Instructions

1. The aim of this assignment is practice and demonstrate knowledge of file IO, dictionaries, iteration, and conditionals.
2. The grade of the assignment is out of 100 pts, with three parts and optional extra credit.
3. Students must use Python3 (NOT Python2) to write the code.
4. Don't share code or get code from the internet, but working together is encouraged. If you choose to work with someone you must mention them in one of the first comments in the file.
5. Students are expected to write the code that is easily readable. To this end, they must employ meaningful variable names and comment their code properly.
6. The output of the code requires to be precisely as shown in the sample runs.
7. Students must put the code in a **single** file named `FirstName_LastName_StudentID_HW4.py` and submit it through Canvas. **DO NOT** zip your submission. Zip is more likely used when there is more than one file submission.
8. It is important that you practice commenting your code, so **each file submitted without comments will have 5 points deducted** from its score (even if the program works perfectly)!
9. Use the skeleton code provided for your submission. It introduces the use of `argparse` and will help lay the foundation for your work.
10. Use PyCharm for this assignment! You will not be able to develop in Colab for this type of python script.

Introduction

We want you to write a python program that can read in a CSV (comma separated values) file, and perform calculations on the data inside. To get input on what options to use and the name of the CSV file, we want you to use a library called `argparse`. This allows you to set up different arguments a user can specify to the script. For this assignment, all the argparse code is written for you, and examples for interacting with it are given in the sample code. **Be sure to download all the files from the assignment location on Canvas!**

CSV Data

The CSV file you will have to parse (`avocados.csv`) contains data about sales of avocados. It is real data, and came from the site "Kaggle", which has many public datasets for examination. The dataset for this assignment came from here: https://www.kaggle.com/neuromusic/avocado-prices/.

It contains a header as its first row. For this file, the header row will start with a pound sign (`#`) and should be ignored during your calculations. The header row describes the ordering of values in each row of the dataset. The dataset will contain more data than you need for this assignment. When developing code in "real life", it is best to parse the header to determine the index of the columns you care about. In this example, you can just hard code in the column indices when performing your calculations.

There are TWO data files given to you: `avocados.csv` and `avocados.small.csv`. The first file has all the data available, and the second has been reduced down to only 30 records to make it easier to develop your code. The examples given (and the correct values) relate to the small dataset, but your program will be tested on the large dataset.
Here is the first few rows of the small dataset:

```
#WeekIdx,Date,AveragePrice,Total Volume,4046,4225,4770,Total Bags,Small Bags,Large Bags,XLarge Bags,type,year,region
0,2016-12-25,0.94,5414937.46,1579234.89,2192460.84,99997.75,1543243.98,1405419.8,99069.58,38754.6,conventional,2016,California
0,2016-12-25,0.84,2641775.31,794840.17,669585.49,65359.34,1111990.31,1032762.48,49937.35,29290.48,conventional,2016,LosAngeles
0,2016-12-25,1.07,428247.63,119611.9,263467.16,811.18,44357.39,39136.56,2868.61,2352.22,conventional,2016,Sacramento
0,2016-12-25,0.89,473730.19,118030.19,191754.65,14438.25,149507.1,125660.66,22080.88,1765.56,conventional,2016,SanDiego
0,2016-12-25,1.18,690027.33,164595.96,462872.97,2525.58,60032.82,52277.99,7459.83,295.0,conventional,2016,SanFrancisco
0,2016-12-25,1.17,81458.1,11324.9,25698.37,0.0,44434.83,37064.95,7369.88,0.0,organic,2016,LosAngeles
0,2016-12-25,2.41,4567.83,1449.65,2867.8,2.68,247.7,247.7,0.0,0.0,organic,2016,Sacramento
0,2016-12-25,1.43,12022.95,715.07,7537.57,0.0,3770.31,540.0,3230.31,0.0,organic,2016,SanDiego
0,2016-12-25,2.56,16169.17,5599.25,8335.16,0.0,2234.76,2234.76,0.0,0.0,organic,2016,SanFrancisco
```

Parameters

Argparse supports many different types of parameters. For this assignment, three parameters (plus this generated help message) are given:

```
usage: Read CSV input about avocados and print total amount sold
       [-h] --input INPUT [--group_by_region] [--organic]

optional arguments:
  -h, --help            show this help message and exit
  --input INPUT, -i INPUT
                        input CSV file
  --group_by_region, -r
                        Calculate results per region (default: calculate for
```

```
                        all regions)
  --organic, -o         Only calculate for organic avocados (default:
                        calculate for conventional and organic)
```

Each parameter has a long (`--input`) and short (`-i`) form. You can use either one of them to specify the input file.

Some parameters are marked as required; if they are not specified, argparse will print a message describing what error was made and a usage message:

```
usage: Read CSV input about avocados and print total amount sold
       [-h] --input INPUT [--group_by_region] [--organic]
Read CSV input about avocados and print total amount sold: error: the following
arguments are required: --input/-i
```

Some parameters require a value to be specified, such as the `--input` parameter for this file.
Incorrect: `--input`
Correct: `--input avocados.small.csv`

The other parameters (`--group_by_region` and `--organic`) only need to be specified.
Incorrect: `--organic True`
Correct: `--organic`

In order to use these parameters during development, you need to edit your 'Run Configurations' in PyCharm. To do this:

1. Run the program once by right clicking on the file and selecting 'Run'
2. Go to the upper-right where your program is listed and click the drop-down menu
3. Select 'Edit Configurations'
4. In the 'Parameters' box, enter the arguments you want to use. For example: `--input avocados.small.csv`
5. File paths are (by default) relative to where your script is located, so save the two data CSV files in the same location as your script for easiest use.

Part 1: Default Behavior (50 points)

The default behavior of your program is to calculate the total sales of avocados. For each row (or line) in the CSV, this can be calculated by taking the `AveragePrice` (column index 2) and

multiplying it by the `Total Volume` (column index 3). In the first row of the small dataset, the `AveragePrice` is 0.94 and the `Total Volume` is 5414937.46, so the total amount sold is 5090041.21. We want you to calculate the total sales for each row, and sum all these values up for the final result. Be sure to **format them to have two digits after the decimal**.

Arguments: `--input avocado.small.csv`
Output:
```
Total Sales:  34448787.93
```

Part 2: Group By Region (30 points)

If the user specifies the `--group_by_region` flag, then we additionally want you to specify the total sales per region (the last column in the CSV). Your program should **still print the total sales** after printing the sales by region. Note that the same region will occur multiple times in the data, so you must keep a running total for each region (**Hint:** you can use a `dict` to do this). When you print the regions and their total sales, **print the regions in alphabetical order**.

Arguments: `--input avocado.small.csv --group_by_region`
Output:
```
California:  19478378.92
LosAngeles:  8338465.51
Sacramento:  1740771.83
SanDiego:  1644976.51
SanFrancisco:  3246195.16
Total Sales:  34448787.93
```

Part 3: Filter for Organic (20 points)

If the user specifies the `--organic` flag, then we want your program to **only include organic avocado sales**. You can determine whether the sales are organic from the `type` column (the values are either '`conventional`' or '`organic`'). Your program should do this regardless of whether the `--group_by_region` flag is set.

Arguments: `--input avocado.small.csv --organic`
Output:
```
Total Sales:  1449534.29
```

Arguments: `--input avocado.small.csv --organic --group_by_region`

Output:

```
California:  800426.44
LosAngeles:  392172.02
Sacramento:  42216.08
SanDiego:  78896.86
SanFrancisco:  135822.88
Total Sales:  1449534.29
```

Part 4: Output to a file (20 points EXTRA CREDIT)

For up to 20 points of **Extra Credit**, add another parameter `--output` which lets the user specify an output file. If this parameter is NOT set, the program should print to stdout (as it does currently). If it IS set, the program should not print anything on the screen, and instead should **write the results out to the file specified in CSV format**. Both the `--organic` and `--group_by_region` flags should function as normal.

Arguments: `--input avocado.small.csv --output sale_summary.csv`
Contents of `sale_summary.csv`:

```
Total Sales,34448787.93
```

Arguments: `--input avocado.small.csv --group_by_region --output regional_sale_summary.csv`
Contents of `regional_sale_summary.csv`:

```
California,19478378.92
LosAngeles,8338465.51
Sacramento,1740771.83
SanDiego,1644976.51
SanFrancisco,3246195.16
Total Sales,34448787.93
```

Skeleton Code

The skeleton code is given below. Be sure to remove the three `print` statements in the `main()` function and replace them with your logic. We recommend writing separate functions to handle extracting total sales for a line in the file, determining the region from a line, and determining whether a line refers to organic or conventional avocados.

```python
import argparse

def parse_args():
    """
    parse_args takes no input and returns an Namespace dictionary describing
    the arguments selected by the user.  If invalid arguments are selected,
    the function will print an error message and quit.
    :return: Namespace with arguments to use
    """
    parser = argparse.ArgumentParser("Read CSV input about avocados and "
                                     "print total amount sold")
    # required parameters
    parser.add_argument('--input', '-i', dest='input', required=True, type=str,
                        help='input CSV file')
    # optional parameters
    parser.add_argument('--group_by_region', '-r', dest='group_by_region',
                        action='store_true', default=False,
                        help='Calculate results per region '
                             '(default: calculate for all regions)')
    parser.add_argument('--organic', '-o', dest='organic', action='store_true',
                        default=False, help='Only calculate for organic '
                                            'avocados (default: calculate '
                                            'for conventional and organic)')
    return parser.parse_args()

def main():
    """
    The main body of the program.  It parses arguments, and performs
    calculations on a CSV
    """
    # get arguments entered by users
    args = parse_args()
    # TODO remove these print statements
    # This code is provided as an example
    print("Input file:      {}".format(args.input))
    print("Group by region: {}".format(args.group_by_region))
    print("Only organic:    {}".format(args.organic))

if __name__ == "__main__":
    main()
```