# Analysing collections and datasets in F#

## Lab Overview

In this lab session, you will perform a number of analytical queries over a dataset sourced from a flat file.

## Prerequisites

* Visual Studio 2015

## Time Estimate

* 30 minutes

## Table of Contents

[Analysing collections and datasets in F# 1](#_Toc434421300)

[Lab Overview 1](#_Toc434421301)

[Prerequisites 1](#_Toc434421302)

[Time Estimate 1](#_Toc434421303)

[Table of Contents 1](#_Toc434421304)

[Exercise 1: Loading in data from a flat file 2](#_Toc434421305)

[Exercise 2: Finding a specific result 2](#_Toc434421306)

[Exercise 3: Sorting results 2](#_Toc434421307)

[Exercise 4: Composing a complex pipeline 3](#_Toc434421308)

[Bonus Exercise: Using Collect within a pipeline. 3](#_Toc434421309)

## Exercise 1: Loading in data from a flat file

In this exercise, you will load and parse some data from a flat file into an in-memory dataset that you can then perform queries against. The data set contains English Soccer results for an entire year.

1. Open the WorkingWithCollections.fsx file. This file has a set of stubs that you must complete through the rest of this lab.
2. You will notice that the script loads in the contents of the DataAccess.fsx script. You can explore this script to see what it does – essentially it contains two assets: -
   1. A Result record type, which represents a single row in the database.
   2. A parseRow function, which has a signature string -> Result option.
3. Complete the right hand side of the assignment to let data = . After you complete every step below, evaluate the code to ensure it does what you expect, rather than trying to complete the entire pipeline all at once.
4. You will need to use System.IO.File.ReadLines or similar to read every line (which returns a string seq), and then pipe each row into the parseRow function. As this function returns an Option, you’ll need to use choose at some point to remove any None rows and convert the remainder from Result option to just Result.
5. You may wish to skip out the first row of the dataset, as this contains a header row. There is a function to do that in all the collection modules.
6. Finally, you will need to convert the result of your parsing to a List using an appropriate combinator.

Notice also that the parseRow has a try / with block for exception handling. You can read up more on exceptional handling in F# here: https://msdn.microsoft.com/en-us/library/dd233223.aspx

The line starting with “Environment.CurrentDirectory” is needed when working with standalone scripts. The default location when scripts run from is the Temp folder, so either absolute paths must be used for e.g. file access, or changing the current directory using the \_\_SOURCE\_DIRECTORY\_\_ macro.

## Exercise 2: Finding a specific result

We’ll now perform a simple query to retrieve a single row based on two predicates using the functionality contained in the List module.

1. Find the result of the match between the two teams, “Tottenham” and “Arsenal”. Use the Home Team and Away Team properties. You can either use a single predicate using e.g. the find function, or chain two filters together and then return the head of the list (or pattern match).

Observe that properties (and modules / functions etc.) in F# can contain spaces in them as long as you start and finish the symbol name with ``

## Exercise 3: Sorting results

Next we’ll find the record of the game which had the most goals.

1. This is done by identifying the result that had the most "Full Time Home Goals and Full Time Away Goals.
2. You can sort results using the sortBy… functions. One of them takes in a projection which is used to perform the sort.

## Exercise 4: Composing a complex pipeline

In this exercise, we’ll compose a slightly more complicated pipeline that will use a standalone function as well as inline lambdas. We shall identify the team that has won away from their home ground the most times.

1. Create a new function, isAwayWin which will take in a single result and determine whether the result was an away win or not i.e. Full Time Away Goals is greater than Full Time Home Goals.
2. Create a pipeline on data that starts by filtering only results that are away wins.
3. Group and count the results based on the name of the away team. This will give you a result of type (string \* int) list. The string will equal the name of the team, and the int will equal the number of occurrence that existed in the dataset.
4. Sort the results in descending order by the number of occurrences per team (i.e. the int in the tuple).

## Bonus Exercise: Using Collect within a pipeline.

The collect function (<https://msdn.microsoft.com/en-us/library/ee370406.aspx>) allows us to generate multiple results from a single result. In LINQ, it is known as SelectMany. We’ll use it in this exercise to derive details of both home and away teams into a single collection. We want to identify the team that has the worst disciplinary record in the league.

In soccer, teams earn yellow or red cards when fouling opponents. A yellow card can be considered a “warning”, whilst a red card earns the player an instant dismissal from the field of play for the remainder of the game.

1. Apply the collect function over the dataset. In the collection function, you should return back a list containing two rows, one of each of the teams involved. Each row should contain the team name, number of yellow cards and number of red cards.
2. The output so far should be a list of string \* int \* int i.e. Team, Yellow Cards, Red Cards (one row per match). You can confirm that your collection function works as expected as you should now have twice as many rows in your output as you started with (380 -> 760).
3. Group the data up by team, so that the data is now a string \* (string \* int \* int list) list e.g.

[ “TeamA”, [ “TeamA”, 0, 2; “TeamA”, 1, 0 ]

“TeamB”, [ “TeamB”, 1, 1; “TeamB”, 0, 0 ] ]

1. Map each row so that for you return back out the team name and secondly, calculate the total number of yellow and red cards for that team using the sumBy function. To represent the fact that red cards are a more serious offence than yellow cards, times the number of red cards by 3. This will give you back a new list of type (string \* int) which is the team name \* number of disciplinary points.
2. Sort the result in descending order by the number of disciplinary points.