Week 6 - Clustering - Python

October 24, 2018

1 Data Warehousing and Data Mining

1.1 Labs

1.1.1 Prepared by Gilroy Gordon

Contact Information SCIT ext. 3643
ggordonutech@gmail.com
gilroy.gordon@utech.edu.jm

1.1.2 Week 6 - Clustering in Python

Additional Reference Resources:
http://scikit-learn.org/stable/modules/clustering.html

1.2 Objectives

```
> Data Preprocessing
     > Missing Values (Na and nulls)
> Data Mining
     > Clustering (Kmeans)
> Visualizations
     > Elbow method (Within Cluster Sum of Squares)
> Discussion on how to proceed
```

1.3 Aim: Am I able to segment groups based on

1.4 Import required libraries and acquire data

```
In [1]: # import required libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline
```

In [2]:
```python
data_path = './data/credit-card-data.csv' # Path to data file
data = pd.read_csv(data_path)
data.head(15)
```

Out[2]:
```
   cust_id      balance  balance_frequency  purchases  oneoff_purchases  \
0   C10001    40.900749           0.818182      95.40              0.00
1   C10002  3202.467416           0.909091       0.00              0.00
2   C10003  2495.148862           1.000000     773.17            773.17
3   C10004  1666.670542           0.636364    1499.00           1499.00
4   C10005   817.714335           1.000000      16.00             16.00
5   C10006  1809.828751           1.000000    1333.28              0.00
6   C10007   627.260806           1.000000    7091.01           6402.63
7   C10008  1823.652743           1.000000     436.20              0.00
8   C10009  1014.926473           1.000000     861.49            661.49
9   C10010   152.225975           0.545455    1281.60           1281.60
10  C10011  1293.124939           1.000000     920.12              0.00
11  C10012   630.794744           0.818182    1492.18           1492.18
12  C10013  1516.928620           1.000000    3217.99           2500.23
13  C10014   921.693369           1.000000    2137.93            419.96
14  C10015  2772.772734           1.000000       0.00              0.00

    installments_purchases  cash_advance  purchases_frequency  \
0                    95.40      0.000000             0.166667
1                     0.00   6442.945483             0.000000
2                     0.00      0.000000             1.000000
3                     0.00    205.788017             0.083333
4                     0.00      0.000000             0.083333
5                  1333.28      0.000000             0.666667
6                   688.38      0.000000             1.000000
7                   436.20      0.000000             1.000000
8                   200.00      0.000000             0.333333
9                     0.00      0.000000             0.166667
10                  920.12      0.000000             1.000000
11                    0.00      0.000000             0.250000
12                  717.76      0.000000             1.000000
13                 1717.97      0.000000             0.750000
14                    0.00    346.811390             0.000000

    oneoff_purchases_frequency  purchases_installments_frequency  \
0                     0.000000                          0.083333
1                     0.000000                          0.000000
2                     1.000000                          0.000000
3                     0.083333                          0.000000
4                     0.083333                          0.000000
5                     0.000000                          0.583333
6                     1.000000                          1.000000
7                     0.000000                          1.000000
8                     0.083333                          0.250000
```

```
9                     0.166667                          0.000000
10                    0.000000                          1.000000
11                    0.250000                          0.000000
12                    0.250000                          0.916667
13                    0.166667                          0.750000
14                    0.000000                          0.000000

    cash_advance_frequency  cash_advance_trx  purchases_trx  credit_limit  \
0                 0.000000                 0              2          1000
1                 0.250000                 4              0          7000
2                 0.000000                 0             12          7500
3                 0.083333                 1              1          7500
4                 0.000000                 0              1          1200
5                 0.000000                 0              8          1800
6                 0.000000                 0             64         13500
7                 0.000000                 0             12          2300
8                 0.000000                 0              5          7000
9                 0.000000                 0              3         11000
10                0.000000                 0             12          1200
11                0.000000                 0              6          2000
12                0.000000                 0             26          3000
13                0.000000                 0             26          7500
14                0.083333                 1              0          3000

       payments  minimum_payments  prc_full_payment  tenure
0    201.802084        139.509787          0.000000      12
1   4103.032597       1072.340217          0.222222      12
2    622.066742        627.284787          0.000000      12
3      0.000000               NaN          0.000000      12
4    678.334763        244.791237          0.000000      12
5   1400.057770       2407.246035          0.000000      12
6   6354.314328        198.065894          1.000000      12
7    679.065082        532.033990          0.000000      12
8    688.278568        311.963409          0.000000      12
9   1164.770591        100.302262          0.000000      12
10  1083.301007       2172.697765          0.000000      12
11   705.618627        155.549069          0.000000      12
12   608.263689        490.207013          0.250000      12
13  1655.891435        251.137986          0.083333      12
14   805.647974        989.962866          0.000000      12
```

In [3]:
```
# What columns are in the data set ? Do they have spaces that I should consider
data.columns
```

Out[3]:
```
Index(['cust_id', 'balance', 'balance_frequency', 'purchases',
       'oneoff_purchases', 'installments_purchases', 'cash_advance',
       'purchases_frequency', 'oneoff_purchases_frequency',
       'purchases_installments_frequency', 'cash_advance_frequency',
```

```
          'cash_advance_trx', 'purchases_trx', 'credit_limit', 'payments',
          'minimum_payments', 'prc_full_payment', 'tenure'],
         dtype='object')
```

1.4.1 Exploring our data

Methods also available in Kaggle Bot default kernel. Some Modifications made for compatibility

In [4]:
```
# Histogram of column data
def plotHistogram(df, nHistogramShown, nHistogramPerRow):
    nunique = len(df.columns)
    df = df[[col for col in df if nunique[col] > 1 and nunique[col] < 50]] # For display
    nRow, nCol = df.shape
    columnNames = list(df)
    nHistRow = (nCol + nHistogramPerRow - 1) / nHistogramPerRow
    plt.figure(num=None, figsize=(6*nHistogramPerRow, 8*nHistRow), dpi=80, facecolor='w'
    for i in range(min(nCol, nHistogramShown)):
        plt.subplot(nHistRow, nHistogramPerRow, i+1)
        df.iloc[:,i].hist()
        plt.ylabel('counts')
        plt.xticks(rotation=90)
        plt.title('{0} (column {1})'.format(columnNames[i],i))
    plt.tight_layout(pad=1.0, w_pad=1.0, h_pad=1.0)
    plt.show()

# Correlation matrix
def plotCorrelationMatrix(df, graphWidth):
    filename = df.dataframeName
    df = df.dropna('columns') # drop columns with NaN
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are
    if df.shape[1] < 2:
        print('No correlation plots shown: The number of non-NaN or constant columns ({0
        return
    corr = df.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecolor='w', edgeco
    corrMat = plt.matshow(corr, fignum = 1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
    plt.title('Correlation Matrix for {0}'.format(filename), fontsize=15)
    plt.show()


# Scatter and density plots
def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include =[np.number]) # keep only numerical columns
    # Remove rows and columns that would lead to df being singular
```

```
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are
    columnNames = list(df)
    if len(columnNames) > 10: # reduce the number of columns for matrix inversion of ker
        columnNames = columnNames[:10]
    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize, plotSize], diagon
    corrs = df.corr().values
    for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
        ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2), xycoords='axes
    plt.suptitle('Scatter and Density Plot')
    plt.show()
```

In [5]:
```
data.describe()
```

Out[5]:

	balance	balance_frequency	purchases	oneoff_purchases	\
count	8950.000000	8950.000000	8950.000000	8950.000000	
mean	1564.474828	0.877271	1003.204834	592.437371	
std	2081.531879	0.236904	2136.634782	1659.887917	
min	0.000000	0.000000	0.000000	0.000000	
25%	128.281915	0.888889	39.635000	0.000000	
50%	873.385231	1.000000	361.280000	38.000000	
75%	2054.140036	1.000000	1110.130000	577.405000	
max	19043.138560	1.000000	49039.570000	40761.250000	

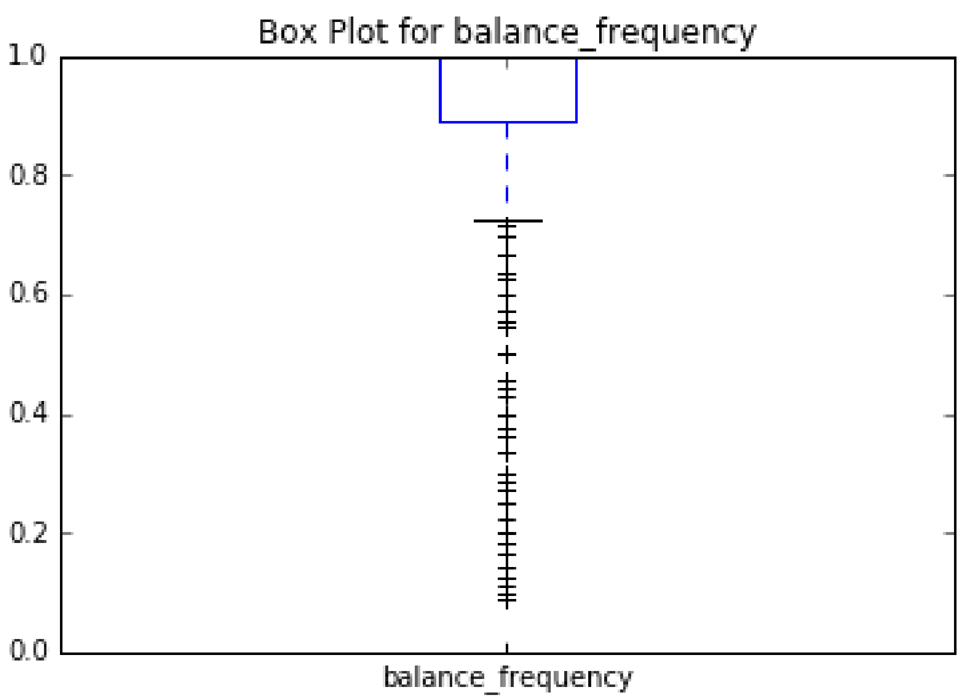
	installments_purchases	cash_advance	purchases_frequency	\
count	8950.000000	8950.000000	8950.000000	
mean	411.067645	978.871112	0.490351	
std	904.338115	2097.163877	0.401371	
min	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.083333	
50%	89.000000	0.000000	0.500000	
75%	468.637500	1113.821139	0.916667	
max	22500.000000	47137.211760	1.000000	

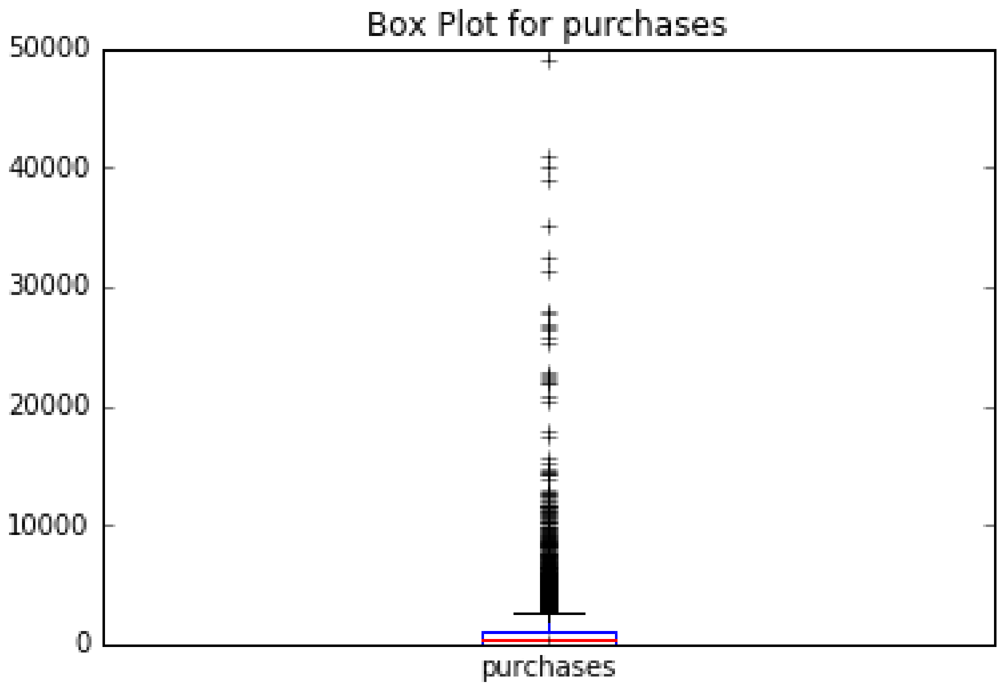
	oneoff_purchases_frequency	purchases_installments_frequency	\
count	8950.000000	8950.000000	
mean	0.202458	0.364437	
std	0.298336	0.397448	
min	0.000000	0.000000	
25%	0.000000	0.000000	
50%	0.083333	0.166667	
75%	0.300000	0.750000	
max	1.000000	1.000000	

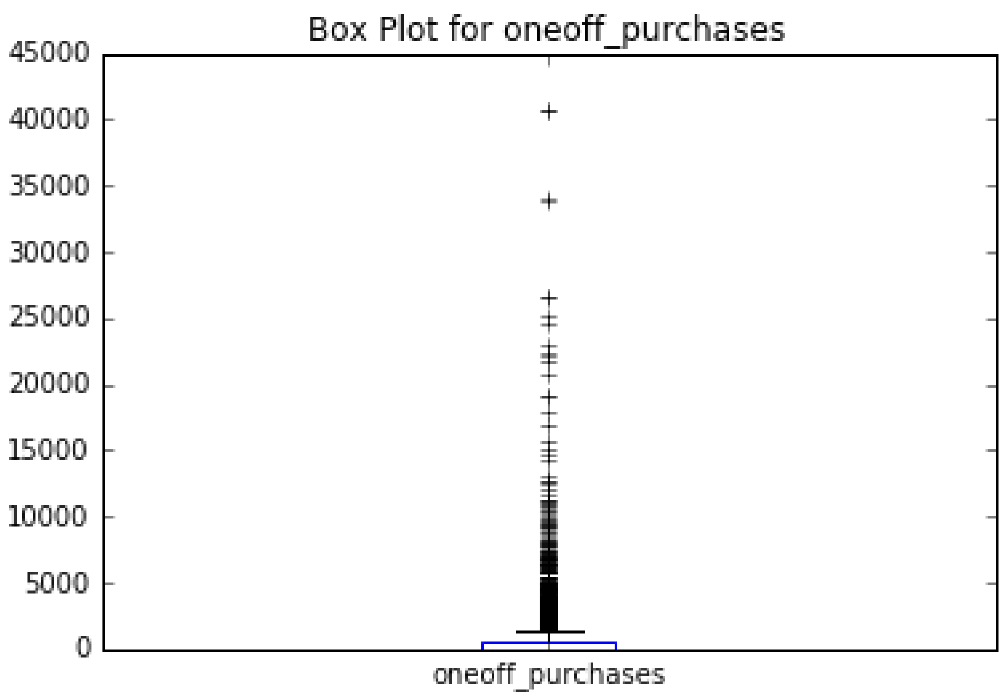
	cash_advance_frequency	cash_advance_trx	purchases_trx	credit_limit	\
count	8950.000000	8950.000000	8950.000000	8949.000000	
mean	0.135144	3.248827	14.709832	4494.449450	

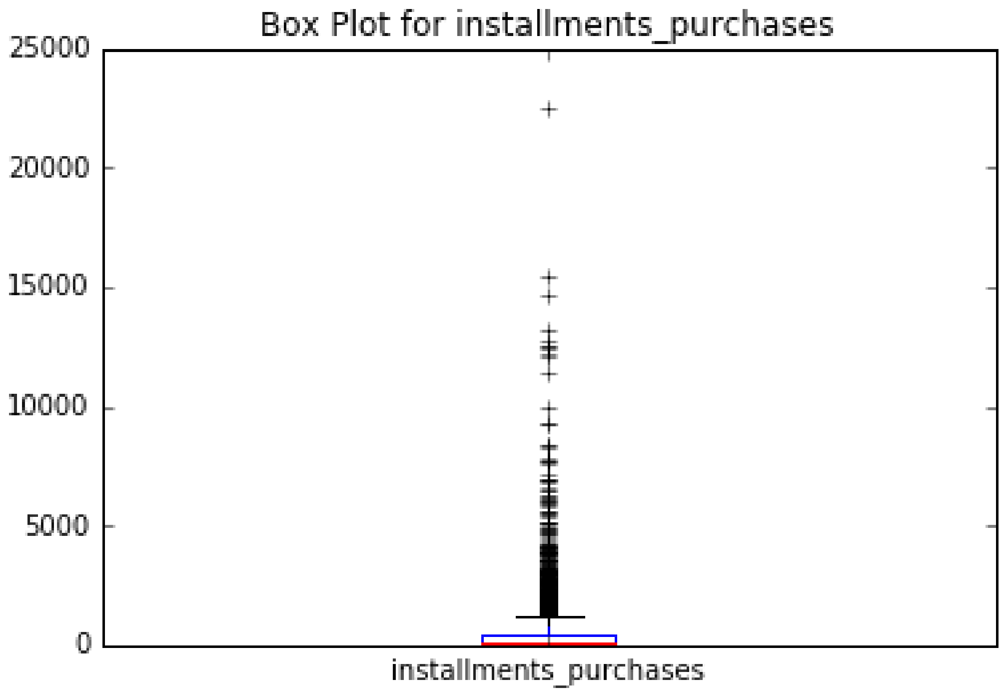
```
std                    0.200121          6.824647      24.857649   3638.815725
min                    0.000000          0.000000       0.000000     50.000000
25%                    0.000000          0.000000       1.000000   1600.000000
50%                    0.000000          0.000000       7.000000   3000.000000
75%                    0.222222          4.000000      17.000000   6500.000000
max                    1.500000        123.000000     358.000000  30000.000000

           payments  minimum_payments  prc_full_payment       tenure
count   8950.000000       8637.000000       8950.000000  8950.000000
mean    1733.143852        864.206542          0.153715    11.517318
std     2895.063757       2372.446607          0.292499     1.338331
min        0.000000          0.019163          0.000000     6.000000
25%      383.276166        169.123707          0.000000    12.000000
50%      856.901546        312.343947          0.000000    12.000000
75%     1901.134317        825.485459          0.142857    12.000000
max    50721.483360      76406.207520          1.000000    12.000000
```

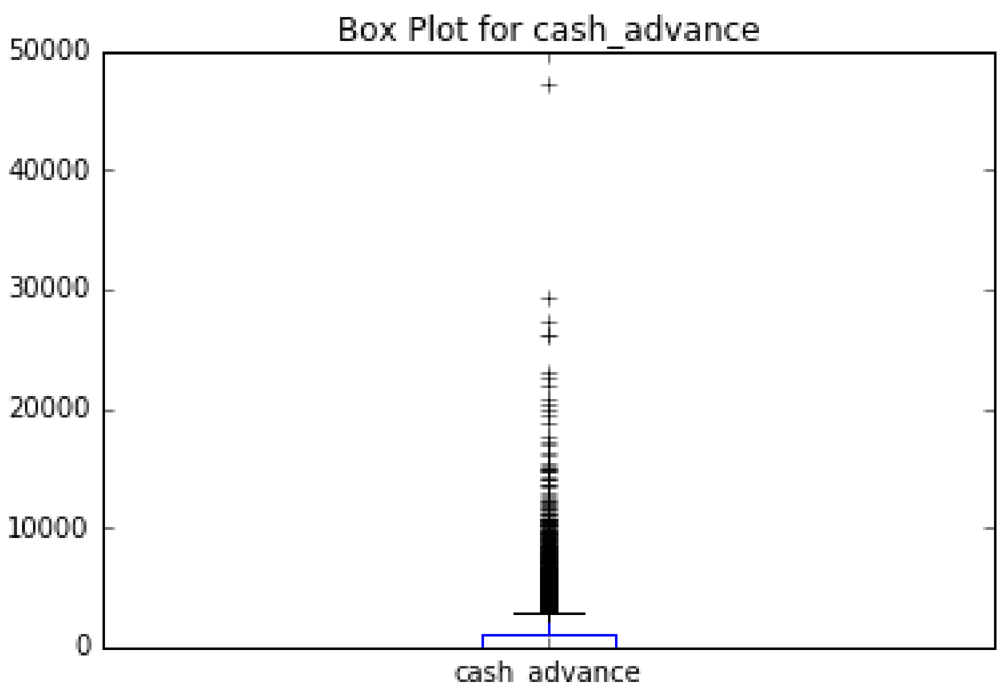
In [6]:
```python
# Let's view the distribution of the data, where is it possible to find groups?
# We are using boxplots of all the columns except the first (cust_id which is a string)
for col in data.columns[2:]:
    data[col].plot(kind='box')
    plt.title('Box Plot for '+col)
    plt.show()
```

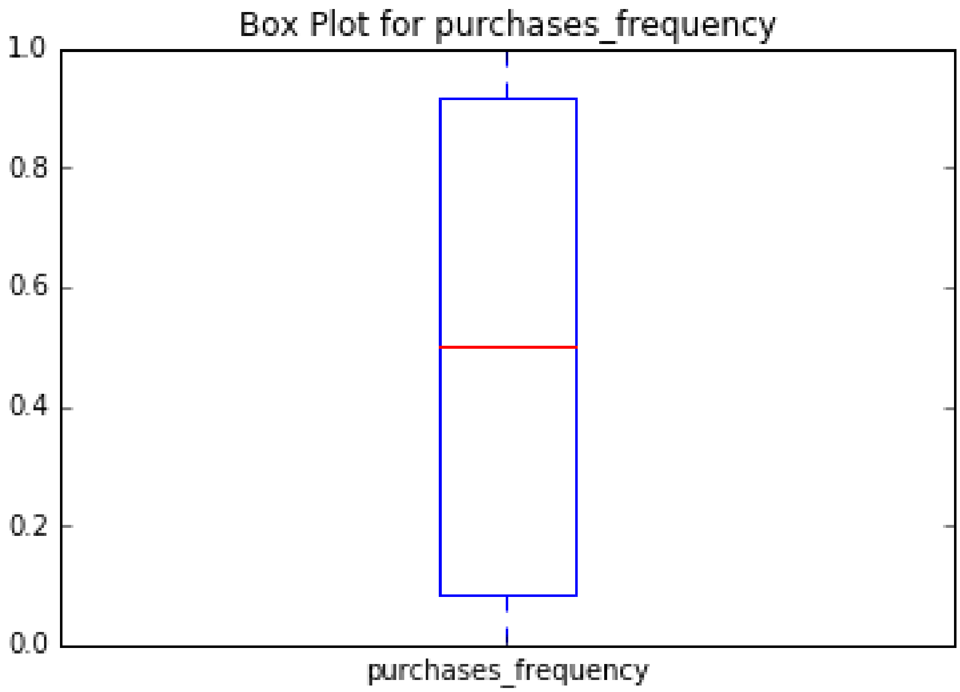


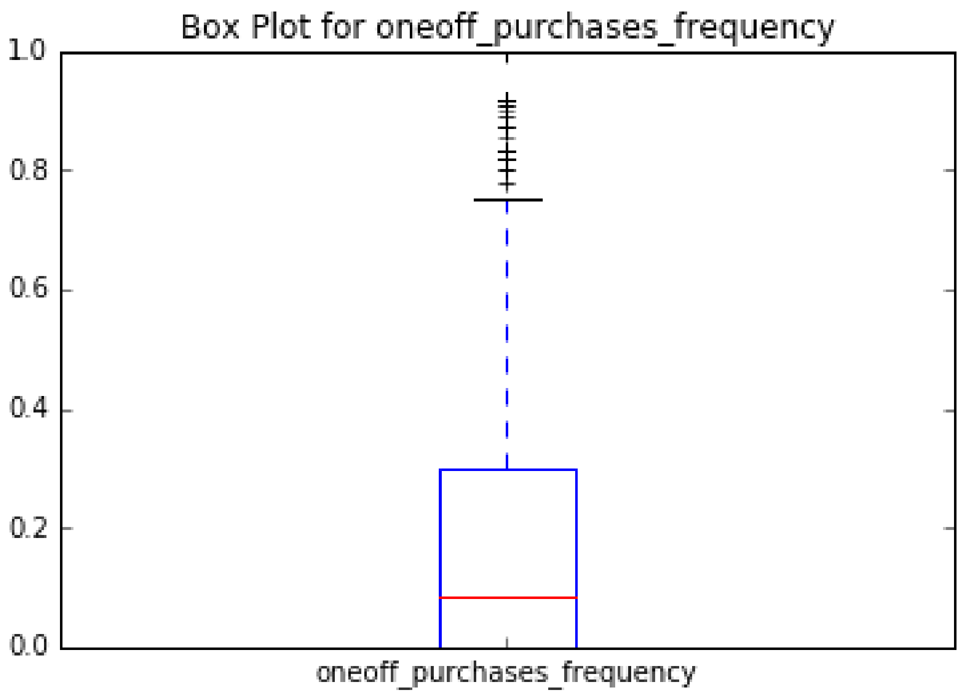

Box Plot for purchases
50000
40000
30000
20000
10000
0
purchases

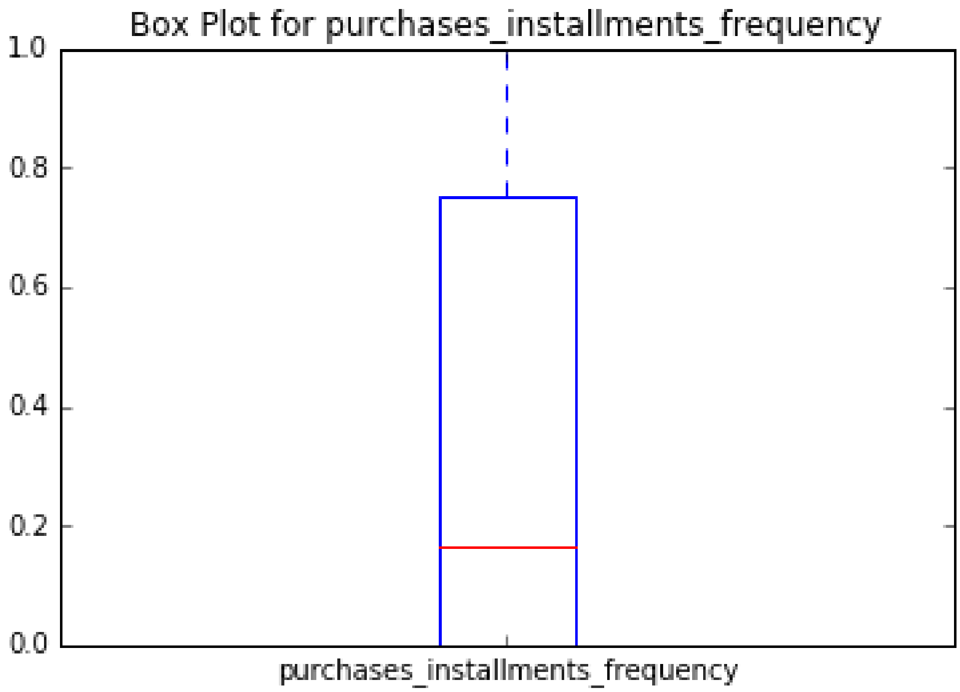

Box Plot for oneoff_purchases
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
oneoff_purchases

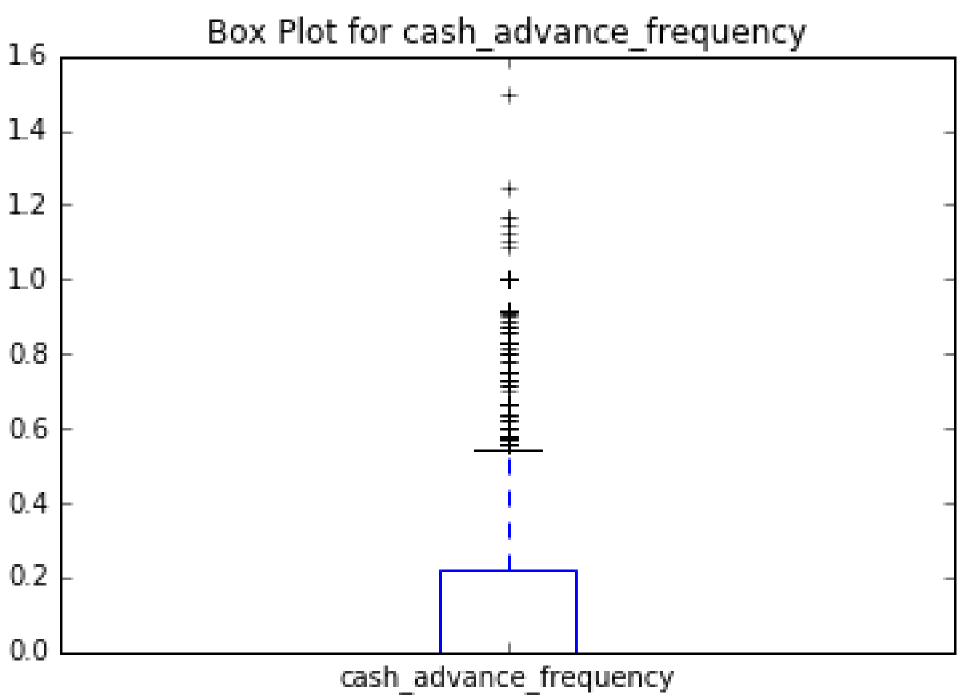

Box Plot for installments_purchases
25000
20000
15000
10000
5000
0
installments_purchases

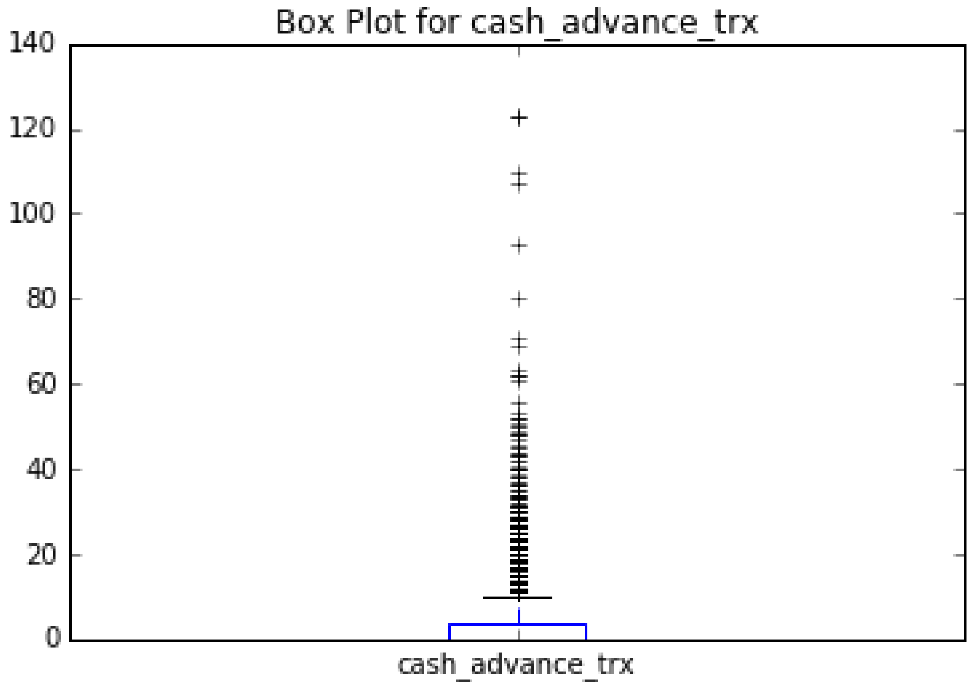

Box Plot for cash_advance
50000
40000
30000
20000
10000
0
cash_advance


Box Plot for purchases_frequency
1.0
0.8
0.6
0.4
0.2
0.0
purchases_frequency

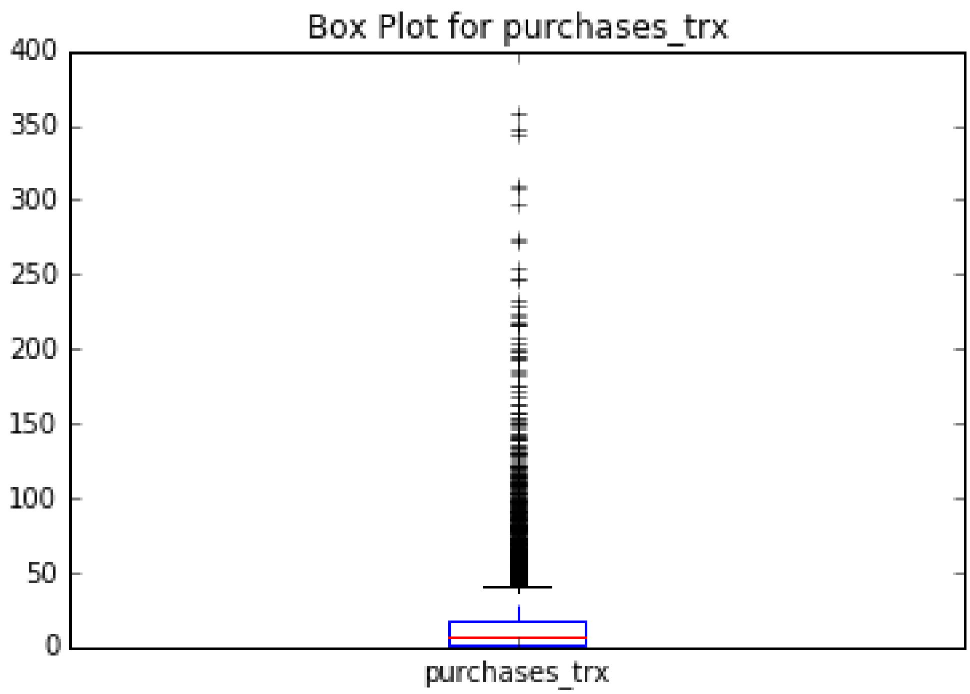

Box Plot for oneoff_purchases_frequency
1.0
0.8
0.6
0.4
0.2
0.0
oneoff_purchases_frequency


Box Plot for purchases_installments_frequency
1.0
0.8
0.6
0.4
0.2
0.0
purchases_installments_frequency

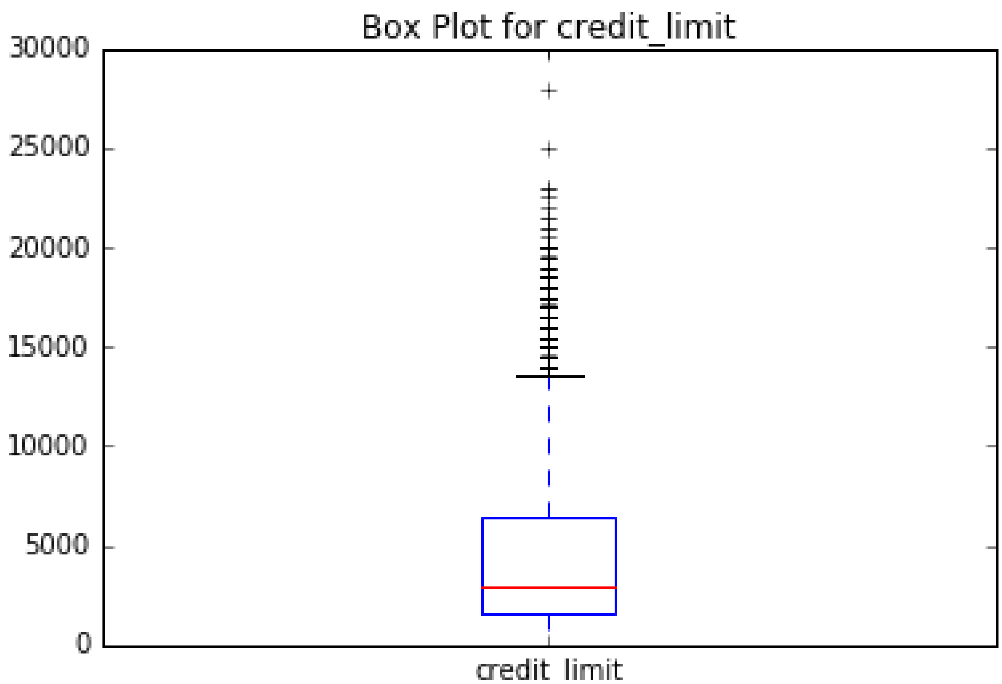

Box Plot for cash_advance_frequency
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
cash_advance_frequency

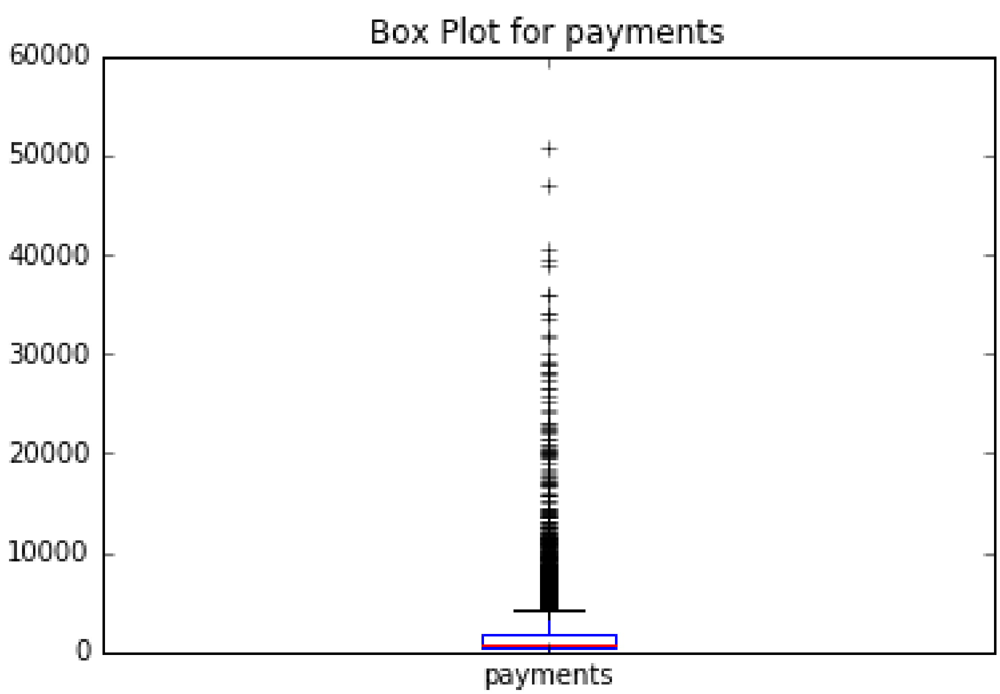


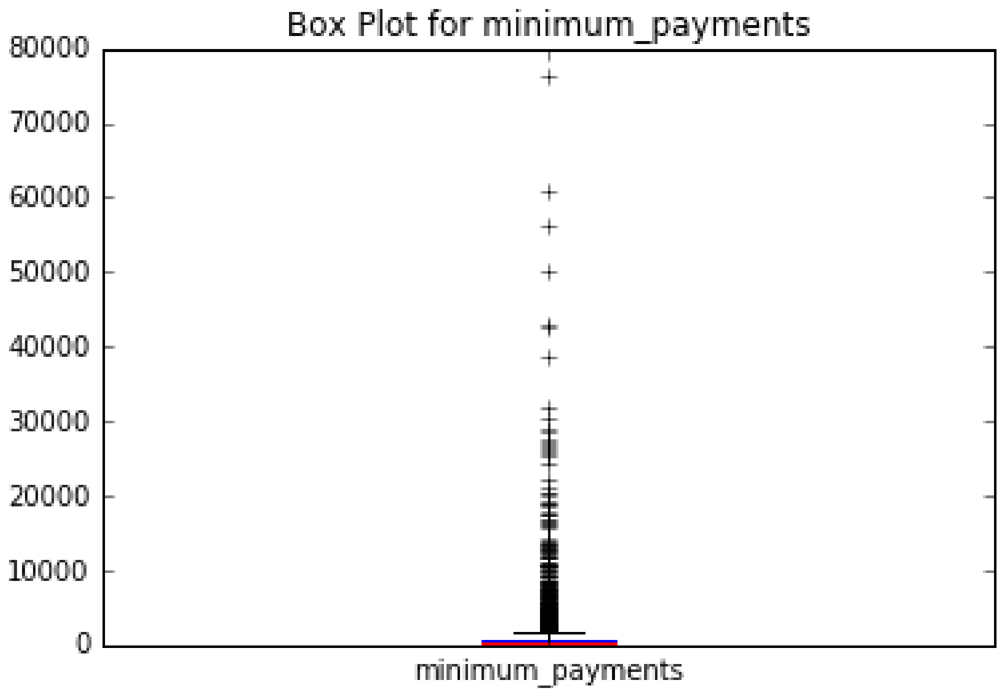
Box Plot for cash_advance_trx
140
120
100
80
60
40
20
0
cash_advance_trx

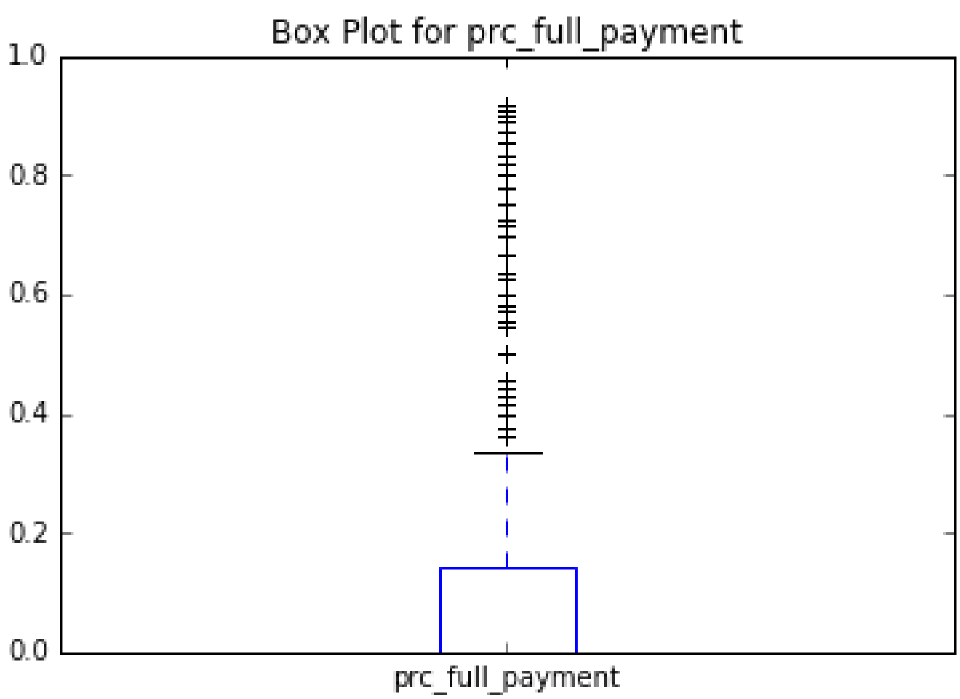


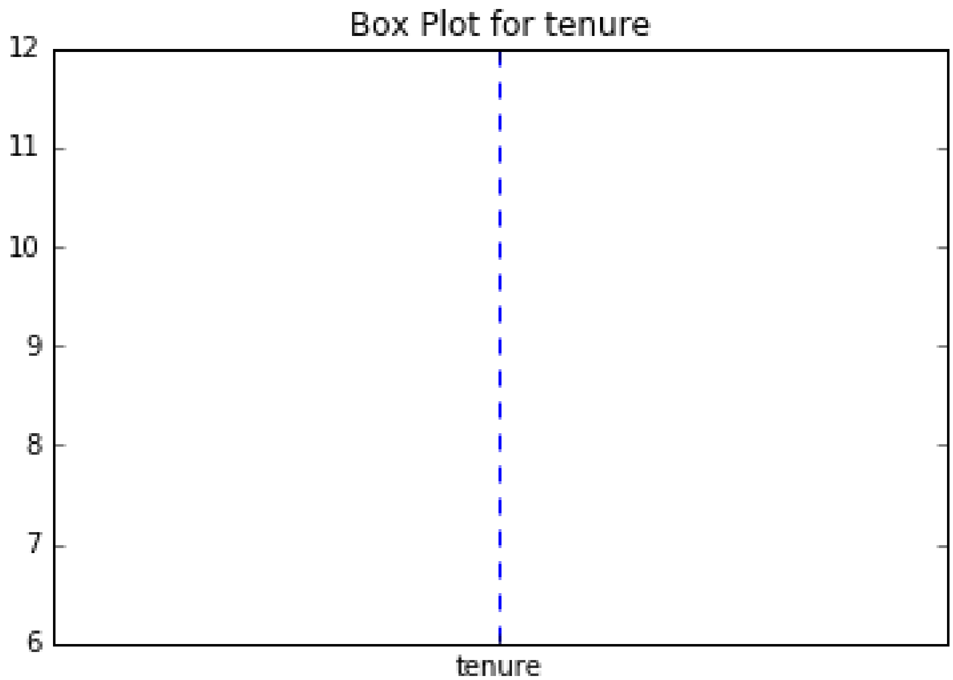
Box Plot for purchases_trx
400
350
300
250
200
150
100
50
0
purchases_trx


Box Plot for credit_limit
30000
25000
20000
15000
10000
5000
0
credit_limit


Box Plot for payments
60000
50000
40000
30000
20000
10000
0
payments


Box Plot for minimum_payments
80000
70000
60000
50000
40000
30000
20000
10000
0
minimum_payments


Box Plot for prc_full_payment
1.0
0.8
0.6
0.4
0.2
0.0
prc_full_payment



1.5 Aim: Can we identify groups based on purchases and payments?

If that is the case, we could offer different payment plans based on different purchases.

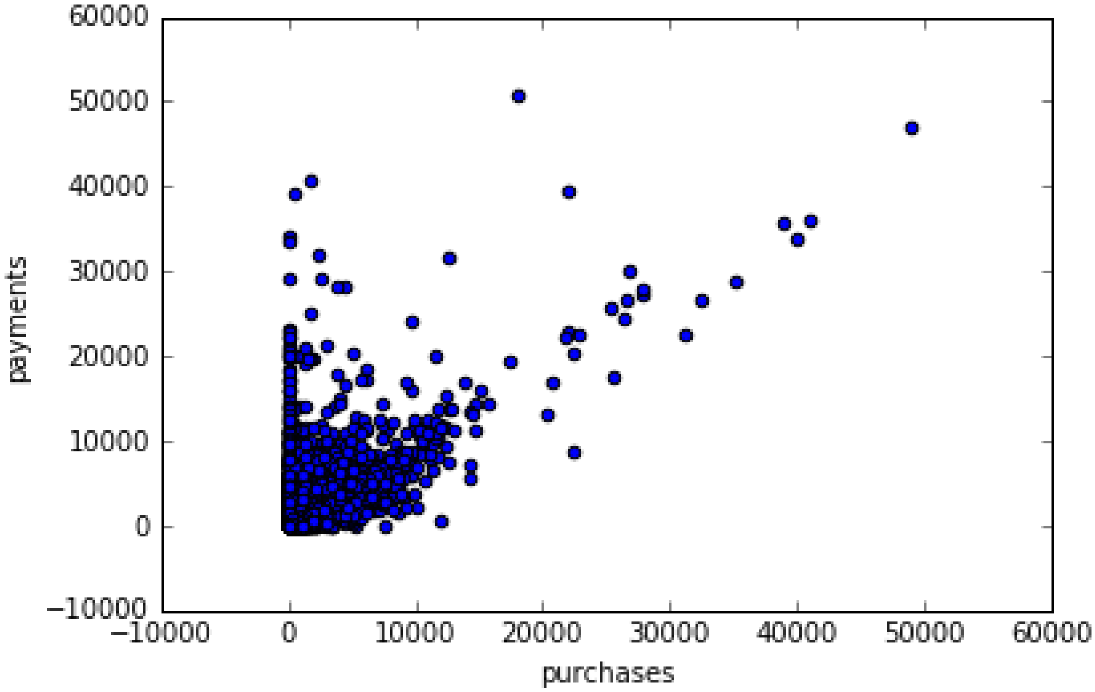
In [7]:
```
cluster_data = data[['purchases','payments']]
cluster_data.head()
```

Out[7]:

	purchases	payments
0	95.40	201.802084
1	0.00	4103.032597
2	773.17	622.066742
3	1499.00	0.000000
4	16.00	678.334763

In [8]:
```
cluster_data.plot(kind='scatter',x='purchases',y='payments')
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7f836b17c4a8>




```
In [9]: # Is there any missing data
        missing_data_results = cluster_data.isnull().sum()
        print(missing_data_results)

        # perform imputation with median values
        # not require since none missing
        #cluster_data = cluster_data.fillna( data.median() )
```

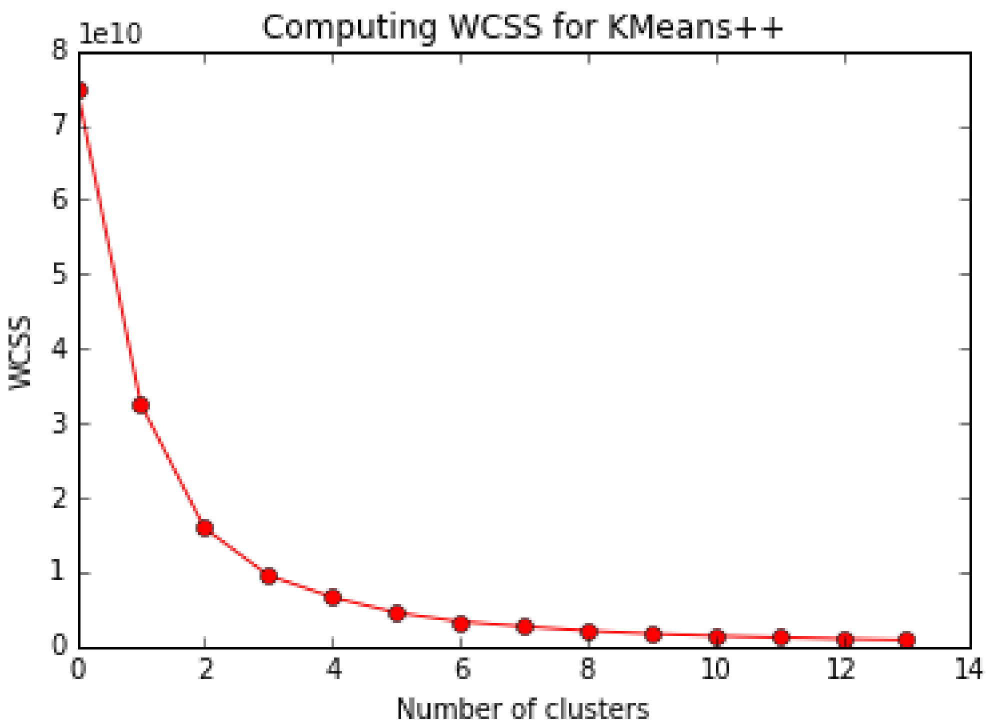
```
purchases    0
payments     0
dtype: int64
```

```
In [10]: #retrieve just the values for all columns except customer id
         data_values = cluster_data.iloc[ :, 1:].values
         data_values
```

```
Out[10]: array([[ 201.802084],
               [4103.032597],
               [ 622.066742],
               ...,
               [  81.270775],
               [  52.549959],
               [  63.165404]])
```

In [11]:
```
#import KMeans algorithm
from sklearn.cluster import KMeans
```

In [12]:
```
# Use the Elbow method to find a good number of clusters using WCSS (within-cluster sum
wcss = []
for i in range( 1, 15 ):
    kmeans = KMeans(n_clusters=i, init="k-means++", n_init=10, max_iter=300)
    kmeans.fit_predict( data_values )
    wcss.append( kmeans.inertia_ )

plt.plot( wcss, 'ro-', label="WCSS")
plt.title("Computing WCSS for KMeans++")
plt.xlabel("Number of clusters")
plt.ylabel("WCSS")
plt.show()
```



We're seeing an elbow at approx 3, so let's try 3 groups

In [13]:
```
kmeans = KMeans(n_clusters=3, init="k-means++", n_init=10, max_iter=300)
cluster_data["cluster"] = kmeans.fit_predict( data_values )
cluster_data
```

```
/usr/local/lib/python3.5/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

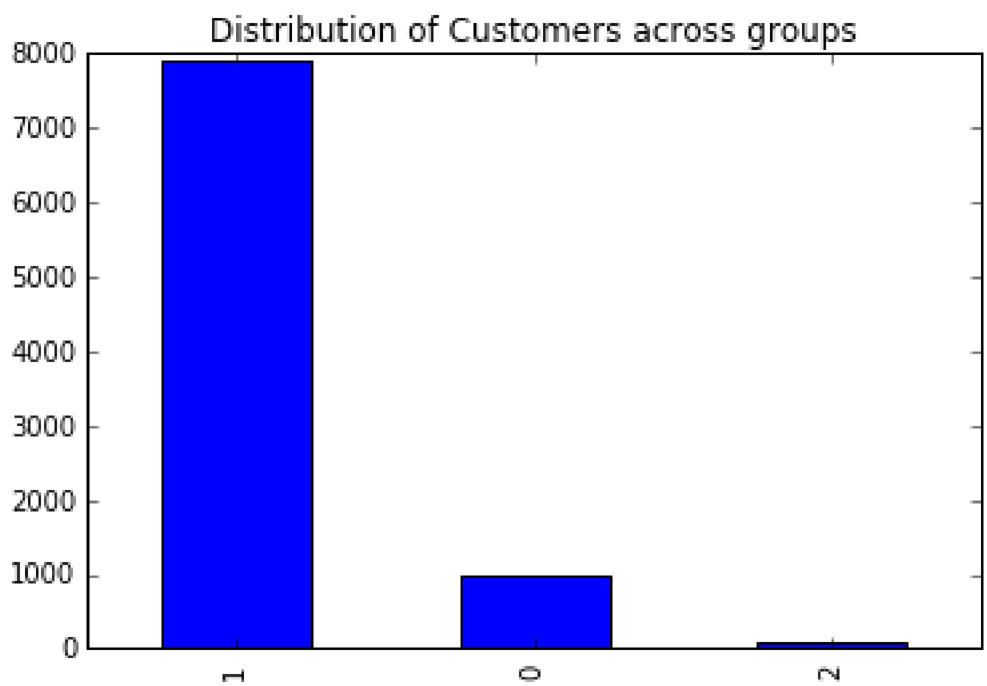
```
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
```

```
Out[13]:       purchases     payments  cluster
         0         95.40   201.802084        1
         1          0.00  4103.032597        0
         2        773.17   622.066742        1
         3       1499.00     0.000000        1
         4         16.00   678.334763        1
         5       1333.28  1400.057770        1
         6       7091.01  6354.314328        0
         7        436.20   679.065082        1
         8        861.49   688.278568        1
         9       1281.60  1164.770591        1
         10       920.12  1083.301007        1
         11      1492.18   705.618627        1
         12      3217.99   608.263689        1
         13      2137.93  1655.891435        1
         14         0.00   805.647974        1
         15      1611.70  1993.439277        1
         16         0.00   391.974562        1
         17       519.00   254.590662        1
         18       504.35  1720.837373        1
         19       398.64  1053.980464        1
         20       176.68   223.068600        1
         21      6359.95  2077.959051        1
         22       815.90  2359.629958        1
         23      4248.35  9479.043842        0
         24         0.00  1422.726707        1
         25       399.60   215.306142        1
         26       102.00   890.178845        1
         27       233.28   207.773715        1
         28       387.05  1601.448347        1
         29       100.00   160.767773        1
         ...         ...          ...      ...
         8920       0.00    54.795084        1
         8921      57.42    68.462579        1
         8922     145.98    53.676054        1
         8923    1898.88   669.039640        1
         8924      74.00   214.921009        1
         8925     418.59   422.538988        1
         8926     580.00   641.303466        1
         8927     315.20   231.274641        1
         8928     500.00   456.745027        1
         8929       0.00     0.000000        1
         8930      84.00   124.373736        1
```

```
8931     235.80   189.090274        1
8932     180.00   138.203240        1
8933     619.60   106.138603        1
8934     110.50   161.476789        1
8935     465.90     0.000000        1
8936     712.50   605.716356        1
8937       0.00   117.738787        1
8938       0.00  1397.770131        1
8939     734.40    72.530037        1
8940     591.24   475.523262        1
8941     214.55   966.202912        1
8942     113.28    94.488828        1
8943      20.90    58.644883        1
8944    1012.73     0.000000        1
8945     291.12   325.594462        1
8946     300.00   275.861322        1
8947     144.40    81.270775        1
8948       0.00    52.549959        1
8949    1093.25    63.165404        1

[8950 rows x 3 columns]
```

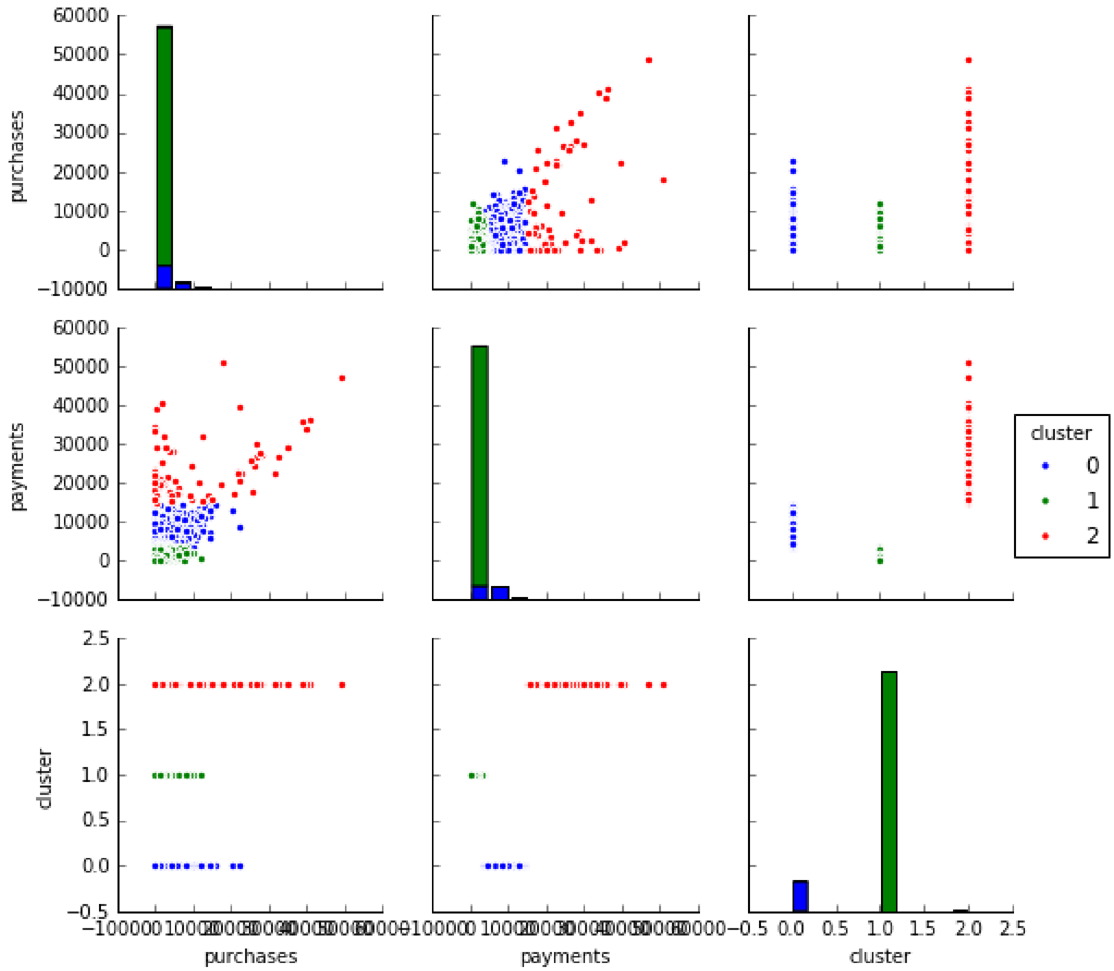
In [24]:
```
cluster_data['cluster'].value_counts().plot(kind='bar',title='Distribution of Customers
```

Out[24]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f83647e24a8>
```



```
In [15]: sns.pairplot( cluster_data, hue="cluster")
```

```
Out[15]: <seaborn.axisgrid.PairGrid at 0x7f836dd89f98>
```



Looks nice, but what are we really seeing? Let's attempt to describe the groups

```
In [17]: grouped_cluster_data = cluster_data.groupby('cluster')
         grouped_cluster_data
```

```
Out[17]: <pandas.core.groupby.DataFrameGroupBy object at 0x7f836531b128>
```

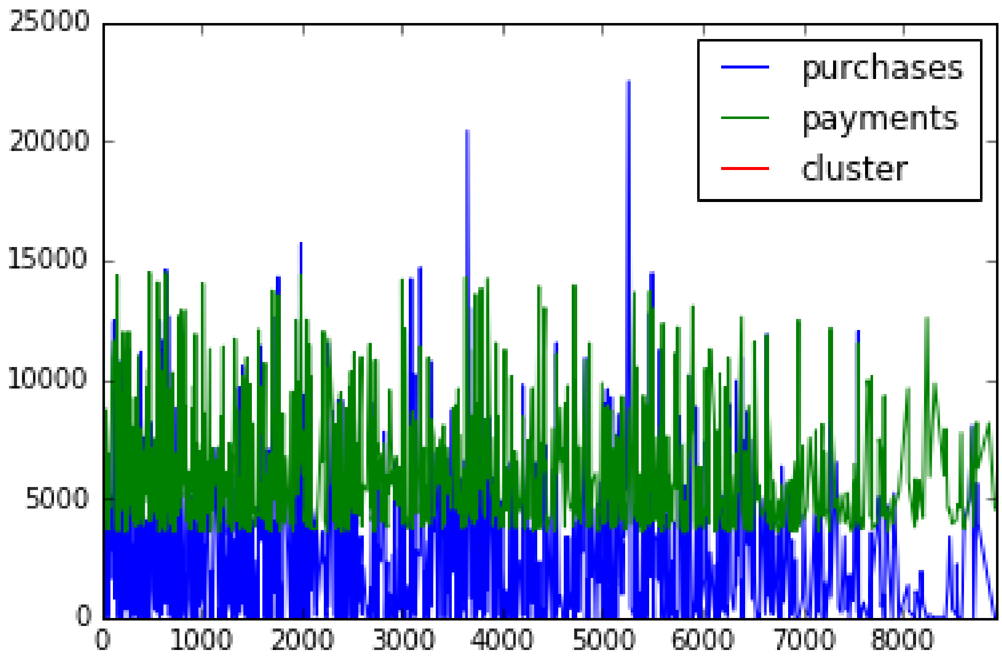
```
In [18]: grouped_cluster_data.describe()
```

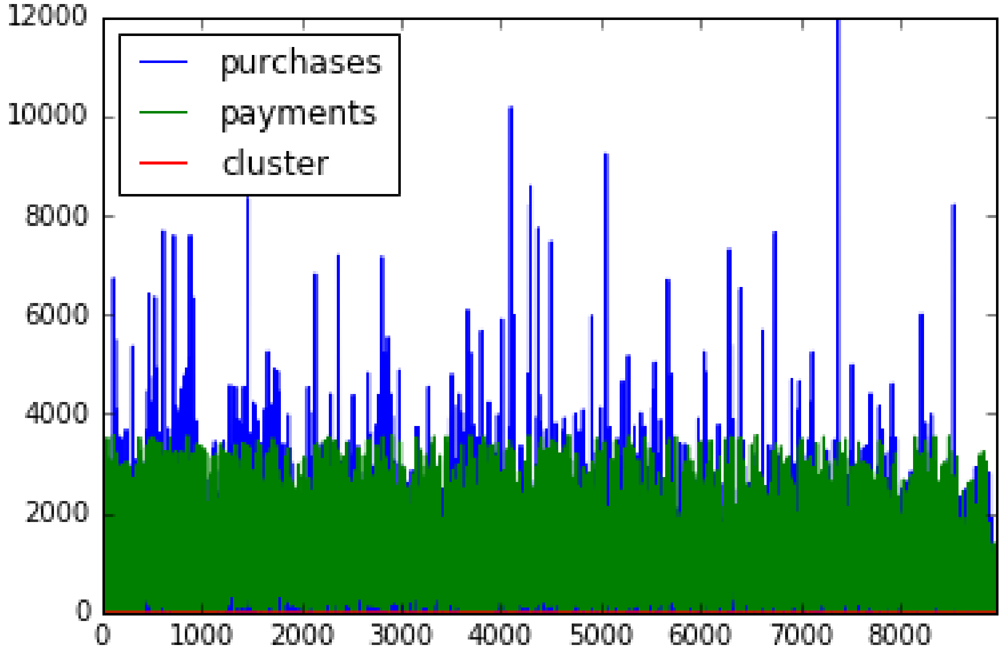
```
Out[18]:                     purchases      payments   cluster
         cluster
         0         count    977.000000    977.000000       977
                   mean    2893.507165   6159.668584         0
                   std     3248.509337   2505.563838         0
```

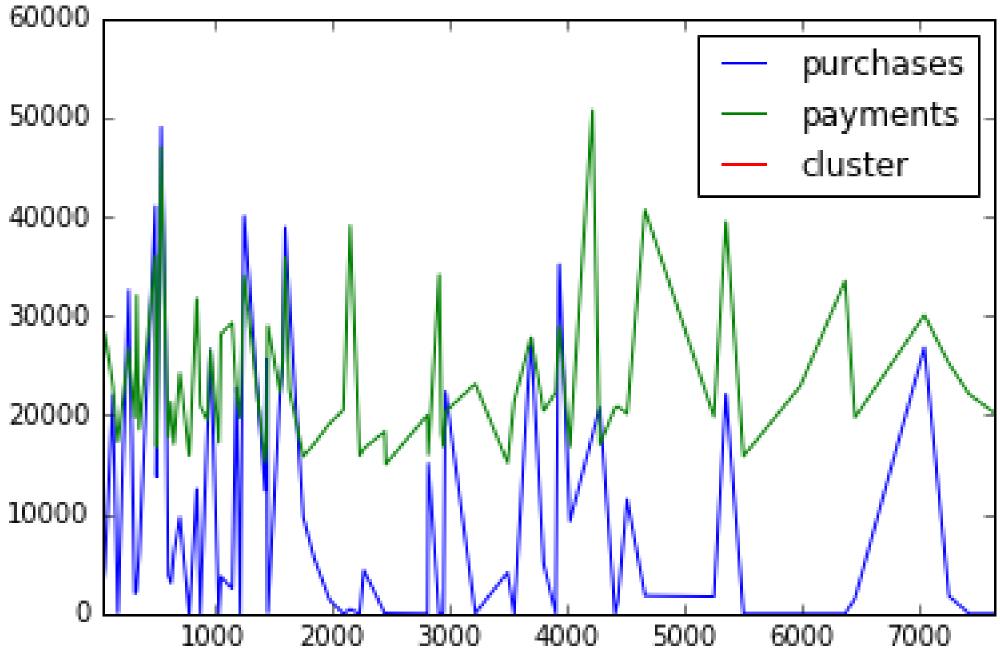
```
        min          0.000000   3569.182969      0
        25%        147.120000   4201.694496      0
        50%       1771.600000   5359.784024      0
        75%       4587.210000   7443.805810      0
        max      22500.000000  14481.465440      0
1       count     7899.000000   7899.000000   7899
        mean       680.066701    980.439425      1
        std        965.459756    829.030774      0
        min          0.000000      0.000000      1
        25%         35.000000    341.445479      1
        50%        324.000000    708.699641      1
        75%        924.645000   1411.360861      1
        max      11994.710000   3567.009988      1
2       count       74.000000     74.000000     74
        mean     10538.917432  23637.165609      2
        std      12900.044316   7698.746165      0
        min          0.000000  15043.665080      2
        25%         30.000000  17827.712822      2
        50%       3976.770000  20907.273725      2
        75%      21538.785000  27693.619535      2
        max      49039.570000  50721.483360      2
```

In [20]:
```
grouped_cluster_data.plot(subplots=True,)
```

Out[20]:
```
cluster
0     Axes(0.125,0.125;0.775x0.775)
1     Axes(0.125,0.125;0.775x0.775)
2     Axes(0.125,0.125;0.775x0.775)
dtype: object
```




purchases
payments
cluster
12000
10000
8000
6000
4000
2000
0
0
1000
2000
3000
4000
5000
6000
7000
8000


purchases
payments
cluster
60000
50000
40000
30000
20000
10000
0
1000
2000
3000
4000
5000
6000
7000

1.6 In - class assignment

1. Provide three(3) plots of the data to assist in describing the initial data set
2. Plot the differences between the groups above using at least two (2) charts.
3. Repeat the clustering activity on different columns in an attempt to provide additional marketing insight. If the results are not insightful state why