After downloading the datasets based on the unicef (http://mics.unicef.org/surveys) the headers were reviewed and since this was converted to a csv. file from spss. Using the open-source software PSPP the values were separated by commas into columns.

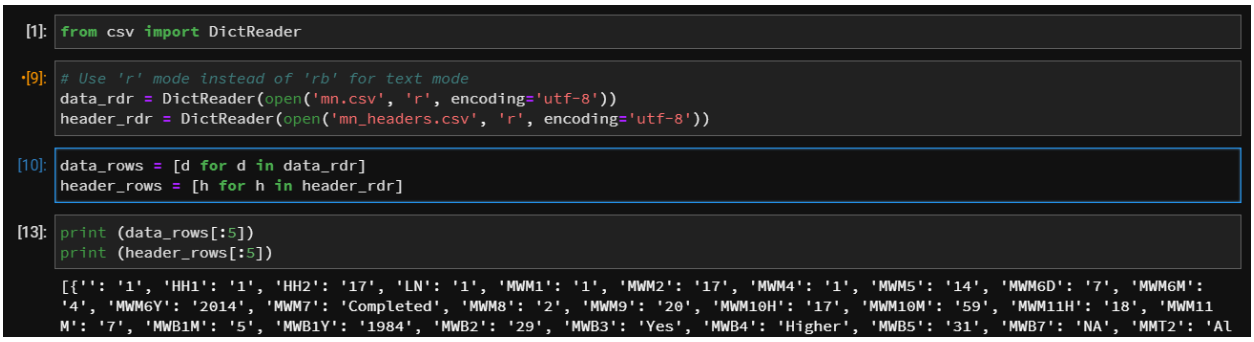
	HH1	HH2	LN	MWM1	MWM2	MWM4	MWM5

As stated in the data wrangling with python book, each of these represents a question or data in the survey and we want more comprehensible versions to understand.

To help translate this, I proceeded to the world bank site for sharing MICS data and they had a translation document to assist with converting the abbreviated headers into full words.

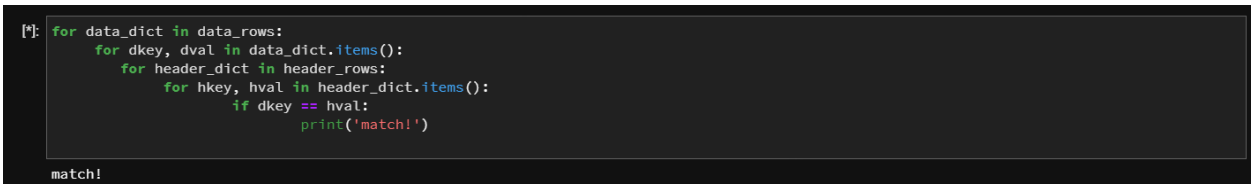
I then entered code that allowed the assigned object to be placed into a list so that it could be preserved and reused.

A small section of the data was then printed so that the content within the file could be determined.



```
[1]: from csv import DictReader

[9]: # Use 'r' mode instead of 'rb' for text mode
     data_rdr = DictReader(open('mn.csv', 'r', encoding='utf-8'))
     header_rdr = DictReader(open('mn_headers.csv', 'r', encoding='utf-8'))

[10]: data_rows = [d for d in data_rdr]
      header_rows = [h for h in header_rdr]

[13]: print (data_rows[:5])
      print (header_rows[:5])

      [{'': '1', 'HH1': '1', 'HH2': '17', 'LN': '1', 'MWM1': '1', 'MWM2': '17', 'MWM4': '1', 'MWM5': '14', 'MWM6D': '7', 'MWM6M':
      '4', 'MWM6Y': '2014', 'MWM7': 'Completed', 'MWM8': '2', 'MWM9': '20', 'MWM10H': '17', 'MWM10M': '59', 'MWM11H': '18', 'MWM11
      M': '7', 'MWB1M': '5', 'MWB1Y': '1984', 'MWB2': '29', 'MWB3': 'Yes', 'MWB4': 'Higher', 'MWB5': '31', 'MWB7': 'NA', 'MMT2': 'Al
```

Each of the data records are then repeated and each of the keys in the data dictionary provided by MICS, so we can now replace the headers with more understandable data. The print result has returned many matches.



```
[*]: for data_dict in data_rows:
         for dkey, dval in data_dict.items():
             for header_dict in header_rows:
                 for hkey, hval in header_dict.items():
                     if dkey == hval:
                         print('match!')

     match!
```

[10]:
```
data_rdr = DictReader(open('mn.csv', 'r', encoding='utf-8'))
header_rdr = DictReader(open('mn_headers.csv', 'r', encoding='utf-8'))
```

[11]:
```
data_rows = [d for d in data_rdr]
header_rows = [h for h in header_rdr]
```

[12]:
```
print(current_directory)
```

```
C:\Users\sahrs\OneDrive\Desktop\Unicef
```

[15]:
```
data_rows = [d for d in data_rdr]
header_rows = [h for h in header_rdr if h[0] in data_rows[0]]
```

[16]:
```
all_short_headers = [h[0] for h in header_rows]
```

[21]:
```
for header in data_rows[0]:
    if header not in all_short_headers:
        print(header)
        index = data_rows[0].index(header)
        if index not in skip_index:
            skip_index.append(index)
        else:
            for head in header_rows:
                if head[0] == header:
                    final_header_rows.append(head)
                    break
```

[23]:
```
def zip_data(headers, data):
    zipped_data = []
    for drow in data:
        zipped_data.append(list(zip(headers, drow)))
    return zipped_data
```

[24]:
```
def create_zipped_data(final_header_rows, data_rows, skip_index):
    new_data = []
    for row in data_rows[1:]:
        new_row = []
        for index, data in enumerate(row):
            if index not in skip_index:
                new_row.append(data)
        new_data.append(new_row)
    zipped_data = zip_data(final_header_rows, new_data)
    return zipped_data
```

[25]:
```
def find_missing_data(zipped_data):
    missing_count = 0
    for question, answer in zipped_data:
        if not answer:
            missing_count += 1
    return missing_count
```

[33]:
```
uniques = [row for row in zipped_data if not
           set_of_keys.remove('%s-%s-%s' %
                              (row[0][1], row[1][1], row[2][1]))]
return uniques, len(set_of_keys)
```

[34]:
```
def save_to_sqlitedb(db_file, zipped_data, survey_type):
    db = dataset.connect(db_file)

    table = db['unicef_survey']
    all_rows = []

    for row_num, data in enumerate(zipped_data):
        for question, answer in data:
            data_dict = {
                'question': question[1],
                'question_code': question[0],
                'answer': answer,
                'response_number': row_num,
                'survey': survey_type,
            }
            all_rows.append(data_dict)

    table.insert_many(all_rows)
```

Above I have inserted functions according to the exercise I have gutted how the program runs. No lines of code executed with the above script. It's just a set of written functions. We need to now work on re-creating how to use all these steps in a main function. The main function is often where Python developers will put code intended to run via the command line. I then added the main function with the code to clean our datasets:

```
def main():
    data_rows = get_rows('data/unicef/mn.csv')
    header_rows = get_rows('data/unicef/mn_headers_updated.csv')
    skip_index, final_header_rows = eliminate_mismatches(header_rows, data_rows)

    zipped_data = create_zipped_data(final_header_rows, data_rows, skip_index)
    num_missing = find_missing_data(zipped_data)
    uniques, num_dupes = find_duplicate_data(zipped_data)
    if num_missing == 0 and num_dupes == 0:
        save_to_sqlitedb('sqlite:///data/data_wrangling.db', zipped_data)
    else:
        error_msg = 'ERROR'
        if num_missing:
            error_msg += 'We are missing {} values. '.format(num_missing)
        if num_dupes:
            error_msg += 'We have {} duplicates. '.format(num_dupes)
        error_msg += 'Please have a look and fix!'
        (print) error_msg
if __name__ == '__main__':
    main()
```

[54]:
```
from csv import reader
import dataset
```

[53]:
```
def get_rows(file_name):
    """Return a list of rows from a given csv filename."""
    rdr = reader(open(file_name, 'rb'))
    return [row for row in rdr]
```

[55]:
```
def eliminate_mismatches(header_rows, data_rows):

    all_short_headers =  [h[0] for h in header_rows]
    skip_index = []
    final_header_rows = []
```

[57]:
```
for header in data_rows[0]:
    if header not in all_short_headers:
        index = data_rows[0].index(header)
        if index not in skip_index:
            skip_index.append(index)
    else:
        for head in header_rows:
            if head[0] == header:
                final_header_rows.append(head)
                break
return skip_index, final_header_rows
```

[61]:
```
def zip_data(headers, data):
    zipped_data = []
    for drow in data:
        zipped_data.append(zip(headers, drow))
    return zipped_data
```

[62]:
```
def create_zipped_data(final_header_rows, data_rows, skip_index):

    new_data = []
    for row in data_rows[1:]:
        new_row = []
        for index, data in enumerate(row):
            if index not in skip_index:
                new_row.append(data)
        new_data.append(new_row)
    zipped_data = zip_data(final_header_rows, new_data)
    return zipped_data
```

[63]:
```
def find_missing_data(zipped_data):
    missing_count = 0
    for response in zipped_data:
        for question, answer in response:
            if not answer:
                missing_count += 1
    return missing_count
```

[64]:
```
def find_duplicate_data(zipped_data):

    set_of_keys = set([
        '%s-%s-%s' % (row[0][1], row[1][1], row[2][1])
        for row in zipped_data])
    #TODO: this will throw an error if we have duplicates- we should find a way
    #around this
    uniques = [row for row in zipped_data if not
                set_of_keys.remove('%s-%s-%s' %
                                   (row[0][1], row[1][1], row[2][1]))]
    return uniques, len(set_of_keys)
```

[66]:
```
def save_to_sqlitedb(db_file, zipped_data, survey_type):

    db = dataset.connect(db_file)

    table = db['unicef_survey']
    all_rows = []

    for row_num, data in enumerate(zipped_data):
        for question, answer in data:
            data_dict = {
                'question': question[1],
                'question_code': question[0],
                'answer': answer,
                'response_number': row_num,
                'survey': survey_type,
            }
            all_rows.append(data_dict)
    table.insert_many(all_rows)
```

[72]:
```python
def main():
    try:
        data_rows = get_rows('mn.csv')
        header_rows = get_rows('mn_headers.csv')
        skip_index, final_header_rows = eliminate_mismatches(header_rows, data_rows)
        zipped_data = create_zipped_data(final_header_rows, data_rows, skip_index)
        num_missing = find_missing_data(zipped_data)
        uniques, num_dupes = find_duplicate_data(zipped_data)

        if num_missing == 0 and num_dupes == 0:
            save_to_sqlite('sqlite:///data_wrangling.db', zipped_data, 'mn')
        else:
            error_msg = 'ERROR'
            if num_missing:
                error_msg += ' We are missing {} values. '.format(num_missing)
            if num_dupes:
                error_msg += ' We have {} duplicates. '.format(num_dupes)
            error_msg += 'Please have a look and fix!'
            print(error_msg)  # Chnaged to print instead of logging for simplicity
            # Optionally, raise an exception or handle the error as appropriate
    except Exception as e:
        print(f"An error occurred: {e}")
```

The required code has been documented and organised, these functions can now be reused with other datasets from unicef.