**1. Data – What sort of data will you need to store and retrieve? For your projects,**

**this can be as simple as a list of variables, their types, and their “scope” – are**

**they global to all methods or local to specific ones?**

A majority of the data will be retrieved from the NOAA GHCN (Global Historical Climate Network) database in the form of CSV files containing several atmospheric observation values. A smaller amount of data will come from NOAA’s NAO/AO historical database, also in the form of a CSV file. Due to the timescale of this project (30 – 40 years), it is necessary to use monthly data (NOAA calculated averages, min/ max, totals) as opposed to daily records which would require an overwhelming amount of storage space as well as computational time taken to preform calculations using such high-resolution data.

Most of the data will be floating point data types such as temperature values, NAO/AO index values and snow water equivalence totals, however some categorical data will be string data types such as precipitation quality (snow, rain, mix) or NAO/AO phase descriptions (‘positive’ or ‘negative’). An organized list of data and their corresponding data types is detailed as follows:

* Precipitation
  + Quality – String
  + Snow Water Equivalence – Float
  + Snow Depth – Float
  + Number of days with 1+” of precipitation (storm frequency) - Float
* Temperature
  + Min/Max – Float
  + RMS Average – Float
* NAO/AO
  + Phase Description – String
  + Peak/ Valley Frequency – Integer
  + RMS Average – Float

The variables will be organized into three main data frames: Precipitation, Temperature, NAO/AO. Since the data frames will be used in a variety of analysis techniques, they will be global to all methods.

**2. Architecture – This is a map of all the moving parts of your application. What**

**input is expected from the user or environment? What output does the user or**

**environment expect? And what components manage that process? For your**

**application, there is only one component unless you are using an external library.**

The overall process involves typical data science techniques. Raw data is first loaded into the program (R Studio) and merged into defined data frames. Missing data is interpolated using the na.approx{} data frame function. Due to the volume of data, a judicious determination will be made on a subset of data for how consistent the data appears upon filling missing values.

Once the cleansing of data is complete, the data frames will act as inputs to linear and or polynomial regression models that seek to find trends in the climate data. Additionally, correlation coefficients between the NAO/AO index trend (if one is found) and precipitation and temperature trends will also be explored using the cor.test() R function which returns both the correlation coefficient as well as the significance level (p-value).

Temporal trends are compared using the above-mentioned machine learning techniques as well as plotting libraries such as ggplot, and external programs/ libraries such as tableau or matplotlib which output PNG images of the analysis. Examples include polynomial and linear regression graphs of variables vs time.

Spatial trends are compared using GIS software such as ArcPro. Therefore, spatial data will need to be transformed from a data frame to a CSV file in order to be imported correctly. Examples include subsection of Maine into three broad regions:

* Western to Northern Maine (Rangely, Moosehead Lake, Katahdin, Caribou)
* Interior Maine (Rumford, Augusta, Newport, Bangor, Lincoln)
* Costal Maine (York, Portland, Downeast, Acadia Lubec)

The purpose of this is to differentiate weather trends as clearly as possible. For instance, the type of weather seen in Rangely is not the same as Portland and so changes should be visualized relative to their respective zonal regions.

In summary, the main program (using R Studio) will house three major components: data loading/cleansing, machine learning analysis, and data transformation/ exporting. Libraries such ggplot will be used for temporal visualizations, an outside program ArcPro will be used to visualize spatial data analysis.

**3. User Interface – What will your application look like to the user? This section of**

**the document should be at least one “mock-up” picture where you sketch out**

**what the program will look like to the user.**

The UI will simply be R Studio’s interface. The main input expected from the user is a file location that houses the collection of downloaded climate CSV files. This input function will read in each CSV file and append the initially empty data frames with the appropriate data values until all the raw data is compiled. Since NOAA has a unique web-based database service, it is not possible to directly access the data without downloading it in discrete packages, therefore this process is necessary to load in the 40 years of climate data and organize it effectively.

Additional input expected from the user is a choice for which analysis technique to use on what data frame. An example may be that the user wishes to generate a linear regression for a specific precipitation type – rain. The user will first be given an opportunity to select a variable by it’s corresponding data frame column name, and then given a choice of the two regression techniques (linear and polynomial). The output of this function will be a generated graph of the desired analysis.

**4. Procedural – What methods or functions will you need to accomplish your**

**task? How will the program flow in terms of what methods or functions are called**

**and when?**

The very first method to be called is for user input (using readline()) of the raw data file path as well as a variable name for the ensuing data frame. For simplicity it can be called loadData(file,dfName). This function calls another function read.csv(), passing the ‘file’ parameter to it, which loads the data into a data frame object. The data frame is assigned to the user defined parameter ‘dfName’ which acts as the variable name of the data frame.

All data of a specific variable and year is merged into the data frame assuming a hierarchical structure to the stored data. That is, the raw data folder that houses each variable has within it, 40 subfolders by year. Finally, the data frame is returned to the user at the end of the function for their inspection. Missing values are generated using the na.approx() function.

The user is then asked to choose one of five options, load data, preform data analysis, preform statistical analysis, export a data frame to CSV, or exit. This function is called userOptions()

While the user does not choose exit…

* If the user choses load data, the loadData(filename) function is called again in case they wanted to load additional data. userOptions() is then called
* If the user choses data analysis the second method analyzeData(df) asks the user which data frame they would like to use. The parameter ‘df’ is simply whatever name they choose for a particular data frame. The user is then asked for which variable of that data frame they would like to analyze. At the same time, a list of available variables (col names) will be printed out. The user will type in the variable name using readline() which is then used as the input to the df.head() function to select just that column’s data from the data frame and assign it to a local variable to be used within the regression function.
  + The user is then asked what type of regression analysis they would like to preform on the selected data, they will be given two options: Linear and Polynomial.
  + Once either one is chosen, the chosen regression function will be called and passed in the selected data from the data frame. The output of either function will return a graph (using ggplot or a similar library) of the selected data. The user (assuming they are competent with R) is now able to export the graph if they choose, using the graph section of R Studio.
  + The userOptions() function is called again
* If the user chooses export data to csv, a third method selectExport(df,filepath) is called, where the user passes in the name of the data frame they wish to export, and the location they wish to export it to. write.csv() then uses these parameters to export the desired data frame. Then the userOptions() function is called again.
* If the user chooses statistical analysis, a fourth method dataStats(df) is called. Similar to the analyzeData function, the user will be prompted by readline() to enter the name of a data frame column they wish to see statistics on. This will be assigned to a temporary variable ‘colName’ several statistics will be generated using the stats, DescTools, and distributional libraries (imported/ installed in the beginning of the program) on the variable values:
  + Mean - stats.weighted.mean(colName)
  + Median – median(colName)
  + Mode – DescTools.Mode(colName)
  + Standard Deviation – sd(colName)
  + Correlation Coefficient - cor.test() (R function)
  + Variance – distributional.variance(colName)
  + RSM – stats.summary(colName)

Once the user selects ‘exit’ the program concludes by removing currently used variables, data frames, etc. from R studio.