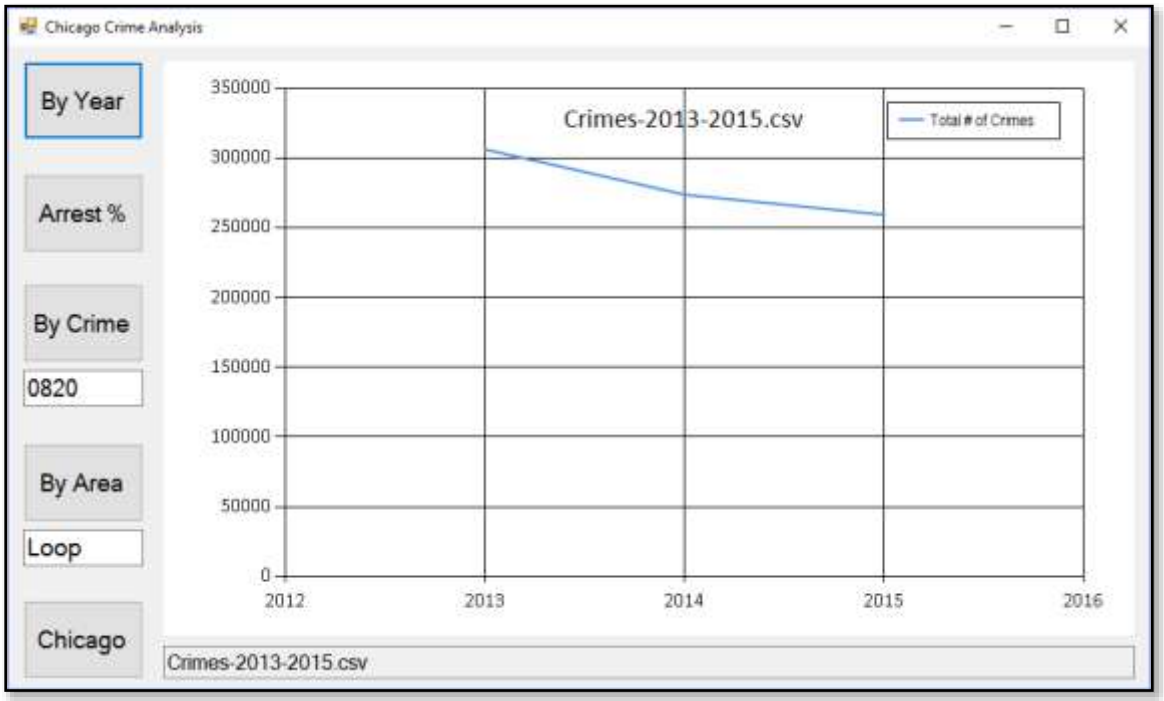
Homework #5

Complete By: Saturday, October 22nd @ 11:59pm
Assignment: completion of Chicago Crime Analysis GUI + F#
Policy: Individual work only, late work *is* accepted
Submission: electronic submission via Blackboard

Chicago Crime Analysis

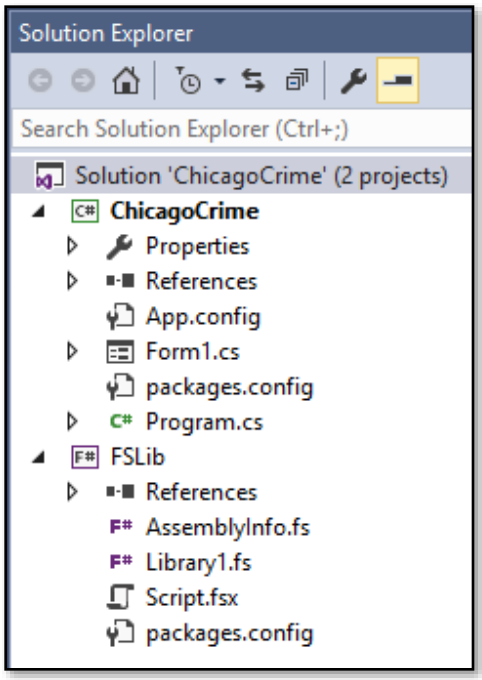
In this homework you are going to develop an F# library to analyze Chicago crime data. To make things more interesting, we'll display the analysis results using an F# charting library and a C# Windows Forms desktop application. Here's a screenshot:



Keep in mind that all computation must be done in the F# library — the C# code is merely to display the chart returned by the F# library. The C# code may not input the data nor perform any of the analysis. [*What type of language is C#? C# is an imperative language, nearly identical to Java.*]

Getting Started

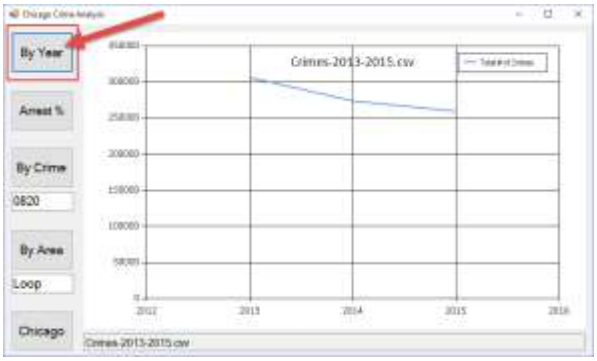
To get started, download the provided Visual Studio solution "ChicagoCrime.zip". If the link doesn't work, look on the course web site under "Homeworks", then "hw5-files". Once downloaded, double-click to open the .zip and then *extract* the folder **ChicagoCrime** to your desktop. Close and delete the .zip file, and work only with the folder you extracted.

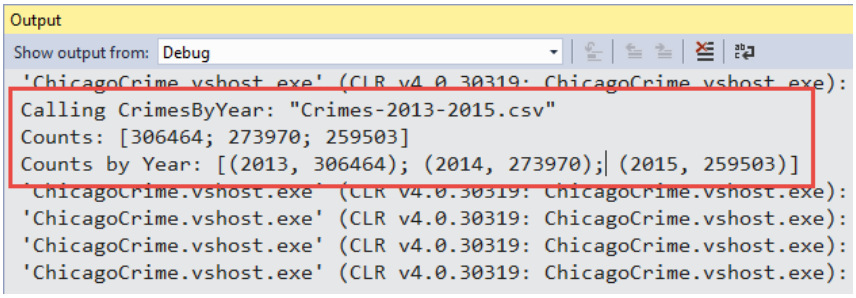


Open the provided solution in Visual Studio by double-clicking on the ChicagoCrime SOLUTION file (.sln). This will open the program in Visual Studio. Take a close look at the Solution Explorer window on the right side: 2 projects will be listed: one project is the C# GUI code, and the other is the F# library code.

Your normal mode of operation will be to work on the F# code, and then build that project using the Build menu, "Build FSLib". Once that builds without error, then you can work on the GUI code to call and test the F# code.

At this point, go ahead and run the program via F5 — the code as provided will run, and the first button "By Year" is completely coded and working for you. When you run, also take a look at the Output window in Visual Studio (if this window is not visible, drop the View menu, and select Output). This window can be used to display console output, which is very handy for debugging purposes:





As given, the F# code outputs the data that is being plotted. If you can read the screenshot, there were a total of 306,464 crimes in 2013, 273,970 in 2014, and 259,503 in 2015.

Your assignment, in a nutshell, is to add the other 4 buttons as shown (and 2 text boxes), and implement the necessary functionality by calling over to functions you have written in the F# library. The purpose of the GUI is to simply handle the button click events, call the F# code, and display the returned chart.

Assignment Details

There are 4 inputs files to the program:

1. Crimes-2013-2015.csv
2. IUCR-code.csv
3. Areas.csv
4. Crimes.csv

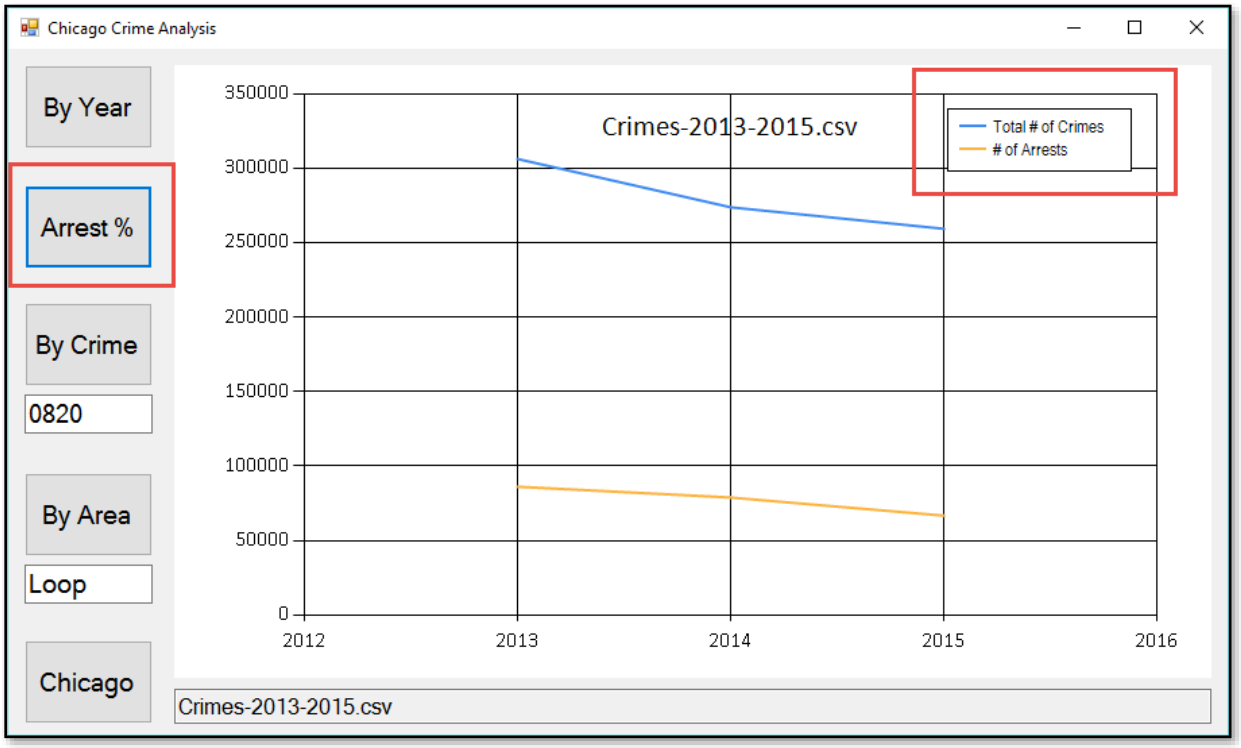
The first 3 files are already installed in the appropriate sub-folder of the provided solution: look in ChicagoCrime\bin\Debug or ChicagoCrime\bin\Release. The 4th file is a larger version of #1 — crime data for the years 2001 .. 2015. Since this file is much larger, it is provided separately on the course web site under "Homeworks", "hw5-files": Crimes.zip.

The program as given already opens and parses the crime data file, either "Crimes-2013-2015.csv" or "Crimes.csv". So you may want to start by reviewing the code that is provided in "Form1.cs" and "Library1.fs". Here's the start of the F# code:

```
module FSAnalysis

#light

open System
open FSharp.Charting
open FSharp.Charting.ChartTypes
open System.Drawing
open System.Windows.Forms

//
// Parse one line of CSV data:
//
//   Date,IUCR,Arrest,Domestic,Beat,District,Ward,Area,Year
//   09/03/2015 11:57:00 PM,0820,true,false,0835,008,18,66,2015
//   ...
//
// Returns back a tuple with most of the information:
//   (date, iucr, arrested, domestic, area, year)
// as string*string*bool*bool*int*int.
//
let private ParseOneCrime (line : string) =
  let elements = line.Split(',')
  let date = elements.[0]
  let iucr = elements.[1]
  let arrested = Convert.ToBoolean(elements.[2])
  let domestic = Convert.ToBoolean(elements.[3])
  let area = Convert.ToInt32(elements.[elements.Length - 2])
  let year = Convert.ToInt32(elements.[elements.Length - 1])
  (date, iucr, arrested, domestic, area, year)
```

Your task is to implement 4 additional buttons, 2 of which require input from a text box. Here are the four additional buttons in terms of functionality and output:

#1) Arrest %: total crimes per year vs. number of arrests per year

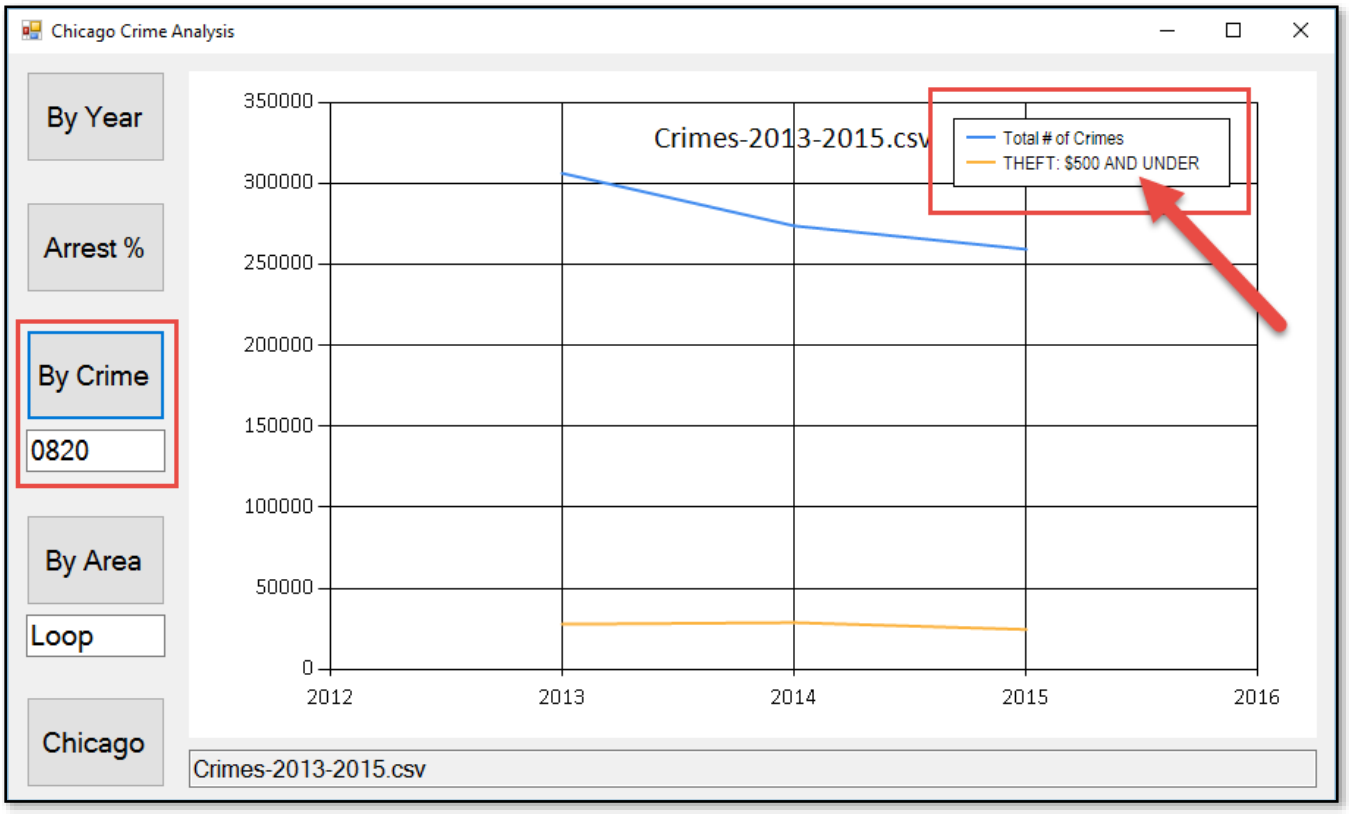


To the right is the console output for the values being plotted. Note that for this button you need to create 2 charts and "combine" them. Here's the code to do that:

```
Calling CrimesVsArrests: "Crimes-2013-2015.csv"
[306464; 273970; 259503]
[(2013, 306464); (2014, 273970); (2015, 259503)]
[86161; 78898; 66711]
[(2013, 86161); (2014, 78898); (2015, 66711)]
```

```
//
// plot:
//
let myChart =
  Chart.Combine([
                  Chart.Line(countsByYear, Name="Total # of Crimes")
                  Chart.Line(arrestsByYear, Name="# of Arrests")
                ])
let myChart2 =
  myChart.WithTitle(filename).WithLegend();
let myChartControl =
  new ChartControl(myChart2, Dock=DockStyle.Fill)
```

#2) By Crime: total crimes per year vs. number of a particular crime per year

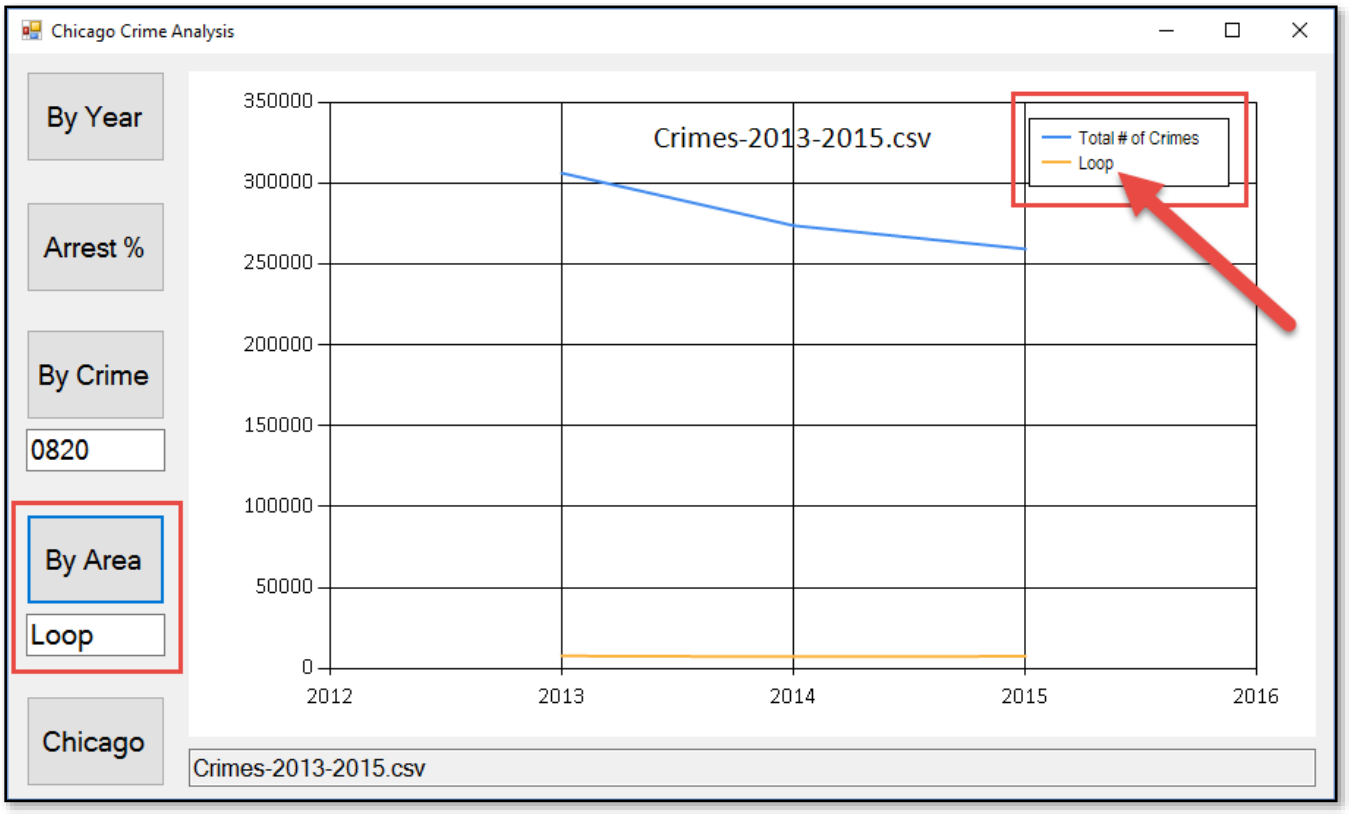


The user enters a string-based code in the text field, in this case "0820". This code is turned into a human-readable string for the legend, based on the code's primary and secondary description. This can be found by loading and searching the provided file "IUCR-codes.csv". If the user enters a code for which a match is not found, plot 0's and the legend should display "unknown crime code". Here's the console output for the crime code "0820":

```
Calling GivenCrimeByYear: "Crimes-2013-2015.csv"
[306464; 273970; 259503]
[(2013, 306464); (2014, 273970); (2015, 259503)]
[27898; 28840; 24582]
[(2013, 27898); (2014, 28840); (2015, 24582)]
```

Again, keep in mind that the F# code is the one that must load and search this additional input file; the C# code may not.

#3) By Area: total crimes per year vs. number of crimes that occurred in a particular area of the city

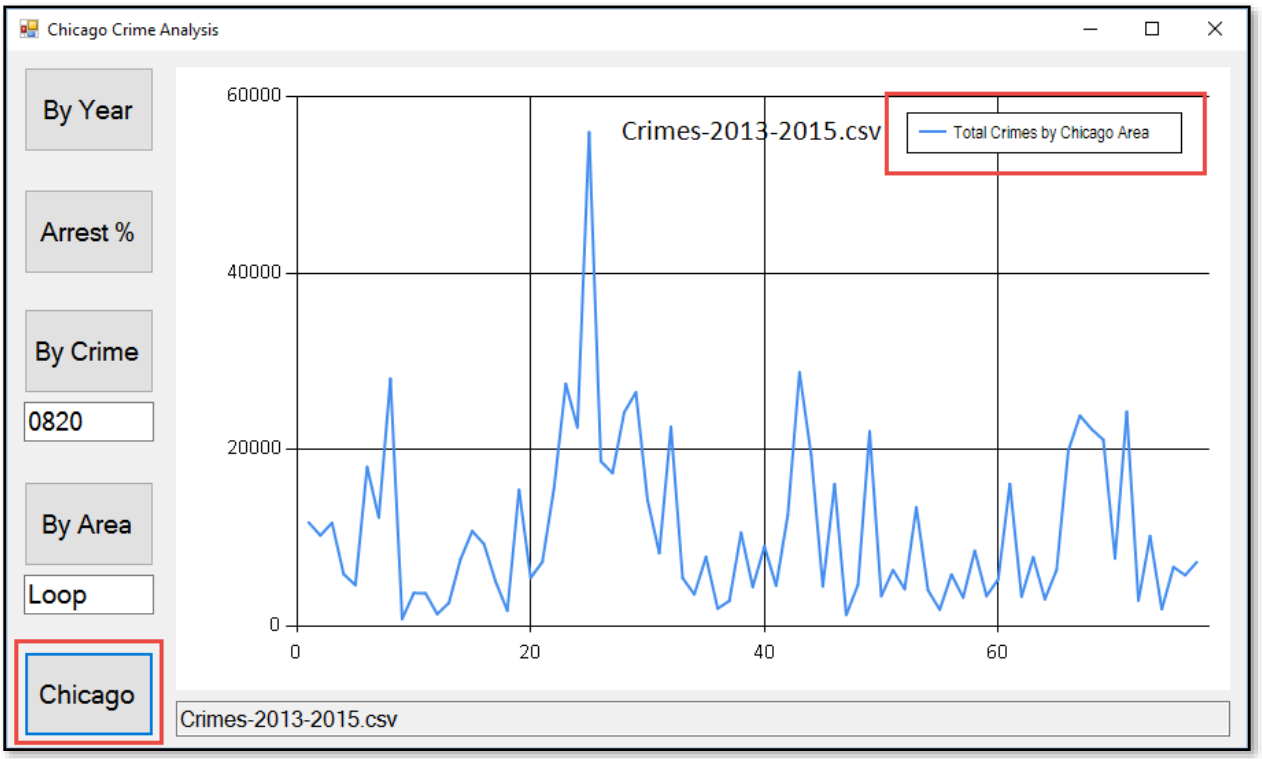


Chicagoland is divided into 70+ areas — the exact areas are defined in the provided input file "Areas.csv". Here's a PDF (Areas.pdf). In this case the user inputs an area such as "Loop", and the program compares overall crime to the crime in this particular area. The crime data is in terms of an area code, e.g. the "Loop" is area 32. So you'll need to load and search the file "Areas.csv" to find the corresponding area code so you can then analyze the crime data. If the user enters an area that is not found, use a code of 0 and plots 0's. Note that strings in F# are case-sensitive by default, so if the user enters "loop" then no match will be fine; this is fine, or you can perform a case-insensitive search, it's up to you.

Here's the correct console output for the "Loop":

```
Calling CrimesByArea: "Crimes-2013-2015.csv"
[306464; 273970; 259503]
[(2013, 306464); (2014, 273970); (2015, 259503)]
area code: 32
[7776; 7273; 7517]
[(2013, 7776); (2014, 7273); (2015, 7517)]
```

#4) Chicago: total crimes for each area of the city



This last button analyzes the data for each area of the city, all 70+ areas. So the X-axis are the areas, and the Y-axis is the total # of crimes that occurred in that area across the entire time period. The spike in the graph is area 25, Austin. Here's the corresponding console output for the plotted data:

```
[(1, 11738); (2, 10263); (3, 11679); (4, 5865); (5, 4637); (6, 18023);
 (7, 12287); (8, 28035); (9, 793); (10, 3740); (11, 3706); (12, 1347);
 (13, 2616); (14, 7578); (15, 10749); (16, 9296); (17, 5028); (18, 1727);
 (19, 15429); (20, 5455); (21, 7268); (22, 15672); (23, 27448); (24, 22511);
 (25, 55959); (26, 18670); (27, 17325); (28, 24208); (29, 26475); (30, 14250);
 (31, 8255); (32, 22566); (33, 5432); (34, 3583); (35, 7830); (36, 1975);
 (37, 2842); (38, 10566); (39, 4423); (40, 9017); (41, 4571); (42, 12571);
 (43, 28753); (44, 19361); (45, 4495); (46, 16070); (47, 1251); (48, 4655);
 (49, 22060); (50, 3398); (51, 6341); (52, 4187); (53, 13462); (54, 4044);
 (55, 1845); (56, 5834); (57, 3238); (58, 8515); (59, 3399); (60, 5279);
 (61, 16093); (62, 3338); (63, 7784); (64, 3038); (65, 6369); (66, 19865);
 (67, 23832); (68, 22293); (69, 21105); (70, 7645); (71, 24283); (72, 2898);
 (73, 10198); (74, 1904); (75, 6678); (76, 5756); (77, 7242)]
```

Efficiency…

Just for fun, run your program against the larger data file, "Crimes.csv" — just change the filename in the text field that's below the chart. This data file denotes all crimes from 2001 to 2015, and is 6x larger. It's okay if a given button takes a minute or so to run. But anything over 2 minutes is a problem and you should investigate.

If time permits, an interesting challenge is trying to get the performance of the F# code more in line with what we would expect from a C# or C++ program, i.e. a few seconds per button…

Electronic Submission

First, add a header comment to the top of your F# source code file, along the lines of:

```
//
// F# library to analyze Chicago crime data
//
// <<YOUR NAME HERE>>
// U. of Illinois, Chicago
// CS341, Fall 2016
// Homework 5
//
```

Exit out of Visual Studio, and find the project folder "ChicagoCrime". First, if you have installed the large "Crimes.csv" data file, please delete that file from both ChicagoCrime\bin\Debug and ChicagoCrime\bin\Release. Then create an archive (.zip) of this entire folder for submission: right-click, Send To, and select "Compressed (zipped) folder". Submit the resulting .zip file on Blackboard (http://uic.blackboard.com/) under the assignment "HW05".

Your program should be readable with proper indentation and naming conventions; commenting is expected where appropriate. You may submit as many times as you want before the due date, but we grade the last version submitted. This implies that if you submit a version before the due date and then another after the due date, we will grade the version submitted after the due date — we will *not* grade both and then give you the better grade. We grade the last one submitted. In general, do not submit after the due date unless you had a non-working program before the due date.

Policy

Late work *is* accepted. You may submit as late as 24 hours after the deadline for a penalty of 25%. After 24 hours, no submissions will be accepted.

All work is to be done individually — group work is not allowed. While I encourage you to talk to your peers and learn from them (e.g. via Piazza), this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot

submit someone else's work (partial or complete) as your own. The University's policy is described here:

http://www.uic.edu/depts/dos/docs/Student%20Disciplinary%20Policy.pdf .

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. Academic dishonesty is unacceptable, and penalties range from failure to expulsion from the university; cases are handled via the official student conduct process described at http://www.uic.edu/depts/dos/studentconductprocess.shtml .