# Coffee & Coding: Updating Shiny dashboards from Excel via MongoDB (the Good, the Bad and the Shiny)

Mark McPherson

## Introduction

- Current process used to update dashboards in MI team
- A more efficient method
- Step 1: Excel -> MongoDB
- Step 1: Excel macro
- Step 1: MongoDB setup
- Step 2: MongoDB -> Shiny
- Step 2: Shiny code
- App demonstration
- Potential problems and improvements
- Summary
- Questions?

## Current process used to update dashboards in MI team

- Data provider downloads data
- They email it or make it available in a shared folder
- MI team adds the new data to an Excel file
- They run a data prep script on it
- They check the dashboard for any errors, fixing as necessary
- The dashboard is redeployed with the new data
- Again the dashboard is checked for any errors
- They let the data provider know it is now up to date

## A more efficient method

- Data provider downloads data and saves in macro file
- They run the macro - the data is sent to MongoDB
- They visit the dashboard online and check it is as expected
- If it is not they let MI know so it can be investigated and fixed
- Note that if something is not right, it is almost definitely due to a change in the data - I will cover how this can be handled in a manner that at least makes identifying the cause of the error very easy towards the end of the session

All the code including excel macro files and MongoDB Realm app is at https://github.com/MarkMc1089/FIC. Note that no real data or credentials are in the repo of course! If you want to try it out just get in touch and I will supply the credentials :)

## Step 1: Excel -> MongoDB

The data needs to be pasted into a macro enabled Excel file. The macro sends active sheet to a MongoDB database with name matching the sheet name before first underscore and collection matching the full sheet

name.

To run the macro, while on the sheet you want to send, just use `CTRL+W` key combination.

## Step 1: MongoDB setup

MongoDB provides a way of creating endpoints in the cloud (known as Mongo Realm). Since I first set this up, the method I used has been deprecated but it still works for now. The new process is not that different anyway.

An endpoint is associated with a JavaScript function. The function is used to tell Mongo how to handle the data, e.g. which database and collection to add it to. Since it is JavaScript, it provides an extremely versatile way of processing the data prior to saving it in the database!

An app set up in Realm can be downloaded and saved in source control along with the dashboard it is linked to.

## Step 2: MongoDB -> Shiny

Once the data is in MongoDB, the next step is to be able to access this from Shiny. For this there are at least 3 packages. I chose to use the package `mongolite` as it seems the most recently active and well maintained.

A current issue is that on the VPN access to MongoDB is blocked. This means that when run locally, the app will always fall back to using the most recent data. Luckily, there are no problems with access when the app is deployed to `shiny.io`!

## Step 2: Shiny code

In the Shiny app, there are 2 helper functions: `get_mongo_collection` and `load_data`. The first will get all records from a specified database and collection (and also save them to disk, i.e. the Shiny server). The second one attempts to use the first to get the data. If there is any error, it will instead fall back on using the most recently saved data. This prevents a complete catastrophe if the database has an outage for example.

## Potential problems and improvements

- Each time the data is updated, all data must be sent. This is not particularly slow (much faster than redeploying the whole app!), but would be better to allow for new data to be inserted rather than entirely replace existing data in Mongo
- If the data changes in any way, e.g. name or existence of columns, or a change in the possible values, the app is likely to show at least some errors, if not entirely crashing; the best solution here is to include data validation in the endpoint JS function and only allow an update to proceed if everything matches what is expected. With an informative error message given to the data provider, it will allow the change to be pinpointed for faster resolution
- As mentioned, there are some access issues while on VPN; these should be resolved to make development easier

## Potential problems and improvements

- By taking data from the cloud, this opens the possibility of a fully live dashboard, something which I have heard could be desirable for some use cases in the organisation
- Currently the app always reads all the data from Mongo when run and hence also runs the data prep which can be time consuming for large apps and/or datasets; ideally there would be a method of flagging the last update, so that the app only reads data when new data is available and uses the most recent saved data when not

## Summary

We have went from this...

- Data provider downloads data and filters it to most recent month *~60s*
- ~~They email it or make it available in a shared folder~~ *~30s*
- ~~MI team adds the new data to an Excel file~~ *~60s*
- ~~They run a data prep script on it~~ *~60s*
- ~~They check the dashboard for any errors, fixing as necessary~~ *~180s + fix time if needed*
- ~~The dashboard is redeployed with the new data~~ *~450s*
- Again the dashboard is checked for any errors *~180s*
- ~~They let the data provider know it is now up to date~~ *~30s*
- Total time (when no errors) = **17.5 mins**

## Summary

...to this

- Data provider downloads data and saves in macro file *~60s*
- They run the macro - the data is sent to MongoDB *~30s*
- They visit the dashboard online and check it is as expected *~180s*
- If it is not they let MI know so it can be investigated and fixed *+ fix time*
- Total time (when no errors) = **4.5 mins**, a time saving of **75%**

## Questions?