

PROJECT 2

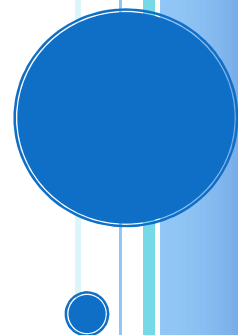
<https://workload-mis.herokuapp.com/>

This is an interactive data visualization website built using crossfilter.js, dc.js, d3.js and queue.js libraries. It includes a 'tutorial-like' feature using the intro.js library. Data is stored in a ClearDB SQL database and is queried in a variety of ways in order to present meaningful MIS views. Python is the language used for accessing the database. The site is hosted on Heroku.com. It was built using the Flask Framework, a bootstrap grid layout and is a site that sizes appropriately on tablet or PC.

Maria Hynes

30th August 2016

<https://git.heroku.com/workload-mis.git>



PROJECT 2

Contents

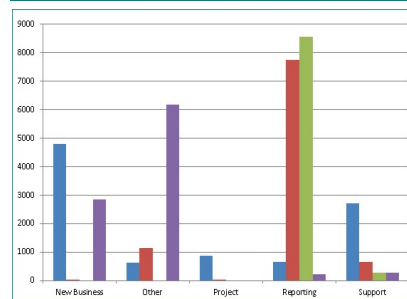
- 1) THE BRIEF 2
- 2) MEETING THE BRIEF 3
 - Overview.....3
 - Home Page.....4
 - Pre-Filtered Pages.....6
 - Chart Page.....8
 - Tutorial feature10
- 3) THE DATA 11
 - Overview of the Data11
 - Preparing the Data11
 - Database11
- 4) DEPLOYING 12
 - Deploying the Database12
 - Deploying the Code13
- 5) FILE STRUCTURE..... 15
- 6) CODE REFERENCES..... 16

1) THE BRIEF

In my company there are two managers in separate geographical locations who each manage a team of Analysts. Every month the data collected from these teams is presented in two separate reports, prepared for Senior Management. The data details the time each person on each team has spent on various projects and functions. To date, this data has been analysed and presented monthly using Excel charts and pivot tables. The Excel report files are stored on a Sharepoint server. The use of Excel requires managers to open a file in order to view their team's data and they then have limited choice on how their data is filtered. Currently, unless the managers directly edit the pivot tables and create their own filters, there is no easy way for them to change views. In addition, should they make changes and then save the file, their changes are saved for all other users. My brief was to create a suite of management reports that:

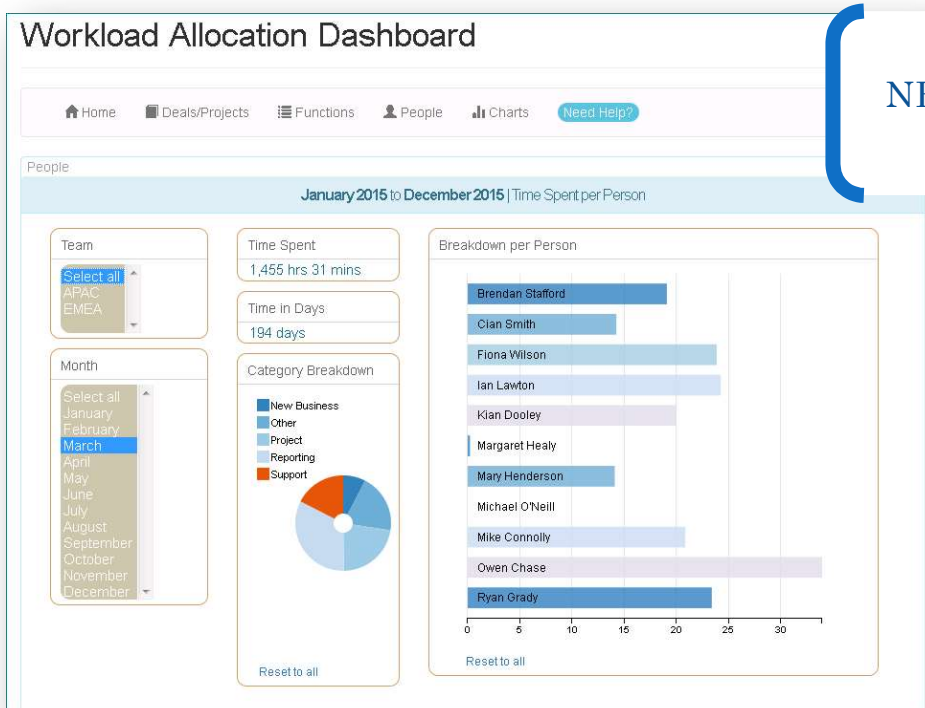
- Would be accessible through a browser rather than a file
- Would combine data from both teams
- Would allow managers to slice and dice the data quickly and easily without affecting other users

OLD (Excel)

[illegible]

Workload Allocation Dashboard

NEW (Website)

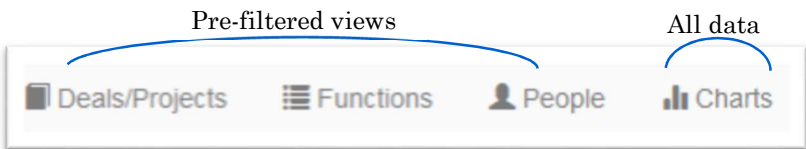
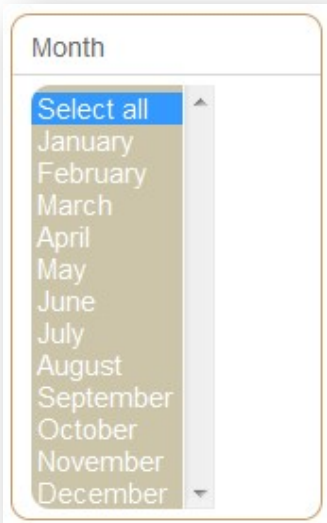
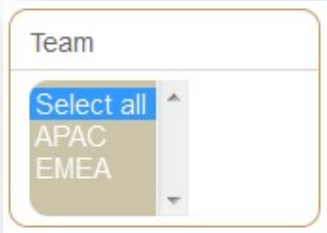


2) MEETING THE BRIEF

Overview

The **Workload Allocation Dashboard** site was created to meet this brief. In this project example, all data from 2015 for both teams is available. On every webpage on the dashboard site, the team's manager can select their own team from a drop-down list or, more conveniently for Senior Management, data from both teams combined can be contrasted and compared.

With the exception of the Home Page, each the site page contains data from all available months and each has a focus on a specific data point. As each month passes and new records become available, the queries populating the drop-down lists will include all available months and therefore the Month drop-down lists will automatically grow. In this example a full year of data is available and so the user can see and select any month of that year:



For convenience three of the site pages show 'pre-filtered' views. Depending on what information may be required, these pages provide a quick focus on a different aspect of the data. There is also a 'Chart' page allowing managers to filter all data.

As Management will always need to see a high-level breakdown of the main work categories, the Home Page provides this summary. It takes only the *latest* monthly data and displays:

- the top five projects per team (based on amount of time spent)

December 2015 | Top 5 Deals per Team

- a breakdown of time spent per Category

December 2015 | Time Spent per Category

Home Page

The summary data displayed on the Home page requires two separate queries, one to display Deal/Project statistics and another to display the Category breakdown. To make this possible, I updated the code so that the queue function uses two API calls instead of oneⁱ. As a result I was able to use a different source for each pie chart.

The queries on this Home page have been set up to retrieve the *latest* month's data. In this project example, the latest month is December 2015. This page will automatically update to the next month each time another month of data has been uploaded. A function called `latest_month()` is used for this purposeⁱⁱ:



For the Category breakdown pie-chart (above), I reviewed the dc.js code and used the legend parameter to include the legend on the left side of the chartⁱⁱⁱ. I also updated my html to include the 'reset' functionality^{iv} as described in the DC.JS GETTING STARTED AND HOW-TO GUIDE^v and provided the user with the link 'Reset to select all Categories' so that there is a quick and easy way to reset the chart after selecting a section(s), without having to reload the entire page.



For the 'Top 5 Deals per Team' chart (above) I needed to create a query that would return 10 records of the top five deals per team. This was made possible by the use of a UNION query. I added new UNION query function^{vi} to the mysql.py python class. This function expects two query strings as parameters and concatenates them into a UNION query. To get the query strings, I added a new optional kwarg argument to the select function which, if included when calling the select function, will return the query string instead of the query results^{vii}. The output is the results combined into one recordset.

The members of the Teams are required to enter their times in minutes, but for reporting purposes, to use only the sum of minutes is meaningless when we are viewing large numbers. I edited the code to convert the sum of minutes into **hours** and **minutes** and **days**^{viii}. In order to then use the words 'hrs', 'mins' and 'days' beside the numbers in the display, so that it would make sense, I worked on the html code to find a way to display the numbers alongside text as though in a sentence^{ix}.

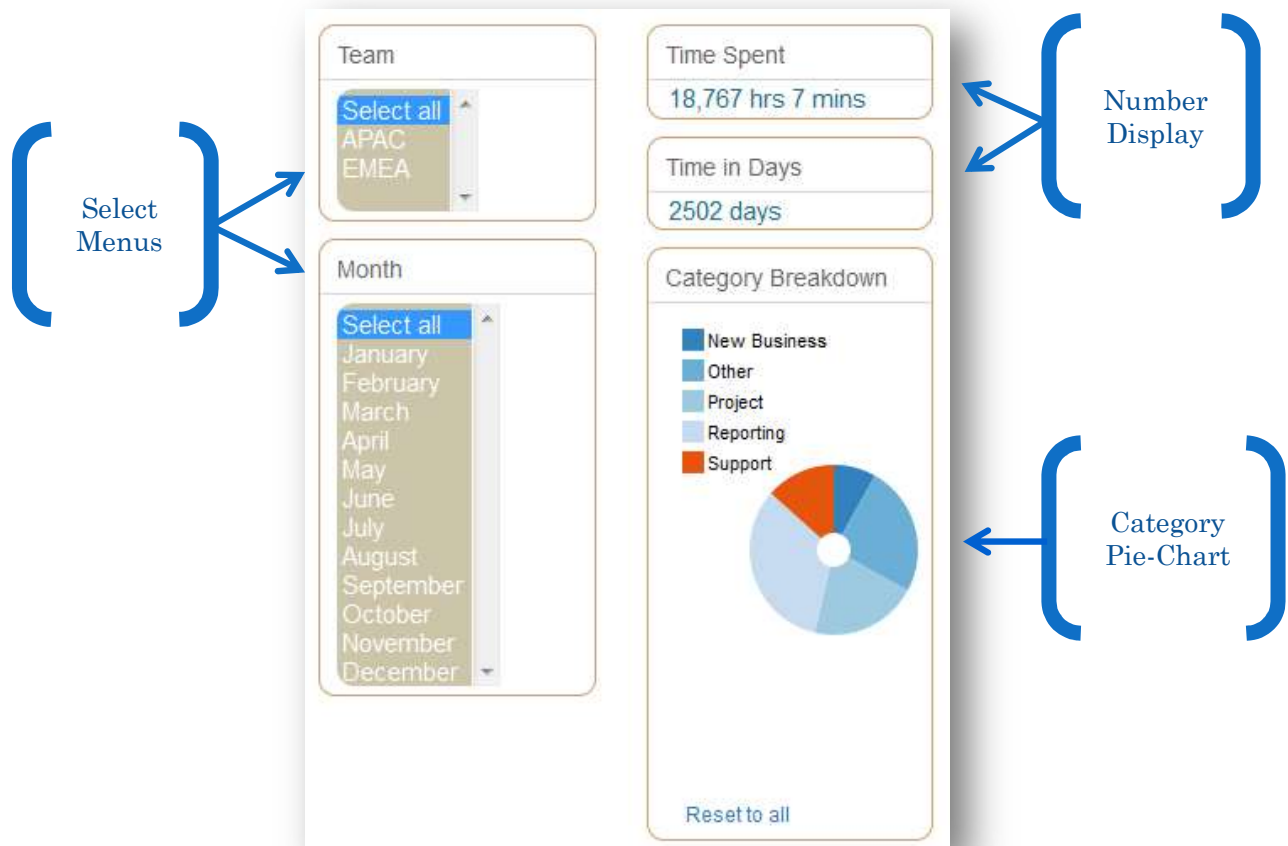


To do this, both text and number had to be the same size, so I also edited the css code and created a custom class for the font size of the number display to match the size of the text beside it. The default transitioning behaviour on the Number Display in the dc.js library is more visually effective when the number display has a large font size because the transition duration is set to 250. When the font is smaller, like here, the transition effect becomes an annoying flicker. To counteract this, I updated the transition duration to 10^x. This effectively turns off the transitioning effect on the number so that it appears to change/update immediately when the chart is filtered.

Pre-Filtered Pages

The pre-filtered pages are the **Deals/Projects** page, the **Functions** page and the **People** page. Although each has a specific focus, for ease of viewing they all contain the same left side bar functionality. It is the only the large main chart on the right that is different.

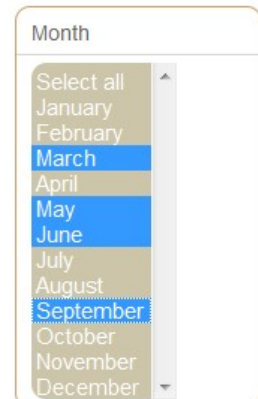
On the left of each page, the user can select from the Team and Month menus. Changing these will update the Number Displays to show corresponding total time spent in Hours and Minutes and total time spent in Days. The small pie-chart of the Categories will also be updated.



For these pages, where the pie-chart is smaller than the pie-chart on the Home Page, I turned off the feature which displays the name on each slice of the pie-chart^{xi}. Instead the user can see (and click) the legend items corresponding to each pie section.

I also updated the select list in the dc.js library to turn on the multi-select option^{xii}. This enables the user to select more than one month at the same time if required:

The main chart on each page is set to display a line-chart containing a breakdown of one of the three main data points (see next page).



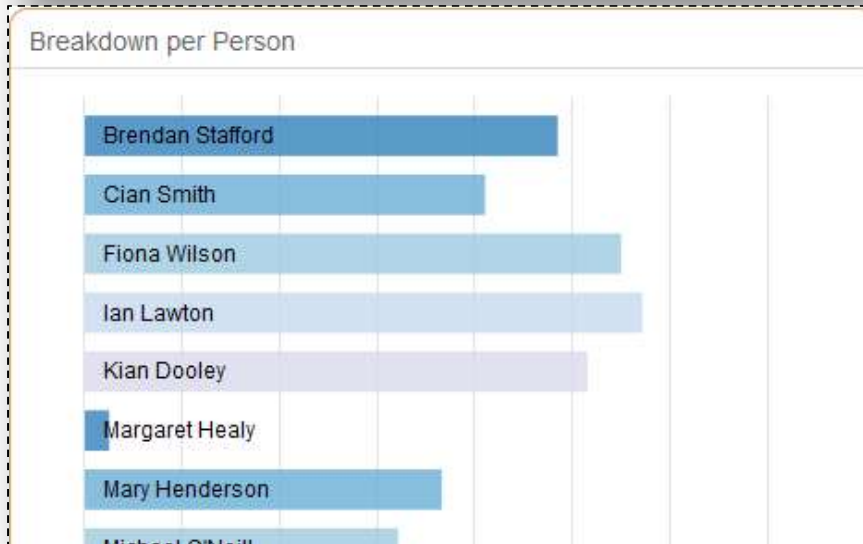
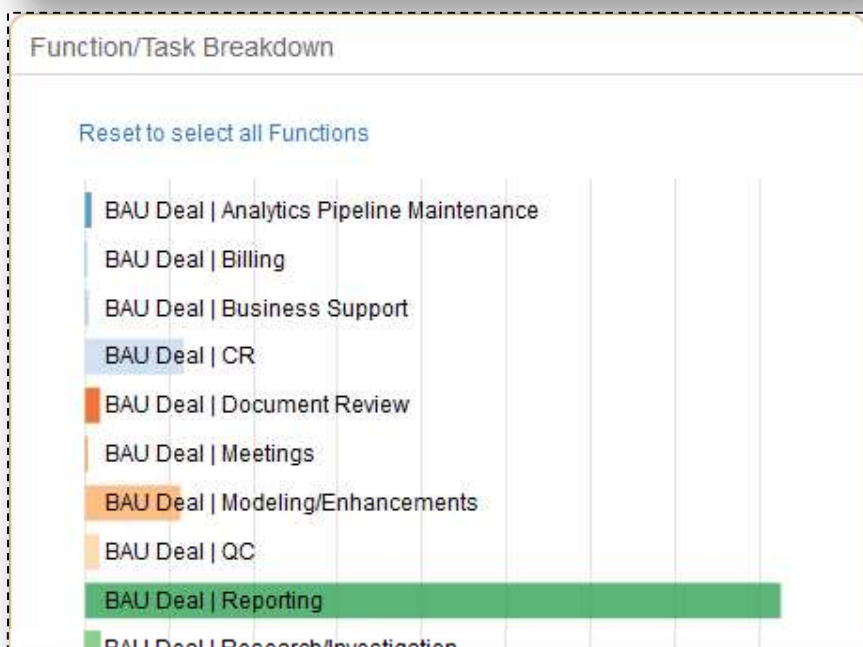
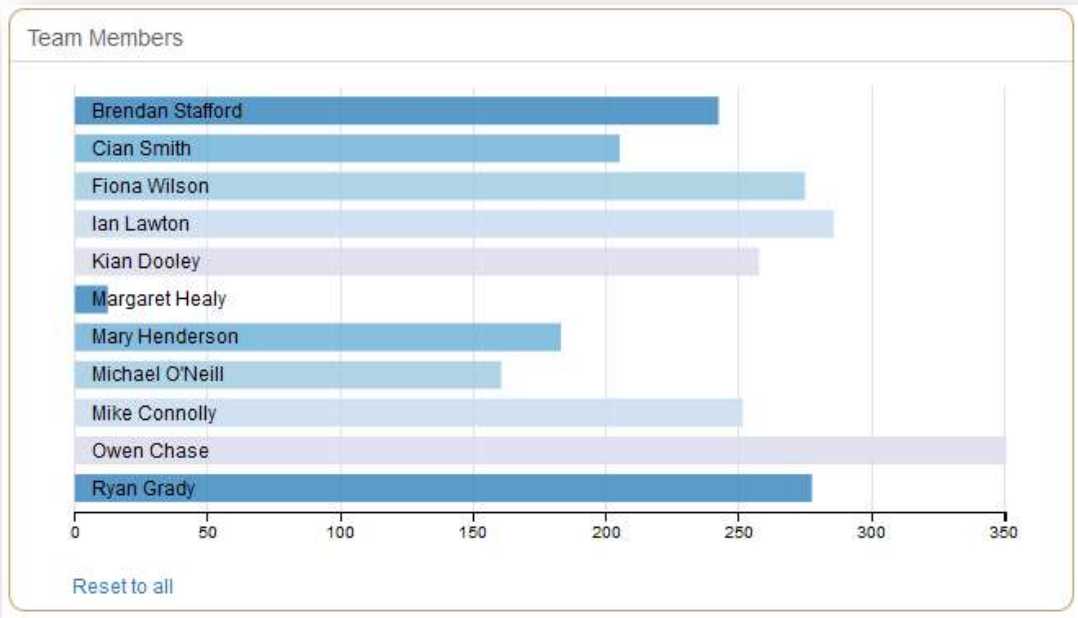
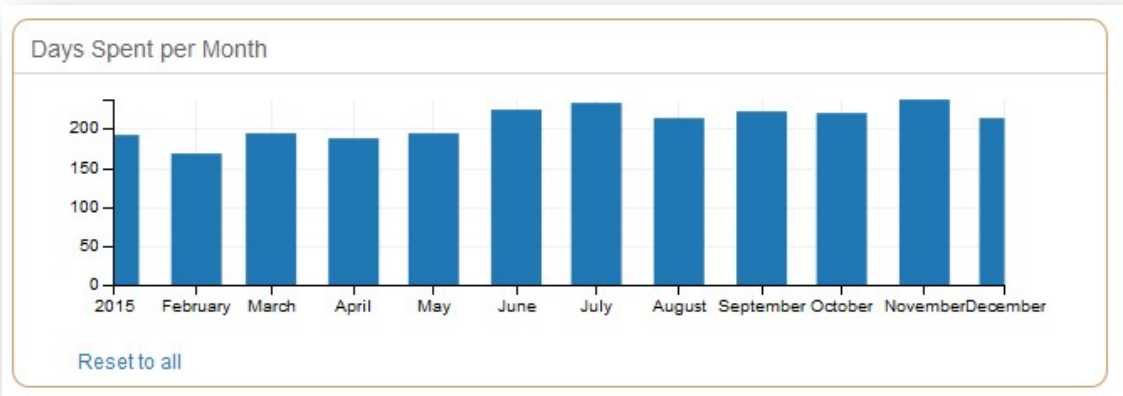


Chart Page

The Chart Page allows the user free-reign to pick and choose what way to view the data by providing additional drop-down select menus, essentially offering managers to ability to view all data at the same time. I included the usual Team and Month menus and then added Function and Deal/Project menu lists. The team members are on display through a line-chart:



A bar-chart at the top of the page shows the spread of days spent per month:



The Category pie-chart is also presented, along with the time-spent number displays. 'Items Logged' is an additional piece of information which shows the count of records entered by the teams. This equates to the number of records in the database (one record per item entered):

Items Logged

16,522 items

To make this page more appealing to navigate I needed to be able to adjust the height of the select menus. This option is not available in the current dc.js library and as a result all the menus defaulted to the same size regardless of the number of items in the menu list. I edited the dc.js code to include a new 'size' parameter^{xiii}. I then used this parameter in my javascript making each select menu a different height^{xiv}:

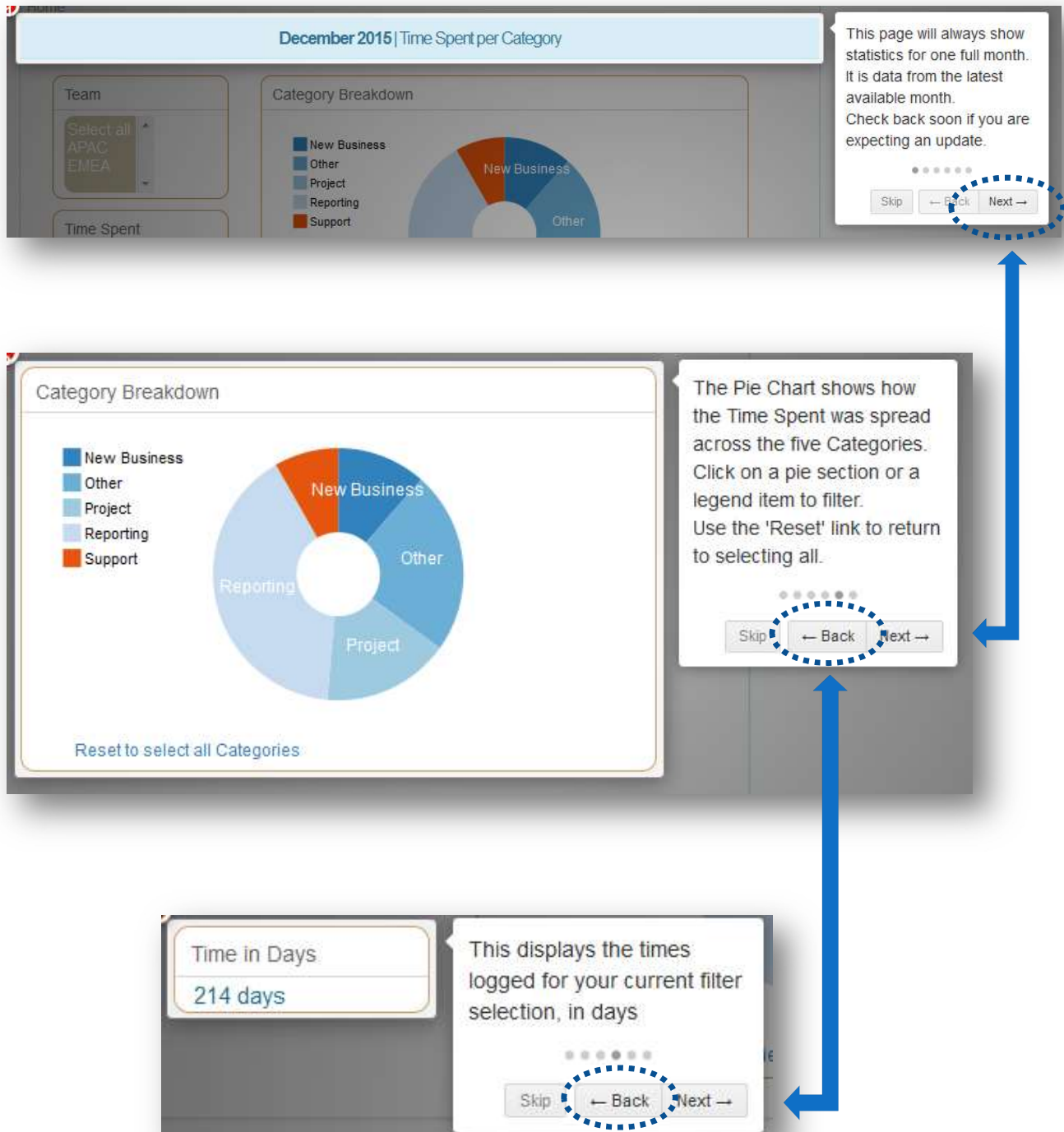


Tutorial feature

I used the intro.js library to create the tutorial functionality triggered by clicking the 'Need Help?' button.

Need Help?

This button is available on every page and when clicked will guide the user to through each section of the page to give them an understanding of what every menu, chart and number display signifies.



3) THE DATA

Overview of the Data

Team Members log the time they have spent each day on each of their tasks. They must enter the following for each item:

- 1) The Deal/Project name (selected from a defined list 'Model Name')
- 2) The function they performed (selected from a defined list)
- 3) The time (in minutes) that they spent on the item
- 4) The date they performed the function

The system then saves each record with this additional information:

- 5) The user's name
- 6) The users' team name
- 7) The Category (based on the function selected in 2 above)
- 8) The date the item was logged

Preparing the Data

The data chosen for this project was the full year of 2015. All 16,522 records for this year were exported from the company's online database and all were desensitized so that no confidential data is used for the purpose of this project submission. Every employee and project/deal name has been altered. Before importing to the SQL database, I first expanded on the date field and extracted the Month Name, Year and Month number into new fields for ease of querying later. These are the final data fields used in this **Workload Allocation Dashboard**:

Model Name	Date	Name	Function	TimeSpent	Team	Category	Month	Year	MonthName	MonthNumber
------------	------	------	----------	-----------	------	----------	-------	------	-----------	-------------

Database

During testing and development phase, data was stored on a local MySQL database. Importing these records to this local development version was carried out by importing the csv file of records through the **Table Data Import Wizard** function on MySQL Workbench.

When online, the data is stored on ClearDB, which is an add-on SQL database available through Heroku.com.

4) DEPLOYING

Deploying the Database

Uploading the records to the online ClearDB was difficult in that both the import functionality on MySQL Workbench AND the MySQL Dump facility gave me upload error messages. The two methods I tried were:

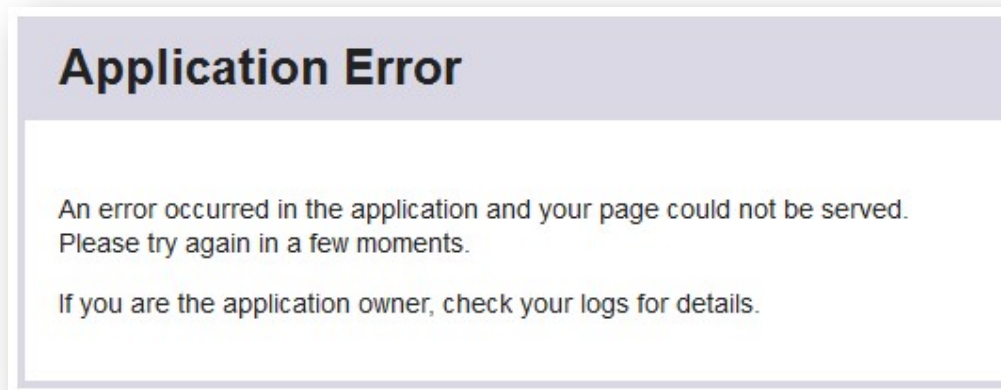
Upload Method	Issue
The Table Data Import Wizard	This kept timing out only allowing only a few hundred records per upload.
The SQL Dump facility	This did not allow me to upload because of a “SELECT command denied to user...” error.

In the end I decided I had no choice but to use the Table Data Import Wizard and to avoid the timeout errors, to split the large csv file into many smaller files. Even at that, some of the files still gave timeout upload errors and because of this, I had to first upload each file into a temporary table, so that I kept the main table ‘clean’ and only imported into the main table after each *successful* temporary table upload. I ran an INSERT query to copy the records from the temporary table into the main table each time, ensuring to run record count queries to check that the expected number of records uploaded and again to check that the main table had the expected number of records after each INSERT query. Before each upload, I deleted the temporary table and recreated it again with the query for the next upload file. It took over two hours to import all 16,000 records in this way.

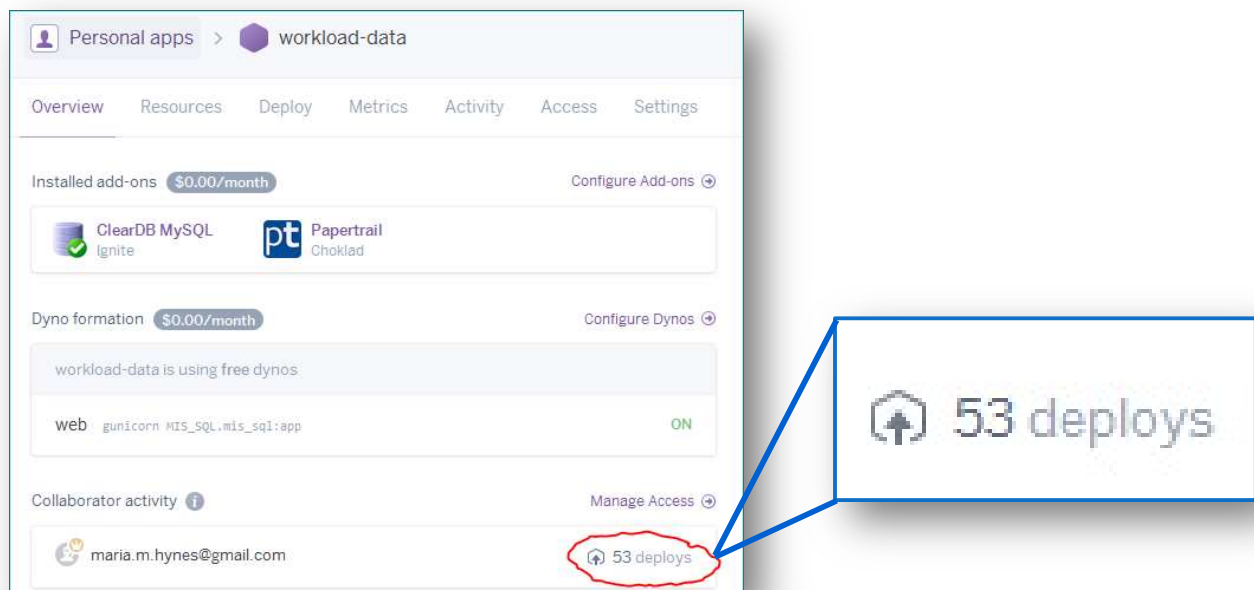
Even though I had finally successfully uploaded all records, I still needed to know why it wasn’t possible to upload a large number of records in bulk. After searching in many online forums to find out if others had the same issue, I finally raised a support ticket on ClearDB explaining the upload problems I was having. To my relief, I received a reply from ClearDB telling me that the version of MySQL Workbench (v6.3.6) I was using had a bug that caused this issue. I was advised to download a newer version or a version prior to 6.3.4.

Deploying the Code

The code deployment was unfortunately another headache! I created an Heroku app called 'workload-data' which unfortunately never opened successfully in the browser. I kept getting an 'Application Error' screen:

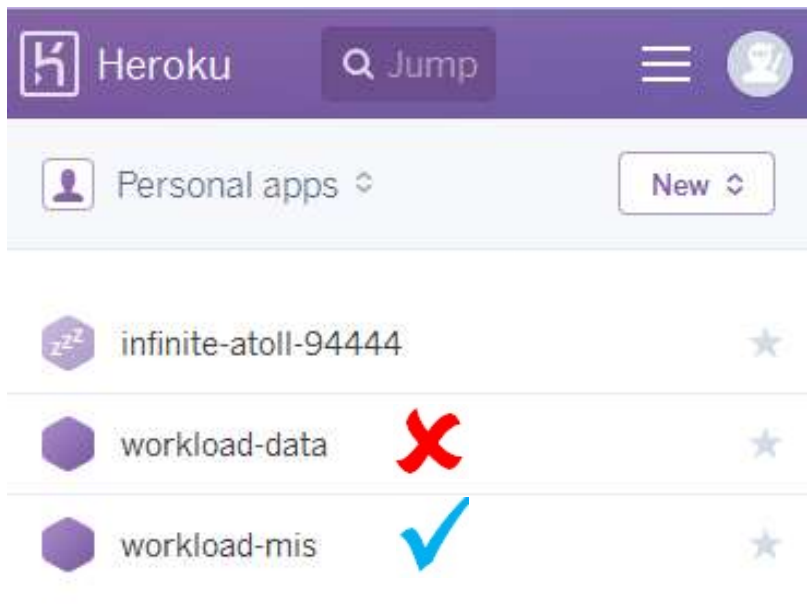


The reason for this error was difficult to ascertain because the code and the database ran perfectly when on my local machine and it even ran perfectly even when connecting to the ClearDB and running locally, but always gave an error when I tried to run it online from Heroku.

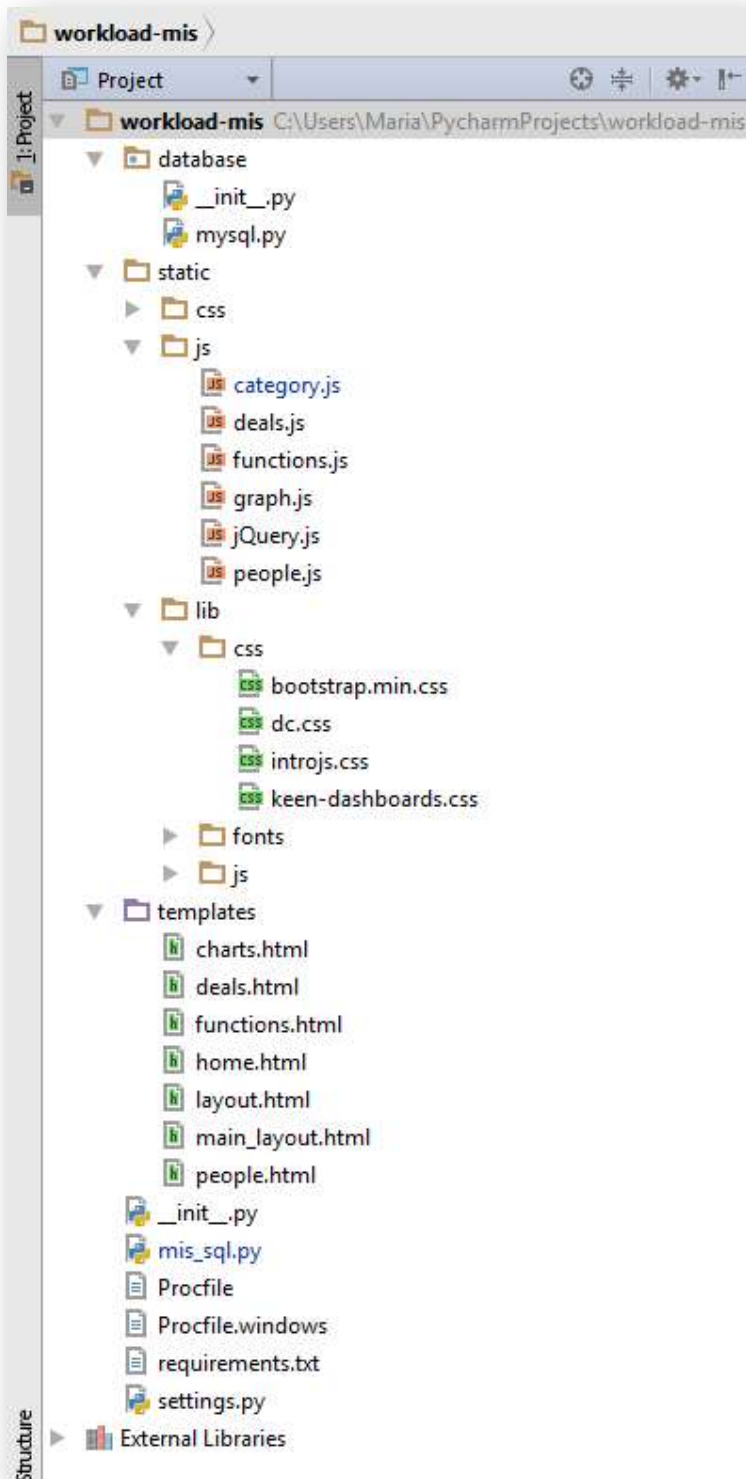


53 separate deploys later, after trying many suggestions I found on online forums, I was advised (by another Code Institute Student – thank you Alun Price!) to try creating a brand new app on Heroku and following the deployment steps again. On this advice, I created the 'Workload-MIS' app and this deployed correctly the very first time, following all the same deployment instructions as before. I can only guess

that the first app issue was to do with the Dyno Workers on Heroku. But I will probably never know for sure.



5) FILE STRUCTURE



6) CODE REFERENCES

i Section of code from category.js file:

```
queue()
    .defer(d3.json, "/MIS/categoryMonth")
    .defer(d3.json, "/MIS/dealDataMonth")
    .await(makeGraphs);

function makeGraphs(error, categoryDataJson, dealDataJson) {

    //Clean dataJson data
    var teamData = categoryDataJson;
    var dealData = dealDataJson;

    //Create two Crossfilter instances
    var ndx = crossfilter(teamData);
    var ndx2 = crossfilter(dealData);
```

ii Section of code from mis_sql.py:

```
def latest_month():
    # this will search the table for the LATEST month data available and return the month name
    the_latest_month_gry = db.select(data_table(), ['month_name', 'the_year'], named_tuples=True,
                                     ORDERBY='the_year DESC, month_number DESC', LIMIT='1')

    for row in the_latest_month_gry:
        the_latest_month = row.month_name

    return the_latest_month
```

iii Section of code from category.js:

```
categoryChart
    .height(220)
    .radius(90)
    .innerRadius(30)
    .legend(dc.legend().x(20).y(20).itemHeight(13).gap(5))
    .minAngleForLabel(0.6)
    .transitionDuration(1500)
    .turnOnControls(true)
    .dimension(categoryDim)
    .renderLabel(true)
    .group(totalTimeSpentByCategory);
```

iv Section of code from home.html:

```
<a class='reset'
href='javascript:categoryChart.filterAll();dc.redrawAll();' style='...' >Reset to select all Categories</a>
```

v <http://dc-js.github.io/dc.js/docs/stock.html>

vi Section of code showing **union** function from mysql.py:

```
def union(self, qry_str1, qry_str2, named_tuples = False, union_all = False):  
  
    #This currently only caters for two queries unioned  
    #Can add more if needed later  
  
    strsql = qry_str1  
  
    if union_all:  
        strsql += " UNION ALL "  
    else:  
        strsql += " UNION "  
  
    strsql += qry_str2  
  
    # then close the query with ';'   
    strsql += ";"  
  
    self.qry_string = strsql  
  
    cursor = self.db.cursor()  
    cursor.execute(strsql)  
  
    if named_tuples:  
        results = self.convert_to_named_tuples(cursor)  
    else:  
        results = cursor.fetchall()  
  
    cursor.close()  
  
    return results
```

vii Section of **select** function from mysql.py:

```
# if UNION then don't run the query just return the full sql  
if kwargs.has_key("UNION"):  
    return self.qry_string
```

viii Section of code from category.js:

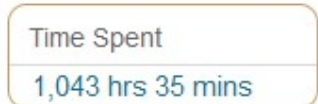
```
totalHoursSpentND2
    .formatNumber(d3.format(", "))
    .valueAccessor(function (d) {
        return Math.floor(d/60);
    })
    .group(totalTimeSpent2);

totalMinsSpentND2
    .formatNumber(d3.format(""))
    .valueAccessor(function (d) {
        return d % 60;
    })
    .group(totalTimeSpent2);

totalDaysSpentND2
    .formatNumber(d3.format(", "))
    .valueAccessor(function (d) {
        return Math.round((d/60)/7.5);
    })
    .group(totalTimeSpent2);
```

ix Section of html from home.html and its rendered output:

```
<div class="chart-title">
    Time Spent
</div>
<div id="total-hours-spent-nd" class="customNDtext">&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</div>
<div id="total-mins-spent-nd" class="customNDtext"> &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</div> <span class="customNDtext">&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</span>
</div>
```



Time Spent
1,043 hrs 35 mins

x Line 8007 code from edited dc.js library:

```
_chart.transitionDuration(10); // good default (MH: this was originally 250)
```

xi Section of code from deal.js:

```
.renderLabel(false)
```

xii Line 6173 from edited dc.js:

```
var _multiple = true;
```

xiii Sections of code from dc.js (edited to include new variable `_theSize`):

```
dc.selectMenu = function (parent, chartGroup) {  
  var SELECT_CSS_CLASS = 'dc-select-menu';  
  var OPTION_CSS_CLASS = 'dc-select-option';  
  
  var _chart = dc.baseMixin({});  
  
  var _select;  
  var _promptText = 'Select all';  
  var _theSize = 4; //added by MH 20160726 THIS IS THE DEFAULT SIZE FOR SELECT MENU  
  var _multiple = true;
```

```
  _chart.data(function (group) {  
    return group.all().filter(_filterDisplayed);  
  });  
  
  _chart._doRender = function () {  
    _chart.select('select').remove();  
    _select = _chart.root().append('select')  
      .classed(SELECT_CSS_CLASS, true);  
  
    switchMultipleSelectOption();  
    switchSizeOption(); //added by MH 20160726  
    _select.append('option').text(_promptText).attr('value', '');  
    renderOptions();  
    return _chart;  
  };
```

```
  _chart._doRedraw = function () {  
    switchMultipleSelectOption();  
    switchSizeOption(); //added by MH 20160726  
    renderOptions();  
    // select the option(s) corresponding to current filter(s)  
    if (_chart.hasFilter() && _multiple) {  
      _select.selectAll('option')  
        .filter(function (d) {  
          return d && _chart.filters().indexOf(String(_chart.keyAccessor()(d))) >= 0;  
        })  
        .property('selected', true);  
    } else if (_chart.hasFilter()) {  
      _select.property('value', _chart.filter());  
    } else {  
      _select.property('value', '');  
    }  
    return _chart;  
  };
```

```

//added by MH 20160726
function switchSizeOption () {
    //if the user gives a size it uses this size
    //or else the default size of 4

    _select.attr('size',_theSize);
}

```

xiv Section of code from graph.js using new size parameter:

```

selectField3 = dc.selectMenu('#menu-select3')
    .dimension(nameDim)
    .group(MISByName)
    .theSize(10)
    .title(function(d) {
        return d.key;
    });

selectField4 = dc.selectMenu('#menu-select4')
    .dimension(modelNameDim)
    .group(MISByModelName)
    .theSize(43)
    .title(function(d) {
        return d.key;
    });

selectField5 = dc.selectMenu('#menu-select5')
    .dimension(functionDim)
    .group(MISByFunction)
    .theSize(20)
    .title(function(d) {
        return d.key;
    });

```