Introduction:
Utility analysis is a tool written in Python to be used by employees at SunSolar Solutions with the goal of reducing and simplifying the work needed to complete a utility analysis. The program is supplied with a folder containing all information available for a HO (APS, Bills, Usage SE, Sunrun, Array layout, etc.) and generates a report with the system's monthly production, what the HO used and bought, and the expected production according to PVWatts.

Definitions:

Program – The program being designed, Utility Analysis
Python packages – Separate modules to be installed. They will be used by the program to do calculations, file reading, file outputting, HTTP requests, and error handling.
Model View Controller – Architecture Design pattern used to separate components of the program into three distinct components, namely model, view and controller.
CSV – Comma Separated value sheet, a file format used with excel and the Pandas package.
Command Line Interface (CLI): Text based interface used for displaying info to the user, and where the user will enter input for the program. Also called the Terminal
HO – Homeowner

Description:
Utility Analysis is a helper tool intended to make report creation easier. Written entirely in Python we take advantage of the pandas and numpy packages to perform all of the data science. Upon initialization the program receives a folder with data for the HO and prompts the user for more information if the folder is found. If not an error is thrown and the program terminates. Using the billing dates, either from "APS-Usage.csv" or the user, the solar production is generated and output to a CSV file (APS, SE, Sunrun). Using the Requests and Pypvwatts packages the program will use the PVWatts API to get the PVWatts information for the system. The values are slightly different as it is a different database, but only by a small amount. We have several databases to choose from with the PVWatts API. A final report will be generated with all the information needed to analyze the system (Address, Billing Dates, Solar Production, PVWatts, etc.) The program will then terminate.

The program will be built using the Model-View-Controller (MVC) design pattern. The MVC design pattern will separate the aspects of the program into 3 parts: Model, View, and Controller.

Model: The Model component is responsible for all of the logic that happens behind the scenes. It is in charge of reading all files, outputting all files, and manipulating all of the data necessary to perform an Analysis. Only the model component is allowed to access the data, no other component has direct access. Catches all error messages and passes them to the controller component.

View: The View component is in charge of displaying all information to the user in the CLI. The view component does not handle ANY LOGIC OR USER INPUT, ONLY WHAT IS DISPLAYED IN THE CLI. The view component does not handle any data, nor does it take any user input. Takes error messages from the controller component and displays them to the user.

Controller: The Controller component is in charge of the program flow. This is the main chunk of the program that delegates all tasks, handles all user input and passes it to the model component. Any information that needs to be displayed to the user from the model goes to the controller component, then from controller to the view component. The controller component is responsible for handling errors from the model component and passing them to the view component to be displayed.

The reason I chose the MVC design pattern for this tool is that it maximizes the modularity, maintainability, readability, and scalability of the program. Although the original intent was for this program to be run in the CLI, the MVC design pattern allows for a user interface to be added by replacing/updating the View Component. The MVC design pattern allows for the program to be easily expanded and modified through the individual components.

Intended Use and Dependencies:

This program is written entirely in python, and requires the following packages to be installed:

Pandas – Reading, writing, manipulating csv files
Numpy – Used for computations on Pandas Data Structures
Pypvwatts – PVWatts API, used to retrieve PVWatts information for the HO's system
Requests – Helper packaged used for HTTP requests, needed to make PVWatts API request

The user will initialize the program from the command prompt by typing "python controller.py 'fname'" where fname is the path to the folder for the HO. The contents of the folder have to be set up in a specific way for the tool to be able to properly recognize and label everything, more detail below. The user will then enter the type of analysis they would like done, and the PTO date, and prior usage. Using that info the program will generate a production report for whatever folders are present (APS, SE, Sunrun). An error message will be thrown if any of the **necessary** folders are not found, or empty. The user will then enter the information needed for PVWatts (system size, address, array size, array tilt, etc.) and a PVWatts report will be generated. A final report will be generated with all the information needed to analyze the system (Address, Billing Dates, Solar Production, PVWatts, etc.) The program will then terminate.

As it stands, the folder containing all of the information for the HO has to be set up and named correctly. The directory contents will vary depending on the analysis that needs to be done.

APS(RCP or Net Metering)

Input:

1. **APS-Production: Contains all ".csv" files with APS production data

2. SE-Production (IF USING SE MONITORING, OMIT IF NOT): Contains all ".csv" files with SE production data
3. Sunrun-Production (IF USING SUNRUN, OMIT IF NOT): Contains all ".csv" files with sunrun production data.
4. **Solar-Exported: Contains all ".csv" files with production data
5. *APS-Usage.csv: APS Usage sheet for the HO

Output:

1. APS-Production.csv: CSV file with all the APS production data gathered and summed up per our billing cycle determined from the PTO date and APS-Usage.csv, sorted by date.
2. SE-Production.csv: CSV file with all the SE production data gathered and summed up per out billing cycle determined from the PTO date and APS-Usage.csv, sorted by date.
3. Sunrun-Production.csv: CSV file with the Sunrun production data in a single sheet, and summed according to our billing cycle.
4. Solar-Exported.csv: CSV file with all the Solar-Exported data gathered and sorted by date
5. Report.csv: Report file with APS-Usage data matching out billing cycles, APS Production, Solar-Exported, SE Production, and PVWatts daily adjusted numbers to be entered into the PVWatts calculator

TEP

Input:

1. **SE-Production(IF USING SE MONITORING): Contains all ".csv" files with SE production data in it

   OR

2. **Sunrun-Production (IF USING SUNRUN, OMIT IF NOT): Contains all ".csv" files with sunrun production data.

Output:

1. SE-Productiuon.csv (IF PRESENT): CSV file with all the SE production data gathered and summed up per out billing cycle determined from the PTO date and our billing cycle, must be entered manually
2. Sunrun-Production.csv (IF PRESENT): CSV file with the Sunrun production data in a single sheet and summed according to our billing cycle.
3. Report.csv: Report file with the HO address, billing rate plan, Billing cycle, SE Production, and PVWatts adjusted daily values

SRP

Input:

1. **SE-Production(IF USING SE MONITORING): Contains all ".csv" files with SE production data in it

Output:

1. SE-Productiuon.csv (IF PRESENT): CSV file with all the SE production data gathered and summed up per out billing cycle determined from the PTO date and our billing cycle, must be entered manually
2. Report.csv: Report file with the HO address, billing rate plan, Billing cycle, SE Production, and PVWatts adjusted daily values

NVE

Input:

1. **SE-Production(IF USING SE MONITORING): Contains all ".csv" files with SE production data in it
   OR
2. **Sunrun-Production (IF USING SUNRUN, OMIT IF NOT): Contains all ".csv" files with sunrun production data.

Output:

1. SE-Productiuon.csv (IF PRESENT): CSV file with all the SE production data gathered and summed up per out billing cycle determined from the PTO date and our billing cycle, must be entered manually
2. Sunrun-Production.csv (IF PRESENT): CSV file with the Sunrun production data in a single sheet and summed according to our billing cycle.
3. Report.csv: Report file with the HO address, billing rate plan, Billing cycle, SE Production, and PVWatts adjusted daily values

*As it stands, "APS-Usage.csv" can only handle one address inside the sheet. If more than one address is found, the program will print all of the address and ask the user to remove excess data

****Folder is NECESSARY to complete the analysis.**

Requirements and System Features:

Requirement	Description	Status
getAPS	**Gather all APS data and put it in a single sheet. Takes our billing range, sums production data according to our billing cycle. Outputs to APS-Production.csv**	**Implemented**
getSE	**Gather all SE data and put it in a single sheet. Takes our billing range and sums production data according to**	**Partially implemented: Needs to handle summation with manually entered**

	our billing cycle. Outputs to SE-Production.csv	**billing cycle for Non APS utility companies**
getSunrun	**Gather all SE data and put it in a single sheet if it is not already. Sums up production values based on our billing range from APS or manually entered**	**Not implemented**
getSolarExported	**Gather all Solar Exported data (Credit-To-APS) in a single sheet. Outputs to Solar-Exported.csv**	**implemented**
getPVWatts	**Using the PVWatts API gets the monthly production values for each unique array in the HO system. Outputs monthly values scaled by our billing range, just like our PVWatts calculator**	**Partially implemented. Currently only outputs daily adjusted values. Does not properly scale according to our billing cycle.**
getPriorUsage	**Asks the User to enter any prior usage data we have, either from AMPS or from the Utility Company**	**Not implemented**
APSAnalysis	**Runs an APS analysis on the HO system. Calls getAPS, getSolar-Exported, and getSE(if folder present). Takes all production and usage data and outputs a report with APSProduction, SE Production, Solar Exported, Billed Amount, Total-Bought, Service Plan, Billing Days, Billing Cycle, Prior Usage, and PVWatts Adjusted monthly Values to report.csv**	**Partially implemented. Still need to get prior usage from User. Also handle case where HO has sunrun and not SE**
TEPAnalysis	**Runs a TEP analysis on the HO system. Calls getSEProduction or getSunrun production. Outputs SE or Sunrun production, PVWatts**	**Not Implemented**

	adjusted monthly values. Outputs all data to report.csv	
SRPAnalysis	**Runs a SRP analysis on the HO system. Calls getSEProduction or getSunrun production. Outputs SE or Sunrun production, PVWatts adjusted monthly values. Outputs all data to report.csv**	**Not Implemented**
NVEAnalysis	**Runs a NVE analysis on the HO system. Calls getSEProduction or getSunrun production. Outputs SE or Sunrun production, PVWatts adjusted monthly values. Outputs all data to report.csv**	**Not Implemented**

Nonfunctional requirements:

Requirement	**Description**	**Current Rating/5**
Usability	**The software must be easy to use. Must be clear what the program is asking of the User.**	**3**
Forgiveness	**The program must be forgiving in handling user input. Allow for users to re renter information instead of exiting program**	**2**
Security	**The user should be not able to access information outside of the HO in question**	**3**
Reliability/Accuracy	**The program must provide reliable information in 99.99 percent of cases**	**3, pvwatts needs tweaking**

Maintainability	**The program should be easy to maintain and update to stay relevant**	**3**
Design	**The program must use the Model View Controller Design pattern to keep data, control, and display separate**	**1**
Operating System	**The program must be usable on both MacOS and Windows 10.**	**2, have not tested anything on mac. Shouldn't be a problem since Python is virtually the same across both systems.**