Designing Sustainable Water Management

NOTES ON FINAL REPORT

▸Use high resolution images for publication (> 300ppi)

▸Use BibTeX for reference

▸Tables! (<https://www.tablesgenerator.com/>)

▸There may be blank pages (even numbered page)

▸HMC Clinic Report class—adapted and modified to fit the CMC Capstone Report format

https://www.overleaf.com/4174268287gcnmcgsdxyhh

1. Introduction (Min)
   1. Client Overview
   2. Problem Description
2. Data
   1. Demand Prediction (Min)
      1. Background
      2. Data Cleaning
      3. Descriptive Statistics
   2. Cost Minimization (Zach and Rishi)
      1. Background
         1. What is a zone
         2. What is a well
         3. What is a booster
         4. What is a PRV
      2. Data Cleaning
      3. Descriptive Statistics
3. Models (Solutions and Back-end Results)
   1. Demand Prediction (Back end)
      1. Regression (Min)
      2. Classification (Jen)
   2. Cost Minimization (Back end)
      1. Linear Programming Model
      2. Units used: kWh
      3. EVWD Constraints
      4. Adaptability
      5. Platforms Used
   3. User Interface (Back end) (Zach and Jen)
4. Results: User Interface (Front end)
   1. Demand (Min + Jen)
   2. Cost-minimized results (Zach and Rishi)
      1. Pump Output Results
      2. Emphasis on Wells
      3. PRV inefficiency
      4. Zone by Zone analysis
      5. Notes about how this changes under maintenance
5. Discussion and Conclusions
   1. What were our stretch goals and explain how we completed all of them
   2. Future steps for future teams
6. Appendix

## Client Overview

East Valley Water District (“the District”) is a public rate-based organization in Highland, CA, that provides water and wastewater services to over 100,200 residential and commercial customers. Every day, the District supplies roughly 60 gallons of water per person and over 16 million gallons of water throughout its 30.1 square mile area.

## Problem Statement

Currently, the optimal decisions to water flow routing are made by key staff with expansive knowledge and years of experience operating the distribution system. When they are unavailable, the District resorts to “do-what-we-did-yesterday” type decisions which usually meet the needs of customers and satisfy the District’s core mission in water provision. As experienced staff leave the District and systematic and environmental conditions change, it is uncertain whether relying on institutional knowledge and past decisions would be sustainable in the long term. The District has requested that the students in the Team develop an easy-to-use tool that can inform and guide new Operations staff, in order to make flow routing decisions in the most optimal, low-cost way. Building upon the work from the last semester, the Team will develop a water demand prediction and linear programming model along with an user interface designed for regular usage by Operations staff.

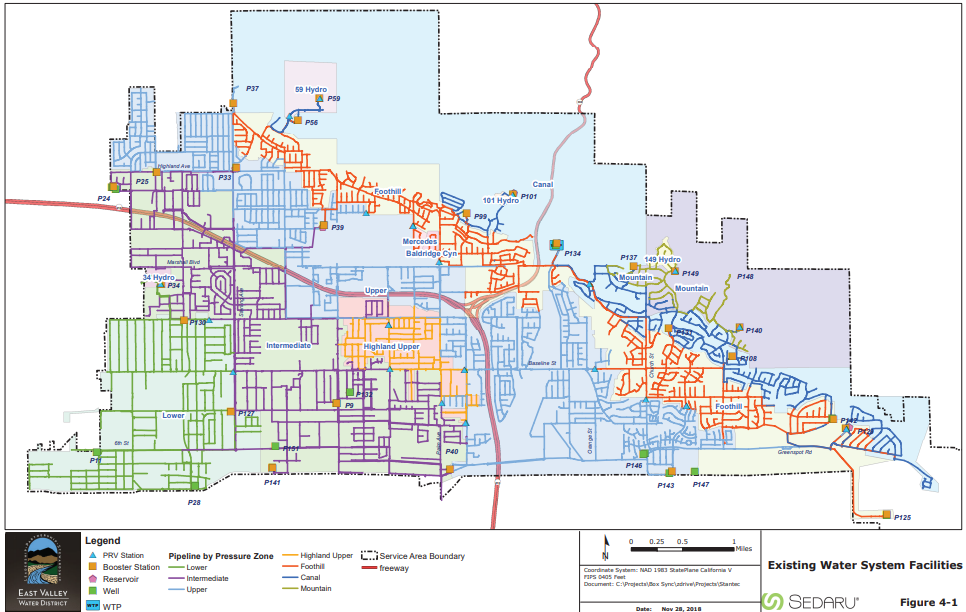
## Stakeholders

Stakeholders in this project are listed below:

* **Rocky Welborn:** Mr. Welborn is the Water Reclamation Manager at EVWD and provided our team with data, supported our team in model development, and answered questions regarding the flow routing tool.
* **Kerrie Bryan:** Ms. Bryan is the Director of Administrative Services at EVWD and was responsible for overseeing the project and organizing the team’s visit to EVWD’s Board of Directors.
* **Rate-payers**: The District’s residential and commercial customers using water services of the District are indirect stakeholders in our project as our final deliverables will have an impact on the District’s water distribution strategy.
* **Board of Directors:** EVWD’s Board of Directors is a direct stakeholder as it approves important decisions for the District and represents the District’s rate-payers. Additionally, we presented our final deliverables to the Board of Directors.

Data for Demand Prediction

**Background:**

Figure 1. Map of pressure zones in the District

Our original data sets supplied by the District consisted of 546 daily consumption files from 2020-09-20 to 2022-04-16 from the District’s customer meters and 15 zone files containing meter numbers belonging to each of the District’s 15 pressure zones as shown in Figure 1. The daily consumption data contained 9 different variables describing how much water in cubic feet a particular meter consumed in that day while the meter numbers data for each zone contained 44 variables, covering the characteristics and conditions of meters. They both contained *MeterNumber*, which could be used to assign a pressure zone to a particular daily consumption of a meter. Also, daily climate data for Highland, CA, for the corresponding period were obtained from the California Irrigation Management Information System (CIMIS) Station Reports. This CIMIS dataset contained 19 variables, including *Date* that could be used to assign a weather condition to a particular daily consumption.

**Data Import and Variable Selection:**

The daily consumption files were Excel spreadsheets in XLSX and titled with their respective dates. However, the District switched the title format on 2021-09-30 from”M-D-YYYY.xlsx” to “M-D-YY.xlsx”, and there were two differently titled groups of daily consumption files. Therefore, they were separately placed in “M-D-YYYY” and “M-D-YY” folders, each with a RMD file, “M-D-YYYY.Rmd” for “M-D-YYYY.xlsx” and “M-D-YY.Rmd” for “M-D-YY.xlsx”, which was created to import and bind every XLSX file in that folder in a single data frame with its title as *Date* variable and 3 other variables, *AccountNumber, MeterNumber,* and *Consumption*. The data types of *Date* and *Consumption* were converted from character to date and numeric, respectively, for time-dependent numerical analysis. The resulting two data frames were loaded into the global environment and combined together in “DataWrangling.Rmd” as “df” for further wrangling. Since *Consumption* was in cubic feet, it was multiplied by 7.48051948 gallons/cubic feet to convert its unit to gallons.

The 15 zone files were imported in “DataWrangling.Rmd” as “meter\_df” with only MeterNumber selected out of 44 variables and filtered to remove any rows with NA value in MeterNumber. A new variable *Zone* was created for each of them to indicate their respective pressure zone. Then all of them were bound together to produce a data frame with *Zone* and *MeterNumber*. During this process, “Baldridge Cyn” and “Mercedes” zones were added to “Foothill”, and “Highland Upper” was added to the “Upper” zone because they share similar geography and sources of water. Since we were primarily concerned with the usability of demand prediction model by Operations staff on a daily basis, the CIMIS file was imported in “DataWrangling.Rmd” as “weather\_df” with only the climate variables that are readily accessible on weather forecasts, namely *Precip (in), Max Air Temp (F), Min Air Temp (F), Max Rel Hum (%), Min Rel Hum (%), and Avg Wind Speed (mph)*, in addition to *Date*.

**Data Cleaning:**

Initially, the merged daily consumption data frame had 10,956,684 observations from the customer meter readings, which totalled 20,955,968,284 gallons of water. Among them were 16,261 observations with negative *Consumption* values that totalled -921,152,678 gallons. While the exact cause of negative readings was unknown, they could be due to corrupted radio transmissions or malfunctioning meters and, therefore, were removed from the data set, leaving 10,940,423 observations accounting for 21,877,120,962 gallons.

The resulting daily consumption data frame was merged with “meter\_df” to assign a pressure zone to each observation. Approximately 23% or 2,550,977 observations totalling 2,275,857,026 gallons of water were unmatched and removed. The remaining observations were grouped by *Date* and *Zone* to calculate the daily consumption with those on 2020-09-20 being removed due to having only 4 zones out of 12, resulting in 573 days of consumption data for each zone. Due to the presence of extreme values in most zones, however, it was difficult to see any trend in the data, and the daily consumption was heavily skewed to the right. Figure 2 and 3 show the daily consumption trend and distribution in the Lower pressure zone as an example.

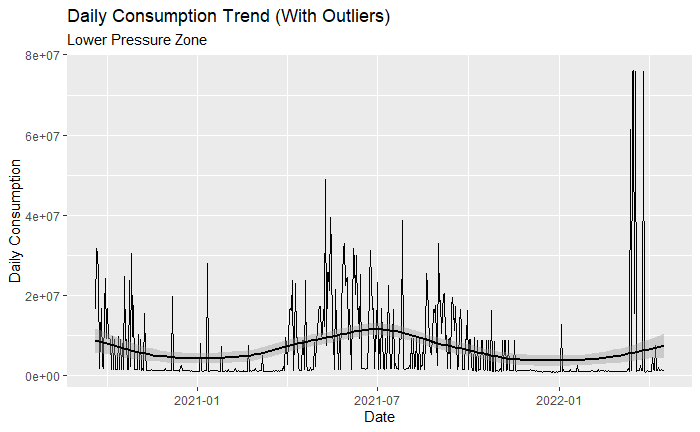


Figure 2. 573 days of daily water consumption at Lower Pressure Zone with extreme outliers present. geom\_smooth was applied to smooth the trend.

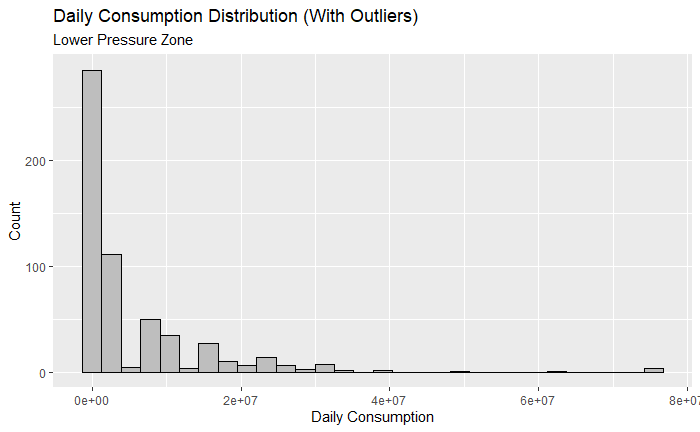


Figure 3. Distribution of 573 days of daily water consumption at Lower Pressure Zone with extreme outliers present.

If these outliers were not errors in the data, removing them would not help increasing the predictive power of models, and we could have used a more robust approach that weights extreme values to a lower degree. For example, instead of relying on reducing MSE (Mean Squared Error) that squares the residuals and enlarges the influence of extreme values, we could have employed MAE (Mean Absolute Error) that lowers their influence due to the lack of squaring.

Upon further investigation with the District, many of the extreme *Consumption* values appeared to be customer meter reading or reporting failures. These erroneous outliers had some common characteristics as well, such as a round number like 100,000 or a same account/meter number. Table 1 provides notes by the District on some of the accounts having these errors.

Table 1. Notes on accounts with extreme values

| Account Number | Notes: |
| --- | --- |
| 0036-0098-00 | This is a 2-inch irrigation meeting for a condo complex and there is no way it accounts for 40% of flow. It is possible that there was a meter swap out that skewed the data. |
| 0123-1141-05 | This account has a bill correction in December of 2021 for erroneous meter reads. |
| 0042-0277-04 | This account had a bad read in June 2021 that erroneously reported very high usage. |
| 0143-0191-00 | This site had a large water leak in February 2022 that does not represent normal usage. |
| 0026-0171-02 | This meter was swapped out in March 2022 and may have bad data that reports high usage. |
| 0111-0272-00 | This account had a voided bill in December 2021 that had a meter failure. |
| 0123-0693-08 | This account had a voided bill in December 2021 that had a meter failure. |
| 0044-0171-04 | This account shows a water use adjustment is June 2021 for a meter swap out. |

There were also outliers that seemed to be non-errors due to accounts with extremely high but irregular consumption profiles. Patton State Hospital (0032-1462-00) and the gaming center of San Manuel Band of Mission Indians (0132-0248-01, 0115-0205-00, 0115-0201-00, and 0021-0210-00) were some of them. While we attempted to identify all of the erroneous outliers from non-errors and eliminate them, given our lack of knowledge about specific accounts, it was a too time-consuming task especially with the limited project timeline.

Therefore, we decided to remove observations with Consumption in the top 1% percentile in its *Zone* group since each *Zone* has different levels of daily consumption and those that deviate significantly from the rest could be errors. Although only 1% or 86,066 observations were eliminated, they accounted for 15,482,514,981 gallons or around 78.5% of the total water consumption across all the zones. This meant that our predictive models built on the data without outliers would significantly underestimate the future water demand in return for a clearer trend and a better fit. Another approach we could take here was to group observations by *AccountNumber* or *MeterNumber* and remove those in the top percentile. However, there were a significant number of accounts/meters with infrequent usage such that they had less than 5 observations during the entire 574 days, which made it impractical to use this approach.

After removing the outliers, the daily consumption data set had 8,520,130 observations totalling 4,249,722,667 gallons of water. They were grouped by *Date* and *Zone* again to calculate the daily consumption with those on 2020-09-20 being removed due to having only 4 zones, leading to 573 days of consumption data for each zone. The summary statistics of *Consumption* for all 12 zones are shown below in Table 2.

Table 2. Summary statistics of daily water consumption in each zone

| Zone | Min | Mean | Median | Max |
| --- | --- | --- | --- | --- |
| Upper | 1333416 | 2711049 | 2643450 | 3944407 |
| Intermediate | 1037658 | 1798252 | 1759331 | 2629231 |
| Foothill | 528086.1 | 1146640 | 1109362 | 1852914 |
| Lower | 651175 | 1016701 | 987280.3 | 1477682 |
| Canal 3 | 205191.9 | 504423.5 | 487424.1 | 810679 |
| Mountain | 36128.66 | 100752.3 | 97212.79 | 179842.5 |
| Canal 2 | 23656.02 | 44415.93 | 43323.58 | 72985.93 |
| 149 Hydro | 8474.98 | 34813.82 | 34175.8 | 68331.7 |
| 59 Hydro | 5789.324 | 27528.64 | 27101.1 | 49669.75 |
| Canal 1 | 2286.944 | 14312.8 | 13823.7 | 30963.07 |
| 34 Hydro | 3478.666 | 9227.217 | 8804.197 | 17443.67 |
| 101 Hydro | 3025.272 | 8437.275 | 8181.145 | 15225.62 |

Without the presence of extreme values, there were visible trends in most zones, and their *Consumption* values were closer to normal distribution than before. Moreover, during the summer days, the daily consumption rose while it fell during the winter days, which coincided with our expectation. There also appeared to be regular, periodic increases in consumption, perhaps due to some large consumers using water on a periodic basis. Figure 4 and 5 show the daily consumption trend and distribution in the Lower pressure zone after cleaning the outliers as an example. The figures for other pressure zones are listed in Appendix.

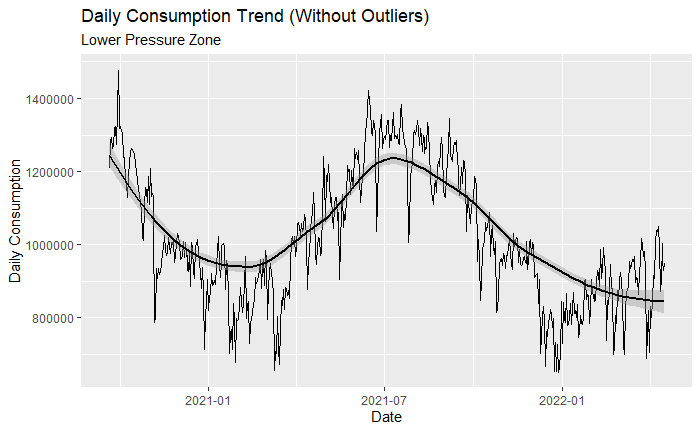


Figure 4. 573 days of daily water consumption at Lower Pressure Zone without extreme outliers present. geom\_smooth was applied to smooth the trend.

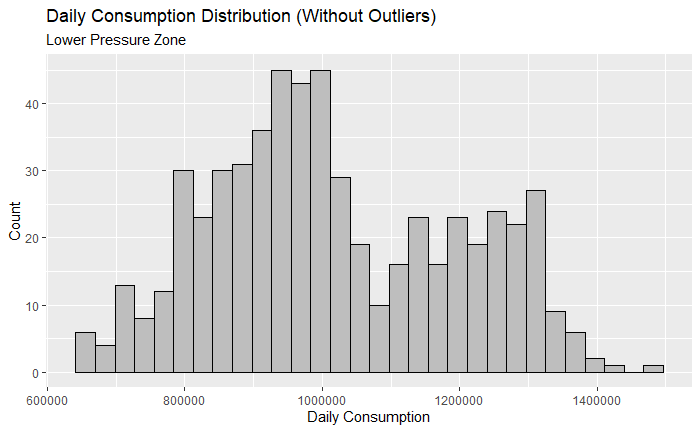


Figure 5. Distribution of 573 days of daily water consumption at Lower Pressure Zone without extreme outliers present.

**Variables Creation:**

Given the periodicity observed in the daily consumption trend and the expected consumer behavior, it was reasonable to assume that the daily consumption of water varies across 7 days of the week as well as during public holidays in CA. To capture these differences, day dummy variables, *Monday* to *Sunday* and *Holiday*, were created as binary variables from Date in the data set. Also, lagged variables on Consumption were created to not only adjust for the periodicity but also capture both the fixed differences – that is, relatively constant consumption profiles such as percentage of residential consumers, population, or income level – between each zone and the time order dependence between the daily consumption levels. These variables were lagged until 28 days before, and, therefore, the initial 28 days (2020-09-21 to 2020-10-18) were removed from the data set, resulting in 545 daily observations for each zone.

Finally, the daily consumption data frame with the new lagged and dummy variables were merged with “weather\_df” to create the completed data set. To facilitate zone-by-zone prediction modeling, it was separated into 12 different data frames based on pressure zones with *Dates* removed. Since they needed to be transferred from RStudio to VS Code for modeling and user interface, these data frames were exported to 12 new XLSX workbooks. Table 2 shows the list of 25 variables selected and created at the end.

Table 2. Final variables selected and created for every pressure zone

| Lagged Consumption (Continuous) | Day in Week (Binary) | Weather Condition (Continuous) |
| --- | --- | --- |
| Consumption | Monday | Max Air Temp (F) |
| Consumption\_t1 | Tuesday | Min Air Temp (F) |
| Consumption\_t2 | Wednesday | Max Rel Hum (F) |
| Consumption\_t3 | Thursday | Min Rel Hum (F) |
| Consumption\_t4 | Friday | Precip (in) |
| Consumption\_t5 | Saturday | Avg Wind Speed (mph) |
| Consumption\_t6 | Sunday |  |
| Consumption\_t7 | Holiday |  |
| Consumption\_t14 |  |  |
| Consumption\_t21 |  |  |
| Consumption\_t28 |  |  |

Data for Cost Optimization

For the cost optimization model, we relied on two datasets: the East Valley Water District 2019 Master Plan and the Southern California Edison Hydraulic Cost Analysis Summary

**Demand Prediction Modeling Section:**

After the data wrangling was finished, we started exploring different models that we could use to predict the next day's water demand for each zone. The tested models were machine learning classification and regression models. 2 supervised learning algorithms were selected for classification approach, while 8 supervised learning algorithms and 1 unsupervised learning algorithm were selected for regression approach. They were chosen based on our technical knowledge as well as how effective and widely used they are in the data science community. Since the measures of predictive accuracy are different for classification (accuracy) versus regression (RMSE, MAE, etc.), we compared the models within their approach.

When building a predictive model from sample data, it is common practice to divide the data into the training set and the testing set to avoid overfitting and evaluate the model’s accuracy. Empirical studies show that the best results are obtained if we use 20-30\% of the data for testing, and the remaining 70-80\% of the data for training. Therefore, for both classification and regression approaches, each zone’s data set was randomly split into 80\% training set and 20\% testing set. Our models were first trained on the training set with 10-fold cross-validation to find the best settings or hyperparameters for estimators, and then the data from the testing set was used to gauge the accuracy of the resulting model. We also considered Repeated K-Fold Cross-Validation for better performance measure, but this approach was too computationally expensive.

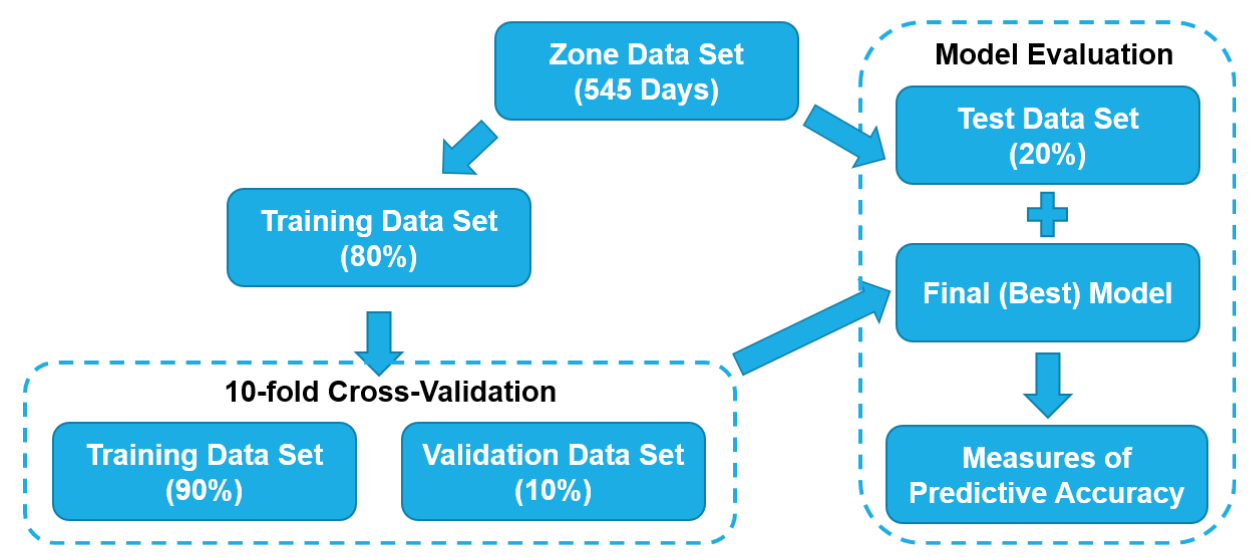
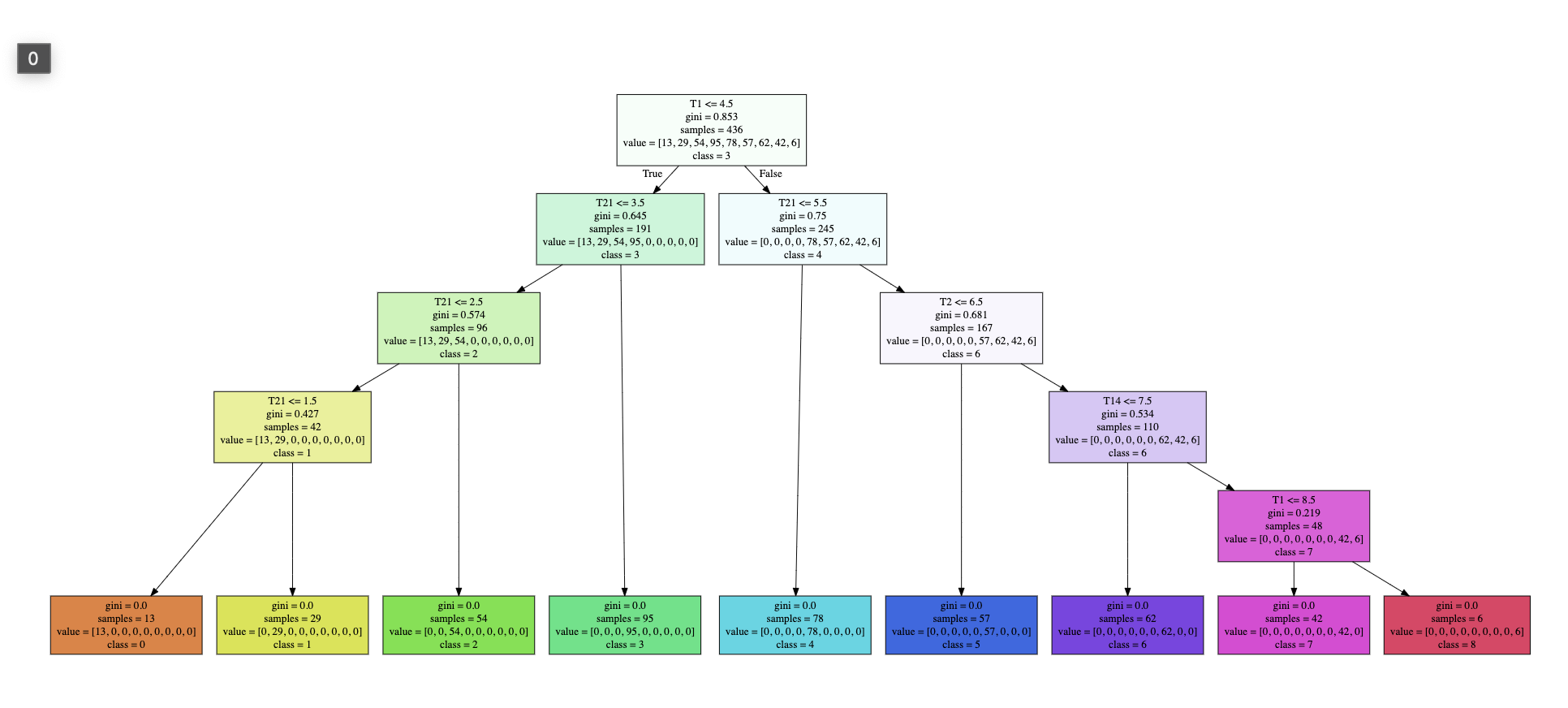


Figure 6. Validation and evaluation set-up for predictive models

**Classification Approach:**

Two different classification models based on K-Nearest Neighbors and Random Forest were trained and tested in Python, but we settled on the Random Forest model for user interface because it was relatively fast to train, and its accuracy rate was higher. Random Forest is a supervised machine learning algorithm that uses ensemble learning to create multiple decision trees on different subsets of the input data set and then aggregates them to create the best decision tree possible. Figure 7. Decision Tree for 59 Hydro Pressure Zone

In order to train a Random Forest classification model, the predictive variable in the training data set must be categorical since the output of the model is the name of the category that the data point passed in by the user falls into. Because of this, our first step was to write a python script that would convert the continuous consumption data into categorical data and then save the updated data frame. The Jupyter notebook with this code is called CreatingCategoricalConsumptionBins.ipynb, and it is saved in the DataCleaningFiles folder in the project’s GitHub repository. When running the script the user can input the desired bin size that the consumption values should be broken into. To determine the optimal bin size for each zone we tried to keep bin sizes as small as possible while keeping the accuracy of our models high. The size of the bins that we chose for each zone are documented in the appendix.

After the data sets for each zone had been updated, we used the code in the RandomForestByZone.ipynb file located in the ModelingFiles folder of the GitHub repository to train the model for each of the twelve zones. There are five main steps of the model training process, and we have included code snippets of each one below:

1. The first main step is to break up the full data set into four new data sets. X\_train and y\_train will contain 80% of the data points and will be used to train the model. X\_test and y\_test will contain 20% and will be used to test the accuracy of the model.

# Separate data into test data and training data

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_all, y\_all, test\_size=0.2, random\_state=42)

print(f"training with {len(y\_train)} rows; testing with {len(y\_test)} rows\n" )

1. Next, we use the training data sets to train and build 20 different decision trees of different depths and then check the average accuracy of each one. This process is called cross validation; it allows us to determine the best depth of the optimal decision tree that will result in the highest average classification accuracy rate.

# Use cross validation to compare different tree-depths

from sklearn.model\_selection import cross\_val\_score

from sklearn import tree

best\_d = 1

best\_accuracy = 0.0

for d in range(1,20):

cv\_model = tree.DecisionTreeClassifier(max\_depth=d)

cv\_scores = cross\_val\_score( cv\_model, X\_train, y\_train, cv=5 ) # 5 means 80/20 split

average\_cv\_accuracy = cv\_scores.mean()

print(f"depth: {d:2d} cv accuracy: {average\_cv\_accuracy:7.4f}")

if average\_cv\_accuracy > best\_accuracy:

best\_accuracy = average\_cv\_accuracy

best\_d = d

# assign best value of d to best\_depth

best\_depth = best\_d

print()

print(f"best\_depth = {best\_depth} is our choice for an underfitting/overfitting balance.")

1. Once we have found the best depth using cross validation, we use this value to train our final Random Forest Decision Tree Classifier model

# Use the best Depth to build a new model

from sklearn import tree

dtree\_model\_final = tree.DecisionTreeClassifier(max\_depth=best\_depth)

dtree\_model\_final.fit(X\_train, y\_train)

print("Created and trained a DT classifier with max depth =", best\_depth)

1. Finally, we print and save the list of feature (variable) importance in order to gain insights into which variables are most significant in helping the model determine how to classify a data point.

# Print Feature Importances

print(dtree\_model\_final.feature\_importances\_)

IMPs = dtree\_model\_final.feature\_importances\_

for i, importance in enumerate(IMPs):

perc = importance\*100

print(f"Feature {COLUMNS[i]:>12s} has {perc:>7.2f}% of the decision-making importance.")

1. Finally, save the final trained model as a compressed pickle file so that we can import it into our User Interface’s backend code file and then use the model to predict the water consumption of the zone in our finalized tool.

# Create a pickle file of the model

pickle.dump(dtree\_model\_final, open("MountainModel.pkl", "wb"))

We used the approach detailed above to train 12 random forest models for each zone. The accuracy results, feature importances, and image of the final decision tree created by the mode for each zone are all included in the Appendix. One notable aspect of our results is that all of our final models have an accuracy rate of about 99%. At first we were concerned that the decision trees had depths that were too high, leading to overfitting of the models. However, after looking at the feature importances, we noticed that the weather variables were features with very low importance in our models. Most of the weather importances were 0%, so most of the decision making process in our final models is driven by the five previous consumption features that we used to train the model: *Consumption\_t1, Consumption\_t2, Consumption\_t7, Consumption\_t14, and Consumption\_t21*. It turns out that the five previous consumption values tend to fall into the same bin or bins that are very close to each other. Though there is variation in the number of gallons that a zone uses each day, the values tend to not fluctuate too drastically so they are caught in the same bin. If the previous consumption values all fall into the same bin, there is a high chance that the next day consumption will too, which is why when we tested our models, the accuracies were quite high.

Although our models can accurately predict the bin that the next day’s consumption will fall into, it is likely that our final predictions for each zone are rough estimates of the actual values. This is because if the bin size of a zone is 10,000 and the model predicts that the next day consumption will fall in bin 3, the consumption value will be between 30,000 and 40,000 gallons. We chose to return the value at the highest end of the possible range as our prediction to compensate for removing extreme values from the dataset during cleaning.

The other machine learning model that we tested was a K-Nearest Neighbors model. K-Nearest Neighbors is a supervised machine learning algorithm that uses labeled input data to produce a function that returns the correct output when given a new set of unlabeled data. The KNN algorithm looks at the clusters of similar data points in the training set to find patterns and determine how to create the prediction function.

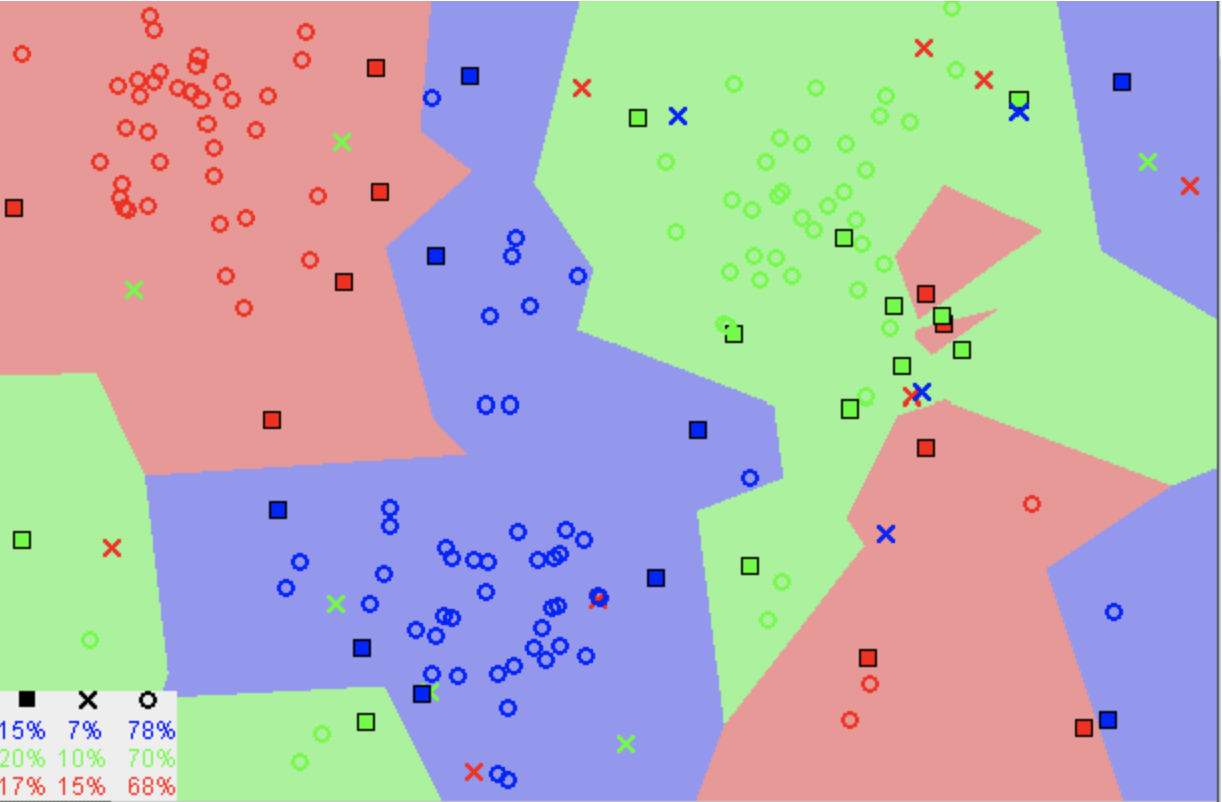


Figure 8. Example visualization of data point cluster

Before deciding to create separate water demand predictions for each zone, we considered creating a model that would use account-level data to predict the water consumption for each user in the water district and then aggregate all of the individual water consumptions in each zone to determine the water demand prediction for the whole zone. Because of this, we trained the KNN model to predict the Individual Consumption amount using a different set of variables from the original dataset: *Holiday, ETo (in), Precipitation, Sol Rad (Ly/day), Avg Vap Pres (mBars), Max Air Temp (F), Min Air Temp (F), Avg Air Temp (F), Max Rel Hum (%), Min Rel Hum (%), Avg Rel Hum(%), Dew Point (F), Avg Wind Speed (mph), Wind Run (miles), Avg Soil Temp (F), DayofWeek, Zone Number, Account Number*. The bin size used was 20 gallons.

When training the model, we used cross validation to determine that the KNN Model had the highest accuracy when the number of nearby data points that were used to make a new prediction was 12. By using a best k of 12, the model’s best accuracy rate was 68.82%. This accuracy rate was not as high as we had hoped, which is why we transitioned to using a Random Forest Classification Model instead. Using Random Forest we were able to get much higher accuracy rates, but the variables that we trained with changed, and we trained the random forest model with consumption data aggregated by zone level. Therefore, we are unsure if the higher accuracy is due to the Random Forest Modeling approach being more accurate, or if the final data prediction variables that we chose do a much better job at explaining the variation in the data and therefore lead to more accurate predictions. If in the future another team trains water demand predictions with new data, they should train a KNN model and Random Forest Model using the same data set to be able to do a fair comparison of the two models.

**Regression Approach:**

7 different machine learning regression models based on Support Vector Machine (SVM), XGBoost, Gradient Boosting Machine (GBM), Random Forest, Stepwise, Elastic Net, and Principal Component Analysis (PCA) were used. All of the algorithms except PCA are supervised learning models, and all of them are non-parametric models where the number of parameters varies with respect to the sample size. For each of the 12 pressure zones, 7 models were trained and tested according to the abovementioned validation and evaluation procedures. SVM, XGBoost, GBM, and Random Forest had many hyperparameters to be tuned using a grid search, so they were computationally more intensive although SVM was the fastest among them.

The first machine learning model we developed before other models was Stepwise Regression, which is step-by-step iterative construction of a multivariable regression model that adds or removes potential explanatory variables in succession and evaluates statistical significance after each iteration. It combines both forward selection, trying out one independent variable at a time and including it in the regression model if it is statistically significant, and backward elimination, including all potential independent variables in the model and eliminating those that are not statistically significant. While Stepwise can be less predictive and prone to overfitting, its final model is easier to interpret than those of other ML models and allows researchers to see the size and direction of the impact of selected variables. For example, Table 6 shows the stepwise regression results for Foothill and Upper zone. In both, *Precip (in)* has a negative impact on *Consumption*, while *Max Air Temp (F)* has a positive impact, and their coefficients are statistically significant at 1% level. This agreed with our expectation that higher rain level would reduce the water demand, and higher temperature would increase it. Indeed, all of the weather variables were significant for these zones, refuting the earlier finding from the Random Forest classification model.

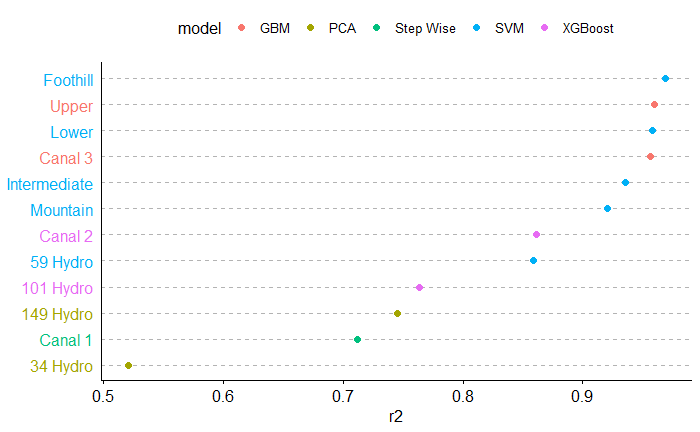
The best-performing model with the least RMSE for each pressure zone is displayed in Figure 9 along with its R-squared, which measures the share of variations in the test set explained by the model. 

Figure 9. Best-performing ML regression model with the least RMSE for each pressure zone

Most advanced and widely used machine learning models for prediction, SVM, GBM, and XGBoost, outperformed others in large and medium-sized zones, namely Upper, Intermediate, Foothill, Lower, Canal 3, Mountain, and Canal 2. Among the three, the clear winner was SVM regression. Unlike other regression models that aim to minimize the error between the actual and predicted value, it tries to fit the best line or the hyperplane in n-dimensional space that distinctively classifies data points within a threshold value. However, these advanced models seemed to underperform compared to rather rudimentary models, PCA and Stepwise, in small zones, such as Canal 1 and 34 Hydro. Figure 10 and 11 illustrate the difference in the predictive power of SVM between Foothill and 34 Hydro. It should be noted that more training and testing with larger data sets need to be done to not only increase the predictive accuracy but also effectively compare different regression models. Table 7 summarizes the testing results of the seven regression models for each zone.

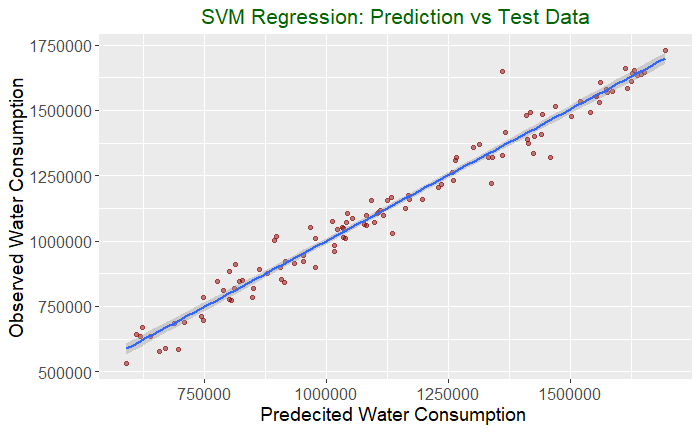


Figure 10. Predicted versus actual test consumption values for Foothill zone using SVM. Blue line represents the actual values while the red dots are predictions

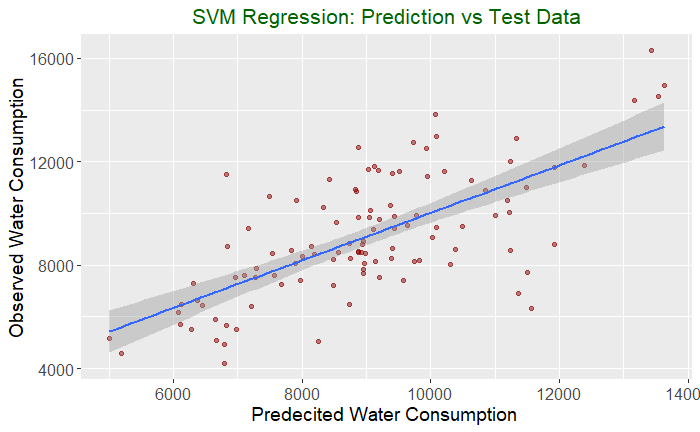


Figure 11. Predicted versus actual test consumption values for 34 Hydro zone using SVM. Blue line represents the actual values while the red dots are predictions

**User Interface Technical Section:**

After we trained both the Water Demand Prediction Models and the Cost Minimization Models, our final products were multiple python files. A user who is familiar with python and using the terminal could access the demand prediction code files in our code repository, run the files and input values using the terminal, and then get and save the results from the terminal. However, most operators at EVWD do not have the necessary coding experience and knowledge so this process may have been inaccessible to them. To overcome this issue we hoped to connect our models to an Excel spreadsheet interface where an operator could input the necessary values and the predictions/calculations would take place in the backend. This was not possible due to two reasons. First, Excel was not able to handle all of the inputs and constraints of the Cost Minimization Model and, second, the plug-ins required to connect python models to Excel were difficult to work with. Due to these challenges, we decided to build our own small full stack web application that has a backend with the models connected to an easy to use interface front-end where operators can input values and see the results.

Since the model files were written in Python we chose to use Flask as our Web Framework. Flask allows you to quickly build small full-stack web applications using primarily python so it met our needs and requirements for this project quite well. For the backend portion of the web application we primarily used python and a couple of Jinja templates to add some coding logic in our HTML files. To build the pages displayed in the frontend we used HTML to structure the text, inputs, and outputs on our pages and CSS to style all aesthetic aspects of the pages such as color, font size, centering, margins, the nav bar, etc..

There are two files that make up the backend portion of our code. The primary one is called webapp.py and in it we define the Flask Application, import the twelve random forest models, and include the code functions that generate prediction results using the models and then render each page. For each page in the web application, the user must define a route name and create a function that takes in user inputs, uses the inputs to generate prediction results, structures the results, and passes them into the correct HTML page by rendering the correct template. We have included the code for one of our simpler pages below as an example of how pages are defined in the backend:

@app.route("/costoptimizationresults", methods = ["POST"])

def costoptimizationresults():

# get the inputs from the users

user\_inputs = [float(x) for x in request.form.values()]

# Pass the user inputs into the linear programing model function and get back the dataframe with the solutions

model\_output = linear\_programming\_model(user\_inputs)

data\_frame = model\_output[0]

final\_cost = model\_output[1]

# Create the list of headers for the table

headers = ["Pump Name","Optimized Usage", "Max Capacity", "Percentage Input" ]

# Define the rows of data list, we will append the rows to this list

rows\_of\_data = []

# Iterate through rows of the data frame returned by the linear programming model, extract the info, save it in rows\_of\_data list

for index,row in data\_frame.iterrows():

pump = row["Pump Name"]

optimized\_usage = row["Optimized Usage"]

max\_capacity = row["Max Capacity"]

percentage\_input = row["Percentage Input"]

rows\_of\_data += [[pump, optimized\_usage, max\_capacity, percentage\_input]]

# Render the results page by passing in the name of the html file with front end code and the values that will be used to generate the table

return render\_template("resultspage.html",

headers = headers, rowsofdata= rows\_of\_data, finalcost = final\_cost, user\_inputs = user\_inputs

)

The second main file that is a part of the backend code is called evwdlinearprogramming.py. In this file the linear programming model that our team created is wrapped in a function that takes in user inputs and then returns the final data frame with the model's results and the final minimized cost. By wrapping the model in a function we are able to keep it in a separate file and then import it into the webapp.py file in one line of code (the asterisk means import all):

# Import the Linear Programming Model

from evwdlinearprogramming import \*

Next, each of the pages displayed to the user in the frontend has a corresponding HTML page in the templates folder in our GitHub repository. There are six HTML files (one for each page of the web application) and each has four main parts: the Head of the file defines the metadata needed, the Styles section includes all of the CSS code we use to style the page, at the start of the body section we define the navigation bar and the routes associated with each page, then in the rest of the body we define the text of the page and all of the input fields necessary.

We have included the code necessary to define the label and input field for one of the pumps in the cost minimization page below. Each of the inputs is defined using a similar code line which includes a class name that dictates which CSS styling is applied to the input field, the input type, a unique name (we chose name the inputs according to the order of appearance on the page), and a default value set to 1 which gets overridden using the onBlur function if the user inputs a different value in the field.

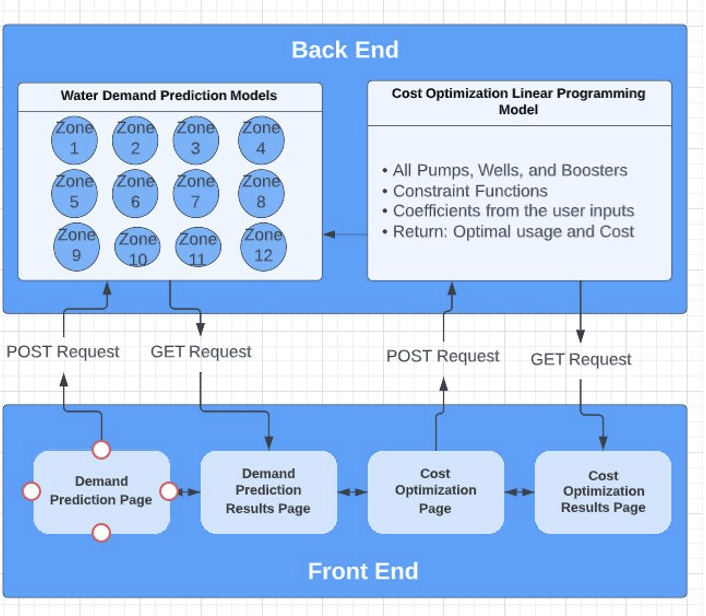
<label>P9 B1</label>

<input class = "costinput" type="text" required="required" name = "12" value = "1" onBlur="this.value=this.value==''

? 'default'

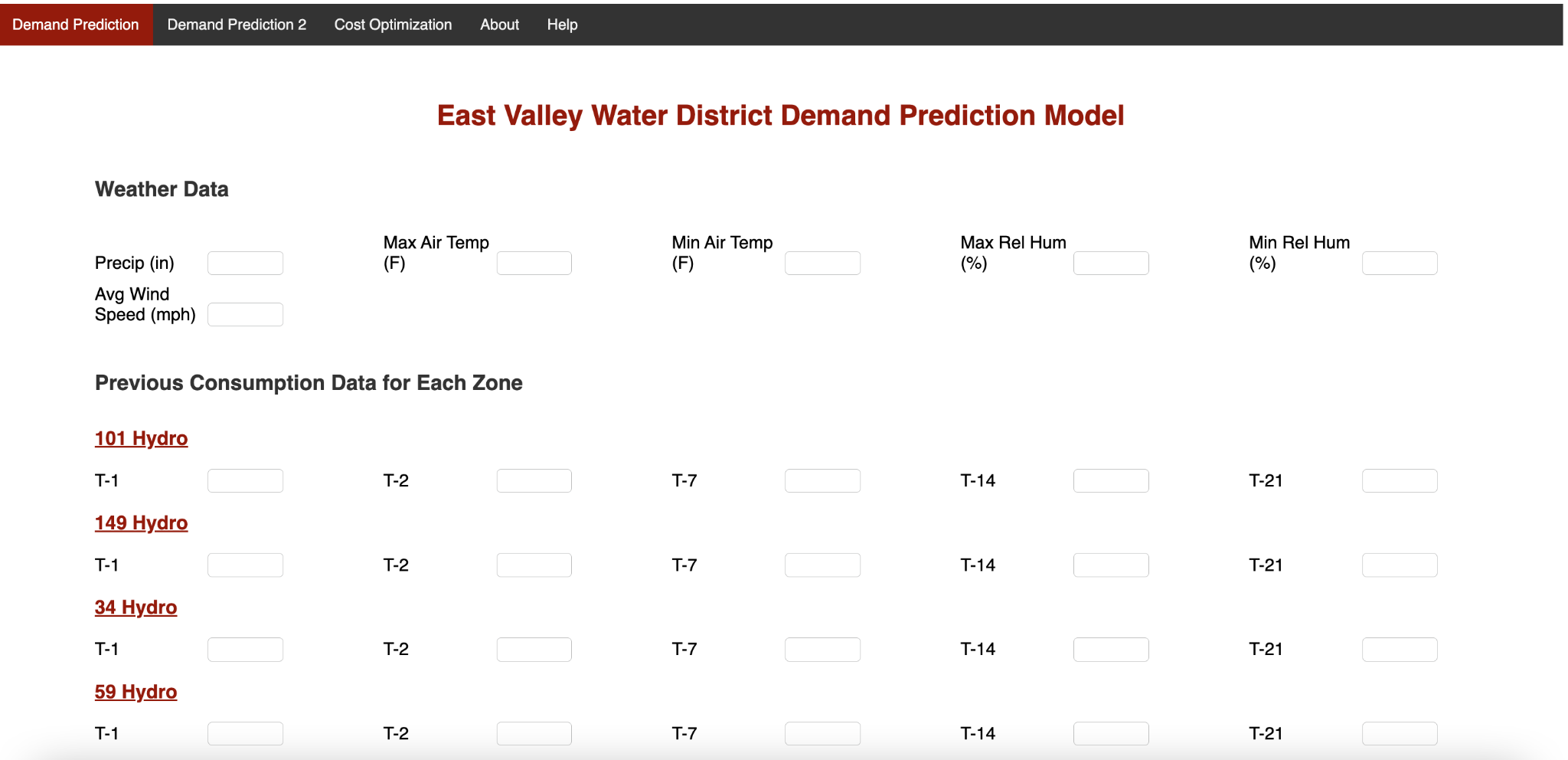
: this.value;"/>

Lastly, when building our full stack web application we prioritized using a modular software design approach to ensure that the code for the different functionalities of our web app is connected but separate. By splitting up the code into different files and functions it enables future students who continue working with this code base to only update and modify the sections they need knowing that their changes will not accidentally break other components of the application. In the diagram below, we have attempted to depict the organizational structure of the web application’s code.

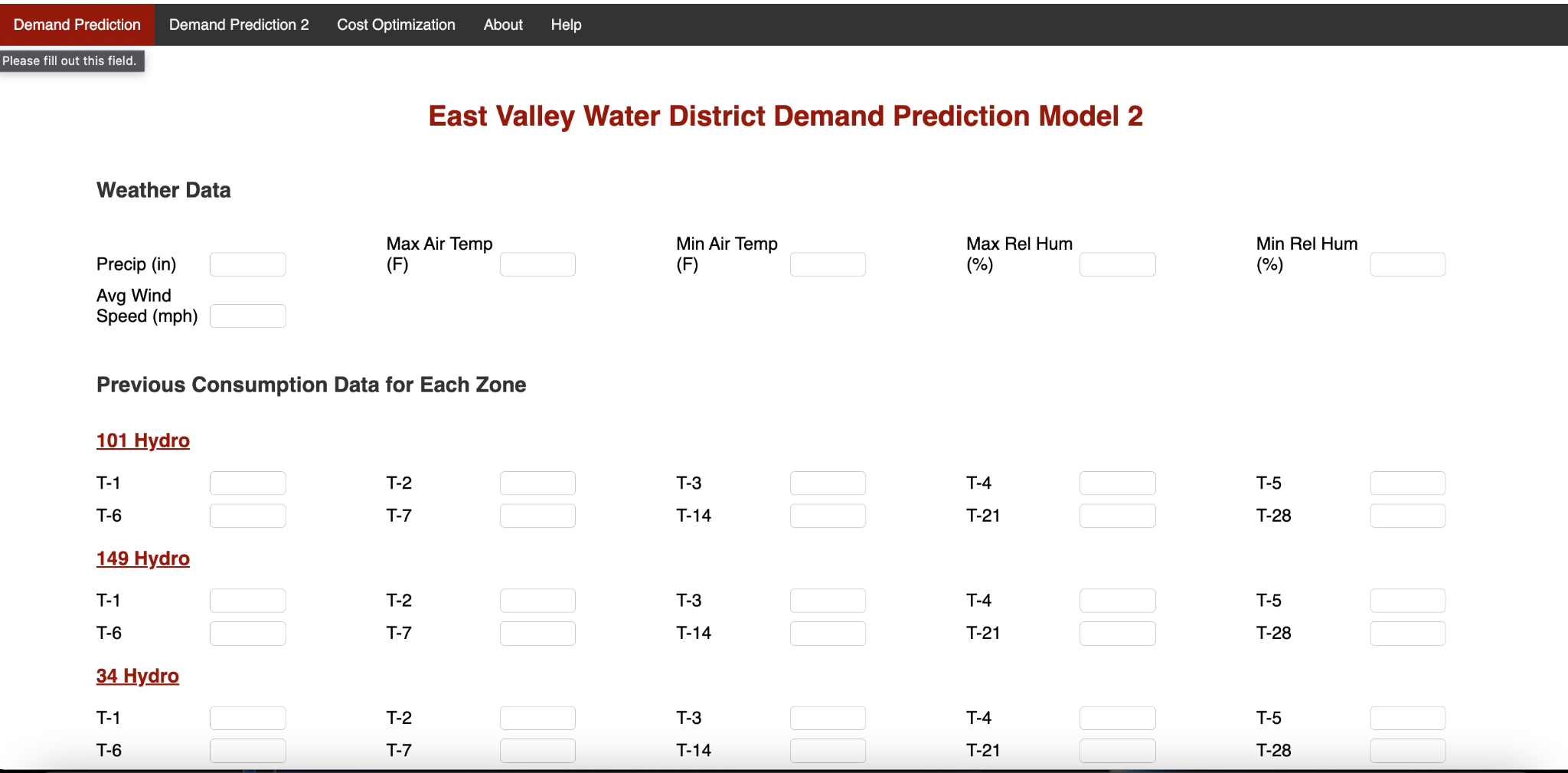


**User Interface Results Section**

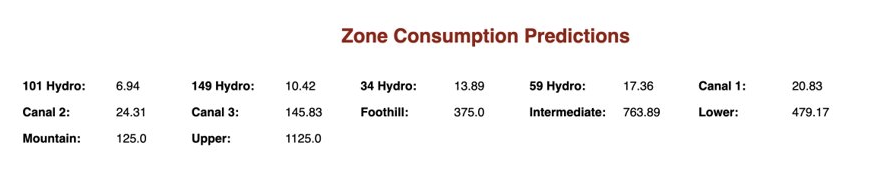
Random Forest Models Demand Prediction Page:



Regression Models Demand Prediction Page:

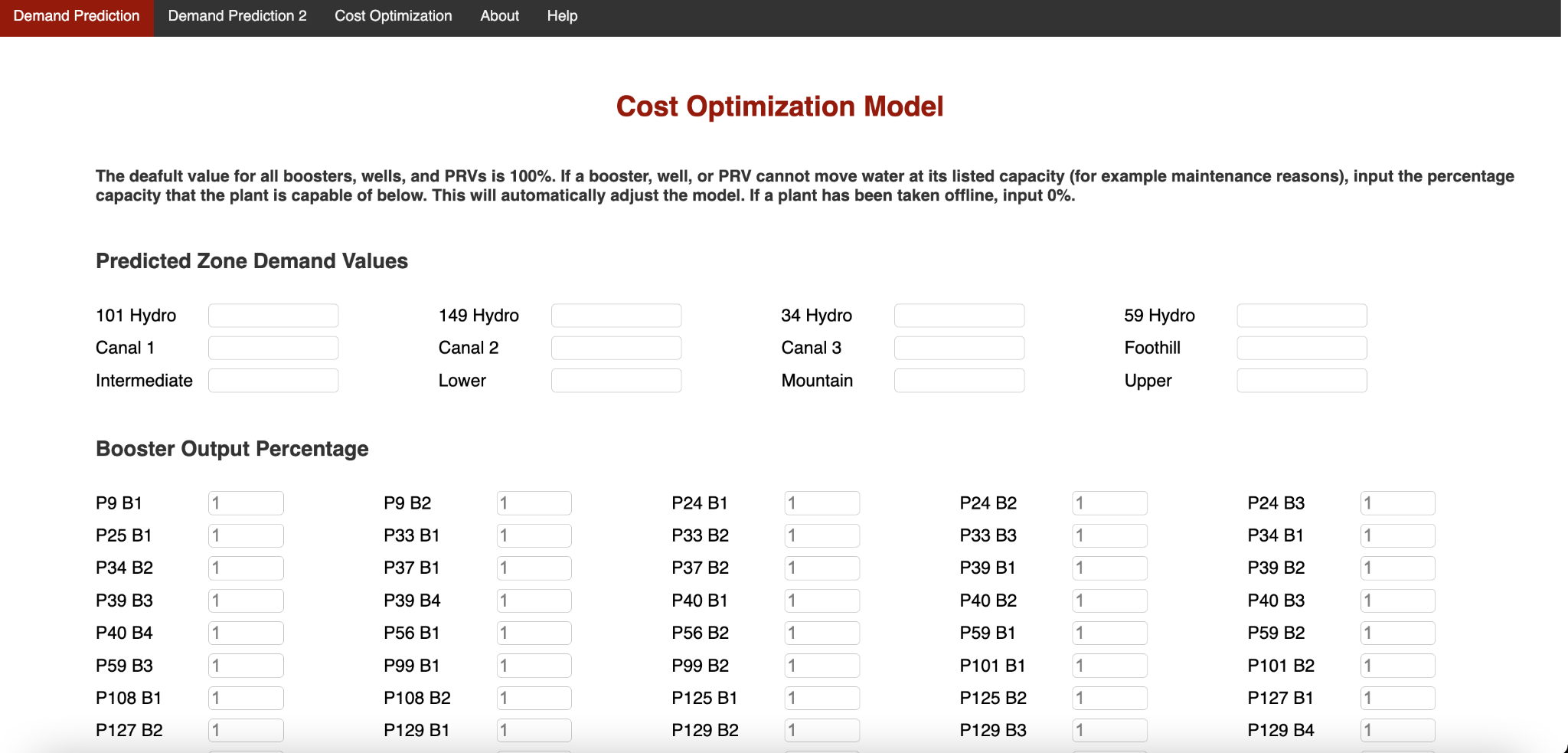


Demand Results:

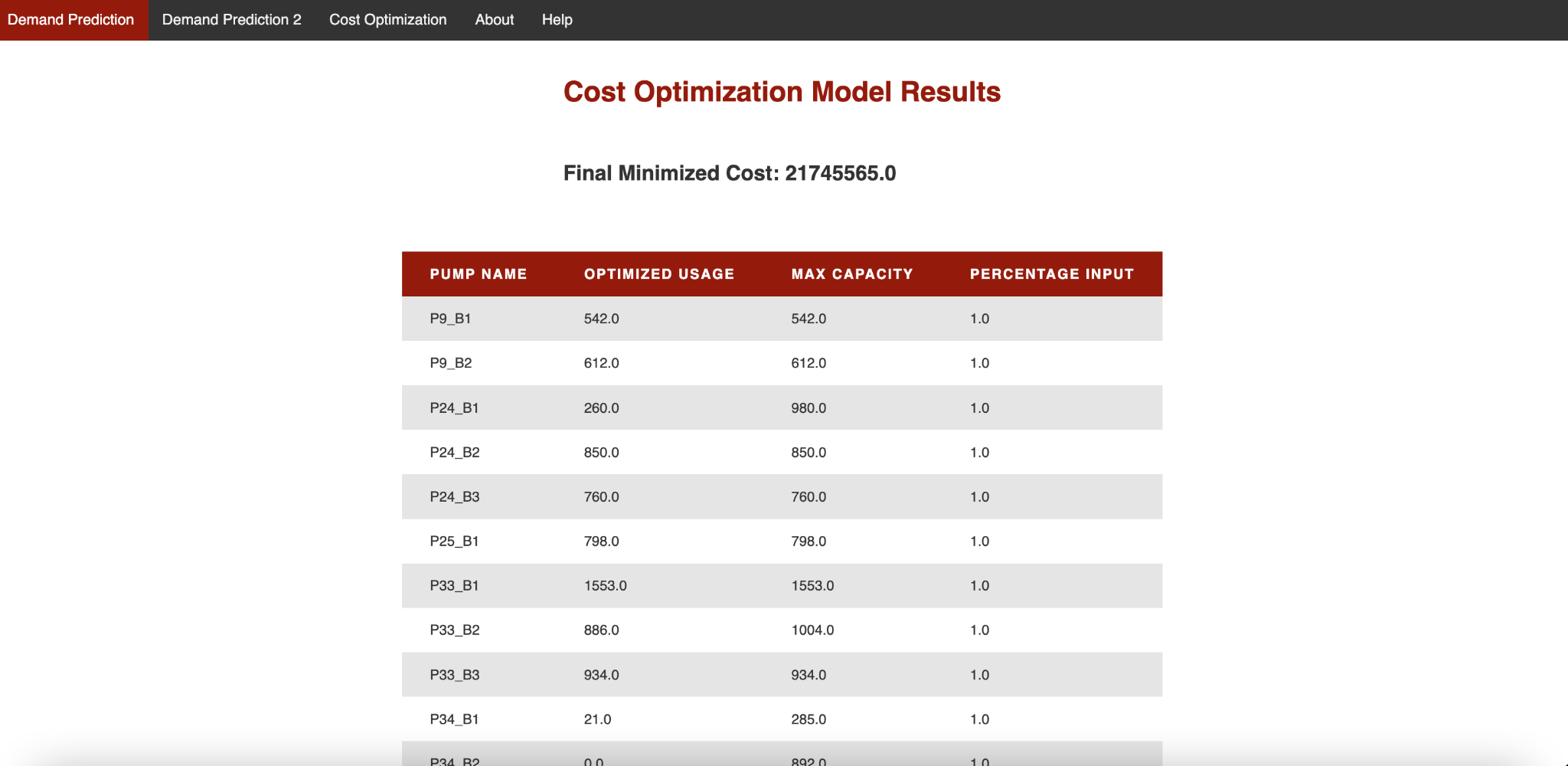


(Units are gallons/minute)

Cost Minimization Model Page:



Cost Minimization Results Page:



**Models**

* 1. Demand Prediction (Back end)
     1. Regression (Min)
     2. Classification (Jen)
  2. Cost Minimization (Back end)

The main objective of this portion of our project is to find a way to minimize the costs EVWD incurs when operating their existing water infrastructure. Each pump they utilize requires varying amounts of energy to function, so we can take this metric for cost (kilowatt-hours) and apply it to a cost-minimization model. Keeping in mind that each pump has its specific limitations, we want to find a way to optimize each pump’s output to lower EVWDs total pump operating costs. When encountered with a problem that involves a variety of resource constraints, linear programming can generate the best possible solutions.

* + 1. Linear Programming Model

Linear programming is a mathematical technique that data scientists can use to determine an optimal amount of resource usage. This process helps anyone make decisions about the most efficient ways to use limited resources such as time, labor, money, or machinery.

Typically, there are four key elements in the process of creating a linear programming model:

1. Determine the problem specific decision variables
2. Identify the objective function
3. State all relevant constraints
4. Mention explicit non-negativity constraint

Before formulating the problem, we will define each element of our situation and how it relates to the model. The **decision variables** are the variables that will decide our operating costs and they will represent our final results. The EVWD pumps will make up each of our decision variables and their respective pump outputs will be the values we will look to optimize. From this, we can attain our **objective function**: minimized cost as a function of each pump. The **relevant** **constraints** are the limitations and restrictions on the pump decision variables and they usually put a lower limit and/or an upper limit on the decision variables. Lastly, the **non-negativity constraint** is an important inclusion as we make sure each pump’s output has an additional lower bound of at least 0. For all linear program models, the decision variables should always produce non-negative values as their outputs.

* + 1. EVWD Constraints

For the purposes of our model, we must find the specific constraints that relate to the output of each individual pump. Firstly, each EVWD pump has a physical capacity for the amount of water they can hold which is measured in gallons per minute. The pump capacities are determined by efficiency testing and vary greatly depending on the size, location, and altitude of the pump, and they make up our first category of constraints. Additionally, the pressure reduction valves included in our model will have a capacity of roughly 1800 gallons per minute even though they are pipes instead of actual pumps.

The second category of constraints includes the specific zone demand constraints. As stated above, there are certain pumps that feed water into a zone, and certain pumps that pull water away from the same zone. Accounting for the exact inputs and outputs for each of the EVWD zones allows us to create separated inequality equations for the amount of water in each zone. The zone constraints will represent not only the exact pumps that correspond to a zone, but also the amount of water left in the zone in a day. The most important element of these inequalities are the bounds that each zone is restricted to. For EVWD to satisfy their customers' demands, we must make sure each zone’s water amount is greater than or equal to the amount of water demanded by the consumers in that zone.

\_\_\_\_\_\_ Zone Water Demand ≤ ​Well1 + Well2 + Booster1 ​- Booster2 - Booster3

Equation \_\_\_\_\_\_\_\_\_ above demonstrates a simple outline mapping the possible intake and removal of water in a zone. The water-demanded values will be extracted from our predictive demand model and used as inputs for the lower bound of each of the zone inequalities. EVWD’s zone areas and water usages have vastly different sizes, so a zone-by-zone analysis will allow us to compare the volume metrics and see interesting collated results. Using the information from the hydraulic map, we were able to construct the following constraints:

Lower Demand ≤ P11 W + P28 W - P34 B1 - P34 B2 - P127 B1 - P127 B2 - P130 B1 - P130 B2 + PRV127 + PRV309 + PRV311

Intermediate Demand ≤ P25 W + P39 W + P132 W + P141 W + P151 W + P24 W1 + P24 W2 + P9 B1 + P9 B2 + P127 B1 + P127 B2 + P130 B1 + P130 B2 - P25 B1- P33 B1 - P33 B2 - P33 B3 - P39 B1 - P39 B2 - P39 B3 - P39 B4 - P40 B1 - P40 B2 - P40 B3 - P40 B4 + PRV33 + PRV40 - PRV127 - PRV309 - PRV311 + PRV325 + PRV326​

Upper Demand ≤ P25 B1 + P33 B1 + P33 B2 + P33 B3 + P39 B1 + P40 B1 + P40 B2 + P40 B3 + P40 B4 + P143 B1 + P143 B2 + P143 B3 - P37 B1 - P37 B2 - P129 B1 - P129 B2 - P129 B3 - P129 B4 - P129 B5 - P134 B1 - P134 B2 - P134 B3 - P134 B4 - P134 B5 - P134 B6 - P134 B7 - P134 B8 - PRV33 - PRV40 + PRV305 + PRV324 - PRV325 - PRV326​

Foothill Demand ≤ P37 B1 + P37 B2 + P39 B2 + P39 B3 + P39 B4 + P125 B1 + P125 B2 + P129 B1 + P129 B2 + P129 B3 + P134 B1 + P134 B2 + P134 B3 + P134 B4 + P134 B5 + P142 B1 + P142 B2 - P56 B1 - P56 B2 - P99 B1 - P99 B2 - P108 B1 - P108 B2 - P131 B1 - P131 B2 - P131 B3 - PRV305 - PRV324​

Mountain Demand ≤ P137 B1 + P137 B2 + P140 B1 + P140 B2 - P149 B1 - P149 B2 - P149 B3 - P149 B4

Canal 1 ≤ P56 B1 + P56 B2 - P59 B1 - P59 B2 - P59 B3

Canal 2 ≤ P99 B1 + P99 B2 - P101 B1 - P101 B2 - P137 B1 - P137 B2 - P140 B1 - P140 B2

Canal 3 ≤ P108 B1 + P108 B2 + P129 B4 + P129 B5 + P131 B1 + P131 B2 + P131 B3 + P134 B6 + P134 B7 + P134 B8 + P142 B3

Mountain ≤ P137 B1 + P137 B2 + P140 B1 + P140 B2 - P149 B1 - P149 B2 - P149 B3 - P149 B4

Hydro34 ≤ P34 B1 + P34 B2

Hydro59 ≤ P59 B1 + P59 B2 + P59 B3

Hydro101 ≤ P101 B1 + P101 B2

Hydro149 ≤ P149 B1 + P149 B2 + P149 B3 + P149B4

* + 1. Adaptability

After receiving advice from our client liaison, Rocky Welborn, it became apparent that we needed to introduce an adaptive ability to our model to account for the issue of offline pumps. We were told that plants going under maintenance or facing other operative issues are common occurrences. If some plants are taken offline, the optimized flow routing and pump outputs should change to reflect a new optimal set of results that take into account the irregularity. We chose to incorporate this into our model through the individual pump constraints by multiplying each pump’s capacity by a numerical input reflective of their operative ability. If a pump is taken completely offline, then their capacity for the day should reflect a constraint of 0 gallons per minute. Taking this further, we can set the numerical input to represent a percentage value for the pump’s capacity if it is working at a capacity in between 0 and their maximum capacity. For example, a capacity constraint for Pump 9 Booster 1 would include a percentage value *P9\_B1P* to multiply against the maximum capacity, as in the following inequality:

P9\_B1 ≤ 542 gpm \* P9\_B1P

By doing this we will be able to account for changes to the infrastructure of EVWD’s plants, property, and equipment. While this will require an additional requirement that their Operations staff must handle, this addition makes our model adaptive to significant changes in the overall system. Even one pump’s altered maximum capacity may yield many subsequent changes to another pump’s final optimized output values in the same zone as well as other zones that are impacted. With this feature of our model, we ensure that EVWD will always get the most accurate optimization regardless of the pump functionality on that day.

* + 1. Platforms Used

After completing the process for setting up the linear programming problem, we needed to find a platform to perform the mathematical minimization. Initial research brought us to Microsoft Excel and their add-in software Solver which had all the features necessary to complete the modeling. Ultimately, we input 15 zone demand constraints, 98 capacity constraints for the boosters, wells, and PRVs, and 98 non-negativity constraints for each of the decision variables. This led us to our first major problem, the program was too large for Solver to compute on its own and Excel would crash when running it.

Although this realization came up late in the semester, we decided to transfer our model over to Python using Google Collab because we believed our model would work perfectly under the conditions we had set up, and because a user interface could easily accommodate the model in the later portion of our project. Fortunately, the transition proved fruitful as Python’s Pulp package enabled us to import all 211 constraints along with the 98 decision variables into a linear programming model with successful optimal outputs.

CODE:

Lp\_prob = p.LpProblem('kWhMinimization', p.LpMinimize)

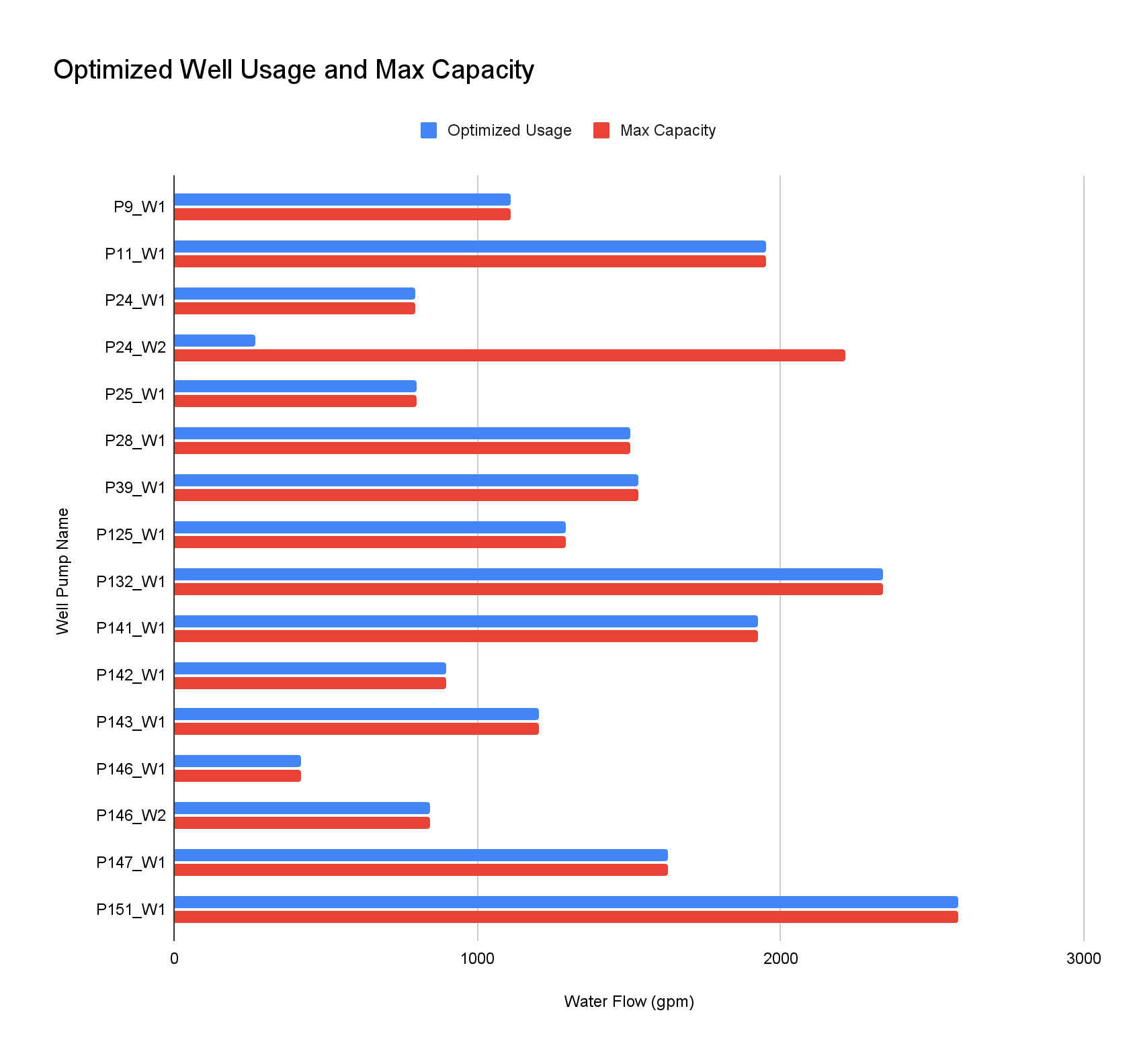
P9\_B1 = p.LpVariable(P9\_B1, lowBound = 0)

## Cost Minimization Results

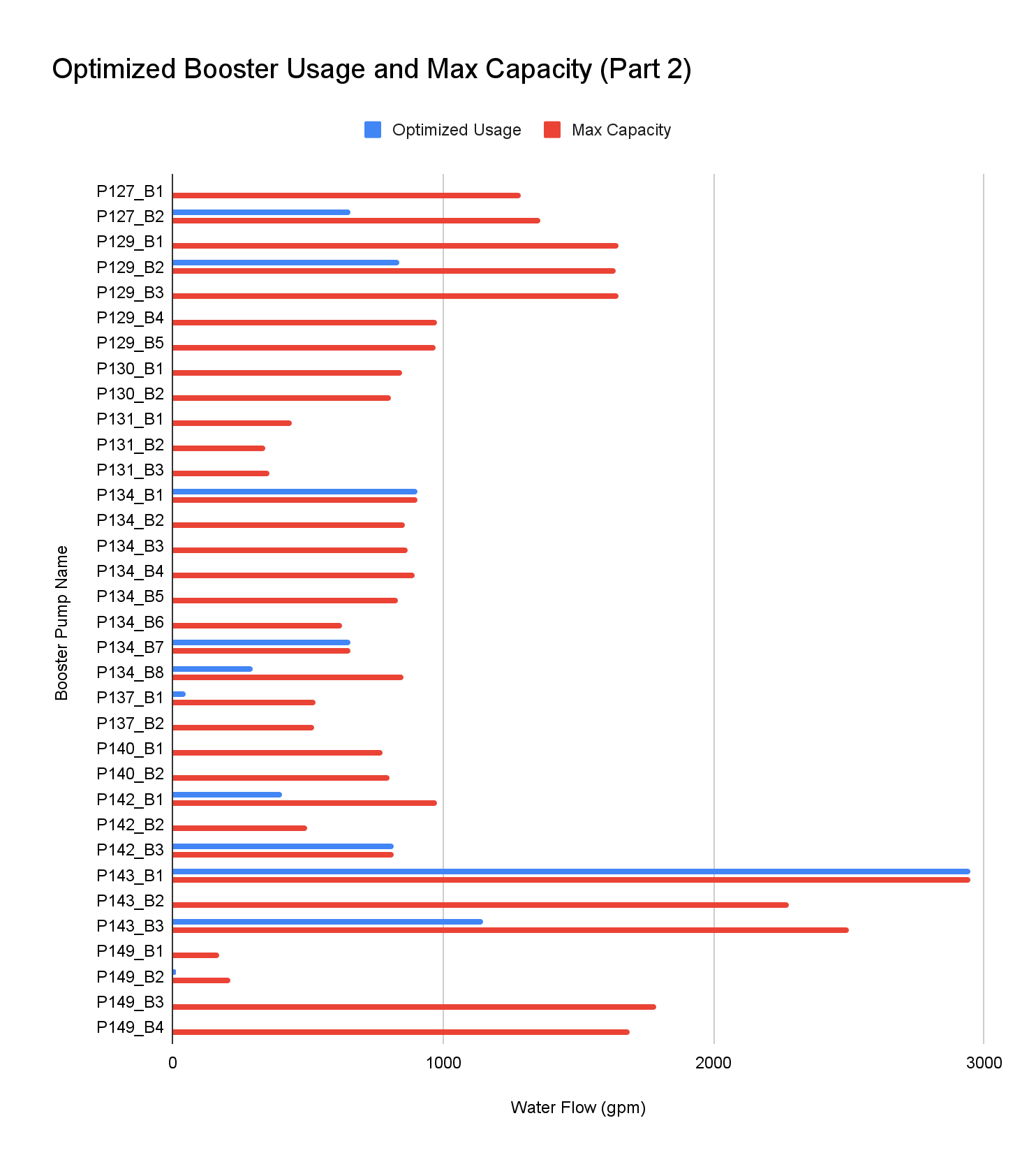
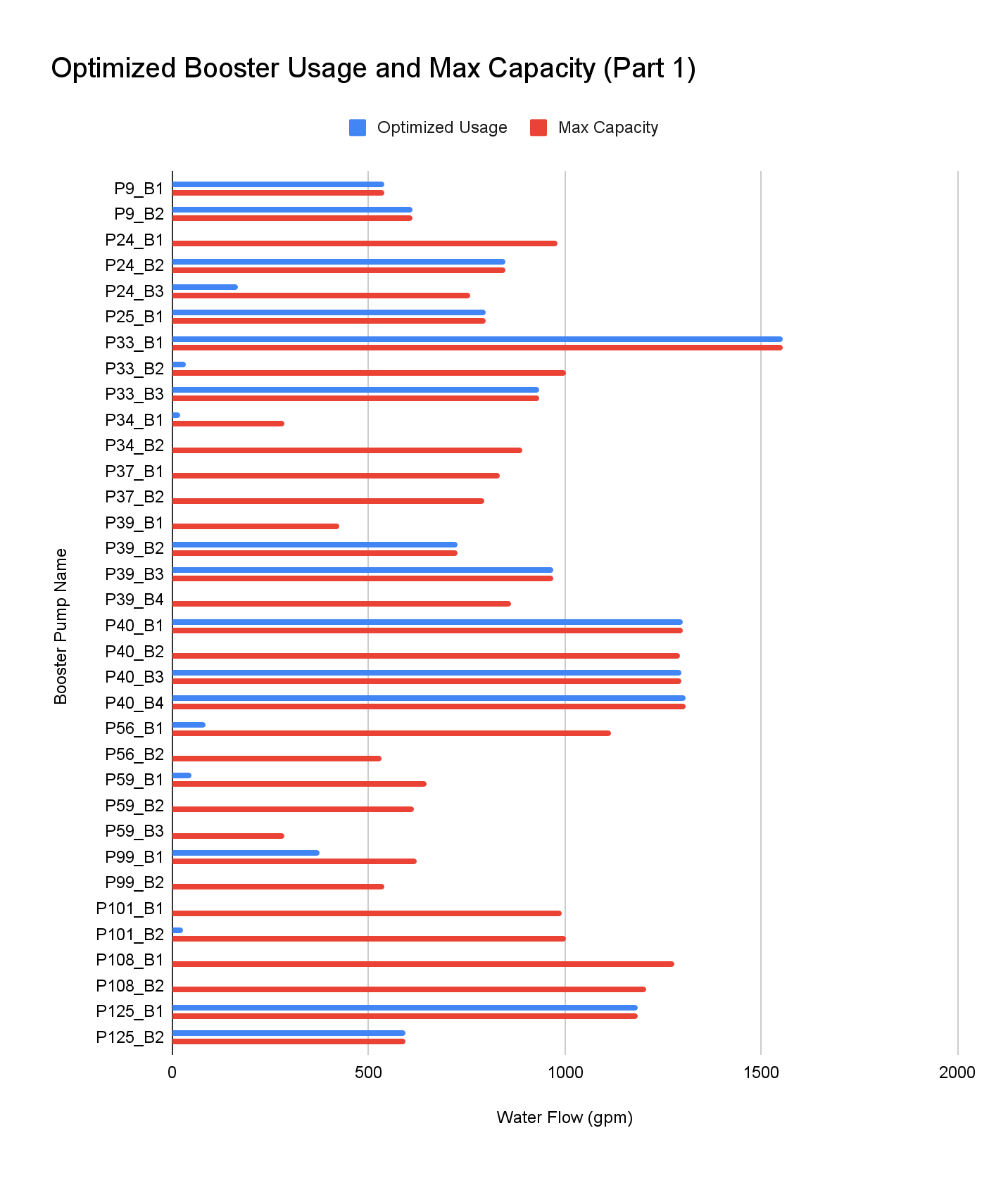
The results of our cost minimization model produced meaningful insights for the East Valley Water District. These insights change when parts of the District’s infrastructure are in maintenance. We produced results for both no maintenance and maintenance scenarios. First, we will discuss our results under the assumption of no maintenance.

## Cost Minimization Results: No Maintenance

Under the no maintenance situation, every well, booster, and PRV is capable of running at its full capacity. These conditions produced interesting results. First, they proved that the District’s infrastructure requires significant well production. Every well except for P24\_W2 is optimized to run at full capacity. This can be visualized in the below chart:



Second, our linear programming results proved that only 45.6% of boosters need to run. Specifically, 31 of the 68 total boosters need to run. It is also important to note that only 17 of the 31 boosters are running at their maximum capacity. The following two graphs visualize these results:



Third, our linear programming results proved that the PRVs are inefficient and do not need to be a part of daily operations. Under the no maintenance scenario, zero PRVs are in operation.

## Zone Analysis

Finally, it is also worthwhile considering how the above graphs apply to each specific demand zone. In particular, which wells and boosters are turned on and which wells and boosters are turned off in each zone. Below is an analysis, under the zero maintenance assumption, of what infrastructure can be turned off under the zero maintenance assumption. Infrastructure that is turned off contains a red strikethrough. The PRVs, which are turned off in every zone under this assumption, contain a yellow strikethrough.

Lower Demand ≤ P11 W + P28 W - P34 B1 - P34 B2 - ~~P127 B1~~ - P127 B2 - ~~P130 B1 - P130 B2 + PRV127 + PRV309 + PRV311~~ ​

Intermediate Demand ≤ P25 W + P39 W + P132 W + P141 W + P151 W + P24 W1 + P24 W2 + P9 B1 + P9 B2 + ~~P127 B1~~ + P127 B2 + ~~P130 B1 + P130 B2~~ - P25 B1- P33 B1 - P33 B2 - P33 B3 - ~~P39 B1~~ - P39 B2 - P39 B3 - ~~P39 B4~~ - P40 B1 - ~~P40 B2~~ - P40 B3 - P40 B4 ~~+ PRV33 + PRV40 - PRV127 - PRV309 - PRV311 + PRV325 + PRV326~~​

Upper Demand ≤ P25 B1 + P33 B1 + P33 B2 + P33 B3 + ~~P39 B1~~ + P40 B1 + ~~P40 B2~~ + P40 B3 + P40 B4 + P143 B1 + ~~P143 B2~~ + P143 B3 - ~~P37 B1 - P37 B2~~ - P129 B1 - P129 B2 - P129 B3 - P129 B4 - P129 B5 - P134 B1 - ~~P134 B2 - P134 B3 - P134 B4 - P134 B5 - P134 B6~~ - P134 B7 - P134 B8 ~~- PRV33 - PRV40 + PRV305 + PRV324 - PRV325 - PRV326~~​

Foothill Demand ≤ ~~P37 B1~~ + ~~P37 B2~~ + P39 B2 + P39 B3 + ~~P39 B4~~ + P125 B1 + P125 B2 + ~~P129 B1~~ + P129 B2 + ~~P129 B3~~ + P134 B1 + ~~P134 B2 + P134 B3 + P134 B4 + P134 B5~~ + P142 B1 + ~~P142 B2~~ - P56 B1 - ~~P56 B2~~ - P99 B1 - ~~P99 B2~~ - ~~P108 B1 - P108 B2~~ - ~~P131 B1 - P131 B2 - P131 B3 - PRV305 - PRV324~~​

Mountain Demand ≤ P137 B1 + ~~P137 B2~~ + ~~P140 B1~~ + ~~P140 B2~~ - ~~P149 B1~~ - P149 B2 - ~~P149 B3~~ - ~~P149 B4~~

Canal 1 Demand ≤ P56 B1 ~~+ P56 B2~~ - P59 B1 ~~- P59 B2 - P59 B3~~

Canal 2 ≤ P99 B1 ~~+ P99 B2~~ - P101 B1 ~~- P101 B2~~ - P137 B1 ~~- P137 B2 - P140 B1 - P140 B2~~

Canal 3 ≤ ~~P108 B1 + P108 B2 + P129 B4 + P129 B5 + P131 B1 + P131 B2 + P131 B3 + P134 B6~~ + P134 B7 + P134 B8 + P142 B3

Hydro34 =< P34 B1 ~~+ P34 B2~~

Hydro59 =< P59 B1 ~~+ P59 B2 + P59 B3~~

Hydro101 =< ~~P101 B1~~ + P101 B2

Hydro149 =< ~~P149 B1~~ + P149 B2 ~~+ P149 B3 + P149 B4~~

## Cost Minimization Results: Maintenance

When infrastructure in the District is taken offline for maintenance purposes, our results change. Due to the adaptability of our model design, we are able to minimize total cost under any maintenance circumstances. Because every maintenance situation is unique, it is not worthwhile visualizing this in graph format. Under maintenance, our model increases the percentage of boosters used in order to make sure the correct amount of water reaches every demand zone. Because of the importance of wells in the District’s system, the amount of boosters turned on significantly increases when a well is taken offline.

When it comes to PRVs, our model will recommend that operators turn on certain PRVs under specific maintenance situations. This type of situation is the most common when wells are taken offline in a low elevation zone. An example of this is under the situation where the two wells feeding the Lower Zone (P11\_W1 and P28\_W1) are turned off. Since the Lower Zone has the lowest elevation, there are no boosters feeding into that zone. In this situation, the only way to satisfy Lower Zone demand is to send water through the PRVs that connect the Intermediate Zone to the Lower Zone.

**Trained Random Forest Models Information by Zone:**

​​