# Finding the Best Threshold

May 4, 2021

## 1 Finding the Best Threshold

This notebook determines the best threshold/time window combinations for identifying episodic and chronic shelter users.



[2]:
```
%load_ext autoreload
%autoreload 1
```

[3]:
```
import numpy as np
import pandas as pd
import datetime, copy, imp
import time
import matplotlib.pyplot as plt
import sys


from tqdm.auto import tqdm, trange
from tqdm.notebook import tqdm
tqdm.pandas()

%aimport di_data
from di_data import *
import KMeansCluster as km
```

### 1.0.1 Pre-Processing

[4]:
```
dirStr = '~/data/plwh/'
```

[5]:
```
def PreProcess():

    tblAll = pd.read_hdf(dirStr + 'UniversityExportAnonymized.hd5')

    tblAll = tblAll[tblAll.Date >= pd.to_datetime('2007-07-01')]

    print('Total Entries: {}'.format(len(tblAll.index)))
    print('Dates: ',min(tblAll.Date),' to ',max(tblAll.Date))

    tbl = copy.deepcopy(tblAll[tblAll.EntryType == 'Sleep'][␣
↪['Date','ClientId'] ])

    # To address left censoring: Remove all clients with first sleep date␣
↪within a year of the 2008 data import.
    leftStart = tbl.Date.min()
    leftEnd = pd.to_datetime('2009-07-01')

    # To address right censoring: Remove all clients with first sleep date␣
↪within approximately 2 years of the end
    # of the data.  Reasoning: We want to allow a 2 year window to give the␣
↪clients a chance to become chronic.
    rightStart = pd.to_datetime('2018-01-20')
    rightEnd = tbl.Date.max()

    nClientsAll = len(tbl.ClientId.unique())

    tbl = RemoveByStartDate(tbl,leftStart,leftEnd)
    nLeftRemoved = nClientsAll - len(tbl.ClientId.unique())

    tbl = RemoveByStartDate(tbl,rightStart,rightEnd)
    nRightRemoved = nClientsAll - nLeftRemoved - len(tbl.ClientId.unique())


    # Discard all data before September 1, 2008 due to import errors from␣
↪previous database.
    tbl = tbl.loc[tbl.Date >= pd.to_datetime('2008-09-01')]

    nClients = len(tbl.ClientId.unique())

    print('Total Clients: {:d}/{:d} ({:.1f}%) ({:d} removed left, {:d} removed␣
↪right)'
          .format(nClients,nClientsAll,100.0*nClients/
↪nClientsAll,nLeftRemoved,nRightRemoved))
```

```
    return tbl
```

[6]:
```
tbl = PreProcess()
```

```
Total Entries: 5431521
Dates:  2007-07-01 00:00:00  to  2020-01-20 00:00:00
Total Clients: 18398/34577 (53.2%) (12609 removed left, 3570 removed right)
```

[ ]:
```

```

### 1.0.2 Identify Episodes and Stays

[7]:
```
def GenStays():
    return tbl.groupby('ClientId').progress_apply(CalculateStaySequence)
```

[9]:
```
stays = GenStays()
```

```
  0%|          | 0/18398 [00:00<?, ?it/s]
```

[10]:
```
def GenEpis():
    return tbl.groupby('ClientId').progress_apply(CalculateEpisodeSequence)
```

[11]:
```
epis = GenEpis()
```

```
  0%|          | 0/18398 [00:00<?, ?it/s]
```

[12]:
```
# Determine the total number of stays and episodes for each client and store in␣
 ↪seDat.
def JointStaysAndEpis():
    nClients = len(tbl.ClientId.unique())
    clientInds = stays.index.get_level_values(0).drop_duplicates()
    seDat = pd.DataFrame({
        'nSty': [0]*nClients,    # Stores total stays.
        'nEpi': [0]*nClients }, # Stores total episodes.
        index=clientInds)
    for id in tqdm(clientInds):
        seDat.loc[id,'nSty'] = stays.loc[id].Ind.max() # Number of stays is max␣
 ↪stay index.
        seDat.loc[id,'nEpi'] = epis.loc[id].Ind.max()  # Number of episodes is␣
 ↪max episode index.

    return seDat
```

[13]:
```
seDat = JointStaysAndEpis()
```

```
  0%|          | 0/18398 [00:00<?, ?it/s]
```

[ ]:

```

```

[14]:

```
# Determine the number of new clients that register per month.
regDates = list(tbl.groupby('ClientId').Date.min())  # List of first dates that␣
 ↪each client appears.

# Resample the dates to determine how many new registrants we have per month.
nClients = len(tbl.ClientId.unique())
monthlyReg = pd.DataFrame([1]*nClients,index=regDates).groupby(pd.
 ↪Grouper(freq='M')).count()

print('New Clients per Month - Mean: %g, Median: %g' % (monthlyReg.
 ↪mean(),monthlyReg.median()))
```

```
New Clients per Month - Mean: 178.621, Median: 172
```

### 1.0.3 Perform Cluster Analysis

[15]:

```
k = 3  # Established in previous work to yield the transitional, chronic and␣
 ↪episodic clusters.

cluster = km.KMeansCluster(k,seDat)
(mus,clstrAsgn,pTable) = cluster.Solve(40,0.005)

# The cluster indices representing chronic, episodic and transitional can␣
 ↪change from run to
# run due to how the cluster analysis is randomly initialized.  This code␣
 ↪determines those indices.
chrId = np.argmax(mus[:,0])  # Chronic clients have the highest number of stays.
epiId = np.argmax(mus[:,1])  # Episodic clients have the highest number of␣
 ↪episodes.
tmpId = list( set([0,1,2]) - set([chrId,epiId]) )[0]  # Transitional clients␣
 ↪are the group left over.
styMuInd = 0
epiMuInd = 1

pTable
```

[15]:

| | i | j | pVal |
|---|---|---|---|
| 0 | 0 | 1 | 1.110223e-16 |
| 1 | 0 | 2 | 1.110223e-16 |
| 2 | 1 | 2 | 1.110223e-16 |

[16]:

```
# Calculate statistics on the three clusters to compare to previously published␣
 ↪results.

tmpN = sum(clstrAsgn == tmpId)  # Total number of clients in each group.
```

```
chrN = sum(clstrAsgn == chrId)
epiN = sum(clstrAsgn == epiId)

print('Trans (%d/%d, %.1f%%) > AvgStays: %g, AvgEpisodes: %g, '
      % (tmpN,nClients,100*tmpN/
 ↪nClients,mus[tmpId,styMuInd],mus[tmpId,epiMuInd]))
print('Epi   (%d/%d, %.1f%%) > AvgStays: %g, AvgEpisodes: %g, '
      % (epiN,nClients,100*epiN/
 ↪nClients,mus[epiId,styMuInd],mus[epiId,epiMuInd]))
print('Chron (%d/%d, %.1f%%) > AvgStays: %g, AvgEpisodes: %g, '
      % (chrN,nClients,100*chrN/
 ↪nClients,mus[chrId,styMuInd],mus[chrId,epiMuInd]))

print('Max stays for single episode transitional client: ',
      seDat[(clstrAsgn == tmpId) & (seDat.nEpi == 1)].nSty.max())
```

```
Trans (15675/18398, 85.2%) > AvgStays: 30.3157, AvgEpisodes: 1.81684,
Epi   (2184/18398, 11.9%) > AvgStays: 166.994, AvgEpisodes: 9.19048,
Chron (539/18398, 2.9%) > AvgStays: 1273.07, AvgEpisodes: 3.65863,
Max stays for single episode transitional client:  662
```

**Demographics**

[18]:
```
def CalculateClientDemographics():
    return tbl.groupby('ClientId').progress_apply(ShelterGroupDemographics)
```

[19]:
```
demog = CalculateClientDemographics()
```

```
  0%|          | 0/18398 [00:00<?, ?it/s]
```

[ ]:
```

```

### 1.0.4 Threshold/Window Optimization

[20]:
```
dayWinRange = [ 30, 90, 180, 365, 547 ]
styThshFrac = np.array([ 0.5, 0.75, 0.9 ])
epiThshRange = [ 2, 3, 4, 5 ]
```

[21]:
```
def OptimizeStayThreshold():
    styOpt = { d: { t: pd.DataFrame() for t in (styThshFrac*d).astype(int)} for␣
 ↪d in dayWinRange }
    tBar = tqdm( total = len(styThshFrac)*len(dayWinRange) )
    for win in dayWinRange:
        for thsh in (win*styThshFrac).astype(int):
            styOpt[win][thsh] = stays.groupby('ClientId').
 ↪apply(TimeWinThresholdTest,posFlag='sty',negFlag='tmp',thresh=thsh,winSzDays=win)
            tBar.update()
```

```
    tBar.close()
    return styOpt
```

[22]:
```
styOpt = OptimizeStayThreshold()
```

```
0%|          | 0/15 [00:00<?, ?it/s]
```

[23]:
```
def OptimizeEpiThreshold():

    epiOpt = { d: { t: pd.DataFrame() for t in epiThshRange} for d in␣
 ↪dayWinRange }
    tBar = tqdm( total = len(epiThshRange)*len(dayWinRange) )
    for win in dayWinRange:
        for thsh in epiThshRange:
            epiOpt[win][thsh] = epis.groupby('ClientId').
 ↪apply(TimeWinThresholdTest,posFlag='epi',negFlag='tmp',thresh=thsh,winSzDays=win)
            tBar.update()

    tBar.close()
    return epiOpt
```

[24]:
```
epiOpt = OptimizeEpiThreshold()
```

```
0%|          | 0/20 [00:00<?, ?it/s]
```

### 1.0.5 Evaluate Impact of Interventions

[25]:
```
# Determines the number of shelter stays and days of shelter interaction saved␣
 ↪if a client
# identified as chronic or episodic is placed in a house the day they're␣
 ↪identified.
def CalcImpact(
    testList,  # Client list after being processed by a shelter stay test.
    stays      # Stay table.
    ):

    staysSaved = 0
    tenureDaysSaved = 0

    # Loop through all clients that are not transitional.
    for clientId in testList[testList.Flag != 'tmp'].index:

        # Find the stays that occur after the client is identified
        saved = stays.loc[clientId][stays.loc[clientId].Date >= testList.
 ↪loc[clientId].Date]
```

```
        staysSaved += saved.Ind.count()-1
        tenureDaysSaved += (saved.Date.iloc[-1]-testList.loc[clientId].Date).
↪days

    nClientsIdentified = sum(testList.Flag != 'tmp')

    return np.array([
        nClientsIdentified, staysSaved, tenureDaysSaved,
        testList[testList.Flag!='tmp'].Time.sum(),
        testList[testList.Flag!='tmp'].Time.median()
    ])
```

[26]:
```
def EvalThresholdImpacts(opt):

    keys1dim = list(opt.keys())
    nKeys = len(keys1dim) * len(opt[keys1dim[0]])
    impact = np.zeros((nKeys,7))

    iRow = 0
    tBar = tqdm(total=nKeys)
    for win in opt.keys():
        for thsh in opt[win].keys():
            impact[iRow,:] = np.concatenate( (np.
↪array([win,thsh]),CalcImpact(opt[win][thsh],stays)) )
            iRow += 1
            tBar.update()

    tBar.close()
    return impact
```

[27]:
```
impInd = { 'Thsh': 0, 'Win': 1, 'N': 2, 'TotStySv': 3, 'TotTnSv': 4, 'IdTTot':␣
↪5, 'IdTMd': 6 }
```

[28]:
```
impactEpi = EvalThresholdImpacts(opt=epiOpt)
```

```
0%|          | 0/20 [00:00<?, ?it/s]
```

[29]:
```
impactSty = EvalThresholdImpacts(opt=styOpt)
```

```
0%|          | 0/15 [00:00<?, ?it/s]
```

[30]:
```
def PrintImpact(tbl,typeFlg,nClients,nTop=10):
    iThsh=1; iWin=0; iN=2; iTotStySv=3; iTotTnSv=4; iIdTTot=5; iIdTMd=6

    print('Win/Thsh     n       StySvPr TenRdPr AvgSpd MdSpd')
    print('-------------------------------------------------')
```

```
    for rw in tbl[0:nTop]:
        print('%3d/%3d  %4d,%4.1f%%  %5.1f   %5.1f   %5.1f   %5.1f'
              % (int(rw[iWin]), int(rw[iThsh]), int(rw[iN]), 100.0*rw[iN]/
↪nClients,
                 rw[iTotStySv]/rw[iN], rw[iTotTnSv]/rw[iN], rw[iIdTTot]/rw[iN],␣
↪rw[iIdTMd] ) )
```

[31]:
```
print('\nEpi Test (Tenure Reduction Ranking)')
order = ( -impactEpi[:,4] / impactEpi[:,2] ).argsort()
PrintImpact(impactEpi[order,:],'Epi',nClients)
```

```

Epi Test (Tenure Reduction Ranking)
Win/Thsh     n       StySvPr TenRdPr AvgSpd MdSpd
-------------------------------------------------
180/  4   310, 1.7%   87.5   932.9   933.3   731.5
365/  5   618, 3.4%   91.7   906.6   985.2   748.5
365/  3  3476,18.9%   99.0   872.8   602.2   313.0
547/  5  1246, 6.8%   85.3   868.1   925.4   666.5
180/  3  1903,10.3%   85.3   864.7   712.0   460.0
547/  4  2331,12.7%   90.9   858.1   769.9   485.0
365/  4  1581, 8.6%   86.3   857.0   816.9   569.0
547/  3  4201,22.8%  101.2   854.7   572.9   367.0
 90/  2  4277,23.2%   93.6   829.7   477.7   181.0
 90/  3   275, 1.5%   78.3   829.3   871.9   663.0
```

[32]:
```
print('Sty Test (Stays Saved Ranking)')
order = ( -impactSty[:,3] / impactSty[:,2] ).argsort()
PrintImpact(impactSty[order,:],'Chr',nClients)
```

```
Sty Test (Stays Saved Ranking)
Win/Thsh     n       StySvPr TenRdPr AvgSpd MdSpd
-------------------------------------------------
547/492   415, 2.3%  721.4   980.2   852.6   538.0
365/328   594, 3.2%  687.9   996.8   716.4   376.5
547/410   661, 3.6%  656.6   1000.3   787.7   496.0
365/273   904, 4.9%  619.5   1017.8   658.8   350.5
180/162  1075, 5.8%  588.4   1030.1   515.3   222.0
547/273  1138, 6.2%  555.7   1011.6   673.0   421.5
180/135  1583, 8.6%  504.4   1006.4   496.2   188.0
365/182  1536, 8.3%  499.7   1006.3   558.4   287.0
 90/ 81  1815, 9.9%  471.2   979.7   419.3   137.0
180/ 90  2370,12.9%  416.6   997.4   414.7   142.0
```

[33]:
```
rpdChrThsh = 81
rpdChrWin = 90
rpdEpiThsh = 2
```

```
rpdEpiWin = 90
```

### 1.0.6 Compare Definitions

```
[34]: def CalcRapidChronicDefinition():
          return stays.groupby('ClientId').
      ↪progress_apply(TimeWinThresholdTest,posFlag='chr',negFlag='tmp',thresh=rpdChrThsh,winSzDays
```

```
[36]: defRpdChr = CalcRapidChronicDefinition()
```

```
  0%|          | 0/18398 [00:00<?, ?it/s]
```

```
[37]: def CalcRapidEpisodicDefinition():
          return epis.groupby('ClientId').
      ↪progress_apply(TimeWinThresholdTest,posFlag='epi',negFlag='tmp',thresh=rpdEpiThsh,winSzDays
```

```
[38]: defRpdEpi = CalcRapidEpisodicDefinition()
```

```
  0%|          | 0/18398 [00:00<?, ?it/s]
```

```
[39]: defRpd = ChooseEarliestTest(defRpdChr,defRpdEpi)
```

```
[40]: # Checks for clients are are 'continually homeless' for a threshold number of␣
      ↪days.
      # - We define this as having an episode of homelessness longer than the␣
      ↪threshold.
      def continually_homeless(tbl,flagStr,reqDuration):

          gapVals = tbl.Date.diff().astype('timedelta64[D]')  # Gaps are the␣
      ↪difference between demog dates.

          # Give each episode of shelter demoges a unique index number.
          gapInd = (gapVals >= episodeGap).astype('int').cumsum()

          for iGap in range(max(gapInd)+1):
              startDate = tbl[gapInd==iGap].Date.min()
              endDate = tbl[gapInd==iGap].Date.max()
              curDuration = (endDate - startDate).days + 1

              if curDuration > reqDuration:
                  # Find first date the client's episode exceeded the threshold.
                  idDate = tbl[(tbl.Date - startDate).dt.days >= reqDuration].Date.
      ↪min()

                  epiStartDate = tbl[gapInd==iGap].Date.iloc[0]
                  overMaxStays = (tbl.Date - epiStartDate).dt.days >= reqDuration
```

```
            return pd.Series({
                'Flag': flagStr,  # Flag indicating test was satisfied.
                'Date': idDate,   # Date client satisfied the test.
                'Time': (idDate - startDate).days  # How long it took to␣
↪satisfy the test.
            })

    # Returned if client doesn't satisfy the test.
    return pd.Series({
        'Flag': 'tmp',
        'Date': pd.NaT,
        'Time': np.nan
    })

# Function test:
#continually_homeless(stays.loc[21910],'cnt',365)
```

[41]:
```
# Helper function that prints out demog statistics for a cohort of clients.
def print_stats(tblStr,tbl,fields):
    print('--- %s ---' % (tblStr))
    nEntry = len(tbl.index)
    print( 'Clients in cohort: %d/%d (%.1f%%)' % (nEntry,nClients,100*nEntry/
↪nClients))
    print( 'Size of 10%% of cohort: %d/%d (%.1f%%)' % (nEntry*0.
↪1,nClients,100*nEntry*0.1/nClients))
    for field in fields:
        print('%s:' % (field))
        nEntry = sum(~np.isnan(tbl[field]))

        print(' Avg: %.1f, Med: %.1f, 10thPct: %.1f, 90thPct: %.1f'
              %(tbl[field].mean(),tbl[field].median(),
                tbl[field].sort_values().iloc[int(nEntry*0.1)],
                tbl[field].sort_values().iloc[int(nEntry*0.9)]))
```

[42]:
```
def CalcAlbertaDefinition():
    cnt = stays.groupby('ClientId').
↪progress_apply(continually_homeless,flagStr='cnt',reqDuration=365)
    epi = epis.groupby('ClientId').
↪progress_apply(TimeWinThresholdTest,posFlag='epi',negFlag='tmp',thresh=4,winSzDays=365)
    return ChooseEarliestTest(cnt,epi)
```

[43]:
```
defGoa = CalcAlbertaDefinition()
```

```
0%|          | 0/18398 [00:00<?, ?it/s]

0%|          | 0/18398 [00:00<?, ?it/s]
```

[44]:
```
def CalcCanadaDefinition():
    sty = stays.groupby('ClientId').
↪progress_apply(TimeWinThresholdTest,posFlag='sty',negFlag='tmp',thresh=180,winSzDays=365)
    epi = stays.groupby('ClientId').
↪progress_apply(TimeWinThresholdTest,posFlag='sty',negFlag='tmp',thresh=546,winSzDays=1095)
    return ChooseEarliestTest(sty,epi)
```

[45]:
```
defGoc = CalcCanadaDefinition()
```

```
0%|          | 0/18398 [00:00<?, ?it/s]

0%|          | 0/18398 [00:00<?, ?it/s]
```

### Group Demography

[46]:
```
demogGoaTmp = demog[defGoa.Flag == 'tmp']
demogGoaFlg = demog[defGoa.Flag != 'tmp']
demogGocTmp = demog[defGoc.Flag == 'tmp']
demogGocFlg = demog[defGoc.Flag != 'tmp']

demogRapidTmp = demog[defRpd.Flag == 'tmp']
demogRapidFlgChr = demog[defRpdChr.Flag == 'chr']
demogRapidFlgEpi = demog[defRpdEpi.Flag == 'epi']
```

### Stats for Clients Identified by a Definition

[47]:
```
fields = [ 'Tenure', 'UsagePct', 'AvgGapLen', 'TotalStays', 'TotalEpisodes' ]
fields = [ 'TotalStays', 'TotalEpisodes', 'Tenure', 'UsagePct' ]

print_stats('Government of Canada Chronic Definition',demogGocFlg,fields)
print_stats('Government of Alberta Chronic Definition',demogGoaFlg,fields)
print_stats('RAPID Chronic Definition',demogRapidFlgChr,fields)
print_stats('RAPID Episodic Definition',demogRapidFlgEpi,fields)
```

```
--- Government of Canada Chronic Definition ---
Clients in cohort: 1549/18398 (8.4%)
Size of 10% of cohort: 154/18398 (0.8%)
TotalStays:
 Avg: 702.7, Med: 522.0, 10thPct: 235.0, 90thPct: 1430.0
TotalEpisodes:
 Avg: 4.4, Med: 3.0, 10thPct: 1.0, 90thPct: 9.0
Tenure:
 Avg: 1564.1, Med: 1439.0, 10thPct: 404.0, 90thPct: 2940.0
UsagePct:
 Avg: 53.0, Med: 48.6, 10thPct: 17.6, 90thPct: 95.8
--- Government of Alberta Chronic Definition ---
Clients in cohort: 2443/18398 (13.3%)
Size of 10% of cohort: 244/18398 (1.3%)
TotalStays:
```

```
 Avg: 438.6, Med: 216.0, 10thPct: 19.0, 90thPct: 1128.0
TotalEpisodes:
 Avg: 7.4, Med: 7.0, 10thPct: 2.0, 90thPct: 14.0
Tenure:
 Avg: 1672.2, Med: 1592.0, 10thPct: 549.0, 90thPct: 2974.0
UsagePct:
 Avg: 28.3, Med: 14.1, 10thPct: 1.7, 90thPct: 84.1
--- RAPID Chronic Definition ---
Clients in cohort: 1815/18398 (9.9%)
Size of 10% of cohort: 181/18398 (1.0%)
TotalStays:
 Avg: 601.0, Med: 411.0, 10thPct: 131.0, 90thPct: 1329.0
TotalEpisodes:
 Avg: 4.0, Med: 3.0, 10thPct: 1.0, 90thPct: 9.0
Tenure:
 Avg: 1399.9, Med: 1241.0, 10thPct: 226.0, 90thPct: 2837.0
UsagePct:
 Avg: 53.1, Med: 49.1, 10thPct: 13.7, 90thPct: 96.2
--- RAPID Episodic Definition ---
Clients in cohort: 4277/18398 (23.2%)
Size of 10% of cohort: 427/18398 (2.3%)
TotalStays:
 Avg: 136.9, Med: 39.0, 10thPct: 4.0, 90thPct: 376.0
TotalEpisodes:
 Avg: 6.3, Med: 5.0, 10thPct: 2.0, 90thPct: 12.0
Tenure:
 Avg: 1308.4, Med: 1149.0, 10thPct: 90.0, 90thPct: 2763.0
UsagePct:
 Avg: 11.2, Med: 5.2, 10thPct: 0.7, 90thPct: 30.4
```

### Impact Comparison

[48]:
```
def CalcDefinitionImpact(definition):
    return CalcImpact(definition,stays)
```

[49]:
```
impactGoa = CalcDefinitionImpact(definition=defGoa)
```

[50]:
```
impactGoc = CalcDefinitionImpact(definition=defGoc)
```

[51]:
```
impactRpd = CalcDefinitionImpact(definition=defRpd)
```

[52]:
```
def print_impact_table(imp,titleStr):
    nClients = 18346
    print("%s & %d (%.1f\%%) & %.1f & %.1f & %.1f & %.1f \\\\ "
          % (titleStr, imp[0], 100.0*imp[0]/nClients, imp[1]/imp[0], imp[2]/
 ↪imp[0], imp[3]/imp[0], imp[4]))
```

[53]:
```
print_impact_table(impactGoa,'Government of Alberta')
print_impact_table(impactGoc,'Government of Canada')
print_impact_table(impactRpd,'RAPID')
```

```
Government of Alberta & 2443 (13.3\%) & 264.8 & 914.2 & 609.8 & 365.0 \\
Government of Canada & 1549 (8.4\%) & 498.4 & 1007.9 & 555.2 & 285.0 \\
RAPID & 5507 (30.0\%) & 194.8 & 874.4 & 402.2 & 98.0 \\
```

**Program Load**

[54]:
```
idRpd = defRpd[defRpd.Flag != 'tmp'].groupby(pd.Grouper(key='Date',freq='M')).
↪Flag.count()
```

[55]:
```
print('Mean ID: %g, Median ID: %g' % ( idRpd.mean(), idRpd.median() ))
```

```
Mean ID: 44.4113, Median ID: 46
```

[ ]:
```

```

**Stats for Transitional Clients not Identified by Definitions**

[56]:
```
fields = [ 'Tenure', 'UsagePct', 'AvgGapLen', 'TotalStays', 'TotalEpisodes' ]
fields = [ 'TotalStays', 'TotalEpisodes', 'Tenure', 'UsagePct' ]
print_stats('RAPID Transitional Clients',demogRapidTmp,fields)
```

```
--- RAPID Transitional Clients ---
Clients in cohort: 12891/18398 (70.1%)
Size of 10% of cohort: 1289/18398 (7.0%)
TotalStays:
 Avg: 14.9, Med: 2.0, 10thPct: 1.0, 90thPct: 40.0
TotalEpisodes:
 Avg: 1.6, Med: 1.0, 10thPct: 1.0, 90thPct: 3.0
Tenure:
 Avg: 382.7, Med: 7.0, 10thPct: 1.0, 90thPct: 1450.0
UsagePct:
 Avg: 59.6, Med: 93.5, 10thPct: 0.4, 90thPct: 100.0
```

**Transitional Client 90th Percentile Investigation**

- These are the group of clients in the program delivery transitional population that are in the top 90th percentile in terms of the length of their shelter tenure.
- There are lots of these folks (more than the chronic population), they interact with shelter over a very long period and are likely flying under the radar of most support programs.

[57]:
```
tenure90thValue = 1289 # Taken from the output provided above.
demog90th = demogRapidTmp[demogRapidTmp.Tenure > tenure90thValue]
```

[58]:
```
fields = [ 'TotalStays', 'TotalEpisodes', 'Tenure', 'UsagePct', 'AvgGapLen' ]
print_stats('Transitional Client 90th Percentile',demog90th,fields)
```

```
--- Transitional Client 90th Percentile ---
Clients in cohort: 1518/18398 (8.3%)
Size of 10% of cohort: 151/18398 (0.8%)
TotalStays:
 Avg: 35.8, Med: 7.5, 10thPct: 2.0, 90thPct: 85.0
TotalEpisodes:
 Avg: 3.3, Med: 3.0, 10thPct: 2.0, 90thPct: 5.0
Tenure:
 Avg: 2073.6, Med: 1935.5, 10thPct: 1399.0, 90thPct: 2999.0
UsagePct:
 Avg: 1.8, Med: 0.4, 10thPct: 0.1, 90thPct: 4.4
AvgGapLen:
 Avg: 635.8, Med: 302.4, 10thPct: 23.0, 90thPct: 1747.0
```

[ ]: