William Kim

NFL Stadium Attendance Dataset
https://www.kaggle.com/datasets/sujaykapadnis/nfl-stadium-attendance-dataset

Reason for choosing this dataset:
The dataset I found had a CSV file containing the attendance records of NFL games from 2000 to 2019. The data contained information on the weekly attendance, the total attendance at home games for the year, the total attendance that year during away games, and the total attendance that team had for the whole season. In other CSV files, the data set also had data on who won those games, how many points they scored, their standings that year, their success, point differentials, turnovers, and more. I selected this dataset initially because of my interest in and passion for American football. However, as I looked at the dataset more in-depth, I became more interested in specific correlations and patterns in the data. I was curious whether there were correlations between stadium attendance and a team's performance during the season. Which team had the highest average attendance? Which team had the lowest? Were they related to the team's performance/success? I chose this dataset because of my love for football and because of all these questions I had and wanted to answer as I began to look at and think about the dataset.

Question I am trying to answer:
The question I am going to seek to answer is whether there is a relationship between stadium attendance and an NFL team's success. Can you predict the attendance of the team's game based on their success, which will be measured by the number of wins they have.

Cleaning the data:
The very first thing I did was to clean my data to make it presentable and easier to read and so that I could organize my data. The kaggle dataset gave me 3 CSV files, attendance, games, and standings. Looking at the columns and data in each CSV, I decided that I only needed to use attendance and games because those two CSV files had records from every single game, whereas the standings CSV had data about each time for that whole season. I then cleaned up the attendance and games CSV files by eliminating the columns I needed. A problem I ran into later was that for the "week" column, which was supposed to be a column of all numbers referring to each week in the NFL season, of which there are 17, the playoff games, which were the Wild Card, Divisional, Conference, and Super Bowl, were written as strings. This would interfere with my code and would mess up my calculations and data parsing, so I converted all the playoff strings to be weeks 18, 19, 20, and 21. For the attendance and games data sets, I now only had the columns I needed. For attendance CSV, a column called "team" which was a string with the city or location the NFL team is from, then a column called "team_name" which was a string with the team's name/mascot, then a column called "year" which had the year a game was played, then a column called "week" which had a number corresponding to the week of that NFL season, and then finally a column called "weekly_attendance" which had the number of people in attendance for that game for that week. Now that my data was all cleaned

and all the extra unnecessary columns were removed and ready to be used, I could begin to write my code.

Data reading and preparing module (Data.rs module):
So the very first module I created was the “data.rs” which seeks to load the data, parse through it, and then prepare it for statistical analysis that I seek to do with the data. The first thing I did was to import “Reader” from the CSV crate which will allow me to read through the CSV files and parse through them. I then import “Deserialize” from the serde crate which will allow me to deserialize the data so that I can use it in my Rust structs. I then import the “Error” trait from the std library so that I can perform error handling. I then make sure to go to my Cargo.toml file and add all the things I need in dependencies with the correct and latest versions. After importing everything I need and fixing my dependencies, I start by creating a struct called “Attendance” which will hold onto different fields of data with different information in the CSV files that will hold onto the names of the columns and the types of each columns. “Team” will be a string, “team_name” will be a string, “year” will be u16, “week” will be u8, and “weekly_attendance” will be Option<f64>, since the data could potentially be a float number or could be missing. **I make sure to make the struct public by putting the ‘pub’ keyword before it so that I can reference the struct later in the main.rs module since I am in a separate module right now. I make sure to do this for all the functions in all the modules that I want to call and use in my main.rs module.** I used a struct here because structs are good for holding onto and grouping together data to make it easier to read and easier to use later down the road. For the first column and first field in the struct, “team”, I put “#[allow(dead_code)]” in front of it, because I found out later after I ran my code that “team” was unused, but important when running my tests later on. This allows the code to process and run, but not return an error statement when “team” goes unused. For “weekly_attendance” I put “#[serde(deserialize_with = "deserialize_attendance")]” in front of it, so that I can use a custom function to deserialize “weekly_attendance” because there are some cells where “NA” was written for a week where the attendance was not recorded or was unavailable. I then create the “deserialize_attendance” function. The function takes in the deserialize from serde and then returns an option<f64> depending on whether the cell passes if its valid or None if the cell is NA. The function defines ‘s’ as an optional string in case it is a number of NA, and then it uses a match statement to match if s is a string ‘NA’ it returns None and if it is a valid number then it parses it as the f64. The next struct, “Game” which is for the game CSV, is very very similar to the Attendance struct I just made. ‘Year’ will be u16, ‘week’ will be u8, ‘winner’ will be a string, and then ‘home_team’ and ‘away_team’ will be strings, and both of them will have #[allow(dead_code)] in front of them for the same reason as ‘team’ in the Attendance struct did. Next, I created functions “load_attendance_data” and “load_games_data” that both are very similar and accomplish similar tasks, which are to load the CSV files and to parse through them, and add them to a vector. The function finds the CSV file and then using Reader, reads it and assigns it to the mutuable variable rdr. Then it takes it as input, then creates a mutable variable called data which is an empty vector to store data, then using a for loop we iterate through each row in the CSV file, and using a match statement, we either add it to the data vector using push, or it returns an error and continues. Both ‘load_attendance_data’ and ‘load_games_data’ accomplish work this way and are similar but the difference is that one works with the attendance data set

and struct and the other works with the games data set and struct. The next function in the data.rs module is the “prepare_features” function which prepares the data for regression and decision trees. This function works by taking in the Attendance and Games structs as input and then iterating through each row to check if each weekly_attendance works using an if let statement, and then proceeds to create a variable called ‘wins’ which will filter the data to match the years, weeks, and game winner and team name in both datasets. The attendance and then the number of wins is then placed into a vector called ‘features’ as a tuple, which is then returned. This prepares and organizes the data for regression analysis and decision trees.
Regression Module (regression.rs module):
The next step now that the data has all been read and prepared, is to perform the first statistical analysis method, Regression. First, I imported the Error trait to handle errors, then created a public function ‘perform_regression’ which will carry out the regression on the data and will be accessible and callable by the main.rs module since it has pub before it. The function will take in the number of wins a team has and the weekly attendance number in the form of a reference to a slice of tuples where each tuple has a pair of f64 values which represents the wins and attendance. The function then uses an if statement to go through each data point in features and make sure that features is not empty, and if it is empty, it will print an error statement and return 0s. Next, the function will define x as the number of wins, and y to be the attendance numbers, splitting up ‘features’, and then iterates through each data point, using .map to map each tuple to its opposite version, and then using .unzip to split the tuples into 2 separate vectors. Next, I calculate the mean of x and the mean of y, dividing each by n, which is the length/number of data points. I use this in the next step to calculate the slope and intercept. I then set the numerator and denominator to be mutable variables starting at 0, and then iterate through each x and y pair and set the numerator to be the covariance, calculated by taking each x value and subtracting the mean from it, then multiplying it by the paired y value minus the y mean, and then the denominator is set to be the variance of x which is each x minus the x mean, squared. I then input an error statement in case the denominator is too small and close to 0, which would make regression pointless. Next, I define the variable ‘slope’ to be the slope which is found by dividing the numerator by the denominator, and then the variable ‘intercept’ which is found by subtracting the y mean by the slope times the x mean. The value of y when x is 0. Next, I calculate R-squared by first defining the mutable variables ‘ss_total’ and ‘ss_residual’ to be set to 0. These variables represent the sum of the squares total and the residual of the sum of squares. Next, using a for loop, I iterate through the data to calculate the sum of squares total and residual. I do this by defining the variable ‘y_pred’ to be the predicted value using the intercept and slope found before (slope* x + intercept). I then set the ss_total to be the y value - the y mean, all squared using powi(2), and then set the residual to be the actual y value - y_pred, all squared. Next, I return an error if all the y values are the same, meaning that there is no variance, and so it returns r_squared = 0. I then find r_squared by setting it to be 1.0 - (ss_residual/ss_total). Finally, the intercept, slope, and r_squared are all returned.

Decision Trees Module (decision_tree.rs module):
Next, I created a module that contains the function that will perform decision tree analysis to see if I could predict the attendance of a game based on the number of wins. The very first thing I do is ‘#[derive(Clone, Debug)]’ to allow me to clone things and allow me to print things properly. I

then create a public struct called 'DecisionNode' made up of feature: usize, the index of the feature, threshold: f64, the value of the feature, left: Option<Box<DecisionNode>>, the pointer to the left child, right: Option<Box<DecisionNode>>, the pointer to the right child, value: Option<f64>, the value of the node. This struct will hold onto and group information associated with the decision tree and its nodes. I then use an impl block to implement the methods related to the DecisionNode struct. I then create a function called 'predict' that will take in a slice of f64 values and transverse through the decision tree to correctly predict the outcome. The function first uses an if statement to return the value if the DecisionNode is a leaf node, and then using another if else statement to see if it is less than the threshold value, which if it is then it will predict using the left child node using as_ref() to convert the option into a reference to the inner value and then for all other cases where the value is greater than or equal to the threshold value, it will be predicted using the right child node. This function predicts the output and splits the choices. The next function is called 'train_decision_tree' and it seeks to train the decision tree using a data set. The function takes in the data which are references to the tuple with wins and attendance, along with the maximum depth of the tree, and then it returns a trained DecisionNode at the very end. The function starts by using an if statement, to see if the tree has already reached its maximum depth, where depth = 0, or if the length of data is only one. If these are the cases, then the average of the attendances are found and assigned to the variable avg_value, and then a DecisionNode is returned with the avg_value as the value. Next, I set the feature to be 0, and then calculate and set the threshold value to be the average of the x values, and then split the data into a vector with all the points where x is less than threshold to be on the left, and a vector with all the points greater than or equal to the threshold value to be on the right. I then train the left subtree and the right subtree using the train_decision_trees function with depths -1 because we moved one level down and to make sure when we reach depth = 0 the recursion stops. I then set them to the variable left_node and right_node respectively and return the internal decision tree node with the new left and right pointers which are set to the trained decision trees left_node and right_node, and value set to none because it is not a leaf. Now that the decision tree has been trained, I now create a function that will test the accuracy of the decision tree. The function called: 'evaluate_decision_tree' will take in the trained tree and the features which are the test data, and returns the accuracy of the decision tree model. I then set the variable predictions to be all the predictions for all the x values in the input, and then create a variable called 'mse' which calculates the mean squared error by iterating through each value by pairing the predicted and the actual values and then calculating the squared error, which is the predicted - the actual, all squared. Next the mean squared error is found by diving the sum of all the squared errors by the number of total data points. I then let 'rmse' be the square root of the mean squared error, which would be the root mean squared error, and then subtract the rmse from 1 to give us the accuracy of how well the decision tree model is able to predict the attendance of a game based on the number of wins a team has.

Tests (Test.rs module):

Next, I created three tests to test the functionality of my code and to make sure that all was running properly and correctly. First, I put '#[cfg(test)]' at the top of my code to make sure that all my tests will run as tests and 'mod tests {' to declare a new module as tests. I then import the necessary functions and things that I am going to run tests on from the other modules using my dependencies and crate. From the data.rs I pull in the Attendance and Game structs along with the prepare_features function, and then from regression.rs I import the perform_regression function and from the decision_tree.rs I import the train_decision_trees and evaluate_decision_trees functions. The first test I created was a test of the functionality of the ‘prepare_features’ function. I do this by making a function called ‘test_prepare_featuress’ that will create fake attendance and game data that will be in a vector of Attendance and Game structs, each with fake team names, example years, example weeks, made-up weekly attendances, fake home and away teams, and a made-up winner of the games. I then use those vectors with structs and then run it through the prepare_features function and then use ‘assert_eq!’ to check the inputs and ensure that two features were generated, that one of the teams won with a certain attendance, and that one team lost with a certain number of attendance. This test ensures that the prepare_features function runs properly and correctly converts the features properly. The next test I made tests that the outputs of regression are correct and that they come out properly. I perform this test by making the function ‘test_perform_regression’ and defining the variable features to be a vector of tuples, each representing mock data of the attendance and wins. The function then performs the perform_regression function and uses ‘assert_eq!’ to check that the intercept is positive, the slope is positive, and that there is a valid R-squared value. Finally, the last test I created was a test to test the functionality of the decision_tree function. I create a function called ‘test_decision_tree’ that takes in the same fake data set used in the previous regression test, and then set the variable tree to be the train_decision_tree function run with the fake data and a depth of 3, and then define the variable accuracy to be the accuracy found from running the evaluate_decision_tree function. I then used ‘assert_eq!’ to confirm that the accuracy value found is within 0 and 1.

Main.rs module:
The final step is to create my main function which will put everything together and then will run all the code and give me the output that will help me answer the question of whether there is a correlation between the weekly stadium attendance at a football game and the success of an NFL team measured in the number of wins they have. First, I put #[cfg(test)] at the top to make sure that I can run my tests, and then I import all the modules that I made with all the public code so that I can bring them into scope and call the in my main function. I then import all the functions I need from all the modules, and also import the error trait to help with errors. Next, I create my main function that will put everything together and use all the functions I called in and perform the analysis and code. I first use the ‘load_attendance_data’ and ‘load_game_data’ to load in and read the CSV files with the attendance and game data, which I put in the SRC folder under the project, and then I define features to be the prepared features after running the ‘prepare_features’ function with the attendance and game data, and return an error message if the data is empty. Then, I define regression_result to be the result of the function ‘perform_regression’ with the features variable, and also set regression_result to be the intercept, slope, and r-sqaured, all with _ infront of them because I am not directly calling them and will not be directly using them. I then use a println! Statement to print out the regression coefficients with the intercept and slope, which will be in ‘regression_result’. I then create the decision tree and then train the tree and return the accuracy of the tree using the

‘train_decision_tree’ and ‘evaluate_decision_tree’ functions with the data, and then I use a print statement to print out the accuracy of the decision tree model and how well it was able to predict the outcome. Finally, I add one more print statement to print out the R-squared value, and then return everything. I then run cargo build, cargo run, cargo run –release, and cargo clean.

Results, implications, and future improvements:

```
Regression Coefficients: Intercept = 67455.88, Slope = 207.60
Decision Tree Accuracy (Predicting Attendance from Success): 0.50
Regression R-squared: 0.00
```

From these results, we found that the intercept to be 67,455.88. This means that when the team had 0 wins, their attendance was approximately 67,455.88. This is the average weekly attendance whether a team won the game or not. Meaning, that whether they won or not, a team on average had approximately 67,455.88 fans in attendance weekly. The slope was 207.60, meaning that with every win, approximately 207 more fans attended the games. The regression R-squared being 0 means that there is no correlation or a very weak correlation between the number of wins and weekly stadium attendance. The decision tree accuracy being 0.50 means that it is basically a 50-50 chance of correctly the attendance based on a team’s success. The accuracy being so low means that the model cannot properly predict attendance based on success.

The decision tree accuracy being 0.50 and the regression R-squared 0.00 means that there is no correlation or a very weak correlation between a team’s success and their weekly stadium attendance. This means that there are probably other factors that have a greater influence on weekly stadium attendance. This could be factors such as weather, day of the week, team popularity, stadium size, or even marketing. The slope being 207 means that even though there is a slight increase in attendance after each win, it did not raise attendance that much, and that wins did not really drive attendance rates. In the future, I would try to improve my model and try to improve accuracy by perhaps introducing other factors other than wins that can influence attendance. The models and accuracy are very limited by wins alone being the deciding factor to predict attendance.