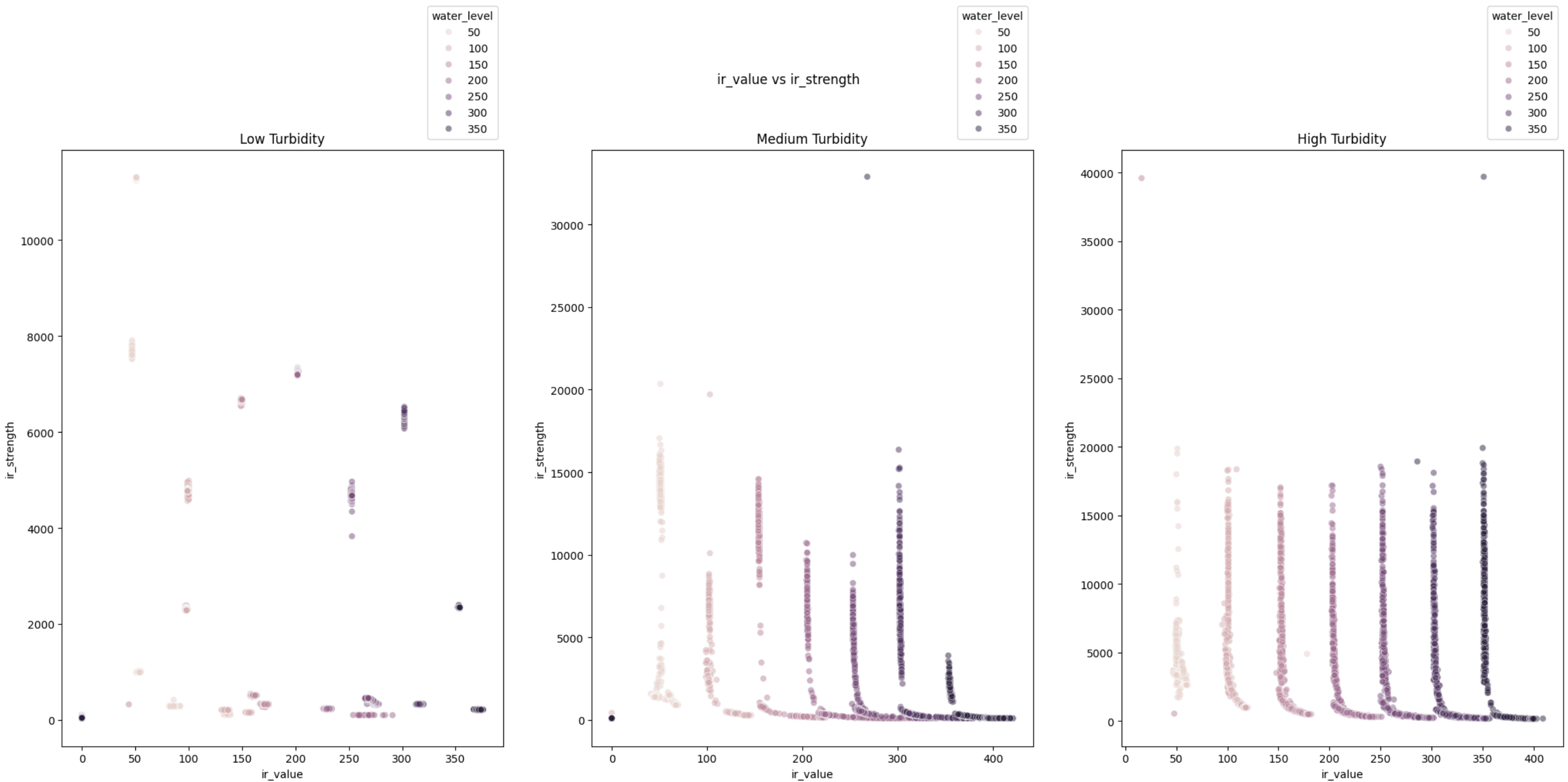
Light Detection and Ranging (LiDAR) is an emerging sensing technology that is beginning to make the job of mapping natural landscapes with a high degree of precision a more reasonable endeavor.1 It utilizes pulsing light from a laser1 along with data from other sensors, such as accelerometers and gyroscopes2, to map the top heights of natural and artificial features, such as buildings, trees, and the ground.1 Recently, LiDAR technology has begun to see use in flood risk management, especially with rivers and creeks.2 To do so, it’s starting to be used to measure a stream’s water level, and unlike ultrasonic sensors, they don’t deviate due to external conditions, including dust, fog, and rain.2 Combining this sensor data with Inertial Measurement Units (IMU) allows for the use of machine learning to be integrated for the purpose of improving water level forecasting.2

The goal of this machine learning project was create a machine learning model that predicts the water level of a laboratory-controlled artificial river given readings from various sensors. Data was collected from water under three states of turbidity: low, medium, and high. The sensors used to collect data included a LiDAR sensor, ultrasonic sensor, accelerometer, and gyroscope.2

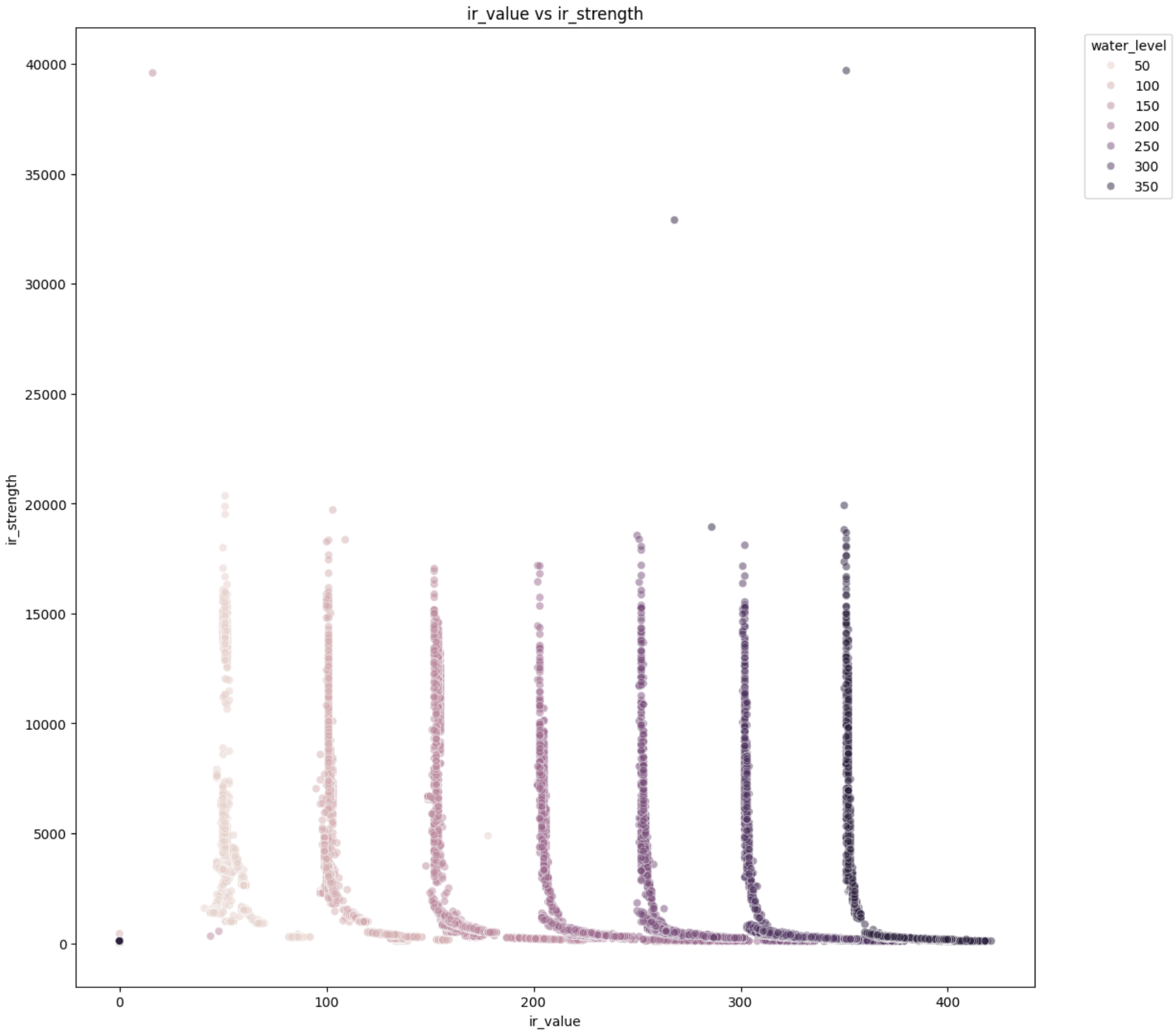
Prior to making any predictions, the data had to be explored in order to understand the relationships between the variables (sensor data) and the labels (actual water levels) better. The datasets were initially divided based on the turbidity of the water in each experiment, and so they were analyzed individually first as cohorts and then later combined and re-analyzed together. None of the datasets had any null values, and both the features and labels were numeric, so this project was a regression task, and the only encoding required was that of the turbidity feature added when concatenating the data sets. Since this feature had designations of ‘low’, ‘medium’, and ‘high’, it was ordinally encoded on a scale from 0 to 2.

The first two plots of exploratory data analysis for both individual cohort and combined analysis involved comparing the distance measured by the LiDAR sensor with its signal strength.3 For the medium and high turbidity cohorts, the relationship appeared to be a repeating pattern of exponential decay in the signal strength as the measured distance increased within bands of approximately 50 units of distance. The low turbidity dataset initially appeared to show a more sporadic relationship, but when combined with the other datasets, it turned out to follow the overall periodic decay function of the rest of the data. Each plot also had its data points colored by another variable to explore the relationship between these two features and others: first by the water level label and then by the ultrasonic sensor feature. The water level appeared to have a close relationship with the LiDAR distance readings, and the ultrasonic sensor value had a similar relationship, but to a much lesser extent.

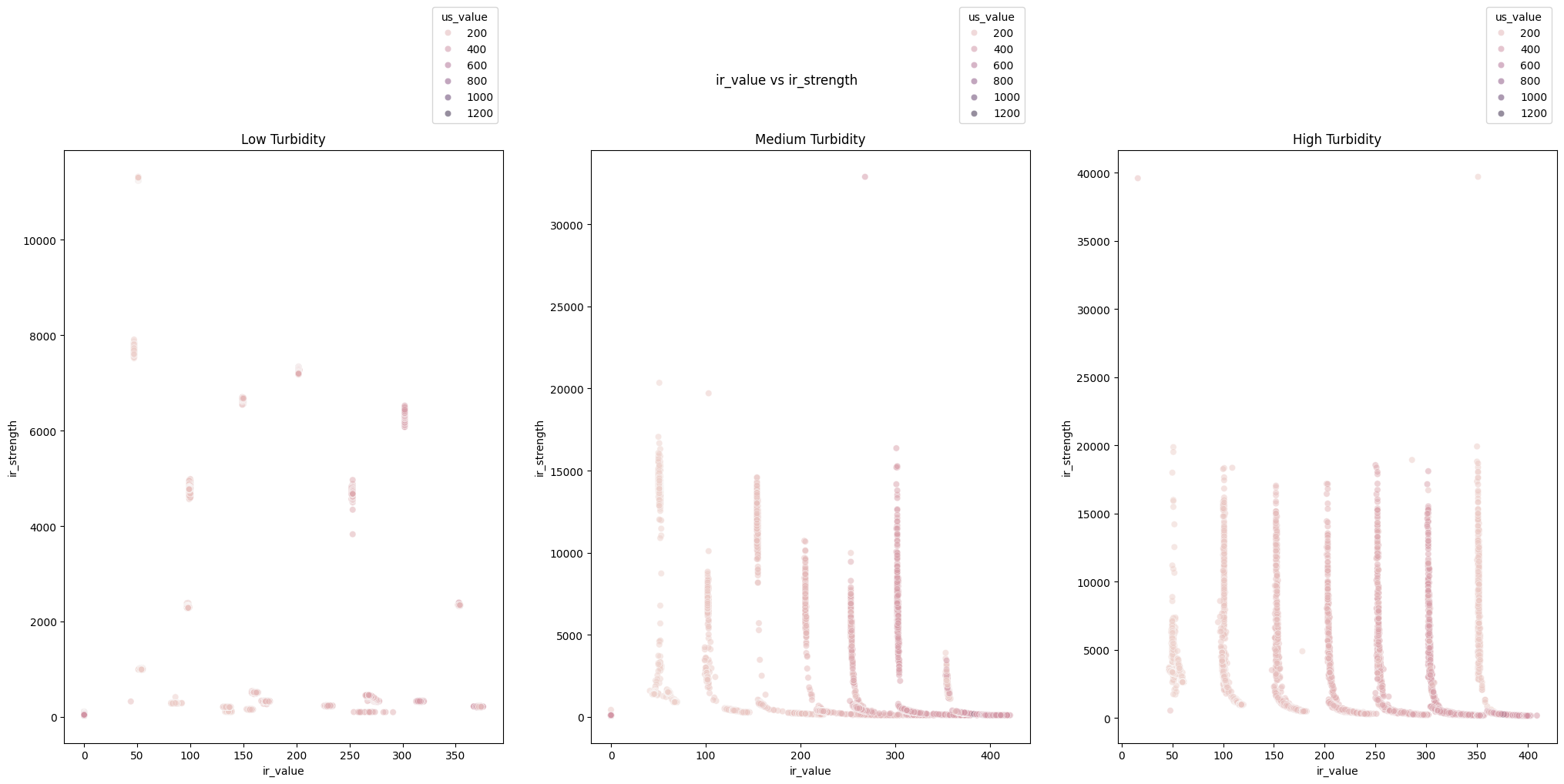
* LiDAR Distance (ir\_value) vs LiDAR Strength (ir\_strength) vs Water Level (water\_level):3
  + Cohorts:



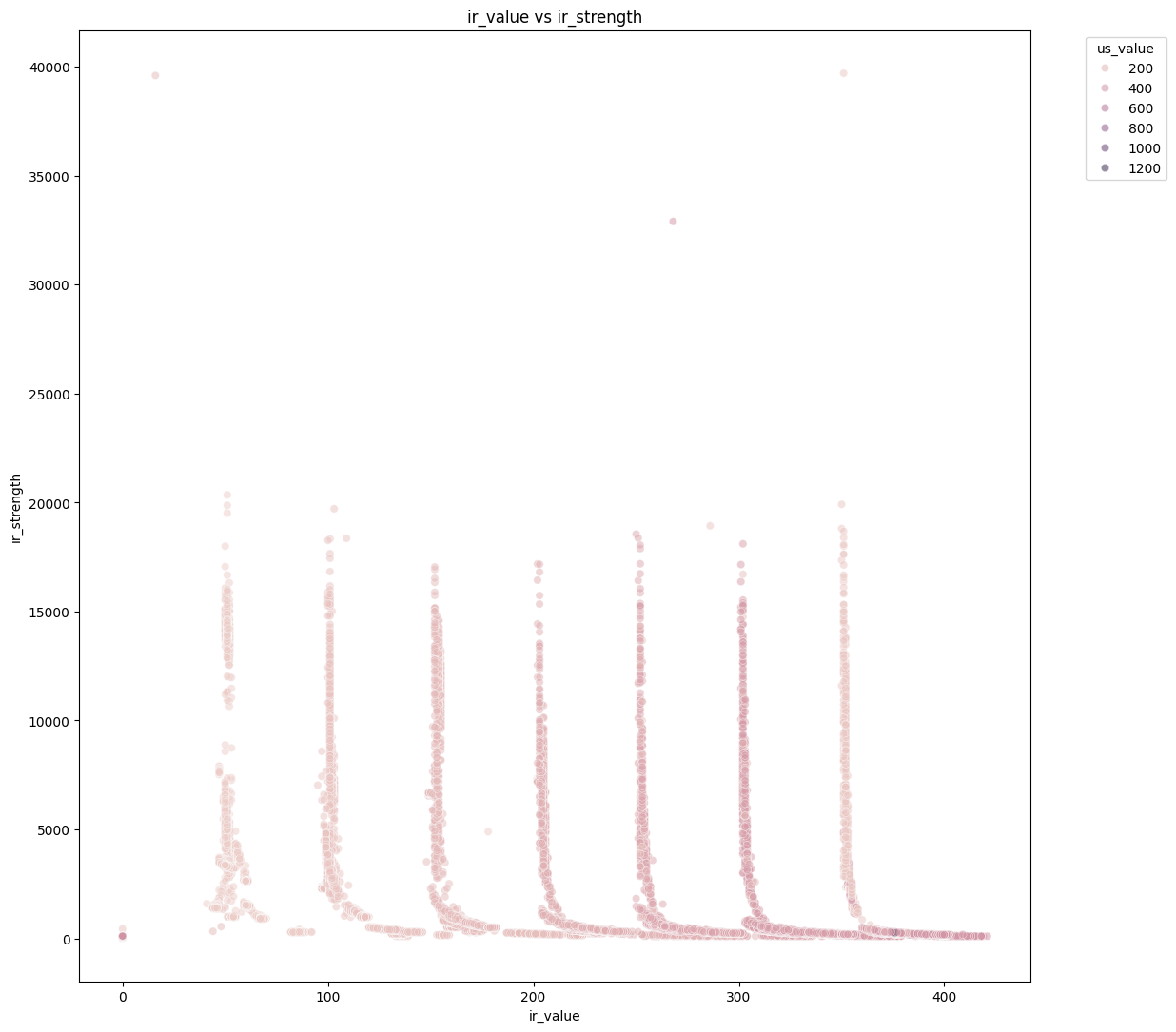
* + Full Dataset:



* LiDAR Distance (ir\_value) vs LiDAR Strength (ir\_strength) vs Ultrasonic Sensor Distance (us\_value):3
  + Cohorts:

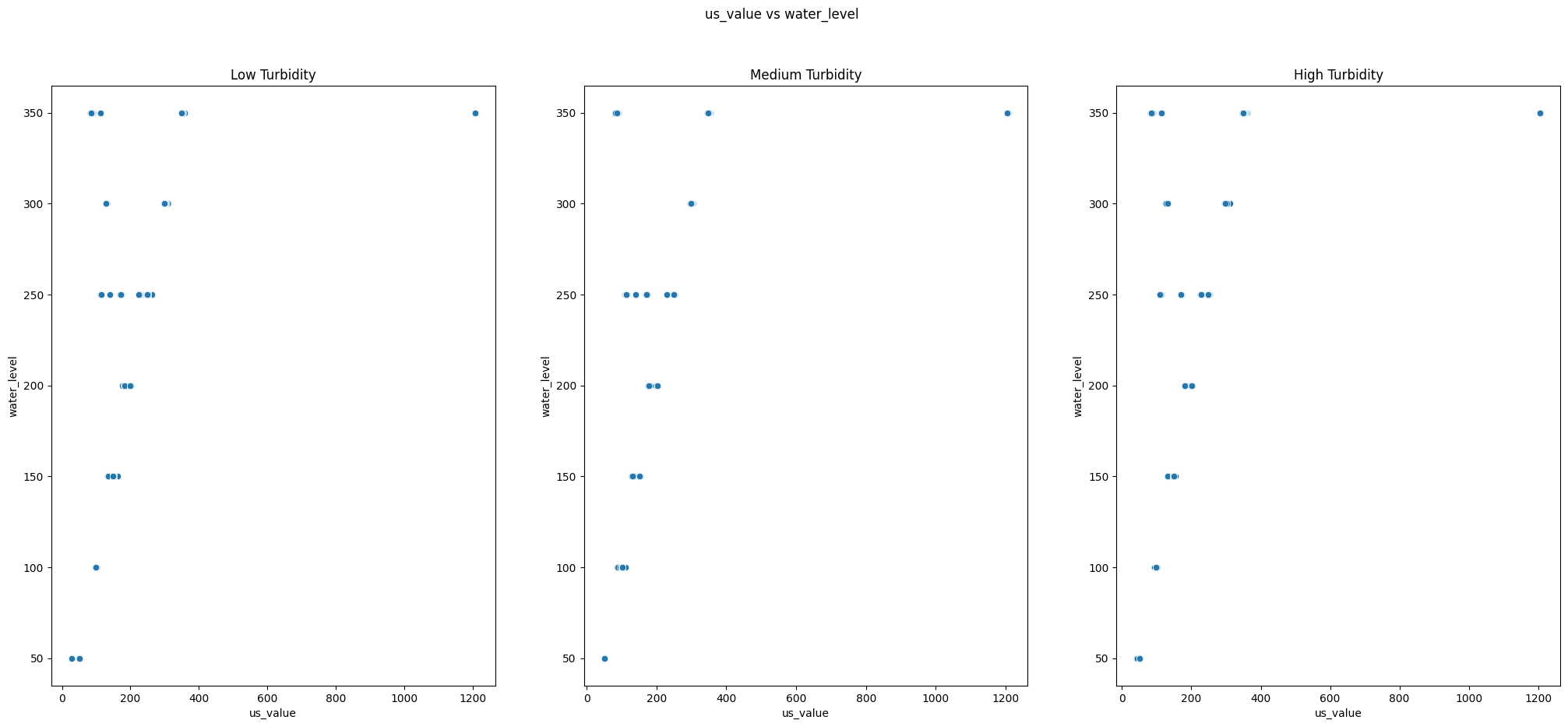


* + Full Dataset:

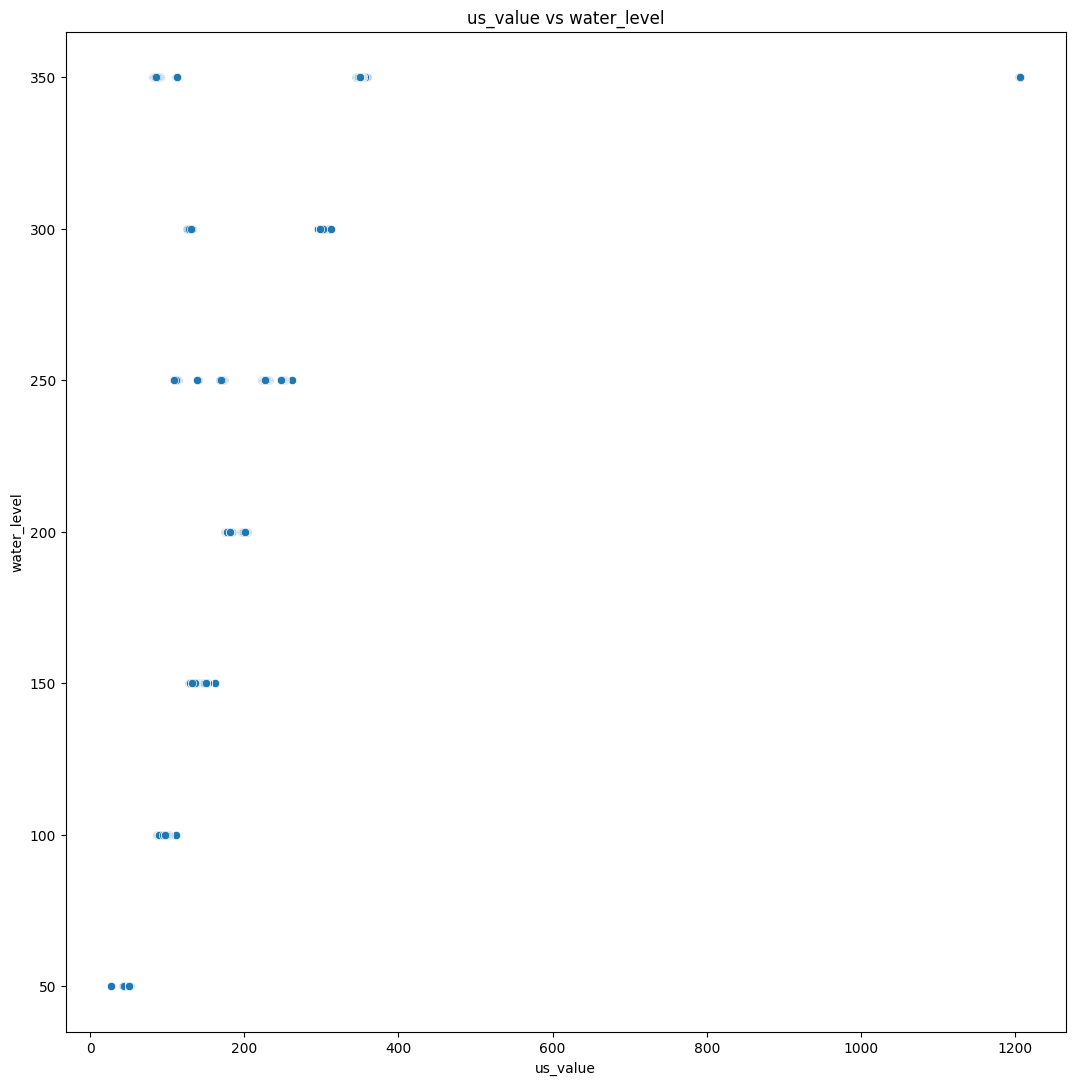


The next three plots revealed much weaker relationships between the remaining features and the water level labels. For example, there may be a faint linear relationship between the ultrasonic sensor readings and the water level, but there are many outliers in multiple directions that prove otherwise.

* Ultrasonic Sensor Distance vs Water Level:3
  + Cohorts:

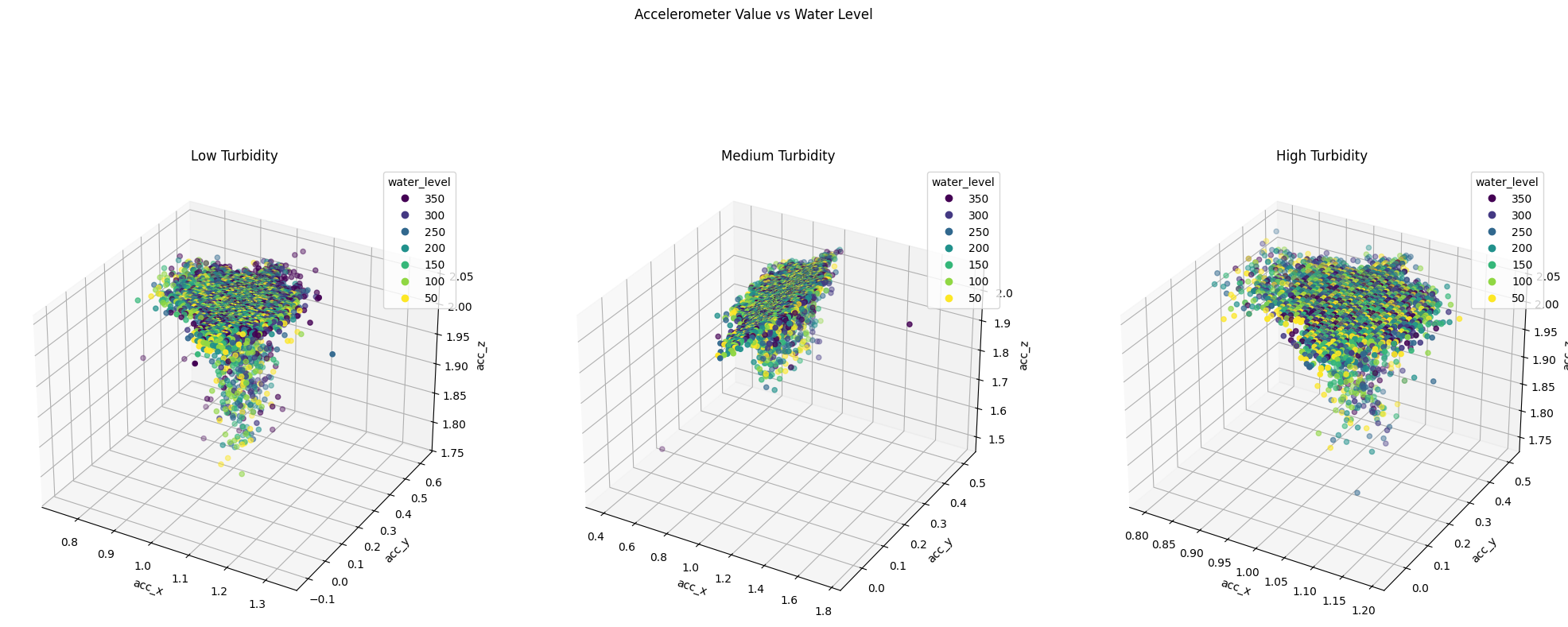


* + Full Dataset:

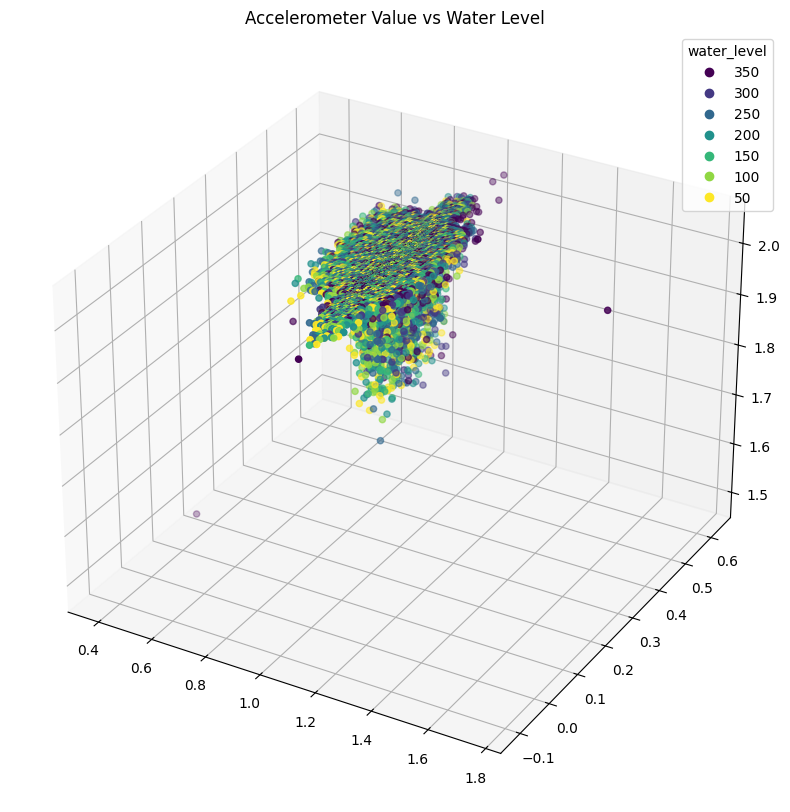


Next, the x, y, and z aspects of the accelerometer readings were compared to their corresponding water level labels. However, there were no clear relationships between either the values themselves or with the water level.

* Accelerometer X, Y, and Z Values vs Water Level:3
  + Cohorts:

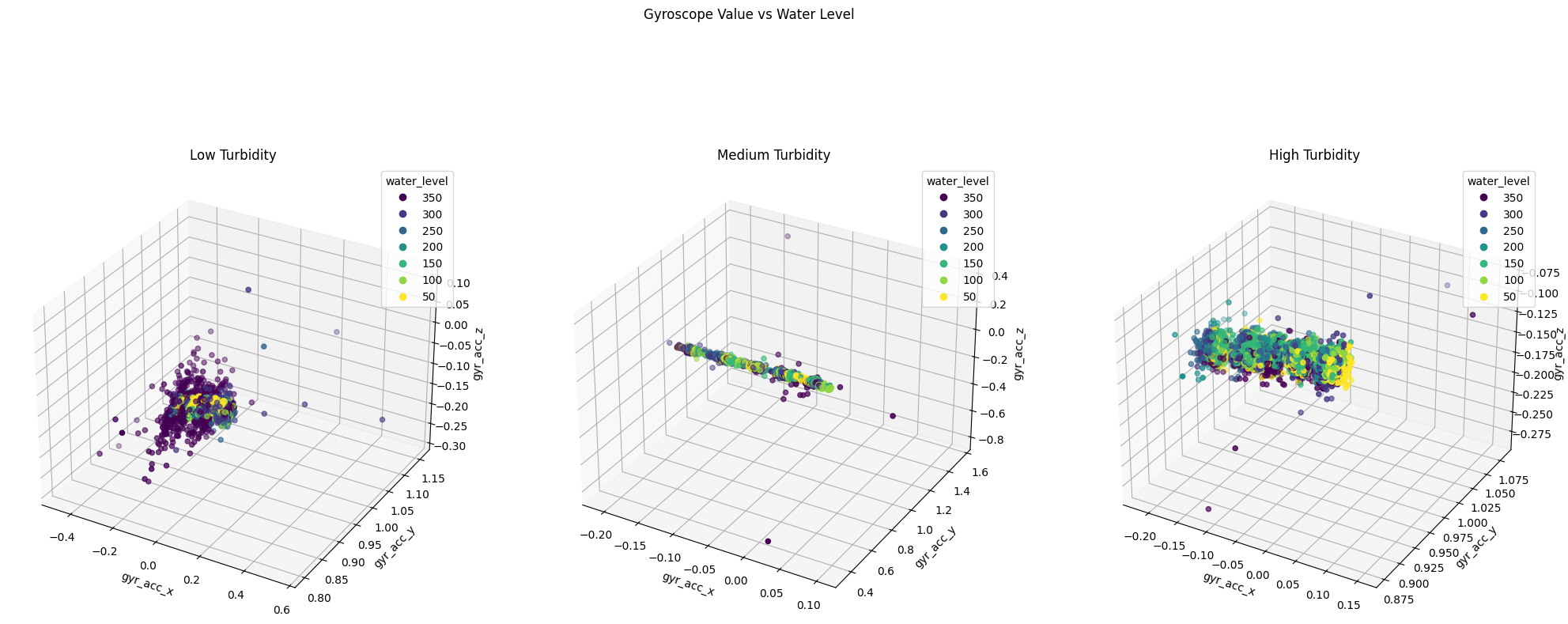


* + Full Dataset:

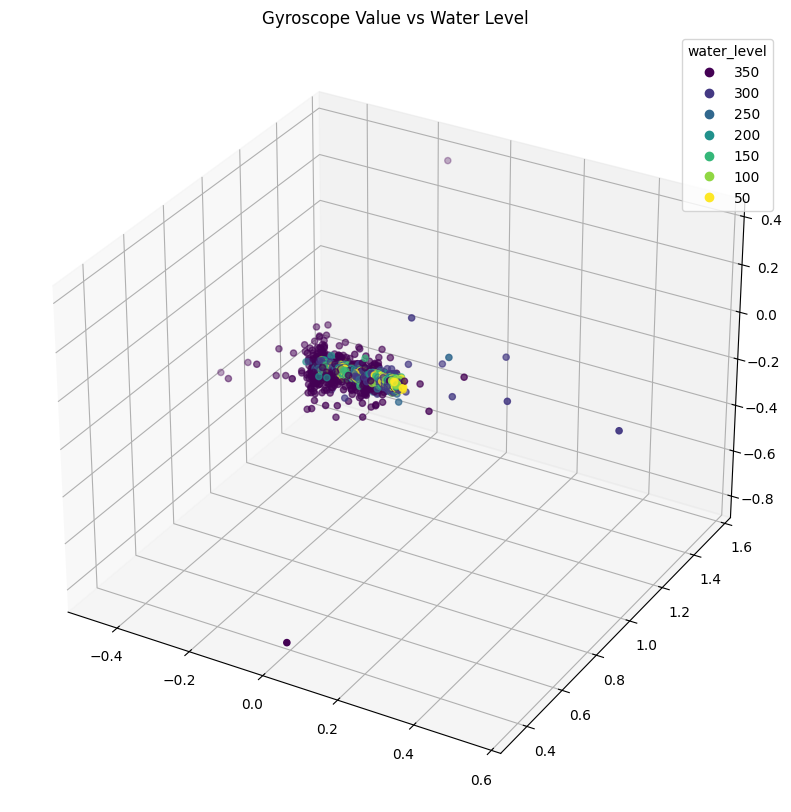


Lastly, the above was repeated for the Gyroscope’s X, Y, and Z elements, although this showed a weak cluster-esque relationship between the points and their water level labels, but with several outliers.

* Gyroscope X, Y, and Z Values vs Water Level:3
  + Cohorts:

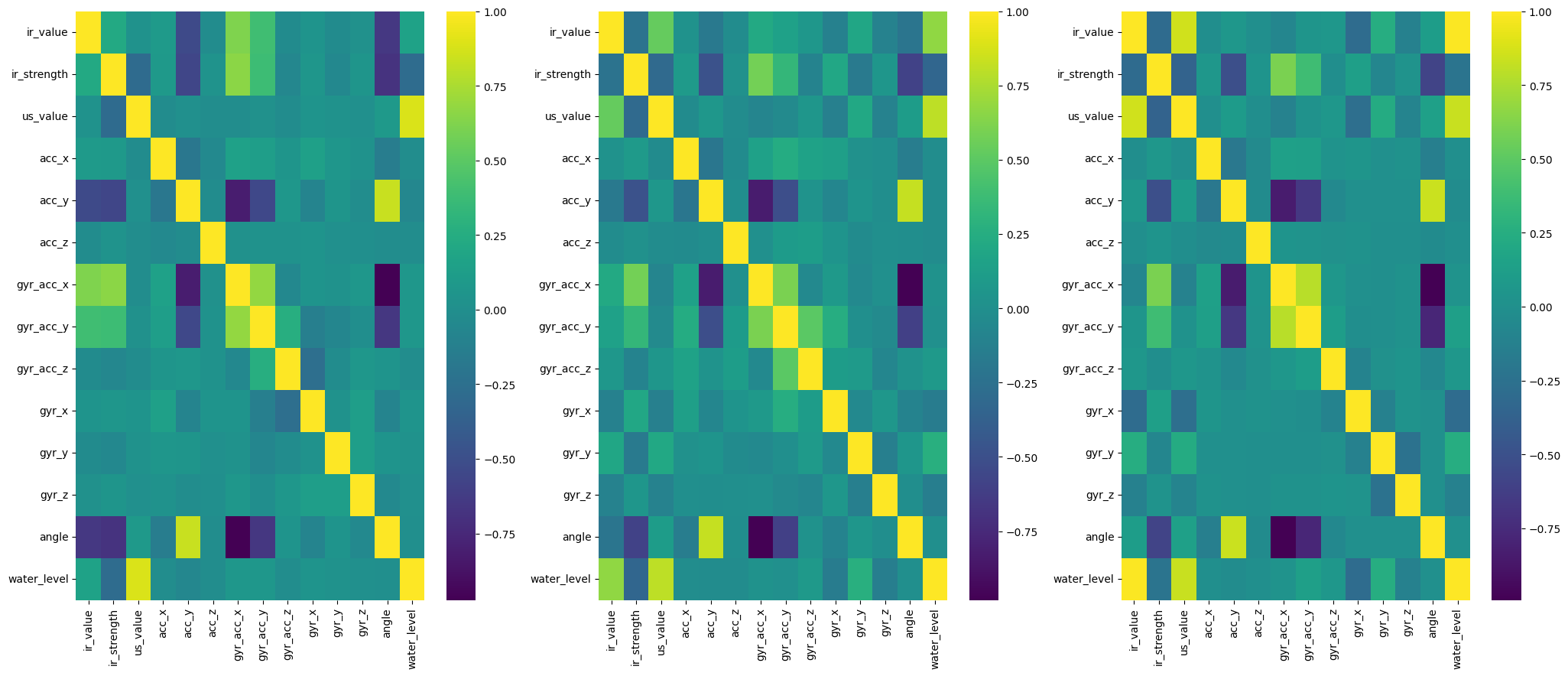


* + Full Dataset:

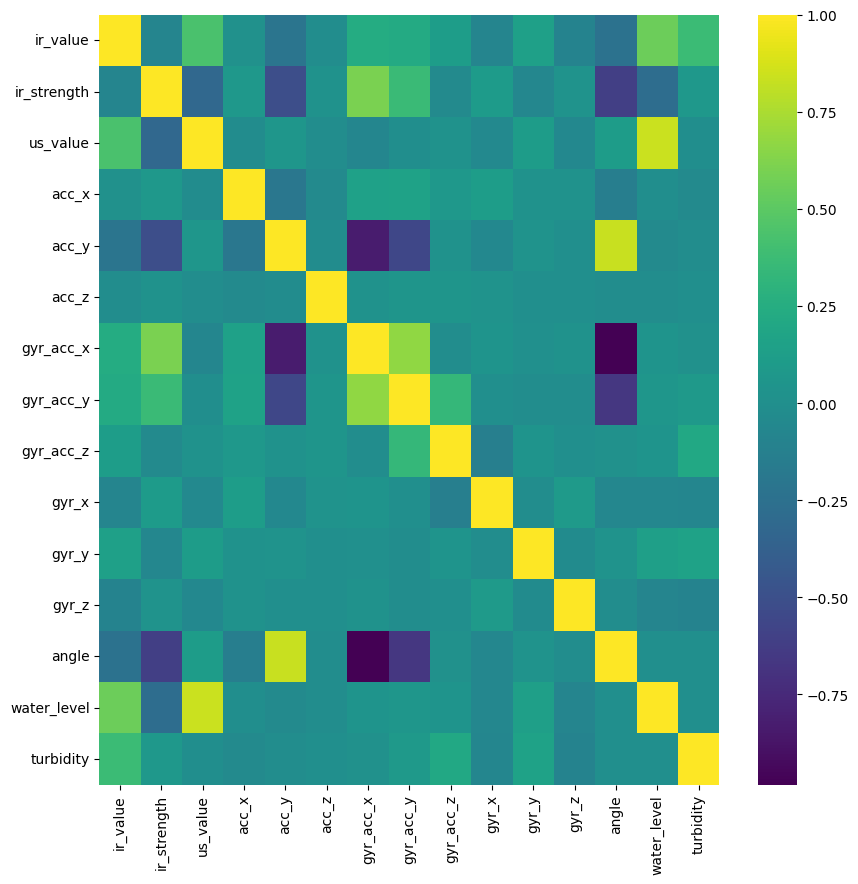


Before moving onto data preparation, the final exploratory data analysis plot generated was a heatmap showcasing the correlation between all of the variables as well as the water level label. When repeated on the complete dataset, the newly encoded turbidity variable was included.

* Cohorts:



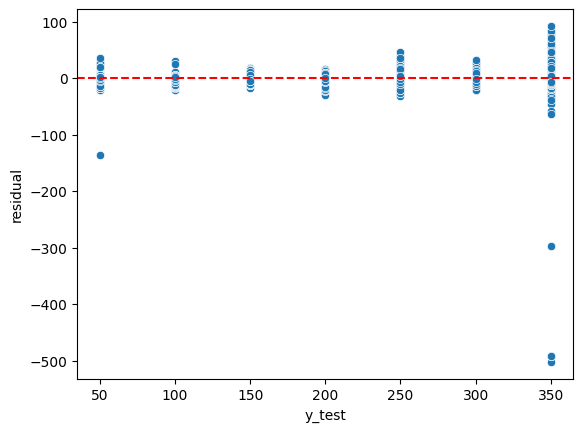
* Full Dataset:

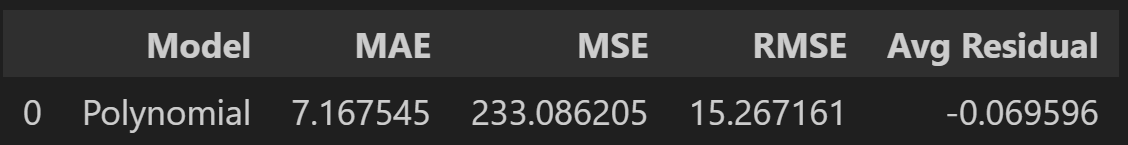


After gathering sufficient information about the data, it was time to prepare it for being used by machine learning algorithms. In supervised machine learning problems, such as this one, the features and labels are fed into the estimators separately, so they were split into an X dataset, containing all of the features, and a y dataset, with the original labels. Furthermore, supervised machine learning algorithms fit their models upon training data and utilize test data for performance evaluation. Therefore, these two datasets were further split into four ordered datasets where each row of X corresponds with the same row of y: X\_train, X\_test, y\_train, and y\_test. Finally, algorithms that rely on distance metrics perform better when the features are scaled so that each variable has a mean of 0 and a standard deviation of 1.4 Thus, copies of the X training and testing datasets, known as scaled\_X\_train and scaled\_X\_test, were created where the features were scaled to have such conditions. Additionally, a table was created to track four error statistics about each model to be trained. These were the mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), and the average residual.3 After each model was fitted, its scoring metrics would be appended to the bottom of the table.

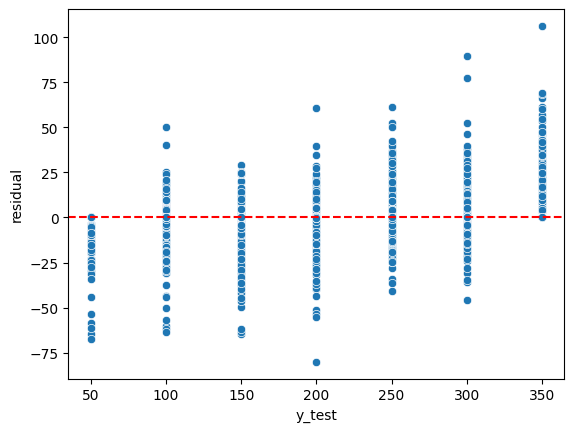
For every machine learning model created, except the cumulative voting regressor which combined most of the previous models described, various external attributes unrelated to the data that affect each model’s predictions, known as hyperparameters, are tuned to obtain the best performing model of a particular algorithm.5 As there were often large amounts of hyperparameters for each algorithm, a process known as grid search was performed on each. This involves choosing hyperparameters to explore settings with, choosing a number of options for each, and using a scoring metric to determine the combination of hyperparameters that lead to the optimal performance for a particular type of model.5 For this project, mean squared error was selected as this scoring metric.

The first machine learning model trained was a polynomial regression model. A polynomial features transformer was used to convert the feature set into a squared feature set for the 2nd degree options. The hyperparameters to grid search were the regressor’s power (capped at 2), regularization factor, and regularization ratio, and the best combination was a power of 2, a regularization factor of 0.1, and a regularization ratio of 0.99.3 Most of the residuals from testing this model had a magnitude of less than 100 units, although there were a few outliers with magnitudes up to 500 units. This model performed with the seventh best MAE, MSE, and RMSE among all 9 models tested.



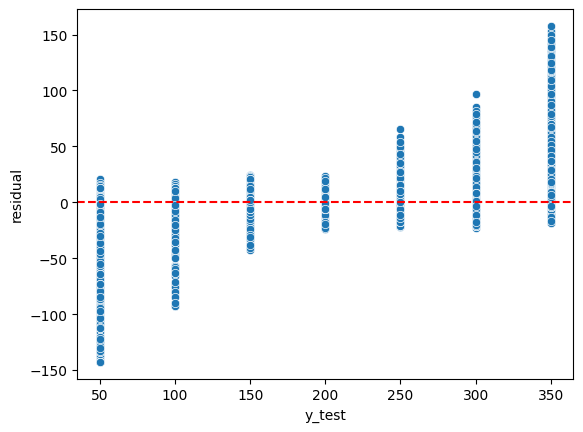


The K-Nearest Neighbors Regressor was the second algorithm selected for training on the data. The number of neighbors, distance metric (taxicab or Euclidean), and neighbor weight parameter (uniform or distance) were the critical hyperparameters to tune for this model. 10 neighbors, taxicab distance measurement, and distance-based neighbor weight were found to be the best combination of hyperparameters. While the residuals were more skewed in the edge cases (water levels of 50 and 300) and the average residual had a slightly higher magnitude, this model overall performed slightly better, with the sixth best MAE, MSE, and RMSE between all models.



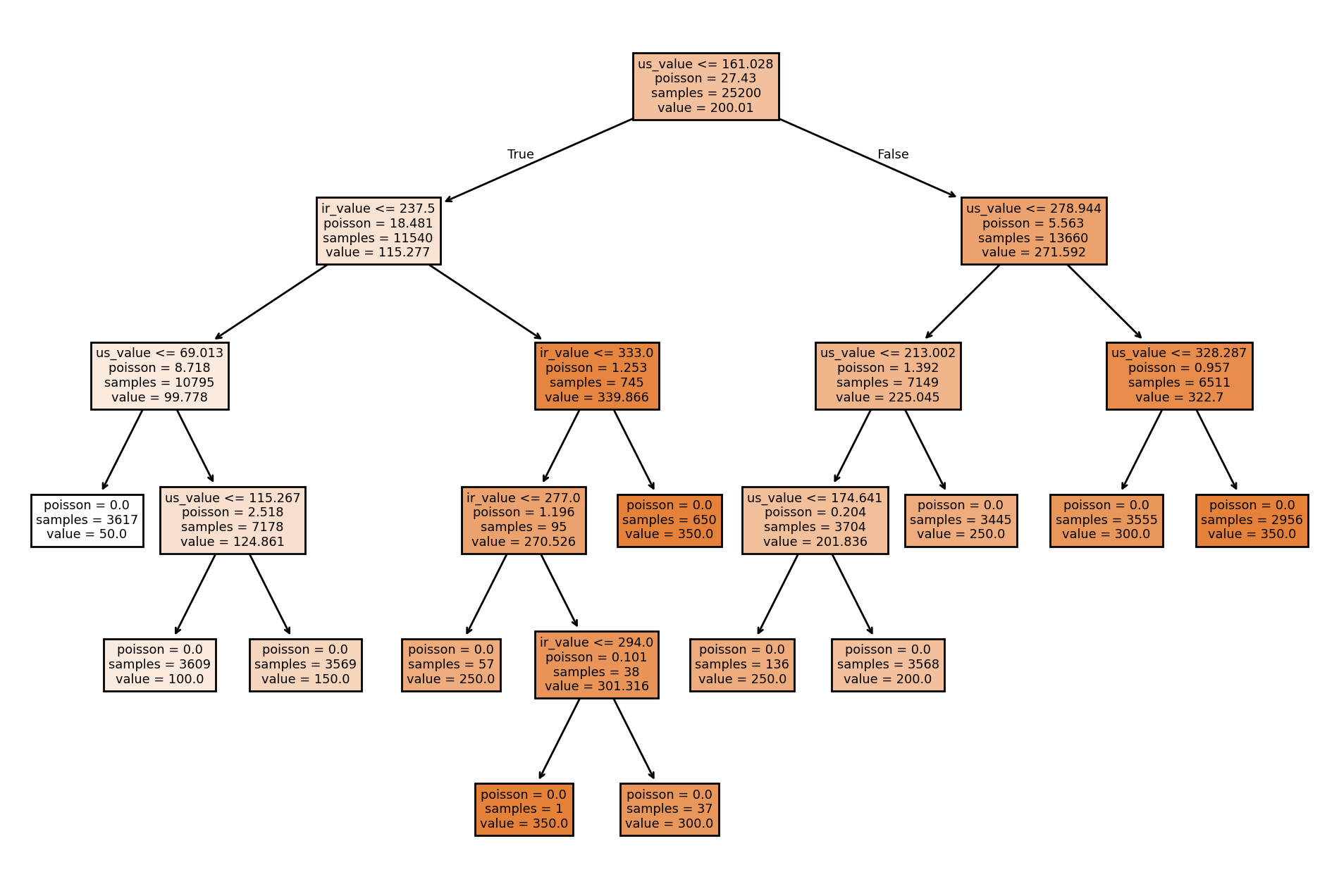


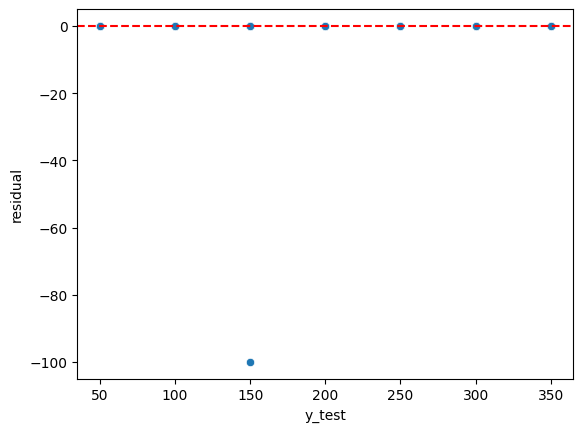
A Support Vector Machine was chosen to be the third model tested. This model took the longest to train, as it had the largest amount of and most complex set of hyperparameters to explore. These parameters were the kernel used to map the data into a higher-degree space6 (set to ‘rbf’), degree for a polynomial kernel (2 but ignored due to being rbf), kernel coefficient (auto), regularization parameter (1), and error tolerance (0.01). Unfortunately, this model turned out to be the second worst overall, with the residuals more skewed, the average residual higher than 1, and the second worst MAE, MSE, and RMSE metrics.





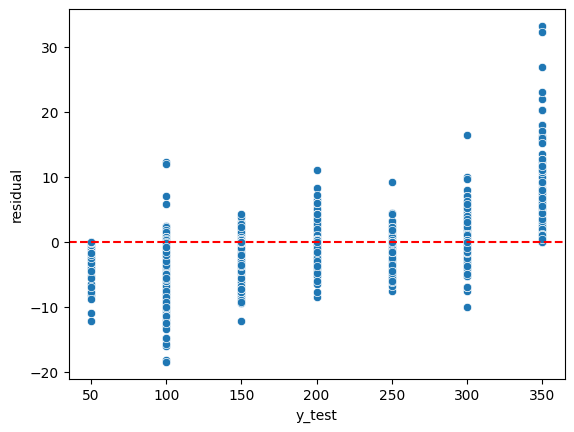
The rest of the models leading up to the cumulative voting model were based on the decision tree, which doesn’t utilize distance metrics, so the original split datasets were able to be passed in for these methods rather than the scaled versions. The first of these was a single decision tree, with its hyperparameters of maximum depth (5), maximum number of leaf nodes (45), and splitting criterion (Poisson loss function) to prevent overfitting. Due to these limitations in the creation of the tree, this model performed particularly well on the testing data, with the lowest, and therefore best, average residual and MAE as well as the second best MSE and RMSE.





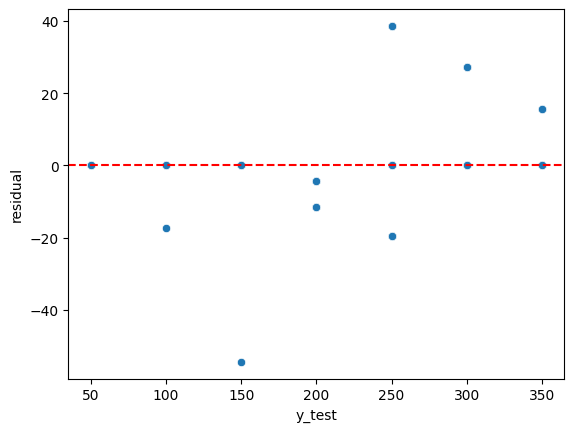


The random forest model was the first model powered by an ensemble of decision tree models, although these trees are generally restricted further than just using one tree. This is because the goal of the next few models is to combine a number of weak learners into a strong model.7 The key hyperparameters adjusted for this model were the number of decision trees to generate (200), whether to allow features to be reused (false), the maximum number of features to be randomly drawn for each tree (the square root of the total number of features), and the splitting criterion (Friedman MSE). This lead to another well-performing model, with the fourth best average residual and the third best MAE, MSE, and RMSE among all 9 models.



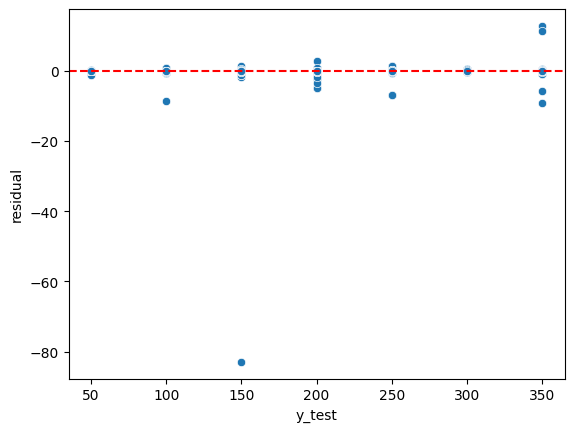


Unlike the random forest model, where each decision tree is trained independently of each other, boosting methods, such as Adaboost and Gradient Boosting, involve using each decision tree to improve the next one generated.7 For the Adaboost model, the number of trees generated (25), loss function (exponential) and learning rate (0.3) were grid searched to find the optimal model settings. Yet another high performance model was generated as a result of the hyperparameter selections. While the average residual was only the seventh best and slightly skewed in the negative direction, the MAE, MSE, and RMSE were all the 4th best among all models trained.



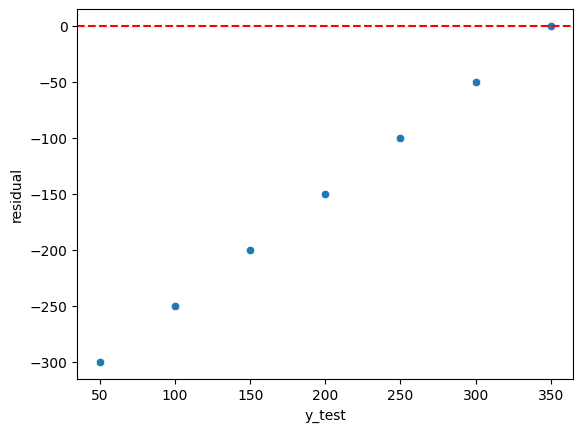


The other major type of tree boosting ML algorithm is the Gradient Boosting model. It is powered by the same three hyperparameters that Adaboost is, being the number of trees (200), loss function (squared error), and learning rate (0.2). However, there is also an additional hyperparameter to control the individual trees, being the criterion hyperparameter (Friedman MSE). Gradient Boosting is generally more suitable for regression tasks than Adaboost8, so unsurprisingly, this model performed significantly better. In fact, between every model created, this was the best performing one across the board, with the best MAE, MSE, RMSE, and average residual values. Additionally, the residual with the greatest magnitude was less than 100 distance units off from its corresponding true label.



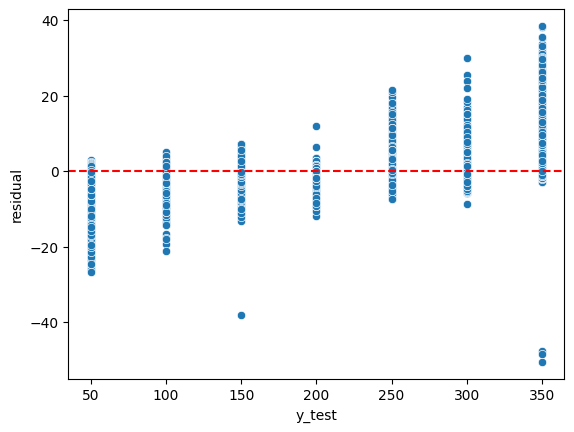


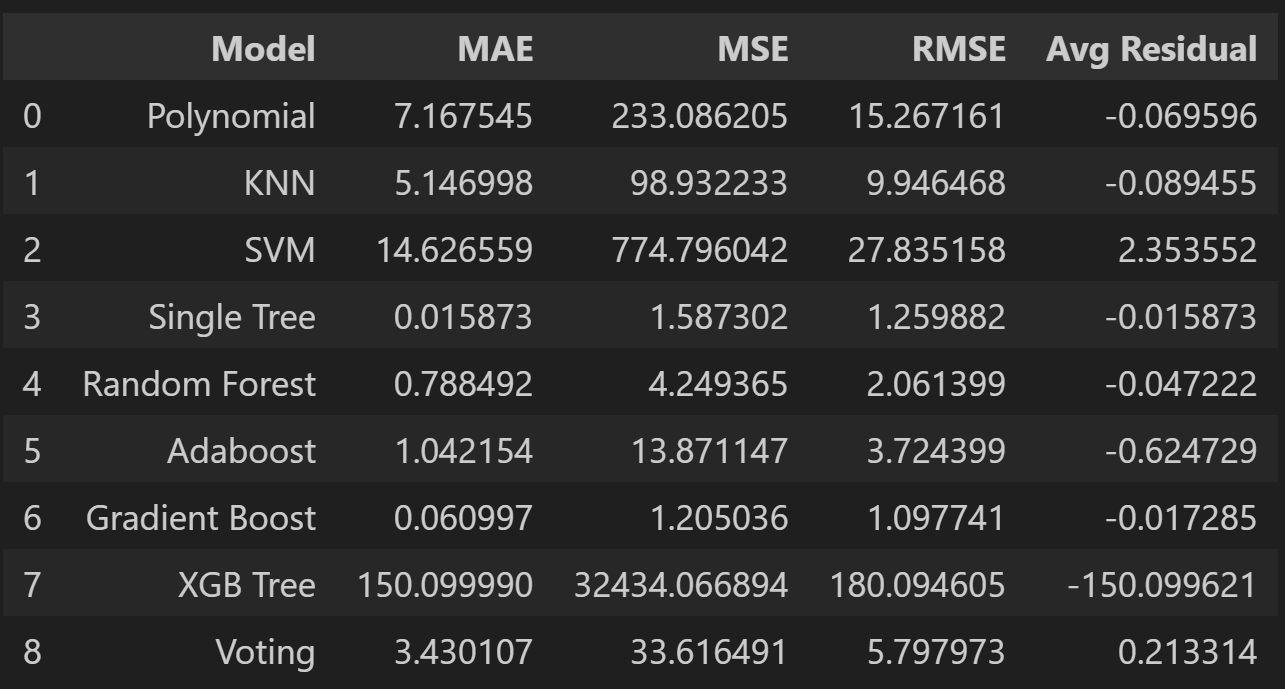
A variant of the Gradient Boosting algorithm designed outside of the Scikit-Learn library is the Extreme Gradient Boosting algorithm (XGBoost), which can be perfomed with a number of unique base models.9 Due to immense training time and unexpected results, XGBoost was only run utilizing the decision tree as the base model for now. The key hyperparameters for this model were the number of estimators (1000), max tree depth (6), learning rate (0.4), lambda regularization term (100), and alpha regularization (1). However, this algorithm produced the worst model, with the highest error metrics in every category by a severe margin. The individual residuals show that the model may have predicted the same value for every test value for some reason. Therefore, this model was left out of the final voting model.





The final model generated combined all of the previous models (except the XGBoost model) into a single voting model that uses the average prediction of all its input models as its own output. To produce this model, a copy of each model with its best hyperparameter combination was fed into the estimator list parameter of this model. For models based on distance metrics (every non-tree based model), a pipeline was created with the standard scaler and the model itself as the two steps. The polynomial sub-model’s pipeline also contained the polynomial features generator as the first step ahead of the scaler and model itself. The result was a relatively average model, with the sixth best average residual and the fifth best MAE, MSE, and RMSE. This was expected behavior, as the less accurate models affected the error metrics than the more accurate models.





These models demonstrate the capability of utilizing machine learning to predict river flooding utilizing data from a LiDAR sensor, regardless of the water’s turbidity. Although some models performed better than others, with the exception of the XGBoost model, they all performed with tolerable accuracy, especially the other tree-based models. Neural Network algorithms could also be explored, although they may require immense training time due to their complexity and the high number of times they would have to view the data. For now, the classic machine learning algorithms offer a good insight into the future of LiDAR technology.

References:

1. <https://harcresearch.org/news/when-technology-and-nature-collide-lidar-and-the-importance-of-biodiversity/?gad_source=1&gad_campaignid=9558509408&gbraid=0AAAAACg5UaQcPomhU5IOhcKP991S_Zt33&gclid=Cj0KCQjw-ZHEBhCxARIsAGGN96JqAcTmzLz95JuFdmWgYF7HZ0INDpa1naJ_Ixy2ex9dtBtA26Crut0aAgWUEALw_wcB>
2. <https://www.kaggle.com/datasets/caetanoranieri/water-level-identification-with-lidar>
3. See Attached File
4. <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>
5. <https://aws.amazon.com/what-is/hyperparameter-tuning/>
6. <https://medium.com/@abhishekjainindore24/svm-kernels-and-its-type-dfc3d5f2dcd8>
7. <https://duchesnay.github.io/pystatsml/machine_learning/ensemble_learning.html>
8. <https://medium.com/analytics-vidhya/what-is-gradient-boosting-how-is-it-different-from-ada-boost-2d5ff5767cb2>
9. <https://xgboost.readthedocs.io/en/stable/parameter.html>