly different. A blind guess without taking any of the parameters into consideration or knowing the relationships between the features will essentially lead to the blind guest having a 50/50 shot on guessing whether or not the worker left the union correctly. The models that I built take all of those parameters and factors into consideration before making a prediction. As evident by my results the model was not 100% accurate, but the models had a significantly higher accuracy than a 50/50 shot.
* Describe how you would improve your project if you had more time?
  + If I had more time, there is really only two things that I would want to improve on. I would want to be able to take the time to figure out why my models are overfitting so that I can address it and have a more accurate model. The second thing would be to take more time to fully understand the results of the PCA analysis and see how incorporating more of the results would affect the accuracy of the models. I think that this may have helped solve the overfitting of the model.

On the next page, please see the code that I wrote for this assignment as an HTML page embedded into this word document. I will also be submitting the notebook as a part of this assignment.

# Read in Required Modules[¶](#Read-in-Required-Modules)

In [1]:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn import decomposition

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix, classification\_report

# Read in dataset and do some exploratory analysis[¶](#Read-in-dataset-and-do-some-exploratory)

In [2]:

df = pd.read\_csv('training\_data.csv')

df.head()

Out[2]:

|  | **Unnamed: 0** | **ID** | **gender** | **Management** | **USAcitizen** | **Married** | **MonthsInUnion** | **ContinuingEd** | **FeatureA** | **Connectivity** | **...** | **FeatureE** | **FeatureF** | **FeatureG** | **FeatureB** | **DuesFrequency** | **PaperlessBilling** | **PaymentMethod** | **MonthlyDues** | **TotalDues** | **LeftUnion** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 936 | Daniel1 | Female | 0 | No | No | 12 | No | Maryville | DSL | ... | No | Yes | No | No | One year | Yes | Mailed check | 34 | 435 | No |
| **1** | 4102 | Daniel2 | Male | 0 | Yes | No | 50 | Yes | No | Dial-in | ... | Maryville | Maryville | Maryville | Maryville | Two year | No | Electronic check | 20 | 1013 | No |
| **2** | 3888 | Daniel3 | Female | 1 | Yes | No | 72 | Yes | Yes | Fiber optic | ... | No | No | Yes | Yes | One year | Yes | Bank transfer (automatic) | 107 | 7677 | No |
| **3** | 3864 | Daniel4 | Male | 0 | Yes | Yes | 70 | Yes | Yes | Dial-in | ... | Maryville | Maryville | Maryville | Maryville | Two year | No | Mailed check | 25 | 1782 | No |
| **4** | 4371 | Daniel5 | Male | 0 | No | No | 3 | No | Maryville | DSL | ... | No | No | No | No | Month-to-month | No | Electronic check | 25 | 67 | Yes |

5 rows × 22 columns

In [3]:

# Check for NaN values

print(df.isna().sum())

Unnamed: 0 0

ID 0

gender 0

Management 0

USAcitizen 0

Married 0

MonthsInUnion 0

ContinuingEd 0

FeatureA 0

Connectivity 0

FeatureC 0

FeatureD 0

FeatureE 0

FeatureF 0

FeatureG 0

FeatureB 0

DuesFrequency 0

PaperlessBilling 0

PaymentMethod 0

MonthlyDues 0

TotalDues 0

LeftUnion 0

dtype: int64

In [4]:

#Going to check for outliers. Start with seeing which columns need to be checked (numeric collumns)

print(df.dtypes)

Unnamed: 0 int64

ID object

gender object

Management int64

USAcitizen object

Married object

MonthsInUnion int64

ContinuingEd object

FeatureA object

Connectivity object

FeatureC object

FeatureD object

FeatureE object

FeatureF object

FeatureG object

FeatureB object

DuesFrequency object

PaperlessBilling object

PaymentMethod object

MonthlyDues int64

TotalDues object

LeftUnion object

dtype: object

In [5]:

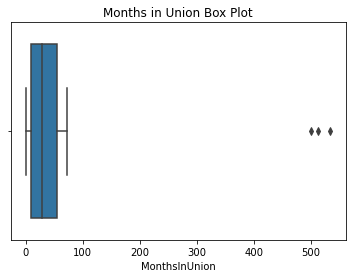
#outlier check list

sns.boxplot(data=df, x='MonthsInUnion',hue='LeftUnion')

plt.title('Months in Union Box Plot')

Out[5]:

Text(0.5, 1.0, 'Months in Union Box Plot')



In [6]:

sns.boxplot(data=df, x='MonthlyDues',hue='LeftUnion')

plt.title('Monthly Dues Box Plot')

Out[6]:

Text(0.5, 1.0, 'Monthly Dues Box Plot')



# Visualize the data further[¶](#Visualize-the-data-further)

## Scatter Plot[¶](#Scatter-Plot)

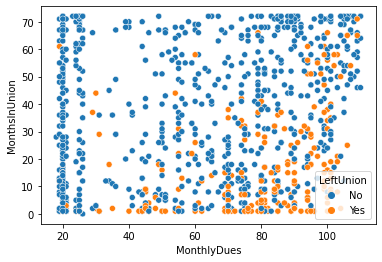
In [7]:

df\_filtered = df[df['MonthlyDues']<=110]

sns.scatterplot(data=df\_filtered, x='MonthlyDues', y='MonthsInUnion', hue='LeftUnion')

Out[7]:

<AxesSubplot:xlabel='MonthlyDues', ylabel='MonthsInUnion'>



In [8]:

categories = df['Connectivity'].unique()

colors = plt.cm.Set3.colors

x\_column, y\_column = 'MonthlyDues', 'MonthsInUnion'

for cat, color in zip(categories, colors):

# x-axis: Monthly dues

x\_data = df\_filtered[df\_filtered['Connectivity'] == cat][x\_column]

x\_q1, x\_median, x\_q3 = np.percentile(x\_data, [25, 50, 75])

# y-axis: Months in the Union

y\_data = df\_filtered[df\_filtered['Connectivity'] == cat][y\_column]

y\_q1, y\_median, y\_q3 = np.percentile(y\_data, [25, 50, 75])

plt.boxplot(x\_data, positions=[y\_median], widths=y\_q3 - y\_q1, vert=False,

showbox=False, manage\_ticks=False)

plt.boxplot(y\_data, positions=[x\_median], widths=x\_q3 - x\_q1,

showbox=False, manage\_ticks=False,

boxprops={'facecolor': color, 'label': cat})

plt.gca().add\_patch(plt.Rectangle((x\_q1, y\_q1), x\_q3 - x\_q1, y\_q3 - y\_q1,

facecolor=color, edgecolor='black', label=cat))

plt.title('Bivariate Comparison')

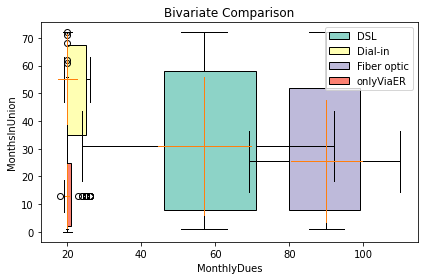
plt.xlabel(x\_column)

plt.ylabel(y\_column)

plt.legend()

plt.tight\_layout()

plt.show()



# Evidence of outliers. Need to Standardize the data[¶](#Evidence-of-outliers.-Need-to-Standardi)

In [9]:

#split dataset into numeric and categorical features

#first need to investigate why the TotalDues column is of object type

print(df['TotalDues'].unique())

['435' '1013' '7677' '1782' '67' '1399' '615' '1039' '4563' '3012' '20'

'889' '2165' '54' '789' '330' '416' '1391' '5309' '1428' '470' '372'

'914' '2152' '2215' '1110' '197' '1052' '4234' '1709' '2586' '5073'

'1071' '1517' '42' '2110' '4534' '2387' '5154' '932' '2511' '6293' '1359'

'2949' '100' '1030' '730' '3196' '4903' '964' '1346' '5614' '627' '679'

'46' '44' '2439' '1360' '7209' '36' '2362' '1718' '260' '1862' '810'

'2680' '5657' '1375' '5931' '1012' '1929' '883' '5295' '5018' '511'

'3688' '1733' '1620' '2201' '3770' '4126' '6298' '2647' '294' '6369'

'521' '2245' '89' '1510' '261' '1025' '3480' '3619' '76' '1303' '357'

'2472' '1611' '279' '610' '6999' '3759' '6307' '252' '5468' '158' '762'

'256' '1017' '534' '1582' '1752' '147' '2659' '5424' '2634' '1759' '912'

'5301' '1431' '494' '1091' '2920' '825' '1194' '144' '162' '1378' '234'

'2384' '7854' '223' '720' '555' '6496' '4972' '2893' '625' '6813' '340'

'655' '2249' '154' '335' '6076' '149' '1318' '185' '8306' '4177' '1785'

'6638' '1791' '1337' '5333' '55' '19' '4805' '1350' '4896' '488' '1535'

'769' '3162' '847' '21' '855' '77' '352' '4070' '94' '3558' '4911' '923'

'3789' '70' '3307' '419' '2417' '2375' '4849' '7557' '3229' '8062' '2227'

'5315' '1284' '2049' '1753' '245' '1547' '3067' '220' '4749' '395' '4214'

'4634' '5318' '2754' '274' '5647' '5375' '3176' '6010' '1761' '1760'

'862' '927' '8332' '1270' '1425' '5792' '1171' '62' '2303' '2553' '1329'

'227' '813' '287' '7099' '5950' '3626' '6431' '1258' '5376' '5043' '551'

'5220' '5883' '280' '8129' '790' '6767' '25' '3942' '369' '3474' '4135'

'327' '341' '1269' '38' '58' '919' '1121' '343' '132' '1885' '876' '6373'

'6142' '4847' '1238' '1826' '1493' '452' '5373' '596' '632' '5826' '75'

'5731' '284' '113' '829' '3496' '3539' '226' '2054' '6404' '2800' '7882'

'1183' '2697' '384' '734' '4164' '5941' '3341' '532' '1478' '117' '346'

'1276' '1800' '5919' '1476' '2244' '3673' '461' '568' '995' '83' '2277'

'2258' '1743' '24' '7298' '4713' '2345' '103' '1876' '3894' '3260' '3122'

'6518' '1390' '3090' '265' '1073' '3598' '212' '2059' '50' '5358' '866'

'1422' '177' '989' '5229' '51' '131' '4760' '5512' '712' '4677' '909'

'519' '71' '324' '7384' '305' '5811' '242' '507' '2693' '2570' '3522'

'1388' '3110' '1442' '7052' '3411' '5608' '99' '1308' '190' '659' '221'

'3410' '43' '7476' '1206' '3237' '563' '684' '5714' '1490' '6018' '1000'

'7544' '3973' '4301' '3211' '1953' '1927' '6697' '6992' '82' '7459' '373'

'3966' '1818' '5254' '5638' '1232' '2196' '7029' '165' '422' '5431'

'1108' '2030' '135' '2532' '1048' '6995' '1728' '1765' '243' '145' '6469'

'4448' '7943' '271' '1776' '3479' '41' '6224' '3439' '2627' '69' '1014'

'368' '3843' '1637' '4595' '536' '554' '2922' '744' '2855' '3255' '966'

'7062' '3523' '320' '440' '1868' '809' '1908' '579' '6563' '4536' '1058'

'3335' '6652' '2379' '4473' '1716' '4930' '1401' '1162' '81' '371' '3069'

'3954' '1184' '129' '2272' '230' '6463' '4052' '607' '2013' '979' '4080'

'1780' '5330' '451' '3582' '892' '4433' '593' '765' '8685' '3183' '4784'

'3006' '856' '6983' '840' '3682' '#VALUE!' '4628' '248' '4812' '5885'

'1874' '5000' '1156' '1199' '39' '1259' '1192' '586' '535' '5986' '1279'

'2737' '1457' '5648' '86' '1992' '985' '3420' '6549' '3365' '92' '80'

'120' '2768' '1196' '7711' '1029' '819' '6558' '79' '3326' '63' '163'

'1205' '947' '73' '4107' '6986' '3541' '687' '2263' '3120' '5155' '6751'

'5072' '1952' '391' '561' '600' '3214' '60' '680' '2357' '6253' '4331'

'3581' '7983' '61' '7288' '5684' '703' '1504' '5932' '1125' '571' '1129'

'74' '956' '5497' '3400' '7245' '698' '434' '875' '7031' '1043' '95'

'411' '2723' '4913' '2840' '31' '315' '3297' '6384' '6333' '2796' '1703'

'1616' '2031' '6155' '2823' '3991' '1715' '3112' '262' '1079' '553'

'1580' '1418' '3171' '332' '7050' '45' '5827' '194' '151' '6043' '1779'

'1126' '667' '293' '1286' '1278' '6411' '400' '2105' '2657' '28' '3356'

'1463' '4921' '5125' '2514' '3145' '1639' '606' '4139' '1554' '2758'

'2546' '431' '228' '5981' '218' '5174' '4514' '72' '3027' '1744' '1530'

'4720' '7920' '5499' '5284' '4348' '7560' '1272' '974' '7323' '90' '1648'

'161' '6311' '2633' '2549' '714' '614' '152' '445' '199' '6481' '306'

'35' '4017' '1293' '777' '3567' '460' '4246' '1949' '5763' '171' '6393'

'1567' '3951' '5325' '828' '940' '1212' '6980' '7635' '960' '438' '7016'

'1623' '3020' '1239' '1231' '96' '110' '504' '176' '107' '3046' '179'

'4741' '439' '1688' '1597' '6683' '1751' '85' '2067' '640' '583' '4113'

'1679' '186' '1916' '6202' '3187' '4751' '2970' '3822' '1942' '5430'

'5969' '965' '3849' '4590' '6997' '1837' '3650' '1340' '1540' '297'

'1087' '1262' '1237' '2598' '7083' '3127' '418' '2655' '678' '6300'

'7840' '3564' '2719' '1111' '634' '497' '6029' '4813' '567' '1564' '1729'

'602' '4060' '750' '938' '4385' '2495' '2807' '4845' '56' '868' '4599'

'1137' '5589' '2649' '1038' '344' '1600' '235' '106' '1174' '1830' '278'

'6145' '3734' '336' '2187' '420' '123' '4117' '7691' '3021' '6591' '4824'

'273' '2975' '1083' '127' '1078' '7325' '836' '3167' '198' '2089' '1410'

'348' '1335' '350' '2147' '313' '3778' '741' '1654' '3453' '508' '1834'

'1426' '2524' '2794' '1229' '1382' '8078' '2447' '6938' '1804' '40' '832'

'2560' '1255' '6841' '3142' '8318' '390' '1074' '403' '548' '3708' '2405'

'3476' '6152' '1236' '3527' '4632' '487' '2879' '2122' '308' '1396' '143'

'3089' '763' '1505' '1983' '738' '8478' '827' '3321' '3163' '3895' '1626'

'2931' '1010' '4578' '7406' '6162' '1520' '3776' '5681' '1413' '425'

'739' '5813' '6872' '950' '916' '5720' '653' '229' '4266' '1851' '654'

'4806' '6111' '930' '1948' '1024' '1248']

In [10]:

#Need to replace the '#VALUE with numeric value'

df['TotalDues'] = np.where(df['TotalDues']=='#VALUE!', 0, df['TotalDues'])

#check

print(np.sum(df['TotalDues']=='#VALUE!'))

0

In [11]:

#scale numerical columns

scaler = StandardScaler()

df[['MonthsInUnion', 'MonthlyDues', 'TotalDues']] = scaler.fit\_transform(df[['MonthsInUnion', 'MonthlyDues', 'TotalDues']])

df

Out[11]:

|  | **Unnamed: 0** | **ID** | **gender** | **Management** | **USAcitizen** | **Married** | **MonthsInUnion** | **ContinuingEd** | **FeatureA** | **Connectivity** | **...** | **FeatureE** | **FeatureF** | **FeatureG** | **FeatureB** | **DuesFrequency** | **PaperlessBilling** | **PaymentMethod** | **MonthlyDues** | **TotalDues** | **LeftUnion** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 936 | Daniel1 | Female | 0 | No | No | -0.598502 | No | Maryville | DSL | ... | No | Yes | No | No | One year | Yes | Mailed check | -0.123991 | -0.808685 | No |
| **1** | 4102 | Daniel2 | Male | 0 | Yes | No | 0.451890 | Yes | No | Dial-in | ... | Maryville | Maryville | Maryville | Maryville | Two year | No | Electronic check | -0.161044 | -0.548602 | No |
| **2** | 3888 | Daniel3 | Female | 1 | Yes | No | 1.060011 | Yes | Yes | Fiber optic | ... | No | No | Yes | Yes | One year | Yes | Bank transfer (automatic) | 0.069212 | 2.450001 | No |
| **3** | 3864 | Daniel4 | Male | 0 | Yes | Yes | 1.004728 | Yes | Yes | Dial-in | ... | Maryville | Maryville | Maryville | Maryville | Two year | No | Mailed check | -0.147811 | -0.202574 | No |
| **4** | 4371 | Daniel5 | Male | 0 | No | No | -0.847279 | No | Maryville | DSL | ... | No | No | No | No | Month-to-month | No | Electronic check | -0.147811 | -0.974274 | Yes |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **995** | 3145 | Daniel996 | Male | 0 | Yes | Yes | -0.570860 | Yes | Yes | DSL | ... | Yes | Yes | Yes | No | Month-to-month | Yes | Electronic check | -0.020773 | -0.585949 | No |
| **996** | 3429 | Daniel997 | Female | 0 | Yes | No | -0.349725 | Yes | Yes | Fiber optic | ... | No | No | Yes | Yes | Month-to-month | Yes | Electronic check | 0.034806 | -0.127879 | No |
| **997** | 4918 | Daniel998 | Female | 0 | No | No | -0.874921 | Yes | Yes | Fiber optic | ... | No | No | No | No | Month-to-month | No | Electronic check | -0.002247 | -0.936026 | No |
| **998** | 3108 | Daniel999 | Female | 0 | No | No | 0.451890 | Yes | No | Dial-in | ... | Maryville | Maryville | Maryville | Maryville | Two year | No | Credit card (automatic) | -0.163690 | -0.543652 | No |
| **999** | 1506 | Daniel1000 | Female | 0 | No | No | -0.432651 | Yes | No | Fiber optic | ... | No | No | No | No | Month-to-month | Yes | Electronic check | -0.026066 | -0.442859 | No |

1000 rows × 22 columns

In [12]:

#encode categorical columns

le = LabelEncoder()

df['gender'] = le.fit\_transform(df['gender'])

df['USAcitizen'] = le.fit\_transform(df['USAcitizen'])

df['Married'] = le.fit\_transform(df['Married'])

df['ContinuingEd'] = le.fit\_transform(df['ContinuingEd'])

df['FeatureA'] = le.fit\_transform(df['FeatureA'])

df['Connectivity'] = le.fit\_transform(df['Connectivity'])

df['FeatureC'] = le.fit\_transform(df['FeatureC'])

df['FeatureD'] = le.fit\_transform(df['FeatureD'])

df['FeatureE'] = le.fit\_transform(df['FeatureE'])

df['FeatureF'] = le.fit\_transform(df['FeatureF'])

df['FeatureG'] = le.fit\_transform(df['FeatureG'])

df['FeatureB'] = le.fit\_transform(df['FeatureB'])

df['DuesFrequency'] = le.fit\_transform(df['DuesFrequency'])

df['PaperlessBilling'] = le.fit\_transform(df['PaperlessBilling'])

df['PaymentMethod'] = le.fit\_transform(df['PaymentMethod'])

df['LeftUnion'] = le.fit\_transform(df['LeftUnion'])

df

Out[12]:

|  | **Unnamed: 0** | **ID** | **gender** | **Management** | **USAcitizen** | **Married** | **MonthsInUnion** | **ContinuingEd** | **FeatureA** | **Connectivity** | **...** | **FeatureE** | **FeatureF** | **FeatureG** | **FeatureB** | **DuesFrequency** | **PaperlessBilling** | **PaymentMethod** | **MonthlyDues** | **TotalDues** | **LeftUnion** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 936 | Daniel1 | 0 | 0 | 0 | 0 | -0.598502 | 0 | 0 | 0 | ... | 1 | 2 | 1 | 1 | 1 | 1 | 3 | -0.123991 | -0.808685 | 0 |
| **1** | 4102 | Daniel2 | 1 | 0 | 1 | 0 | 0.451890 | 1 | 1 | 1 | ... | 0 | 0 | 0 | 0 | 2 | 0 | 2 | -0.161044 | -0.548602 | 0 |
| **2** | 3888 | Daniel3 | 0 | 1 | 1 | 0 | 1.060011 | 1 | 2 | 2 | ... | 1 | 1 | 2 | 2 | 1 | 1 | 0 | 0.069212 | 2.450001 | 0 |
| **3** | 3864 | Daniel4 | 1 | 0 | 1 | 1 | 1.004728 | 1 | 2 | 1 | ... | 0 | 0 | 0 | 0 | 2 | 0 | 3 | -0.147811 | -0.202574 | 0 |
| **4** | 4371 | Daniel5 | 1 | 0 | 0 | 0 | -0.847279 | 0 | 0 | 0 | ... | 1 | 1 | 1 | 1 | 0 | 0 | 2 | -0.147811 | -0.974274 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **995** | 3145 | Daniel996 | 1 | 0 | 1 | 1 | -0.570860 | 1 | 2 | 0 | ... | 2 | 2 | 2 | 1 | 0 | 1 | 2 | -0.020773 | -0.585949 | 0 |
| **996** | 3429 | Daniel997 | 0 | 0 | 1 | 0 | -0.349725 | 1 | 2 | 2 | ... | 1 | 1 | 2 | 2 | 0 | 1 | 2 | 0.034806 | -0.127879 | 0 |
| **997** | 4918 | Daniel998 | 0 | 0 | 0 | 0 | -0.874921 | 1 | 2 | 2 | ... | 1 | 1 | 1 | 1 | 0 | 0 | 2 | -0.002247 | -0.936026 | 0 |
| **998** | 3108 | Daniel999 | 0 | 0 | 0 | 0 | 0.451890 | 1 | 1 | 1 | ... | 0 | 0 | 0 | 0 | 2 | 0 | 1 | -0.163690 | -0.543652 | 0 |
| **999** | 1506 | Daniel1000 | 0 | 0 | 0 | 0 | -0.432651 | 1 | 1 | 2 | ... | 1 | 1 | 1 | 1 | 0 | 1 | 2 | -0.026066 | -0.442859 | 0 |

1000 rows × 22 columns

In [13]:

#clean up the unnamed column and id column

df = df.rename(columns={'Unnamed: 0': 'alt\_id'})

print(df['alt\_id'].nunique())

print(df['ID'].nunique())

1000

1000

In [14]:

#no need to have both then. Drop ID since it contains a string

df = df.drop(columns=['ID'])

df

Out[14]:

|  | **alt\_id** | **gender** | **Management** | **USAcitizen** | **Married** | **MonthsInUnion** | **ContinuingEd** | **FeatureA** | **Connectivity** | **FeatureC** | **...** | **FeatureE** | **FeatureF** | **FeatureG** | **FeatureB** | **DuesFrequency** | **PaperlessBilling** | **PaymentMethod** | **MonthlyDues** | **TotalDues** | **LeftUnion** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 936 | 0 | 0 | 0 | 0 | -0.598502 | 0 | 0 | 0 | 2 | ... | 1 | 2 | 1 | 1 | 1 | 1 | 3 | -0.123991 | -0.808685 | 0 |
| **1** | 4102 | 1 | 0 | 1 | 0 | 0.451890 | 1 | 1 | 1 | 0 | ... | 0 | 0 | 0 | 0 | 2 | 0 | 2 | -0.161044 | -0.548602 | 0 |
| **2** | 3888 | 0 | 1 | 1 | 0 | 1.060011 | 1 | 2 | 2 | 2 | ... | 1 | 1 | 2 | 2 | 1 | 1 | 0 | 0.069212 | 2.450001 | 0 |
| **3** | 3864 | 1 | 0 | 1 | 1 | 1.004728 | 1 | 2 | 1 | 0 | ... | 0 | 0 | 0 | 0 | 2 | 0 | 3 | -0.147811 | -0.202574 | 0 |
| **4** | 4371 | 1 | 0 | 0 | 0 | -0.847279 | 0 | 0 | 0 | 1 | ... | 1 | 1 | 1 | 1 | 0 | 0 | 2 | -0.147811 | -0.974274 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **995** | 3145 | 1 | 0 | 1 | 1 | -0.570860 | 1 | 2 | 0 | 1 | ... | 2 | 2 | 2 | 1 | 0 | 1 | 2 | -0.020773 | -0.585949 | 0 |
| **996** | 3429 | 0 | 0 | 1 | 0 | -0.349725 | 1 | 2 | 2 | 1 | ... | 1 | 1 | 2 | 2 | 0 | 1 | 2 | 0.034806 | -0.127879 | 0 |
| **997** | 4918 | 0 | 0 | 0 | 0 | -0.874921 | 1 | 2 | 2 | 1 | ... | 1 | 1 | 1 | 1 | 0 | 0 | 2 | -0.002247 | -0.936026 | 0 |
| **998** | 3108 | 0 | 0 | 0 | 0 | 0.451890 | 1 | 1 | 1 | 0 | ... | 0 | 0 | 0 | 0 | 2 | 0 | 1 | -0.163690 | -0.543652 | 0 |
| **999** | 1506 | 0 | 0 | 0 | 0 | -0.432651 | 1 | 1 | 2 | 1 | ... | 1 | 1 | 1 | 1 | 0 | 1 | 2 | -0.026066 | -0.442859 | 0 |

1000 rows × 21 columns

# PCA[¶](#PCA)

In [15]:

X = df.drop(columns=['LeftUnion'])

pca = decomposition.PCA(n\_components=3)

pca.fit(X)

X = pca.transform(X)

components = pca.components\_

## Scree Plot[¶](#Scree-Plot)

In [16]:

PC\_values = np.arange(pca.n\_components\_) + 1

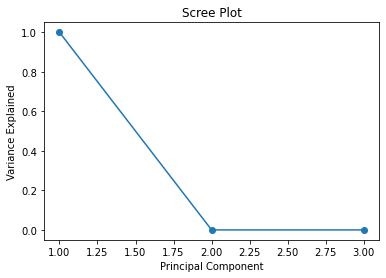
plt.plot(PC\_values, pca.explained\_variance\_ratio\_, 'o-')

plt.title('Scree Plot')

plt.xlabel('Principal Component')

plt.ylabel('Variance Explained')

plt.show()



# Models[¶](#Models)

### Decision Tree Classifier[¶](#Decision-Tree-Classifier)

In [17]:

#split data into train and test

X = df.drop(columns=['LeftUnion'])

y = df['LeftUnion']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=.3)

In [18]:

#train tree

dt\_train\_acc = []

depth\_range = range(2,30)

for i in depth\_range:

np.random.seed(1)

temp\_tree = DecisionTreeClassifier(max\_depth=i)

temp\_tree.fit(X\_train, y\_train)

dt\_train\_acc.append(temp\_tree.score(X\_train, y\_train))

dt\_idx = np.argmax(dt\_train\_acc)

dt\_opt\_depth = depth\_range[dt\_idx]

print(f'Optimal value for max\_depth: {dt\_opt\_depth}')

print(f'Training Accuracy for Optimal Model: {np.round(dt\_train\_acc[dt\_idx], 4)}')

Optimal value for max\_depth: 16

Training Accuracy for Optimal Model: 1.0

In [19]:

#Might be overfitting the model. Feed test data to see how it performs

tree = DecisionTreeClassifier(max\_depth=15)

tree.fit(X\_train, y\_train)

print(np.round(tree.score(X\_test,y\_test),4))

predictions = tree.predict(X\_test)

cm = confusion\_matrix(y\_test, predictions)

cm\_df = pd.DataFrame(cm, columns=['1','2'])

print(cm\_df)

cr = classification\_report(y\_test, predictions)

print(cr)

0.7367

1 2

0 186 32

1 47 35

precision recall f1-score support

0 0.80 0.85 0.82 218

1 0.52 0.43 0.47 82

accuracy 0.74 300

macro avg 0.66 0.64 0.65 300

weighted avg 0.72 0.74 0.73 300

### Random Forrest Classifier[¶](#Random-Forrest-Classifier)

In [20]:

df\_train\_acc = []

for i in depth\_range:

np.random.seed(1)

temp\_forrest = RandomForestClassifier(max\_depth=i, n\_estimators=100)

temp\_forrest.fit(X\_train, y\_train)

df\_train\_acc.append(temp\_forrest.score(X\_train, y\_train))

rf\_idx = np.argmax(df\_train\_acc)

rf\_opt\_depth = depth\_range[rf\_idx]

print(f'Optimal value for max depth: {rf\_opt\_depth}')

print(f'Training Accuracy for Optimal Model: {np.round(df\_train\_acc[rf\_idx], 4)}')

Optimal value for max depth: 12

Training Accuracy for Optimal Model: 1.0

In [21]:

np.random.seed(1)

forest = RandomForestClassifier(max\_depth=13, n\_estimators=100)

forest.fit(X\_train, y\_train)

print(f'Training Accuracy for Random Forest: {np.round(forest.score(X\_train, y\_train), 4)}')

predictions = forest.predict(X\_test)

cm = confusion\_matrix(y\_test, predictions)

cm\_df = pd.DataFrame(cm, columns=['1','2'])

print(cm\_df)

cr = classification\_report(y\_test, predictions)

print(cr)

Training Accuracy for Random Forest: 1.0

1 2

0 203 15

1 52 30

precision recall f1-score support

0 0.80 0.93 0.86 218

1 0.67 0.37 0.47 82

accuracy 0.78 300

macro avg 0.73 0.65 0.67 300

weighted avg 0.76 0.78 0.75 300

### Neural Network[¶](#Neural-Network)

In [22]:

clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(5,2), random\_state=1)

X = df.drop(columns=['LeftUnion'])

y = df['LeftUnion']

clf.fit(X,y)

clf.score(X\_train, y\_train)

Out[22]:

0.7614285714285715

In [23]:

clf.score(X\_test, y\_test)

Out[23]:

0.7266666666666667

In [24]:

predictions = clf.predict(X\_test)

cm = confusion\_matrix(y\_test, predictions)

cm\_df = pd.DataFrame(cm, columns=['1','2'])

print(cm\_df)

cr = classification\_report(y\_test, predictions)

print(cr)

1 2

0 218 0

1 82 0

precision recall f1-score support

0 0.73 1.00 0.84 218

1 0.00 0.00 0.00 82

accuracy 0.73 300

macro avg 0.36 0.50 0.42 300

weighted avg 0.53 0.73 0.61 300

C:\Anaconda\envs\PythonAdv\lib\site-packages\sklearn\metrics\\_classification.py:1221: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

In [ ]: